Explorations into MapReduce based Parallel Reduct Computation

A thesis submitted during 2021 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in School of Computer and Information Sciences

by

PANDU SOWKUNTLA

Reg. No: 15MCPC20

School of Computer and Information Sciences
University of Hyderabad
(P.O.) Central University, Gachibowli,
Hyderabad - 500046
Telangana, India

2021

CERTIFICATE

This is to certify that the thesis entitled **"Explorations into MapReduce based Parallel Reduct Computation"** submitted by **Pandu Sowkuntla** bearing **Reg. No: 15MCPC20** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by him under my supervision and guidance.

The thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

The student has the following publications before submission of the thesis for adjudication and has produced evidence for the same in the form of acceptance letter or the reprint in the relevant area of his research:

1. **Pandu Sowkuntla** and P. S. V. S. Sai Prasad. *MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme.* **Knowledge -Based Systems**,189:105104, Feb 2020, Elsevier. `https://doi.org/10.1016/j.knosys.2019.105104` [**Indexed in SCI, SCOPUS**]. The work reported in this publication appears in Chapter 3.

2. **Pandu Sowkuntla**, Sravya Dunna, and P. S. V. S. Sai Prasad. *MapReduce based parallel attribute reduction in Incomplete Decision Systems.* **Knowledge -Based Systems**, 213:106677, Feb 2021, Elsevier. `https://doi.org/10.1016/j.knosys.2020.106677` [**Indexed in SCI, SCOPUS**]. The work reported in this publication appears in Chapter 4.

3. **Pandu Sowkuntla** and P. S. V. S. Sai Prasad. *MapReduce based parallel fuzzy-rough attribute reduction using discernibility matrix.* **Applied Intelligence**, pages 1–20, Apr 2021, Springer. `https://doi.org/10.1007/s10489-021-02253-1` [**Indexed in SCI, SCOPUS**]. The work reported in this publication appears in Chapter 5.

Further the student has passed the following courses towards fulfilment of course work requirement for Ph.D.

Course Code	**Name of the course**	**Credits**	**Pass/Fail**
CS801	Data Structures and Algorithms	4	Pass
CS802	Operating Systems and Programming	4	Pass
CS811	High Performance Computing	4	Pass
AI851	Trends in Soft Computing	4	Pass

(Dr. P. S. V. S. Sai Prasad)
Supervisor
School of Computer and Information Sciences
University of Hyderabad
Hyderabad – 500046, India

(Prof. Chakravarthy Bhagvati)
Dean
School of Computer and Information Sciences
University of Hyderabad
Hyderabad – 500046, India.

DECLARATION

I, **Pandu Sowkuntla**, hereby declare that this thesis entitled **"Explorations into MapReduce based Parallel Reduct Computation"** submitted by me under the guidance and supervision of **Dr. P. S. V. S. Sai Prasad** is a bonafide research work and is free from any plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be submitted in Shodganga/INFLIBNET.

A report on plagiarism statistics from the University Librarian is enclosed.

Date :

Signature of the Student

Name: Pandu Sowkuntla

Reg. No.: 15MCPC20

To my daughter, ***Shanmukha Shree Koumudi****, whose presence makes me feel a great serenity. I love you* ***Siri****.*

To my supervisor, ***Dr. P. S. V. S. Sai Prasad****, without whose support and encouragement, this would not have been possible.*

Acknowledgements

The journey to become a doctor is lengthy and difficult. It is my school of thought which inspired me to go on this adventure and to remain intellectually curious, and I hope it never leaves me. I am grateful to everyone who has assisted me in my quest to obtain a Ph. D.

First and foremost, I would like to express my sincere gratitude to my supervisor, **Dr. P. S. V. S. Sai Prasad**, for his much time, effort, and knowledge in assisting me in completion of this research work. I can not describe how much time he spent encouraging me, correcting me, and instilling hope in me. *He has fostered in his students a passion of guidance and assistance via his example.* Thank you for your patience, subject expertise, and being a *nice human being.*

I would like to convey my heartfelt thanks to RAC members **Prof. Rajeev Wankar** and **Dr. Y. V. Subba Rao**, whose insights and comments aided me in completing this doctoral research work. I would like to express my appreciation and thanks to **Prof. Chakravarthy Bhagvati**, Dean, School of Computer and Information Sciences (SCIS), for his cooperation, particularly during the COVID 19 pandemic.

I would like to express my gratitude to **Prof. C. Raghavendra Rao** for allowing me to attend the *Rough Computing* course, which sparked my interest in the field and substantially aided me in learning the fundamental principles of rough set based attribute reduction.

I would like to show my thankfulness to **Prof. Rajeev Wankar** for permitting me to attend the *Parallel Computing* course, which immensely aided me in grasping the fundamental concepts of parallel/distributed processing.

I had also encountered difficulties along this journey. A special thanks goes to **Dr. K. Swarupa Rani**, **Prof. Vineet C Padmanabhan Nair** and **Prof. Salman Abdul Moiz** for their valuable suggestions and help during these hard times. I would like to express my sincere thanks to the office and technical staff of the SCIS, as well as the administration staff of the University of Hyderabad, for their assistance.

I would like to extend my sincere thanks to **Dr. Ni Peng**, School of Information, Renmin University of China, for sharing the source code of the PARA algorithm, which is used in the experimental study of one of the proposed approaches.

I am grateful to the Ministry of Electronics and Information Technology, Government of India (Digital India Corporation), for funding my research under the **Visvesvaraya Ph. D. scheme** with the unique id: **MEITY-PHD-1039**. I owe a debt of gratitude to **Dr. Y. V. Subba Rao** (Coordinator for PhD scheme) who encouraged me to apply this fellowship and for taking required steps in getting fellowship on time.

My sincere thanks to all of my Research Scholars Lab friends, whose friendship helped me a lot on this journey. Special thanks to my co-scholars, **Abhimanyu Bar**, **Anil Kumar** and **D. Bheekya** for their friendship and support, as well as for sharing fun, frustration, and companionship. The tea break was the favorite moment of the day for us along with the supervisor. Not simply for the tea, or the break, but also for the pleasant company. Thank you very much.

I cannot end without expressing deepest gratitude to my parents (**Aagaiah** and **Dhanamma**) for all of the love, care, support, and prayers they have sent my way during this journey. I hope you have become proud of me. And to my daughter **Shanmukha Shree Koumudi**, thank you for being there for me at the end of the day. *Your presence gives me a sense of calm and refreshment. I love you* ***Siri****.*

Pandu Sowkuntla

Abstract

Feature selection is one of the data reduction techniques, and it is the process of selecting a minimal subset of features that provide the same classification ability as the given whole set of attributes. In 1982, Prof. Pawlak introduced Rough Set Theory (RST) as Soft Computing paradigm, which has emerged as a robust mathematical framework for feature selection. Rough sets based feature selection is known as *attribute reduction* or *reduct computation*.

Nowadays, the volume of data is increasing at an unprecedented rate. As the data grows in object space and/or attribute space in the data set, the attribute reduction has become an expensive preprocessing step. In the last decade, many rough set-based parallel/distributed approaches have been proposed using the MapReduce model for scalable attribute reduction. The majority of these approaches are hindered by the challenges presented by today's massive data sets. The current research is motivated by these challenges, which include *big dimensionality* (huge attribute space in the data set), *variety of the data* (different data in decision systems), and *data partitioning* strategy used in dividing the input data set.

All the existing MapReduce based reduct computation approaches in categorical data sets (decision systems with categorical attributes) adopted horizontal partitioning strategy for partitioning the data to the cluster of computers, where the data set is partitioned in object space. This strategy results in computational overheads for big dimensional data sets. Because, with this strategy, considerable amount of data to be communicated across shuffle and sort phase and a complex reduce phase is involved in any MapReduce framework. Furthermore it presents an immense problem if the large-scale data set contains missing (incomplete) values (in incomplete decision systems), or if the data set contains different types of attributes (in hybrid decision systems). Since extensions to classical rough sets are used to deal with incomplete and hybrid decision systems, attribute reduction in these decision systems pose much severe computational challenges and involve higher space and time complexities in building MapReduce based approaches.

This research focuses on scalable attribute reduction in large-scale data sets using MapReduce model, with an emphasis on the data set's big dimensionality. Along with horizontal partitioning strategy, an alternative strategy known as "*vertical partitioning*" is being explored in dealing with big dimensional data

sets. Therefore, this *thesis aim is to explore MapReduce based parallel/distributed reduct computation in categorical, incomplete and hybrid decision systems, where the relevance of horizontal and vertical partitioning strategies are investigated in partitioning the input data set to the nodes of the cluster.*

This thesis proposes a classical rough sets based approach using MapReduce for attribute reduction. This approach investigates vertical partitioning strategy, that is used to partition the input data set in the attribute space to the nodes of the cluster. The applicability of this strategy is explored for attribute reduction in large-scale categorical data sets with big dimensionality.

Different strategies in MapReduce framework are needed for parallelizing the existing extensions to classical rough sets for attribute reduction in large-scale incomplete and hybrid decision systems. Thus, in this thesis, MapReduce based attribute reduction approaches for incomplete decision systems are proposed using Novel Granular Framework (NGF) (an extension to classical rough sets) and adopt horizontal and vertical partitioning strategies for data partitioning. Fuzzy-rough set model (an extension to classical rough sets) is used to deal with hybrid decision systems. A fuzzy discernibility matrix based accelerator is introduced, and based on this accelerator, MapReduce based reduct computation approaches are proposed using horizontal and vertical partitioning strategies.

The proposed approaches are implemented using Apache Spark. Extensive experimental analysis carried out on different benchmark large-scale data sets with the variance in object and attribute space. The efficiency of the proposed approaches are evaluated based on the *computational evaluation* (Running time, reduct, and reduct size metrics are used), *performance evaluation* (Speedup, scaleup and sizeup metrics are used) and *impact of the data partitioning strategy.*

It is empirically proved that, the proposed approaches perform better than the existing state-of-the-art approaches. The experimental results along with theoretical validation show that the horizontal partitioning based approaches perform well for the larger object space data sets with moderate attribute space. And the vertical partitioning based approaches are relevant and scale well for moderate object space data sets with big dimensionality. In future, this research has a scope to explore viable MapReduce based reduct computation approaches that can simultaneously scale in both huge object space and huge attribute space in the data sets. Since, the proposed approaches deal with *volume* and *variety* characteristics of the big data, this research also has the scope to deal with *velocity* property by proposing MapReduce based incremental attribute reduction approaches for streaming data.

Contents

CONTENTS

CONTENTS

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Notations and Abbreviations

$[x]_P$: Equivalence class of an object x using indiscernibility relation $IND(P)$

$\perp$: T-conorm or S-norm

$\mu_{DIS_a}(x, x')$: The measure of degree to which the objects x and x' are dissimilar for numerical attribute a

$\mu_{SIM_a}(x, x')$: The measure of degree to which the objects x and x' are similar for numerical attribute a

$\overline{P}(X)$: Upper approximation of X defined using $IND(P)$

$\underline{P}(X)$: Lower approximation of X defined using $IND(P)$

$\{d\}$: Decision attribute

A: Set of conditional attributes

$C_{xx'}$: An entry in the discernibility matrix

$DIS(P)$: Discernibility relation of subset of attributes P

f_a: Information mapping from set of objects U to set of dommain values V_a

$FDM\_F(R)$: The subset of entries of FDM, which have reached the maximum SAT with subset of attributes R

$FDM\_UF(R)$: The subset of entries of FDM, which have not yet reached the maximum SAT with subset of attributes R

gr: A granule (a set of objects) belonging to granular space

$NP\_GR(U/R)$: Non positive region granules formed from U/R

$P\_GR(U/R)$: Positive region granules formed from U/R

NOTATIONS AND ABBREVIATIONS

POS**:** Positive region

R**:** Reduct set

$S_B(x)$**:** Similarity class of an object x using similarity relation $SIM(B)$

$SAT(P)$**:** Satisfiability value of subset P for all the entries in the matrix

$SAT_P(C_{xx'})$**:** Satisfiability value of subset P for an entry in the matrix

$SATUF(P)$**:** Satisfiablity of subset P for all the entries in the matrix after SAT-region removal

$SIM(B)$**:** Similarity relation of subset of attributes B

T**:** T-norm

U**:** Set of objects

$U/IND(P)$ **or** U/P**:** Granular space formed using $IND(P)$ on objects set U of CDS

$U/SIM(B)$**:** Granular space formed using $SIM(B)$ on objects set U of IDS

V_a**:** Set of domain values of attribute a in a decision system

$IND(P)$**:** Indiscernibility relation of subset of attributes P

CDS: Categorical Decision Systems

DAG: Directed Acyclic Graph

DARA: Discernibility matrix based Attribute Reduction Accelerator

DFDM: Distributed Fuzzy Discernibility Matrix

DFRS: Distributed Feature Selection for Big Data Using Fuzzy Rough Sets

FDM: Fuzzy Discernibility Matrix

FDMFS: Fuzzy Discernibility Matrix based Feature Selection

HDS: Hybrid Decision Systems

IDS: Incomplete Decision Systems

IFDMFS: Improved Fuzzy Discernibility Matrix based Feature Selection

IN_MRIQRA_IG: MapReduce based IQRA_IG

NOTATIONS AND ABBREVIATIONS

IND: Indiscernibility relation

IQRA_IG: Improved Quick Reduct Algorithm using Information Gain

KDD: Knowledge Discovery in Databases

MR_FRDM_SBE: Fuzzy Rough Discernibility Matrix Based Feature Subset Selection With MapReduce

MR_IFDMFS: MapReduce based Improved Fuzzy Discernibility Matrix Feature Selection

MR_IMQRA: An Efficient MapReduce Based Approach for Fuzzy Decision Reduct Computation

MR_IQRA_VP: MapReduce based Improved Quick Reduct Algorithm with Vertical Partitioning technique

MR_VFDMFS: Vertically partitioned Fuzzy Discernibility Matrix based Feature Selection using MapReduce

MRIDS_HP: MapReduce based parallel attribute reduction in IDS using Horizontal Partitioning technique

MRIDS_VP: MapReduce based parallel attribute reduction in IDS using Vertical Partitioning technique

NGF: Novel Granular Framework

PFSPA: Parallel Feature Selection using Positive Approximation

PLAR: Parallel Large-Scale Attribute Reduction

QRA: Quick Reduct Algorithm

RDD: Resilient Distribute Data set

RST: Rough Set Theory

SBE: Sequential Backward Elimination strategy

SFS: Sequential Forward Selection strategy

SIM: Similarity relation

vFDM: Vertically partitioned Fuzzy Discernibility Matrix

Chapter 1

Introduction

Recent advances in computing technologies such as Internet, Internet of Things, social networks, mobile communication systems, transportation systems have contributed to the increasing amounts of data in size. The *data set* is a structured collection of the data in the form of objects (samples) and features (attributes or dimension). The growing data volumes produce the large-scale data sets with huge object space or/and feature space. Large-scale data sets are typically affected by a significant amount of redundancy which can hinder knowledge discovery, and is, in fact, misleading.

Knowledge management plays a key role in generating a value from the data. It is necessary to go through a process to induce value from the data. The Knowledge Discovery in Databases (KDD) process [34, 36] is a general framework that describes the various steps needed to obtain useful knowledge from a collection of data. The primary goal of the KDD process is to find relevant knowledge from the data in huge databases. The KDD process has the steps: (i) Data selection (ii) Data cleaning/preprocessing (iii) Data reduction (iv) Data mining and (v) Interpretation/Evaluation. The third step, data reduction is the crucial step in the KDD process. This step deals with dimensionality reduction and *feature selection* is one of the prominent ways of doing the same. Feature selection helps in reducing the feature space and improves the performance of the later steps of the KDD process.

1.1 Feature selection

As given by Guyon et al. [39, 40], *feature selection is the process of finding relevant features and discarding those that are irrelevant and redundant, so as to obtain a subset of features that accurately describe a given problem with minimum degradation of performance.* It is a widely used preprocessing step for machine learning, data mining, and pattern recognition [39, 115]. Feature selection exploits the data redundancy to reduce the uncertainty from

large-scale data sets. It also acts as a solution that helps in mining knowledge from multi-dimensional large-scale data sets [10, 27]. The aim of the feature selection is: (i) to enhance the performance of the predictive models; (ii) to build them efficient in terms of their resource costs and (iii) to provide an insight into the underlying procedure that generated the data. Furthermore, the feature selection helps to reduce the adverse impact induced by the *curse of dimensionality*[2] [9].

Feature selection methods [40, 63] are divided into three categories according to their relationship with the learning (classification) algorithms, as filter, wrapper, and embedded.

- *Filter*: In filter methods, the search in feature space is done prior to the classification process. That is, filter method performs feature selection independently of any learning algorithm. As a result, these are the methods take less computing and memory resources. Additionally, filters may be uni-variate or multivariate, based on whether the feature assessment is performed individually or collectively. Filter methods use different metrics for evaluating features, and these metrics are categorised into: information based, distance based, correlation based and consistency based. Filter based feature selection methods are fast and scalable, and they are independent of the classifier.

- *Wrapper*: This method search the feature space depending on the classification accuracy assessment of the learning algorithm. That is, different subset of features are identified and evaluated them by using the classifier. Generally, the wrapper method is better in terms of classification accuracy, but computationally costly since it trains the classifier multiple times in each step of the feature space search, and subset selected is biased towards the classifier.

- *Embedded*: This method searches for an optimal subset of features that is built into the classifier construction. The advantage of this method is that it is less computationally intensive than a wrapper method. And, embedded method is computationally slower than filter method, but some times filter method may fail to select best features. Thus, embedded method lies in between the filter and wrapper methods.

Each of the above feature selection methods further categorised into the following two sub-categories depending on the output they produce.

- *Feature ranking*: Depending on the evaluation metric used, methods in this category generate an output that consists of an ordered list of features graded by their importance.

[2]The curse of dimensionality refers to a set of problems that emerge while working with high-dimensional data.

- *Feature subset selection*: The method in this category produces the output which consists of the subset of features that are considered most important and the remaining features are removed.

The feature subset selection is performed in four basic steps [53], (i) Feature subset generation (ii) Subset evaluation (iii) Stopping criterion, and (iv) Validation of results. The feature subset is generated based on a *search strategy* [63]. Four search strategies are existed, namely, forward search, backward search, bidirectional search and random search. The forward and backward search strategies are prominent and widely used search strategies.

- *Forward search strategy*: In this strategy, the feature subset generation starts by initializing the subset as an empty set, and features are added incrementally until a specified stopping criterion is satisfied or number of features in the subset is reached to a threshold value. Since the features are selected sequentially, it is named as *Sequential Forward Selection (SFS)* strategy.

- *Backward search strategy*: In this strategy, the feature subset generation starts from complete set of features, and the redundant features are removed one by one from the complete set based on a criterion that checks whether a feature is redundant or not. This strategy is also called as *Sequential Backward Elimination (SBE)*.

The SFS strategy may results in a subset of features that contain redundant features. But it is observed that the SBE strategy guarantees minimal subset features without any redundant features. However the computational efficiency of SFS strategy over SBE makes it suitable for building scalable feature subset selection approaches for large-scale data sets.

1.2 Rough Set Theory for feature subset selection

In 1982, Prof. Pawlak [77] introduced Rough Set Theory (RST) as Soft Computing paradigm. In recent years, RST has emerged as a robust mathematical framework for attribute reduction [78, 79, 80]. It is effective in dealing with uncertainty and vagueness in the data. RST has become an area of great interest to the researchers for the following reasons.

- Requiring no additional information, by just using the data alone, RST enables the reduction of the attributes and the discovery of data dependencies in a data set [78, 80].

- Rough sets have been complemented by other soft computing technologies such as neural networks, fuzzy sets and many successful hybrid models have been generated. Some of the popular hybrid models include: fuzzy-rough sets and rough-fuzzy sets [32, 33], rough-neural networks [61], rough-genetic algorithms [43, 62].

- Several extensions [66, 94, 123] to rough sets[1] are available to deal with many real-world problems such as decision analysis, data mining, intelligent control and pattern recognition. The prominent extensions include: variable precision rough sets, tolerance rough sets, probabilistic rough sets, fuzzy-rough sets and dominance-based rough sets.

Generally the data in a data set is stored in the form of a table, where the rows represent objects and columns represent features. In rough sets terminology, this data table is known as *information system*. An information system can be extended to *decision system* by the inclusion of decision attributes. Feature subset selection using rough sets principles is known as *attribute reduction*[2]. The selected feature subset in the case of information systems is termed as *reduct* and *relative reduct* in the case of decision systems. In the remainder of the thesis, as the thesis is restricted to decision systems, the term "reduct" refers to "relative reduct", and the *reduct computation* is considered synonymous to attribute reduction. Thus the *reduct is a minimal subset of conditional attributes that provide the same classification ability as the set of conditional attributes in the decision system.* In RST, the reduct computation methods are classified into many categories. However the primary categories include: (i) *Dependency measure*, and (ii) *Discernibility matrix* [44, 53]. In this thesis, the proposed methods are developed based on both the dependency measure and discernibility matrix approaches. Theoretical background of these two approaches is provided in the second chapter.

Due to the exponential growth of data, if the data set is large-volume or/and high dimensional, the traditional (sequential) attribute reduction algorithms can not perform well. These algorithms face problems from both data storage and computational complexity viewpoints. Scalability of attribute reduction suffers with the large-scale data sets due to insufficient memory space available in a single node [42, 103]. Most of the researchers found parallel/distributed computation as the good solution for scalable attribute reduction. Therefore, researchers try to parallelize the traditional attribute reduction algorithms to improve their efficiency on large data sets.

1.3 Parallel/distributed computation

Nowadays, the volume of the data is growing at an unprecedented rate [10, 38, 59]. Prior to the year 2003, mankind generated only 5 exabytes of data, but currently 5 exabytes of data is produced in just two days, and the rate of the growth continues to rise [10]. As the

[1] "Rough sets" introduced by Prof. Pawlak termed as "*classical rough sets*" in the rest of the thesis in order to distinguish from its extensions and hybrid models.

[2] The term "feature subset selection" is replaced with "attribute reduction" in the rest of the thesis.

data volume escalates, knowledge discovery has become a challenging task because of the uncertainty and inconsistency in the data. Due to the continuous increase in the volume of the data sets, attribute reduction techniques have become essential in extracting relevant information from vast amounts of data. Thus, we require parallel/distributed solutions for attribute reduction in large-scale data sets.

In parallel/distributed computation, several calculations are carried out concurrently in task and/or data parallel [6]. Task parallelism attempts to run many tasks concurrently, while data parallelism targets to perform the same task on several data sets. Some of the parallel attribute reduction methods [87] work on decomposing the entire computation into smaller sub-tasks, which are processed on separate nodes. Some other parallel attribute reduction approaches [26] employ data parallelism. Both task and data parallelism are used to perform parallel attribute reduction [7, 60, 104, 106]. Traditional parallel/distributed computation models such as MPI, OpenMP, BSP, etc., are used for processing large-scale data sets.

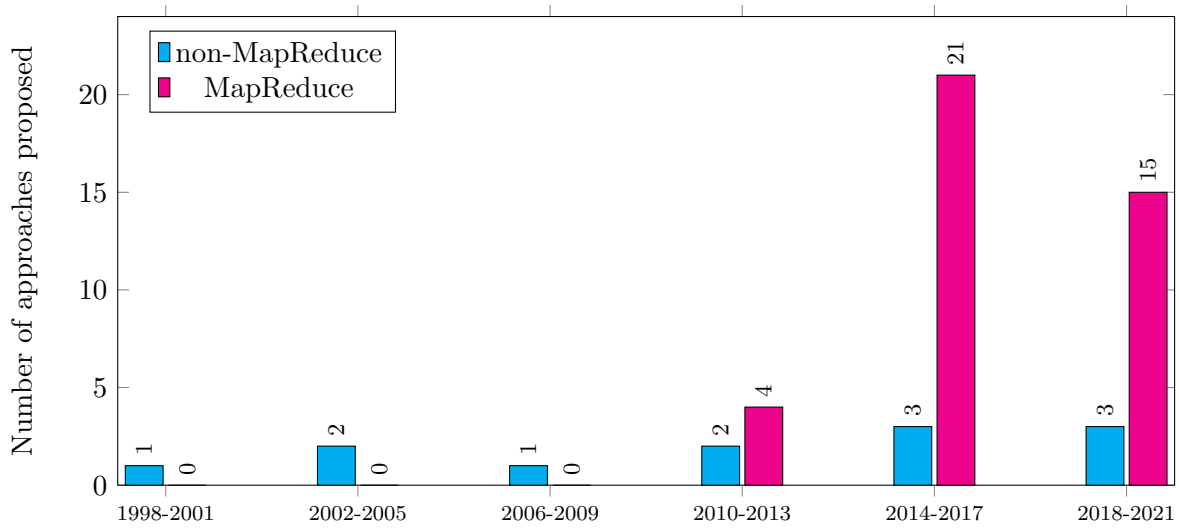


Figure 1.1: Number of parallel/distributed reduct computation approaches proposed based on non-MapReduce and MapReduce model

Based on a comprehensive review of the literature of reduct computation over the last two decades, it is observed that, researchers are more likely to adopt the MapReduce paradigm for creating parallel/distributed approaches than non-MapReduce (traditional parallel/distributed computation) models. Figure 1.1 shows the number of parallel/distributed approaches proposed during the last two decades based on non-MapReduce and MapReduce models. From this figure, it can be noticed that, with the advancement of the MapReduce model [25], most of the researchers switched their attention from traditional parallel/distributed computation to MapReduce based parallel/distributed computation. Because the MapReduce framework

provides a consistent structure for deriving granular aggregated information in the *reduce* phase using constructed partial granules information in the *map* phase, which is crucial for achieving rough set based attribute reduction. Thus, in recent years (especially last decade) several MapReduce based parallel/distributed approaches were proposed for attribute reduction in large-scale data sets [8, 81, 96, 100, 101, 102, 114, 124, 125].

MapReduce can be referred as an execution framework for robust and scalable implementation of parallel/distributed algorithms. The framework has mainly three phases or steps: *Map*, *Shuffle and Sort*, and *Reduce*. The framework coordinates these phases of processing over massive amounts of data on large cluster of computing nodes. Hadoop [1], Apache Spark [121], Twister [35], etc., are some of the existing MapReduce frameworks. MapReduce model overcomes the drawbacks in traditional parallel/distributed computation, and has the following significant benefits.

- MapReduce model conceals a number of system-level details from the user and facilitates parallelization of the computation on large-scale data, across cluster of computers.
- This model is capable of detecting machine failures and coordinating between machines for optimal use of networks and storage devices (i.e., fault tolerance).

MapReduce has been proven to be helpful to design an effective solution to a complex task on massive data, and, it is currently being applied in many areas such as data mining [42], machine learning [17, 103, 131]. Thus, in this thesis, the approaches are proposed based on MapReduce programming model.

1.4 Research motivation

Several researchers have been interested in attribute reduction in large-scale data sets, and many approaches have been proposed. The majority of these approaches are hindered by the challenges presented by today's massive data sets. In order to deal with these challenges, the existing attribute reduction methods must be improved or new ones must be proposed. The current research is motivated by these challenges, which include, *big dimensionality*, *variety of the data* in the large-scale data sets, and *data partitioning* strategy used in splitting the data set.

1.4.1 Big dimensionality

Attribute reduction in large-scale data sets is not only impacted by the number of objects a data set has but also by the number of attributes. The size of a data set in object space or/and

in attribute space grows more and more as data increases rapidly. Similarly to big data, the term "*big dimensionality*" [10, 11] has been invented to describe the enormous amount of attributes reaching to levels that render existing attribute reduction approaches ineffective. According to extensively used UCI Machine Learning repository [31], the maximum dimensionality of data set in the years 1980s was only approximately 100. By the 1990s, this number increased to more than 1500, and by the year 2009, it had risen to more than 3 million. Figure 1.2 depicts the number of attributes in the highest dimensionality data sets that have been added to UCI Machine Learning repository during the previous 12 years. Some of the most prominent areas that deal with big dimensionality challenges are, microarray analysis [4, 11], text and image classification [3, 10, 65, 93].

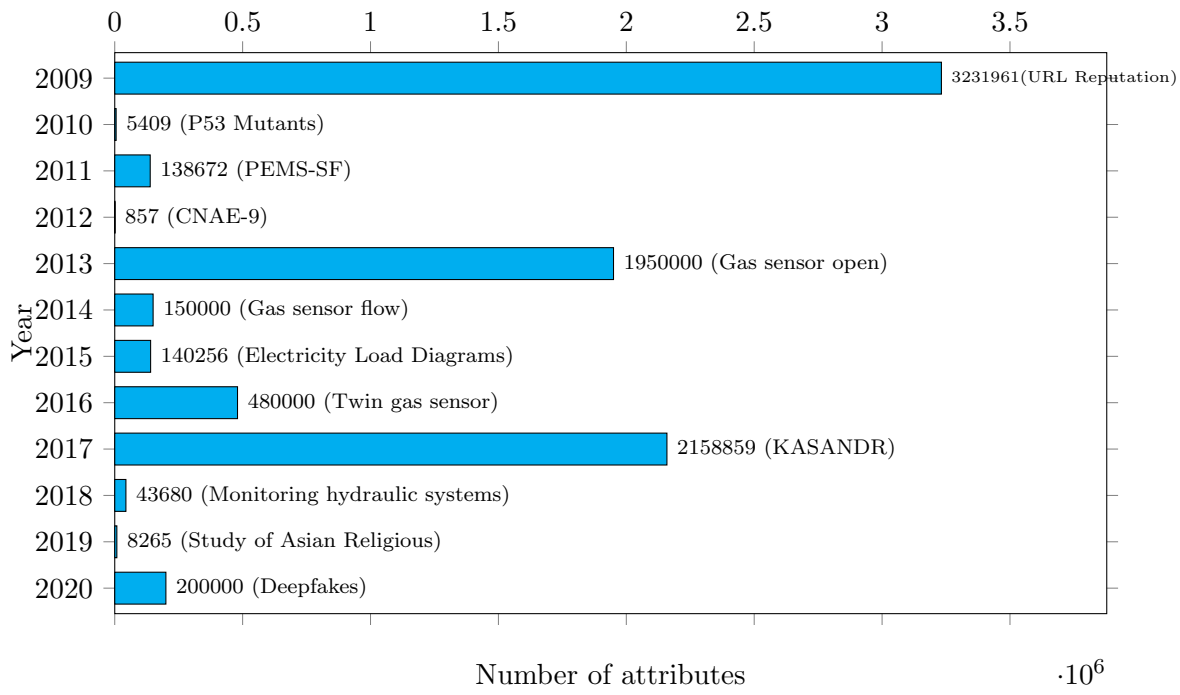


Figure 1.2: Big dimensionality of the data sets added to UCI repository since the year 2009

To accelerate the attribute reduction in large-scale data sets, in the last decade, many classical rough set-based methods have been developed using the MapReduce programming model, and some of them have shown their strengths in comparison to the rest. Despite the effectiveness of the existing MapReduce based rough set attribute reduction methods, they are confronted with various issues and are unable to effectively and efficiently handle the large data sets, especially data sets with big dimensionality. The big dimensionality in the data set presents a tremendous challenge to the researchers.

1.4.2 Impact of the data partitioning

All the existing rough set theory based attribute reduction approaches [15, 45, 81, 92, 114, 124] using the MapReduce programming model adopted horizontal partitioning strategy for partitioning the input data to the cluster. With this strategy, the object space of the input data set is partitioned, and the data partitions are distributed to the nodes of the cluster. Hence an attribute values of different objects are scattered throughout all the partitions of the data set, which are located in different nodes in the cluster. Horizontal partitioning based approaches are proved to be efficient for the data sets having huge object space with moderate attribute apace. In data sets involving smaller object space and more attribute space such as many bioinformatical microarray and document classification data sets (i.e., big dimensionality data sets), the horizontal partitioning strategy results in computational overheads. Because with this strategy, considerable amount of data to be communicated across shuffle and sort phase and a complex reduce phase is involved in any MapReduce framework.

Because of the horizontal partitioning strategy, researchers concentrated only one aspect of large-scale data sets, namely large number of objects aspect in the data set in designing attribute reduction algorithms, while paying little attention to the attributes aspect. This has inspired us to look into alternative data partitioning strategy that avoid the problems of horizontal partitioning and efficient in handling the data sets with big dimensionality.

1.4.3 Variety of the data

In addition to the size of the data, the *variety of the data* has a significant influence on the reduct computation process. The variety of the data consist of structured, unstructured, and semi-structured data from various sources. Just as data previously had to be obtained from spreadsheets and databases, today data is found in various formats, such as categorical, real, boolean, images, audio, video, and other social media platforms. Attribute reduction in large-scale data sets poses an immense problem if the data contains missing (incomplete) values, or if the data contains different types of attributes.

The decision systems with categorical (or discrete) attributes is known as complete symbolic decision systems (also known Categorical Decision Systems (CDS)). The decision systems that include objects with missing attribute values are referred to as incomplete symbolic decision systems (also known Incomplete Decision Systems (IDS)). The decision systems with different types of attributes (e.g., categorical, numerical,...etc.) is known as hybrid decision systems (HDS). Classical rough set model uses crisp equivalence classes in attribute reduction. As a consequence, it is suitable to perform the attribute reduction in symbolic (categorical)

data sets (i.e., CDS). The decision systems, IDS and HDS are frequently occurring data sets in decision-making problems. Thus, extensions to classical rough sets [57, 58, 99, 112] are available to deal with IDS for attribute reduction. Various fuzzy-rough set models [19, 32, 86, 117] are available to handle different types of attributes in data set for attribute reduction.

From the literature, it is observed that, a lot of research works have been done on attribute reduction in IDS [23, 30, 64, 73, 84, 108, 129, 134]. But all the existing approaches are sequential methods and they can not handle the large-scale incomplete data sets. Parallel/distributed approaches are not proposed for attribute reduction in large-scale IDS. Because, the processing of large-scale IDS is difficult due to two challenges, incompleteness involved in the data, and the large size of the data set. Thus, different strategies in MapReduce framework are needed for parallelizing the existing extensions to classical rough sets for attribute reduction in large-scale IDS.

From extensive review of current literature [28, 46, 54, 56, 72, 85, 91], it is observed that, the approaches for attribute reduction in HDS involve higher space and time complexities compared to classical rough sets. It is also observed that a substantial decrease in memory usage is achieved in the discernibility matrix based approach relative to the dependency measure based approach. Further discernibility matrices are more suitable for performing parallel/distributed computation. It is noticed that, the discernibility matrix based accelerators and the corresponding parallel/distributed approaches do not exist for attribute reduction in HDS. Thus, MapReduce based parallel/distributed methods using discernibility matrix are needed to overcome the higher space complexities of dependency measure based approaches.

1.5 Research objectives

This research focuses on scalable attribute reduction in large-scale data sets using MapReduce, with an emphasis on the data set's big dimensionality. Each of the research problems mentioned in Section 1.4 form the objectives of this research work. And each of these concerns are addressed one by one as follows.

- The first objective of this thesis is to investigate an alternative strategy known as "*vertical partitioning*", which is used to partition the input data set in the attribute space and distribute the data partitions to the nodes of the cluster. The applicability of this strategy is explored for rough set based attribute reduction in Categorical Decision Systems (CDS) with big dimensionality.

- The second objective of this thesis is to investigate MapReduce based attribute reduction approaches for large-scale IDS that use existing Novel Granular Framework (NGF) to

handle the incompleteness in the data and adopt horizontal and vertical partitioning strategies.

- The third and fourth objectives of thesis are to explore discernibility matrix based attribute reduction in large-scale HDS using MapReduce with the strategies of horizontal and vertical partitioning.

The summary of aforementioned objectives of this thesis can be enunciated as follows:

"***This thesis objective is to explore MapReduce based parallel/distributed reduct computation in categorical, incomplete and hybrid decision systems, where the relevance of horizontal and vertical partitioning strategies are investigated in partitioning the input data to the nodes of the cluster.***"

1.6 Contributions and publications

Contributions to this thesis are made in relation to the research objectives outlined in the preceding section. Therefore, contributions are categorized according to the type of the decision system, i.e., categorical (CDS), incomplete (IDS), or hybrid (HDS). Each contribution and its corresponding publication are enumerated below.

1. **Contribution 1**: A MapReduce based algorithm MR_IQRA_VP is proposed using vertical partitioning strategy for attribute reduction in CDS. Here, the vertical partitioning strategy partitions the input data set in attribute space to the nodes of the cluster. This strategy is used alternative to horizontal partitioning strategy.

   The work in this contribution is resulted in the following publication.

   - Pandu Sowkuntla* and P. S. V. S. Sai Prasad. **MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme**. *Knowledge-Based Systems*, Elsevier, 189:105104, Feb 2020. `https://doi.org/10.1016/j.knosys.2019.105104` (Indexed in SCI, SCOPUS).

2. **Contribution 2**: MapReduce based parallel/distributed approaches are proposed based on the Novel Granular Framework (NGF) [73] for attribute reduction in large-scale IDS using horizontal and vertical partitioning strategies. Briefly, this contribution includes the following:

   - An alternative representation of the NGF is proposed and adopted to develop the MRIDS_HP algorithm. This algorithm uses the strategy of horizontal partitioning.

- Algorithm MRIDS_VP is developed by parallelizing the existing NGF based on the strategy of the vertical partitioning.

The work in this contribution is published as given below.

- Pandu Sowkuntla*, Sravya Dunna and P. S. V. S. Sai Prasad. **MapReduce based parallel attribute reduction in Incomplete Decision Systems**. *Knowledge-Based Systems*, Elsevier, 213:106677, Feb 2021. `https://doi.org/10.1016/j.knosys.2020.106677` (Indexed in SCI, SCOPUS)

3. **Contribution 3**: A fuzzy discernibility matrix based attribute reduction accelerator (DARA) is introduced for scalable attribute reduction in HDS. Based on this accelerator, a sequential algorithm IFDMFS (Improved Fuzzy Discernibility Matrix based Feature Selection) is developed. In order to enhance scalability even further, an algorithm MR_IFDMFS is proposed using horizontal partitioning strategy. This algorithm is a MapReduce based parallel/distributed version of IFDMFS.

   The work in this contribution resulted in the following publication.

   - Pandu Sowkuntla* and P. S. V. S. Sai Prasad. **MapReduce based parallel fuzzy-rough attribute reduction using discernibility matrix**. *Applied Intelligence*, Springer, pages 1–20, April 2021. `https://doi.org/10.1007/s10489-021-02253-1` (Indexed in SCI, SCOPUS).

4. **Contribution 4**: Based on DARA accelerator (see contribution 3), an algorithm MR_VFDMFS is proposed using vertical partitioning strategy. It is also a MapReduce based parallel/distributed version of IFDMFS. This algorithm is proposed for achieving scalability in big dimensional HDS.

   The work in this contribution will be communicated soon to the following journal.

   - Pandu Sowkuntla and P. S. V. S. Sai Prasad. **MapReduce based parallel attribute reduction in high dimensional hybrid decision systems**. *International Journal of Machine Learning and Cybernetics* (to be communicated).

1.6.1 Supplementary Contributions

Throughout my Doctoral research, I also contributed to the following collaborative publications. They are not acknowledged as contributions in this thesis.

- Kiran Bandagar, **Pandu Sowkuntla***, Salman Abdul Moiz, and P. S. V. S. Sai Prasad. **MR_IMQRA: An Efficient MapReduce Based Approach for Fuzzy Decision Reduct Computation**. In *International Conference on Pattern Recognition and Machine Intelligence (PReMI)*, pages 306–316. Springer International Publishing, 2019. `https://doi.org/10.1007/978-3-030-34869-4_34` (Indexed in SCOPUS, DBLP).

- Neeli Lakshmi Pavani, **Pandu Sowkuntla***, K. Swarupa Rani, and P. S. V. S. Sai Prasad. **Fuzzy Rough Discernibility Matrix Based Feature Subset Selection With MapReduce**. In *IEEE Region10 Conference (TENCON)*, pages 389–394. IEEE, OCT 2019. `DOI:10.1109/TENCON.2019.8929668` (Indexed in SCOPUS, DBLP).

1.7 Organization of Thesis

The thesis is divided into chapters based on the approaches proposed for the decision systems: CDS, IDS and HDS. Figure 1.3 depicts the structure of the thesis. The present chapter (Chapter 1) provides the introduction to this thesis, where it reviews the feature selection, rough set theory and parallel/distributed computation. It also presents the research motivation, objectives and the contributions made to this thesis.

Chapter 2 introduces the fundamental principles of classical rough sets and rough set-based attribute reduction (or reduct computation). A brief overview of the MapReduce programming model is given. This chapter also includes a detailed discussion of the Apache Spark MapReduce framework, which is used to implement the proposed approaches of this research work.

The contributions made to this research work are discussed in Chapters 3, 4 and 5. In Chapter 3, we explore into parallel attribute reduction in CDS based on classical rough sets. This chapter provides a MapReduce-based approach for big dimensional data sets that uses a vertical partitioning strategy for partitioning the input data set.

In Chapter 4, we investigate at parallel attribute reduction in IDS. Initially, this chapter discusses the extension of rough sets for IDS. And, the proposed MapReduce-based parallel/distributed approaches employing horizontal and vertical partitioning strategies are discussed. Both proposed approaches utilize the existing NGF to deal with incompleteness in the data.

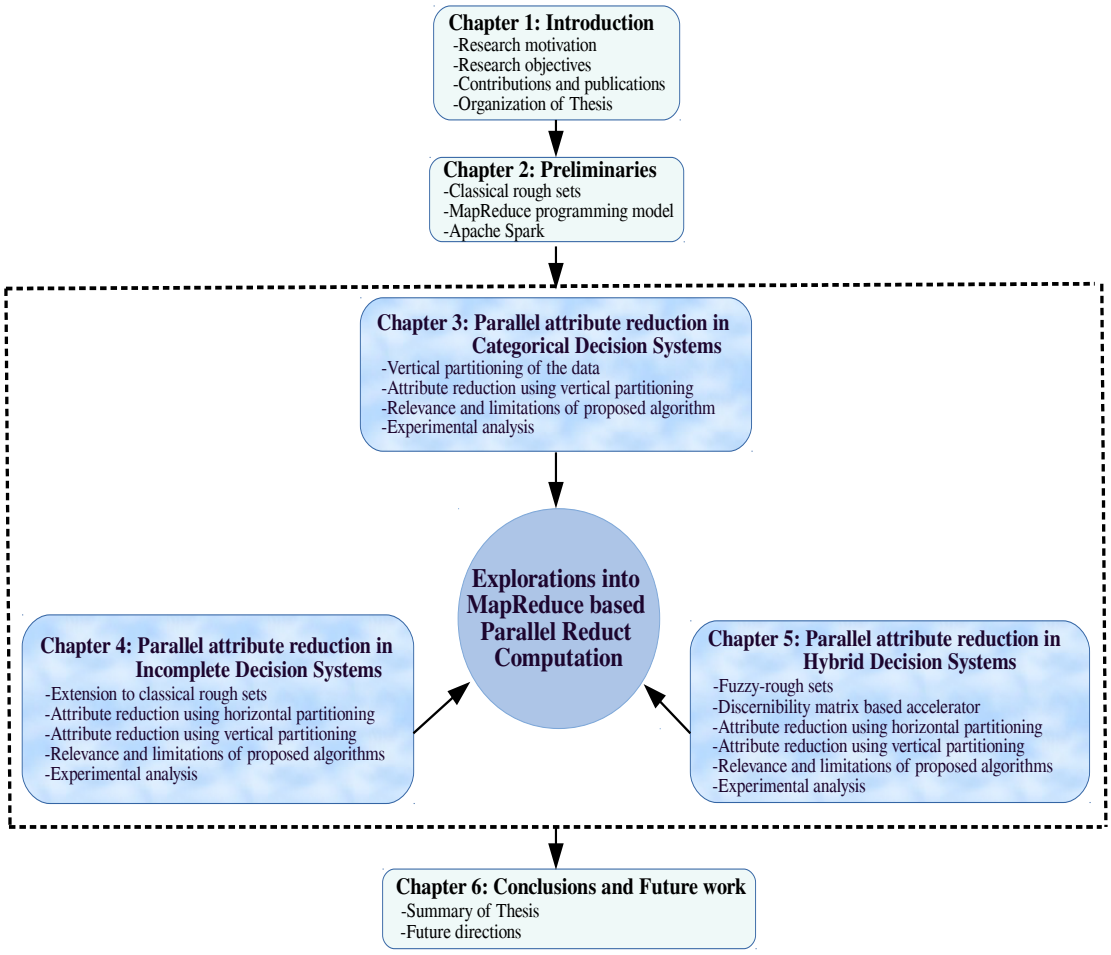


Figure 1.3: Structure of Thesis

Chapter 5 presents parallel attribute reduction in HDS. Fuzzy-rough set theory, which is an extension to classical rough sets is discussed in this chapter. This chapter introduces a fuzzy discernibility matrix-based attribute reduction accelerator (DARA) to accelerate the attribute reduction. And, proposes a sequential approach based on DARA, and corresponding MapReduce based approaches using horizontal and vertical partitioning strategies.

Chapter 6 concludes the thesis, summarizes the research work's key contributions and potential future scope.

Chapter 2

Preliminaries

This chapter provides the theoretical background for the research work provided in this thesis. A brief overview of classical rough set theory is provided for understanding the approaches proposed in this thesis. A foundation basis for attribute reduction using classical rough sets is provided. And the discussion moves on to the core concepts of MapReduce programming model and one of its framework Apache Spark which is used to implement the approaches proposed as part of this thesis. It is to be noted that all the preliminary notions relating to the rough sets are limited to the scope of this thesis.

2.1 Classical rough sets

A set in *classical set theory* is uniquely determined by its elements. In other words, a set is well-defined because it does not allow for any ambiguities when it comes to determining whether or not an element belongs to the set. Thus, a set adheres to a *crisp* (precise) notion. That is, the membership value is crisp either 1 (if element belongs to set) or 0 (if element does not belong to the set). For example, "the set of even numbers" is crisp because every number is even or not even (odd). But, ambiguity occurs when referring to concept like "attractive picture", where clear belongingness is difficult to define for the word "attractive". We can not classify all the given set of pictures into "attractive" or "not attractive". Thus the concept of "attractive picture" is not crisp (imprecise) but it is *vague*. Therefore, classical set theory does not support vague concepts.

Fuzzy sets proposed by Lotfi Zadeh [118] successfully deals with vagueness. In this approach, a set is defined by partial memberships (that lies between the values 0 and 1), as opposed to crisp membership used in classical set theory. For example, for the concept of "attractive picture", we can give a membership degree 0.7 to a picture of given set of pictures, in other words we can say that the picture is 70% attractive. Fuzzy set theory and its

applications have grown in popularity over the years, drawing the attention of researchers, logicians, and philosophers all over the world.

Rough set theory (RST) [34, 77, 78, 97] is another approach to deal with vagueness. It is an evolution of classical set theory that facilitates approximations in decision-making instead of using partial memberships. In RST, a vague concept is represented with a pair of crisp concepts called *lower approximation* and *upper approximation*. The lower approximation defines the domain objects that are certainly to be part of the subset of interest while a description of the objects that possibly belong to the subset is the upper approximation. A vague concept is said to be crisp set (definable set) if its lower approximation is equal to upper approximation, otherwise it is said to be *rough set*. For the same example of the concept "attractive picture", among the given set of pictures, those that are considered to be certainly attractive fall into lower approximation, while those that are considered to be possibly attractive fall into upper approximation.

Prof. Z. Pawlak introduced the classical rough sets in [77] and basics related to reduct computation are given in [78, 79, 115]. Classical rough sets are defined for categorical decision systems, where a categorical decision system is defined as,

$$CDS = (U, A \cup D, \{V_a, f_a\}_{a \in A \cup D})$$

Here, $U = \{x_1, x_2, ...x_m\}$ is a finite nonempty set of objects, $A = \{a_1, a_2, ...a_n\}$ is a finite nonempty set of conditional attributes, $D = \{d_1, d_2, ...d_q\}$ is a finite set of decision attributes that represent classes of objects. In this thesis, we assume $D = \{d\}$, where d is a single decision attribute having different decision values, V_a is the domain of attribute a and $f_a : U \rightarrow V_a$ is a function that maps an object x in U to exactly one value in V_a. The notation $f_a(x)$ denotes the object x value of attribute a. In this thesis, for simplicity, the notation $a(x)$ used for referring $f_a(x)$, and the decision system can be represented in short form as $CDS = (U, A \cup \{d\})$

The *indiscernibility relation* is the main concept for defining approximations.

Definition 2.1. For the given decision system CDS, let $P \subseteq A$, an indiscernibility relation $IND(P)$ is defined as [77],

$$IND(P) = \{(x, x') \in U^2 \mid \forall a \in P \ (a(x) = a(x'))\} \tag{2.1}$$

For two objects $x, x' \in U$ and if $(x, x') \in IND(P)$ then x and x' are indiscernible (indistinguishable) by all the attributes of P. The indiscernibility relation determined by P is called as P-indiscernibility relation. $IND(P)$ is an equivalence relation as it satisfies the reflexive, symmetric, and the transitive properties. The equivalence relation $IND(P)$ induces

a *partition*[1] of the universe of objects U into a family of disjoint subsets called equivalence classes. The set of equivalence classes of U that are determined by the indiscernibility relation $IND(P)$ are denoted as $U/IND(P)$ (or U/P), and the equivalence class that includes x is denoted as $[x]_P$, where $[x]_P = \{y \in U | (x, y) \in IND(P)\}$. The set of equivalence classes U/P is also called as *approximation space* or *granular space*, and each equivalence class in U/P is also called as a *granule*. Since U/P is a partition of U, the following properties are satisfied.

1. If $gr \in U/P$ is any granule, then $gr \subseteq U$.
2. For any two distinct granules $gr, gr' \in U/P$, $gr \cap gr' = \phi$.
3. $\bigcup_{gr \in U/P} gr = U$

According to Prof. Pawlak [77], rough set approximations are defined as given below.

Definition 2.2. For the given decision system CDS, let $P \subseteq A$ and $X \subseteq U$. The concept X can be approximated using only the information contained in P by constructing the *P-lower* and *P-upper approximations* of X, denoted by $\underline{P}(X)$ and $\overline{P}(X)$ respectively as given below.

$$\underline{P}(X) = \{x \in U \mid [x]_P \subseteq X\} \tag{2.2}$$

$$\overline{P}(X) = \{x \in U \mid [x]_P \cap X \neq \phi\} \tag{2.3}$$

Definition 2.2 states that, the lower approximation $\underline{P}(X)$ of the concept X (in the space of P) is a set of objects x which belongs to the equivalence classes contained in X. And, the upper approximation $\overline{P}(X)$ of the concept X (in the space of P) is a set of objects x from the union of all the equivalence classes, which have non-empty intersection with X.

X is said to be *definable set* if $\underline{P}(X) = \overline{P}(X)$ otherwise it is said to be *rough set*.

From [77, 78, 79], for the given decision system CDS, the *positive region* is defined as given below.

$$POS_P(\{d\}) = \bigcup_{X \in U/\{d\}} \underline{P}(X) \tag{2.4}$$

The positive region $POS_P(\{d\})$ contains the objects of U that are classified certainly into one of the decision granules of $U/\{d\}$ using the information of attribute set P.

2.2 Rough set based attribute reduction

As stated earlier, the reduct is a minimal subset of conditional attributes that preserves the original classification as defined by conditional attribute set. Reduct computation methods

[1]Partition of a set is the collection of disjoint subsets where it does not contain an empty set, the union of all the subsets is equal to the given set, and the intersection of any two subsets is an empty set.

are classified into many categories. However the primary categories include: (i) *Dependency measure* based, and (ii) *Discernibility matrix* based [44, 53, 107, 116, 132]. This section explains the basic concepts underlying both approaches.

2.2.1 Dependency measure based reduct computation

The dependency measure denotes the classifiability of a decision system, in other words it represents dependency of decision attribute on the conditional attribute set of a decision system. Different dependency measures exist in the literature, however, we use *gamma measure (γ)* to propose dependency measure based approaches in this thesis.

For the given decision system CDS, the dependency measure (gamma measure[1]) of decision attribute $\{d\}$ over the subset of conditional attributes P is given by [78, 79],

$$\gamma_P(\{d\}) = \frac{|POS_P(\{d\})|}{|U|} \tag{2.5}$$

(Note: In this thesis, the notation $|Z|$ for any set Z denotes the cardinality of Z)

The *gamma* measure $\gamma_P(\{d\})$ gives the proportion of objects belonging to the positive region of P. If $\gamma_P(\{d\}) = 0$ then classification $\{d\}$ is independent of the attributes in P, and the information in P is not useful for classification. If $\gamma_P(\{d\}) = 1$ then $\{d\}$ is completely dependent on P. And the values in $0 < \gamma_P(\{d\}) < 1$ indicate the partial dependency. If $\gamma_A(\{d\})$ is 1, then decision system is said to be *consistent* (i.e., all the objects in the decision system are classifiable) otherwise it is said to be *inconsistent*.

From Eq. (2.5), an attribute $a \in P$ is said to *indispensable* (useful or essential) attribute, if $\gamma_{P-\{a\}}(\{d\}) < \gamma_P(\{d\})$, otherwise it is said to be *dispensable* (redundant). Note that the dispensable attributes are superfluous and they do not contribute in the classifiability of the system, thus these attributes should be removed in the process of reduct computation. Therefore, based on the dependency measure approach the reduct is defined as given below.

Definition 2.3. For the given decision system CDS, let R be the subset of conditional attributes ($R \subseteq A$), and R is said to be reduct if and only, if R satisfy the following two conditions,

i). $\gamma_R(\{d\}) = \gamma_A(\{d\})$ (*jointly sufficient*)

ii). $\gamma_{R'}(\{d\}) < \gamma_R(\{d\})$ for any $R' \subset R$ (*individually necessary*)

In the above definition, the jointly sufficient condition states that, the *gamma* measure of reduct attribute set is collectively sufficient to induce the same *gamma* measure of conditional

[1]The terms "dependency measure" and "*gamma* measure" are used interchangeably in the rest of the thesis.

attribute set (i.e., $\gamma_R(\{d\}) = \gamma_A(\{d\})$), and the individually necessary condition states that none of the reduct attributes can be omitted as each of them are necessary (i.e., $\gamma_{R'}(\{d\}) < \gamma_R(\{d\})$ for any $R' \subset R$). Note that, the minimal subset of attributes (reduct) is computed using an attribute reduction algorithm. Quick Reduct Algorithm (QRA) [16] is regarded as an illustration of dependency measure based reduct computation.

2.2.1.1 Quick reduct algorithm

Algorithm 2.1: Quick Reduct Algorithm (QRA)

Input: Decision sytsem $CDS = (U, A \cup \{d\})$
Output: Reduct R

```
1  Initial Reduct R = φ
2  repeat
3  |  Temp = R
4  |  for each a ∈ (A − R) do
5  |  |  if (γ_{R∪{a}}({d}) > γ_Temp({d})) then
6  |  |  |  Temp = R ∪ {a}
7  |  |  end
8  |  end
9  |  R = Temp
10 until (γ_R({d}) == γ_A({d}))
11 Return R
```

Chounchoulas et al. [16] proposed Quick Reduct Algorithm (QRA) for reduct computation in CDS. The QRA uses search strategy of Sequential Forward Selection (SFS) for generating the reduct. The pseudo code of QRA is given in Algorithm 2.1.

In Algorithm 2.1, the QRA takes decision system CDS as input and produces super reduct as the output. According to this algorithm, initial reduct R is set as empty set (ϕ). Initially, the dependency measure $\gamma_A(\{d\})$ is computed for checking the end condition of the algorithm. From every iteration, an attribute $a \in (A - R)$ is selected for which maximum gamma gain (i.e., maximum dependency gain $\gamma_{R\cup\{a\}}(\{d\}) - \gamma_R(\{d\})$) is obtained. Algorithm is terminated when gamma measure of the obtained Reduct $\gamma_R(\{d\})$ is equals to the gamma measure of all attributes set $\gamma_A(\{d\})$. The computation of $\gamma_{R\cup\{a\}}(\{d\})$, $\forall a \in (A-R)$ is the main complexity in each iteration of the algorithm.

From Algorithm 2.1, it can be observed that the QRA computes the next best attribute in each iteration that should be added to the reduct set R. The termination condition of the algorithm satisfies the first condition of the reduct given in Definition 2.3 (i.e., $(\gamma_R(\{d\}) = \gamma_A(\{d\}))$). But the algorithm may not satisfy the second condition, where the reduct should not contain dispensable attributes. Because, QRA follows SFS strategy that can not assure

the addition of only indispensable attribute to the reduct in each iteration. There is no guarantee that SFS strategy generates minimal reduct, but it generates the reduct with the size almost close to minimal. Thus, the reduct produced by QRA may contain dispensable attributes. Therefore, the reduct generated by QRA is a *super reduct*[1]. In practice, it has been observed that the redundancy in the reduct of QRA is extremely small and does not hamper the quality of the reduct in inducing the good classification models.

2.2.2 Discernibility matrix based reduct computation

As previously described, the main idea in dependency measure based attribute reduction approaches is indiscernibility relation. Alternatively, the complementary relation of indiscernibility relation named as *discernibility relation* is used for reduct computation. This can be determined by complementing the indiscernibility relation given in Eq. (2.1). For the given decision system CDS, for a subset $P \subseteq A$, the discernibility relation is given as,

$$DIS(P) = \{(x, x') \in U^2 \mid (x, x') \notin IND(P)\} \tag{2.6}$$

The above Eq. (2.6) can be rewritten as given below.

$$DIS(P) = \{(x, x') \in U^2 \mid \exists a \in P \; [a(x) \neq a(x')]\} \tag{2.7}$$

Eq. (2.6) or (2.7) states that, The discernibility relation $DIS(P)$ contains the pair of objects which discern on at least one attribute in P. Further, the discernibility relation in a decision system is constructed for the pair of objects which discern and their decision classes are different, hence it is *decision relative discernibility relation* and is given below.

$$DIS(P) = \{(x, x') \in U^2 \mid \exists a \in P \; [a(x) \neq a(x')] \wedge d(x) \neq d(x')\} \tag{2.8}$$

The discernibility relation $DIS(P)$ satisfies the symmetric property, but it does not satisfy the reflexive and transitive properties.

The discernibility relation of a decision system can be represented with a discernibility matrix (DM) [98], in which each entry contains a set of attributes that discern a pair of objects. For the given decision system CDS, the discernibility matrix is a symmetric matrix $U \times U$. Thus, we can consider either only lower diagonal or upper diagonal entries. And, the matrix contains the entries between the objects of different decision classes, the remaining

[1] In the rest of the thesis, super reduct is termed as reduct, as all the proposed approaches follow the SFS strategy for reduct generation.

entries are empty. Hence, the matrix is a *decision relative discernibility matrix*[1] [55, 98]. In the matrix, an entry $C_{xx'}$ between each pair of objects $x, x' \in U$ is given as,

$$C_{xx'} = \begin{cases} \{a \in A \mid a(x) \neq a(x')\} & if\ d(x) \neq d(x') \\ \emptyset & otherwise \end{cases} \tag{2.9}$$

An entry $C_{xx'}$ in the matrix states that the objects x and x' can be distinguished by any attribute in $C_{xx'}$. A *discernibility function* can be derived from the discernibility matrix. The discernibility function is a boolean function defined on the power set of attribute set. If the given attribute set has the ability to discern all the possible pair of objects of the decision system belonging to different decision classes, then this function evaluates to TRUE; otherwise, it evaluates to FALSE.

For the given decision system CDS, let $a_1^*, a_2^*, ...a_n^*$ be boolean variables correspond to the attributes $a_1, a_2, ...a_n$ respectively. The discernibility function f_{CDS} is defined in terms of boolean variables as given below.

$$f_{CDS}(a_1^*, a_2^*, ...a_n^*) = \wedge\{\vee C_{xx'}^* | 1 \leq x' \leq x \leq |U|, C_{xx'} \neq \emptyset\} \tag{2.10}$$

Here, $C_{xx'}^* = \{a^* | a \in C_{xx'}\}$. The expression $\vee C_{xx'}^*$ in the above equation denotes the disjunction of boolean variables associated with the attributes in $C_{xx'}$. By applying the absorption and distribution laws, the discernibility function can be simplified, where conjunction of reduced disjunctive normal forms are obtained. Every conjunctor in the reduced disjunctive form is referred to as a *prime implicant*. The set of prime implicants of discernibility function is equivalent to the set of all minimal reducts of the given decision system [98]. Based on discernibility matrix approach [98], the reduct is defined as given below.

Definition 2.4. For the given decision system CDS, let $R \subseteq A$, $C_{xx'}$ $(\forall x, x' \in U)$ be an entry of the discernibility matrix (DM) of the given decision system, and R is said to be reduct if and only if R satisfy the following two properties,

i). $\forall C_{xx'} \in DM\ [C_{xx'} \neq \emptyset \Rightarrow R \cap C_{xx'} \neq \emptyset]$ (*jointly sufficient*)

ii). $\forall a \in R,\ \exists C_{xx'} \in DM\ [C_{xx'} \neq \emptyset \wedge ((R - \{a\}) \cap C_{xx'} = \emptyset)]$ (*individually necessary*)

Property (i) demonstrates that, in the given decision system, the reduct R is jointly sufficient for distinguishing all discernible object pairs. And, Property (ii) implies that each attribute in reduct R is individually necessary.

[1] In this thesis, the discernibility matrix is used as synonymous to decision relative discernibility matrix

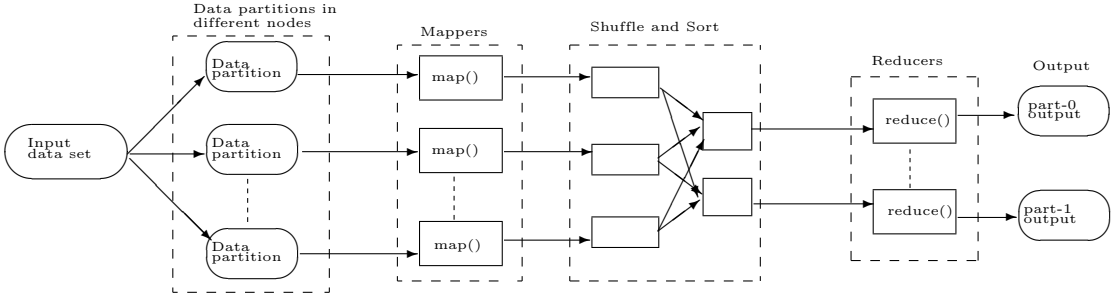


Figure 2.1: Overview of MapReduce programming model

2.3 MapReduce programming model

MapReduce framework was first introduced by Google Inc. researchers Jeffrey Dean and Sanjay Ghemawat [25]. It is a parallel/distributed programming model used for large-scale data processing. It reduces the amount of work required to write code that can run on a cluster of computers (nodes) because it provides a simple API that the programmer can use. The MapReduce system provides an abstraction that allows programmers to use a simple model while hiding the specifics of parallelization, load balancing, and fault tolerance.

The nodes in the cluster of any MapReduce framework are categorized as master (driver node), workers (slave nodes). The master is in-charge of assigning jobs to the workers, monitoring the jobs, and re-executing failed tasks. Workers follow instructions from their master and carry out their assigned tasks. A MapReduce program runs in three phases: *map, shuffle and sort*, and *reduce*. The main code of the MapReduce program runs on the master, the mapper and reducer codes run on the worker nodes. To accomplish the task given by the master, the mappers and reducers run on all the nodes in the cluster in an isolated environment, that is, they are unaware of each other and their jobs are equal on every node.

The $< key, value >$ pair forms the basic data structure in any MapReduce framework. In this programming model, the programmer writes the code in the form of mapper and reducer with the following signatures:(The convention [...] denotes a list.)

$$map :< key_1, value_1 > \rightarrow [< key_2, value_2 >]$$
$$reduce :< key_2, [value_2] > \rightarrow [< key_3, value_3 >]$$

As shown in Figure 2.1, the execution framework can be summarized as follows.

1. The driver gets input data from the distributed file system and distributes it as partitions (data splits) to different mappers located in different nodes (workers) of the cluster. The mapper code is applied on every $< key, value >$ pair and after processing, it produces an arbitrary number of intermediate $< key, value >$ pairs.

2. All these intermediate $< key, value >$ pairs are grouped by the key, that can be achieved by a large-scale distributed shuffling that involves all the nodes that executed map tasks and all the nodes that will execute reduce task. Hence this intermediate data must be copied over the network, and lot of communication takes place across the cluster of nodes, this phase of the framework known as *shuffle and sort*. A job with M mappers and R reducers involve maximum $M \times R$ distinct copy (data transfer) operations, this leads to significant burden on the framework, since many disk and network I/O operations are required to transfer the entire data and lot of communication happens across the network. Under big data work loads, to get the high performance, minimizing the shuffle and sort work and distributed coordination is important.

3. Each reducer gets intermediate data in order, that is sorted by key. The reducer is applied on all values correspond to the same intermediate key to produce output $< key, value >$ pairs. Now the driver collects this output data and writes it to distributed file system.

Following are the key advantages of the MapReduce paradigm.

- **Horizontal scalability**: It is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands. The MapReduce system's computing power can be increased by adding more nodes to the cluster. By default, MapReduce splits input data into partitions and distributes them across the available machines, so having more machines means having less data to process.

- **Fault tolerance**: It refers to the ability of a system to keep running even though one of its components fails. In MapReduce framework, to identify a failure in the system, the master will ping every worker on a periodic basis. If any worker does not give response after certain time, then it is set as "failed", and the task of that worker is rescheduled to other worker in the cluster. If a machine fails, all map tasks that have completed must be executed again because their output is stored on the failed machine's local hard disk(s) and therefore inaccessible. The reducer tasks that have been completed should not be re-executed because their output is stored in a distributed file system.

- **Simple programming model**: MapReduce has simple programming model, where a programmer needs not look into implementation details of parallelism, distributed data passing, or any other complexity. In a MapReduce framework, a programmer needs to write a program in terms of map and reduce functions. MapReduce not only makes the coding process ease and efficient but it also reduces the time to develop the programs.

Apache Hadoop [1], Apache Spark [121], Twister [35], HaLoop [12], etc., are some of the existing MapReduce frameworks. Even though, Apache Spark and Apache Hadoop are two different large-scale data processing frameworks, they are alike in many ways. Apache Spark offers several benefits over Hadoop. It was designed mainly to process iterative tasks in memory, which was one of Hadoop's major limitation. Additionally, from authors' [95, 119, 120] assessments have found that, Spark outperforms Hadoop by running applications up to 100 times faster in memory and 10 times faster on disk. Further, we can translate Hadoop programs directly to Spark: i.e., the Spark primitives are a superset of Hadoop. It should be noted that other frameworks such as Twister [35] and HaLoop [12] have attempted to address the inefficiency of iterative job handling of Hadoop. Despite favoring iterative MapReduce jobs, they could be regarded as subsets of Spark functionality [50, 51]. The proposed approaches in this thesis are implemented on Apache Spark framework. One of the reasons to choose Apache Spark is that, it supports iterative and in-memory computations. Another reason is that, as compared to other iterative in-memory frameworks (such as Twister [35], HaLoop [12]), Spark provides robust support for fault tolerance [50, 51].

2.4 Apache Spark

Apache Spark [121] is a fast computing framework which has the compatibility with Hadoop MapReduce model. Apache Spark is developed at the University of California, Berkeley's AMPLab, which published it in the year 2014, and the Apache Software Foundation now maintains it. Even after all these years, it is currently one of the most popular big data analysis frameworks. In an increasingly wide range of industries, Apache Spark has become the standard for large-scale data processing and data science. Spark has built in libraries such as Spark SQL, Spark Streaming, MLlib (for machine learning) and GraphX (for graph processing). These libraries are frequently used by businesses and academics throughout all sectors to handle complex problems.

2.4.1 Resilient Distributed Data set (RDD)

The special feature of Spark, that made it as unique computing framework is its primary data structure RDD. *It is an immutable collection of data items distributed across the nodes of the cluster, and can be manipulated in parallel.* The word *immutable* meaning is that, RDD can not be changed once it is created. Each word in the abbreviation of RDD has its own significance.

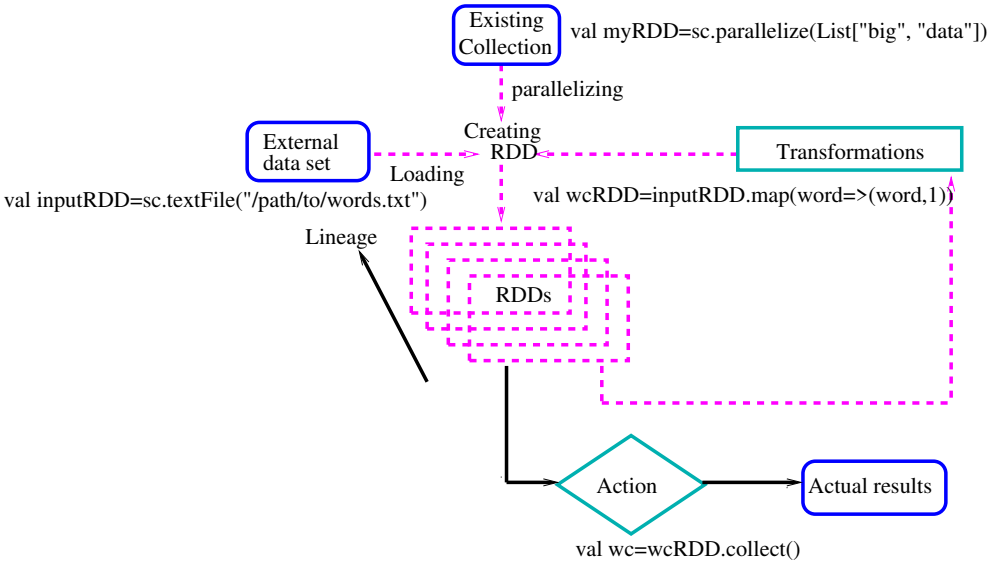


Figure 2.2: Creation of Spark RDD and its operations

- *Resilient*: This word in RDD represents the fault tolerant, where the missing or damaged partitions due to node failures can be recomputed.
- *Distributed*: It denotes that the data is distributed across the nodes of the cluster.
- *Data set*: It represents the records of the data.

Operations performed on RDD can be classified into two categories, namely *transformations* and *actions*. The transformation operation is used to generate a new RDD from the existing RDD. The actual results from the RDDs are produced by performing action operation. The new RDDs can be generated by performing transformations on the existing RDDs or by loading external data sets or by parallelizing the existing collection. Figure 2.2 shows the creation of the RDDs and few operations on RDDs with associated code lines given in Scala programming language.

Spark utilizes RDD and DAG (Directed Acyclic Graph) in achieving important features: lazy evaluation, fault tolerance, iterative and in-memory computation. DAG contains the lineage of RDD with all operations (transformations and actions) required to complete a task. Here, RDD's lineage refers to the previous RDDs on which it depends. The DAG gives the logical execution flow of RDDs.

- *Lazy evaluation*: All transformations performed on RDDs are lazy in nature, meaning that the actual result is not generated immediately after the operation, but instead a new RDD is constructed from the old one. All these transformations are added to the

DAG, and the actual results are obtained when an action operation is invoked. Unless the action is executed, the input file is not even read into memory. This enables Spark to make optimization decisions, because all transformations are seen before any action is taken by Spark. Disk and memory usage is improved with lazy evaluation.

- *Fault tolerance*: Spark is built to deal with the failure of worker nodes. This fault tolerance is achieved by using RDD and DAG. Since DAG contains the lineage of RDD, when a worker node fails, the same results of that node can be obtained by re-executing the steps of the lineage in the existing DAG. Note that, in lineage of an RDD, it remembers how it was created from other RDD to recreate itself.

- *Iterative and in-memory computation*: Spark RDDs have in-memory computing facility as it stores intermediate results in memory (RAM) instead of disk. This feature greatly boosts Spark performance. Additionally, through RDDs, Spark is capable of *caching* the intermediate results to help future iterations. By doing this, Spark gets an added better performance for iterative and repetitive processes, that can generate results and data in one step that can be reused later. Another way that a programmer may show which RDDs should be re-used is with the *persist* method. Usually, persistent RDDs are saved to RAM but can be dumped to disk if there is not enough memory. The programmer may provide additional options in persistence methods, such as saving the RDD on disk and memory or only on disk or replicating it between nodes.

2.4.2 Operations on RDD

Spark RDD supports two types of operations: *transformations* and *actions*. In the definition of RDD, it is given that the RDD is immutable, that is, RDD can not be changed once it is created. Here, the meaning of the word "immutable" should be understood correctly. Its meaning is that, when we perform transformations or actions on RDD, then new RDDs or results are produced without changing the existing RDD.

2.4.2.1 Transformations

Transformation is a function that generates new RDD from the existing one. Because RDDs are read-only, the transformation does not affect the original RDD (existing RDD). All the transformations applied on RDD built an RDD lineage which is represented with a DAG. As mentioned earlier, transformations are lazy in nature, they are not executed immediately after their creation, instead they are materialized once action operation is performed. There

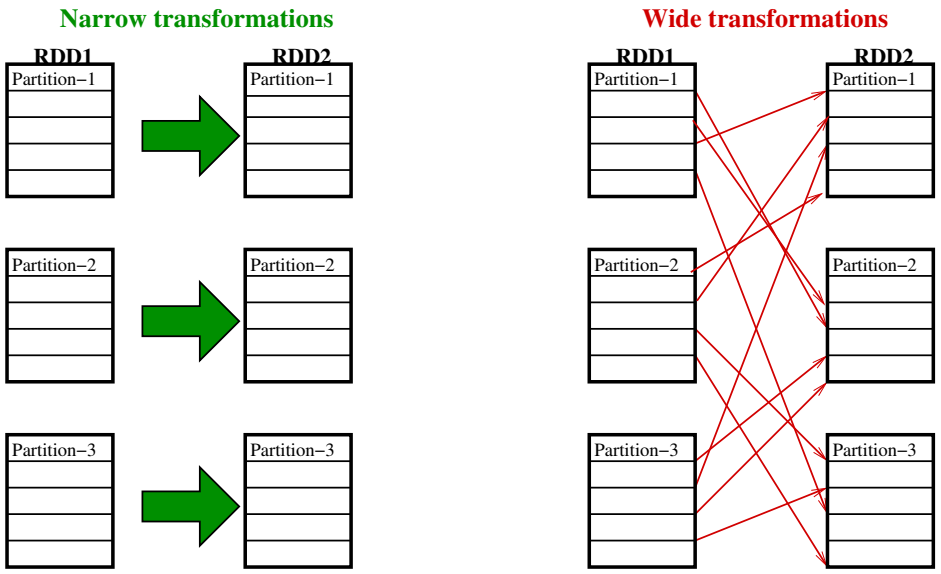


Figure 2.3: Types of RDD transformations

are two types of transformations: *narrow transformations* and *wide transformations*. These two types of transformations are shown in Figure 2.3.

- *Narrow transformations*: The transformations in this category convert each input partition to a single output partition. These transformations occur, if each partition of the parent RDD is utilized by single partition of the child RDD, or if each child RDD partition is created by or is dependent on a single parent RDD partition. This type of transformation is essentially faster, because it does not necessitate any shuffling of data or data movement through the cluster network. Some of the narrow transformations frequently used in implementing the approaches in this thesis are: map(), flatMap(), mapPartitions() and filter().

  - *map()* and *flatMap()*: These two transformations are similar in functionality. With map() method, every item in RDD is transformed into one item in the resulting RDD. That is it performs one-to-one transformation. Where as flatMap() method transforms every item in RDD into multiple (0 or more) items in the resulting RDD. That is, it is a one-to-many transformation.

  - *mapPartitions()*: This method is similar to map(). Using the map() method, we utilize the function at a per-element level, while using the mapPartitions() method, we work with partition level. The mapPartitions() transformation is faster because it calls the function once per partition, rather than once per element.

  - *filter()*: While executing a filter() transformation, we supply it with a boolean function that is responsible for identifying RDD elements satisfying the given condition and returns a subset of the RDD containing elements on which the boolean function returns true.

- *Wide transformations*: The transformations in this category will have input partitions contributing to multiple output partitions. These transformations occur, if each partition of the parent RDD is utilized by multiple partitions of the child RDD, or if each child RDD partition is created by or is dependent on multiple parent RDD partitions. This implies that data would be moved between partitions in order to carry out wider transformations. These are also known as shuffle transformations because they shuffle the data. These transformations are slow in comparison to narrow transformations when constructing new RDD partitions, it may be necessary to shuffle data across various nodes, which may have a major impact on processing speed of the framework. Examples for wide transformations include: groupByKey(), aggregateByKey(), reduceByKey(). One of the frequently used wide transformations in implementing the proposed approaches is *reduceByKey()*.

  - *reduceByKey()*: This method gets an RDD in the form of $< key, value >$ pair as input, and aggregates *value* portions of the same *key*, and generates the output RDD in the form of $< key, value' >$ pair. Here, the data type of *value* and $value'$ is same.
  - *aggregateByKey()*: This method is logically equivalent to reduceByKey() but it allows to return result in different type. It aggregates the values of each key, using given aggregate, combine functions and a neutral "zero value". Here, the "zero value" input argument denotes the start value of an accumulator.
  - *groupByKey()*: It creates a single sequence from the values for each key in the RDD. During this transformation, lots of unnecessary data transfer over the network. This method receives $< key, value >$ pairs as an input, group the values based on the *key* and generates an RDD of $< key, [value] >$ pairs as an output. Here, the notation [ ] indicates list of values.

From the Figure 2.3, it can be understood that, to make applications run faster when operating with Spark, it is best to use narrow transformations as much as possible while reducing the use of wide transformations. Some times usage of wide transformations is unavoidable, in that case alternative strategies should be incorporated to reduce the shuffling of the data or movement of the data (in shuffle and sort phase) in the cluster network.

2.4.2.2 Actions

Action is a function that produces actual results from the given RDD. Thus, action operation on RDD returns a non-RDD values. An action is one of the approaches to transfer the data from executor (worker node) to the driver (master node). The action results are saved to driver or to an external storage system. The action operations that are frequently used in implementing the approaches in this thesis are: reduce(), collect() and count().

- *reduce()*: This action computes the aggregation of an RDD's elements by repetitively applying a function that gets two RDD elements as input and returns a new element as an output. Finally, a single aggregated value is returned.
- *collect()*: This action returns an array to driver containing all the data items in the RDD. The driver's (master machine) maximum memory can be exceeded if care is not taken when running this action.
- *count()*: This function returns the number of data items present in the input RDD.
- *saveAsTextFile()*: This function saves the path of a file and writes the content of the RDD to that file.

2.4.3 Data partitioning in Apache Spark

Apache Spark framework reads the input file as an RDD and partitions and distributes it into the cluster of computers using the horizontal partitioning strategy. In this strategy, the input file is partitioned row wise, and each partition is distributed to a node of the cluster. Thus, every node in the cluster gets one or more data partitions.

In any MapReduce framework, the data partitioning help in parallelizing distributed data and processing with minimal network traffic across the cluster of computers. The number of partitions used in Spark is adjustable, and having too few (which results in reduced concurrency, data skewing, and inefficient resource usage) or too many (which results in task scheduling taking longer than real execution time) partitions is undesirable. Hence, making a decision on selection of the number of data partitions is a crucial step in achieving maximum performance of MapReduce based algorithm in Apache Spark. As given in the literature [50, 51], the number of data partitions recommended to be equal to the number of cores (some times may be 2 or 3 times of available cores) in the cluster to achieve the maximum parallelism. Thus, one or more partitions available in a node. Spark assigns a task per partition, and each core executes a task.

Internally, Apache Spark supports two types of partitioners to partition the list of $< key, value >$ pairs. They are *hash partitioner* and *range partitioner*. Depending on how keys in the data are distributed or sequenced, and the action that is to be performed on the data, the user can choose the appropriate partitioner. Spark uses HashPartitioner as its default partitioner. The data will be distributed uniformly across all partitions using HashPartitioner. The data is distributed to nodes based on the result of the hash function applied to each *key*. Hash partitioning has the potential to make distributed data skewed. With the range partitioning, tuples with *keys* in the same range will appear on the same computer. That is, a range partitioner partitions *keys* depending on the set of sorted range of *keys* and *key* ordering.

2.4.4 Run time architecture of Spark application

Spark follows master/slave run time architecture where, master (driver) acts as central coordinator that coordinate the slaves (workers or executors). The combination of user program, driver and its executors form a Spark application. The run time architecture of the Spark application contains three major components: driver, cluster manager and executors. Figure 2.4 shows the run time architecture of the Spark application. The role of each component is given below.

2.4.4.1 Role of driver

The *main()* method of the user program runs in the driver. In this method the *sparkContext* (referred with *sc*) is created. SparkContext is the core component of any Spark application. It is an handle to an instance of the Spark execution environment. And it is used to build RDDs, accumulators, and broadcast variables in Spark, as well as to access and run Spark services.

The execution is done and actual results are returned if an action is performed on RDD. In other words, when an action is performed then the driver creates a *job* from the user program. Then the driver creates a DAG which is a logical execution plan. After creating the DAG, the driver converts it into physical execution plan by splitting it into a number of *stages*. These stages are then subdivided into smaller *tasks*, which are then allocated to executors.

- *Job*: It is a parallel computation that consists of several tasks that are launched in response to Spark actions.
- *Stage*: Each job is subdivided into smaller groups of tasks called stages that are interdependent.

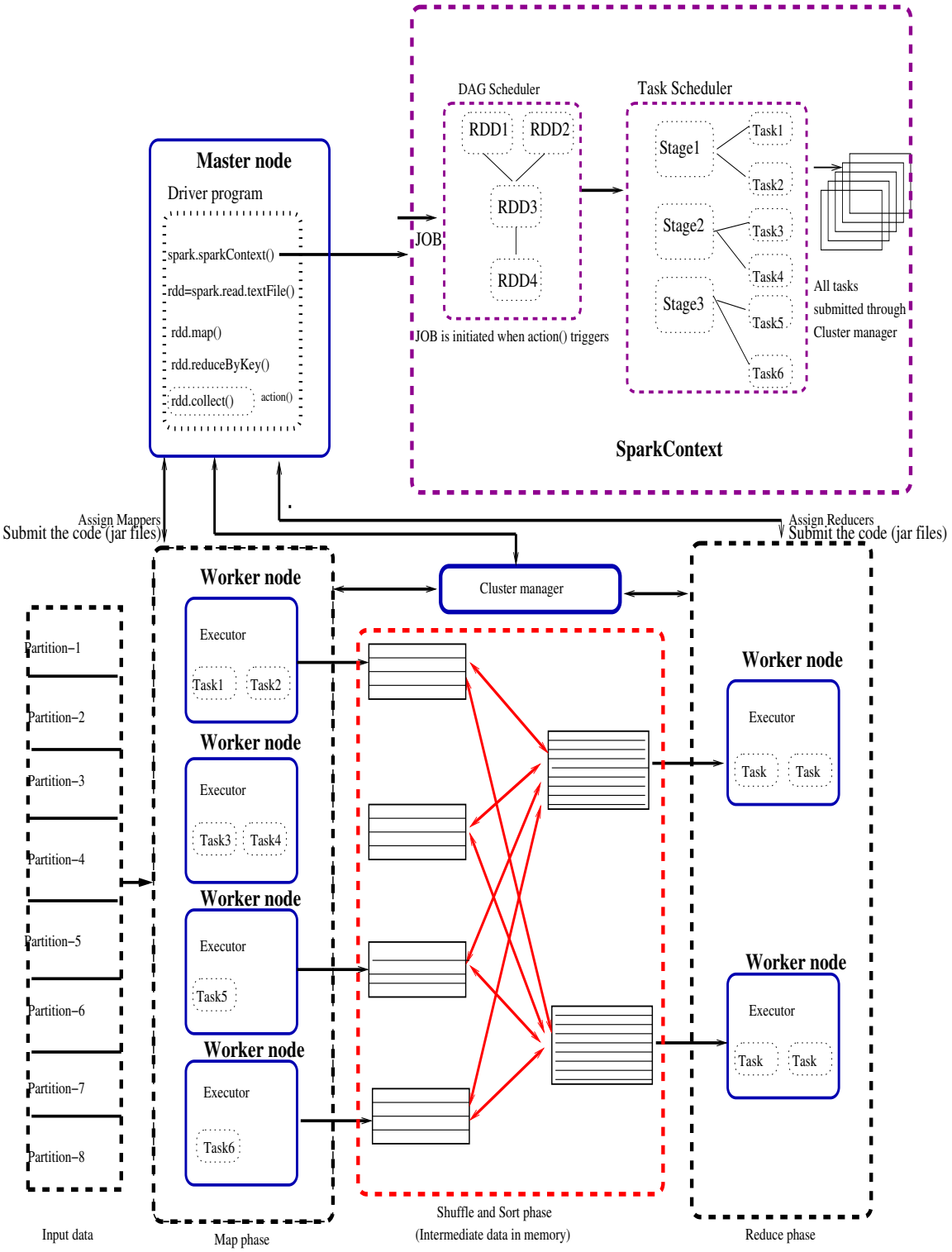


Figure 2.4: Run time architecture of Spark application

- *Task*: A task is a specific piece of work that is assigned to the executor. For each partition, a task is created.

Each stage contains the number of tasks which are equal to number of partitions in the cluster. Thus the number of tasks in a job can be calculated by multiplying the number of jobs with number of stages. Here, the creation of the stage depends on the shuffling of the data in the cluster network. When a transformation needs shuffling of the data to other partitions then driver creates stages for other partitions. If a transformation does not require shuffling then driver creates a single stage for it. after converting a job into number of tasks, the driver schedules them to the executors through a cluster manager. And finally, the driver collects the results from the executors on successful execution of Spark application.

2.4.4.2 Role of cluster manager

After converting the job into tasks, the driver communicates with cluster manager and negotiates for resources. A cluster manager is responsible for obtaining and allocating resources on the cluster to Spark jobs. Thus a Spark application is launched on a cluster of nodes using the cluster manager. Spark's default built-in cluster manager is *standalone cluster manager*. Apart from its integrated cluster manager, Spark supports other open source cluster managers, including *Hadoop Yarn*, *Apache Mesos*.

2.4.4.3 Role of executors

Executor is a distributed machine that is in control of task execution. An executor can run multiple tasks in parallel. Here, generally a task is allocated to a CPU core of the executor. The executor is responsible for all data processing. It interacts with the storage systems. The executor stores the results of computations in memory, a cache, or on hard disk or can return to the driver.

One of the primary aims of this thesis work is to examine different strategies for simplifying the shuffle and sort phase in the design of rough set based attribute reduction algorithms using Apache Spark. In the next chapter (Chapter 3), we investigate these alternative strategies for attribute reduction in categorical decision systems. And, in Chapters 4 and 5, we explore these strategies for attribute reduction in incomplete and hybrid decision systems.

Chapter 3

Parallel attribute reduction in Categorical Decision Systems

This chapter introduces the first contribution for this doctoral thesis. In this chapter, the relevance of vertical partitioning strategy is investigated for classical rough set based attribute reduction in Categorical Decision Systems (CDS) with big dimensionality. A MapReduce based parallel/distributed algorithm MR_IQRA_VP is developed in which vertical partitioning strategy is used for partitioning the input data set. With the vertical partitioning strategy, the data set is split over attribute space. It overcomes the problems involved in horizontal partitioning strategy which partitions the data over object space. The advantages and limitations of the proposed MR_IQRA_VP algorithm is theoretically and experimentally studied, inferences are obtained through comparative analysis with state-of-the-art horizontal partitioning based algorithms. The work discussed in this chapter has been published in [102].

3.1 Review of existing approaches

Rough set theory [77] has been successfully identified as an effective framework for attribute reduction in CDS. Several algorithms [16, 24, 47, 48, 67, 68, 69, 74, 83] in rough set theory have been developed over the previous decades to accomplish efficient reduct computation. Quick Reduct Algorithm (QRA) [16] is one of the key traditional reduct computation algorithms. Sai Prasad et al. extended this algorithm to IQRA_IG (Improved QRA) [74] by adding the features of *handling the trivial ambiguous situation*, *granular refinement*, and *positive region removal*.

The amount of data generated each day has increased exponentially during the previous several years. If the data set is enormous in volume or/and dimension, the aforementioned classical reduct computing algorithms fail to perform adequately due to their sequen-

tial nature. For scalable attribute reduction in CDS, the researchers considered that parallel/distributed computation is the optimal method. As a result, researchers [7, 26, 60, 87, 104, 106] attempted to parallelize standard attribute reduction approaches in order to increase their efficiency when dealing with large-scale data sets.

In recent years with the proliferation of MapReduce model for parallel/distributed computation, several scalable reduct computation algorithms [21, 45, 81, 114] have been developed for large-scale CDS using Apache Hadoop MapReduce framework [1]. But implementing iterative parallel algorithms on the Hadoop platform was found to be inefficient. The lapse was because of the problem of frequent storing/loading of the data into/from distributed memory.

In order to overcome the problems of Hadoop MapReduce framework, many authors proposed parallel algorithms [15, 29, 92, 96, 124] based on in-memory iterative MapReduce frameworks such as Apache Spark [2], and Twister [35]. J. Zhang et al. proposed Parallel Large-Scale Attribute Reduction (PLAR) [124] algorithm based on Apache Spark framework. Sai Prasad et al. developed a scalable IN_MRIQRA_IG algorithm [92], which is a parallel version of IQRA_IG [74], and it is implemented on the Twister framework.

Generally, a data set is viewed as a matrix, where rows indicate objects and columns indicate attributes. In any MapReduce framework, the input data set to the cluster is partitioned using the *horizontal partitioning strategy*. In this partitioning strategy, the data is partitioned in object space, and each partition is distributed to a node of the cluster (distributed by samples), as shown in Figure 3.1. Here, each data partition gets the information of all the attributes over a subset of objects. This means that if the data set is horizontally partitioned to the nodes of the cluster, every node has information of all the attributes over a subset of objects. Thus, in rough set based attribute reduction, the granules or equivalence classes (refer Section 2.2 in Chapter 2) construction is dependent on the information available across the nodes, which results in significant data movement across the nodes of the cluster. The number of candidate subsets over which the computations need to be performed for finding the reduct is directly proportional to the number of attributes in the decision system. From the existing MapReduce based parallel/distributed reduct computation algorithms such as IN_MRIQRA_IG (based on Twister [35]), PLAR (based on Apache Spark [2]), and PFSPA (based on Hadoop [1]), we observed that, as attribute space size increases, the running time of horizontal partitioning based reduct computation algorithm also grows significantly. This has been especially observed while working with big dimensional data sets of Bioinformatics, i.e., microarray data sets.

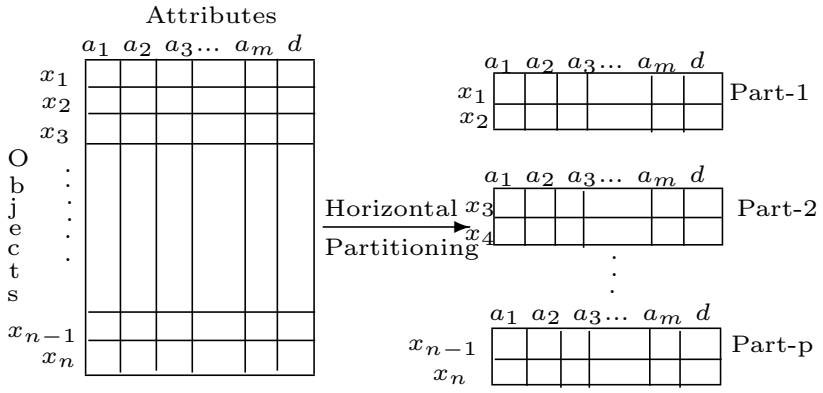


Figure 3.1: Horizontal partitioning of the input data

From the review of literature, it is observed that, horizontal partitioning based approaches for attribute reduction in rough set theory are efficient for the data sets with huge number of objects. Researchers have concentrated on only one aspect of large-scale data sets, namely massive number of objects, while paying little attention to the attributes aspect. Hence, all the rough set theory based attribute reduction approaches using the MapReduce programming model [15, 21, 29, 45, 81, 92, 96, 114, 124] adopted horizontal partitioning strategy. Thus, in the present work, we explore alternate partitioning strategy that perform well for the large-scale data sets which are big dimensional.

From the literature, we noticed that, two non-rough set based methods [88, 89] used *vertical partitioning strategy* to deal with big dimensional data sets. This strategy is an alternative to horizontal partitioning strategy. Authors of these methods demonstrated the utility of vertical partitioning strategy for feature ranking (but not for feature subset selection) using MapReduce programming model. Thus, in this chapter the relevance of vertical partitioning strategy is investigated in rough set based attribute reduction. Here, a MapReduce based algorithm MR_IQRA_VP using vertical partitioning strategy is proposed which is designed based on the existing *Improved Quick Reduct Algorithm (IQRA_IG)* [74]. The proposed MR_IQRA_VP algorithm is implemented and compared using the Apache Spark MapReduce framework [2]. The relevance and limitations of this algorithm is provided with extensive experimental analysis along with theoretical validation.

3.2 Related work

This section describes the sequential and parallel version of IQRA_IG algorithm, their significant features. It also presents the process of reduct computation involving horizontal partitioning of the data.

3.2.1 Sequential IQRA_IG algorithm

Sai Prasad et al. proposed IQRA_IG algorithm [74], which is an improved version of Quick Reduct Algorithm (QRA) [16]. The significant features of this algorithm are, granular refinement, positive region removal and handling of a trivial ambiguous situation.

By recollecting from Section 2.2.1, the QRA algorithm starts its first iteration with the reduct R as an empty set (ϕ). For each conditional attribute $a \in A$, the dependency (*gamma*) measure (γ) that depends on the positive region has to be computed. To compute the positive region, the corresponding granular space is formed through sorting. Based on the attribute values, objects are sorted. The place of transition from one value of the attribute to the next attribute is identified. This results in the formation of granules. That is, granules $U/\{a\}$ are formed for any attribute $a \in A$. For sorting, Quick sort like comparison sorting based algorithm takes $\mathcal{O}(|U|log|U|)$ time and Radix sort like linear sorting algorithm takes $\mathcal{O}(|U|)$ time for forming granules. After the computation of positive region for each conditional attribute $a \in A$, the *gamma* measure (γ) is computed. Attribute for which maximum *gamma* obtained is included into the reduct set R.

In the subsequent iterations of QRA, when reduct is nonempty, the granules need to be computed with $R \cup \{a\}$, $\forall a \in (A - R)$. Granules $U/(R \cup \{a\})$ can be computed based on the values of objects in $R \cup \{a\}$. But this computation becomes redundant since computed granules U/R are available from the previous iteration. This redundant computation in each iteration is avoided by using *granular refinement* (refer Section 3.2.1.1) in IQRA_IG algorithm. Thus, by forming the granules $U/(R \cup \{a\})$, the *gamma* measure $\gamma_{R\cup\{a\}}(\{d\})$ is computed. From the granules of $U/(R \cup \{a\})$, the positive region granules are removed by using *positive region removal* (refer Section 3.2.1.2). From the next iteration, the granules are formed based on the non-positive region granules of $U/(R \cup \{a\})$. An attribute a is selected to the reduct, for which the *gamma* gain $(\gamma_{R\cup\{a\}}(\{d\}) - \gamma_R(\{d\}))$ is maximum. The algorithm terminates when *gamma* measure of the obtained reduct $\gamma_R(\{d\})$ equals the *gamma* measure of all conditional attributes set $\gamma_A(\{d\})$.

3.2.1.1 Granular refinement

Definition 3.1. For a decision system $CDS = (U, A \cup \{d\})$, let $Q \subseteq P \subseteq A$, granules U/P is a refinement over U/Q that denoted by $U/P \preceq U/Q$ where,

$$\forall gr \in U/P \Rightarrow \exists gr' \in U/Q \wedge gr \subseteq gr' \tag{3.1}$$

The above Eq. (3.1) is the outcome of the indiscernibility relation being an equivalence relation.

Definition 3.2. For a decision system $CDS = (U, A \cup \{d\})$, let $R \subseteq A$, $a \in (A - R)$ and $U/R = \{gr_1, gr_2, ...gr_r\}$, granular refinement for the computation of $U/(R \cup \{a\})$ is given by,

$$\begin{aligned} U/(R \cup \{a\}) &= GranularRefinement(U/R, a), \\ &where\ GranularRefinement(U/R, a) = \bigcup_{i=1}^{r} gr_i/\{a\} \end{aligned} \tag{3.2}$$

The granular refinement feature given in Eq. (3.2) is incorporated into the IQRA_IG algorithm. As a consequence, the granules $U/(R \cup \{a\})$ are computed by splitting the existed granules of U/R using the attribute values of a. That is, in each iteration of the algorithm, instead of newly forming the granules by using all the attributes of $R \cup \{a\}$, only granules of the previous iteration are refined by using the present attribute a. This results in huge computational gain for each iteration of the algorithm. Through granular refinement feature in the algorithm, sorting operation is not required on total objects set U. But objects in each granule $gr \in U/R$ are sorted independently to get the required granules of $U/(R \cup \{a\})$. Using the Quick sort algorithm to form the granules $U/(R \cup \{a\})$ the time complexity becomes $\sum_{i=1}^{r} \mathcal{O}(|gr_i|log|gr_i|)$. Granular refinement has the computational gains since $\sum_{i=1}^{r} \mathcal{O}(|gr_i|log|gr_i|) \leq \mathcal{O}(|R||U|log|U|)$.

3.2.1.2 Positive region removal

In an iteration of IQRA_IG, let R denotes the set of attributes already selected into reduct set. The granules of U/R are categorized into either *positive region granules* ($P\_GR(U/R)$) or *non-positive region granules* ($NP\_GR(U/R)$). A granule $gr \in U/R$ is a positive region granule when it is pure or consistent, It becomes pure if all the objects of gr belong to a single decision class, otherwise, gr is categorized as an inconsistent or non-positive region granule. If $gr \in P\_GR(U/R)$, then $\forall gr' \in gr/(R \cup \{a\})$ for any $a \in (A - R)$, we have $gr' \in P\_GR(U/(R \cup \{a\}))$. Because, if a granule is pure, then any of its sub granules is also pure. Based on granular refinement (Eq. (3.2)), computations in IQRA_IG in subsequent iterations are performed for each granule of U/R independently of other granules. Therefore, the omission of objects in $P\_GR(U/R)$ has no effect on future computations. This phenomenon is called positive region removal [74] or positive approximation [45].

Definition 3.3. For a decision system CDS, let $R \subseteq A$, and $P\_GR(U/R)$ denotes the positive region granules, then the positive region removal is given by,

$$U/R = NP\_GR(U/R) = U/R - P\_GR(U/R) \tag{3.3}$$

In an iteration of IQRA_IG algorithm, after the removal of positive region, only the non-positive region granules ($NP\_GR(U/R)$) remain in U/R. In the next iteration, only the granules of $NP\_GR(U/R)$ are used in selecting the next best attribute to the reduct set by applying granular refinement as $GranularRefinement(NP\_GR(U/R), attribute)$. This shows that the removal of positive region restricts the future computations to granules falling into the non-positive region. This results in a decrease of space utilization in successive iterations of the algorithm.

3.2.1.3 Handling trivial ambiguous situation

In an iteration of the algorithm, if there is no *gamma* gain, then the selection of an attribute from $(A - R)$ becomes difficult. And it may lead to the inclusion of redundant attribute into reduct. This situation is called trivial ambiguous situation in QRA. And this situation handled in IQRA_IG algorithm using the secondary heuristic of *information gain*. In our current study, we did not incorporate the trivial ambiguous situation and its resolution, because, in large-scale data sets, such occurrence is a rarity. Instead, a random selection from available attributes for inclusion into reduct has been incorporated. Hence a detailed explanation of this feature of IQRA_IG algorithm is ignored here (for the details refer [74]).

3.2.2 Horizontal partitioning based parallel IQRA_IG: IN_MRIQRA_IG

Sai Prasad et al. proposed MapReduce based parallel version of IQRA_IG algorithm as IN_MRIQRA_IG [92]. This section provides a summary of IN_MRIQRA_IG algorithm as an illustration for distributed computation involved in horizontal partitioning based MapReduce reduct computation algorithm. It should be noted that, IN_MRIQRA_IG algorithm is one of the few algorithms in the field of MapReduce based reduct computation having the aspect of positive region removal.

3.2.2.1 Horizontal partitioning based reduct computation

In horizontal partitioning strategy, the data is partitioned over the object space and distributed to the nodes of the cluster. Through this data distribution, every node has information of all the attributes over a subset of objects. Horizontally partitioned decision system can be formally represented as follows.

Definition 3.4. For a decision system $CDS = (U, A \cup \{d\})$, let $CDS = \bigcup_{i=1}^{p} CDS^i$ be a horizontally partitioned system, where $CDS^i = (U^i, A \cup \{d\})$ is i^{th} data split. It satisfies (i) $U = \bigcup_{i=1}^{p} U^i$ (ii) $U^i \cap U^j = \phi$, $\forall i, j \in \{1, 2, ...p\}$ and $i \neq j$.

From Definition 3.4, the decision system CDS is divided into p sub-decision tables or data splits (data partitions), which are distributed to the nodes of the cluster.

In the driver, reduct R is initialized to empty set (ϕ). Each mapper is associated with a data partition. The current reduct R is broadcasted to all the nodes by the driver. Each mapper can only construct partial granules, that is i^{th} mapper working on the data partition CDS^i and broadcasted R can construct partial granules $gr^i \in U^i/(R \cup \{a\})$ for all competing attributes $a \in (A - R)$. If gr^i is consistent, then $< key, value >$ pair is generated with the key as $< a, GS(gr^i) >$ and $value$ as $< |gr^i|, d(gr^i) >$. Here $GS(gr^i)$ denotes *granule signature* that contains attribute's unique value combination which are satisfied by objects of gr^i. Only $|gr^i|$ without objects information is included in $value$ portion because it is sufficient for computing $\gamma_{R\cup\{a\}}(\{d\})$. Notation $d(gr^i)$ denotes the unique decision value of objects of gr^i. And in cases where gr^i is inconsistent, the $key =< a, GS(gr^i) >$ and $value =< 0, -1 >$ are generated for representing inconsistency.

After shuffle and sort phase, the reducer receives a list of values corresponding to unique key. The reducer aggregates all the values as single value that results in formation of granule $gr \in U/(R \cup \{a\})$. Here $gr = \bigcup_{i=1}^{r} gr_i$ where, $GS(gr^i) = GS(gr^j)$, $\forall i, j \in \{1, 2, ...p\}$, and it follows that $GS(gr) = GS(gr^i)$, $\forall i \in \{1, 2, ...p\}$. If the value of $d(gr^i)$ from all the mappers is same, that is, if the granule is consistent, then corresponding $|gr^i|$ are added to the result as $|gr|$ and it produces a single $< key, value >=< a, |gr| >$ pair. But, if the granule is inconsistent then $< a, 0 >$ pair is generated. The driver computes $|POS_{R\cup\{a\}}(\{d\})|$, $\forall a \in (A - R)$ based on $< key, value >$ pairs received from reducers associated with $key = a$. Since the granules formed in reducers are same as the granules formed in sequential implementation such as in IQRA_IG algorithm, the result of IN_MRIQRA_IG algorithm is same as that of IQRA_IG algorithm. Finally, best attribute is selected, and added to the reduct R and the end condition is tested. Based on the test result, the driver either returns reduct R or continues the next iteration of the algorithm.

3.2.2.2 Positive region removal

Positive-region removal in IN_MRIQRA_IG algorithm is incorporated by obtaining positive granules signature of the current reduct set R in a separate MapReduce job called *Posgather*. Driver collects the $GS(gr)$, $\forall gr \in P\_GR(U/R)$ from the *Posgather*. In the subsequent iterations for attribute selection, the information of $GS(gr)$, $\forall gr \in P\_GR(U/R)$ is broadcasted. In the mapper phase, for the partial granule computations, only objects which are not satisfying any of the positive region granule signatures (i.e., objects that are of the $NP\_GR(U/R)$) are included. This results in effecting the positive region removal. Thus, in constructing

$< key, value >$ pairs in mapper phase, attribute information of $R \cup \{a\},\ \forall a \in (A - R)$ is required.

3.2.2.3 Granular refinement

The existing MapReduce based reduct computation approaches [15, 21, 29, 45, 81, 90, 92, 96, 114, 124, 125] do not incorporate the granular refinement aspect described in Section 3.2.1.1. In the horizontal partitioning approach, the knowledge of granules is realized at reducer phase as each has partial granules information only. To correctly obtain $U/R, \forall a \in A$ in the first iteration requires *object id* to be placed as part of *value* portion along with decision information. This results in $|U| * |A|$ amount of data (equals the original data set size) movement in shuffle and sort phase. The large amount of data can become a bottleneck for the realization of specific granular signature and hence have not been incorporated in existing algorithms. The granular refinement of the proposed work with vertical partitioning strategy described in Section 3.3 overcomes this limitation of horizontal partitioning strategy.

3.3 Proposed vertical partitioning of the data

The vertical partitioning strategy partitions the input data set over the attribute space (vertically) and distributed to the nodes of the cluster. With this strategy, all the values of an attribute are available in one record of one data partition located in a node of the cluster. By default, any MapReduce framework partitions the input data set by using horizontal partitioning strategy. This strategy divides the data set row wise (in object space), and partitions are distributed to the nodes of the cluster. Thus, the vertical partitioning strategy is realized by preprocessing the input data set before supplying it to the algorithm, either locally for data sets fitting in RAM or by using MapReduce approach otherwise. The given data set is preprocessed in such a way that all the rows indicate attributes, and the columns show the objects. Additionally, an entry is included at the beginning of the record for denoting the *attribute id*. The microarray data sets used for representing the gene expression data in Bioinformatics [4] are usually stored in rows represent attributes, and hence preprocessing is not required.

The preprocessed data is horizontally partitioned over attribute set A, and each partition represents data pertaining to a subset of A. Each partition requires decision attribute information for subsequent operations, and therefore the decision attribute $\{d\}$ is broadcasted to all the nodes by the driver. The vertical partitioning of a given decision system can be formally defined as given below.

3. PARALLEL ATTRIBUTE REDUCTION IN CATEGORICAL DECISION SYSTEMS

Definition 3.5. For a decision system $CDS = (U, A \cup \{d\},)$, let $CDS = \bigcup_{i=1}^{p} CDS^i$, where $CDS^i = (U, A^i \cup \{d\})$ is i^{th} data split. It satisfies (i) $A = \bigcup_{i=1}^{p} A^i$ (ii) $A^i \cap A^j = \phi$, $\forall i, j \in \{1, 2, ...p\}$ and $i \neq j$.

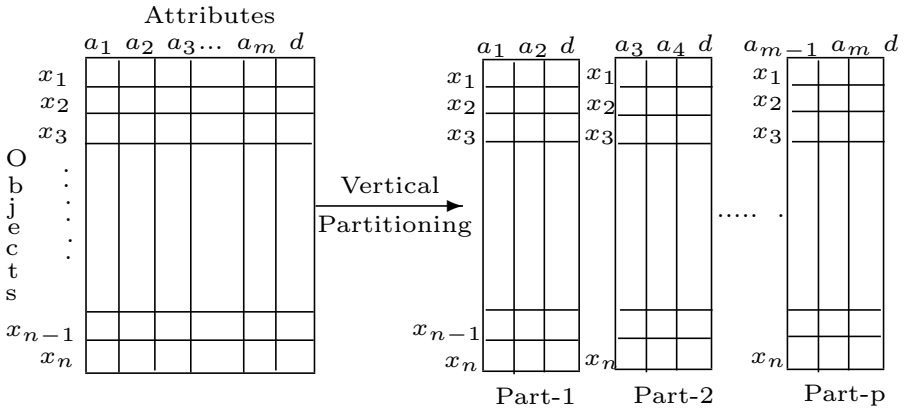


Figure 3.2: Vertical partitioning of the input data

As we can see in Figure 3.2, with vertical partitioning strategy, the decision system CDS is divided into p sub-decision tables or data splits (data partitions). A data split CDS^i in a node contains A^i attributes with all the objects U and broadcasted decision attribute $\{d\}$ (as shown in Figure 3.2). Let t be the number of nodes in the cluster, and p be the number of data partitions of the given data set. Without loss of generality we assume $p > t$. In the experiments, we adopted equal division of the load and each node receives $\lfloor \frac{p}{t} \rfloor$ data partitions. Therefore proposed algorithm initiates $\lfloor \frac{p}{t} \rfloor$ number of mapper tasks per node.

The primary operation of IQRA_IG algorithm [74] is the selection of the next best attribute to be included into the reduct set. In this section, we discuss the equivalence of attribute selection (in sequential approach) in a decision system CDS to parallel attribute selection over vertically partitioned sub decision tables $\left\{CDS^i\right\}_{i=1}^{p}$. In IQRA_IG algorithm, over the data set CDS, the selected next best attribute $a^{best} \in (A - R)$ satisfies the following property.

$$\gamma_{R \cup \{a^{best}\}}(\{d\}) = \max_{a \in (A-R)} \gamma_{R \cup \{a\}}(\{d\}) \tag{3.4}$$

Using Eq. (2.5), canceling the denominator on either side of Eq. (3.4) results in,

$$\left|POS_{R \cup \{a^{best}\}}(\{d\})\right| = \max_{a \in (A-R)} \left|POS_{R \cup \{a\}}(\{d\})\right| \tag{3.5}$$

In the proposed approach, as the data set CDS is available as CDS^i $(i = 1, 2, ...p)$ in the nodes of the cluster, the next best attribute selection need to be done locally for each sub

decision table (in mapper phase) in parallel, and globally through reducer phase. Hence, the selection criteria of IQRA_IG is equivalently expressed over $CDS^i,\ \forall i = \{1, 2, ...p\}$ as,

$$\left|POS_{R\cup\{a^{best}\}}(\{d\})\right| = \max_{a\in(A-R)} \left|POS_{R\cup\{a\}}(\{d\})\right| =$$
$$max\left(\max_{a\in(A^1-R)} \left|POS_{R\cup\{a\}}(\{d\})\right|, \max_{a\in(A^2-R)} \left|POS_{R\cup\{a\}}(\{d\})\right|, ... \max_{a\in(A^p-R)} \left|POS_{R\cup\{a\}}(\{d\})\right|\right) \tag{3.6}$$

Thus, the attribute selection process is equivalent in both sequential and vertical partitioning based distributed approaches of IQRA_IG algorithm.

3.4 Parallel attribute reduction in CDS using vertical partitioning

In this section, the main (driver) algorithm for proposed approach MR_IQRA_VP is given in Algorithm 3.1. The mapper phase algorithm (Algorithm 3.2: MR_IQRA_VP: map()) for the local best attribute selection is given in Section 3.4.1. The reducer phase algorithm (Algorithm 3.3: MR_IQRA_VP: reduce()) for the global best attribute selection is described in Section 3.4.2. And, the computation of $\gamma_A(\{d\})$ for the end condition of the main algorithm (Algorithm 3.1) is given in Section 3.4.3.

In the driver (Algorithm 3.1), initially the data set CDS is vertically partitioned into CDS^i $(i = 1, 2, ...p)$, and decision attribute information is broadcasted to all the nodes of the cluster. Reduct Red is initialized to empty set (ϕ), and U/Red contains only $\{U\}$ which is also equal to $NP\_GR(U/Red)$. As described in IQRA_IG algorithm, for effective positive region removal, the attribute selection computations are conducted only on $NP\_GR(U/Red)$. Hence, $NP\_GR(U/Red)$ is broadcasted to all the cluster nodes. The variable $total\_PosCount$ (initialized to zero) represents the $|POS_{Red}(\{d\})|$.

The next best attribute $best\_Attr$, and its positive region count $bestAttr\_PosCount$ are obtained through invocation of Algorithm 3.2 followed by Algorithm 3.3. The next best attribute $best\_Attr$ is included into reduct set Red. The variable $bestAttr\_PosCount$ represents the number of objects in non-positive region being added into positive region resulting from the granular refinement of $NP\_GR(U/Red)$ with $best\_Attr$. Consequently, the total positive region count is updated as $total\_PosCount = total\_PosCount + bestAttr\_PosCount$ and the $\gamma_{Red}(\{d\})$ is updated accordingly. If the required end condition $(\gamma_{Red} == \gamma_A)$ is reached, then algorithm returns reduct Red as the result of the algorithm. Otherwise, one needs to obtain $NP\_GR(U/Red)$ for the next iteration. Towards this objective, the record pertaining

to *best_Attr* is fetched to the driver using a map only job. The granular space of U/Red is refined with *best_Attr* information, and $NP\_GR(U/Red)$ is computed by removal of positive region granules from U/Red. The resulting $NP\_GR(U/Red)$ is broadcasted to the nodes of the cluster.

Algorithm 3.1: MR_IQRA_VP: driver()

Input: Input file: data set $CDS = (U, A \cup \{d\})$
Output: Reduct Red

1 Distribute input data set CDS with vertical partitioning over A, and broadcast decision attribute $\{d\}$ into the nodes of the cluster so that each data partition becomes $CDS^i = (U, A^i \cup \{d\})$, $\forall i \in \{1, 2, ...p\}$ where p is the number of data partitions in the cluster.
2 Broadcast initial reduct $Red = \phi$, and initial non-positive region granules list: $NP\_GR(U/Red) = \{U\}$ to all the nodes of the cluster.
3 Compute $\gamma_A(\{d\})$
4 $total\_PosCount = 0$
5 **repeat**
  `/* =========Phase 1: Finding the best attribute========== */`
6 Initiate MapReduce job such that each mapper computes local best attribute ($localBest\_Attr$) and its positive count ($localBest\_PosCount$) by using Algorithm 3.2 and reducer computes global best attribute ($best\_Attr$) and its positive count ($bestAttr\_PosCount$) by using Algorithm 3.3.
7 Collect the data $< key, value >=< best\_Attr, bestAttr\_PosCount >$ from the reducer.
8 $Red = Red \cup best\_Attr$
9 $total\_PosCount = total\_PosCount + bestAttr\_PosCount$
10 Compute $\gamma_{Red} = \frac{total\_PosCount}{|U|}$
11 **if** $(\gamma_{Red} < \gamma_A)$ **then**
    `/* ========Phase 2: Updating` $NP\_GR(U/Red)$ `============ */`
12 Fetch the record of best attribute ($best\_Attr$) from the cluster `// using map only operation`
13 Compute $U/Red = GranularRefinement(NP\_GR(U/(Red - best\_Attr)), best\_Attr)$ `// Applying granular refinement`
14 $NP\_GR(U/Red) = U/Red - P\_GR(U/Red)$ `// Incorporating positive region removal`
15 Broadcast $NP\_GR(U/Red)$
16 **end**
17 **until** $(\gamma_{Red} == \gamma_A)$
18 Return Red

Algorithm 3.2: MR_IQRA_VP: map()

Input: 1. Data split $CDS^i = (U, A^i \cup \{d\})$ with each record as
$< key, value >=< attrNo, attr\_Data >$
2. Broadcasted reduct: Red, non-positive region granules: $NP\_GR(U/Red)$

Output: $< key', value' >=< dummyKey, (localBest\_Attr, localBest\_PosCount) >$
where $dummyKey$ is common key for all the values of $value'$,
$localBest\_Attr \in A^i$ is local best attribute in the partition, and
$localBest\_PosCount$ is its positive count

```
1  maxPos_Count = 0, localBest_Attr = −1
2  for each record rec ∈ CDS^i as < attrNo, attr_Data > do
3      if attrNo ∉ Red then
4          U/(Red ∪ {attrNo}) = GranularRefinement(NP_GR(U/Red), attrNo)
5          Compute POS_{Red∪{attrNo}}({d}) using U/(Red ∪ {attrNo})
6          pos_Count = |POS_{Red∪{attrNo}}({d})|
7          if pos_Count > maxPos_Count then
8              localBest_Attr = attrNo
9              maxPos_Count = pos_Count
10         end
11     end
12 end
13 Construct < key', value' >, where key' = dummyKey, and
     value' = (localBest_Attr, localBest_PosCount)
14 Emit intermediate < key', value' >
```

3.4.1 MR_IQRA_VP: map() algorithm

The algorithm MR_IQRA_VP: map() given in Algorithm 3.2 is invoked in each iteration of the main algorithm of MR_IQRA_VP: driver() for selection of next best attribute into reduct Red. The mapper process associated with a data split CDS^i receives the current reduct Red, and the associated non-positive region granules $NP\_GR(U/Red)$ through broadcasting from the driver. For each attribute, $attrNo \in (A^i - Red)$, the granules $U/(Red \cup \{attrNo\})$ are computed using $GranularRefinement(NP\_GR(U/Red), attrNo)$. The pos_Count is evaluated by summing the cardinalities of positive region granules of $U/(Red \cup \{attrNo\})$. The local best attribute $localBest\_Attr$ is selected from $(A^i - Red)$ based on obtaining maximum pos_Count (as $localBest\_PosCount$). The information of local best attribute is communicated to MR_IQRA_VP: reduce() job in a single $< key, value >$ pair, where $key = dummyKey$ and $value = (localBest\_Attr, localBest\_PosCount)$. In this manner, p number of $< key, value >$ pairs are generated from decision sub tables CDS^i $(i = 1, 2, ...p)$ and participate in shuffle and sort phase. As all of $< key, value >$ pairs contain the same $dummyKey$ portion as the key, only a single reducer will be invoked facilitating global best attribute selection.

3.4.2 MR_IQRA_VP: reduce() algorithm

Algorithm 3.3: MR_IQRA_VP: reduce()

Input: $< key, V >$ pair where key is a "$dummyKey$" (common key from all the mappers), V is a list of values, where each value is $(localBest\_Attr, localBest\_PosCount)$ generated from each mapper

Output: $< key', value' >=< best\_Attr, bestAttr\_PosCount) >$ where $best\_Attr$ is the best attribute and $bestAttr\_PosCount$ is it's positive count

```
1 bestAttr_PosCount = 0, best_Attr = −1
2 for each value v ∈ V as (localBest_Attr, localBest_PosCount) do
3 |   if localBest_PosCount > bestAttr_PosCount then
4 |   |   best_Attr = localBest_Attr
5 |   |   bestAttr_PosCount = localBest_PosCount
6 |   end
7 end
8 Construct < key', value' >, where key' = best_Attr, and value' = bestAttr_PosCount
9 Emit < key', value' >
```

The algorithm MR_IQRA_VP: reduce() given in Algorithm 3.3 receives $< key, V >$ as the input resulting from shuffle and sort phase over outputs of MR_IQRA_VP: map() jobs. Here, key is the $dummyKey$, and V is the list of associated values from all mappers. Each value in V is in the form $(localBest\_Attr, localBest\_PosCount)$ containing the local best result of each mapper. The global next best attribute is selected from the local best attributes having the maximum positive count. The selected best attribute information is communicated to the driver in the form of $< key', value' >$ pair, where $key' = best\_Attr$ and $value' = bestAttr\_PosCount$.

3.4.3 Computation of $\gamma_A(\{d\})$

The computation of $\gamma_A(\{d\})$ requires construction of U/A and categorizing the granules into $P\_GR(U/A)$, and $NP\_GR(U/A)$. In each mapper, using the decision sub table $CDS^i = (U, A^i \cup \{d\})$, $\forall i \in \{1, 2, ...p\}$, one can compute U/A^i, and the granules can be categorized into $P\_GR(U/A^i)$ and $NP\_GR(U/A^i)$. From the explanation of positive region removal in Section 3.2.1.2, objects of $P\_GR(U/A^i)$ are the objects in $POS_A(\{d\})$. Here, each mapper communicates the information of $NP\_GR(U/A^i)$ to a single reducer. The reducer then computes refinement of $NP\_GR(U/A^i)$, $\forall i \in \{1, 2, ...p\}$ and arrives at $NP\_GR(U/A)$. If $NP\_GR(U/A)$ is empty then $\gamma_A(\{d\}) = 1$ otherwise $POS_A(\{d\})$ is computed as, $POS_A(\{d\}) = U - \bigcup_{gr \in NP\_GR(U/A)} gr$. Based on this positive region $POS_A(\{d\})$, the value of $\gamma_A(\{d\})$ is computed, and communicated to the driver.

3.4.4 Complexity analysis of MR_IQRA_VP algorithm

Given that the MapReduce programming model consists of three phases: map, shuffle and sort, and reduce, the time complexities of the proposed MapReduce based algorithms in this thesis are determined using these phases along with the driver's complexity. In its theoretical complexity analysis, the MapReduce model also considers communication costs and barrier synchronization costs. In this thesis, for the complexity analysis, we assume that the preprocessed data set (transposed data set) is given as the input to the proposed vertical partitioning based algorithms and the original data set to the proposed horizontal partitioning based algorithms. Furthermore, we also assume that the number of data partitions in the MapReduce cluster is the same as the number of processors (cores).

In the time complexity analysis of MR_IQRA_VP, the following variables are used.

- $|U|$: the number of objects in the data set
- $|A|$: the number of conditional attributes in the data set
- p: the number of processors
- t_w: the number of time units to transfer one word of memory
- s: the number of time units to complete the synchronization

Table 3.1: Time complexity analysis of MR_IQRA_VP algorithm

Algorithm (phase)	**Step* in Algorithm**	**Time complexity**
Driver → (Algorithm 3.1)	1. Partitioning the data vertically	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert}{p} * t_w)$
	2. Broadcasting NP_GR and $\{d\}$	$\mathcal{O}(\lvert U\rvert * t_w)$
	3. $\gamma_A(\{d\})$ computation	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert log\lvert U\rvert}{p})+\mathcal{O}(p*\lvert U\rvert*t_w)$
	12. Fetching a^{best} record	$\mathcal{O}(\lvert U\rvert * t_w)$
	13. Finding granules based on a^{best} record	$\mathcal{O}(\lvert U\rvert log\lvert U\rvert)$
Mapper → (Algorithm 3.2)	4-6. Creating granules and finding positive region counts	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert log\lvert U\rvert}{p})$
	7-10. Finding la^{best} and Barrier synchronization	$\mathcal{O}(\frac{\lvert A\rvert}{p}) + \mathcal{O}(s)$
Shuffle and sort →	Transferring all la^{best} and their positive region counts	$\mathcal{O}(p * t_w)$
Reducer → (Algorithm 3.3)	2-7. Finding a^{best} and Barrier synchronization	$\mathcal{O}(p) + \mathcal{O}(s)$

* Step denotes the line number in the associated algorithm

Table 3.1 shows the time complexities of each step of the phase in the MR_IQRA_VP algorithm for one iteration. In the table, from step 12 to step 13 in Algorithm 3.1 (driver) and all the steps in Algorithm 3.2 (mapper), shuffle and sort phase and Algorithm 3.3 (reducer) are repeated until ($\gamma_{Red} == \gamma_A$) condition is satisfied. That is, these steps are repeated $|A|$ times (in worst case). Hence, by adding up all the complexities, the total time complexity of the algorithm MR_IQRA_VP is obtained as given below.

$$(\frac{|A| * |U|}{p} * t_w) + (|U| * t_w) + (\frac{|A| * |U|log|U|}{p}) + (p * |U| * t_w) + \\ |A| * \Bigg((|U| * t_w) + (|U|log|U|) + (\frac{|A| * |U|log|U|}{p}) + \frac{|A|}{p} + s + (p * t_w) + p + s \Bigg) \quad (3.7)$$

Above equation can be approximated as, $\mathcal{O}(\frac{|A|^2 * |U|log|U|}{p}) + \mathcal{O}(|A| * ((p * |U| * t_w) + s))$. Thus the time complexity of the MR_IQRA_VP algorithm is $\mathcal{O}(\frac{|A|^2 * |U|log|U|}{p})$ in addition with its communication cost: $\mathcal{O}(|A| * ((p * |U| * t_w) + s))$.

The entire decision system is required to be present in memory for reduct computation using MR_IQRA_VP algorithm. Thus, the space complexity of sequential MR_IQRA_VP algorithm is $\mathcal{O}(|A| * |U|)$. But, in MapReduce framework environment, the input decision system is partitioned and distributed to the nodes of the cluster where the workload is divided equally into p data partitions. Hence, each partition has the complexity of $\mathcal{O}(\frac{|A| * |U|}{p})$.

In the worst-case scenario, the aforementioned theoretical time and space complexities of the proposed MR_IQRA_VP algorithm are described. However, since the MR_IQRA_VP algorithm incorporates positive region removal and granular refinement features, the actual time and space complexities are significantly reduced.

3.5 Salient features and limitations of MR_IQRA_VP

The removal of positive region, granular refinement, and simplification of shuffle and sort phase are the main features of the proposed MR_IQRA_VP algorithm. In this section, the main features, and the limitations of proposed MR_IQRA_VP algorithm are discussed.

3.5.1 Positive region removal

In IQRA_IG algorithm, the removal of positive region is done physically, i.e., the rows corresponding to positive region objects are removed from memory. Our experimental simulations have established that the removal of positive region data from distributed data set incurs significant computational overhead. Even in horizontal partitioning based IN_MRIQRA_IG algorithm, positive region data is not physically removed owing to the same reasons. This

led us to incorporate positive region removal based on the methodology described in Section 3.2.1.2. Instead of physically removing the positive region data, the computations in mapper phase were restricted to objects in $NP\_GR(U/R)$. As information of each attribute is stored in random accessible memory unit such as array, we found that performing computation based on objects present in $NP\_GR(U/R)$ resulted in exactly the same amount of computational time savings as that of post physical removal of positive region.

3.5.2 Granular refinement

The implementation of $GranularRefinement(NP\_GR(U/R), a)$ of MR_IQRA_VP algorithm is identical to that of sequential IQRA_IG algorithm because the proposed algorithm uses vertical partitioning strategy. In contrast to IQRA_IG, the required information of $NP\_GR(U/R)$ is broadcasted from driver to worker nodes. In the implementation of IQRA_IG, the Quick sort is used in splitting $gr \in NP\_GR(U/R)$ using attribute values of a. In the implementation of MR_IQRA_VP, $HashMap$ [5] is used for the same. $HashMap$ is a data structure that maintains records of paired data $< key, value >$. It manages the retrieval and updation of a record associated with a key through hashing. Here, each object in a granule of $NP\_GR(U/R)$ is visited in sequence. Using the $HashMap$, objects associated with unique values based on attribute a are obtained through updations of $HashMap$ with $< key, value >$ being unique attribute value. After processing all the objects in gr, $HashMap$ contains $|gr/\{a\}|$ number of entries. The key corresponds to unique attribute values of objects in gr based on a. In a $< key, value >$ pair, value corresponds to list of *object ids* having the same attribute value of key based on a. Hence, the required refined granules of $U/(R \cup \{a\})$ resulting from splitting of gr are extracted from value portions of $HashMap$ entries. Therefore it can be observed that $HashMap$ based granular refinement aids in improving the computational performance.

It is to be noted that, in the i^{th} mapper, the granules of $U/(R \cup \{a\}),\ \forall a \in (A - R)$ are utilized for the computation of $|POS_{R\cup\{a\}}(\{d\})|,\ \forall a \in (A - R)$. In order to optimize the memory utilization, the memory occupied by $U/(R \cup \{a\}),\ \forall a \in (A - R)$ is released after obtaining the required positive region counts. Even though the driver requires attribute information of the next best attribute for granular refinement of U/R, in our algorithm we did not communicate the local best attribute record to the reducer. This decision was motivated by the objective of simplifying the most complex operation of MapReduce job, i.e.,shuffle and sort phase. Hence, an additional map only job for extracting the best attribute information was initiated, so that the required best attribute record is directly transferred from corresponding worker (slave) node to the driver.

3.5.3 Simplification of shuffle and sort phase

Without loss of generality, consider a decision system having k distinct values for each attribute. From Section 3.2.2, in any horizontal data partitioning based reduct computation algorithm, it is observed that, in each iteration in a mapper, the size of key space is $k^{|R\cup\{a\}|}$, $\forall a \in (A-R)$. As p denotes the number of data partitions, then a total $p * |A-R| * k^{(|R|+1)}$ size of $< key, value >$ pairs are transferred in the network of the cluster of computers. As a result, shuffle and sort phase work with data of this order leading to huge bottleneck for the algorithm.

In the proposed design, as given in the Section 3.4.1 of mapper phase, each mapper produces a single $< key, value >$ pair corresponding to the local best attribute, and the local best attribute's positive count. This results in a total p size of $< key, value >$ pairs being transferred in the network of the cluster, which leads to a considerable reduction in the work of shuffle and sort phase. This is because only a small size of data is transferred and communicated when compared to the horizontal partitioning based algorithm. The simplification of shuffle and sort phase is an essential facet of vertical partitioning based MapReduce reduct computation algorithm.

3.5.4 Limitations of MR_IQRA_VP

In the proposed MR_IQRA_VP algorithm, the broadcasting decision attribute information (of size $|U|$) and granules of $NP\_GR(U/R)$ in every iteration (of size $\leq |U|$) is needed. And also fetching the next best attribute information from the worker node to the driver (of size $|U|$) in every iteration is needed. For very large object spaces, these operations become complex and computationally expensive.

The time complexity of IQRA_IG algorithm is $\mathcal{O}(|A|^2|U|log|U|)$, and the complexity of an iteration is $\mathcal{O}(|A-R||U|log|U|)$. In the vertical partitioning strategy, the time complexity of an iteration in mapper is $\mathcal{O}(\frac{|A-R|}{p}|U|log|U|)$, where p is the number of data partitions. Therefore, for very large object space data sets the gain obtained through attribute space division can be compensated by increased computations with respect to object space. In view of the above reasons, MR_IQRA_VP algorithm is suitable for moderate object space data sets while being scalable to very large attribute space data sets as the approach is horizontally scalable in attribute space, i.e., very large-scale big dimensional data sets can be handled by addition of new nodes to the cluster. Theoretical explanation of this section is validated experimentally in Section 3.6.4.

3.6 Experimental analysis

The approaches proposed in this thesis are parallel/distributed methods, therefore the empirical evaluation of these methods are given based on the following criteria.

- *Computational evaluation.*
- *Performance evaluation.*
- *Impact of the partitioning strategy.*

The *running time, reduct* and *reduct size* are used in the computational evaluation. And the performance is measured using three metrics: *speedup, scaleup,* and *sizeup* [105]. Since horizontal and vertical partitioning strategies are investigated in the proposed approaches, the influence of these strategies is examined.

As stated earlier, the proposed approaches are parallel/distributed versions of the existing sequential methods. It should be noted that the process of converting sequential algorithms to parallel/distribute algorithms has only improved scalability, and all of the parallel/distribute algorithms designed in this thesis produce the same reduct as the corresponding sequential versions. All the sequential algorithms considered in the thesis are well established in inducing good classification models. As a result, the experimental evaluation in this thesis focuses on assessing computational improvement as well as the performance evaluation of parallel/distributed algorithms.

The computational evaluation of proposed MR_IQRA_VP algorithm is presented in Section 3.6.2. The proposed algorithm's performance evaluation is provided in Section 3.6.3. Section 3.6.4 presents impact of the partitioning strategy that shows the relevance and limitations of the proposed algorithm.

3.6.1 Experimental set up

We carried out the experiments in two stages to evaluate the efficiency of the proposed MR_IQRA_VP algorithm. In the first stage of experiments, we compared MR_IQRA_VP algorithm with the existing MapReduce based parallel/distributed reduct computation algorithms, PLAR [124] and PFSPA [45]. Since the source code of these algorithms is not available, we compared these algorithms with the proposed algorithm based on the data sets and experimental set up given in respective publication sources of PLAR and PFSPA algorithms. Table 3.2 gives the details of the experimental set up of PLAR, PFSPA and the proposed MR_IQRA_VP algorithms. The comparison of the proposed algorithm with the existing PLAR and PFSPA algorithms is given in Section 3.6.2.1 and 3.6.2.2 respectively.

Table 3.2: Experimental set up of MR_IQRA_VP, PLAR, PFSPA and IN_MRIQRA_IG algorithms

	MR_IQRA_VP	$PLAR^{*}$	$PFSPA^{\#}$	IN_MRIQRA_IG
Cluster Size	6 Nodes	19 Nodes	6 Nodes	6 Nodes
RAM Size	8 GB	At least 8 GB	4 GB	8 GB
Cores	4	At least 8	4	4
Operating System	Ubuntu 18.04	Cent OS 6.5	-	Ubuntu 18.04
Framework	Spark 2.3.1	Spark 1.x	Hadoop	Spark 2.3.1

* Experimental set up as reported in [124], and # experimental set up as reported in [45]

In the second stage of experiments, the proposed MR_QRA_VP algorithm is compared with the existing IN_MRIQRA_IG algorithm. Sai Prasad et al. [92] developed a parallel version of IQRA_IG (IN_MRIQRA_IG) algorithm using in-memory iterative MapReduce framework of Twister. But Apache Spark has better fault tolerance than Twister. For this reason, algorithm IN_MRIQRA_IG is re-implemented on Apache Spark framework. The proposed MR_IQRA_VP algorithm has been compared with Apache Spark version of IN_MRIQRA_IG algorithm and the results are reported in Section 3.6.2.3.

In Section 3.4, we have described that the proposed algorithm is suitable for the data sets having a moderate size of objects with large number of attributes, and the same would be demonstrated empirically along with theoretical validation. Accordingly, for comparative analysis, data sets have been considered to meet this criterion. A series of experiments have been conducted on the benchmark data sets such as Gisette, Gene expression Cancer RNA-Seq (renamed as Genes), Basehock, KDDcup and Semeion Handwritten Digit (renamed as Handwritten) data sets. The Basehock data set is available in Arizona State University feature selection data set repository [109], and the rest of the data sets are available in UCI data set repository [31]. Details of these data sets are provided along with their object and attribute space sizes in Table 3.3. All the data sets with different sizes are chosen according to limited hardware configuration of the cluster. In the selection of these data sets, we considered the aspect of variance in sizes of object space and attribute space, and few data sets such as Genes, Basehock and Gisette are replicated several times in attribute space to illustrate the efficiency of the proposed MR_IQRA_VP approach in attribute space. For example, the original "Genes" data set has 801 objects and 20561 attributes, and after replication, the data set "genes-S801-A5000k" contains 801 objects ("S" denotes Samples) and approximately 5 million attributes ("A" denotes Attributes).

Table 3.3: Data sets used in the experiments of MR_IQRA_VP algorithm

S.No	Data set	Objects	Attributes	Classes
1	Gisette	6000	5000	2
2	Basehock	1993	4864	2
3	Genes	801	20561	5
4	Handwritten	1593	256	10
5	KDDCup	4898431	41	23
6	genes-S801-A5000k	801	5009564	5
7	gisette-S50k-A50k	50000	50000	2
8	basehock-S2k-A53k	1993	53000	2

The replication of a data set is used throughout the thesis, both in object space and attribute space, with the goal of analysing the impact of increasing the sizes with respect to the partitioning strategy used. It should also be noted that replication in object and attribute space increases computational complexity proportionally, as analysed in complexity analysis (in Section 3.4.4).

3.6.2 Computational evaluation of MR_IQRA_VP

The efficiency of the proposed algorithm is shown by comparing the results with existing MapReduce based parallel/distributed reduct computation algorithms on different data sets. For reproducible research, obtained reducts and their respective γ_C and γ_R values of proposed MR_IQRA_VP algorithm for different original data sets (the data sets without replication) are given in Table 3.4.

Table 3.4: The obtained reduct of MR_IQRA_VP for different data sets

Data set	Reduct	γ_C	γ_R
Gisette	{3058, 1523, 4734, 1923, 4165, 4272, 1555, 3708, 4159, 3354, 4694, 1408, 3470, 3166, 4958}	1.0	1.0
Genes	{18493, 15864, 15442, 16327, 6917, 17651, 19940}	1.0	1.0
Basehock	{3281, 2471, 1193, 577, 3282, 2965, 4052, 2000, 3302, 3756, 1791, 369, 1035, 4315,1366, 2005, 356, 1722, 882, 250, 3300, 1275, 3292, 2631, 4345, 3825, 4544, 4355, 4776, 4751, 2219, 4682, 383, 203, 3892, 1188, 3922, 4148, 4757, 1947, 3254, 4706, 3475, 4351, 593}	1.0	1.0
Handwritten	{256, 113, 49, 31, 20, 72, 178, 111, 84, 109, 191, 29, 230, 120, 196, 153, 43, 233, 250, 89, 185, 174}	1.0	1.0
KDDcup	{1, 2, 22, 28, 37, 4, 30, 5, 32, 10, 35, 39, 36, 34, 31, 3, 12, 16, 8, 33, 19, 23, 6, 40, 17, 29, 38, 26, 24, 13, 11}	1.0	1.0

3.6.2.1 MR_IQRA_VP comparison with PLAR

The authors of Parallel Large scale Attribute Reduction (PLAR) [124], have developed different algorithms: PLAR_PR, PLAR_LCE, PLAR_SCE, and PLAR_CCE based on different heuristic functions. All the algorithms are iterative MapReduce based reduct computation algorithms, and they are implemented on Apache Spark framework. The results for Gisette data set obtained by PLAR_SCE are reported in [124]. The authors of PLAR_SCE focused on calculating computational time per iteration incurred while running the algorithm. They selected five attributes in five iterations, resulting in a sub reduct. They could not complete the remaining iterations to get the entire reduct because of high dimensionality of the data set. For this reason, MR_IQRA_VP is compared with PLAR_SCE in terms of iterations. The comparison results are shown in Table 3.5. The proposed algorithm computed the reduct with the length 15 (refer Table 3.4). Accordingly, the estimated computational time of PLAR_SCE for 15 iterations is calculated as 5187 seconds.

Table 3.5: Comparative results of MR_IQRA_VP with PLAR_SCE (Time: Seconds)

Iteration	$PLAR\_SCE^{*}$	MR_IQRA_VP
1	350	22.38
2	343	19.70
3	344	18.31
4	344	6.92
5	348	4.81
:	:	:
:	:	:
Time for all iterations	$5187^{\#}$	89.02

* results as reported in [124], and $^{\#}$ indicates the estimated time of PLAR_SCE algorithm

The algorithm MR_IQRA_VP completed first five iterations in 72.12 seconds against 1729 seconds incurred in PLAR_SCE algorithm. The MR_IQRA_VP algorithm achieved a significant computational gain of 95.82%. MR_IQRA_VP obtained the complete reduct in 89.02 seconds against estimated computational time of 5187 seconds in PLAR_SCE achieving 98.29% computational gain. Both algorithms used MapReduce framework of Apache Spark for the implementation. Even though MR_IQRA_VP algorithm is implemented in an inferior cluster configuration than PLAR_SCE (refer Table 3.2), we could get better results than PLAR_SCE algorithm. Therefore, this comparative analysis of MR_IQRA_VP with PLAR_SCE demonstrates the relevance of MR_IQRA_VP. And, it establishes the relevance of vertical partitioning strategy for Gisette kind of data sets with larger attribute space.

3.6.2.2 MR_IQRA_VP comparison with PFSPA

Qing He et al. [45] proposed a MapReduce based parallel algorithm of PFSPA (Parallel Feature Selection using Positive Approximation). Algorithm PFSPA is implemented on Hadoop MapReduce framework. MR_IQRA_VP is compared with PFSPA on Handwritten data set. From the experimental design in [45], data set is replicated several times, and experiments were conducted on the replicated data set. The comparative results are shown in Table 3.6.

Both PFSPA [45] and MR_IQRA_VP obtained similar length reducts with different sizes of Handwritten data set. MR_IQRA_VP achieved a significant computational gain over PFSPA in the order of 85% to 99%. As the number of objects was increased, the computational gain percentage was reduced indicating MR_IQRA_VP's computational complexity is proportional to the size of object space. These significant results are partly because of utilizing the iterative MapReduce framework of Apache Spark against Hadoop framework in PFSPA, and also partly due to the proposed methodology.

Table 3.6: Comparative results of MR_IQRA_VP with PFSPA (Time: Seconds)

Data set	Objects	Attributes	$PFSPA^*$		MR_IQRA_VP	
			Running time	Reduct size	Running time	Reduct size
Handwritten	1593	256	1086	22	10.04	22
Handwritten 3 times	4779	256	1205	22	15.34	22
Handwritten 6 times	9558	256	1282	22	23.84	21
Handwritten 12 times	19116	256	1596	22	67.58	21
Handwritten 24 times	38232	256	2062	22	307.88	22

* results as reported in [45]

3.6.2.3 MR_IQRA_VP comparison with IN_MRIQRA_IG

The experiments of the proposed parallel/distributed approach MR_IQRA_VP, and existing approach IN_MRIQRA_IG are carried out on a 7-node cluster. In the cluster, one node is set as master (driver) as well as slave, and the rest are set as workers (slaves). The master node uses Intel (R) Xeon (R) Silver 4110 CPU @ 2.10GHz processor with 32 cores and 64 GB of main memory. All the worker nodes use Intel (R) Core (TM) i7-8700 CPU@3.20GHz processor with 12 cores and 32 GB of main memory. All the nodes run on Ubuntu 18.04 LTS operating system and they are connected via Ethernet (with 1000 Mbps speed). Each node is installed with Java 1.8.0_171, Apache Spark 2.3.1, and Scala 2.11.4.

Table 3.7: Comparative results of MR_IQRA_VP with IN_MRIQRA_IG (Time: Seconds)

Data set	Attributes	IN_MRIQRA_IG		MR_IQRA_VP	
		Running time	Reduct size	Running time	Reduct size
Handwritten	256	16.63	22	10.04	22
Basehock	4864	76.68	45	36.23	45
Genes	20561	36.06	07	12.04	07
Gisette	5000	123.63	16	89.02	15
KDDcup	41	4785.60	31	17928.45	31
genes-S801-A5000k	5009564	12065.28	07	264.36	07
gisette-S50k-A50k	50000	1605.77	16	1914.42	16
basehock-S2k-A53k	53000	237.36	45	115.36	45

The proposed MR_IQRA_VP algorithm has been compared with Apache Spark version of IN_MRIQRA_IG algorithm on all the data sets (original data sets and replicated data sets). The comparison results are shown in Table 3.7.

MR_IQRA_VP achieved 52.75% computational gain in Basehock data set, 66.61% in Genes data set, 39.62% in Handwritten data set, and 27.99% in Gisette data set over IN_MRIQRA_IG algorithm. The gain percentage is inversely proportional to the size of the object space. The best computational gains are obtained for Genes data set having smaller object space with huge attribute space. In contrast, IN_MRIQRA_IG algorithm obtained better computational gains on KDDcup data set which has huge object space and very less attribute space.

3.6.3 Performance evaluation of MR_IQRA_VP

The performance of MR_IQRA_VP has been evaluated based on Speedup, Scaleup and Sizeup metrics. The proposed algorithm's performance is compared with the existing IN_MRIQRA_IG algorithm. The performance evaluation experiments of both the algorithms are performed on the experimental set up given in Table 3.2.

3.6.3.1 Speedup

Speedup can be measured by increasing the number of machines in the parallel system while keeping the data set constant. That is, it refers to the ratio of a job's running time on the parallel system compared with a single system: $Speedup(n) = \frac{Running\ time\ on\ one\ computer}{Running\ time\ on\ n\ computers}$. Theoretically, a perfect parallel algorithm can demonstrate a linear speedup, a system with n times the number of computers gets a speedup of n. Due to the serial computing, communication costs and other overheads of the parallel system, it is difficult to achieve linear speedup. MR_IQRA_VP's speedup has been evaluated on the data sets with different varied

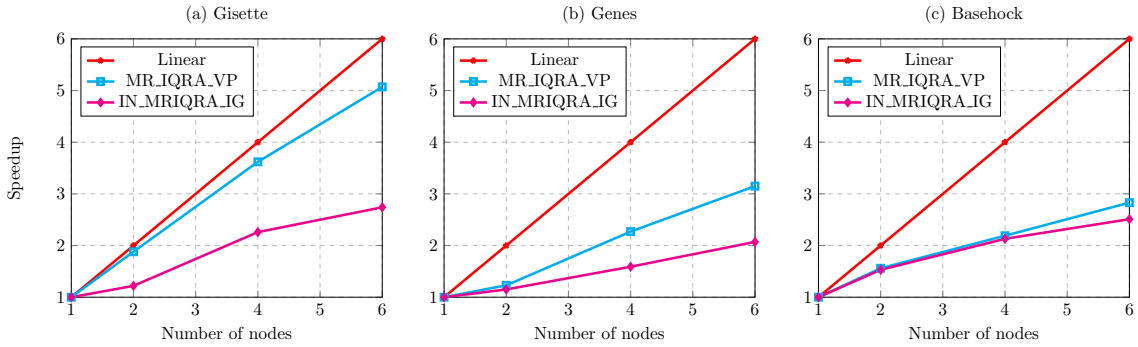


Figure 3.3: Speedup of MR_IQRA_VP and IN_MRIQRA_IG for different data sets

nodes from 1 to 6. Figure 3.3 shows the speedup performance results of MR_IQRA_VP and IN_MRIQRA_IG algorithms for different data sets.

As shown in Figure 3.3, MR_IQRA_VP has achieved better speedup performance (nearing expected linear speedup for Gisette data set) over IN_MRIQRA_IG, this is mainly due to simplified shuffle and sort phase in MR_IQRA_VP as described in Section 3.5.3. Algorithm IN_MRIQRA_IG and other horizontal partitioning based algorithms are effected by elaborate shuffle and sort phase which hampered the speedup performance. It can be observed that the speedup achieved for the proposed MR_IQRA_VP algorithm is lower in the basehock data set than in the other data sets. This is due to the higher reduct size (i.e., 45), which results in increased sequential computation cost of fetching the best attribute (a^{best}) record into the driver and processing, as well as the cost of broadcasting.

3.6.3.2 Scaleup

Scaleup is defined as the ability of an n-times larger cluster to perform n-times larger data set in the same run time as the original system, i.e., $Scaleup(n) = \frac{Time\ of\ data\ on\ one\ computer}{Time\ of\ n{-}times\ data\ on\ n\ computers}$. To find the scaleup of the proposed algorithm, we increased the size of the data set in proportion to the number of computers in the cluster. Here each data set size started from 20%, 40%, 60%, 80%, and 100% of attributes in the data set (that is, data set size is divided in the attribute space), and the number of nodes are increased from 1 node, 2 nodes, 3 nodes, 4 nodes and 5 nodes respectively. Figure 3.4 shows the scaleup performance results of MR_IQRA_VP and IN_MRIQRA_IG algorithms for different data sets.

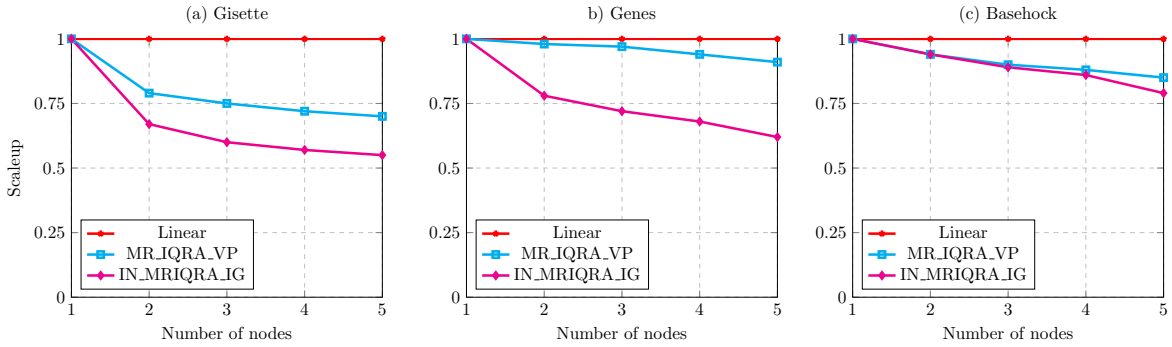


Figure 3.4: Scaleup of MR_IQRA_VP and IN_MRIQRA_IG for different data sets

The scaleup analysis results shown in Figure 3.4 demonstrates that, increasing the attribute space results in better scaleup for MR_IQRA_VP than IN_MRIQRA_IG. As we are keeping object space constant while increasing the attribute space in every data set, the complexity involved in the mapper phase of the proposed algorithm is constant on every data set. Thus, the proposed algorithm is producing better scaleup for big dimensional data sets such as "Genes". In contrast, in IN_MRIQRA_IG algorithm, the complexity of shuffle and sort phase increases, if attribute space increases in the data sets. Thus, in comparison with IN_MRIQRA_IG algorithm, our proposed algorithm exhibited better scaleup for the data sets that have moderate object space and larger attribute space. The scaleup values of more than or equal to 0.7 indicate that the proposed MR_IQRA_VP scale well in attribute space of the data set. It is difficult to achieve ideal scaleup values (i.e., 1) due to barriers like reducers, sequential computations in the driver and broadcast operations.

3.6.3.3 Sizeup

Sizeup measures the time it takes on a given system when the data set is n-times larger than the original data set. It calculates the increase in computational time based on the size of data sets. The sizeup is specified as: $Sizeup(n) = \frac{Time\ for\ processing\ n-times\ data}{Time\ for\ processing\ data}$. To find the sizeup performance of the proposed algorithm, we kept the number of nodes as constant, and changed the size of the data set. The number of computers were kept as six nodes. Each data set size was increased with 20%, 40%, 60%, 80%, and 100% of attributes in the data set.

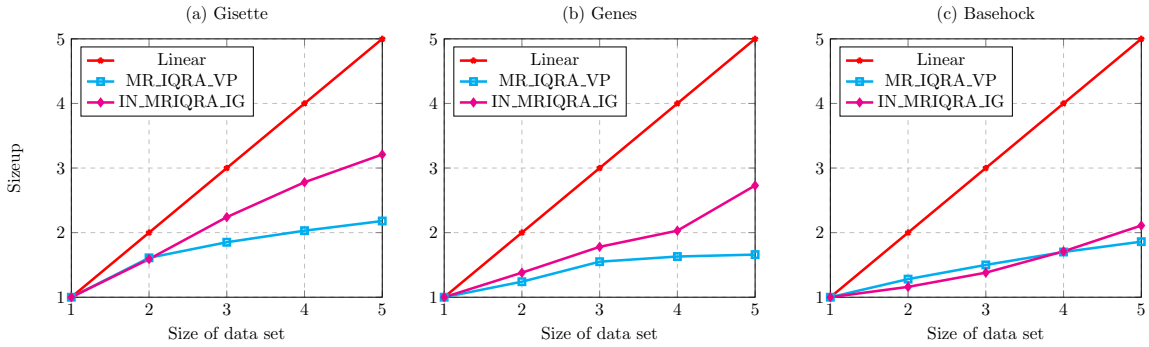


Figure 3.5: Sizeup of MR_IQRA_VP and IN_MRIQRA_IG for different data sets

Figure 3.5 shows the sizeup performance results of MR_IQRA_VP and IN_MRIQRA_IG algorithms for different data sets with varying sizes of attribute space. From the figure, we can observe that, both the existing and proposed algorithms produce better sizeup results, as their plots are much lower than the linear plots in the figures for all the data sets.

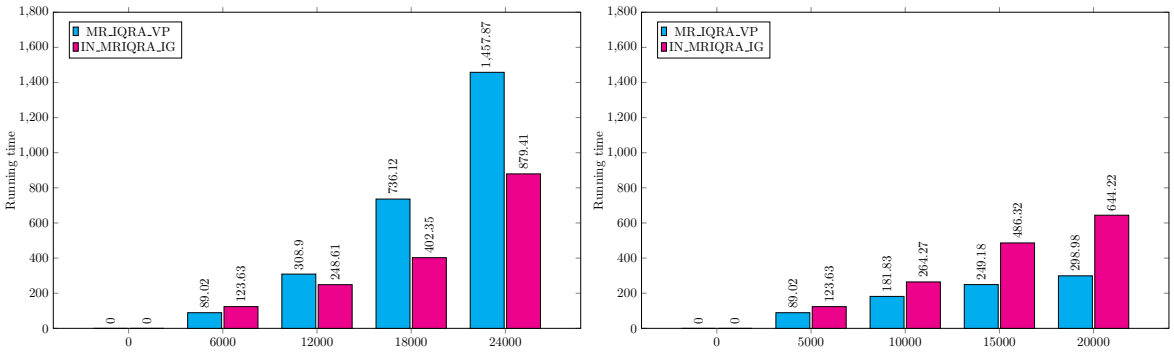
3.6.4 Impact of the data partitioning strategy

The experimental results have suggested that MR_IQRA_VP algorithm is suitable for data sets with moderate object space and larger attribute space. Considering IN_MRIQRA_IG as the representative horizontal partitioning based reduct computation algorithm, we conducted an experiment between IN_MRIQRA_IG and MR_IQRA_VP algorithms. The objective of the experiment is to determine the nature of data sets relevant for horizontal partitioning and vertical partitioning based reduct computation algorithms.

In this experiment, Gisette data set (containing almost equal size object space and attribute space) was replicated in object space. The results of both algorithms are reported in Table 3.8, under the serial number 2, 3 and 4. Similarly, Gisette was replicated in attribute space and the results are reported in Table 3.8, under the serial number 5, 6 and 7. Figure 3.6 demonstrates the computational time analysis for scalability in object space in Figure 3.6a and the attribute space in Figure 3.6b.

Table 3.8: Comparison of MR_IQRA_VP, IN_MRIQRA_IG for varying objects and attributes of Gisette data set (Time: Seconds)

S.No	Objects	Attributes	IN_MRIQRA_IG Running Time	MR_IQRA_VP Running Time
1	6000	5000	123.63	89.02
2	12000	5000	248.61	308.90
3	18000	5000	402.35	736.12
4	24000	5000	879.41	1457.87
5	6000	10000	264.27	181.83
6	6000	15000	486.32	249.18
7	6000	20000	644.22	298.98



(a) Scalability in the object space

(b) Scalability in the attribute space

Figure 3.6: Behavior of MR_IQRA_VP and IN_MRIQRA_IG for varying object space and attribute space of Mushroom

It is evidently clear from these results that, increase in object space resulted in a considerable increase in computational time of MR_IQRA_VP. And, similarly increase in attribute space resulted in a more significant increase in computational time of IN_MRIQRA_IG. In Table 3.8, the details of original Gisette data set are reported in serial number 1. The analysis of results shows that MR_IQRA_VP is a highly scalable algorithm for scalability in attribute space. But it is not recommended for data sets of larger object space. The horizontal partitioning based IN_MRIQRA_IG algorithm is found more suitable for scalability in object space. Hence, the vertical partitioning based algorithms are suitable for big dimensional data sets with moderate object space frequently found in the areas of Bioinformatics and Web mining.

3.7 Summary

In this chapter, we have proposed and implemented a highly scalable MapReduce based reduct computation algorithm MR_IQRA_VP using vertical partitioning strategy for categorical decision systems. With this strategy we have managed a massive reduction in data

transformation and communication in the shuffle and sort phase of the MapReduce framework of Apache Spark, which is a primary bottleneck of the horizontal partitioning MapReduce based reduct algorithms. Extensive experimental results showed that MR_IQRA_VP is a more suitable and scalable algorithm for the data sets having moderate size object space and larger size attribute space such as microarray data sets in Bioinformatics.

Vertical partitioning strategy can also be used in designing MapReduce based reduct computation approaches for large-scale incomplete decision systems (IDS). In the next chapter (Chapter 4), the usefulness of horizontal and vertical partitioning strategies are discussed in designing MapReduce based parallel/distributed approaches for attribute reduction in IDS using extensions to classical rough sets.

Chapter 4

Parallel attribute reduction in Incomplete Decision Systems

The second contribution to this dissertation is provided in this chapter. MapReduce based parallel/distributed approaches are proposed for attribute reduction in incomplete decision systems (IDS) based on the existing Novel Granular Framework (NGF). The NGF is used to deal with incompleteness in the data. One of the proposed approaches adopts an alternative representation of the NGF and uses a horizontal partitioning of the data to the nodes of the cluster to handle the incomplete data sets that are large-scale in terms of number of objects. Another approach embraces the existing NGF and uses a vertical partitioning strategy to handle the big dimensional incomplete data sets. It is worth to mention that, to the best of our knowledge, the proposed approaches are the first research of its kind on parallel/distributed attribute reduction in IDS. The work presented in this chapter has been published in [100].

4.1 Review of existing approaches

The number of data sources is rising rapidly in this age of big data. The volume of the data collected at the end of each day has increased to huge levels, resulting in large-scale data sets. This large size of data presents more difficulties for data processing. Additional challenges arise if this large-scale data contains missing (incomplete) values. Based on the data source, missing data can occur due to several reasons such as human mistakes, sensor malfunctions, operator failures, malformed device, human reluctance to declare private information or because of their insufficient knowledge [133]. Decision systems that include objects with missing attribute values are referred to as the Incomplete Decision Systems (IDS). These decision systems are frequently occurring data sets in decision-making problems.

Classical rough set theory uses crisp equivalence classes (through indiscernibility relation, which is an equivalence relation) in attribute reduction. In IDS, it is not possible to form equivalence relation because of the missing values in the decision system. Thus, classical rough set model is not suitable for attribute reduction in IDS. Therefore, the attribute reduction can be performed by converting the IDS to complete Categorical Decision System (CDS) with the imputation of missing values. Grzymala et al. [37] described several approaches for imputing missing values. And some probabilistic methods [70, 71] also exist for the imputation of missing values. But, the preprocessing strategy used in imputation methodology determines the performance of the induced system using the obtained CDS.

Many researchers have used extensions to classical rough sets [57, 58, 99, 112] to deal with IDS for attribute reduction without performing the imputation of missing values. Attempts were made in the development of efficient attribute reduction algorithms [23, 30, 64, 129, 134] in IDS by using these extensions. And, few accelerators have also been proposed [73, 84, 108] to improve the computational efficiency. Sai Prasad et al. [73] have proposed a "*Novel Granular Framework (NGF)*" based on the findings of Kryszkiewicz et al. [57] to handle the complexity involved in the incompleteness of the data. And, the authors have adopted the NGF to develop the IQRAIG_Incomplete algorithm for attribute reduction in IDS. This algorithm is one of the efficient algorithms among the existing approaches, along with approach proposed by Qian et al. [84].

From the literature, it is observed that, a lot of research works have been done on attribute reduction in IDS. But all the existing approaches are sequential methods and they can not handle the large-scale incomplete data. Parallel/distributed approaches have not been proposed for attribute reduction in large-scale IDS. Hamed et al. [41] and Zhang et al. [126] have developed MapReduce based parallel methods to compute rough set approximations in IDS. But they did not propose any approaches for attribute reduction. This has motivated us to investigate MapReduce based parallel/distributed approaches to deal with massive incomplete data in attribute reduction.

According to review of relevant literature [81, 82, 92, 96, 102, 124] and Chapter 3, horizontal partitioning-based methods for attribute reduction in rough set theory are effective for data sets with a huge number of objects. And the vertical partitioning based approaches perform well for data sets with big dimensionality. Almost all the rough set theory based attribute reduction approaches using the MapReduce programming model [81, 82, 92, 96, 124] adopted horizontal partitioning strategy except for the approach in [102]. Thus, in this chapter, we explore the approaches that perform well for the big incomplete data sets with huge number of objects, and for the big dimensional incomplete data sets.

This chapter propose MapReduce based parallel/distributed approaches based on the NGF [73] for attribute reduction in IDS using horizontal and vertical partitioning strategies. Briefly, the contributions in this chapter include the following:

1. An alternative representation of the NGF is proposed and adopted to develop the MRIDS_HP algorithm. This algorithm uses the distributed strategy of horizontal partitioning.

2. Algorithm MRIDS_VP is developed by parallelizing the existing NGF based on the distributed strategy of the vertical partitioning.

Both algorithms are implemented and compared using the Apache Spark MapReduce framework [2]. The relevance and limitations of both algorithms are provided with extensive experimental analysis along with theoretical validation.

4.2 Rough sets extension to IDS

The proposed approaches use Kryszkiewicz's model [57] which is an extension of rough set theory for IDS. The basic notions of this model are discussed in this section.

A symbolic incomplete decision system is represented as, $IDS = (U, A \cup \{d\}, \{V_a, f_a\}_{a \in A \cup \{d\}})$. Here, $U = \{x_1, x_2, ..., x_n\}$ is a finite non-empty set of objects, $A = \{a_1, a_2, ..., a_m\}$ is a finite non-empty set of conditional attributes, and $\{d\}$ is a decision attribute that represent classes of objects. The notation V_a is the domain of attribute a, and $f_a : U \rightarrow V_a$ is a function that maps an object x in U to exactly one value in V_a. In this chapter, for simplicity, the notation $a(x)$ used for referring $f_a(x)$, and the decision system can be represented in short form as $IDS = (U, A \cup \{d\})$. The missing object values of attributes are denoted by a character "*." A missing value is treated as an unknown value. It should be noted that, without loss of generality, it is assumed that the decision attribute $\{d\}$ does not contain the missing values. That is, for all the objects, the decision class is known. If the missing object values are not present, then the decision systems are known as complete decision systems (CDS).

In classical rough set theory, *indiscernibility relation* [77, 78, 116] is the basic notion for attribute reduction in CDS. Corresponding to indiscernibility relation for CDS, the "*similarity relation*" [57] is the main idea for reduct computation in IDS. The similarity relation is used based on the assumption that the missing values are indiscernible with all other possible values in the attribute domain.

Definition 4.1. In the given incomplete decision system IDS, for the subset of attributes $B \subseteq A$, the similarity relation is defined as,

$$SIM(B) = \{(x, x') \in U^2 \mid \forall a \in B \ [a(x) = a(x') \vee a(x) = * \vee a(x') = *]\} \tag{4.1}$$

The similarity relation $SIM(B)$ satisfies the reflexive and symmetry properties, but it does not meet transitive property in all the cases. Hence it is a *tolerance relation*. The similarity relation $SIM(B)$ generates a cover of the universe of the objects U into distinct *similarity classes*. The set of similarity classes of U induced by similarity relation $SIM(B)$ is denoted as $U/SIM(B)$. For an object x, the similarity class is denoted by $S_B(x)$, and is given as,

$$S_B(x) = \{x' \in U \mid (x, x') \in SIM(B)\}$$

The set of similarity classes $U/SIM(B)$ are called as *approximation space* or *granular space*, and each similarity class in $U/SIM(B)$ is also called *granule*[1].

A rough set is formulated by a pair of lower and upper approximations for the given concept $X \subseteq U$. These approximations are defined below.

Definition 4.2. In the given incomplete decision system IDS, for a concept $X \subseteq U$, let $B \subseteq A$, the *lower approximation* ($\underline{B}(X)$) and *upper approximation* ($\overline{B}(X)$) of X in terms of similarity class are defined as,

$$\underline{B}(X) = \{x \in U \mid S_B(x) \subseteq X\}, \ \overline{B}(X) \ = \{x \in U \mid S_B(x) \cap X \neq \phi\} \tag{4.2}$$

The *positive region* $POS_B(\{d\})$ represents the objects that are classified with full certainty as members of decision equivalence class.

Definition 4.3. In the given incomplete decision system $IDS = (U, A \cup \{d\})$, for $B \subseteq A$, the positive region with respect to B is defined as,

$$POS_B(\{d\}) = \bigcup_{X \in U/IND(\{d\})} \underline{B}(X) \tag{4.3}$$

The *dependency measure (gamma measure)* $\gamma_B(\{d\})$ denotes the proportion of objects that belong to the positive region, and is given by,

$$\gamma_B(\{d\}) = \frac{|POS_B(\{d\})|}{|U|} \tag{4.4}$$

A minimal subset of conditional attributes $R \subseteq A$ is said to be *reduct*, if R preserves the original classification as defined by attributes set A.

[1]The term "granule" is used instead of "similarity class" in the rest of the chapter.

Definition 4.4. For the given incomplete decision system IDS, let R be the subset of conditional attributes ($R \subseteq A$), and R is said to be reduct if and only if,

i). $\gamma_R(\{d\}) = \gamma_A(\{d\})$ (*jointly sufficient*)

ii). $\gamma_{R'}(\{d\}) < \gamma_R(\{d\})$ for any $R' \subset R$ (*individually necessary*)

4.3 Related work

P. S. V. S. Sai Prasad et al. [73] proposed Novel Granular Framework (NGF) to overcome the limitations of Kryszkiewicz's model [57] (discussed in the preceding section). And, this section also presents the algorithm IQRAIG_Incomplete which uses NGF. This algorithm achieved a significant computational gain in reduct computation over other approaches for IDS since the NGF allows the incorporation of the *granular refinement* and *positive region removal* features. Therefore, the MapReduce based proposed approaches are developed by parallelizing this existing NGF.

4.3.1 Overview of Novel Granular Framework

In any dependency measure-based approach of attribute reduction, the efficiency of the approach depends on the computation of the positive region, which is calculated based on the lower approximation of each of the decision classes. With the classical rough set theory in CDS, the positive region is computed by forming the equivalence classes using indiscernibility relation. If an object of an equivalence class goes into the positive region, then the remaining objects of the same equivalence class also go into the positive region, because of the *sharing* property of equivalence class. But, in the Kryszkiewicz's model [57] for IDS, the positive region is computed by finding the granule of each object separately. This is because the granule does not support *sharing* property. Hence, finding granule for each object leads to repeated computations in attribute reduction.

Sai Prasad et al. [73] developed a *Novel Granular Framework (NGF)* to overcome the aforementioned problems in the Kryszkiewicz's model. In this framework, the structure of the granule is changed such that it supports sharing property. Thus, a granule is redefined according to NGF, as given below.

Definition 4.5. In the given incomplete decision system IDS, for $B \subseteq A$, if $U/SIM(B)$ is the granular space, then a granule $gr \in U/SIM(B)$ is defined as the collection of objects $gr \subseteq U$ such that, if $x, x' \in gr$ then $x \in S_B(x')$ and $x' \in S_B(x)$. And, the granule gr is further divided into two portions *base* and *tail* as,

$$base = \{x \in gr \mid S_B(x) = gr\}, tail = gr - base, where, base \subseteq U, tail \subseteq U \tag{4.5}$$

The *base* portion of the granule gr is denoted with $gr.base$, and it contains all the objects in gr having the similarity class as gr. And the *tail* portion of the granule gr is denoted with $gr.tail$, and it contains the remaining objects of the similarity class. Note that, for an empty set of attributes $B = \phi$, $U/SIM(B) = \{gr\}$, where, $gr.base = U$, and $gr.tail = \phi$. From the Eq. (4.5), it can be ascertained that, all the *base* portions of the granules constitute $U/IND(B)$ by assuming that the *missing value* is considered as a discernible domain value. Thus, the *base* portions of the granules correspond to the equivalence classes of the indiscernibility relation $IND(B)$.

For any subset of attributes $B \subseteq A$, the lower approximation of $X \subseteq U$ using $U/SIM(B)$ based on NGF is given as,

$$\underline{B}(X) = \bigcup_{(gr \in U/SIM(B)) \ \wedge \ (gr \subseteq X)} gr.base \tag{4.6}$$

From Eq. (4.6), the positive region using $U/SIM(B)$ becomes,

$$POS_B(\{d\}) = \bigcup_{X \in \ U/IND(\{d\})} \underline{B}(X) \tag{4.7}$$

Eqs. (4.6) and (4.7) are modified versions of Eqs. (4.2) and (4.3). Here, Eq. (4.7) states that, the positive region is the union of *base* portions of the granules $U/SIM(B)$, which have the same decision class for the objects in gr (i.e., consistent). Thus, the reduct is computed by using Definition 4.4, where Eq. (4.7) is used to compute the dependency measure (γ) in Eq. (4.4).

4.3.2 IQRAIG_Incomplete algorithm

Based on IQRA_IG algorithm [74], the IQRAIG_Incomplete algorithm [73] is proposed for IDS. This algorithm is developed using the *Sequential Forward Selection (SFS)* strategy, and the *dependency measure* (γ measure) [53] approach for reduct computation. And, IQRAIG_Incomplete uses NGF to deal with incompleteness in the data. The description of the procedure for reduct computation in IQRAIG_Incomplete algorithm is given below.

IQRAIG_Incomplete algorithm starts its first iteration by initializing the reduct set $R = \phi$ (empty set). As this algorithm uses dependency measure (γ) approach, for each attribute $a \in A$, dependency measure is computed using Eq. (4.4) and Eq. (4.7). To compute the positive region, the granular space (granules) $U/SIM(\{a\})$ $\forall a \in A$ is formed. The attribute a^{best} for which the maximum dependency measure obtained is included into the reduct set R. Now the algorithm categorise the granules $U/SIM(R)$ into *positive region* denoted by

$P\_GR(U/SIM(R))$ and *non-positive region* denoted by $NP\_GR(U/SIM(R))$. From Eq. (4.7), a granule $gr \in U/SIM(R)$ is said to be positive region granule, if the objects of $gr.base$ go into positive region (i.e, when the decision class of objects of gr is the same). The granules of $U/SIM(R)$ which are not in positive region are said to be non-positive region granules.

If $gr \in P\_GR(U/SIM(R))$ then $\forall gr' \in gr/SIM(R \cup \{a\})$ for any $a \in (A - R)$, we have $gr' \in P\_GR(U/SIM(R \cup \{a\}))$. That is, if a granule is going into positive region, then any of its sub granules can also go into positive region. Thus, the removal of the *base* portions of the granules in positive region from $U/SIM(R)$ has no effect on future computations. Hence, the algorithm performs *positive region removal* where the *base* portions of the granules of $U/SIM(R)$ which are part of positive region ($P\_GR(U/SIM(R))$) are removed. After removal of positive region, only the non-positive region granules ($NP\_GR(U/SIM(R))$) remain in granular space $U/SIM(R)$.

In the successive iterations of the IQRAIG_Incomplete algorithm, if the reduct is nonempty, the granules need to be formed for $R \cup \{a\}$, $\forall a \in (A - R)$ to compute the *gamma* measure. The granules of $U/SIM(R \cup \{a\})$ are computed based on the objects values of attributes $R \cup \{a\}$. But this computation becomes unnecessary because we have already computed granules of $U/SIM(R)$ in previous iteration. Hence, this redundant computation in each iteration of the algorithm is avoided by using the *granular refinement*, where the granules for attribute a are computed based on the existing granules of $U/SIM(R)$. Note that, from the previous iteration the granular space $U/SIM(R)$ contains non-positive region granules ($NP\_GR(U/SIM(R))$). Thus, if $NP\_GR(U/SIM(R)) = \{gr_1, gr_2, ...gr_s\}$, then by using the granular refinement, the granules of $U/SIM(R \cup \{a\}), \forall a \in (A - R)$ are computed as,

$$U/SIM(R \cup \{a\}) = \bigcup_{i=1}^{s} gr_i/SIM(\{a\}) \tag{4.8}$$

It should be noted that the process of positive region removal and granular refinement in IQRAIG_Incomplete algorithm is similar to IQRA_IG algorithm given in Section 3.2.1. Hence, in the algorithm, after computation of the granules $U/SIM(R \cup \{a\}), \forall a \in (A - R)$ using Eq. (4.8), the $\gamma_{R \cup \{a\}}$ measure is computed using Eq. (4.4) and Eq. (4.7). The best attribute a^{best} which gets maximum *gamma* gain is added to the reduct set R. And, the positive region removal is performed to remove the positive region granules from $U/SIM(R)$. The above procedure is repeated until the algorithm stops. The algorithm terminates when the dependency measure of the reduct attributes ($\gamma_R(\{d\})$ is equal to the dependency measure of all the conditional attributes ($\gamma_A(\{d\})$.

4.4 Proposed parallel attribute reduction in IDS using horizontal partitioning

Horizontal partitioning of the given IDS can be defined as given below.

Definition 4.6. For the given incomplete decision system $IDS = (U, A \cup \{d\})$, let $IDS = \bigcup_{i=1}^{p} IDS^i$, where, $IDS^i = (U^i, A \cup \{d\})$ is i^{th} data partition, and satisfies (i) $U = \bigcup_{i=1}^{p} U^i$, (ii) $U^i \cap U^j = \emptyset, \forall i, j \in \{1, 2, ..., p\}$ *and* $i \neq j$, where p is the number of data partitions.

From Definition 4.6, we can observe that each node gets data partition having subset of objects' information of all the attributes.

4.4.1 Alternative representation of the NGF

According to Definition 4.5, the construction of granule involves the formation of *base* and *tail* portions. Because of these portions, objects in one granule can be repeated in another granule. In IQRAIG_Incomplete algorithm [73], it can be observed that, the *base* and *tail* portions of a granule are represented with object identifiers (*id*s). By default the row number of the object in IDS is considered as object *id*. But, in a parallel/distributed approach with the horizontal partitioning strategy, the object *id*s based representation is found not suitable due to increased shuffle and sort phase complexity. From the existing horizontal partitioning based attribute reduction algorithms in classical rough sets [81, 82, 92, 96, 124], it is observed that the granular structure is usually represented with the *granular signature* instead of object *ids*. In this section, the granular signature-based formulation for the NGF is introduced. Here, this alternative representation of the NGF is discussed in the standalone scenario, and its parallel/distributed outline is given in subsequent sections. This representation is adopted in the proposed horizontal partitioning based approach.

If we project the *base* portions of the granular space for the subset of attributes $B \subseteq A$, we can notice that the collection of all the *base* portions are the equivalence classes induced from the indiscernibility relation (i.e., $U/IND(B)$) on the information of the attributes B by considering the missing value also as a known discernible value. Hence, in the alternative representation of NGF, the *base* portions are first computed using $IND(B)$, and then the *tail* portions are constructed from the existing *base* portions.

Consider the distinct similarity classes for $B \subseteq A$ as $U/SIM(B) = \{gr_1, gr_2, ..., gr_k\}$, and $U/IND(B) = \{gr_1.base, gr_2.base, ..., gr_k.base\}$. In the alternative representation of NGF, the *base* portion of a granule is represented with a pair of $< bSig, (decVal, posCount) >$ instead of object *id*s. The $bSig$ represents the unique value combination for attributes in B attained by objects of granule. While computing *base* portion, the consistency of the *base* portion

is extracted. If the decision class of all the objects in the *base* portion are the same then the *base* portion is consistent, and the corresponding unique decision class label $decVal$ is preserved, otherwise $decVal$ is represented with -1. If the *base* portion is consistent, then the cardinality of the *base* portion is stored in $posCount$. Preserving aforementioned information is sufficient for the computation of the *tail* portions, positive region count ($|POS_B(\{d\})|$), and the dependency measure ($\gamma_B(\{d\})$) as shown below.

From Definition 4.5, the objects of the *tail* portion in a granule are similar to the corresponding *base* portion using SIM relation. Hence, the *tail* portion of a granule can be viewed as the union of other *base* portions whose *base signature* ($bSig$) is similar to the current *base signature* under the SIM relation. The *tail* portions of the existing *base* portions of the granules are computed as given below.

Definition 4.7. In the given incomplete decision system $IDS = (U, A \cup \{d\})$, for $B \subseteq A$, let $U/IND(B) = \{gr_1.base, gr_2.base, ..., gr_k.base\}$. For the *base* portion of an i^{th} granule gr_i, the tail portion $gr_i.tail$ is computed as,

$$gr_i.tail = \bigcup_{j=1}^{k} \{gr_j.base \mid (i \neq j) \ \wedge \ (SIM(bSig(gr_i.base), bSig(gr_j.base)) == TRUE)\} \tag{4.9}$$

Note that, $SIM(bSig(gr_i.base), bSig(gr_j.base))$ becomes $TRUE$ when the following criteria is satisfied. If $bSig(gr_i.base) = (a_1, a_2, ...a_{|B|})$ *and* $bSig(gr_j.base) = (b_1, b_2, ...b_{|B|})$, *then* $\forall k \in \{1, 2, ...|B|\}$, *we have* $a_k == b_k$ *or* $a_k = *$ *or* $b_k = *$.

From Eq. (4.9), we say that gr_i is a consistent granule if $gr_i.base$ and all the *base* portions in $gr_i.tail$ are consistent and they all have same $decVal$. Let $ConsistentGranules(B)$ represents the set of granules of $U/SIM(B)$ which are consistent. To find the positive region count $|POS_B(\{d\})|$, Eq. (4.7) in NGF is modified as given below.

$$|POS_B(\{d\})| = \sum_{gr \in ConsistentGranules(B)} posCount(gr.base) \tag{4.10}$$

The dependency measure $\gamma_B(\{d\})$ given in Eq. (4.4) is computed using Eq. (4.10).

In this proposed approach, the attribute reduction is performed majorly in two steps. In the first step, the granules for the given data set are computed based on the alternative representation of NGF. In the second step, the *best attribute* to include in the reduct set is computed based on the granules. Parallelization of these two steps and the incorporation of the positive region removal facet is described using MapReduce-based parallel/distributed algorithms. These algorithms are given in the form of a driver, mapper, and reducer. The driver

algorithm MRIDS_HP: driver() is given in Algorithm 4.1, the mapper algorithm MRIDS_HP: map() is given in Algorithm 4.2, and the reducer algorithm MRIDS_HP: reduce() is given in Algorithm 4.3. And, the algorithm MRIDS_HP: mapValues() is given in Algorithm 4.4. This algorithm is used to find the best attribute.

4.4.2 Parallel computation of the base portions

With the horizontal partitioning strategy, each data partition of a node gets a subset of objects information of an attribute. So within a node, partial *base* portions are formed in the mapper phase, and the complete *base* portions are realized after the reducer phase. As given in Algorithm 4.1, in any iteration, the driver maintains non-positive region *base* signatures for the current reduct R in $nPosbSig$ variable. At the beginning, the driver initializes the variables $R = \phi$ and $nPosbSig[\,][\,] = \phi$. Before starting its first iteration, the driver assumes that all the objects of U belong to non-positive *base* signature of $nPosbSig$. The driver starts each iteration by broadcasting R and $nPosbSig$ variables to all the nodes of the cluster. In the mapper phase, the driver computes the partial *base* portions $U^i/IND(R \cup \{a\})$, $\forall a \in (A - R)$ by invoking Algorithm 4.2.

The mapper (Algorithm 4.2) works on each *record* of $IDSRDD$. The mapper computes the *base* signature $RbSig$ of all the attributes of current reduct R for the given *record*. If $RbSig$ does not belong to $nPosbSig$, then $RbSig$ is a positive region *base* signature, hence that particular *record* is not considered for further computations. And if $RbSig$ belongs to $nPosbSig$, then that particular *record* is considered to generate the intermediate $< key', value' >$ pair for each attribute $attr \in (A - R)$. The key' contains $< bId, attr, objVal >$ where the bId is *base* signature index in $nPosbSig$, $objVal$ is the object value in the given *record* for the attribute $attr$. And $value'$ contains $< decVal, posCount >$ where $decVal$ is the decision value in the *record*, and $posCount$ is the positive region count, which is initialized with 1. These $< key', value' >$ pairs form the new RDD as $parbSigRDD < (bId, attr, objVal), (decVal, posCount) >$. From Algorithm 4.2, it can be noticed that the objective of the mapper is to generate the requisite information for constructing *base* portions $U/IND(R \cup \{attr\})$, $\forall attr \in (A - R)$. Here for each $attr \in (A - R)$, the mapper collates the *base* signature information corresponding to $U/IND(R \cup \{attr\})$ using the combination of bId and $objVal$ of the $attr$ in the given *record* and the information pertaining to the R attributes that can be extracted by using broadcasted variable $nPosbSig$.

4. PARALLEL ATTRIBUTE REDUCTION IN INCOMPLETE DECISION SYSTEMS

Algorithm 4.1: MRIDS_HP: driver()

Input: Input data set $IDS = (U, A \cup \{d\})$, Positive threshold: α, Threshold for trivial iterations: $thVal$

Output: Reduct R

1 $IDSRDD = readAsRDD(IDS)$ `/* Data set is divided using horizontal partitioning strategy, and distribute to the nodes of the cluster as` $IDS^i = (U^i, A \cup \{d\})\ \forall i \in \{1, 2, ..., p\}$ `where` p `is number of data partitions. */`

2 Initialize $trivialIteration = 0$, $additionalAttrs = \phi$, $posTh = \alpha$

3 Broadcast reduct $R = \phi$, non-positive region base signatures list $nPosbSig[\,][\,] = \phi$

4 **repeat**

`// Initiate mapper job by invoking Algorithm 4.2`

5 $val\ parbSigRDD = IDSRDD.map(record => \{var\ m = map()\})$

`// Initiate reducer job by invoking Algorithm 4.3`

6 $val\ bSigRDD = parbSigRDD.reduceByKey((x, y) => \{var\ r = reduce()\}))$

7 Using a map() only operation, transform the $< key, value >=$ pairs of $bSigRDD$ such that $key = attr$ and $value = (bId, objVal, decVal, posCount)$

`// Aggregate all values of same key as a list of tuples`

8 $val\ aggrRDD = bSigRDD.aggregateByKey()$

`//` $aggrRDD$ `gets` $< key, value > =< attr, List[(bId, objVal, decVal, posCount), ...] >$

`// Find the best attribute by invoking Algorithm 4.4`

9 $val\ attrPosRDD = aggrRDD.mapValues()$

`/*` $attrPosRDD$ `gets` $< attr, (attrPosCount, attrnPosbSig) >$ `pairs */`

10 Select $attr$ as a^{best} which gets maximum $attrPosCount$ (denoted as $bestPosCount$), store its $attrnPosbSig$ into $nPosbSig$ and broadcast.

11 $R = R \cup \{a^{best}\}$, $totalPC = totalPC + bestPosCount$

12 Compute $\gamma_R = \frac{totalPC}{|U|}$

13 **if** $(\gamma_R >= posTh)$ **then**

14 Filter the objects from $IDSRDD$ which are not in $nPosbSig$

15 $posTh = \alpha + \gamma_R$

16 **end**

17 **if** $(\gamma_R < 1\ AND\ bestPosCount == 0)$ **then**

18 Store a^{best} to $additionalAttrs$

19 $trivialIteration++$

20 **end**

21 **else**

22 $trivialIteration = 0$

23 Remove attributes from $additionalAttrs$

24 **end**

25 **if** $(trivialIteration == thValue)$ **then**

26 Remove attributes of $additionalAttrs$ from R

27 **end**

28 **until** $(\gamma_R < 1\ AND\ trivialIteration < thValue)$

29 Return R

Algorithm 4.2: MRIDS_HP: map()

Input: 1. A *record* of $IDSRDD$
2. Broadcasted reduct: R, and non-positive region base signatures: $nPosbSig$

Output: Partial base portions in the form of $< key', value' >$

```
1  Construct base signature for the attributes of R based on record and store into RbSig
2  decVal = record[d]
3  posCount = 1
4  if RbSig ∈ nPosbSig then
5  |  bId=index of RbSig in nPosbSig
6  |  for each attribute attr ∈ (A − R) do
7  |  |  objVal = record[attr]
8  |  |  Construct < key', value' > pair, where key' = (bId, attr, objVal) and
   |  |   value' = (decVal, posCount)
9  |  |  Generate intermediate < key', value' > pair
10 |  end
11 end
```

In Algorithm 4.3, the driver performs $reduceByKey()$ operation on the $parbSigRDD$. Each reducer gets a key as $< bId, attr, objVal >$, and a corresponding list of values V, where each $v \in V$ is $< decVal, posCount >$. That is, in the reducer phase, the complete form of the *base* portion is obtained by collecting all the *value* portions $< decVal, posCount >$ of same $key =< bId, attr, objVal >$. After forming the complete *base* portions, the consistency of the *base* portion is checked. As shown in the algorithm, for a *base* portion (key), if all the $decVal$s are same and $decVal! = -1$, then that *base* portion is consistent and the *value* portion becomes $< decVal, sum\ of\ all\ posCount >$, otherwise it becomes $< -1, 0 >$. Here, -1 indicates inconsistent *base* portion. Hence, $< key', value' >$ pairs of reducer form complete *base* portion and results in the formation of new RDD of $bSigRDD < (bId, attr, objVal), (decVal, posCount) >$. It should be noted that, with the $reduceByKey()$ operation, internally Spark performs local optimization by applying the operation on local values in mapper phase prior to its global optimization. Additionally, both in mapper and reducer, it can be observed that, instead of *base* signature its bId is communicated in the key portion. With this, the amount of data movement in the shuffle and sort phase is reduced.

Algorithm 4.3: MRIDS_HP: reduce()

Input: $< key, V >$ pair where, key is a $(bId, attr, objVal)$ received from all the mappers, V is a list of values, where each value v is $(decVal, posCount)$ generated from a mapper

Output: Complete *base* portion of an attribute in the form of $< key', value' >$ pair

1 **if** *($decVal$ is same in the list of V AND $decVal! = -1$)* **then**
2 | $value'$ =($decVal$, sum of all $posCounts$)
3 **end**
4 **else**
5 | $value' = (-1, 0)$
6 **end**
7 Construct $< key', value' >$ pair, where $key' = (bId, attr, objVal)$, and $value' = (decVal, posCount)$
8 Emit $< key', value' >$

4.4.3 Parallel computation of the best attribute

Algorithm 4.4: MRIDS_HP: mapValues()

Input: 1. $< key, V >$ pair where, key is an $attr$, V is a list of tuples, where each tuple v is $(bId, objVal, decVal, posCount)$
2. Broadcasted reduct R, non-positive base signatures list $nPosbSig$

Output: An attribute $attr$, its positive region count $attrPosCount$ and its non positive region base signatures list $attrnPosbSig$

1 For each tuple $v \in V$, construct *base* signature of *base* portion of granule in $U/SIM(R \cup \{attr\})$ with the information of R attributes from $nPosbSig[bId]$ and information of $attr$ in $objVal$
2 Compute *tail* for each *base* of granule $gr \in U/SIM(R \cup \{attr\})$ using Eq. (4.9)
3 $attrPosCount = 0$, $attrnPosbSig[\,][\,] = \phi$
4 **for** *each tuple $gr.base \in V$ of $attr$* **do**
5 | **if** $(Consistency(gr.base)! = TRUE)$ **then**
6 | | $attrnPosbSig = attrnPosbSig \cup bSig(gr.base)$
7 | **end**
8 | **else if** $(Consistency(gr)! = TRUE)$ **then**
9 | | $attrnPosbSig = attrnPosbSig \cup bSig(gr.base)$
10 | **end**
11 | **else**
12 | | $attrPosCount = attrPosCount + posCount(gr.base)$
13 | **end**
14 **end**
15 Emit $< key', value' >=< attr, (attrPosCount, attrnPosbSig) >$

In an iteration of the proposed algorithm (Algorithm 4.1), the attribute which gets maximum positive region count among the attributes of $(A - R)$ is selected as the best attribute a^{best}. To compute the required positive region count of an attribute $attr \in (A-R)$, we need to gather all the *base* portions of the $attr$ to a particular location. To do this, the driver first per-

forms $map()$ operation on $bSigRDD$ to transform the $< key, value >$ pair, so that, $key =< attr >$ and $value =< bId, objVal, decVal, posCount >$. And, then the driver performs $aggregateByKey()$ for aggregating $value$ portions ($< bId, objVal, decVal, posCount >$) of a $key =< attr >$. The driver gets $aggrRDD < attr, List[(bId, objVal, decVal, posCount), ...] >$ as the result of $aggregateByKey()$ operation. That is $aggrRDD$ contains a key as $attr$, and all the $base$ portions $U/IND(R \cup \{attr\})$ as a list of tuples in the $value$ portion $\forall attr \in (A-R)$. By performing $mapValues()$ operation on $aggrRDD$, the computation of $tail$ portions from $U/IND(R \cup \{attr\})$ and the computation of the positive region count ($|POS_{R\cup\{attr\}}(\{d\})|$) are parallelized.

As given in Algorithm 4.4, by using Definition 4.7, the $tail$ portion for each $base$ portion of an attribute is computed. After the formation of $tail$ portions, each granule gets its $base$ and $tail$ portions so that the granule gr becomes a similarity class in $U/SIM(R \cup \{attr\})$. If the granule gr is consistent then it's $gr.base$ portion goes to positive region (using Eq. (4.10)). Otherwise the $gr.base$ portion is added to $attrnPosbSig$. For the consistent granules, the positive region count $|POS_{R\cup\{attr\}}(\{d\})|$ is computed by using Eq. (4.10). For each attribute $attr \in (A - R)$, Algorithm 4.4 generates $< key', value' >=< attr, (attrPosCount, attrnPosbSig) >$ that result in $attrPosRDD$.

As given in Algorithm 4.1, the best attribute is selected (which gets maximum positive count) by performing a reduce operation on $attrPosRDD$. The best attribute a^{best} is added to the reduct set R, and the dependency measure γ_R is computed. From Definition 4.4, the driver algorithm should be terminated when the value of $\gamma_R(\{d\})$ reaches the value of $\gamma_A(\{d\})$. Here, the computation of $\gamma_A(\{d\})$ may not be feasible for the massive data sets with the horizontally partitioned data. Hence, an alternative approach is used to terminate the algorithm. As mentioned earlier, the attribute which gets maximum $gamma$ gain is added to the reduct set, but in an iteration, if there is no $gamma$ gain, then the algorithm selects an attribute randomly and added it to the reduct set, and then the next iteration is repeated. The non-increasing nature of the dependency measure can be because of the data set or the obtained reduct set already achieved $\gamma_R(\{d\})$ equal to $\gamma_A(\{d\})$. To resolve this ambiguity, the algorithm continue the iterations for some threshold value ($thValue$) number of times and even if there is no increase in $\gamma_R(\{d\})$ then the algorithm safely assumes that the required reduct is obtained. Note that, the value of $thValue$ variable is taken as input to the algorithm based on the number of attributes in the input data set.

4.4.4 Positive region removal

The driver (in Algorithm 4.1) incorporates a stage based positive region removal. Without performing positive region removal it can be observed that in the mapper phase, on each *record*, the *base* signature ($RbSig$) is computed and verified whether it belongs to $nPosbSig$ or not. If $RbSig$ does not belong to $nPosbSig$, then the objects of the $RbSig$ are considered in the positive region and they are not used in further computations. This verification becomes an expensive and repetitive operation. But effecting the physical removal of the positive region in every iteration results in a lot of reorganization of the $IDSRDD$ with $filter()$ operation. If the number of objects being removed is very small, then the overhead cost incurred with the reorganization of $IDSRDD$ dominates the benefits obtained by positive region removal.

To avoid the aforementioned problems, a stage based positive region removal is performed in which a threshold α is taken as input from the user and it is assigned to $posTh$ variable (in our implementation α is taken as 0.25). And, the positive region removal is not performed till $\gamma_R(\{d\}) >= posTh$. If $\gamma_R(\{d\})$ exceeds $posTh$, then the objects in positive region are filtered out from the input data set $IDSRDD$. And $posTh$ is set to $\gamma_R(\{d\}) + \alpha$. In the next stage, the positive region is removed based on the current $posTh$ value.

4.4.5 Complexity analysis of MRIDS_HP algorithm

Table 4.1: Time complexity analysis of MRIDS_HP algorithm

Algorithm (phase)	**Step* in Algorithm**	**Time complexity**
Driver → (Algorithm 4.1)	1. Partitioning the data horizontally 3. Broadcasting $nPosbSig$	$\mathcal{O}(\frac{\|A\|*\|U\|}{p} * t_w)$ $\mathcal{O}(\|A\| * \|U/IND(A)\| * t_w)$
Mapper → (Algorithm 4.2)	4-10. Creating partial base signatures	$\mathcal{O}(\frac{\|A\|*\|U\|log\|U\|}{p})$
Shuffle and sort1 →	Transferring partial base signatures	$\mathcal{O}(p*\|A\|*\|U/IND(A)\|*t_w)$
Reducer1 → (Algorithm 4.3)	1-6. Creating complete base signatures and Barrier synchronization	$\mathcal{O}(\frac{p*\|A\|*\|U/IND(A)\|}{q}) + \mathcal{O}(s)$
mapValues() → (Algorithm 4.4)	4-14. Forming complete granules (based on alternative NGF)	$\mathcal{O}(\frac{\|A\|*\|U/IND(A)\|^2}{p'})$
Shuffle and sort2 →	Transferring data after mapValues()	$\mathcal{O}(p'*\|A\|*\|U/IND(A)\|*t_w)$
Reducer2 → (Algorithm 4.1)	10. Finding a^{best} and Barrier synchronization	$\mathcal{O}(p') + \mathcal{O}(s)$

* Step denotes the line number in the associated algorithm

In the time complexity analysis of MRIDS_HP algorithm, the following variables are used.

- $|U|$: the number of objects in the data set
- $|A|$: the number of conditional attributes in the data set
- p: the number of processors
- t_w: the number of time units to transfer one word of memory
- s: the number of time units to complete the synchronization
- $|U/IND(A)|$: the number of granules in granular space formed by A
- q: the number of reducers invoked by Algorithm 4.3
- p': the number of data partitions after $aggregateByKey()$ operation

Table 4.1 shows the time complexities of each step of the phase in the MRIDS_HP algorithm for one iteration. Note that, in the table, all the steps in all the phases (i.e., Algorithm 4.2 (mapper), shuffle and sort1, Algorithm 4.3 (reducer1), shuffle and sort2 and Algorithm 4.4 (mapValues)) and step 10 of the driver (Algorithm 4.1) are repeated until ($\gamma_R < 1$ AND $trivialIteration < thValue$) condition is satisfied. That is, these steps are repeated $|A|$ (in worst case) times. Hence, by adding up all the complexities, the total time complexity of the MRIDS_HP algorithm is obtained as given below.

$$(\frac{|A| * |U|}{p} * t_w) + (|A| * |U/IND(A)| * t_w) + |A| * ((\frac{|A| * |U| log|U|}{p}) + (p * |A| * |U/IND(A)| * t_w) + (\frac{p * |A| * |U/IND(A)|}{q}) + (s) + (\frac{|A| * |U/IND(A)|^2}{p'}) + (p' * |A| * |U/IND(A)| * t_w) + (p') + (s)) \tag{4.11}$$

Above equation can be approximated as: $\mathcal{O}(\frac{|A|^2 * |U| log|U|}{p}) + \mathcal{O}(|A| * (|A| * |U/IND(A)| * (p * t_w + \frac{p}{q} + \frac{|U/IND(A)|}{p'} + p, * t_w) + p' + s)$. Thus the time complexity of the MRIDS_HP algorithm is $\mathcal{O}(\frac{|A|^2 * |U| log|U|}{p})$ in addition with its communication cost: $\mathcal{O}(|A| * (|A| * |U/IND(A)| * (p * t_w + \frac{p}{q} + \frac{|U/IND(A)|}{p'} + p, * t_w) + p' + s)$.

The entire decision system is required to be present in memory for reduct computation using MRIDS_HP algorithm. Thus, the space complexity of MRIDS_HP algorithm is $\mathcal{O}(|A| * |U|)$. Additionally, the driver of MRIDS_HP algorithm has to maintain broadcasting non-positive region base signatures list, thus it has the complexity of $\mathcal{O}(|A| * |U/IND(A)|)$. In the

MapReduce framework environment, the input decision system is partitioned and distributed to the nodes of the cluster where the workload is divided equally into p data partitions. Hence, each partition has the complexity of $\mathcal{O}(\frac{|A|*|U|}{p})$.

In the worst-case scenario, the aforementioned theoretical time and space complexities of the proposed MRIDS_HP algorithm are described. However, since the MRIDS_HP algorithm incorporates positive region removal feature, the actual time and space complexities are further reduced.

4.5 Proposed parallel attribute reduction in IDS using vertical partitioning

The horizontal partitioning based attribute reduction algorithm MRIDS_HP is not scalable for big dimensional data sets. Because, the granular signature representation of granules in NGF suffers from overheads of data movement across the cluster of nodes in the MapReduce framework. Furthermore, it does not allow the incorporation of a granular refinement feature which gives enormous computational gains in parallel/distributed computing as proved in Chapter 3. Thus, vertical partitioning based approach is required to achieve scalability in attribute reduction of big dimensional incomplete data sets. With the vertical partitioning strategy given in Section 3.3, all the objects' information of a subset of attributes is available in a data partition of a node in the cluster. Vertical partitioning of the incomplete data set is defined below.

Definition 4.8. For the given incomplete decision system IDS, let $IDS = \bigcup_{i=1}^{p} IDS^i$ be the vertically partitioned decision system, where, $IDS^i = (U, A^i \cup \{d\})$ is i^{th} data partition, and satisfies (i) $A = \bigcup_{i=1}^{p} A^i$, (ii) $A^i \cap A^j = \emptyset, \forall i, j \in \{1, 2, ..., p\}$ *and* $i \neq j$, where p is the number of data partitions.

Each data partition contains $A^i \cup \{d\}$ attributes with the information of all the objects of U. Therefore, adopting the representation of the granules in the existing NGF using object *id*s is naturally carried forward from the standalone scenario to vertical partitioning based parallel approach because all the objects pertaining to the attributes are available at the same location (within a node). As a result, the proposed MapReduce based method (MRIDS_VP) adopts vertical partitioning strategy to partition the input data set to cluster nodes, and it uses the NGF to handle the incompleteness in the data.

In the proposed approach, attribute reduction is performed majorly in two steps. In the first step, the given data set's granules are computed based on the NGF. In the second step, the *best attribute* to include in the reduct set is computed based on the granules. Parallelization of

these two steps and the incorporation of granular refinement and positive region removal facets are described by using MapReduce based parallel/distributed algorithms. These algorithms are given in the form of a driver, mapper, and reducer. The driver algorithm MRIDS_VP: driver() is given in Algorithm 4.5, the mapper algorithm MRIDS_VP: map() is given in Algorithm 4.6, and the reducer algorithm MRIDS_VP: reduce() is given in Algorithm 4.7.

The driver (Algorithm 4.5) initializes the reduct $R = \phi$ (empty set). The initial granules $U/SIM(R) = \{\{U\}, \{\phi\}\}$ (when $R = \phi$) are assigned to non-positive region granules list NP_GR, where *base* portion of the granule is initialized with U and *tail* portion is initialized with ϕ (empty set). The driver then broadcasts these initialized variables along with the decision attribute $\{d\}$ to all the nodes of the cluster.

Algorithm 4.5: MRIDS_VP: driver()

Input: Input data set $IDS = (U, A \cup \{d\})$
Output: Reduct R

1 $IDSRDD = readAsRDD(IDS)$ `/* Reads the input data set as RDD, divides it using vertical partitioning strategy, and distribute to the nodes of the cluster as` $IDS^i = (U, A^i \cup \{d\})$, $\forall i \in \{1, 2, ..., p\}$ `where` p `is number of data partitions in the cluster. */`
2 Broadcast decision attribute $\{d\}$, initial reduct $R = \phi$, and initial non-positive region granules list $NP\_GR = \{\{U\}, \{\phi\}\}$ to all the nodes of the cluster.
3 Compute $\gamma_A(\{d\})$
4 **repeat**
  `// Initiate mapper job by invoking Algorithm 4.6`
5 $val\ lBestRDD = IDSRDD.mapPartitions(part => \{var\ mp = map()\})$
  `// Initiate reducer job by invoking Algorithm 4.7`
6 $val\ gBestRDD = lBestRDD.reduceByKey((x, y) => \{var\ rp = reduce()\}))$
7 Collect the data $gBestRDD$ as $< key, value >=< a^{best}, globalPC >$ from the reducer.
8 Add the best attribute a^{best} to the reduct set R
9 Calculate total positive region count $|POS_R(\{d\})|$ of reduct set R
10 Compute dependency measure γ_R using Eq. (4.4)
11 **if** $(\gamma_R < \gamma_A)$ **then**
12 Filter the record of the attribute a^{best} from the input data set $IDSRDD$
13 Remove the attribute a^{best} from the reduct set R
14 Compute $U/SIM(R \cup \{a^{best}\})$ using Eq. (4.8) `// Granular refinement`
15 Perform positive region removal
16 Broadcast non-positive region granules $NP\_GR(U/SIM(R \cup \{a^{best}\})$
17 Add the attribute a^{best} to the reduct set R
18 **end**
19 **until** $(\gamma_R == \gamma_A)$
20 Return R

Algorithm 4.6: MRIDS_VP: map()

Input: 1. Data partition $IDS^i = (U, A^i \cup \{d\})$, each record read as $< key, value >=< a, aData >$
2. Broadcasted reduct: R, non-positive region granules: NP_GR

Output: $< key', value' >=< commonKey, (la^{best}, localPC) >$ here, $commonKey$ is some common key, $la^{best} \in A^i$ is local best attribute within the partition, and $localPC$ is its positive region count

$max = 0$, $la^{best} = -1$
for *each* $record \in IDS^i$ *as* $< a, aData >$ **do**
  if $a \notin R$ **then**
    Compute $U/SIM(R \cup \{a\})$ using Eq. (4.8) `// Granular refinement`
    Compute $POS_{R\cup\{a\}}(\{d\})$ using Eq. (4.7)
    if $|POS_{R\cup\{a\}}(\{d\})| > max$ **then**
      $la^{best} = a$
      $max = |POS_{R\cup\{a\}}(\{d\})|$
    end
  end
end
Construct $< key', value' >$ pair, where $key' = commonKey$, and $value' = (la^{best}, localPC)$, here $localPC = max$
Emit intermediate $< key', value' >$

Algorithm 4.7: MRIDS_VP: reduce()

Input: $< key, V >$, where key is a $commonKey$ received from all the mappers, V is a list of values, where each value is $(la^{best}, localPC)$ generated from a mapper

Output: $< key', value' >=< a^{best}, globalPC) >$ here, a^{best} is the best attribute, and $globalPC$ is it's positive region count

$globalPC = 0$, $a^{best} = -1$
for *each* $v \in V$ *as* $(la^{best}, localPC)$ **do**
  if $localPC > globalPC$ **then**
    $a^{best} = la^{best}$
    $globalPC = localPC$
  end
end
Construct $< key', value' >$ pair, where $key' = a^{best}$, and $value' = globalPC$
Emit $< key', value' >$

4.5 Proposed parallel attribute reduction in IDS using vertical partitioning

4.5.1 Parallel computation of the best attribute

The driver computes the granules by invoking the mapper (Algorithm 4.6). A mapper gets a data partition IDS^i as input and broadcasted variables R and NP_GR from the driver. With the vertical partitioning strategy, for an attribute $a \in (A^i - R)$, since all the objects' information is available within the data partition, the complete granules of $U/SIM(R \cup \{a\})$ are formed locally within the node based on Eq. (4.5). Thus, each mapper computes the granules $U/SIM(R \cup \{a\})$, $\forall a \in (A^i - R)$ by using the granular refinement as per Eq. (4.8). Here, broadcasted granules of $NP\_GR(U/SIM(R))$ are refined based on the values of the current attribute a. Based on these granules, the positive region count $|POS_{R\cup\{a\}}(\{d\})|$, $\forall a \in (A^i - R)$ can also be computed locally within the node using Eq. (4.7).

As shown in Algorithm 4.6, after computation of the granules, the mapper computes the positive region count for each attribute $a \in (A^i - R)$ within the data partition IDS^i. The attribute which gets maximum positive region count $localPC$ is selected as local best attribute la^{best}. Each mapper generates $< key', value' >$ pair that result in $lBestRDD < commonKey, (la^{best}, localPC) >$. Here, $commonKey$ is a common key from all the mappers used to invoke a single reducer.

In Algorithm 4.7, the reducer gets a set of $< commonKey, [(la^{best}, localPC)] >$ pairs as the input from all the mappers located in different nodes of the cluster. Here, $[(la^{best}, localPC)]$ represents a list of values. The reducer selects a global best attribute a^{best} which gets maximum positive region count among the local best attributes received from different mappers. The reducer generates global best attribute and its positive region count. In the driver, the best attribute a^{best} is added to the reduct set R, and the total positive region count $|POS_R(\{d\})|$ of reduct set R is calculated. The γ_R is computed based on $|POS_R(\{d\})|$.

4.5.2 Granular refinement and positive region removal

In an iteration of the driver (Algorithm 4.5), after adding the best attribute a^{best} to the reduct set R, the end condition ($\gamma_R == \gamma_A$) of the algorithm is checked, where γ_A is the dependency measure of all the conditional attributes (refer Section 4.5.3). If the condition is true, then the algorithm returns reduct set R and terminates. But if the condition is false, then the record of the best attribute a^{best} is fetched by using a $filter$ operation on the input data set $IDSRDD$. The granules $U/SIM(R \cup \{a^{best}\})$ are constructed based on the fetched record of the best attribute using granular refinement (line number 14 of the Algorithm 4.5). Notice that we already computed the granules $U/SIM(R \cup \{a^{best}\})$ in the mapper phase, but the granules are not communicated to the reducer to simplify the shuffle and sort phase, which is most complex phase of the MapReduce framework. Thus, we avoid a lot of data movement in

the shuffle and sort phase, which is much more expensive than performing a $filter$ operation to fetch the record of the best attribute in the driver.

The positive region granules $P\_GR(U/SIM(R \cup \{a^{best}\}))$ are removed from the granules of $U/SIM(R \cup \{a^{best}\})$, and the remaining non-positive region granules are assigned to $NP\_GR(U/SIM(R \cup \{a^{best}\}))$. These updated non-positive region granules are broadcasted to all the nodes of the cluster. And, the best attribute a^{best} is added back to reduct set R. Consequently, in the next iteration of the algorithm, the granules are formed based on these updated non-positive region granules using granular refinement. Hence, without physically removing the positive region objects from the data, the positive region removal is incorporated by restricting the computations of the next best attribute selection to the non-positive region granules. The positive region removal leads to a reduction of the number of objects involved in the computations of every iteration in the algorithm. Therefore, both the facets of granular refinement and positive region removal enhance the efficiency of the proposed MRIDS_VP algorithm.

4.5.3 Computation of $\gamma_A(\{d\})$

The dependency measure for all the conditional attributes $\gamma_A(\{d\})$ is computed by forming the granules of $U/SIM(A)$. For each data partition $IDS^i = (U, A^i \cup \{d\})$, the mapper computes the granules of $U/SIM(A^i)$, and they are categorised as positive region granules $P\_GR(U/SIM(A^i))$ and non-positive region granules $NP\_GR(U/SIM(A^i))$. Here, the mapper removes granules of $P\_GR(U/SIM(A^i))$ from $U/SIM(A^i)$ (positive region removal), and communicates the granules of $NP\_GR(U/SIM(A^i))$ to the reducer. Now, the reducer performs the refinement of $NP\_GR(U/SIM(A^i))$, $\forall i = \{1, 2, ..., p\}$ and arrives at $NP\_GR(U/SIM(A))$. If $NP\_GR(U/SIM(A))$ is empty then $\gamma_A(\{d\})$ becomes 1, otherwise the $POS_A(\{d\})$ is computed as, $POS_A(\{d\}) = U - \bigcup_{gr \in NP\_GR(U/SIM(A))} gr.base$. The value of $\gamma_A(\{d\})$ is computed based on $POS_A(\{d\})$.

4.5.4 Complexity analysis of MRIDS_VP algorithm

In the time complexity analysis of MRIDS_VP algorithm, the following variables are used.

- $|U|$: the number of objects in the data set
- $|A|$: the number of conditional attributes in the data set
- p: the number of processors
- t_w: the number of time units to transfer one word of memory

Table 4.2: Time complexity analysis of MRIDS_VP algorithm

Algorithm (phase)	Step* in the Algorithm	Time complexity
Driver → (Algorithm 4.5)	1. Partitioning the data vertically	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert}{p} * t_w)$
	2. Broadcasting NP_GR and $\{d\}$	$\mathcal{O}(\lvert U\rvert * t_w)$
	3. $\gamma_A(\{d\})$ computation	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert log\lvert U\rvert+\lvert U/IND(A)\rvert^2}{p}) + \mathcal{O}(p * \lvert U\rvert * t_w)$
	12. Fetching a^{best} record	$\mathcal{O}(\lvert U\rvert * t_w)$
	14. Finding granules based on a^{best} record	$\mathcal{O}(\lvert U\rvert log\lvert U\rvert)$
Mapper → (Algorithm 4.6)	5-6. Creating granules and finding positive region counts	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert log\lvert U\rvert+\lvert U/IND(A)\rvert^2}{p})$
	7-10. Finding la^{best} and Barrier synchronization	$\mathcal{O}(\frac{\lvert A\rvert}{p}) + \mathcal{O}(s)$
Shuffle and sort →	Transferring all la^{best} and their positive region counts	$\mathcal{O}(p * t_w)$
Reducer → (Algorithm 4.7)	2-7. Finding a^{best} and Barrier synchronization	$\mathcal{O}(p) + \mathcal{O}(s)$

* Step denotes the line number in the associated algorithm

- s: the number of time units to complete the synchronization
- $|U/IND(A)|$: the number of granules in granular space formed by A

Table 4.2 shows the time complexities of each step of the phase in the MRIDS_VP algorithm for one iteration. Note that, in the table, from step 12 to step 14 in driver (Algorithm 4.5) and all the steps in mapper (Algorithm 4.6) and reducer (Algorithm 4.7) are repeated until ($\gamma_R == \gamma_A$) condition is satisfied. That is, these steps are repeated $|A|$ (in worst case) times. Hence, by adding up all the complexities, the total time complexity of the algorithm is obtained as given below.

$$(\frac{|A| * |U|}{p} * t_w) + (|U| * t_w) + (\frac{|A| * |U|log|U| + |U/IND(A)|^2}{p}) + (p * |U| * t_w) + |A| * \left(|U| * t_w + |U|log|U| + (\frac{|A| * |U|log|U| + |U/IND(A)|^2}{p}) + \frac{|A|}{p} + (s) + (p * t_w) + (p) + (s) \right) \tag{4.12}$$

Above equation is approximated as, $\mathcal{O}(\frac{|A|^2*|U|log|U|}{p}) + \mathcal{O}(|A| * ((|U| + p) * t_w + \frac{|A|+|U/IND(A)|^2}{p} + s))$. Thus the time complexity of the MRIDS_VP algorithm is $\mathcal{O}(\frac{|A|^2*|U|log|U|}{p})$ in addition with its communication cost: $\mathcal{O}(|A| * ((|U| + p) * t_w + \frac{|A|+|U/IND(A)|^2}{p} + s))$.

The entire decision system is required to be present in the memory for reduct computation using MRIDS_VP algorithm. Thus, the space complexity of MRIDS_VP algorithm is $\mathcal{O}(|A| * |U|)$. Furthermore, the driver of MRIDS_VP algorithm has to maintain broadcasting non-positive region granules list, thus it has the complexity of $\mathcal{O}(|U| * |U/IND(A)|)$. In MapReduce framework environment, the input decision system is partitioned and distributed to the nodes of the cluster where the workload is divided equally into p data partitions. Hence, the complexity of a data partition becomes $\mathcal{O}(\frac{|A|*|U|}{p})$.

In the worst-case scenario, the aforementioned theoretical time and space complexities of the proposed MRIDS_VP algorithm are described. However, because the MRIDS_VP algorithm incorporates positive region removal and granular refinement features, the actual time and space complexities are significantly reduced.

4.6 Experimental analysis

In this section, the proposed algorithms are evaluated experimentally. The experimental set up is described in Section 4.6.1. Computational time analysis and performance evaluation are two concerned metrics for parallel/distributed algorithms. Since the proposed work is the first research of its kind on parallel attribute reduction in IDS, the experimental analysis is provided only between the proposed approaches. The comparative analysis of computational time and the results are reported in Section 4.6.2. In the experimental analysis, we focused mainly on the performance evaluation with various metrics such as *speedup*, *scaleup* and *sizeup* [13] of the proposed parallel algorithms. The results of performance evaluation are reported in Section 4.6.3. The relevance and limitations of the proposed algorithms are evaluated experimentally in Section 4.6.4.

4.6.1 Experimental setup

The experiments are performed on Apache Spark and they are carried out on a seven node cluster. In the cluster, one node is fixed as a driver (master) as well as a worker, and the rest are set as workers (slaves). The master node uses Intel (R) Xeon (R) Silver 4110 CPU @ 2.10GHz processor with 32 cores and 64 GB of main memory. All the worker nodes use Intel (R) Core (TM) i7-8700 CPU@3.20GHz processor with 12 cores and the main memory of 32 GB. All nodes run on Ubuntu 18.04 LTS operating system and they are connected via Ethernet (with 1000Mbps speed). Each node is installed with Java 1.8.0_171, Apache Spark 2.3.1, and Scala 2.11.4.

Table 4.3: Data sets used in the experiments of MRIDS_HP and MRIDS_VP algorithms

Data set	Rename	Objects	Attributes	Classes	Missing %
Genes	genes-S801-A5000k-one	801	5009564	5	1
	genes-S801-A5000k-two	801	5009564	5	2
	genes-S801-A5000k-three	801	5009564	5	3
Gisette	gisette-S54k-A55k-one	54000	55000	2	1
	gisette-S54k-A55k-two	54000	55000	2	2
	gisette-S54k-A55k-three	54000	55000	2	3
Mushroom	mushroom-S40k-A40k-one	40620	40590	2	1
	mushroom-S40k-A40k-two	40620	40590	2	2
	mushroom-S40k-A40k-three	40620	40590	2	3
	mushroom-S80k-A40k-one	81240	40590	2	1
	mushroom-S120k-A40k-one	121860	40590	2	1
	mushroom-S40k-A80k-one	40620	81180	2	1
	mushroom-S40k-A120k-one	40620	121770	2	1
KDDcup	kdd-S4900k-A40-one	4898431	41	23	1
	kdd-S4900k-A40-two	4898431	41	23	2
	kdd-S4900k-A40-three	4898431	41	23	3

For the experiments, we have chosen four data sets from machine learning data repository UCI [31]. They are "Gene expression Cancer RNA-Seq (Genes)," "Gisette," "Mushroom," and "KDDcup 99 (KDDcup)" data sets. The original "Genes" data set contains 801 objects, one decision attribute, and 20531 conditional attributes. The "Gisette" data set contains 6000 objects, one decision attribute and 5000 conditional attributes. The "Mushroom" consists of 8124 objects, one decision attribute, and 22 conditional attributes. And, the original "KDDcup" data set contains 4898431 objects, 40 conditional attributes and one decision attribute. The object space and attribute space of the original data sets (except KDDcup) are replicated several times to check the efficiency of the proposed algorithms on the data sets with huge number of objects, and on big dimensional data sets. For example the conditional attribute space of the "Genes" data set is replicated 244 times (i.e., $20531 \times 244 = 5009564$) by keeping the object space constant to create an big dimensional data set. We name this data set as "genes-S801-A5000k".

From each of these data sets, three incomplete data sets are generated by randomly setting the missing values of 1%, 2% and 3% size of $|U| \times |A|$, and they are named accordingly with a suffix "one," "two" and "three" respectively. Therefore, each data set is categorised in terms of its incompleteness percentage, and then the experiments are conducted. The details of the various data sets after the replication with different incompleteness in the data are given in Table 4.3. Note that, in the table a name "gisette-S54k-A55k-two" indicates Gisette data set

Table 4.4: Running time (seconds) and reduct size of MRIDS_HP and MRIDS_VP for varying incompleteness percentage in the data sets

Data set	MRIDS_HP		MRIDS_VP	
	Running time	Reduct size	Running time	Reduct size
mushroom-S40k-A40k-one	421.18	05	519.67	05
mushroom-S40k-A40k-two	502.91	05	564.23	05
mushroom-S40k-A40k-three	597.18	05	591.63	05
gisette-S54k-A55k-one	2839.11	18	2891	18
gisette-S54k-A55k-two	3744.09	18	3127.51	18
gisette-S54k-A55k-three	3938.50	18	3463.62	18
genes-S801-A5000k-one	76276.71	07	205.63	07
genes-S801-A5000k-two	78532.82	07	213.87	07
genes-S801-A5000k-three	85647.13	07	224.56	07
kdd-S4900k-A41-one	1860.11	35	148050.32	35
kdd-S4900k-A41-two	7664.05	35	151720.26	35
kdd-S4900k-A41-three	61780.53	35	154110.41	35

having 54000 objects and 55000 attributes with 2 % missing values. All the data sets with different sizes are chosen according to limited hardware configuration of the cluster.

4.6.2 Computational evaluation

The reduct is computed for the data sets given in Table 4.3 using MRIDS_HP and MRIDS_VP algorithms. And the results of each algorithm are reported separately in Table 4.4. Computational time and reduct length of the obtained reduct for each data set with different percentages of incompleteness in the data are reported.

From the results in Table 4.4, it can be noticed that the different data sets of the "Mushroom" and "Gisette" have almost equal number of objects and attributes. For these data sets, both the proposed algorithms produced the reducts in almost similar computational times. It can also be observed that, when the data sets have massive attribute space (e.g., "Genes"), the MRIDS_VP algorithm performs well. Whereas, when the data sets have massive object space (e.g., "KDDcup"), the MRIDS_HP algorithm performs well. In contrast the computational times incurred by MRIDS_HP for big dimensional data sets, MRIDS_VP for massive object space data sets are so huge which establishes the need for proposing two approaches with different partitioning strategies.

By comparing the results of both algorithms, we can observe that MRIDS_VP has produced the reduct on almost similar time frame for all the incomplete percentages of the data. But the MRIDS_HP algorithm incurred more computational time when the incompleteness

percentage is increased. That is, the increase of the missing values in the data has more influence on the computational time of horizontal partitioning based MRIDS_HP algorithm than the vertical partitioning based MRIDS_VP algorithm. The reason is that the complexity in shuffle and sort phase and $mapValues()$ increases with the increase in incompleteness in the data sets. The resilience of the MRIDS_VP algorithm for increase in the incompleteness is majorly due to the simplified shuffle and sort phase which is independent of the incompleteness percentage, and iteration wise incorporation of granular refinement and positive region removal features.

For the reproducible research, the reducts and their γ_A and γ_R values obtained for different original data sets (the data sets without replication) are reported in Table 4.5. From the results, it can be noticed that the increase in incompleteness percentage leads to reduction in the γ_A and γ_R values.

4.6.3 Performance evaluation

The performance of the proposed parallel algorithms is evaluated concerning speedup, scaleup and sizeup metrics. The "Gisette" data sets with all the percentages of incompleteness (gisette-S54k-A55k-one, -two, and -three) are used to find the performance metrics of both algorithms.

4.6.3.1 Speedup evaluation

The speedup of the proposed algorithms have been evaluated on the data sets with different cores ranging from 20 to 100. Figure. 4.1 shows the speedup results of the "Gisette" data set with different percentages of incompleteness in the data.

From Figure. 4.1, it can be observed that speedups obtained by both algorithms are almost close to each other. By observing plots, it is also clear that with an increase in the percentage of incompleteness in the data, the speedup performance of MRIDS_VP is better than the MRIDS_HP algorithm. The reason why the MRIDS_VP algorithm is better is because of its granular refinement feature. And, another reason is that the significant amount of computations in MRIDS_VP occur in the mapper phase for best attribute selection in each iteration. The computations in the associated reducer phase are very less. This simplified synchronization barrier of reducer and parallelizability of mappers through horizontal scaling produces better speedup to MRIDS_VP than the MRIDS_HP algorithm.

Table 4.5: Reduct obtained by MRIDS_HP and MRIDS_VP algorithms for varying incompleteness percentage in the data sets

	MRIDS_HP			MRIDS_VP		
Data set	Reduct	γ_A	γ_R	Reduct	γ_A	γ_R
Genes-one	{6521, 16215, 14771, 11528, 133, 10526, 33, 30003, 18500}	0.99	0.99	{6521, 16215, 14771, 11528, 15867, 3859, 18500}	0.99	0.99
Genes-two	{5588, 17485, 16327, 15863, 18493, 16341, 538, 16501}	0.98	0.98	{5588, 17485, 16327, 15863, 18493, 16341, 15776, 14684}	0.98	0.98
Genes-three	{16898, 15864, 16915, 17925, 4223, 2278, 6602, 12778, 18400}	0.98	0.98	{16898, 15864, 16915, 17925, 4223, 2278, 20086, 15360}	0.98	0.98
Gisette-one	{1995, 1851, 155, 558, 2092, 1466, 4833, 3976, 1392, 4694, 1229, 3328, 4739, 3606, 3828}	1.00	1.00	{1995, 1851, 155, 4879, 3328, 2555, 1664, 3976, 1129, 67, 819, 1392, 3354, 4694, 2394, 4739, 3606, 3828}	1.00	1.00
Gisette-two	{2015, 1080, 3544, 4554, 140, 617, 1559, 3198, 2223, 215, 1568, 3354, 4467, 1445, 4694, 840, 4375, 4502}	1.00	1.00	{2015, 1080, 3544, 4554, 1559, 3085, 3198, 3976, 1607, 2223, 215, 1568, 4467, 4694, 840, 3670, 4937, 4502 }	1.00	1.00
Gisette-three	{902, 3001, 1533, 1329, 3966, 2743, 1600, 4950, 4617, 3354, 4571, 4694, 2893, 3446, 4739, 3502, 4813, 456 }	1.00	1.00	{902, 3001, 1533, 1329, 4990, 1226, 3415, 3966, 456, 4272, 107, 2367, 3354, 2893, 4694, 3446, 3502, 4813 }	1.00	1.00
Mushroom-one	{4, 19, 7, 11, 20}	1.00	1.00	{5, 20, 8, 11, 4}	1.00	1.00
Mushroom-two	{5, 20, 19, 8, 11}	1.00	1.00	{7, 20, 11, 8, 5}	1.00	1.00
Mushroom-three	{4, 19, 11, 7, 20}	1.00	1.00	{4, 19, 5, 20, 8}	1.00	1.00
KDDcup-one	{1, 2, 34, 3, 33, 22, 4, 29, 10, 5, 31, 23, 35, 32, 36, 37, 30, 12, 39, 13, 6, 8, 16, 19, 28, 26, 40, 17, 38, 14, 24, 11, 18, 7, 9}	0.98	0.98	{1, 2, 34, 3, 33, 22, 4, 29, 10, 5, 31, 23, 35, 32, 36, 37, 30, 12, 39, 13, 6, 8, 16, 19, 28, 26, 40, 17, 38, 14, 24, 11, 18, 15, 20}	0.98	0.98
KDDcup-two	{1, 2, 34, 3, 33, 22, 4, 29, 10, 5, 31, 23, 35, 32, 36, 37, 30, 12, 39, 13, 6, 8, 16, 19, 28, 26, 40, 17, 38, 14, 24, 11, 18, 7, 9}	0.96	0.96	{1, 2, 34, 3, 33, 22, 4, 29, 10, 5, 31, 23, 35, 32, 36, 37, 30, 12, 39, 13, 6, 8, 16, 19, 28, 26, 40, 17, 38, 14, 24, 11, 18, 7, 41}	0.96	0.96
KDDcup-three	{1, 2, 34, 3, 33, 22, 4, 29, 10, 5, 31, 23, 35, 32, 36, 37, 30, 12, 39, 13, 6, 8, 16, 19, 28, 26, 40, 17, 38, 14, 24, 11, 18, 7, 9}	0.96	0.96	{1, 2, 34, 3, 33, 22, 4, 29, 10, 5, 31, 23, 35, 32, 36, 37, 30, 12, 39, 13, 6, 8, 16, 19, 28, 26, 40, 17, 38, 14, 24, 11, 18, 41, 9}	0.96	0.96

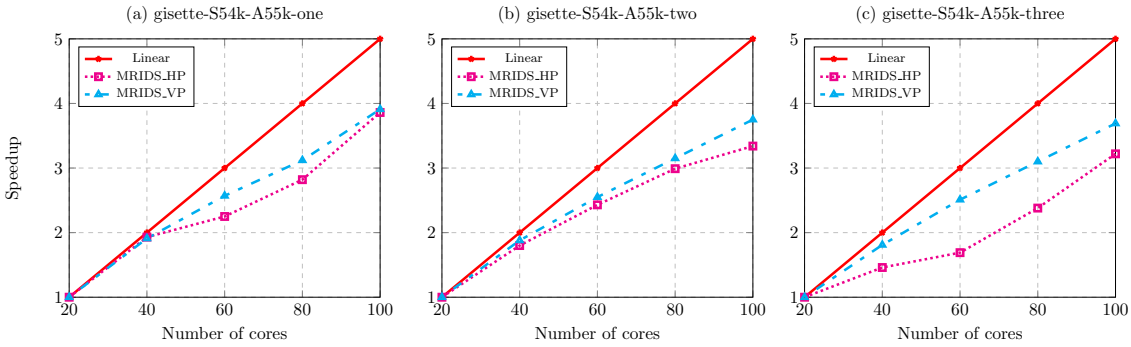


Figure 4.1: Speedup of MRIDS_HP and MRIDS_VP for Gisette data set with different percentages of incompleteness

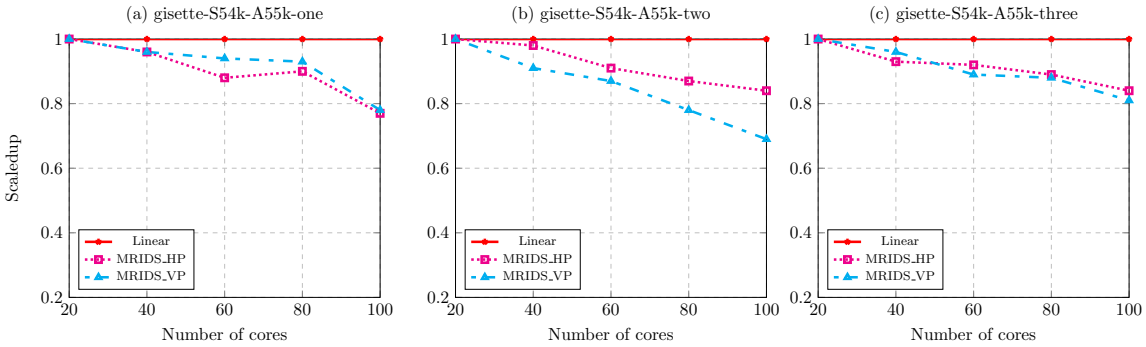


Figure 4.2: Scaleup of MRIDS_HP and MRIDS_VP for Gisette data set with different percentages of incompleteness

4.6.3.2 Scaleup evaluation

To find the scaleup performance of the proposed algorithms,, the data set size is increased in proportion to the number of cores in the cluster. Each data set is divided into 20%, 40%, 60%, 80% and 100% sizes of original data set (divided in object space), and the number of cores in the cluster increased from 20, 40, 60, 80 and 100 respectively. Figure. 4.2 shows the scaleup results of the "Gisette" data set with different percentages of incompleteness in the data.

The higher scaleup value shows the better performance of the algorithms. From the results shown in Figure. 4.2, it can be observed that, in each data set, the scaleup value of MRIDS_HP becomes better while the object space of the data set increases. In contrast, the scaleup value of MRIDS_VP is better, when the object space in the data set is small in size (e.g., 20%, 40%). However, the scaleup values of both algorithms are higher than 0.7, that indicates the proposed algorithms scale well for different percentages of incompleteness in the data.

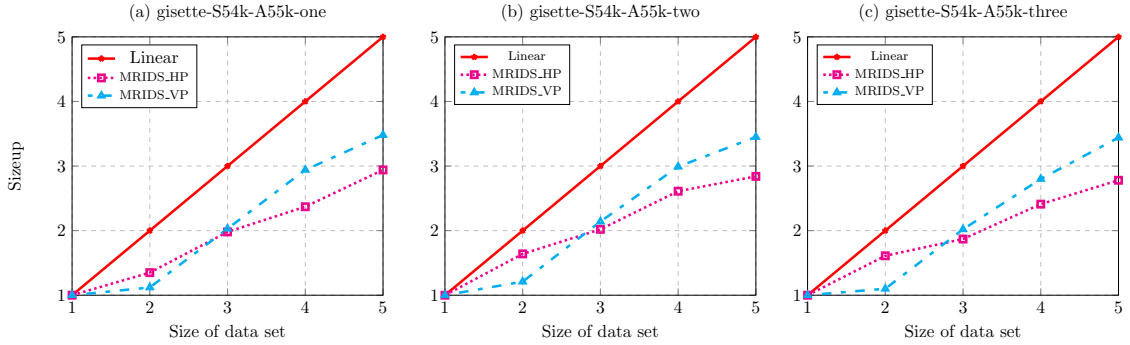


Figure 4.3: Sizeup of MRIDS_HP and MRIDS_VP for Gisette data set with different percentages of incompleteness

4.6.3.3 Sizeup evaluation

To find the sizeup of the proposed algorithms, we changed the size of data set by keeping the number of nodes constant. The number of nodes kept are seven, and the object space of the data set is increased in the order of 20%, 40%, 60%, 80%, and 100%. Figure. 4.3 shows the sizeup results of the "Gisette" data set with different percentages of incompleteness in the data.

From Figure. 4.3, it is observed that the vertical partitioning based MRIDS_HP algorithm obtained better sizeup results than MRIDS_VP algorithm. This is because in the MRIDS_VP algorithm, the computational load in mappers is increasing with the increase in the object space of the data set. In addition, the computational load in the mapper phase of the next best attribute selection in each iteration is significantly more than the associated reducer phase. Hence the sizeup results for MRIDS_HP are expected to be better, and the same is observed in the results obtained. Figure. 4.3 also shows that both proposed algorithms produce better sizeup results, as their plots are much lower than the linear plots in the figures for all data sets.

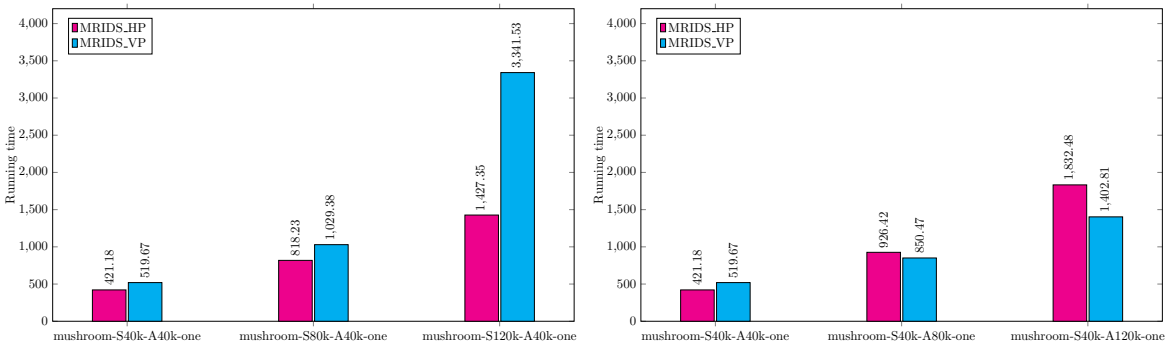
4.6.4 Impact of the partitioning strategy

An experiment is conducted to validate the relevance and limitations of the proposed algorithms. In this experiment, the "Mushroom" data set which has almost the same number of objects and attributes with one percentage of incompleteness in the data is considered. The first row of Table 4.6 shows this data set. We have increased the size of object space by keeping the attribute space constant while conducting the experiments. The results are reported in the 2nd and 3rd rows of the Table 4.6. Similarly, the size of the attribute space is increased by keeping the object space constant. The results are reported in the 4th and 5th

rows of the Table 4.6. The behavior of both the algorithms with the variation in the size in object space and attribute space is plotted in Figure. 4.4a and 4.4b respectively.

Table 4.6: Comparison of MRIDS_HP and MRIDS_VP with varying object and attribute space of Mushroom (Time: Seconds)

S.No	Data set	MRIDS_HP Running time	MRIDS_VP Running time
1.	mushroom-S40k-A40k-one	421.18	519.67
2.	mushroom-S80k-A40k-one	818.23	1029.38
3.	mushroom-S120k-A40k-one	1427.35	3341.53
4.	mushroom-S40k-A80k-one	926.42	850.47
5.	mushroom-S40k-A120k-one	1832.48	1402.81



(a) Scalability in the object space

(b) Scalability in the attribute space

Figure 4.4: Behavior of MRIDS_HP and MRIDS_VP for varying object space and attribute space of Mushroom

From the results, it is clear that if the size of the attribute space increases, then the MRIDS_HP algorithm computed the reduct by incurring more computational times than the MRIDS_VP algorithm. Because the partitioning strategy (horizontal partitioning) of the algorithm allows enormous data movement between the nodes of the cluster when attribute space increases. And also this strategy does not allow incorporation of the granular refinement aspect. But, if the size of the object space in the data set is increased, then the vertical partitioning based MRIDS_VP algorithm incurred a considerable amount of computational time than horizontal partitioning based MRIDS_HP algorithm. Because with the increase in object space, the serial computation in the mapper phase within a local node increases. Even though the algorithm has advantages like granular refinement and simplified shuffle and sort phase, the serial computation degrades the performance of the MRIDS_VP algorithm. From the aforementioned analysis, it is established that the horizontal partitioning based MRIDS_HP algorithm is scalable and ideal for the data sets having huge number of objects

and moderate number of attributes and it is not recommended for data sets with big dimensionality. Likewise, the vertical partitioning based MRIDS_VP algorithm is efficient and scalable for the big dimensional data sets with moderate number of objects.

4.7 Summary

In this chapter, MapReduce based parallel/distributed attribute reduction approaches are investigated for large-scale incomplete decision systems. Both the approaches adopted the Novel Granular Framework for handling the complexity involved in dealing incompleteness in the data. The MRIDS_HP algorithm was developed based on the horizontal partitioning strategy by adopting alternative representation of existing NGF. And the MRIDS_VP algorithm was developed based on the vertical partitioning strategy. With extensive experimental analysis and theoretical validation, the proposed MRIDS_HP algorithm has been proven to be efficient and more suitable for the incomplete data sets with massive number of objects and moderate number of attributes. Similarly, the MRIDS_VP algorithm has been shown to be effective and ideal for the big dimensional data sets having modest object space. The computational and performance evaluation demonstrated that the proposed methods are efficient in attribute reduction even if we have huge number of missing values in the data.

This chapter discussed MapReduce based attribute reduction in IDS, as well as the implications of horizontal and vertical partitioning strategy. In the next chapter (Chapter 5), we explore MapReduce based attribute reduction approaches for hybrid decision systems (HDS), where we investigate the relevance of both partitioning strategies.

Chapter 5

Parallel attribute reduction in Hybrid Decision Systems

In this chapter, third and fourth contributions to this thesis work are discussed. This chapter introduces a fuzzy discernibility matrix-based attribute reduction accelerator (DARA) to accelerate the attribute reduction in hybrid decision systems (HDS). The accelerator DARA is used to build a sequential approach and the corresponding MapReduce based parallel/distributed approaches for attribute reduction in large hybrid data sets. The proposed MapReduce based approaches follow horizontal and vertical partitioning strategies to handle the data sets that are large in terms of number of objects and attributes respectively. The experimental study, along with theoretical validation show that the proposed approaches are effective and perform better than the existing state-of-the-art approaches. The work presented in this chapter is derived from the article published in [101], and the manuscript to be submitted to a reputed journal.

5.1 Review of existing approaches

The classical rough set model uses crisp equivalence classes in attribute reduction. As a consequence, it is only applicable to perform the attribute reduction in categorical data sets. For attribute reduction in numerical data sets, this classical model requires the discretization of numerical attributes. Discretization, however, causes loss of information [49, 52]. And therefore, classical rough sets are restricted to Categorical Decision Systems (CDS). But the hybrid decision systems are more frequently occurring data sets in real-time applications. The decision system with different types of attributes (e.g., categorical, numerical, boolean,...etc.) is known as Hybrid Decision System (HDS). Therefore, various fuzzy-rough set models [19, 32, 86, 117] have been proposed to handle different types of attributes in attribute reduction.

5. PARALLEL ATTRIBUTE REDUCTION IN HYBRID DECISION SYSTEMS

Dubois and Prade [32] proposed a fuzzy-rough set model and the generalised version of this model is given by Radzikowska et al. [86]. Fuzzy-rough set models overcome the limitations of classical rough sets in being applicable to hybrid data [20, 75, 130].

In classical rough set theory, reduct computation is primarily done in two approaches: *dependency measure* approach and *discernibility matrix* approach [53]. These methods are generalised to the fuzzy-rough set model, and a number of fuzzy-rough attribute reduction algorithms [20, 22, 52, 75, 110, 111, 122, 127, 128, 130] are proposed. In addition to these algorithms, various approaches have been proposed to further improve the efficiency of fuzzy-rough attribute reduction [14, 54, 72, 85, 91].

Sai Prasad et al. [91] improved the MQRA (Modified Quick Reduct Algorithm) [20] to IMQRA (Improved MQRA) by incorporating a simplified computational model, and by absolute positive region removal. In [54], Jensen et al. developed the nnFRFS (nearest neighbour Fuzzy Rough Feature Selection) algorithm and the nnFDM (nearest neighbour Fuzzy Discernibility Matrix) algorithm for scalable fuzzy-rough feature selection. In nnFRFS, fuzzy-rough membership degree is determined only for the objects that are closest neighbours. Thus the number of calculations in the algorithm is greatly decreased. Similarly, in nnFDM, the matrix is constructed only for the objects that are nearest neighbours. Jinkun Chen et al. [14] developed an approach for fuzzy-rough attribute reduction based on graphs. They demonstrated in this approach that the attribute reduction is equivalent to finding the minimal traversal of a derivative hyper graph. Qian et al. [85] developed an accelerator called forward approximation (FA-FPR) to improve the process of fuzzy-rough attribute reduction. The experimental findings have shown that the FA-FPR is much faster than its predecessors. Later, Peng Ni et al. [72] developed a positive region based attribute reduction accelerator (PARA) that outperformed FA-FPR. The accelerator PARA is developed by removing object pairs that have been discerned in the process of attribute reduction. Through carefully studying all these existing methods, it can be observed that each approach accelerates the attribute reduction process by ignoring or removing the objects that are no longer useful or cause redundant computations. Notice that all of these current methods are sequential approaches. Even with the accelerators, the memory requirements of attribute reduction in fuzzy-rough sets restrict the applicability to small data sets, requiring parallel/distributed solutions.

From the review of literature, it is found that there are currently a few parallel/distributed approaches available in the literature for fuzzy-rough attribute reduction in large numeric data sets [8, 28, 46, 56, 76]. Pavani et al. [76] developed an algorithm MR_FRDM_SBE based on the discernibility matrix. This algorithm uses sequential backward elimination

(SBE) strategy for reduct generation. Kiran et al. [8] developed an algorithm MR_IMQRA based on dependency measure approach that uses a vertical partitioning of the input data to the nodes of the cluster. The algorithm MR_IMQRA has been found to be more effective for larger attribute space data sets with moderate object space (i.e., big dimensional data sets). Qinghua Hu et al. [46] proposed an approach for hybrid data, where they developed a MapReduce based multi modality attribute reduction method based on multi kernel fuzzy rough sets model. This method has a presumption that entire data set to be available at each node of the cluster, and in every partition, computation with respect to a subset of objects is involved. This assumption of availability of data set at each node could hinder the scalability of the approach. L. Kong et al. [56], developed a distributed fuzzy rough set (DFRS) method for attribute reduction in cloud computing. The DFRS methodology involves parallel computation of reducts on overlapping subsets of given data set, and union of the individual solutions obtained becomes the solution of the approach. This approach has the advantage of avoiding intermediate data transfer across the nodes but has overheads with respect to computations over overlapping subsets. W. Ding et al. [28] developed a Multi granulation Consensus Fuzzy-Rough Attribute Reduction Algorithm (MCFR). This algorithm is capable of handling granular and structurally-complex large attributes to find the attribute reduction sets. It can be noticed that the approaches in [8, 46, 76] are MapReduce based, and the approaches in [28, 56] are non-MapReduce based parallel/distributed approaches.

As mentioned in Chapter 3 and 4, any MapReduce framework uses *horizontal partitioning strategy* to partition the object space of the input data set to the cluster. The *vertical partitioning* is an alternative strategy that partitions the input data set in attribute space. From the literature, it is observed that, all the researchers developed MapReduce based methods in fuzzy-rough set theory [28, 46, 56, 76] (except the approach in [8]) using horizontal partitioning. The approach in [8] uses vertical partitioning strategy, and it is dependency measure based approach. From the extensive study of the literature and from the findings of Chapter 3 and 4, it is observed that, horizontal partitioning based approaches are scalable in larger object space data sets while vertical partitioning based approaches are scalable in larger attribute space data sets (data sets with big dimensionality).

Except a few discernibility matrix based methods [54, 76], most of the scalable methods either sequential [72, 85, 91] or parallel/distributed [8, 28, 46, 56] are developed based on the dependency measure approach. As specified in [53, 54, 85], in the dependency measure based approach of attribute reduction, for each attribute, a similarity matrix is constructed that contains the similarity measure of each pair of objects in the data set. Furthermore, in each iteration, the similarity matrices should be constructed for different subsets of attributes. If a

data set contains $|U|$ objects and $|A|$ conditional attributes, then the memory utilization for constructing similarity matrices is $\mathcal{O}(|A| * |U|^2)$. In the discernibility matrix based approach, a matrix is constructed for each pair of objects in the data set. And each entry in the matrix contains discernible value for each attribute between a pair of objects. Since the discernibility matrix is symmetric, only the upper diagonal or lower diagonal entries are computed. And the entries are computed between the objects that are from different decision classes. Because of the above reasons, the memory utilization of the discernibility matrix based approach is at most $\mathcal{O}(|A| * ((|U|^2/2) - |U|))$. As a result, a substantial decrease in memory usage is achieved in the discernibility matrix based approach relative to the dependency measure based approach. The advantages of discernibility matrix over dependency measure, and also non availability of discernibility matrix based accelerators in the literature inspired us to investigate accelerators and the corresponding parallel/distributed approaches based on discernibility matrix.

In this chapter, a discernibility matrix based accelerator for scalable fuzzy-rough attribute reduction is proposed. This accelerator is used to build a sequential approach and the corresponding MapReduce based parallel/distributed approaches using horizontal and vertical partitioning strategies. In summary, the contributions in this chapter include the following.

1) A fuzzy discernibility matrix based attribute reduction accelerator (DARA) is introduced. Based on this accelerator, a sequential algorithm IFDMFS (Improved Fuzzy Discernibility Matrix based Feature Selection) is developed for scalable attribute reduction in HDS.

2) The following MapReduce based parallel/distributed versions of IFDMFS are also developed to further improve the scalability.

   - MR_IFDMFS algorithm is developed using horizontal partitioning strategy.
   - MR_VFDMFS algorithm is developed using vertical partitioning strategy.

The merits and limitations of the sequential and parallel/distributed approaches are proved through extensive experimental analysis along with theoretical validation.

5.2 Related work

In this section, the basics of fuzzy-rough set model are provided. Since the proposed approaches are based on fuzzy discernibility matrix (FDM), the principles of FDM based attribute reduction are discussed.

5.2.1 Fuzzy-rough set theory

In this section, the principles of fuzzy-rough set theory are discussed on the basis of [19, 32, 52, 86]. Let $HDS = (U, A \cup \{d\})$ be a hybrid decision system, where, U denotes the set of objects, A denotes the set of hybrid conditional attributes, and $\{d\}$ is the categorical decision attribute. The core principle for the attribute reduction using classical rough set model is the *indiscernibility relation* [77], which is an equivalence relation. The idea of indiscernibility relation is generalised by using a *fuzzy similarity relation* [32] in fuzzy-rough set model.

Definition 5.1. For a given hybrid decision system HDS, a fuzzy similarity relation SIM_a is a fuzzy relation on U using the knowledge of the attribute $a \in A$. The similarity relation satisfies the following conditions.

1) Reflexivity: $\forall x \in U, \mu_{SIM_a}(x, x) = 1$

2) Symmetry: $\forall x, x' \in U, \mu_{SIM_a}(x, x') = \mu_{SIM_a}(x', x)$

3) T-transitivity: $\forall x, y, z \in U,\ \mu_{SIM_a}(x, z) \geq T(\mu_{SIM_a}(x, y), \mu_{SIM_a}(y, z))$

Here, $\mu_{SIM_a}(x, x')$ denotes the similarity measure between the objects x and x' of the numeric attribute a. And, T is a fuzzy $T-norm$ [18] which is an associative aggregation operator $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$. Note that if the fuzzy similarity relation meets the first and second conditions (Reflexivity and Symmetry), the relation is assumed to be a *fuzzy tolerance relation*. And if the relation satisfies all three conditions, the relation is considered a *fuzzy T-equivalence relation*.

The similarity measure of an object pair of the fuzzy similarity relation can be computed by using various methods. In the proposed approaches, the fuzzy similarity measure is calculated using the following method [52].

$$\mu_{SIM_a}(x, x') = max\left(min\left(\frac{a(x) - a(x') + \sigma(a)}{\sigma(a)}, \frac{a(x') - a(x) + \sigma(a)}{\sigma(a)}\right), 0\right) \tag{5.1}$$

Here, $\mu_{SIM_a}(x, x')$ is the measure of degree to which the objects x and x' are similar for numerical attribute a, and $\mu_{SIM_a}(x, x') \in [0, 1]$. The notation $a(x)$ denotes the value of the object x for attribute a. And, the notation $\sigma(a)$ represents the standard deviation of the attribute a. If an attribute a is categorical (qualitative), the classical indiscernibility relation is adopted and the $\mu_{SIM_a}(x, x')$ measure is given as follows.

$$\mu_{SIM_a}(x, x') = \begin{cases} 0 & if\ a(x) \neq a(x') \\ 1 & if\ a(x) = a(x') \end{cases} \tag{5.2}$$

For a subset of attributes $P \subseteq A$, the fuzzy similarity relation is expanded by using the $T - norm$ (T) specified below.

$$\mu_{SIM_P}(x, x') = \underset{a \in P}{T}(\mu_{SIM_a}(x, x')), \forall x, x' \in U \tag{5.3}$$

As Eq. (5.1) or (5.2) are used to find the similarity measure between a pair of objects of a numerical attribute or categorical attribute, similarly different methods [113, 122] are used to find similarity measure of different types of attributes such as boolean, set-valued, ...etc. Note that, the proposed approaches in this chapter mainly focus on HDS having numerical and categorical attributes, because these HDSs are more frequently occurring decision systems in real-time applications. However, the proposed approaches are applicable to other types of attributes in given HDS.

5.2.2 Fuzzy discernibility matrix based attribute reduction

Three approaches to fuzzy-rough attribute reduction were proposed by Jensen et al. [52]. The approaches are: (i) *Fuzzy Lower approximation based Feature Selection (FLFS)*, (ii) *Fuzzy Boundary region-based Feature Selection (FBFS)*, and (iii) *Fuzzy Discernibility Matrix-based Feature Selection (FDMFS)*. This section provides the details of the FDMFS approach with its algorithm. Table 5.1 shows the tiny decision system, used to illustrate the basic concepts and the proposed approaches.

5.2.2.1 Fuzzy discernibility

The discernibility relation is determined with the complement to the indiscernibility relation in the classical rough set theory model. In the same way, the discernibility relation $(DIS_a(x, x'))$ in fuzzy-rough set model is obtained by performing fuzzy negation on the fuzzy similarity between the two objects x and x' as given below.

$$\mu_{DIS_a}(x, x') = N(\mu_{SIM_a}(x, x')) \tag{5.4}$$

Here, N represents the fuzzy negation, $\mu_{SIM_a}(x, x')$ is fuzzy similarity measure and $\mu_{DIS_a}(x, x')$ is fuzzy dissimilarity (fuzzy discernibility) measure between the objects x and x' based on the knowledge of attribute a. In the proposed works and in literature [20, 52], the *standard fuzzy negation* is considered for N and is given below.

$$N(\mu_{SIM_a}(x, x')) = 1 - \mu_{SIM_a}(x, x') \tag{5.5}$$

From the above equation, the fuzzy discernibility measure $\mu_{DIS_a}(x, x') \in [0, 1]$.

Table 5.1: An example decision system

	Attributes					
Objects	a	b	c	d	e	f
x_1	3.59	3.52	1.86	0.76	6.30	1
x_2	1.97	4.31	1.57	0.54	3.69	2
x_3	1.51	0.85	7.54	2.31	2.09	2
x_4	2.16	0.50	3.80	1.79	2.12	1
x_5	3.73	1.17	5.68	4.34	3.37	1

Example 5.1. For the decision system given in Table 5.1, $\mu_{SIM_b}(x_1, x_2) = 0.54$ is the fuzzy similarity measure based on Eq. (5.1), by using Eq. (5.4) the fuzzy discernibility measure is given as $\mu_{DIS_b}(x_1, x_2) = 1 - 0.54 = 0.46$.

5.2.2.2 Fuzzy Discernibility Matrix (FDM)

The fuzzy discernibility relation is represented in a fuzzy discernibility matrix (FDM). Each entry (clause) in the matrix is a vector that includes a fuzzy discernibility measure of each attribute of A. The FDM contains the entries between the objects of different decision classes. And the rest of the entries are empty. Therefore, the matrix is a *decision relative fuzzy discernibility matrix*. In the matrix, an entry $C_{xx'}$ between the objects $x, x' \in U$ is given as,

$$C_{xx'} = \begin{cases} < v_1, v_2, ..v_i, ..v_{|A|} >, & if\ d(x) \neq d(x') \\ \emptyset & otherwise \end{cases} \tag{5.6}$$

Here, in an entry $C_{xx'}$ each value $v_i = \mu_{DIS_i}(x, x'), \forall i \in A$.

Example 5.2. From the decision system given in Table 5.1, using Equations 5.1 and 5.4, the fuzzy discernibility measures for all the conditional attributes between the objects x_1 and x_2 are computed as, $\mu_{DIS_a}(x_1, x_2) = 1.0$, $\mu_{DIS_b}(x_1, x_2) = 0.46$, $\mu_{DIS_c}(x_1, x_2) = 0.11$, $\mu_{DIS_d}(x_1, x_2) = 0.14$ and $\mu_{DIS_e}(x_1, x_2) = 1.0$. From Eq. (5.6), an entry $C_{x_1x_2}$ in FDM is represented as $C_{x_1x_2} =< 1.0, 0.46, 0.11, 0.14, 1.0 >$.

The discernibility measure for a subset of attributes $P \subseteq A$ is calculated from discernibility measure of each individual attribute using the following definition.

Definition 5.2. In a given decision system HDS, the discernibility measure or *satisfiability* (SAT) for a subset of attributes $P \subseteq A$ in an entry $C_{xx'}$ of FDM is given by,

$$SAT_P(C_{xx'}) = \bigcup_{a \in P} \{\mu_{DIS_a}(x, x')\} \tag{5.7}$$

In Eq. (5.7), the fuzzy union ($\bigcup$) is computed by using a specified fuzzy $S-norm$ ($T-conorm$) [52]. Here, $S-norm$ is an aggregation operator $\perp : [0,1] \times [0,1] \rightarrow [0,1]$, and for any fuzzy values of p, q, r, and t, it satisfies the following conditions [52].

1) Identity: $\perp(p,0) = \perp(0,p) = p$

2) Commutativity: $\perp(p,q) = \perp(q,p)$

3) Associativity: $\perp(p, \perp(q,r)) = \perp(\perp(p,q), r)$

4) Monotonicity: $\perp(p,q) \leq \perp(r,t)\ if\ p \leq r\ and\ q \leq t$

In the implementation of the proposed algorithms, we use the *Lukasiewicz T-conorm* or *S-norm* ($\perp(x,y) = min(1, x+y)$) [52] to evaluate the discernibility measure between the set of attributes as given in Eq. (5.7).

Example 5.3. By continuing Example 5.2, if we select subset $P = \{b, c\}$, the resultant satisfiability of the entry $C_{x_1x_2}$ is $SAT_P(C_{x_1x_2}) = \perp(0.46, 0.11) = min(1, 0.46 + 0.11) = 0.57$.

In the FDM, the satisfiability of all the entries for a subset of attributes $P \subseteq A$ is computed as,

$$SAT(P) = \frac{\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_P(C_{xx'})}{\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_A(C_{xx'})} \tag{5.8}$$

By using Eq. (5.8), the fuzzy-rough reduct is defined as given below.

Definition 5.3. For a given decision system HDS, the fuzzy-rough reduct R is a minimal subset of the conditional attribute set A ($R \subseteq A$) such that,

- $SAT(R) = SAT(A)$ (*jointly sufficient*)
- $SAT(R') < SAT(A)\ for\ any\ R' \subset R$ (*individually necessary*)

Thus, the attributes set $R \subseteq A$ is said to be fuzzy-rough reduct, if and only if R is a minimal subset of A satisfying $SAT(R) = 1$.

5.2.2.3 Reduct computation using FDM (FDMFS algorithm)

In [52], the authors provided the methodology to compute the reduct by using an FDM. It is referred to as FDMFS (FDM based Feature Selection). Here, we are providing the methodology in the form of an algorithm given in Algorithm 5.1. The FDMFS algorithm starts with the initialization of the current reduct R to the empty set ($\emptyset$). At each iteration of the algorithm, the best attribute a^{best} to be added to the reduct R is computed by using the following equation derived from Eq. (5.8).

Algorithm 5.1: FDMFS algorithm

Input: 1. Fuzzy discernibility matrix: FDM
2. S-norm: $\perp$

Output: Reduct R

1 $R = \emptyset$, $SAT(R) = 0.0$ `// Initialize reduct and its satisfiability`
2 $SAT_A = \sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_A(C_{xx'})$
3 **repeat**
4 $Temp = R$
5 **for** *each* $a \in (A - R)$ **do**
6 Compute $SAT(R \cup \{a\}) = \frac{\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_{(R \cup \{a\})}(C_{xx'})}{SAT_A}$
7 **if** *(*$SAT(R \cup \{a\}) > SAT(Temp)$*)* **then**
8 $Temp = R \cup \{a\}$
9 $SAT(Temp) = SAT(R \cup \{a\})$
10 **end**
11 **end**
12 $R = Temp$
13 **until** *(*$SAT(R) == 1$*)*
14 return R

$$SAT(R \cup \{a^{best}\}) = \max_{a \in (A-R)} (SAT(R \cup \{a\})) \tag{5.9}$$

The denominator value of $SAT(R \cup \{a\})$ is a normalising factor that is calculated as SAT_A and shown in second line of Algorithm 5.1. The best attribute a^{best} is determined using Eq. (5.9) and added to the reduct set R. This procedure is repeated until $SAT(R)$ becomes 1. When the $SAT(R)$ reaches 1, the algorithm returns reduct set R and terminates.

5.3 Proposed Discernibility matrix based Attribute Reduction Accelerator (DARA)

The idea behind introducing DARA is explored in this section. The *SAT-region removal* which is the main feature of DARA is presented. A sequential IFDMFS (Improved Fuzzy Discernibility Matrix-based Feature Selection) algorithm is proposed based on the DARA. The IFDMFS is an improved version of FDMFS [52].

5.3.1 Motivation

From the analysis of literature, it can be observed that the number of objects contained in the data set prevent the scalability of the fuzzy-rough attribute reduction in large data sets. From the algorithms of [72, 85, 91], it is noticed that, in each iteration of the reduct

computation, the algorithms removed the objects which are no longer useful in future computations. Therefore, the basic principle of any accelerator of fuzzy-rough attribute reduction is to remove the redundant objects in the data set. In the proposed works, the removal of the redundant objects is done in two stages. In the first stage, by constructing a decision relative FDM, the computations are restricted to the objects which are from different decision classes. In the second stage, we incorporate the SAT-region removal feature (given in Section 5.3.2) into the proposed algorithm. This feature acts as an accelerator, and is called DARA. This accelerator limit the computations to the subset of entries of FDM in each iteration of reduct computation to avoid the redundant computations. The following theorem explores the redundant computations involved in an iteration of the existing algorithm to select an attribute to the reduct set.

Theorem 5.1. *In a given decision system* $HDS = (U, A \cup \{d\})$*, consider an attribute set* $R \subseteq A$*, if* $C_{xx'}$ *is an entry of FDM for which* $SAT_R(C_{xx'}) = SAT_A(C_{xx'})$ *is satisfied, then* $\forall R' \supseteq R$*,*

$$SAT_{R'}(C_{xx'}) = SAT_R(C_{xx'})$$

Where SAT value of an entry $C_{xx'}$ *is computed using a given fuzzy* $S - norm : \perp$*.*

Proof.
For any two fuzzy values p, q, where $0 \leq p \leq 1$ and $0 \leq q \leq 1$, from identity property of $S - norm$, we have $\perp(p, 0) = p$, and $\perp(0, q) = q$. Similarly, from monotonicity property of $S - norm$, we have $\perp(p, q) \leq \perp(r, t)$ whenever $p \leq r$, $q \leq t$ for any $0 \leq r, t \leq 1$. Using identity property $\perp(0, q) = q$, and as $0 \leq p$, $q \leq q$, we have $q = \perp(0, q) \leq \perp(p, q)$,

$$\therefore q \leq \perp(p, q) \tag{5.10}$$

Similarly,

$$p \leq \perp(p, q) \tag{5.11}$$

From Eq. 5.7, for an entry $C_{xx'}$ in the discernibility matrix, the satisfiability of a subset of attributes $R \subseteq A$ is given by,

$$SAT_R(C_{xx'}) = \bigcup_{att \in R} \{\mu_{DIS_{att}}(C_{xx'})\}$$

For any attribute $a \in (A - R)$, let R' be $R \cup \{a\}$ then

$$\begin{aligned} SAT_{R\cup\{a\}}(C_{xx'}) &= \bigcup_{att\in(R\cup\{a\})} \{\mu_{DIS_{att}}(C_{xx'})\} \\ &= \underset{att\in(R\cup\{a\})}{\perp}(\mu_{DIS_{att}}(C_{xx'})) \\ &= \perp(\underset{att\in R}{\perp}(\mu_{DIS_{att}}(C_{xx'})), \mu_{DIS_a}(C_{xx'})) \\ &= \perp(SAT_R(C_{xx'}), \mu_{DIS_a}(C_{xx'})) \end{aligned}$$

From Eqs. (5.10) and (5.11),

$$\begin{aligned} SAT_R(C_{xx'}) &\leq \perp(SAT_R(C_{xx'}), \mu_{DIS_a}(C_{xx'})) \\ &= SAT_{R\cup\{a\}}(C_{xx'}) \\ &= SAT_{R'}(C_{xx'}) \end{aligned}$$

For any $R' \supseteq R$, the same argument can be successfully applied for each addition of attribute in $R' - R$. For any $R' \supseteq R$, we have,

$$SAT_R(C_{xx'}) \leq SAT_{R'}(C_{xx'})$$

And, since $R \subseteq R' \subseteq A$, we have,

$$SAT_R(C_{xx'}) \leq SAT_{R'}(C_{xx'}) \leq SAT_A(C_{xx'})$$

But it is given that, $SAT_R(C_{xx'}) = SAT_A(C_{xx'})$

$$\therefore SAT_R(C_{xx'}) = SAT_{R'}(C_{xx'})$$

Hence proved. □

5.3.2 SAT-region removal as an accelerator

From Theorem 5.1, it is established that if an entry $(C_{xx'})$ reaches its maximum SAT value (i.e., $SAT_R(C_{xx'}) = SAT_A(C_{xx'})$) for the attributes subset $R \subseteq A$, then calculating SAT value for the same entry in selecting the next attribute to R becomes redundant computation. Therefore, for a given set of attributes $R \subseteq A$, the FDM is divided into two non overlapping sets FDM_F(R) (FDM Fulfilled) and FDM_UF(R) (FDM Unfulfilled). And they are defined below.

Definition 5.4. In a given decision system HDS, let R be a set of attributes such that

$R \subseteq A$, and $FDM\_F(R)$ be a set, where it contains the entries of FDM as given below,

$$FDM\_F(R) = \{C_{xx'} \in FDM \mid SAT_R(C_{xx'}) = SAT_A(C_{xx'})\}$$

From the above equation, $FDM\_F(R)$ includes all the entries of FDM, which satisfy the condition $SAT_R(C_{xx'}) = SAT_A(C_{xx'})$. That is, $FDM\_F(R)$ contains the subset of entries of FDM, which have already reached the maximum SAT value.

Definition 5.5. In a given decision system HDS, let R be a set of attributes such that $R \subseteq A$. And, let $FDM\_UF(R)$ be a set that contains the entries of FDM as given below,

$$FDM\_UF(R) = \{C_{xx'} \in FDM \mid SAT_R(C_{xx'}) < SAT_A(C_{xx'})\}$$

Above equation states that, $FDM\_UF(R)$ includes all the entries of FDM, which strictly satisfy the condition $SAT_R(C_{xx'}) < SAT_A(C_{xx'})$. That is, $FDM\_UF(R)$ contains the subset of entries of FDM, which have not yet reached the maximum SAT value.

Theorem 5.2. *The selection of next best attribute a^{best} from $(A - R)$ into reduct R in algorithm FDMFS can be equivalently performed by restricting the computations to only the entries of $FDM\_UF(R)$.*

Proof.
In the FDMFS algorithm, the next best attribute a^{best} from $(A - R)$ is selected based on the criteria given below.

$$SAT(R \cup \{a^{best}\}) = \max_{a \in (A-R)} (SAT(R \cup \{a\}))$$

$$\text{Here, } SAT(R \cup \{a\}) = \frac{\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_{(R \cup \{a\})}(C_{xx'})}{\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_A(C_{xx'})}$$

$$= \frac{\begin{array}{c}\sum_{C_{xx'} \in FDM\_F(R) \wedge C_{xx'} \neq \emptyset} SAT_R(C_{xx'}) \\ + \sum_{C_{xx'} \in FDM\_UF(R) \wedge C_{xx'} \neq \emptyset} SAT_{(R \cup \{a\})}(C_{xx'})\end{array}}{\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_A(C_{xx'})}$$

$$(\because FDM = FDM\_F(R) \cup FDM\_UF(R))$$

It is observed that, the expression $\sum_{C_{xx'} \in FDM\_F(R) \wedge C_{xx'} \neq \emptyset} \{SAT_R(C_{xx'})\}$ is independent of $'a'$ and the expression $\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} \{SAT_A(C_{xx'})\}$ is constant for all the iterations. Therefore, for the next best attribute a^{best}, we have,

$$SAT(R \cup \{a^{best}\}) = \max_{a \in (A-R)} (SAT(R \cup \{a\}))$$
$$= \max_{a \in (A-R)} \left(\frac{\sum_{C_{xx'} \in FDM\_F(R) \wedge C_{xx'} \neq \emptyset} SAT_R(C_{xx'}) + \sum_{C_{xx'} \in FDM\_UF(R) \wedge C_{xx'} \neq \emptyset} SAT_{(R \cup \{a\})}(C_{xx'})}{\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_A(C_{xx'})} \right)$$

From the above equation, it can be noticed that, the expressions $\sum_{C_{xx'} \in FDM\_F(R) \wedge C_{xx'} \neq \emptyset} SAT_R(C_{xx'})$ and $\sum_{C_{xx'} \in FDM \wedge C_{xx'} \neq \emptyset} SAT_A(C_{xx'})$ are constants in computing $SAT(R \cup \{a\}),\ \forall a \in (A - R)$.
Hence, the next best attribute selection based on all the entries of FDM using Eq. (5.9) is same as the next best attribute a^{best} which achieves $\max_{a \in (A-R)} \left(\sum_{C_{xx'} \in FDM\_UF(R) \wedge C_{xx'} \neq \emptyset} SAT_{(R \cup \{a\})}(C_{xx'}) \right)$.
Therefore, the computations are performed only on $FDM\_UF(R)$ entries and not on all the entries of FDM. □

It is obvious from Theorem 5.2 that in the proposed algorithm, the computations are carried out only on the entries of $FDM\_UF(R)$ in each iteration of the reduct computation. In other words, the Eq. (5.9) in the existing FDMFS algorithm is updated as given below to compute the next best attribute a^{best}.

$$SATUF(R \cup \{a^{best}\}) = \max_{a \in (A-R)} (SATUF(R \cup \{a\})) \tag{5.12}$$

Here $SATUF(R \cup \{a\}) = \sum_{C_{xx'} \in FDM\_UF(R) \wedge C_{xx'} \neq \emptyset} SAT_{(R \cup \{a\})}(C_{xx'})$. The SAT-region removal feature is derived from Theorem 5.2, and is defined below.

Definition 5.6. For a given decision system HDS, in the FDMFS algorithm, let initial reduct $R = \phi$, $FDM\_F(R) = \phi$, and initial $FDM\_UF(R) = FDM$. Then after adding an attribute to the reduct set R at i^{th} iteration of the algorithm, the SAT-region removal is given by,

$$FDM\_UF(R^{i+1}) = FDM\_UF(R^i) - FDM\_F(R^{i+1})$$

Since $FDM\_F(R)$ includes the entries that have reached maximum SAT value, the set of entries of $FDM\_F(R)$ is known to be SAT-region. Thus, the removal of the entries of $FDM\_F(R)$ from $FDM\_UF(R)$ is referred to as SAT-region removal. For each iteration of the proposed algorithm, the removal of the SAT-region is done such that the reduct computation is accelerated. Therefore, SAT-region removal serves as an accelerator, and as it is based

on the discernibility matrix, is referred to as DARA. The proposed IFDMFS algorithm given in Algorithm 5.2 incorporates DARA.

5.3.3 IFDMFS algorithm

The proposed sequential IFDMFS algorithm is developed based on the following Corollary 5.1, which is derived from Theorem 5.2.

Corollary 5.1. *For a given decision system* $HDS = (U, A \cup \{d\})$*, if* $R^0 = \phi \subseteq R^1 \subseteq R^2 ... \subseteq R^i \subseteq ... \subseteq R^n \subseteq A$*, where* R^i *is the set of attributes selected into reduct* R *by the* i^{th} *iteration of the proposed algorithm, which computes the final reduct* R^n *in* n *iterations, then the fuzzy discernibility matrix*

$$FDM = FDM\_UF(R^0) \supseteq FDM\_UF(R^1) \supseteq ... \supseteq FDM\_UF(R^i) \supseteq ... \supseteq FDM\_UF(R^n) = \phi$$

Procedure of IFDMFS algorithm is given in Algorithm 5.2. Initially the FDM for the given decision system is constructed based on the procedure given in Section 5.2.2.2. In the process of reduct computation, IFDMFS algorithm starts its first iteration by initializing the reduct R as an empty set ($\emptyset$), and the satisfiability value is initialized as $SATUF(R) = 0.0$. The set which contains the entries with maximum SAT value is initialized to $FDM\_F(R) = \emptyset$, and the set which has the entries that have not yet reached maximum SAT is $FDM\_UF(R) = FDM$.

The next best attribute a^{best} is computed by using Eq. (5.12), and the attribute is added to the reduct set R. The entries which have reached maximum SAT value are added to the set $FDM\_F(R)$, and the SAT-region removal is performed. In the subsequent iteration, the next best attribute is computed by using only the entries of $FDM\_UF(R)$ after the SAT-region removal process is completed. This procedure is repeated until $FDM\_UF(R)$ becomes empty ($\emptyset$). From Corollary 5.1, it is to be noted that, once $FDM\_UF(R)$ becomes empty ($\emptyset$), then $FDM\_F(R)$ set contains all the entries of FDM. That is, all the entries of FDM have fulfilled the satisfiability, and $SAT(R) = 1$ (i.e., $SAT(R) = SAT(A)$). Hence the algorithm returns

the reduct R, and terminates.

Algorithm 5.2: Sequential IFDMFS algorithm

Input: 1. Input file: data set $HDS = (U, A \cup \{d\})$
2. Fuzzy similarity relation: SIM, Fuzzy negation: N, and S-norm: $\perp$

Output: Reduct R

1 Construct FDM for HDS `// Using the procedure given in Section 5.3.2`
2 $R = \emptyset$, $SATUF(R) = 0.0$ `// Initialize reduct and its satisfiability`
3 $FDM\_F(R) = \emptyset$, $FDM\_UF(R) = FDM$
4 **repeat**
`/* ====== Phase 1: Computation of the best attribute ====== */`
5 $Temp = R$
6 **for** *each* $a \in (A - R)$ **do**
7 Compute $SATUF(R \cup \{a\}) = \sum_{C_{xx'} \in FDM\_UF(R) \wedge C_{xx'} \neq \emptyset} SATUF_{(R \cup \{a\})}(C_{xx'})$
8 **if** $(SATUF(R \cup \{a\}) > SATUF(Temp))$ **then**
9 $Temp = R \cup \{a\}$
10 $SATUF(Temp) = SATUF(R \cup \{a\})$
11 **end**
12 **end**
13 $R = Temp$
`/* ============= Phase 2: SAT-region removal ============= */`
14 **for** *each* $C_{xx'} \in FDM\_UF(R)$ **do**
15 **if** $(SATUF_R(C_{xx'}) == SATUF_A(C_{xx'}))$ **then**
16 $FDM\_F(R) = FDM\_F(R) \cup C_{xx'}$
17 **end**
18 **end**
19 $FDM\_UF(R) = FDM\_UF(R) - FDM\_F(R)$
20 **until** $(FDM\_UF(R) == \emptyset)$
21 return R

5.3.3.1 Complexity analysis of IFDMFS algorithm

For a given decision system HDS, let $|U|$ denotes the number of objects and $|A|$ denotes the number of conditional attributes. If the IFDMFS algorithm gets FDM as the input, then in the worst case (when the reduct set R is equal to all the attributes set A), the algorithm has to perform $\mathcal{O}(|A|^2)$ number of SAT evaluations for finding the reduct. And these evaluations are performed against the FDM of size $\mathcal{O}(|U|^2)$. Thus, the complexity of the proposed IFDMFS algorithm becomes $\mathcal{O}((|A|^2 * |U|^2)$. But the practical time and space complexities of the algorithm are much smaller, since the FDM is symmetric and decision relative. In other words, the construction of the FDM is done either for lower diagonal or upper diagonal entries and are formed only for the objects that are belonging to different decision classes. Thus if the decision attribute has n decision classes with the cardinalities of

$c_1, c_2, ...c_n$, then the number of entries in the FDM are reduced to $E = \sum_{i=1}^{n-1} c_i * (\sum_{j=i+1}^{n} c_j)$ which is much smaller than $|U|^2$. In addition, the size of the FDM is reduced with DARA in each iteration that contributes to reduction in the time and space complexities as the iterations of the algorithm progress. Hence the theoretical time and space complexities of IFDMFS are $\mathcal{O}(|A|^2 * E)$ and $\mathcal{O}(|A| * E)$ respectively, whereas the exact time complexity is much smaller.

5.3.3.2 Illustrative example

This example is based on the decision system shown in Table 5.1 and is meant to illustrate how the IFDMFS algorithm works. For the given decision system, the FDM must be built on the basis of the fuzzy discernibility shown in Eq. (5.4) using the standard fuzzy negation in Eq. (5.5) and the fuzzy similarity in Eq. (5.1). Each entry $C_{xx'}$ of the FDM is formed on the basis of Eq. (5.6). The computed entries of FDM are given below.

$C_{x_1x_2} :< 1.0, 0.46, 0.11, 0.14, 1.0 >$, $C_{x_1x_3} :< 1.0, 1.0, 1.0, 1.0, 1.0 >$,

$C_{x_2x_4} :< 0.19, 1.0, 0.88, 0.82, 0.91 >$, $C_{x_2x_5} :< 0.64, 0.20, 1.0, 0.34, 0.02 >$,

$C_{x_3x_4} :< 1.0, 1.0, 1.0, 1.0, 0.19 >$, $C_{x_3x_5} :< 1.0, 0.18, 0.73, 1.0, 0.74 >$

Initially reduct $R = \emptyset$, satisfiability of reduct $SATUF(R) = 0.0$, $FDM\_F(R) = \emptyset$, and $FDM\_UF(R) = \{C_{x_1x_2}, C_{x_1x_3}, C_{x_2x_4}, C_{x_2x_5}, C_{x_3x_4}, C_{x_3x_5}\}$. Based on Eq. (5.7), the $SATUF_A(C_{xx'})$ value is computed, where $SATUF_A(C_{xx'}) = \bigcup_{a \in A}\{\mu_{DIS_a}(C_{xx'})\}$. The computed values are given below.

$SATUF_A(C_{x_1x_2}) = 1.0$, $SATUF_A(C_{x_1x_3}) = 1.0$,

$SATUF_A(C_{x_2x_4}) = 1.0$, $SATUF_A(C_{x_2x_5}) = 1.0$,

$SATUF_A(C_{x_3x_4}) = 1.0$, $SATUF_A(C_{x_3x_5}) = 1.0$

Based on Eq. (5.8), the $SATUF(A)$ value is computed, and it is given as $SATUF(A) = 6.0$

First iteration:

Using the line number: 7 in Algorithm 5.2, the satisfiability values of individual attributes for all the entries in the FDM are computed and given below.

$SATUF(\{a\}) = 4.83$, $SATUF(\{b\}) = 3.84$, $SATUF(\{c\}) = 4.72$,

$SATUF(\{d\}) = 4.30$, and $SATUF(\{e\}) = 3.86$

Based on Eq. (5.12), the best attribute $a^{best} = \{a\}$ is selected and added to reduct set, thus $R = \{a\}$. Note that, here $R = \{a\}$, and $A = \{a, b, c, d, e\}$. Using Eq. (5.7), the $SATUF_R(C_{xx'})$ value is computed,

$$SATUF_R(C_{x_1x_2}) = 1.0, \qquad SATUF_R(C_{x_1x_3}) = 1.0,$$

$$SATUF_R(C_{x_2x_4}) = 0.19, \qquad SATUF_R(C_{x_2x_5}) = 0.64,$$

$$SATUF_R(C_{x_3x_4}) = 1.0, \qquad SATUF_R(C_{x_3x_5}) = 1.0$$

It can be noticed that, $SATUF_R(C_{x_1x_2}) = SATUF_A(C_{x_1x_2})$, $SATUF_R(C_{x_1x_3}) = SATUF_A(C_{x_1x_3})$, $SATUF_R(C_{x_3x_4}) = SATUF_A(C_{x_3x_4})$ and $SATUF_R(C_{x_3x_5}) = SATUF_A(C_{x_3x_5})$. Hence, the entries $\{C_{x_1x_2}, C_{x_1x_3}, C_{x_3x_4}, C_{x_3x_5}\}$ go into SAT-region, because these entries satisfy the condition in line number: 15 of Algorithm 5.2. After removal of SAT-region from $FDM\_UF(R)$, we get $FDM\_UF(R) = \{C_{x_2x_4}, C_{x_2x_5}\}$, and
$FDM\_F(R) = \{C_{x_1x_2}, C_{x_1x_3}, C_{x_3x_4}, C_{x_3x_5}\}$.

Second iteration:

After the first iteration, algorithm gets $R = \{a\}$, $FDM\_UF(R) = \{C_{x_2x_4}, C_{x_2x_5}\}$, and $FDM\_F(R) = \{C_{x_1x_2}, C_{x_1x_3}, C_{x_3x_4}, C_{x_3x_5}\}$. Now, the entries of $FDM\_F(R)$ in FDM are ignored and the entries of $FDM\_UF(R) = \{C_{x_2x_4}, C_{x_2x_5}\}$ are considered in the computation of the next best attribute a^{best}. So that, the redundant computations on $FDM\_F(R)$ entries are avoided which leads to a lot of reduction in computations. This reduction in computations in each iteration accelerate the attribute reduction process. Now, the satisfiability values of attributes of $(A - R)$ for all the entries in the $FDM\_UF(R)$ are computed and given below.

$$SATUF(R \cup \{b\}) = 1.85,\ SATUF(R \cup \{c\}) = 2.0,$$

$$SATUF(R \cup \{d\}) = 1.98, \text{ and } SATUF(R \cup \{e\}) = 1.66$$

The best attribute $a^{best} = \{c\}$ is selected and added to reduct set, thus $R = \{a, c\}$. And, $SATUF_R(C_{x_2x_4}) = 1.0$ and $SATUF_R(C_{x_2x_5}) = 1.0$. The entries $C_{x_2x_4}$ and $C_{x_2x_5}$ satisfy the condition in line number: 15, hence they go into SAT-region, and are removed from $FDM\_UF(R)$ and added to $FDM\_F(R)$. Now, the set $FDM\_UF(R)$ becomes empty ($\emptyset$), then algorithm returns reduct set $R = \{a, c\}$ and terminates.

5.4 Parallel attribute reduction in HDS using horizontal partitioning

With horizontal partitioning strategy, the input data set HDS is partitioned in object space by MapReduce framework, and the data partitions are distributed to the nodes of the cluster as defined below.

Definition 5.7. For a given decision system HDS, let $HDS = \bigcup_{i=1}^{p} HDS^i$, where $HDS^i = (U^i, A \cup \{d\})$ is i^{th} data partition and satisfies (i) $U = \bigcup_{i=1}^{p} U^i$, (ii) $U^i \cap U^j = \emptyset$, $\forall i, j \in \{1, 2, ...p\}$ *and* $i \neq j$, where p is the number of data partitions.

Here, each data partition which is a sub-table of the form $HDS^i = (U^i, A \cup \{d\})$ is assigned to a mapper in the cluster.

The fuzzy-rough attribute reduction in this MapReduce based approach is performed in two steps. In the first step, DFDM (Distributed FDM) is constructed, and in the second step, the reduct is computed using DFDM. In this section, both the steps are discussed in detail. The equivalence of the reduct generated by both sequential (IFDMFS) and parallel (MR_IFDMFS) algorithms is discussed at the end of this section.

5.4.1 Distributed Fuzzy Discernibility Matrix (DFDM)

In [76], a MapReduce based parallel MR_FRDM_SBE algorithm is developed, which utilizes the DFDM in reduct computation. The procedure for the construction of DFDM in [76] is formulated as Algorithm 5.3 and Algorithm 5.4 for completeness and readability of the proposed work.

The complexity involved in the construction of DFDM for large data sets is handled in two steps. In the first step, as the discernibility matrix is symmetric, we restrict the creation of DFDM for lower diagonal entries. And in the second step, since the discernibility matrix is decision relative, we can only compute the entry $C_{xx'}$ (clause) of DFDM for the pair of objects x and x' which are from different decision classes.

The algorithm to construct the DFDM is given in Algorithm 5.3, and it is written in pseudo-Spark's API. Along with input data set HDS, the algorithm also requires the inputs of fuzzy negation (N), fuzzy similarity relation (SIM), and $S-norm$. Here $S-norm$ is used to find discernibility value for multiple attributes. From the input data set HDS_RDD, the objects of a particular decision class are filtered into $newHDS\_dclass$ and they are removed from HDS_RDD. Depending on the broadcast size, the objects of $newHDS\_dclass$ are broadcasted as bulk or in chunks. All these broadcasted objects are compared with other decision class objects in HDS_RDD using $mapPartitions()$ to form the new entries for

Algorithm 5.3: Construction of DFDM

Input: 1. Input data set: $HDS = (U, A \cup \{d\})$
2. Fuzzy similarity relation: SIM, Fuzzy negation: N, and S-norm: $\perp$

Output: Distributed Fuzzy Discernibility Matrix (DFDM) as RDD

`/* Read the input data set as an RDD, where the data set` HDS `is distributed to the nodes with horizontal partitioning and the decision attribute` $\{d\}$ `is broadcasted, such that each data partition becomes` $HDS^i = (U^i, A \cup \{d\})$, $\forall i \in \{1, 2, ...p\}$, `where` p `is the number of data partitions in the cluster.` `*/`

1 $HDS\_RDD = readASRDD(HDS)$
2 $DFDM = \emptyset$ `// Initialize DFDM as empty`
3 **for** *each* $decisionVal \in decisionClasses$ **do**
4 $\quad newHDS\_dclass = HDS\_RDD.filter(decisionVal == d).collect()$
5 $\quad HDS\_RDD = HDS\_RDD.filter(decisionVal! = d)$
6 $\quad Broadcast(newHDS\_dclass)$
7
8 $\quad dmRDD = HDS\_RDD.mapPartitions(data => \{$
9 $\quad dMat = \emptyset$
10 $\quad$ **for** $x = data$ **do**
11 $\quad\quad$ **for** $x' = newHDS\_dclass$ **do**
12 $\quad\quad\quad dmEntry = DFDMEntry(x, x', N, SIM, \perp)$
13 $\quad\quad$ **end**
14 $\quad$ **end**
15 $\quad$ **if** $dmEntry.maxDissVal \neq 0$ **then**
16 $\quad\quad dMat = dMat.union(dmEntry)$
17 $\quad$ **end**
18 $\quad dMat\ \})$
19
20 $\quad DFDM = DFDM.union(dmRDD)$
21 **end**
22 return $DFDM$

Algorithm 5.4: Computation of DFDM entry: *DFDMEntry*

Input: 1. x and x' are two objects in HDS of different decision classes.
2. Fuzzy negation: N, Fuzzy similarity relation: SIM, and S-norm: $\perp$

Output: $entry =< DissValues, maxDissVal >$

1 $entry.DissValues = N(\mu_{SIM_a}(x, x')),\ \forall a \in A$
2 $entry.maxDissVal = \perp(entry.DissValues)$
3 return $entry$

DFDM. The formation of new entry is done by using Eq. (5.1), (5.2) and (5.5). Each entry is inserted into $dmRDD$, and then this $dmRDD$ is finally added to RDD of $DFDM$ using a *union* operation. This above procedure is repeated to construct DFDM for all decision classes, except for the final decision class.

From Algorithm 5.4, it can be observed that, each entry has two parts: $DissValues$ and $maxDissVal$. The variable $DissValues$ represents the vector of fuzzy dissimilarity values of the entry $C_{xx'}$ of x and x' objects over all conditional attributes A. And, the variable $maxDissVal$ represents $SAT_A(C_{xx'})$ value in Eq. (5.7).

5.4.2 Parallel reduct computation using DFDM

After the construction of DFDM, the next step in fuzzy-rough attribute reduction is the computation of reduct using DFDM. Like in the sequential approach, in the parallel approach also, the reduct is computed in two phases: (i) computation of the best attribute, and (ii) SAT-region removal. The MapReduce based parallel/distributed algorithms for reduct computation are given in the form of a driver (master), mapper, and reducer. The driver algorithm MR_IFDMFS: driver() is given in Algorithm 5.5, the mapper algorithm MR_IFDMFS: map() is given in Algorithm 5.6, and the reducer algorithm MR_IFDMFS: reduce() is given in Algorithm 5.7. All the algorithms are written using pseudo-Spark's API for better readability. Since the proposed method is a parallel/distributed approach, the constructed DFDM is distributed to the nodes of the cluster as defined below.

Definition 5.8. For a given decision system HDS, let $DFDM$ denotes distributed fuzzy discernibility matrix, then $DFDM^i$ $\{i = 1, 2, ...p\}$ denotes a sub-DFDM, and satisfies (i) $DFDM = \bigcup_{i=1}^{p} DFDM^i$, (ii) $DFDM^i \cap DFDM^j = \emptyset$, where, $i, j = 1, 2, ...p$, here p is number of partitions and $i \neq j$.

Each $DFDM^i$ is also called as *DFDM-split*, and each split is given to a mapper located in a node of the cluster.

5.4.2.1 Computation of the best attribute

The driver (Algorithm 5.5) invokes Algorithm 5.3 to construct DFDM. The Algorithm 5.3 returns the DFDM as an RDD. The driver initializes the reduct R and $FDM\_F(R)$ to an empty set ($\emptyset$) and computes the best attribute by invoking the mapper (Algorithm 5.6), and reducer (Algorithm 5.7). As mentioned earlier, each mapper gets a DFDM-split ($DFDM^i$) as input along with fuzzy $S - norm : \perp$. As shown in Algorithm 5.6, from each record $C_{xx'}$ (an entry) of $DFDM^i$, a set of $< key, value >$ pairs are formed $\forall att \in (A - R)$. For each

Algorithm 5.5: MR_IFDMFS: driver()

Input: 1. Input file: data set $HDS = (U, A \cup \{d\})$
2. Fuzzy similarity relation: SIM, Fuzzy negation: N, and S-norm: $\perp$

Output: Reduct R

1 Distribute the input data set HDS into the nodes of the cluster such that each data partition becomes $HDS^i = (U^i, A \cup \{d\})$, $\forall i \in \{1, 2, ...p\}$, where p is the number of data partitions in the cluster.
2 Construct $DFDM$ as an RDD by invoking Algorithm 5.3
3 Initial reduct $R = \emptyset$, $FDM\_F(R) = \emptyset$
4 **repeat**
`/* ==== Phase 1: Computation of the best attribute ====  */`
5 Initiate map job by invoking Algorithm 5.6
6 $val\ SATUF_{(R\cup\{att\})}(C_{xx'})RDD = DFDM.mapPartitions(part => \{var\ mp = map()\})$
`/* Mapper returns collection of` $< key, value >$ `pairs for each entry` $C_{xx'}$ `of` $DFDM^i$`, where` $< key, value >=< att, SATUF_{(R\cup\{att\})}(C_{xx'}) >$`, here` att `is an attribute,` $SATUF_{(R\cup\{att\})}(C_{xx'})$ `is its satisfiability of entry` $C_{xx'}$ `*/`
7 Initiate reduce job by invoking Algorithm 5.7.
8 $val\ SATUF(R \cup \{attNo\})RDD = SATUF_{(R\cup\{att\})}(C_{xx'})RDD.reduceByKey((x, y) => \{var\ rp = reduce()\})$
`/* Reducer returns collection of` $< key, value >=< attNo, SATUF(R \cup \{attNo\}) >$ `pairs, where` $attNo$ `is attribute number, and` $SATUF(R \cup \{attNo\})$ `is satisfiability value of` $attNo$ `for all the entries of` $DFDM$`.  */`
9 Collect the attributes and their respective satisfiability values from the reducers. $var\ SATUF(R \cup \{attNo\}) = SATUF(R \cup \{attNo\})RDD.collect()$
10 Select the best attribute *bestAttNo* which gets maximum satisfiability value.
11 $R = R \cup bestAttNo$
`/* ============ Phase 2: SAT-region removal ===============  */`
12 Filter the entries of $DFDM$ into $FDM\_F(R)$ which satisfy $(SATUF_R(C_{xx'}) == SATUF_A(C_{xx'}))$ using map() only operation.
13 $DFDM = DFDM.filter(if(C_{xx'}\ not\ in\ FDM\_F(R)))$
14 **until** $(DFDM == \emptyset)$
15 return R

$att \in (A - R)$, a $< key, value >$ pair is generated, where key is the attribute identifier (att), and the $value$ is its satisfiability value ($SATUF_{(R \cup \{att\})}(C_{xx'})$). It should be noted that to compute the $SATUF_{(R \cup \{att\})}(C_{xx'})$ value, the mapper uses fuzzy $S - norm : \perp$.

Algorithm 5.6: MR_IFDMFS: map()

Input: 1. $DFDM^i$, each record of $DFDM^i$ read as $< key, value >=< entryNo, entryValues >$, where $entryNo$ represents $C_{xx'}$ and $entryValues$ represent vector of discernible values of all attributes.
2. S-norm: $\perp$

Output: List of $< key', value' >=< att, SATUF_{(R \cup \{att\})}(C_{xx'}) >$ pairs, here att is an attribute and $SATUF_{(R \cup \{att\})}(C_{xx'})$ is its satisfiability value of an entry $C_{xx'}$ of $DFDM^i$

```
for each C_xx' ∈ DFDM^i do
    for each att ∈ R do
        Compute ⊥_{C_xx'}(R)
    end
    for each att ∈ (A − R) do
        SATUF_{(R∪{att})}(C_xx') = ⊥(⊥_{C_xx'}(R), μ_{DIS_att}(C_xx'))
        Construct < key', value' >=< att, SATUF_{(R∪{att})}(C_xx') >
        Emit intermediate < key', value' >
    end
end
```

Algorithm 5.7: MR_IFDMFS: reduce()

Input: $< key, [V] >$, here, $key = attNo$ and $[V]$ is the list of satisfiability values received from the mappers

Output: $< key', value' >=< attNo, SATUF(R \cup \{attNo\}) >$

```
for each v ∈ V of key = attNo do
    SATUF(R ∪ {attNo}) = SATUF(R ∪ {attNo}) + v
end
Construct < key', value' >=< attNo, SATUF(R ∪ {attNo}) >
Emit < key', value' >
```

In Algorithm 5.7, each reducer gets a set of $< att, [SATUF_{(R \cup \{att\})}(C_{xx'})] >$ pairs as the input from all the mappers in the cluster, where $[SATUF_{(R \cup \{att\})}(C_{xx'})]$ represents the list of satisfiability values for attribute att. Based on the same key, the reducer adds the satisfiability values of each attribute received from the different mappers. This sum becomes $SATUF(R \cup \{att\})$ value, which is the satisfiability value of an attribute of all the entries in the matrix. Now, the reducer returns the $< key, value >$ pairs to the driver, where key is an attribute att, and $value$ is its $SATUF(R \cup \{att\})$ value. The driver collects all the attributes and their $SATUF(R \cup \{att\})$ values from all the reducers, and selects the best attribute ($bestAttNo$), which has the maximum $SATUF(R \cup \{att\})$ value (i.e., the attribute att, which satisfies $SATUF(R \cup \{att^{best}\}) = \max_{att \in (A-R)} (SATUF(R \cup \{att\}))$). This best attribute is

added to the reduct set R.

5.4.2.2 Parallel SAT-region removal

SAT-region removal in the proposed parallel approach is defined below.

Definition 5.9. For the given decision system HDS, let $HDS = \bigcup_{i=1}^{p} HDS^i$, where $HDS^i = (U^i, A \cup \{d\})$, and let $DFDM = \bigcup_{i=1}^{p} DFDM^i$, where each $DFDM^i$ is a DFDM-split. Let initial reduct $R = \phi$, $FDM\_F(R) = \phi$ in the proposed parallel algorithm, then after adding an attribute to R at each iteration of the algorithm, the SAT-region removal is given by,

$$DFDM = DFDM - FDM\_F(R)$$

After the computation of the best attribute, the SAT-region removal is incorporated in the second phase of the algorithm as given in Algorithm 5.5. All the entries of the $DFDM$, which satisfy the condition $(SATUF_R(C_{xx'}) = SATUF_A(C_{xx'}))$ are added to the set $FDM\_F(R)$. Now, the entries of $FDM\_F(R)$ are filtered out from the $DFDM$, this leads to SAT-region removal. In the driver algorithm, two phases are repeated until $DFDM$ becomes empty ($\emptyset$). If the $DFDM$ is empty, then no entries are left out in the matrix, the driver returns the reduct R, and the algorithm terminates.

Theorem 5.3. *The reduct generated by the parallel/distributed attribute reduction algorithm is same as the reduct produced by the corresponding sequential method.*

Proof. As mentioned in [53], attribute reduction involves three necessary steps: a subset of attributes generation, subset evaluation, and stopping criterion. The parallel/distributed algorithm and the corresponding sequential algorithm differ at the subset evaluation step of attribute reduction.

For the sequential method, let the decision system be $HDS = (U, A \cup \{d\})$, and fuzzy discernibility matrix be FDM. The corresponding decision system and distributed FDM for parallel/distributed method are given by $HDS = \bigcup_{i=1}^{p} HDS^i$, and $DFDM = \bigcup_{i=1}^{p} DFDM^i$ respectively, where p is the number of partitions. Both sequential and distributed methods differ in evaluating $SATUF(R \cup \{a^{best}\})$. From Eq. (5.12), for the sequential approach, we have,

$$SATUF(R \cup \{a^{best}\}) = \max_{a \in (A-R)} (SATUF(R \cup \{a\}))$$

Where,

$$SATUF(R \cup \{a\}) = \sum_{C_{xx'} \in FDM\_UF(R) \wedge C_{xx'} \neq \emptyset} SATUF_{(R \cup \{a\})}(C_{xx'})$$

In parallel approach, since the $DFDM$ is distributed to the different nodes of the cluster (i.e.,

$DFDM = \bigcup_{i=1}^{p} DFDM^i$), the above equation is expressed as given below.

$$SATUF(R \cup \{a\}) = \sum_{i=1}^{p} \left(\sum_{C_{xx'} \in DFDM^i \wedge C_{xx'} \neq \emptyset} SATUF_{(R \cup \{a\})}(C_{xx'}) \right)$$

Here, $DFDM^i$ is a DFDM-split, and it should be noted that $DFDM$ in the parallel approach is same as $FDM\_UF(R)$ in the sequential approach after the SAT-region is removed. Therefore, in the computation of $SATUF(R \cup \{a^{best}\})$, we have,

$$\max_{a \in (A-R)} \left(\sum_{C_{xx'} \in FDM\_UF(R) \wedge C_{xx'} \neq \emptyset} SATUF_{(R \cup \{a\})}(C_{xx'}) \right) =$$

$$\max_{a \in (A-R)} \left(\sum_{i=1}^{p} \left(\sum_{C_{xx'} \in DFDM^i \wedge C_{xx'} \neq \emptyset} SATUF_{(R \cup \{a\})}(C_{xx'}) \right) \right)$$

Hence, the reduct generated by both sequential and parallel approaches is the same. □

5.4.2.3 Complexity analysis of MR_IFDMFS algorithm

In the time complexity analysis of MR_IFDMFS algorithm, the following variables are used.

- $|U|$: the number of objects in the data set
- $|A|$: the number of conditional attributes in the data set
- p: the number of processors
- t_w: the number of time units to transfer one word of memory
- s: the number of time units to complete the synchronization
- q: the number of reducers
- $E = \sum_{i=1}^{n-1} c_i * (\sum_{j=i+1}^{n} c_j)$ (refer Section 5.3.3.1)

Table 5.2 shows the time complexities of each step of the phase in the MR_IFDMFS algorithm for one iteration. Note that, from the table, all the steps in the mapper and reducer are repeated until ($DFDM == \emptyset$) condition is satisfied in the driver. That is, these steps are repeated $|A|$ (in worst case) times. Hence, by adding up all the complexities, the total time complexity of the proposed MR_IFDMFS algorithm is obtained as given below.

Table 5.2: Time complexity analysis of MR_IFDMFS algorithm

Algorithm (phase)	Step* in Algorithm	Time complexity
Driver → (Algorithm 5.5)	1. Partitioning the data horizontally	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert}{p} * t_w)$
	2. Construct DFDM	$\mathcal{O}(\frac{\lvert A\rvert*E}{p}) + \mathcal{O}((\lvert A\rvert * \lvert U\rvert) * t_w)$
Mapper → (Algorithm 5.6)	1-10. Finding $SAT_{R\cup\{att\}}(C_{xx'})$, $\forall att \in (A-R)$	$\mathcal{O}(\frac{\lvert A\rvert*E}{p})$
	Barrier synchronization	$\mathcal{O}(s)$
Shuffle and sort →	Transferring all attributes and their SAT values of entries of DFDM	$\mathcal{O}((\lvert A\rvert * p) * t_w)$
Reducer → (Algorithm 5.7)	1-5. Find $SAT(R\cup\{att\})$ and Barrier synchronization	$\mathcal{O}(\frac{p*\lvert A\rvert}{q}) + \mathcal{O}(s)$

* Step denotes the line number in the associated algorithm

$$((\frac{|A| * |U|}{p} * t_w) + (\frac{|A| * E}{p}) + ((|A| * |U|) * t_w) + |A| * \left((\frac{|A| * E}{p}) + (s) + ((|A| * p) * t_w) + (\frac{p * |A|}{q}) + (s) \right) \quad (5.13)$$

Above equation can be approximated as: $\mathcal{O}(\frac{|A|^2*E}{p}) + \mathcal{O}(|A| * ((\frac{|A|*p}{q}) * t_w + s))$. Since the time complexity of the sequential IFDMFS algorithm is $\mathcal{O}(|A|^2 * E)$, this is an anticipated outcome for the proposed MR_IFDMFS algorithm in addition with its communication cost: $\mathcal{O}(|A| * ((\frac{|A|*p}{q}) * t_w + s)$. Thus, the time complexity of parallel MR_IFDMFS algorithm is reduced p times than its sequential counterpart in addition with communication overhead.

The entire DFDM is required for reduct computation using MR_IFDMFS algorithm. Thus, the space complexity of MR_IFDMFS algorithm is $\mathcal{O}(|A| * E)$. But, in MapReduce framework environment, the DFDM is partitioned and distributed to the nodes of the cluster where the workload is divided equally into p data partitions. Hence, each partition has the complexity of $\mathcal{O}(\frac{|A|*E}{p})$.

In the worst-case scenario, the aforementioned theoretical time and space complexity of the proposed MR_IFDMFS algorithm are described. However, because the MR_IFDMFS algorithm incorporates accelerator DARA (SAT-region removal), the actual time and space complexities are significantly reduced.

5.5 Parallel attribute reduction in HDS using vertical partitioning

In Chapter 3 and 4, it is demonstrated that horizontal partitioning based reduct computation approaches for CDS and IDS are suitable and scale well for the data sets having large object space and moderate attribute space. And, the vertical partitioning based approaches scale well for the data sets having large attribute space with moderate object space. Similarly, in this section, the relevance of vertical partitioning strategy is investigated for reduct computation in HDS. In this proposed MapReduce based approach, attribute reduction is performed in three steps: (i) Vertical partition of the input data set (ii) Parallel construction of the vertical FDM (vFDM) and (iii) Parallel attribute reduction using vFDM. This section describes these steps of the proposed approach.

5.5.1 Vertical partitioning of the input data set

The MapReduce based approach MR_IFDMFS given in Section 5.4 make use of horizontal partitioning strategy to partition the input data set. Since the information about all the objects is distributed around the cluster's nodes, in the construction of DFDM, broadcasting the input data is required to make data local to the nodes to form an entry of the matrix. Thus broadcasting avoids data shuffling in the network. However, if the data set is big dimensional, broadcasting chunks of data objects becomes complex, resulting in the approach being inefficient.

In the computation of reduct from DFDM, in each iteration, to compute satisfiability $SAT(R \cup \{attr\})$ of an attribute $attr \in (A - R)$ for all the entries in $DFDM$, the data movement is required to get all the entries together which are distributed across the nodes of the cluster. If the data set has larger attribute space, then a lot of data shuffling is required in evaluation of all the subsets of attributes of the data set to find the reduct. Thus, this data movement in *shuffle and sort phase* of the MapReduce framework becomes a bottleneck in fuzzy-rough attribute reduction of big dimensional hybrid decision systems.

In the proposed approach, we use alternative vertical partitioning strategy that avoids drawbacks of horizontal partitioning strategy. With this strategy, all the objects information of an attribute is available in one node. Since the complete data of an attribute is available at a location, broadcasting the data is not required for constructing FDM. And, as demonstrated in Chapter 3 and 4, this strategy avoids huge data movement in shuffle and sort phase for big dimensional data sets. The vertical partitioning strategy given in Section 3.3 of Chapter 3 is adopted to HDS as given in the following definition.

Definition 5.10. For the given decision table $HDS = (U, A \cup \{d\})$, let $HDS = \bigcup_{i=1}^{p} HDS^i$, where, $HDS^i = (U, A^i \cup \{d\})$ is i^{th} data partition, and satisfies (i) $A = \bigcup_{i=1}^{p} A^i$, (ii) $A^i \cap A^j = \emptyset,\ \forall i, j \in \{1, 2, ...p\}\ and\ i \neq j$, where p is the number of data partitions.

5.5.2 Parallel construction of the vertical FDM ($vFDM$)

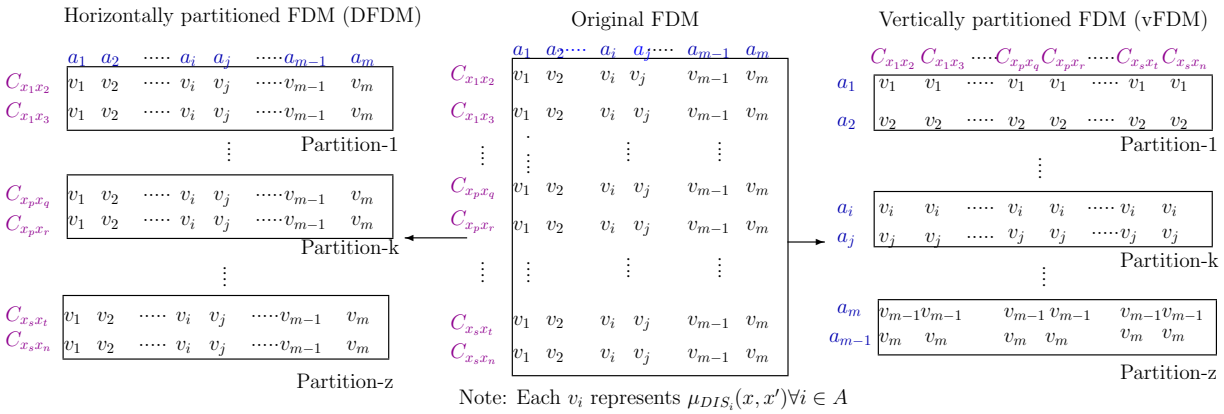


Figure 5.1: Horizontally and vertically partitioned FDM

An FDM is usually made of a row that corresponds to an entry $C_{xx'}$ (between two objects $(x, x') \in U$) and a column that corresponds to a discernible value for an attribute ($\mu_{DIS_{attr}}(x, x')$). Conversely, $vFDM$ is the transpose of an FDM, with each row corresponding to an attribute's discernibility value and each column corresponding to an entry. The original FDM and its horizontal ($DFDM$) and vertical ($vFDM$) forms are depicted in the Figure 5.1. In comparison to $DFDM$, the $vFDM$ has a lot of computational benefits, which are discussed in the subsequent sections. The process for computing vertical FDM ($vFDM$) is described by Algorithm 5.8. To facilitate better reading, the algorithm is presented in Apache Spark's pseudo-code.

The method described in Section 5.2.2.2 is used to construct $vFDM$. Vertically partitioned data simplifies $vFDM$ construction. Since a node contains all the information about the objects associated with an attribute ($attr$), the discernibility value is computed for each object pair, where the objects are from different decision classes. That is, each entry $C_{xx'}\ \forall (x, x') \in U$, associated with the attribute $attr$ is computed locally within a node in the cluster. Thus, within a node, each mapper computes a row of $vFDM$ for each attribute $attr \in A^i$. Each row in $vFDM$ includes an attribute identifier ($attr$) and its discernibility values for all the pairs of objects (i.e., $\mu_{DIS_{attr}}(x, x')\ \forall (x, x') \in U$) in the given data set. In Algorithm 5.8, these rows are computed in parallel by mappers by using $mapPartitions()$ method in Spark. Each mapper returns $< key, value >=< attr, \mu_{DIS_{attr}} >$ pairs, where

$\mu_{DIS_{attr}}$ denotes the list of discernibility values of all the entries (object pairs) for the attribute $attr$.

Algorithm 5.8: Parallel construction of $vFDM$

Input: 1. Input data set: $HDS = (U, A \cup \{d\})$
2. Fuzzy similarity relation: SIM, Fuzzy negation: N, and S-norm: $\perp$

Output: Vertical Fuzzy Discernibility Matrix (vFDM) as RDD

```
/* Read the input data set as an RDD, where the data set HDS is
   distributed to the nodes with vertical partitioning and the decision
   attribute {d} is broadcasted, such that each data partition becomes
   HDS^i = (U, A^i ∪ {d})  ∀i ∈ {1,2,...p}, where p is the number of data
   partitions in the cluster.                                          */
```

1 $hdsRDD = readASRDD(HDS)$
2 $val\ vFDMRDD = hdsRDD.mapPartitions(data => \{$
3 **for** *each* $record \in data$ **do**
4 $\quad attr = record(0)$
5 $\quad$ **for** $x = 1$ *to* $|U|$ **do**
6 $\quad\quad$ **for** $x' = x + 1$ *to* $|U|$ **do**
7 $\quad\quad\quad$ **if** $(d(x) != d(x'))$ **then**
$\quad\quad\quad\quad$ `// Compute similarity measure using Eq.` (5.1)
8 $\quad\quad\quad\quad \mu_{SIM_{attr}}(x, x') = FSmeasure(record(x), record(x'), \sigma(attr))$
$\quad\quad\quad\quad$ `// Compute discernibility measure using Eq.` (5.5)
9 $\quad\quad\quad\quad \mu_{DIS_{attr}}(x, x') = 1 - \mu_{SIM_{attr}}(x, x')$
10 $\quad\quad\quad$ **end**
11 $\quad\quad$ **end**
12 $\quad$ **end**
13 $\quad (attr, \mu_{DIS_{attr}})$
14 **end**
15 $\})$
16 Return $vFDMRDD$

5.5.3 Parallel attribute reduction using vFDM

After the construction of $vFDM$, the next step is to compute the reduct using $vFDM$. The MapReduce based parallel/distributed algorithms for reduct computation are given in the form of a driver, mapper, and reducer. The driver algorithm MR_VFDMFS: driver() is given in Algorithm 5.9, the mapper algorithm MR_VFDMFS: map() is given in Algorithm 5.10, and the reducer algorithm MR_VFDMFS: reduce() is given in Algorithm 5.11. Computation of the reduct from $vFDM$ is done majorly in three steps: (i) Computation of SAT_A, (ii) Computation of the best attribute and (ii) SAT-region removal. These steps are explained in this section.

5.5.3.1 Computation of SAT_A

Computation of SAT_A is required to check the end condition of the driver, which is ($SAT_R >= SAT_A$). As mentioned earlier, with vertically partitioned FDM, for an attribute, the discernibility values of all the entries are available within a node. Therefore, by using $S-norm$ ($\perp$), each mapper computes SAT_{A^i} for subset of attributes $A^i \subseteq A$ within i^{th} partition of the $vFDM$, $\forall i \in \{1, 2, 3, ...p\}$, where p is the number of partitions in the cluster. And, each mapper communicates $< key, value >=< cKey, SAT_{A^i} >$ pair to the reducer. Here, a single reducer is invoked by using the common key ($cKey$). Now, the reducer performs the union of all the satisfiability values received from the mappers. That is, reducer computes $SAT_A = \perp_{i=1}^{p} SAT_{A^i}$. The reducer returns SAT_A value to the driver.

5.5.3.2 Computation of the best attribute

In the driver (from Algorithm 5.9), initially the data set HDS is vertically partitioned into HDS^i ($i = 1, 2, ...p$), and decision attribute information is broadcasted to all the nodes of the cluster. Reduct R is initialized to empty set (ϕ), and initially the variable $nonSATReg$ contains the indices of entries of $vFDMRDD$. The variable $nonSATReg$ is used to incorporate DARA accelerator (to perform SAT-region removal) and in each iteration it is broadcasted to the nodes of the cluster.

The driver (Algorithm 5.9) invokes Algorithm 5.8 to construct $vFDM$. The Algorithm 5.8 returns the $vFDM$ as an RDD (i.e., $vFDMRDD$). The driver initializes the reduct R to an empty set ($\emptyset$) and $SAT(R) = 0.0$ and computes the best attribute by invoking the mapper (Algorithm 5.10), and reducer (Algorithm 5.11). As mentioned earlier, each mapper gets a vFDM-split ($vFDM^i$) as input along with fuzzy $S-norm : \perp$. As shown in Algorithm 5.10, from each record of $vFDM^i$ which contains an attribute and its discernibility values for all the entries, the value of $SAT(R \cup \{attr\})$ is computed by using $S-norm : \perp$. The local best attribute $lattr^{best}$ which gets maximum $SAT(R \cup \{attr\})$ is selected from $(A^i - R)$. A single $< key, value >=< cKey, (lattr^{best}, SAT(R \cup \{lattr^{best}\})) >$ pair is generated from each mapper communicated to the reducer. If cluster has p number of partitions, then all the mappers generate p number of $< key, value >$ pairs. And, since all the mappers are generating the same key ($cKey$), only a single reducer is invoked. Thus, the data movement in the cluster is significantly reduced when compared to horizontal partitioning based approach (MR_IFDMFS) given in Section 5.4 (Note that, in MR_IFDMFS algorithm, each mapper generates a $< key, value >$ pair for each entry in the DFDM).

Algorithm 5.9: MR_VFDMFS: driver()

Input: 1. Input data set: $HDS = (U, A \cup \{d\})$
2. Fuzzy similarity relation: SIM, Fuzzy negation: N, and S-norm: $\perp$

Output: Reduct R

Distribute the input data set HDS with vertical partitioning into the nodes of the cluster and broadcast decision attribute $\{d\}$, such that each data partition becomes $HDS^i = (U, A^i \cup \{d\})$ $\forall i \in \{1, 2, ...p\}$, where p is the number of data partitions in the cluster.
Construct $vFDM$ as an RDD ($vFDMRDD$) by invoking Algorithm 5.8
Initialize reduct $R = \emptyset$, $SAT(R) = 0$
Initial $nonSATReg$ gets the list of indices of entries in the $vFDMRDD$
Broadcast $nonSATReg$ and SAT_R
Compute $SAT(A)$ using procedure given in Section 5.5.3.1
repeat
    /* All operations are performed on the entries in $nonSATReg$ */
    Initiate mapper job by invoking Algorithm 5.10
    $val\ lattrRDD = vFDMRDD.mapPartitions(data => \{val\ mp = map()\})$
    /* Mapper returns $< key, value >=< cKey, (lattr^{best}, SAT(lattr^{best})) >$ pair, here $cKey$ is a common key, $lattr^{best}$ is local best attribute and $SAT(lattr^{best})$ is its satisfiability value */
    Initiate reducer job by invoking Algorithm 5.11
    $val\ gattrRDD = lattrRDD.reduce()$
    /* Reducer returns $< key', value' >=< attr^{best}, SAT(attr^{best}) >$ pair, where $attr^{best}$ is global best attribute and $SAT(attr^{best})$ is its satisfiability value */
    Collect the data from the reducers
    $R = R \cup \{attr^{best}\}$
    Filter $attr^{best}$ record from $vFDM$ as $< attr^{best}, \mu_{DIS_{attr^{best}}} >$
    Compute $SAT_R = \perp(\perp(R - attr^{best}), \mu_{DIS_{attr^{best}}})$ using a map only operation
    // SAT_R is vector of satisfiability values of attributes in R for all the entries in $vFDM$
    // $SAT(R)$ is satisfiability value of R for all entries in $vFDM$
    if $(SAT(R) >= SAT(A))$ **then**
        break
    end
    /* ==========SAT-region removal========== */
    $nonSATReg = vFDMRDD.filter(if(SAT_R(C_{xx'})! = SAT_A(C_{xx'}))$
    Broadcast $nonSATReg$ and SAT_R
until $(R.size < A.size)$
return R

Algorithm 5.10: MR_VFDMFS: map()

Input: 1. $vFDM^i$ is a partition of $vFDM$, each record of this partition read as $< key, value >=< attr, \mu_{DIS_{attr}} >$, where $attr$ represents an attribute and $\mu_{DIS_{attr}}$ represent the list of discernible values of the attribute $attr$.
2. Broadcasted $nonSATReg$, SAT_R and S-norm: $\perp$

Output: A $< key', value' >=< cKey, (lattr^{best}, SAT(lattr^{best})) >$ pair, here $cKey$ is a common key, $lattr^{best}$ is an attribute of A^i which gets maximum satisfiability value ($SAT(lattr^{best})$) within the partition.

```
1  lMax = 0.0, lattr^best = −1
2  for each record ∈ vFDM^i as (attr, μ_DIS_attr) do
3  |  if (attr ∉ R) then
4  |  |  Compute SAT(R ∪ {attr}) = ⊥(SAT_R, μ_DIS_attr)
5  |  end
6  |  if (SAT(R ∪ {attr}) > lMax) then
7  |  |  lattr^best = attr
8  |  |  lMax = SAT(R ∪ {attr})
9  |  end
10 end
11 Construct < key', value' >=< cKey, (lattr^best, SAT(R ∪ {lattr^best}) >
12 Emit intermediate < key', value' >
```

Algorithm 5.11: MR_VFDMFS: reduce()

Input: $< key, [V] >$, here, key is common key and $[V]$ is the list of $(lattr^{best}, SAT(lattr^{best}))$ pairs received from the mappers

Output: $< key', value' >=< attr^{best}, SAT(R \cup \{attr^{best}\}) >$

```
1  gMax = 0.0, attr^best = −1
2  for each v ∈ V as (lattr^best, SAT(lattr^best)) do
3  |  if (SAT(lattr^best) > gMax) then
4  |  |  attr^best = attr
5  |  |  gMax = SAT(R ∪ {attr})
6  |  end
7  end
8  Construct < key', value' >=< attr^best, SAT(R ∪ {attr^best}) >
9  Emit < key', value' >
```

In Algorithm 5.11, the reducer gets a set of $< ckey, [V] >$ pairs as the input from all the mappers in the cluster, where $ckey$ is a common key and $[V] = [(lattr^{best}, SAT(R \cup \{lattr^{best}\})]$ represents the list of pairs of local best attributes and their satisfiability values. The reducer finds the attribute which gets maximum satisfiability value and returns to the driver as best attribute $attr^{best}$ along with its satisfiability value $SAT(attr^{best})$. In the driver, $attr^{best}$ is added to the reduct set R. And the record of attribute $attr^{best}$ is fetched from $vFDM$ as $< attr^{best}, \mu_{DIS_{attr^{best}}} >$. This record is used to find the satisfiability values of the reduct attributes (SAT_R) by using $S - norm$ with a map only operation. Note that the notation SAT_R represents the satisfiability values of the attributes in R for all the entries in the $vFDM$

and it is broadcasted to all the nodes in the cluster. Now the driver performs SAT-region removal which is explained in Section 5.5.3.2 and the updated $nonSATReg$ is broadcasted. This above procedure is repeated until the condition ($SAT(R) >= SAT(A)$) is not satisfied or the condition ($R.size < A.size$) is satisfied. The satisfiability value $SAT(A)$ of the conditional attribute set A for all the entries in the $vFDM$ is computed by using the procedure given in Section 5.5.3.3.

5.5.3.3 SAT-region removal

The feature *SAT-region removal* in MR_IFDMFS acts as an accelerator (DARA). According to MR_IFDMFS algorithm, SAT-region gets the entries of FDM for which maximum satisfiability is reached (i.e., the entry for which ($SAT_R(C_{ij}) == SAT_A(C_{ij})$) is satisfied). These entries are removed from FDM in each iteration to avoid redundant computations in the next iteration. This feature is incorporated in each iteration of the present MR_VFDMFS algorithm, by identifying the indices of entries for which maximum satisfiability is not yet reached. That is, in the driver (Algorithm 5.9) *non-SAT region* is computed by using the condition: ($SAT_R(C_{ij})! = SAT_A(C_{ij})$). These indices are stored into the variable $nonSATReg$ and broadcasted to all the nodes, and in the next iteration of the algorithm, all the computations are performed only on the entries of $vFDM$ for which the indices are present in $nonSATReg$.

5.5.4 Complexity analysis of MR_VFDMFS algorithm

In the time complexity analysis of MR_VFDMFS algorithm, the following variables are used.

- $|U|$: the number of objects in the data set
- $|A|$: the number of conditional attributes in the data set
- p: the number of processors
- t_w: the number of time units to transfer one word of memory
- s: the number of time units to complete the synchronization
- $E = \sum_{i=1}^{n-1} c_i * (\sum_{j=i+1}^{n} c_j)$ (refer Section 5.3.3.1)

Table 5.3 shows the time complexities of each step of the phase in the MR_VFDMFS algorithm for one iteration. Note that, in the table, from step 1-10 of mapper to step 1-5 of reducer, the algorithm is repeated until ($SAT(R) == SAT(A)$) condition is satisfied. That is, these steps are repeated $|A|$ (in worst case) times. Hence, by adding up all the complexities, the total time complexity of the algorithm is obtained as given below.

Table 5.3: Time complexity analysis of MR_VFDMFS algorithm

Algorithm (phase)	Step* in Algorithm	Time complexity
Driver → (Algorithm 5.9)	1. Partitioning the data vertically	$\mathcal{O}(\frac{\lvert A\rvert*\lvert U\rvert}{p} * t_w)$
	2. Construct vFDM	$\mathcal{O}(\frac{\lvert A\rvert*E}{p} * t_w)$
	5. Broadcast $nonSATReg$ indices	$\mathcal{O}(E * t_w)$
	6. $SAT(A)$ computation	$\mathcal{O}(\frac{\lvert A\rvert*E}{p}) + \mathcal{O}(p * E * t_w)$
	14. Fetch a^{best} record from vFDM and find SAT_R	$\mathcal{O}(E * t_w)$
Mapper → (Algorithm 5.10)	1-10. Finding la^{best} attribute	$\mathcal{O}(\frac{\lvert A\rvert*E}{p})$
	Barrier synchronization	$\mathcal{O}(s)$
Shuffle and sort →	Transferring la^{best} attributes and their $SAT(la^{best})$ values	$\mathcal{O}(p * t_w)$
Reducer → (Algorithm 5.11)	1-5. Find $SAT(R \cup \{a^{best}\})$	$\mathcal{O}(p)$
	Barrier synchronization	$\mathcal{O}(s)$

* Step denotes the line number in the associated algorithm

$$((\frac{|A| * |U|}{p} * t_w) + (\frac{|A| * E}{p} * t_w) + (E * t_w) + (\frac{|A| * E}{p}) + (p * E * t_w) + |A| * \left((E * t_w) + (\frac{|A| * E}{p}) + (s) + (p * t_w) + (p) + (s)\right) \quad (5.14)$$

Above equation can be approximated as: $\mathcal{O}(\frac{|A|^2 * E}{p}) + \mathcal{O}(|A| * ((E + p) * t_w) + \mathcal{O}(|A| * (p + s))$. Since the time complexity of the sequential IFDMFS algorithm is $|A|^2 * E$, the time complexity of parallel MR_VFDMFS algorithm is reduced p times (i.e., $\mathcal{O}(\frac{|A|^2 * E}{p})$) than its sequential counterpart in addition with communication overhead $\mathcal{O}(|A| * ((E + p) * t_w) + \mathcal{O}(|A| * (p + s))$.

The entire vFDM is required for reduct computation using MR_VFDMFS algorithm. Thus, the space complexity of MR_VFDMFS algorithm is $\mathcal{O}(|A| * E)$. But, in MapReduce framework environment, the vFDM is partitioned and distributed to the nodes of the cluster where the workload is divided equally into p data partitions. Hence, each partition has the complexity of $\mathcal{O}(\frac{|A| * E}{p})$.

In the worst-case scenario, the aforementioned theoretical time and space complexity of the proposed MR_VFDMFS algorithm are described. However, because the MR_VFDMFS algorithm incorporates accelerator DARA (SAT-region removal), the actual time and space complexities are significantly reduced.

5.6 Experimental analysis

We carried out the experiments in two stages to evaluate the proposed MR_IFDMFS and MR_VFDMFS algorithms. Since both algorithms are MapReduce based parallel/distributed algorithms, they are implemented in Apache Spark (version: 2.3.1) with the Scala programming language (version: 2.11.4).

In the first stage of the experiments, we run the MR_IFDMFS algorithm on a node utilizing a single core to get a pure sequential version of MR_IFDMFS that is given as IFDMFS in Algorithm 5.2. The comparative results of the IFDMFS algorithm are presented by comparing with the existing PARA [72] algorithm, which is an accelerator for fuzzy-rough reduct computation. The PARA algorithm is implemented using C++ programming language. We obtained the source code of the PARA algorithm from the authors and conducted the experiments.

In the second stage of the experiments, the MR_IFDMFS and MR_VFDMFS algorithms are executed on a cluster of nodes. The efficiency of the proposed algorithms is shown by comparing with the existing state-of-the-art parallel/distributed fuzzy-rough attribute reduction algorithms: MR_FRDM_SBE [76] (proposed in the year 2019) and DFRS [56] (proposed in the year 2020). The MR_FRDM_SBE algorithm is implemented in Apache Spark (version: 2.3.1) with the Scala programming language (version: 2.11.4). The DFRS algorithm is a non-MapReduce parallel/distributed algorithm. The authors of DFRS provided MATLAB simulation for parallel fuzzy-rough attribute reduction and the source code made available in GitHub repository: `https://github.com/qu10wenhao/DFRS.git`.

5.6.1 Experimental setup

The experiments of the proposed sequential IFDMFS approach and the existing PARA approach [72] are conducted on a system with Intel (R) Core (TM) i7-8700 CPU@3.20GHz processor with 12 cores and 32 GB of main memory. The system is installed with Ubuntu 18.04 LTS operating system.

The experiments of the proposed approaches MR_IFDMFS, MR_VFDMFS and existing approach MR_FRDM_SBE [76] are carried out on a 7-node cluster. In the cluster, one node is set as master (driver) as well as slave, and the rest are set as workers (slaves). The master node uses Intel (R) Xeon (R) Silver 4110 CPU @ 2.10GHz processor with 32 cores and 64 GB of main memory. All the worker nodes use Intel (R) Core (TM) i7-8700 CPU@3.20GHz processor with 12 cores and 32 GB of main memory. All the nodes run on Ubuntu 18.04 LTS operating system and they are connected via Ethernet (with 1000 Mbps speed). Each

Table 5.4: Small size data sets used in the experiments of IFDMFS algorithm

S.No	Data set	Objects	Attributes	Classes	Attribute type
1	Ionosphere	351	34	2	Numerical
2	Waveform	5000	21	3	Numerical
3	Madelon	2000	500	2	Numerical
4	Satimage	6435	36	7	Numerical
5	Musk	6598	166	2	Numerical
6	Letter	20000	16	26	Numerical
7	Shuttle	58000	09	7	Numerical

Table 5.5: Large size data sets used in the experiments of MR_IFDMFS and MR_VFDMFS algorithms

S.No	Data set	Objects	Attributes	Classes	Attribute type
8	Genes	801	20531	5	Numerical
9	Isolet	7797	617	26	Numerical
10	HAPT	10929	561	12	Numerical
11	Diagnosis	58509	48	11	Numerical
14	Basehock	1993	4862	2	Categorical
15	Thyroid	7200	21	3	Categorical,Numerical
16	Gisette	6000	5000	2	Categorical
17	genes-S801-A101k	801	101320	5	Numerical
18	basehock-S2k-A53k	1993	53482	2	Categorical
19	heart-S270-A60k	270	60000	2	Categorical,Numerical
21	heart-S5k-A5k	5000	5000	2	Categorical,Numerical
22	heart-S10k-A5k	10000	5000	2	Categorical,Numerical
23	heart-S15k-A5k	15000	5000	2	Categorical,Numerical
24	heart-S5k-A10k	5000	10000	2	Categorical,Numerical
25	heart-S5k-A15k	5000	15000	2	Categorical,Numerical

node is installed with Java 1.8.0_171, Apache Spark 2.3.1, and Scala 2.11.4. The experiments of the existing DFRS algorithm [56] are conducted on a system with Intel (R) Core (TM) i7-8700 CPU@3.20 GHz processor having 12 cores and 32 GB of main memory. The system is installed with Ubuntu 18.04 LTS operating system and MATLAB 2017 environment. The DFRS source code is executed on this node and the results of the simulation in 7 nodes are obtained.

All the hybrid data sets used in the experimental analysis are selected from the UCI Machine Learning Repository [31]. These data sets are categorised into two groups: smaller size and larger size. Smaller size data sets are used in the experimental analysis of sequential approaches in the first stage of experiments and larger size data sets are used in the experimental

analysis of parallel/distributed approaches in the second stage of experiments. A detailed description of smaller size data sets and larger size data sets is given in Table 5.4 and Table 5.5 respectively. Since the existing sequential PARA algorithm works on numerical data sets, we used numerical data sets in experimental comparison of IFDMFS and PARA algorithms. Thus, Table 5.4 contains all the numerical data sets. And, Table 5.5 contains hybrid data sets. All the larger size data sets with different sizes are chosen according to limited hardware configuration of the cluster. In the selection of these data sets, we considered the aspect of variance in sizes of object space and attribute space to illustrate the relevance and limitations of the proposed MR_IFDMFS, MR_VFDMFS approaches. Thus, few data sets such as "Gene expression Cancer RNA-Seq" (renamed as Genes), "Basehock" and "Heart" are replicated several times in object and attribute space, details of these data sets are provided along with their object and attribute space sizes in Table 5.5. For example, the original "Heart" data set has 270 objects and 13 attributes, and after replication, the data set "heart-S5k-A15k" contains 5000 objects and 15000 attributes.

5.6.2 Experimental results of IFDMFS algorithm

In this section, the efficiency of the proposed IFDMFS algorithm is shown by comparing its results with the PARA algorithm [72] based on computational time, reduct size. As IFDMFS and PARA are sequential algorithms, the experiments are conducted on small data sets given in Table 5.4.

Reduct is computed for the given data sets using the proposed IFDMFS algorithm and the PARA algorithm. In PARA, the authors used a threshold value α ($0 \leq \alpha \leq 1$) in computing the reduct. The threshold value α is used to bridge the gap between the dependency measures of all conditional attributes and reduct attributes. In PARA, therefore the dependency measure with α is considered as the approximate dependency measure of the reduct set. But we have not used any threshold value in the proposed IFDMFS. Therefore the reduct generated by the proposed algorithm is exact. Since we do not use any threshold value, we have taken the α value as 0.0 for PARA, for appropriate comparison with IFDMFS. In addition, we also performed experiments on PARA with α value of 0.12, as suggested by the authors of PARA. The results are reported in Table 5.6. The running time and the reduct size of the obtained reduct on each data set are separately reported.

The following observations are made from the results:

Table 5.6: Running time (Seconds) and reduct size of PARA and IFDMFS algorithms

Data set	PARA ($\alpha = 0.0$)		PARA ($\alpha = 0.12$)		IFDMFS	
	Running time	Reduct size	Running time	Reduct size	Running time	Reduct size
Ionosphere	6.74	34	1.96	16	2.06	07
Waveform	996.52	21	316.15	14	86.47	08
Madelon	> 259200	*	105341.20	138	231.55	07
Satimage	5993.86	36	345.79	12	286.84	14
Musk	92660.73	166	28531.10	71	554.51	20
Letter	6420.3	16	3558.30	15	2369.03	15
Shuttle	4721.33	09	1520.37	06	2903.80	09

* The run time of the PARA is more than three days. The results are not reported due to manual termination of the program.

i) The PARA algorithm with threshold value of $\alpha = 0.0$ resulted with no dimensionality reduction, and all the attributes were selected as reduct. Thus, even though the proposed IFDMFS algorithm did not use a threshold value, it is compared with the PARA that has $\alpha = 0.12$.

ii) IFDMFS algorithm achieved a minimum of 17% and a maximum of 99% computational gains over PARA ($\alpha = 0.12$) on all the data sets except Shuttle. Both the existing and proposed algorithms obtained similar computational times for Ionosphere data set. These significant results of the proposed IFDMFS algorithm over PARA is due to incorporated DARA.

iii) The existing PARA ($\alpha = 0.12$) algorithm obtained 47% computational gain over the proposed IFDMFS algorithm in the Shuttle data set which has larger object space and much smaller attribute space. This is due to the increase in the construction time of the discernibility matrix with the larger object space is not able to compensate the benefits obtained in the iterations for reduct computation as $|A|$ is much smaller.

iv) It is also noted from the results that the IFDMFS generated the reducts that are significantly smaller in size relative to the PARA ($\alpha = 0.12$) algorithm.

In summary, the comparative study of IFDMFS and PARA has shown experimentally that the proposed algorithm is useful in achieving shorter length reducts with substantial computational gains.

5.6.3 Experimental results of MR_IFDMFS and MR_VFDMFS

In this section, the computational efficiency and performance evaluation of the proposed parallel algorithms (MR_IFDMFS, MR_VFDMFS) is done by comparing their results with the existing parallel approaches MR_FRDM_SBE [76] and DFRS [56]. Since the proposed and existing algorithms are parallel/distributed methods, experiments are performed on all the large data sets given in Table 5.5.

5.6.3.1 Computational evaluation

The proposed MR_IFDMFS and MR_VFDMFS algorithms are numerically compared with the existing MR_FRDM_SBE and DFRS algorithms based on the computational time and reduct size. The experiments of proposed MR_IFDMFS, MR_VFDMFS and the existing MR_FRDM_SBE algorithms are performed on 7-node cluster, and the simulation results of DFRS algoritm are obtained for 7 nodes. Since MR_FRDM_SBE and DFRS algorithms are developed for attribute reduction in numerical data sets, in the first step, the comparative analysis of these algorithms with the proposed algorithms is done on numerical data sets and the results are reported in Table 5.7. In the second step, the proposed algorithms MR_IFDMFS and MR_VFDMFS are compared based on different hybrid data sets with variance in object and attribute space. The comparative results are reported in Table 5.8. The observations on the results from Table 5.7 and Table 5.8 are listed as follows.

Table 5.7: Running time (Seconds) and reduct size results of MR_IFDMFS, MR_VFDMFS, MR_FRDM_SBE, and DFRS algorithms on large numerical data sets

Data set	MR_IFDMFS		MR_VFDMFS		MR_FRDM_SBE		DFRS	
	Running time	Reduct size	Running time	Reduct size	Running time	Reduct size	Running time	Reduct size
Genes	126.89	06	68.73	06	16742.80	10	525.31	39
Isolet	335.69	09	320.83	09	12549.32	11	3682.13	541
HAPT	467.70	09	300.87	09	8715.97	12	1093.73	347
Diagnosis	594.99	12	901.36	12	1662.36	21	8083.69	26

i) From the comparison of the running times, we can observe that the MR_IFDMFS and MR_VFDMFS algorithms performed significantly better than MR_FRDM_SBE algorithm on all the data sets. On all the data sets, the proposed algorithms obtained a minimum of 26%, and a maximum of 99% of computational gains over MR_FRDM_SBE algorithm. In specific, the computational gains of MR_VFDMFS are higher than another proposed MR_IFDMFS algorithm for high dimensional data sets such as Genes,

Table 5.8: Running time (Seconds) and reduct size results of MR_IFDMFS and MR_VFDMFS algorithms on hybrid data sets

Data set	MR_IFDMFS		MR_VFDMFS	
	Running time	Reduct size	Running time	Reduct size
Basehock	263.06	36	68.57	36
Thyroid	21.36	19	49.39	19
Gisette	2268.33	10	5234.83	10
genes-S801-A101k	810.61	06	111.25	06
basehock-S2k-A53k	8123.64	36	4962.97	36
heart-S270-A60k	360.02	07	27.16	07

Isolet and HAPT. And, MR_IFDMFS performs better than MR_VFDMFS on Diagnosis data set which has larger object space.

ii) From the comparison of the running times of MR_IFDMFS and MR_VFDMFS with the existing DFRS, it can be noticed that, on all the data sets the proposed algorithms perform well. Here, computational gains of proposed algorithms varies from 22% to 93%.

iii) It is also noted from the results that for all data sets, the proposed algorithms have generated smaller size reduct set than the DFRS and MR_FRDM_SBE algorithms. And, both proposed algorithms generated same size reduct sets.

iv) From the results on hybrid data sets in Table 5.8, it can be observed that, the vertical partitioning based MR_VFDMFS algorithm performed better than horizontal partitioning based MR_IFDMFS algorithm on the data sets with larger attribute space such as Basehock, genes-S801-A101k, basehock-S2k-A53k and heart-S270-A60k. And, in contrast, MR_IFDMFS algorithm perform better than MR_VFDMFS for larger object space data sets such as Thyroid and Gisette.

The significant computational gains achieved by MR_IFDMFS and MR_VFDMFS algorithms on all the data sets strongly establishes the role of the proposed accelerator DARA in imparting space reduction as the algorithm progresses and there by aiding in reduction of computational time. And, in specific, the notable achievements of MR_VFDMFS over MR_IFDMFS on high dimensional data sets illustrate the advantage of vertical partitioning strategy over horizontal partitioning strategy for such data sets.

Table 5.9: Reduct obtained by IFDMFS, MR_IFDMFS and MR_VFDMFS algorithms for different data sets

Data set	IFDMFS	MR_IFDMFS	MR_VFDMFS
heart	{ 0, 9, 2, 7, 3, 11, 4 }	{ 0, 11, 2, 7, 3, 9, 4 }	{ 0, 11, 2, 7, 3, 9, 4 }
Ionosphere	{ 0, 5, 4, 25, 14, 28, 21 }	{ 0, 5, 4, 25, 14, 28, 21 }	{ 0, 5, 4, 25, 14, 28, 21 }
Waveform	{ 0, 10, 20, 1, 6, 2, 16, 19 }	{ 0, 10, 20, 1, 6, 2, 16, 19 }	{ 0, 10, 20, 1, 6, 2, 16, 19 }
Madelon	{ 153, 475, 28, 118, 455, 378, 239 }	{ 153, 475, 28, 118, 455, 378, 239 }	{ 153, 475, 28, 118, 455, 378, 239 }
Satimage	{ 0, 10, 24, 14, 6, 9, 13, 2, 34, 22, 16, 26, 08, 30 }	{ 0, 10, 24, 14, 6, 9, 13, 2, 34, 22, 16, 26, 08, 30 }	{ 0, 10, 24, 14, 6, 9, 13, 2, 34, 22, 16, 26, 8, 30 }
Musk	{ 5, 125, 157, 106, 28, 9, 53, 41, 105, 54, 86, 7, 135, 35, 48, 50, 87, 94, 131, 83 }	{ 5, 125, 157, 106, 28, 9, 53, 41, 105, 54, 86, 7, 135, 35, 48, 50, 87, 94, 131, 83 }	{ 5, 125, 157, 106, 28, 9, 53, 41, 105, 54, 86, 7, 135, 35, 48, 50, 87, 94, 131, 83 }
Letter	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }
Shuttle	{ 0, 5, 1, 6, 2, 7, 3, 8, 4 }	{ 0, 5, 1, 6, 2, 7, 3, 8, 4 }	{ 0, 5, 1, 6, 2, 7, 3, 8, 4 }
Genes	{9251,7949,1071,17187,15865,10818}	{9251,7949,5397,1071,17187,15865}	{9251,7949,5397,1071,17187,15865}
Isolet	{221,593,547,375,370,11,603,227,364}	{221,593,547,616,370,11,603,227,364}	{221,593,547,616,370,11,603,227,364}
HAPT	{559,115,538,37,105,118,198,557,555}	{559,548,538,37,105,118,198,557,555}	{559,548,538,37,105,118,198,557,555}
Diagnosis	{ 2, 3, 4, 10, 6, 13, 18, 14, 29, 27, 42 }	{ 2, 17, 4, 10, 6, 13, 18, 14, 29, 27, 42 }	{ 2, 17, 4, 10, 6, 13, 18, 14, 29, 27, 42 }
Basehock	{ 2630, 2778, 3399, 1290, 3285, 4543, 3891, 1792, 3280, 881, 4115, 4364, 3301, 1274, 1365, 493, 2509, 4750, 1099, 3755, 3824, 4354, 2578, 1528, 255, 355, 3214, 327, 4314, 4800, 1285, 1721, 4681, 3337, 4350, 3299 }	{ 2630, 2778, 3399, 1290, 3285, 4543, 3891, 1792, 3280, 881, 4115, 4364, 3301, 1274, 1365, 493, 2509, 4750, 1099, 3755, 3824, 4354, 2578, 1528, 255, 355, 3214, 327, 4314, 4800, 1285, 1721, 4681, 3337, 4350, 3299 }	{ 2630, 2778, 3399, 1290, 3285, 4543, 3891, 1792, 3280, 881, 4115, 4364, 3301, 1274, 1365, 493, 2509, 4750, 1099, 3755, 3824, 4354, 2578, 1528, 255, 355, 3214, 327, 4314, 4800, 1285, 1721, 4681, 3337, 4350, 3299 }
Thyroid	{ 0, 5, 10, 20, 1, 9, 13, 2, 17, 12, 7, 3, 18, 16, 11, 8, 19, 4, 15 }	{ 0, 15, 9, 1, 16, 2, 17, 3, 18, 10, 4, 11, 12, 19, 13, 5, 20, 7, 8 }	{ 0, 15, 9, 1, 16, 2, 17, 3, 18, 10, 4, 11, 12, 19, 13, 5, 20, 7, 8 }
Gisette	{ 4570, 1391, 4466, 3656, 4488, 2993, 3469, 4693, 4987, 4868 }	{ 1391, 4466, 4693, 3656, 2391, 4488, 4570, 3469, 4987, 2993 }	{ 1391, 4466, 4693, 3656, 2391, 4488, 4570, 3469, 4987, 2993 }

For the reproducible research, obtained reducts of proposed algorithms IFDMFS, MR_IFDMFS and MR_VFDMFS algorithms for all the given original data sets (the data sets without replication) are given in Table 5.9.

5.6.3.2 Performance evaluation

Using speedup, scaleup and sizeup metrics, the performance of the proposed MR_IFDMFS and MR_VFDMFS algorithms is evaluated and compared to existing MR_FRDM_SBE and DFRS algorithms on various data sets. Three data sets are chosen such that the first data set has large attribute space, the second has large object space, and the third data set has large object space as well as attribute space. Separate figures are given to show the performance results of the algorithms on three data sets.

Speedup evaluation:

The speedup of the proposed algorithms have been evaluated on the data sets with different nodes from 1 to 7. From the experimental setup given in Section 5.6.1, it can be observed that the master node has 32 cores and the remaining slave nodes have 12 cores each. Since we have set the master node also as a slave, the number of cores is mismatched with other slave nodes. Owing to this mismatch, in finding the speedup metric of the system, we took nodes ratio based on the number of cores in the node. Figure 5.2 shows the speedup results of different data sets with a different nodes ratios (number of cores) in the cluster.

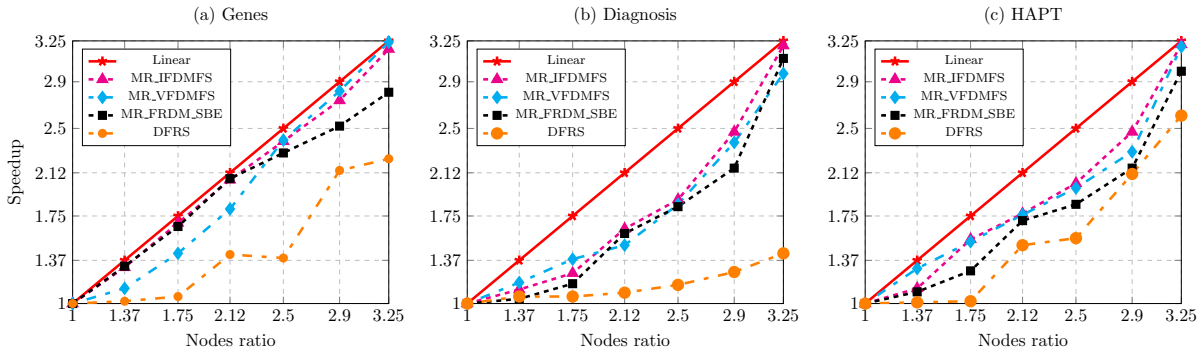


Figure 5.2: Speedup results of MR_IFDMFS, MR_VFDMFS, MR_FRDM_SBE and DFRS algorithms on different data sets

From the findings in Figure 5.2, it can be observed that, on all the data sets, the speedup of the proposed MR_IFDMFS and MR_VFDMFS algorithms is improved with an increase

in the number of cores. The efficiency of proposed algorithms is higher than the existing MR_FRDM_SBE and DFRS algorithms. The plots of both the proposed algorithms are closer to the linear plot, and MR_VFDMFS performs slightly better than MR_IFDMFS on Genes data set (big dimensional data set).

Scaleup evaluation

To find the scaleup performance of the proposed algorithms,, the data set size is increased in proportion to the number of cores in the cluster. Each data set is divided into 20%, 40%, 60%, 80% and 100% sizes of original data set, and the number of cores in the cluster increased from 20, 40, 60, 80 and 100 respectively. Figure 5.3 shows the scaleup results of the proposed algorithms in comparison with the existing MR_FRDM_SBE and DFRS algorithms on different data sets.

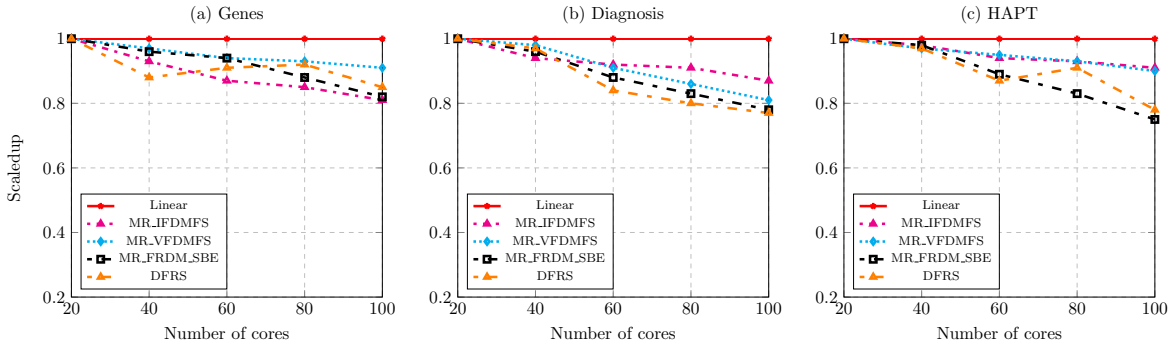


Figure 5.3: Scaleup results of MR_IFDMFS, MR_VFDMFS, MR_FRDM_SBE and DFRS algorithms on different data sets

The higher scaleup value shows the better performance of the proposed algorithms. From the results shown in Figure 5.3, the scaleup values of both proposed algorithms are higher than 0.8 indicates that the proposed algorithms scale well. Note that the existing algorithms also performing on par with proposed algorithms.

Sizeup evaluation:

To find the sizeup metric of the proposed algorithms, we kept the number of nodes unchanged with seven nodes, and changed the size of the data set as 20%, 40%, 60%, 80%, and 100% of objects in the original data set. Figure 5.4 shows the sizeup performance results of the proposed algorithms in comparison with the existing MR_FRDM_SBE and DFRS algorithms on different data sets.

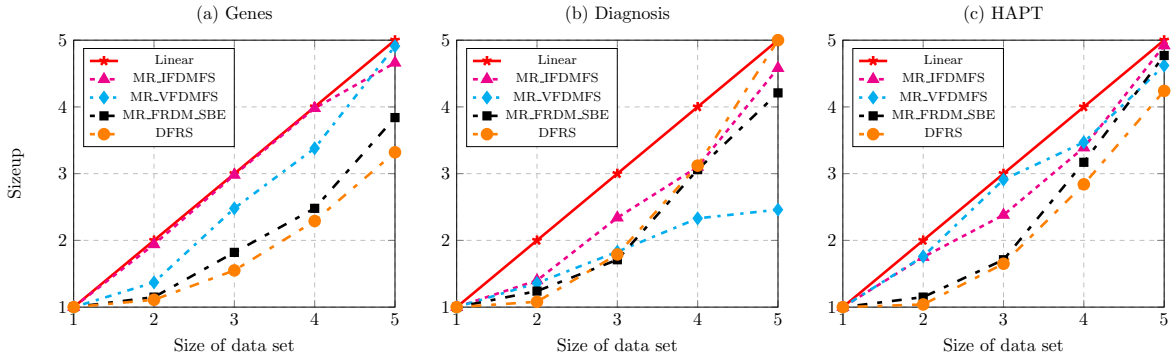


Figure 5.4: Sizeup results of MR_IFDMFS, MR_VFDMFS, MR_FRDM_SBE and DFRS algorithms on different data sets

In Figure 5.4, for all the data sets, the sizeup performance of the proposed algorithms increased when the size of the data set was increased. From the figure, we can observe that, the existing MR_FRDM_SBE and DFRS algorithms are producing better sizeup results than proposed MR_IFDMFS and MR_VFDMFS algorithms. Figure 5.4 also shows that all existing and proposed algorithms produce better sizeup results, as their plots are lower than the linear plots in all data sets.

5.6.3.3 Discussion

It can be observed that the experiments in the cluster for MR_IFDMFS and MR_VFDMFS are conducted on the large data sets having a few hundred thousands of objects or a few thousands of attributes. These data sets are categorised as large due to the fact that the resulting space utilisation for the matrices DFDM and vFDM run into several millions of entries occupying several Giga bytes of memory space. This is further coupled by the overhead involved in the Apache Spark maintenance of RDDs across the transformations and for the meta data in achieving fault tolerance. From the scaleup results shown in Figure 5.3, it is obvious that, in order to scale data sets to much higher sizes, more nodes need to be added to the cluster. As the cost of shuffle and sort phase of DFDM and vFDM construction and distributed reduct computation are minimal, the proposed approaches are scalable to very large data sets under horizontal expansion of the cluster.

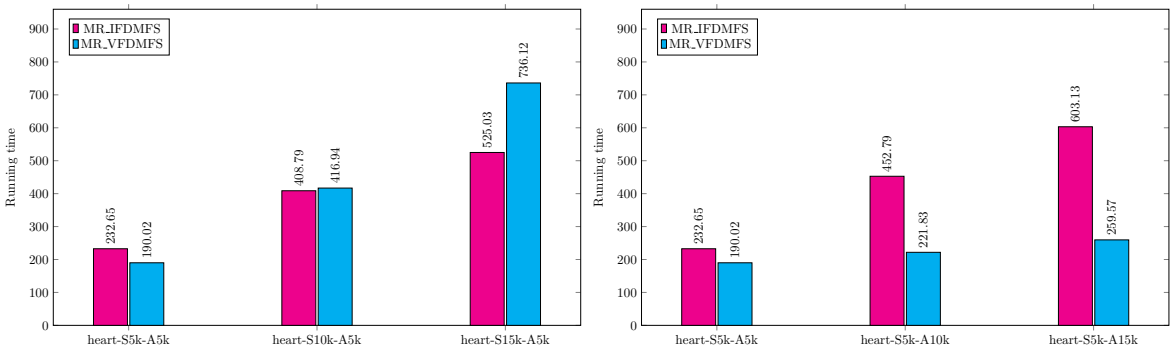
5.6.4 Impact of the data partitioning strategy

The experimental results in Section 5.6.3 have suggested that vertical partitioning based MR_VFDMFS algorithm is suitable for high dimensional data sets, and horizontal parti-

Table 5.10: Comparison of MR_IFDMFS and MR_VFDMFS for varying objects and attributes of Heart data set (Time: Seconds)

S.No	Data set	MR_IFDMFS Running Time	MR_VFDMFS Running Time
1	heart-S5k-A5k	232.65	190.02
2	heart-S10k-A5k	408.79	416.94
3	heart-S15k-A5k	525.03	736.12
4	heart-S5k-A10k	452.79	221.83
5	heart-S5k-A15k	603.13	259.57

tioning based MR_IFDMFS algorithm is ideal for the data sets having moderate attribute space with larger object space. To further investigate the relevance and limitations of the proposed algorithms, we conducted an experiment between MR_IFDMFS and MR_VFDMFS algorithms. The objective of the experiment is to determine the nature of data sets relevant for horizontal partitioning based reduct algorithms and vertical partitioning based reduct algorithms.



(a) Scalability in the object space **(b)** Scalability in the attribute space

Figure 5.5: Behavior of MR_IFDMFS and MR_VFDMFS for varying object space and attribute space of Heart data set

In this experiment, the original Heart data set having 270 objects and 13 attributes is replicated several times in object and attribute space to get heart-S5k-A5k which has equal size object space and attribute space. This data set's object space is replicated by keeping attribute space constant, and the experiments are conducted, the results of both algorithms are reported in Table 5.10 under the serial number 2 and 3. Similarly, heart-S5k-A5k is replicated in attribute space by keeping object space constant and the experiments are conducted, the results are reported in Table 5.10 under the serial number 4 and 5. Figure 5.5 demonstrates the computational time analysis for scalability in object space in Figure 5.5a and the attribute space in Figure 5.5b.

It is evidently clear from these results that, increase in object space resulted in a considerable increase in computational time of MR_VFDMFS. And, similarly increase in attribute space resulted in a more significant increase in computational time of MR_IFDMFS. That is, MR_VFDMFS is more suitable for big dimensional data sets. But it is not recommended for data sets of larger object space. The horizontal partitioning based MR_IFDMFS algorithm is found more suitable for scalability in object space. Hence, as shown in Chapter 3 and 4, this section once again demonstrated that, the vertical partitioning based algorithms are suitable for big dimensional data sets. And, horizontal partitioning based algorithms are more ideal for the data sets having larger object space with moderate attribute space.

5.7 Summary

In this chapter, we introduced a fuzzy discernibility matrix-based accelerator. The idea behind the proposed accelerator is the removal of SAT-region. With this feature, the entries of the discernibility matrix that have reached maximum satisfiability were removed from the matrix in each iteration and the reduct computation performed on the remaining entries of the matrix. Therefore, SAT-region removal served as an accelerator and referred to as DARA. Based on DARA, a sequential IFDMFS algorithm proposed for fuzzy-rough attribute reduction. To deal with large data sets in attribute reduction, we also proposed MapReduce based algorithms using horizontal partitioning strategy (MR_IFDMFS) and vertical partitioning strategy (MR_VFDMFS). These two algorithms are parallel/distributed versions of the IFDMFS algorithm. The experimental results have shown that the proposed algorithms IFDMFS, MR_IFDMFS and MR_VFDMFS performed better than the existing state-of-the-art approaches. Extensive experimental analysis along with theoretical validation establishes the relevance and efficiency of the proposed approaches in handling large hybrid data sets for attribute reduction.

Chapter 6

Conclusions and Future work

This chapter highlights the author's explorations after the research process that resulted in the development of this thesis. The research process began with the motivation and its objectives, which formed a basis for carrying out this scientific work. The research work is motivated by the challenges presented by today's large-scale data sets, which include, *big dimensionality*, *variety of the data* and *data partitioning* strategy used in distributed attribute reduction. Each of these challenges served as the research work's objectives, which were discussed in Chapter 1. Following the formulation of the research objectives, the research process moved on to Chapter 2 to examine and analyse theoretical principles relating to *rough set theory* and the *Apache Spark MapReduce framework*, which served as the foundation for this research. The following section summarizes major contributions and achievements of this thesis. And, Section 6.2 identifies various directions for further research in rough set-based scalable attribute reduction.

6.1 Research summary

This research focused on scalable attribute reduction in large-scale data sets using MapReduce, with an emphasis on the big dimensionality of the data set. *This thesis objective was to explore MapReduce based parallel/distributed reduct computation in categorical, incomplete and hybrid decision systems, where the relevance of horizontal and vertical partitioning strategies were investigated in partitioning the input data to the nodes of the cluster.* The contributions to thesis were made in relation to the research objectives. All the significant contributions were discussed in chapter 3 to 5. Brief summary of each contribution is provided below.

All the existing MapReduce based reduct computation approaches in categorical data sets adopted horizontal partitioning strategy for partitioning the data to the cluster of computers,

where the data set was partitioned in object space. This strategy resulted in computational overheads for big dimensional data sets. As an initial contribution to this thesis, a classical rough sets-based approach (MR_IQRA_VP) to attribute reduction using MapReduce was proposed. An alternative vertical partitioning strategy was examined in this approach, which was utilised to partition the input data set in the attribute space to the cluster nodes. The application of this strategy for attribute reduction in large-scale categorical data sets with big dimensionality was investigated. This vertical partitioning strategy avoided the limitations of horizontal partitioning strategy and enabled the incorporation of granular refinement feature, which fetched significant computational gains for the proposed approach.

Different strategies were used in the MapReduce framework to parallelize existing extensions to classical rough sets for attribute reduction in large-scale incomplete data sets. As a result, as the second contribution of thesis, MapReduce-based attribute reduction approaches for incomplete decision systems were presented, employing Novel Granular Framework (NGF) (an extension to classical rough sets) for handling incompleteness in the data set. An alternative representation of the NGF is introduced and adopted by one of the proposed approaches (MRIDS_HP). This proposed approach used horizontal partitioning strategy to partition the input data. Another approach (MRIDS_VP) incorporated the existing NGF and employed a vertical partitioning strategy. It is worth noting that, to the best of our knowledge, the presented methods are the first of its kind research on parallel/distributed attribute reduction in large-scale IDS.

The advantages of discernibility matrix over dependency measure, and also non availability of discernibility matrix based accelerators in the literature inspired us to investigate accelerators and the corresponding parallel/distributed approaches based on discernibility matrix as part of third and fourth contributions of thesis. Fuzzy-rough set model (an extension to classical rough sets) used to deal with hybrid decision systems. A fuzzy discernibility matrix based attribute reduction accelerator (DARA) was introduced for scalable attribute reduction in hybrid decision systems. Based on this accelerator, a sequential approach IFDMFS (Improved Fuzzy Discernibility Matrix based Feature Selection) and corresponding MapReduce based parallel/distributed versions of IFDMFS (MR_IFDMFS, MR_VFDMFS) were proposed. For input data partitioning, the approach MR_IFDMFS employed a horizontal partitioning strategy, while the approach MR_VFDMFS used a vertical partitioning strategy.

From all the contributions of this thesis, it was observed that, the horizontal partitioning of the input data enabled the incorporation of positive region removal and SAT-region removal features in the approaches (MRIDS_HP and MR_IFDMFS) proposed for parallel attribute reduction in large-scale incomplete and hybrid decision systems respectively. The horizontal

partitioning strategy was not suitable for incorporating granular refinement feature. However, the vertical partitioning strategy allowed all the approaches (MR_IQRA_VP, MRIDS_VP and MR_VFDMFS) to incorporate all the features: positive region removal, SAT-region removal and granular refinement. Furthermore, for all the proposed approaches, vertical partitioning strategy simplified the shuffle and sort phase, which is a complex phase of the MapReduce framework for large-scale data processing.

Apache Spark framework was used to implement the proposed approaches. Extensive experimental study was performed on various benchmark large-scale data sets with variations in object and attribute space. The efficiency of the proposed methods was assessed using *computational evaluation* (running time, reduct, and reduct size were used as metrics), *performance evaluation* (speedup, scaleup, and sizeup were used as metrics), and *impact of the data partitioning strategy* for splitting the input data.

It has been experimentally demonstrated that the proposed approaches outperformed the existing state-of-the-art approaches. The experimental results along with theoretical validation showed that the horizontal partitioning based approaches performed well for the larger object space data sets with moderate attribute space. And the vertical partitioning based approaches were relevant and scale well for moderate object space data sets with big dimensionality.

6.2 Future directions

Various challenges in developing scalable rough set-based attribute reduction approaches were addressed in this thesis. However, scalable reduct computation can be enhanced further by addressing some major concerns that need in-depth analysis and resolution. This section provides some insight into these problems in preparation for future work in this area.

From the experimental study of all the proposed approaches, it is clear that, the horizontal partitioning and vertical partitioning-based algorithms scale well for the data sets with either a huge object space or a huge attribute space (big dimensionality), but they are less effective in dealing with data sets with both a large object space and a large attribute space. As a result, this research has the potential to look at viable rough set-based MapReduce approaches that can simultaneously scale in both huge object space and attribute space.

As stated earlier, the big data is characterised with three V's, namely *volume*, *variety* and *velocity*. Since the proposed approaches are developed for big data, it can be observed that, all the proposed algorithms are scale well for the data sets with huge object space or attribute space. As huge object space or huge attribute space signifies the *volume* of the data, it is clear that, the proposed approaches are dealing with *volume* characteristic of

big data. Furthermore, because the proposed algorithms were developed for the categorical (CDS), incomplete (IDS) and hybrid decision systems (HDS), it is evident that, the proposed algorithms are dealing with *variety* characteristic of big data. Therefore, this research offers the possibility to investigate suitable rough set-based MapReduce methods that can deal with the *velocity* characteristic of big data. We can deal with the *velocity* issue by proposing MapReduce-based incremental reduct computation approaches for streaming data.

In Chapter 4, MapReduce based approaches were proposed for parallel attribute reduction in incomplete data sets, where the incompleteness (missing values) percentage used in the data set was moderate. In a certain scenario in recommender systems, a high percentage of missing values occurs, resulting in the formation of sparse data sets. Alternative representations and appropriate MapReduce-based strategies are required for such sparse data sets, which will be investigated in the future.

We hope that the contributions provided in this thesis will help deliver the benefits of rough set based attribute reduction for large-scale decision systems and will aid knowledge engineering in big data scenarios.

References

[1] **Apache Hadoop**. `https://hadoop.apache.org`.

[2] **Apache Spark**. `https://spark.apache.org`.

[3] **Cornell University VIA Databases**. `http://www.via.cornell.edu/databases`.

[4] **Gene expression data, NCBI**. `https://www.ncbi.nlm.nih.gov/gene`.

[5] **HashMap, Scala 2.11.4 library.** `https://www.scala-lang.org/api/2.11.4/#scala.collection.mutable.HashMap`.

[6] GS Almasi and A Gottlieb. **Highly Parallel Computing Benjamin/Cummings"**. *New York*, **19942**, 1994.

[7] Hasan Asfoor, Rajagopalan Srinivasan, Gayathri Vasudevan, Nele Verbiest, Chris Cornells, Matthew Tolentino, Ankur Teredesai, and Martine De Cock. **Computing fuzzy rough approximations in large scale information systems**. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 9–16. IEEE, 2014.

[8] Kiran Bandagar, Pandu Sowkuntla, Salman Abdul Moiz, and P. S. V. S. Sai Prasad. **MR_IMQRA: An Efficient MapReduce Based Approach for Fuzzy Decision Reduct Computation**. In *International Conference on Pattern Recognition and Machine Intelligence*, pages 306–316. Springer International Publishing, 2019.

[9] Richard E Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 2015.

[10] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. **Recent advances and emerging challenges of feature selection in the context of big data**. *Knowledge-Based Systems*, **86**:33–45, 2015.

[11] Verónica Bolón-Canedo, Noelia Sánchez-Marono, Amparo Alonso-Betanzos, José Manuel Benítez, and Francisco Herrera. **A review of microarray datasets and applied feature selection methods**. *Information Sciences*, **282**:111–135, 2014.

[12] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D Ernst. **HaLoop: efficient iterative data processing on large clusters**. *Proceedings of the VLDB Endowment*, **3**(1-2):285–296, 2010.

[13] Hongmei Chen, Tianrui Li, Yong Cai, Chuan Luo, and Hamido Fujita. **Parallel attribute reduction in dominance-based neighborhood rough set**. *Information Sciences*, **373**:351–368, 2016.

[14] Jinkun Chen, Jusheng Mi, and Yaojin Lin. **A graph approach for fuzzy-rough feature selection**. *Fuzzy Sets and Systems*, **391**:96–116, 2020.

[15] Mincheng Chen, Jingling Yuan, Lin Li, Dongling Liu, and Tao Li. **A fast heuristic attribute reduction algorithm using Spark**. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2393–2398. IEEE, 2017.

[16] Alexios Chouchoulas and Qiang Shen. **Rough set-aided keyword reduction for text categorization**. *Applied Artificial Intelligence*, **15**(9):843–873, 2001.

[17] Cheng Chu, Sang Kyun Kim, Yian Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. **Map-reduce for machine learning on multicore**. *Advances in neural information processing systems*, **19**:281, 2007.

[18] Martine De Cock, Chris Cornelis, and Etienne E. Kerre. **Fuzzy Rough Sets: The Forgotten Step**. *IEEE Transactions on Fuzzy Systems*, **15**(1):121–130, feb 2007.

[19] Chris Cornelis, Martine De Cock, and Anna Maria Radzikowska. **Fuzzy Rough Sets: From Theory into Practice**. In *Handbook of Granular Computing*, pages 533–552. John Wiley & Sons, Ltd, 2008.

[20] Chris Cornelis, Richard Jensen, Germán Hurtado, and Dominik Ślęzak. **Attribute selection with fuzzy decision reducts**. *Information Sciences*, **180**(2):209–224, jan 2010.

[21] Michal Czolombitko and Jaroslaw Stepaniuk. **Attribute reduction based on MapReduce model and discernibility measure**. In *IFIP International Conference on Computer Information Systems and Industrial Management*, pages 55–66. Springer, 2016.

[22] Jianhua Dai, Hu Hu, Wei-Zhi Wu, Yuhua Qian, and Debiao Huang. **Maximal-Discernibility-Pair-Based Approach to Attribute Reduction in Fuzzy Rough Sets**. *IEEE Transactions on Fuzzy Systems*, **26**(4):2174–2187, aug 2018.

[23] Jianhua Dai, Qinghua Hu, Jinghong Zhang, Hu Hu, and Nenggan Zheng. **Attribute selection for partially labeled categorical data by rough set approach**. *IEEE transactions on cybernetics*, **47**(9):2460–2471, 2016.

[24] Manoranjan Dash and Huan Liu. **Consistency-based search in feature selection**. *Artificial intelligence*, **151**(1-2):155–176, 2003.

[25] Jeffrey Dean and Sanjay Ghemawat. **MapReduce: simplified data processing on large clusters**. *Communications of the ACM*, **51**(1):107, jan 2008.

[26] Dayong Deng, Dianxun Yan, and Jiyi Wang. **Parallel reducts based on attribute significance**. In *International Conference on Rough Sets and Knowledge Technology*, pages 336–343. Springer, 2010.

[27] Weiping Ding, Chin-Teng Lin, Senbo Chen, Xiaofeng Zhang, and Bin Hu. **Multiagent-consensus-MapReduce-based attribute reduction using co-evolutionary quantum PSO for big data applications**. *Neurocomputing*, **272**:136–153, 2018.

[28] Weiping Ding, Jiandong Wang, and Jiehua Wang. **Multigranulation consensus fuzzy-rough based attribute reduction**. *Knowledge-Based Systems*, page 105945, 2020.

[29] U Venkata Divya and P. S. V. S. Sai Prasad. **Hashing Supported Iterative MapReduce Based Scalable SBE Reduct Computation**. In *International Conference on Distributed Computing and Internet Technology*, pages 163–170. Springer, 2018.

[30] Wen Sheng Du and Bao Qing Hu. **Dominance-based rough set approach to incomplete ordered information systems**. *Information Sciences*, **346**:106–129, 2016.

[31] Dheeru Dua and Casey Graff. **UCI Machine Learning Repository**. `http://archive.ics.uci.edu/ml`. [Accessed online 2020].

[32] Didier Dubois and Henri Prade. **Rough Fuzzy Sets and Fzzy Rough Sets**. *International Journal of General Systems*, **17**(2-3):191–209, jun 1990.

[33] Didier Dubois and Henri Prade. **Putting rough sets and fuzzy sets together**. In *Intelligent Decision Support*, pages 203–232. Springer, 1992.

[34] Ivo Duntsch and Gunther Gediga. *Rough set data analysis: A road to non-invasive knowledge discovery*. Methodos, 2000.

[35] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. **Twister: a runtime for iterative mapreduce**. In *Proceedings of the 19th ACM international symposium on high performance distributed computing*, pages 810–818. ACM, 2010.

[36] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. **From data mining to knowledge discovery in databases**. *AI magazine*, **17**(3):37–37, 1996.

[37] Jerzy W Grzymala-Busse and Ming Hu. **A comparison of several approaches to missing attribute values in data mining**. In *International Conference on Rough Sets and Current Trends in Computing*, pages 378–385. Springer, 2000.

[38] Susan Gunelius. **The data explosion in 2014 minute by minute–Infographic**. *ACI. Retrieved July*, **29**:2015, 2014.

[39] Isabelle Guyon and André Elisseeff. **An introduction to variable and feature selection**. *Journal of machine learning research*, **3**(Mar):1157–1182, 2003.

[40] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, **207**. Springer, 2008.

[41] Ahmed Hamed, Ahmed Sobhy, and Hamed Nassar. **Distributed approach for computing rough set approximations of big incomplete information systems**. *Information Sciences*, **547**:427–449, 2021.

[42] Liangxiu Han, Chee Sun Liew, Jano Van Hemert, and Malcolm Atkinson. **A generic parallel processing model for facilitating data mining and integration**. *Parallel Computing*, **37**(3):157–171, 2011.

[43] Yasser Hassan and Eiichiro Tazaki. **Combination method of rough set and genetic programming**. *Kybernetes*, 2004.

[44] Aboul Ella Hassanien, Zbigniew Suraj, Dominik Slezak, and Pawan Lingras. *Rough Computing: Theories, Technologies and Applications: Theories, Technologies and Applications*. IGI Global, 2007.

[45] Qing He, Xiaohu Cheng, Fuzhen Zhuang, and Zhongzhi Shi. **Parallel feature selection using positive approximation based on mapreduce**. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on*, pages 397–402. IEEE, 2014.

[46] Q. Hu, L. Zhang, Y. Zhou, and W. Pedrycz. **Large-Scale Multimodality Attribute Reduction With Multi-Kernel Fuzzy Rough Sets**. *IEEE Transactions on Fuzzy Systems*, **26**(1):226–238, 2018.

[47] Qinghua Hu, Witold Pedrycz, Daren Yu, and Jun Lang. **Selecting discrete and continuous features based on neighborhood decision error minimization**. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **40**(1):137–150, 2010.

[48] Qinghua Hu, Daren Yu, Jinfu Liu, and Congxin Wu. **Neighborhood rough set based heterogeneous feature subset selection**. *Information sciences*, **178**(18):3577–3594, 2008.

[49] Qinghua Hu, Daren Yu, and Zongxia Xie. **Information-preserving hybrid data reduction based on fuzzy-rough techniques**. *Pattern Recognition Letters*, **27**(5):414–423, apr 2006.

[50] Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri, and Engelbert Mephu Nguifo. **An experimental survey on big data frameworks**. *Future Generation Computer Systems*, **86**:546–564, sep 2018.

[51] Pelle Jakovits and Satish Narayana Srirama. **Evaluating MapReduce frameworks for iterative scientific computing applications**. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*, pages 226–233. IEEE, 2014.

[52] R. Jensen and Qiang Shen. **New Approaches to Fuzzy-Rough Feature Selection**. *IEEE Transactions on Fuzzy Systems*, **17**(4):824–838, aug 2009.

[53] Richard Jensen. **Rough set-based feature selection: a review**. *Rough computing: theories, technologies and applications*, pages 70–107, 2008.

[54] Richard Jensen and Neil Mac Parthaláin. **Towards scalable fuzzy–rough feature selection**. *Information Sciences*, **323**:1–15, dec 2015.

[55] Jan Komorowski, Zdzislaw Pawlak, Lech Polkowski, and Andrzej Skowron. **Rough sets: A tutorial**. *Rough fuzzy hybridization: A new trend in decision-making*, pages 3–98, 1999.

[56] L. Kong, W. Qu, J. Yu, H. Zuo, G. Chen, F. Xiong, S. Pan, S. Lin, and M. Qiu. **Distributed Feature Selection for Big Data Using Fuzzy Rough Sets**. *IEEE Transactions on Fuzzy Systems*, **28**(5):846–857, 2020.

[57] Marzena Kryszkiewicz. **Rough set approach to incomplete information systems**. *Information sciences*, **112**(1-4):39–49, 1998.

[58] Marzena Kryszkiewicz. **Rules in incomplete information systems**. *Information Sciences*, **113**(3):271 – 292, 1999.

[59] Tianrui Li, Chuan Luo, Hongmei Chen, and Junbo Zhang. **PICKT: a solution for big data analysis**. In *International Conference on Rough Sets and Knowledge Technology*, pages 15–25. Springer, 2015.

[60] Jiye Liang, Feng Wang, Chuangyin Dang, and Yuhua Qian. **An efficient rough feature selection algorithm with a multi-granulation view**. *International Journal of Approximate Reasoning*, **53**(6):912–926, 2012.

[61] Pawan Lingras. **Comparison of neofuzzy and rough neural networks**. *Information Sciences*, **110**(3-4):207–215, 1998.

[62] Pawan Lingras and Cedric Davies. **Applications of rough genetic algorithms**. *Computational Intelligence*, **17**(3):435–445, 2001.

[63] Huan Liu and Lei Yu. **Toward integrating feature selection algorithms for classification and clustering**. *IEEE Transactions on knowledge and data engineering*, **17**(4):491–502, 2005.

[64] Chuan Luo, Tianrui Li, and Zhang Yi. **An Incremental Feature Selection Approach Based on Information Entropy for Incomplete Data**. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 483–488. IEEE, 2019.

[65] Zhigang Ma, Feiping Nie, Yi Yang, Jasper RR Uijlings, and Nicu Sebe. **Web image annotation via subspace-sparsity collaborated feature selection**. *IEEE Transactions on Multimedia*, **14**(4):1021–1030, 2012.

[66] Neil Seosamh MacParthaláin and Qiang Shen. **On rough sets, their recent extensions, and applications**. *Knowledge Engineering Review*, **25**(4):365–395, 2010.

[67] DQ Miao, Yan Zhao, YY Yao, HX Li, and FF Xu. **Relative reducts in consistent and inconsistent decision tables of the Pawlak rough set model**. *Information Sciences*, **179**(24):4140–4150, 2009.

[68] Duo-Qian Miao and Gui-Rong Hu. **A heuristic algorithm for reduction of knowledge**. *Journal of computer research and development*, **36**(6):681–684, 1999.

[69] Fan Min, Huaping He, Yuhua Qian, and William Zhu. **Test-cost-sensitive attribute reduction**. *Information Sciences*, **181**(22):4928–4942, 2011.

[70] Michinori Nakata and Hiroshi Sakai. **Rough sets handling missing values probabilistically interpreted**. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 325–334. Springer, 2005.

[71] DV Nguyen, K Yamada, and M Unehara. **Knowledge reduction in incomplete decision tables using Probabilistic Similarity-Based Rough set Model**. In *12thInternational Symposium on Advanced Intelligent Systems (ISIS 2011)*, pages 147–150, 2011.

[72] Peng Ni, Suyun Zhao, Xizhao Wang, Hong Chen, and Cuiping Li. **PARA: A positive-region based attribute reduction accelerator**. *Information Sciences*, **503**:533–550, nov 2019.

[73] P. S. V. S Sai Prasad and Raghavendra Rao Chillarige. **Novel Granular Framework for Attribute Reduction in Incomplete Decision Systems**. In *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*, pages 188–201. Springer, 2012.

[74] P. S. V. S. Sai Prasad and Chillarige Raghavendra Rao. **Extensions to iQuickReduct**. In *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*, pages 351–362. Springer, 2011.

[75] Neil Mac Parthaláin and Richard Jensen. **Unsupervised fuzzy-rough set-based dimensionality reduction**. *Information Sciences*, **229**:106–121, apr 2013.

[76] Neeli Lakshmi Pavani, Pandu Sowkuntla, K. Swarupa Rani, and P. S. V. S. Sai Prasad. **Fuzzy Rough Discernibility Matrix Based Feature Subset Selection With MapReduce**. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pages 389–394. IEEE, oct 2019.

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

[78] Zdzislaw Pawlak and Rough Sets. **Theoretical aspects of reasoning about data**. *Kluwer, Netherlands*, 1991.

[79] Zdzisław Pawlak and Andrzej Skowron. **Rudiments of rough sets**. *Information sciences*, **177**(1):3–27, 2007.

[80] Lech Polkowski. *Rough sets*. Springer, 2002.

[81] Jin Qian, Duoqian Miao, Zehua Zhang, and Xiaodong Yue. **Parallel attribute reduction algorithms using MapReduce**. *Information Sciences*, **279**:671–690, 2014.

[82] Jin Qian, Min Xia, and Xiaodong Yue. **Parallel knowledge acquisition algorithms for big data using MapReduce**. *International Journal of Machine Learning and Cybernetics*, **9**(6):1007–1021, 2018.

[83] Yuhua Qian, Jiye Liang, Witold Pedrycz, and Chuangyin Dang. **Positive approximation: an accelerator for attribute reduction in rough set theory**. *Artificial Intelligence*, **174**(9-10):597–618, 2010.

[84] Yuhua Qian, Jiye Liang, Witold Pedrycz, and Chuangyin Dang. **An efficient accelerator for attribute reduction from incomplete data in rough set framework**. *Pattern Recognition*, **44**(8):1658–1670, 2011.

[85] Yuhua Qian, Qi Wang, Honghong Cheng, Jiye Liang, and Chuangyin Dang. **Fuzzy-rough feature selection accelerator**. *Fuzzy Sets and Systems*, **258**:61–78, jan 2015.

[86] Anna Maria Radzikowska and Etienne E. Kerre. **A comparative study of fuzzy rough sets**. *Fuzzy Sets and Systems*, **126**(2):137–155, mar 2002.

[87] Mohammad M Rahman, Dominik Ślęzak, and Jakub Wróblewski. **Parallel island model for attribute reduction**. In *International Conference on Pattern Recognition and Machine Intelligence*, pages 714–719. Springer, 2005.

[88] Sergio Ramírez-Gallego, Iago Lastra, David Martínez-Rego, Verónica Bolón-Canedo, José Manuel Benítez, Francisco Herrera, and Amparo Alonso-Betanzos. **Fast-mRMR: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data**. *International Journal of Intelligent Systems*, **32**(2):134–152, 2017.

[89] Sergio Ramírez-Gallego, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, José Manuel Benítez, Amparo Alonso-Betanzos, and Francisco Herrera. **An information theory-based feature selection framework for big data under apache spark**. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, **48**(9):1441–1453, 2018.

[90] Muhammad Summair Raza and Usman Qamar. **A parallel rough set based dependency calculation method for efficient feature selection**. *Applied Soft Computing*, **71**:1020–1034, oct 2018.

[91] P. S. V. S. Sai Prasad and C. Raghavendra Rao. **An Efficient Approach for Fuzzy Decision Reduct Computation**. In *Transactions on Rough Sets XVII*, pages 82–108. Springer Berlin Heidelberg, 2014.

[92] P. S. V. S. Sai Prasad, H. Bala Subrahmanyam, and Praveen Kumar Singh. **Scalable IQRA_IG Algorithm: An Iterative MapReduce Approach for Reduct Computation**. In *Distributed Computing and Internet Technology*, pages 58–69. Springer International Publishing, nov 2016.

[93] Changxing Shang, Min Li, Shengzhong Feng, Qingshan Jiang, and Jianping Fan. **Feature selection via maximizing global information gain for text classification**. *Knowledge-Based Systems*, **54**:298–309, 2013.

[94] Qiang Shen and Richard Jensen. **Rough sets, their extensions and applications**. *International Journal of Automation and Computing*, **4**(3):217–228, 2007.

[95] Juwei Shi, Yunjie Qiu, Umar Farooq Minhas, Limei Jiao, Chen Wang, Berthold Reinwald, and Fatma Özcan. **Clash of the titans: Mapreduce vs. spark for large scale data analytics**. *Proceedings of the VLDB Endowment*, **8**(13):2110–2121, 2015.

[96] Praveen Kumar Singh and P. S. V. S Sai Prasad. **Scalable quick reduct algorithm: Iterative mapreduce approach**. In *Proceedings of the 3rd IKDD Conference on Data Science, 2016*, page 25. ACM, 2016.

[97] Andrzej Skowron, Jan Komorowski, Zdzislaw Pawlak, and Lech Polkowski. **Rough sets perspective on data and knowledge**. In *Handbook of data mining and knowledge discovery*, pages 134–149. 2002.

[98] Andrzej Skowron and Cecylia Rauszer. **The discernibility matrices and functions in information systems**. In *Intelligent decision support*, pages 331–362. Springer, 1992.

[99] Roman Slowinski and Daniel Vanderpooten. **A generalized definition of rough approximations based on similarity**. *IEEE Transactions on knowledge and Data Engineering*, **12**(2):331–336, 2000.

[100] Pandu Sowkuntla, Sravya Dunna, and PSVS Sai Prasad. **MapReduce based parallel attribute reduction in Incomplete Decision Systems**. *Knowledge-Based Systems*, **213**:106677, 2021.

[101] Pandu Sowkuntla and PSVS Sai Prasad. **MapReduce based parallel fuzzy-rough attribute reduction using discernibility matrix**. *Applied Intelligence*, pages 1–20, 2021.

[102] Pandu Sowkuntla and P. S. V. S. Sai Prasad. **MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme**. *Knowledge-Based Systems*, **189**:105104, feb 2020.

[103] Ashwin Srinivasan, Tanveer A Faruquie, and Sachindra Joshi. **Data and task parallelism in ILP using MapReduce**. *Machine learning*, **86**(1):141–168, 2012.

[104] Tomasz Strakowski and Henryk Rybiński. **A new approach to distributed algorithms for reduct calculation**. In *Transactions on Rough Sets IX*, pages 365–378. Springer, 2008.

[105] Xian-He Sun and John L Gustafson. **Toward a better parallel performance metric**. *Parallel Computing*, **17**(10-11):1093–1109, 1991.

[106] Robert Susmaga. **Tree-like parallelization of reduct and construct computation**. In *International Conference on Rough Sets and Current Trends in Computing*, pages 455–464. Springer, 2004.

[107] K Thangavel and A Pethalakshmi. **Dimensionality reduction based on rough set theory: A review**. *Applied Soft Computing*, **9**(1):1–12, 2009.

[108] Nguyen Ngoc Thuy and Sartra Wongthanavasu. **An efficient stripped cover-based accelerator for reduction of attributes in incomplete decision tables**. *Expert Systems with Applications*, **143**:113076, 2020.

[109] Arizona State University. **Feature Selection data sets:http://featureselection.asu.edu/datasets.php**. 2014.

[110] Changzhong Wang, Yang Huang, Mingwen Shao, and Xiaodong Fan. **Fuzzy rough set-based attribute reduction using distance measures**. *Knowledge-Based Systems*, **164**:205–212, jan 2019.

[111] Feng Wang and Jiye Liang. **An efficient feature selection algorithm for hybrid data**. *Neurocomputing*, **193**:33–41, 2016.

[112] Guoyin Wang. **Extension of rough set under incomplete information systems**. In *2002 IEEE World Congress on Computational Intelligence. 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02. Proceedings (Cat. No. 02CH37291)*, **2**, pages 1098–1103. IEEE, 2002.

[113] D Randall Wilson and Tony R Martinez. **Improved heterogeneous distance functions**. *Journal of artificial intelligence research*, **6**:1–34, 1997.

[114] Yong Yang, Zhengrong Chen, Zhu Liang, and Guoyin Wang. **Attribute reduction for massive data based on rough set theory and MapReduce**. In *International Conference on Rough Sets and Knowledge Technology*, pages 672–678. Springer, 2010.

[115] Yiyu Yao. **The two sides of the theory of rough sets**. *Knowledge-Based Systems*, **80**:67–77, 2015.

[116] Yiyu Yao, Yan Zhao, and Jue Wang. **On reduct construction algorithms**. In *Transactions on computational science II*, pages 100–117. Springer, 2008.

[117] Jin Ye, Jianming Zhan, Weiping Ding, and Hamido Fujita. **A novel fuzzy rough set model with fuzzy neighborhood operators**. *Information Sciences*, **544**:266–297, 2021.

[118] Lotfi A Zadeh. **Fuzzy sets**. In *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh*, pages 394–432. World Scientific, 1996.

[119] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. **Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing**. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 15–28, 2012.

[120] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. **Spark: Cluster computing with working sets.** *HotCloud*, **10**(10-10):95, 2010.

[121] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. **Apache spark: a unified engine for big data processing**. *Communications of the ACM*, **59**(11):56–65, 2016.

[122] Anping Zeng, Tianrui Li, Dun Liu, Junbo Zhang, and Hongmei Chen. **A fuzzy rough set approach for incremental feature selection on hybrid information systems**. *Fuzzy Sets and Systems*, **258**:39–60, 2015.

[123] Junhai Zhai, Yuanyuan Gao, Mengyao Zhai, and Xizhao Wang. **Rough set model and its eight extensions**. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3512–3517. IEEE, 2011.

[124] Junbo Zhang, Tianrui Li, and Yi Pan. **Parallel large-scale attribute reduction on cloud systems**. *arXiv preprint arXiv:1610.01807*, 2016.

[125] Junbo Zhang, Tianrui Li, Da Ruan, Zizhe Gao, and Chengbing Zhao. **A parallel method for computing rough set approximations**. *Information Sciences*, **194**:209–223, jul 2012.

[126] Junbo Zhang, Jian-Syuan Wong, Yi Pan, and Tianrui Li. **A parallel matrix-based method for computing approximations in incomplete information systems**. *IEEE Transactions on Knowledge and Data Engineering*, **27**(2):326–339, 2014.

[127] Xiao Zhang, Changlin Mei, Degang Chen, and Jinhai Li. **Feature selection in mixed data: A method using a novel fuzzy rough set-based information entropy**. *Pattern Recognition*, **56**:1–15, 2016.

[128] Xiao Zhang, Changlin Mei, Degang Chen, and Yanyan Yang. **A fuzzy rough set-based feature selection method using representative instances**. *Knowledge-Based Systems*, **151**:216–229, jul 2018.

[129] Hua Zhao and Keyun Qin. **Mixed feature selection in incomplete decision table**. *Knowledge-Based Systems*, **57**:181–190, 2014.

[130] Suyun Zhao, Hong Chen, Cuiping Li, Xiaoyong Du, and Hui Sun. **A Novel Approach to Building a Robust Fuzzy Rough Classifier**. *IEEE Transactions on Fuzzy Systems*, **23**(4):769–786, aug 2015.

[131] Weizhong Zhao, Huifang Ma, and Qing He. **Parallel k-means clustering based on mapreduce**. In *IEEE international conference on cloud computing*, pages 674–679. Springer, 2009.

[132] Yan Zhao, Yiyu Yao, and Feng Luo. **Data analysis based on discernibility and indiscernibility**. *Information Sciences*, **177**(22):4959–4976, 2007.

[133] Bing Zhu, Changzheng He, and Panos Liatsis. **A robust missing value imputation method for noisy data**. *Applied Intelligence*, **36**(1):61–74, 2012.

[134] Huasheng Zou and Changsheng Zhang. **Efficient Algorithm for Knowledge Reduction in Incomplete Information System**. *Journal of Computational Information Systems*, **8**(6):2531–2538, 2012.

Explorations into MapReduce based Parallel Reduct Computation

by Pandu Sowkuntla

Submission date: 24-Sep-2021 11:04AM (UTC+0530)
Submission ID: 1656255732
File name: PhD-Thesis_15MCPC20.pdf (1.1M)
Word count: 53562
Character count: 263930

Explorations into MapReduce based Parallel Reduct Computation

ORIGINALITY REPORT

45%	4%	45%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

Overall similarity score : 45 - (15+15+12) = 3%.

Pendu 24/09/2021

1	Pandu Sowkuntla, P. S. V. S. Sai Prasad. "MapReduce based parallel fuzzy-rough attribute reduction using discernibility matrix", Applied Intelligence, 2021 Publication This publication is by the student	15%
2	Pandu Sowkuntla, Sravya Dunna, P.S.V.S. Sai Prasad. "MapReduce based parallel attribute reduction in Incomplete Decision Systems", Knowledge-Based Systems, 2021 Publication This publication is by the student	15%
3	Pandu Sowkuntla, P.S.V.S. Sai Prasad. "MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme", Knowledge-Based Systems, 2020 Publication This publication is by the student	12%
4	Submitted to University of Hyderabad, Hyderabad Student Paper	<1%
5	dokumen.pub Internet Source	

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

Pendu 24/09/2021

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

Pendu 24/09/2021

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

Pendu 24/09/2021

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

Overall similarity score: 45-(15+15+12) = 3%

-pen-

24/09/2021

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

6	"Rough Sets and Knowledge Technology", Springer Science and Business Media LLC, 2014 Publication	<1%
7	"Rough Sets", Springer Science and Business Media LLC, 2018 Publication	<1%
8	Jin Qian, Duoqian Miao, Zehua Zhang, Xiaodong Yue. "Parallel attribute reduction algorithms using MapReduce", Information Sciences, 2014 Publication	<1%
9	link.springer.com Internet Source	<1%
10	tjzhifei.github.io Internet Source	<1%
11	Chucai Zhang, Jianhua Dai, Jiaolong Chen. "Knowledge granularity based incremental attribute reduction for incomplete decision systems", International Journal of Machine Learning and Cybernetics, 2020 Publication	<1%
12	Wenhao Shu, Hong Shen. "Incremental feature selection based on rough set in	<1%

overall similarity score : 45 - (15+15+12) = 3%.

24/09/202

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
derabad-46. (India)

dynamic incomplete data", Pattern Recognition, 2014
Publication

13	Neeli Lakshmi Pavani, Pandu Sowkuntla, K. Swarupa Rani, P.S.V.S Sai Prasad. "Fuzzy Rough Discernibility Matrix Based Feature Subset Selection With MapReduce", TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019 Publication	<1 %
14	medium.com Internet Source	<1 %
15	Lecture Notes in Computer Science, 2015. Publication	<1 %
16	vtso.geol.vt.edu Internet Source	<1 %
17	"Multi-disciplinary Trends in Artificial Intelligence", Springer Science and Business Media LLC, 2012 Publication	<1 %
18	"Rough Sets and Knowledge Technology", Springer Science and Business Media LLC, 2011 Publication	<1 %
19	"Rough Sets and Current Trends in Soft Computing", Springer Science and Business Media LLC, 2014	<1 %

Overall similarity score: (45-(15+15+12)= 3%)

pevs-gf.

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46 (India)
2/9/2021

	Publication	
20	mafiadoc.com Internet Source	<1 %
21	"Distributed Computing and Internet Technology", Springer Science and Business Media LLC, 2017 Publication	<1 %
22	Lecture Notes in Computer Science, 2014. Publication	<1 %
23	"Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing", Springer Science and Business Media LLC, 2013 Publication	<1 %
24	Submitted to National Open University of Nigeria Student Paper	<1 %
25	Wenhao Shu, Hong Shen. "Incremental Attribute Reduction in Incomplete Decision Systems", 2012 Fifth International Symposium on Parallel Architectures, Algorithms and Programming, 2012 Publication	<1 %
26	usyd.qyjohn.net Internet Source	<1 %
27	www.gartner.com Internet Source	<1 %

Overall similarity score : 45 - (15+15+12) = 3%.

24/09/2021

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

28	Submitted to University of Information Technology, Yangon Student Paper	<1%
29	"Distributed Computing and Internet Technology", Springer Science and Business Media LLC, 2018 Publication	<1%
30	ribuni.uni.edu.ni Internet Source	<1%
31	www.rug.nl Internet Source	<1%
32	intellipaat.com Internet Source	<1%
33	"Rough Sets", Springer Science and Business Media LLC, 2017 Publication	<1%
34	Sai Prasad P.S.V.S., Raghavendra Rao Chillarige. "Chapter 18 Novel Granular Framework for Attribute Reduction in Incomplete Decision Systems", Springer Science and Business Media LLC, 2012 Publication	<1%
35	dx.doi.org Internet Source	<1%
36	ri.ues.edu.sv Internet Source	<1%

Overall similarity score : 45 - (15+15+12) = 3%.

24/09/2021

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India

37	tel.archives-ouvertes.fr Internet Source	<1%
38	"Rough Sets and Knowledge Technology", Springer Science and Business Media LLC, 2008 Publication	<1%
39	epdf.tips Internet Source	<1%
40	www.sciencegate.app Internet Source	<1%
41	researchr.org Internet Source	<1%
42	www.dezyre.com Internet Source	<1%
43	"Rough Sets and Knowledge Technology", Springer Nature, 2011 Publication	<1%
44	docplayer.org Internet Source	<1%
45	"Rough Sets and Knowledge Technology", Springer Nature, 2013 Publication	<1%
46	Submitted to Free University of Bolzano Student Paper	<1%
47	benthamopen.com	

Overall Similarity Score: 45 - (15+15+12) = 31.

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Hyderabad-46. (India)

24/09/2021

	Internet Source	<1 %
48	Hao Ge, Longshu Li, Yi Xu, Chuanjian Yang. "Quick general reduction algorithms for inconsistent decision tables", International Journal of Approximate Reasoning, 2017 Publication	<1 %
49	Submitted to University of Bradford Student Paper	<1 %
50	hdl.handle.net Internet Source	<1 %
51	res.mdpi.com Internet Source	<1 %
52	www.informatica.si Internet Source	<1 %
53	Submitted to Indian Institute of Technology, Kharagpure Student Paper	<1 %
54	Meng, Z.. "A fast approach to attribute reduction in incomplete decision systems with tolerance relation-based rough sets", Information Sciences, 20090720 Publication	<1 %
55	Wu, W.Z.. "Attribute reduction based on evidence theory in incomplete decision systems", Information Sciences, 20080301	<1 %

Overall similarity score: 45-(15+15+12) = 3%.

Associate Professor
School of CIS
Prof. C.R. Rao Road,
Central University
Gachibowli-46. (India)

24/09/2024

	Publication	
56	Wenhao Shu, Wenbin Qian. "A fast approach to attribute reduction from perspective of attribute measures in incomplete decision systems", Knowledge-Based Systems, 2014 Publication	<1 %
57	labreserve.ucslabs.sfu.ca Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		