Knowledge-Based Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme☆

Pandu Sowkuntla *, P.S.V.S. Sai Prasad

School of Computer and Information Sciences, University of Hyderabad, Hyderabad, 500046, Telangana, India

ARTICLE INFO

Article history:
Received 7 February 2019
Received in revised form 7 October 2019
Accepted 9 October 2019
Available online xxxx

Keywords:
Rough sets
MapReduce
Apache spark
Reduct
Horizontal partitioning
Vertical partitioning
Feature subset selection

ABSTRACT

In the last few decades, rough sets have evolved to become an essential technology for feature subset selection by way of reduct computation in categorical decision systems. In recent years with the proliferation of MapReduce for distributed/parallel algorithms, several scalable reduct computation algorithms have been developed in this field for large-scale decision systems using MapReduce. The existing MapReduce based reduct computation approaches use horizontal partitioning (division in object space) of the dataset into the nodes of the cluster, requiring a complicated shuffle and sort phase. In this work, we propose an algorithm MR_IQRA_VP which is designed using vertical partitioning (division in attribute space) of the dataset with a simplified shuffle and sort phase of the MapReduce framework. MR_IQRA_VP is a distributed/parallel implementation of the Improved Quick Reduct Algorithm (IQRA_IG) and is implemented using iterative MapReduce framework of Apache Spark. We have done an extensive comparative study through experimentation on benchmark decision systems using existing horizontal partitioning based reduct computation algorithms. Through experimental analysis, along with theoretical validation, we have established that MR_IQRA_VP is suitable and scalable to datasets of larger size attribute space and moderate object space prevalent in the areas of Bioinformatics and Web mining.

1. Introduction

Over the last few years, there has been an exponential increase in data generation per day [1,2]. Mining knowledge from the big data is a challenging task because of the uncertainty and inconsistency in the data. Feature subset selection is one of the approaches that helps in exploiting the data redundancy to reduce the uncertainty from big data [3–6]. Feature subset selection, also known as variable selection (attribute reduction), is the process of selecting a minimal subset of attributes that provide the same classification ability as the whole attributes.

In 1982, Pawlak [7] introduced the rough set theory, which has evolved as a useful tool for attribute reduction (also known as reduct computation) [8–11]. Over the last few decades, several different algorithms have been proposed for achieving an efficient attribute reduction [3–5,12–18]. 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) [17] by adding the features of *handling the trivial ambiguous situation*, *granular refinement*, and *positive region removal*.

The necessary computations in rough set based reduct computation (with sequential forward selection method of feature selection) algorithm are the evaluation of attribute subsets for attribute dependency (*gamma*) measure. By using the sequential forward selection strategy of attribute selection, the subsets considered are formed incrementally based on the attributes selected into reduct in previous iterations. The computations required for evaluating attributes' subset are the construction of induced equivalence classes (also called as granules). These granules are further categorized into lower approximation (positive region) or boundary region. As given in the algorithm IQRA_IG [17], the granular refinement aspect embeds an incremental approach for granular space construction by minimizing the construction complexity when the process is repeated. The aspect of positive region removal restricts the future computations to granules falling into boundary region resulting in a decrease of space utilization in successive iterations. IQRA_IG proved to be an efficient algorithm among stand-alone reduct computation algorithms, because of the granular refinement and positive region removal features (detailed explanation of granular refinement, positive region removal and related concepts are given in later sections).

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2019.105104.

* Corresponding author.
E-mail addresses: pandu.sowkuntla@uohyd.ac.in (P. Sowkuntla), saics@uohyd.ernet.in (P.S.V.S. Sai Prasad).

https://doi.org/10.1016/j.knosys.2019.105104
0950-7051/

Earlier research works have concluded that parallel computing is the right solution for attribute reduction in large-scale datasets. Several parallel techniques were combined with traditional rough set based methods to improve the reduction process [12,19–25]. Furthermore, to improve the scalability, different MapReduce based distributed/parallel algorithms were proposed for reduct computation [6,26–34]. All these MapReduce based algorithms were implemented on the Apache Hadoop framework [35]. But implementing iterative parallel algorithms on the Hadoop platform was found to be inefficient. The lapse was because of the problem of frequent loading of the data into distributed memory.

In order to overcome the problems of Hadoop MapReduce framework, many authors proposed parallel algorithms [28,30,31,34] based on in-memory iterative MapReduce frameworks such as Apache Spark [36], and Twister [37]. J. Zhang et al. proposed Parallel Large-Scale Attribute Reduction (PLAR) [28] algorithm based on Apache Spark framework. Sai Prasad et al. developed a scalable IQRA_IG algorithm (IN_MRIQRA_IG) [30], which is a parallel version of IQRA_IG [17], and it is implemented on the Twister framework.

From the existing MapReduce based distributed/parallel reduct computation algorithms such as IN_MRIQRA_IG (based on Twister), PLAR (based on Apache Spark), and PFSPA (based on Hadoop), we have come to know that all the algorithms use *horizontal partitioning scheme* for partitioning the input dataset. With this scheme, distribution of the dataset is done over object space, and the data partitions are located in different nodes in the cluster. This means that if the dataset is horizontally partitioned to the nodes of the cluster, every node has information of all the attributes over a subset of objects. Thus, the granules 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 table. We have observed that as attribute space size is increases, the running time of horizontal partitioning based reduct computation algorithm also grows significantly. This has been especially observed while working with high dimensional datasets of Bioinformatics, i.e., microarray datasets.

Horizontal partitioning scheme based algorithms such as PFSPA [33] and IN_MRIQRA_IG [30] have incorporated positive region removal feature. But, they have not incorporated granular refinement feature. It is because in horizontal partitioning scheme, a granule is split across the nodes of the cluster that does not facilitate computation of refined granules without global consolidation (this leads to a lot of data movement across the nodes). Hence at present, no horizontal partitioning based MapReduce approaches for reduct computation incorporate granular refinement. S. Ramirez-Gallego et al. [38] used *vertical partitioning scheme* to distribute the input dataset in their *spark-info-theoretic-feature-selection* framework to deal with high dimensional datasets. In their framework, the utility of vertical partitioning scheme is demonstrated for feature ranking using MapReduce.

In our paper, we have tried to investigate the relevance of vertical partitioning scheme in MapReduce based feature subset selection using rough sets. Here, we propose a MapReduce based distributed/parallel algorithm MR_IQRA_VP in which vertical partitioning scheme will be used for partitioning the input dataset. In a vertical partitioning scheme, the dataset is split over attribute space. It overcomes the problems involved in horizontal partitioning scheme. The complexity and difficulty of incorporating positive region removal and granular refinement features can be avoided using this scheme. Incorporation of these features further improves the efficiency of the proposed algorithm. The relevance and also limitations of the proposed vertical partitioning based MR_IQRA_VP algorithm is experimentally studied, and inferences were obtained through comparative analysis with horizontal partitioning scheme based algorithms.

Organization of the remaining part of this paper is given as follows. Section 2 deals with preliminary concepts in rough set theory and MapReduce programming model. Section 3 highlights the existing works and their problems. The proposed method and its features that overcome the challenges in earlier works are discussed in Section 4. In Section 5, experimental results, their comparison with existing reduct computation algorithms' results, efficiency, and performance of the proposed algorithm is demonstrated. Finally, this paper is concluded in Section 6 along with future plans.

2. Preliminaries

In this section, the fundamentals of rough set theory, attribute reduction, and basic concepts of MapReduce programming model are discussed.

2.1. Rough set theory

Pawlak provided foundations of rough sets in [7] and basics related to reduct computation are given in [8,10,39]. A complete symbolic decision table defined as,

$$Dt = (U, C \cup D, \{V_a, f_a\}_{a \in C \cup D}) \tag{1}$$

where, $U = \{x_1, x_2, \ldots .x_m\}$ is a finite nonempty set of objects, $C = \{a_1, a_2, \ldots .a_n\}$ is a finite nonempty set of conditional attributes, $D = \{d_1, d_2, \ldots .d_q\}$ is a set of decision attributes that represent classes of objects. In this paper, 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.

For the given decision table Dt, let $P \subseteq C$, there is an associated equivalence relation, also known as *indiscernibility relation* $IND(P)$, given by Pawlak [7]:

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

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 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 also called as *approximation space* or *granular space*, and each equivalence class in U/P is also called as a *granule* (we use the term granule instead of an equivalence class in the rest of the paper). The concept of indiscernibility relation is the main idea for rough set based attribute reduction. 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. $\bigcup_{gr \in U/P} gr = U$

As defined in [7], for the given decision table Dt, let $P \subseteq C$ and $X \subseteq U$. The concept X can be approximated using only the

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

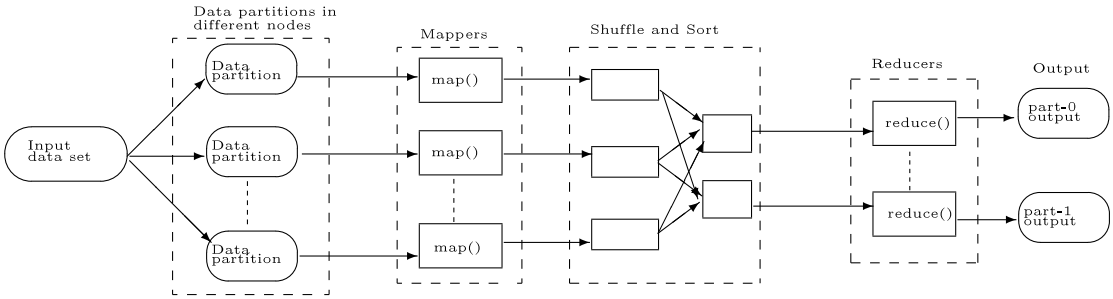


Fig. 1. Overview of MapReduce programming model.

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.

$$\underline{P}(X) = \{x \in U | [x]_P \subseteq X\},\ \overline{P}(X) = \{x \in U | [x]_P \cap X \neq \phi\} \tag{3}$$

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

From [8,10], for the given decision table Dt, the *positive region* is defined as,

$$POS_P(\{d\}) = \bigcup_{X \in U/\{d\}} \underline{P}(X) \tag{4}$$

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.

For the given decision table Dt, the *degree of dependency* of $\{d\}$ over P is given by [8,10],

$$\gamma_P(\{d\}) = \frac{|POS_P(\{d\})|}{|U|} \tag{5}$$

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. Values in $0 < \gamma_P(\{d\}) < 1$ indicate the partial dependency.

2.1.1. Reduct

As given in [8,10], for the given decision table Dt, *reduct* R is a minimal subset of the conditional attribute set C, such that,

$$\gamma_R(\{d\}) = \gamma_C(\{d\}),\ where\ \gamma_{R'}(\{d\}) < \gamma_C(\{d\})\ for\ any\ R' \subset R \tag{6}$$

Intuitively, the reduct R is a minimal subset of conditional attributes that preserves the original classification as defined by conditional attribute set C. A given decision table contains many reducts; Computation of optimal reduct is an NP-hard problem [40]. Thus, many researchers proposed heuristic based solutions to compute the single reduct with polynomial time complexity.

Reduct generation (feature subset selection) can be done in two control strategies, These strategies are: (i) *Sequential Forward Selection (SFS)*, and (ii) *Sequential Backward Elimination (SBE)*. In SFS strategy, the reduct generation starts by initializing the reduct as an empty set, and attributes are added incrementally. Therefore the SFS strategy results in a superset of reduct containing redundant attributes. In SBE strategy, the reduct starts by having all the attributes in it, and the redundant attributes are removed in computing the reduct. It is observed that the SBE strategy guarantees minimal reduct. But the computational efficiency of SFS strategy over SBE strategy makes it suitable for building scalable MapReduce approaches for large-scale decision tables.

In rough set theory, approaches in each of the reduct generation methods are categorized into (i) *Degree of dependency* (reduct is computed by using *gamma* (γ) measure), and (ii) *Discernibility matrix* (discernibility matrix is used in the computation of reduct) [41]. The proposed algorithm is based on the degree of dependency approach that follows the SFS control strategy for reduct generation.

2.2. MapReduce programming model

MapReduce [42] is an execution framework for robust and scalable implementation of distributed/parallel algorithms. The framework coordinates three phases: *map, shuffle & sort*, and *reduce* for processing massive amounts of data on large clusters. The underlying data structure of any MapReduce framework is a $\langle key, value \rangle$ pair. The programmer job is to provide map and reduce functions with the following signatures: $map : \langle k_1, v_1 \rangle \rightarrow [\langle k_2, v_2 \rangle]$, and $reduce : \langle k_2, [v_2] \rangle \rightarrow [\langle k_3, v_3 \rangle]$. The convention $[\ldots]$ denotes a list.

In Fig. 1, the MapReduce programming model is outlined. The execution in any MapReduce framework is done in the following manner. (1) Data input from the distributed file system is dispensed to different mappers located in different nodes as partitions (data splits). Each mapper transforms $\langle k_1, v_1 \rangle$ pair in the associated data partition and produces an arbitrary number of intermediate $\langle k_2, v_2 \rangle$ pairs. (2) Grouping the generated intermediate $\langle k_2, v_2 \rangle$ pairs by the key is achieved by a large scale distributed sort involving all the nodes that execute mapper tasks and all the nodes that will run reducer tasks. This intermediate data must be copied (transferred) over the network, and a lot of communication takes place across the system in this phase of the framework known as *shuffle and sort*. Transferring data and inducing communication across the network lead to a significant burden on the framework. This is because many disk and network I/O operations are required to transfer the entire data. Under big data workloads, minimizing the work of shuffle and sort phase, and distributed coordination are essential to get high performance. (3) Each reducer gets intermediate data in the order sorted by the key k_2. The reducer code is applied to all the values corresponding to the same intermediate key k_2 in producing the output $\langle k_3, v_3 \rangle$ pairs. (4) Driver located in master node runs the program that initiates MapReduce jobs. And, it controls the distributed operations and collects the results from the worker (slave) nodes.

3. Existing sequential and parallel IQRA_IG algorithms

This section provides, an overview of existing work, sequential, and parallel versions of IQRA_IG algorithm and their significant features.

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

3.1. Overview of IQRA_IG algorithm

Sai Prasad et al. proposed IQRA_IG algorithm [17], 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.

In the process of reduct computation, QRA algorithm starts its first iteration with the reduct R as an empty set (ϕ). For each conditional attribute $a \in C$, the *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 $a \in C$. For sorting, Quick sort like comparison sorting based algorithm takes $\bigcirc(|U|log|U|)$ time and Radix sort like linear sorting algorithm takes $\bigcirc(|U|)$ time for forming granules. After the computation of positive region for each conditional attribute $a \in C$, the *gamma* measure (γ) is computed. Attribute for which maximum *gamma* is obtained is included into the reduct R.

In the subsequent iterations of QRA, when reduct is nonempty, the granules need to be computed with $R \cup \{a\}$, $\forall a \in (C - R)$. Granules $U/(R \cup \{a\})$ can be computed by sorting the objects based on their values of $R \cup \{a\}$. 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 3.1.1) in IQRA_IG algorithm. By forming the granules $U/(R \cup \{a\})$, the *gamma* measure $\gamma_{R\cup\{a\}}(\{d\})$ is computed. 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 $\gamma_C(\{d\})$.

3.1.1. Granular refinement

Definition 1. For a decision table $Dt = (U, C\cup\{d\}, \{V_a, f_a\}_{a\in C\cup\{d\}})$, let $Q \subseteq P \subseteq C$, granules U/P is a refinement over U/Q that denoted by $U/P \preceq U/Q$ where,

$$\forall gr \in U/P \Rightarrow \exists gr' \in U/Q \wedge gr \subseteq gr' \tag{7}$$

The property in Definition 1 is the outcome of the indiscernibility relation being an equivalence relation.

Definition 2. For a decision table $Dt = (U, C\cup\{d\}, \{V_a, f_a\}_{a\in C\cup\{d\}})$, let $R \subseteq C$, $a \in (C - R)$ and $U/R = \{gr_1, gr_2, \ldots .gr_r\}$, granular refinement for the computation of $U/(R \cup \{a\})$ is given by,

$$U/(R \cup \{a\}) = Granular\ Refinement(U/R, a),$$
$$where\ Granular\ Refinement(U/R, a) = \bigcup_{i=1}^{r} gr_i/\{a\} \tag{8}$$

By using Definition 2, granular refinement feature 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 $\sum_{i=1}^{r} \bigcirc(|gr_i|log|gr_i|)$. Granular refinement has the computational gains since $\sum_{i=1}^{r} \bigcirc(|gr_i|log|gr_i|) \leq \bigcirc(|R||U|log|U|)$. The time complexity of construction of U/R is $\bigcirc(|R||U|log|U|)$ since each comparison involves $|R|$ attributes. Similarly, computational gains can be achieved using Radix sort technique for forming granules.

3.1.2. 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 $a \in (C - R)$, 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 (Definition 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 (given in Definition 3) [17] or positive approximation [33].

Definition 3. For a decision table $Dt = (U, C\cup\{d\}, \{V_a, f_a\}_{a\in C\cup\{d\}})$, let $R \subseteq C$, and $P\_GR(U/R)$ denotes the positive region granules, then the positive region removal is given by,

$$U/R = NP\_GR(U/R) = U/R - P\_GR(U/R) \tag{9}$$

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 by applying granular refinement as *Granular Refinement* ($NP\_GR(U/R)$, *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.

3.1.3. Handling trivial ambiguous situation

In an iteration of the algorithm, if there is no *gamma* gain, then the selection of an attribute from $(C - R)$ 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 datasets, 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 further details refer [17]).

3.2. Parallel version of IQRA_IG algorithm (IN_MRIQRA_IG)

Sai Prasad et al. proposed MapReduce based parallel version of IQRA_IG as IN_MRIQRA_IG algorithm [30]. 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.

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

3.2.1. Horizontal partitioning based reduct computation

In horizontal partitioning scheme, the data is distributed over the object space. Through this data distribution, every node has information of all the attributes over a subset of objects.

Definition 4. For a decision table $Dt = (U, C \cup \{d\}, \{V_a, f_a\}_{a \in C \cup \{d\}})$, let $Dt = \bigcup_{i=1}^{p} Dt^i$, where $Dt^i = (U^i, C \cup \{d\}, \{V_a, f_a\}_{a \in C \cup \{d\}})$ is ith data split. It satisfies (i) $U = \bigcup_{i=1}^{p} U^i$ (ii) $U^i \cap U^j = \phi$, $\forall i, j \in \{1, 2, \ldots, p\}$ and $i \neq j$.

By Definition 4, decision table Dt 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 ith mapper working on the data partition Dt^i and broadcasted R can construct partial granules $gr^i \in U^i/(R \cup \{a\})$ for all competing attributes $a \in (C - R)$. If gr^i is consistent, then $\langle key, value \rangle$ pair is generated with the key as $\langle a, GS(gr^i) \rangle$ and $value$ as $\langle |gr^i|, d(gr^i) \rangle$. 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 $key = \langle a, GS(gr^i) \rangle$ and $value = \langle 0, -1 \rangle$ 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, $GS(gr^i) = GS(gr^j)$, $\forall i, j \in \{1, 2, \ldots, p\}$, and it follows that $GS(gr) = GS(gr^i)$, $\forall i \in \{1, 2, \ldots, p\}$. 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 $\langle key, value \rangle = \langle a, |gr| \rangle$ pair. But, if the granule is inconsistent then $\langle a, 0 \rangle$ pair is generated. The driver computes $|POS_{R \cup \{a\}}(\{d\})|$, $\forall a \in (C - R)$ based on $\langle key, value \rangle$ 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. 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 $\langle key, value \rangle$ pairs in mapper phase, attribute information of $R \cup \{a\}$, $\forall a \in (C - R)$ is required. The removal of positive region in the proposed work with vertical partitioning scheme is discussed in Section 4.4.1.

3.2.3. Granular refinement

The existing MapReduce based reduct computation approaches [6,25–34] do not incorporate the granular refinement aspect described in Section 3.1.1. In the horizontal partitioning approach, the knowledge of granules is realized at reducer phase as each has partial granules information only. To correctly obtain $U/R, \forall a \in C$ in the first iteration requires *object id* to be placed as part of $value$ portion along with decision information. This results in $|U| * |C|$ amount of data (equals the original dataset size) 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 scheme described in Section 4.4.2 overcomes this limitation of horizontal partitioning scheme.

4. Proposed MapReduce based IQRA using vertical partitioning scheme

This section discusses the proposed MapReduce based IQRA algorithm (MR_IQRA_VP) using vertical partitioning scheme along with its features, advantages and limitations.

4.1. Vertical partitioning of the decision table

In the vertical partitioning scheme, data is distributed over the attribute space. Here, all values of an attribute are available in one record of one data partition located in a node of the cluster. Vertical partitioning scheme is realized by preprocessing the input dataset before supplying it to the algorithm, either locally for datasets fitting in RAM or by using MapReduce approach otherwise. The given dataset 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 datasets used for representing the gene expression data in Bioinformatics [43] are usually stored in rows represent attributes, and hence preprocessing is not required.

The preprocessed data is horizontally partitioned over attribute set C, and each partition represents data pertaining to a subset of C. Each partition requires decision attribute information for subsequent operations, and therefore the decision attribute $\{d\}$ is broadcasted to all the nodes by the driver.

Definition 5. For a decision table $Dt = (U, C \cup \{d\}, \{V_a, f_a\}_{a \in C \cup \{d\}})$, let $Dt = \bigcup_{i=1}^{p} Dt^i$, where $Dt^i = (U, C^i \cup \{d\}, \{V_a, f_a\}_{a \in C^i \cup \{d\}})$ is ith data split. It satisfies (i) $C = \bigcup_{i=1}^{p} C^i$ (ii) $C^i \cap C^j = \phi$, $\forall i, j \in \{1, 2, \ldots, p\}$ and $i \neq j$.

As we can see in Definition 5, with vertical partitioning scheme, the decision table Dt is divided into p sub-decision tables or data splits (data partitions). In Fig. 2 vertical partitioning of the input dataset is shown. A data split Dt^i in a node contains C^i attributes with all the objects U and broadcasted decision attribute $\{d\}$ (as shown in Fig. 2). Let t be the number of nodes in the cluster, and p be the number of data partitions of the given dataset. 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.

4.2. Parallelization of attribute selection

The primary operation of IQRA_IG algorithm [17] 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

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

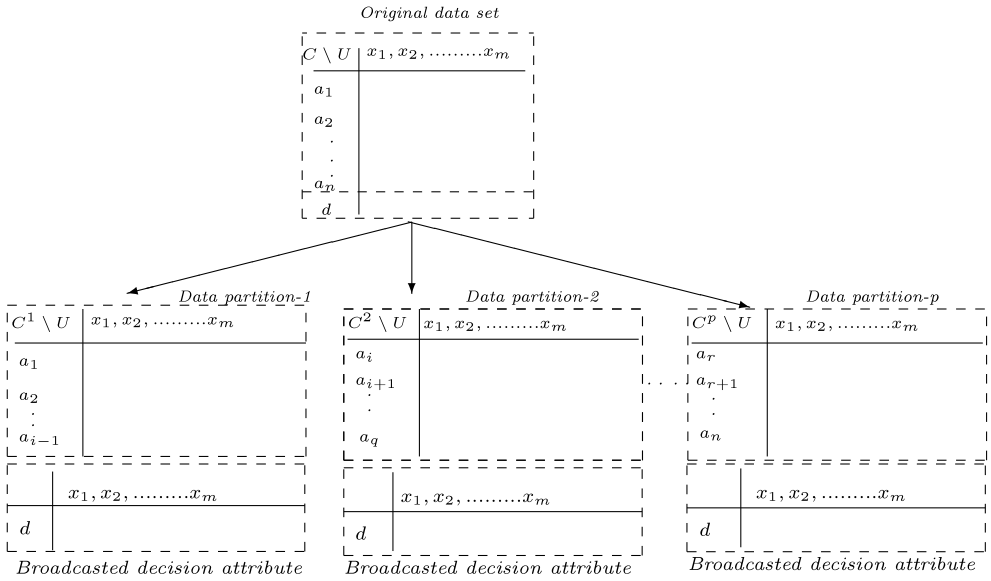


Fig. 2. Vertical data partitioning.

in decision table Dt to parallel attribute selection over vertically partitioned sub decision tables $\{Dt^i\}_{i=1}^{p}$. In IQRA_IG algorithm, over the dataset Dt, the selected next best attribute $a^{best} \in (C-R)$ satisfies the following property.

$$\gamma_{R\cup\{a^{best}\}}(\{d\}) = \max_{a\in(C-R)} \gamma_{R\cup\{a\}}(\{d\}) \tag{10}$$

Using Eq. (5), canceling the same denominator on either side of Eq. (10) results in,

$$\left|POS_{R\cup\{a^{best}\}}(\{d\})\right| = \max_{a\in(C-R)} \left|POS_{R\cup\{a\}}(\{d\})\right| \tag{11}$$

In the proposed approach, as the dataset Dt is available as Dt^i ($i = 1, 2, \ldots, p$) 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 Dt^i, $\forall i = \{1, 2, \ldots, p\}$ as,

$$\left|POS_{R\cup\{a^{best}\}}(\{d\})\right| = \max_{a\in(C-R)} \left|POS_{R\cup\{a\}}(\{d\})\right| = \max\left(\max_{a\in(C^1-R)} \left|POS_{R\cup\{a\}}(\{d\})\right|, \max_{a\in(C^2-R)} \left|POS_{R\cup\{a\}}(\{d\})\right|, \ldots \max_{a\in(C^p-R)} \left|POS_{R\cup\{a\}}(\{d\})\right|\right) \tag{12}$$

Thus, the attribute selection process is equivalent in both standalone and vertical partitioning based distributed approaches.

4.3. MR_IQRA_VP algorithm

In this section, the main (driver) algorithm for proposed work MR_IQRA_VP is given in Algorithm 1. The mapper phase algorithm (Algorithm 2: MR_IQRA_VP-Map) for the local best attribute selection is given in Section 4.3.1. The reducer phase algorithm (Algorithm 3: MR_IQRA_VP-Reduce) for the global best attribute selection is described in Section 4.3.2. And, the computation of $\gamma_C(\{d\})$ for the end condition of the main algorithm (Algorithm 1) is given in Section 4.3.3.

In the driver (from Algorithm 1), initially the dataset Dt is vertically partitioned into Dt^i ($i = 1, 2, \ldots, p$), 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 1: MapReduce based IQRA using vertical partitioning (MR_IQRA_VP)

Input: Input file: Dataset $Dt = (U, C \cup \{d\}, \{V_a, f_a\}_{a\in C\cup\{d\}})$
Output: Reduct Red
1 Distribute input dataset Dt with vertical partitioning over C, and broadcast decision attribute $\{d\}$ into the nodes of the cluster so that each data partition becomes $Dt^i = (U, C^i \cup \{d\}, \{V_a, f_a\}_{a\in C^i\cup\{d\}})$, $\forall i \in \{1, 2, \ldots p\}$ 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 $\gamma_C(\{d\})$
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 2 and reducer computes global best attribute ($best\_Attr$) and its positive count ($bestAttr\_PosCount$) by using Algorithm 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** ($\gamma_{Red} < \gamma_C$) **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** ($\gamma_{Red} == \gamma_C$)
18 Return Red

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 the associated positive region count $bestAttr\_PosCount$ are obtained through invocation of MR_IQRA_VP-Map algorithm followed by MR_IQRA_VP-Reduce algorithm. The next best attribute $best\_Attr$ is included into reduct 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 ($\gamma_{Red} == \gamma_C$) 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 2: MR_IQRA_VP-Map

Input: 1. Data split $Dt^i = (U, C^i \cup \{d\}, \{V_a, f_a\}_{a \in C^i \cup \{d\}})$ with each record as $< key, value > = < attrNo, attr\_Data >$
2. Broadcasted variable: reduct Red
3. Broadcasted non-positive region granules list: $NP\_GR(U/Red)$

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

```
maxPos_Count = 0, localBest_Attr = −1
for each record rec ∈ Dt^i as < attrNo, attr_Data > do
    if attrNo ∉ Red then
        U/(Red ∪ {attrNo}) = GranularRefinement(NP_GR(U/Red), attrNo)
        Compute POS_{Red∪{attrNo}}({d}) using U/(Red ∪ {attrNo})
        pos_Count = |POS_{Red∪{attrNo}}({d})|
        if pos_Count > maxPos_Count then
            localBest_Attr = attrNo
            maxPos_Count = pos_Count
        end
    end
end
Construct < key', value' > pair, where key' = "dummyKey", and
value' = (localBest_Attr, localBest_PosCount)
Emit intermediate < key', value' >
```

Algorithm 3: MR_IQRA_VP-Reduce

Input: $< key, V >$ pair where, key is a "$dummyKey$" (some 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, that is to be added to the reduct, and $bestAttr\_PosCount$ is the best attribute's positive count

```
bestAttr_PosCount = 0, best_Attr = −1
for each value v ∈ V as (localBest_Attr, localBest_PosCount) do
    if localBest_PosCount > bestAttr_PosCount then
        best_Attr = localBest_Attr
        bestAttr_PosCount = localBest_PosCount
    end
end
Construct < key', value' > pair, where key' = best_Attr, and
value' = bestAttr_PosCount
Emit < key', value' >
```

4.3.1. MR_IQRA_VP-Map algorithm

The algorithm MR_IQRA_VP-Map given in Algorithm 2 is invoked in each iteration of the main algorithm of MR_IQRA_VP for selection of next best attribute into reduct Red. The mapper process associated with a data split Dt^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, $attrNo \in (C^i - Red)$, the granules $U/(Red \cup \{attrNo\})$ are computed using $Granular\ Refinement(NP\_GR(U/Red), attrNo)$. 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 $(C^i - Red)$ 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 $\langle key, value \rangle$ pair, where $key = dummyKey$ and $value = (localBest\_Attr, localBest\_PosCount)$.

In this manner, p number of $\langle key, value \rangle$ pairs are generated from decision sub tables Dt^i ($i = 1, 2, \ldots, p$) and participate in shuffle and sort phase. As all of $\langle key, value \rangle$ pairs contain the same $dummyKey$ portion as the key, only a single reducer will be invoked facilitating global best attribute selection.

4.3.2. MR_IQRA_VP-Reduce algorithm

The algorithm MR_IQRA_VP-Reduce given in Algorithm 3 receives $\langle key, V \rangle$ 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 $\langle key', value' \rangle$ pair, where $key' = best\_Attr$ and $value' = bestAttr\_PosCount$.

4.3.3. Computation of $\gamma_C(\{d\})$

In this section, the procedure for computation of $\gamma_C(\{d\})$ is described using vertical partitioning scheme in MR_IQRA_VP algorithm.

The computation of $\gamma_C(\{d\})$ requires construction of U/C and categorizing the granules into $P\_GR(U/C)$, and $NP\_GR(U/C)$. In each mapper, using the decision sub table $Dt^i = (U, C^i \cup \{d\}, \{V_a, f_a\}_{a \in C^i \cup \{d\}})$, $\forall i \in \{1, 2, \ldots, p\}$, one can compute U/C^i, and the granules can be categorized into $P\_GR(U/C^i)$ and $NP\_GR(U/C^i)$. From the explanation of positive region removal in Section 3.1.2, objects of $P\_GR(U/C^i)$ are the objects in $POS_C(\{d\})$. Here, each mapper communicates the information of $NP\_GR(U/C^i)$ to a single reducer. The reducer then computes refinement of $NP\_GR(U/C^i)$, $\forall i \in \{1, 2, \ldots, p\}$ and arrives at $NP\_GR(U/C)$. If $NP\_GR(U/C)$ is empty then $\gamma_C(\{d\}) = 1$ otherwise $POS_C(\{d\})$ is computed as, $POS_C(\{d\}) = U - \bigcup_{gr \in NP\_GR(U/C)} gr$. Based on this positive region $POS_C(\{d\})$, the value of $\gamma_C(\{d\})$ is computed, and communicated to the driver.

4.4. 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 algorithm are discussed.

4.4.1. 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 dataset 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 4.3. 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.

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

4.4.2. Granular refinement

The implementation of granular refinement feature *Granular Refinement*$(NP\_GR(U/R), a)$ of MR_IQRA_VP algorithm is identical to that of IQRA_IG algorithm because the proposed algorithm uses vertical partitioning scheme. In contrast to IQRA_IG, the required information of $NP\_GR(U/R)$ is broadcasted from driver to 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* [44] is used for the same. *HashMap* is a data structure that maintains records of paired data $\langle key, value\rangle$. 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 $\langle key, value\rangle$ 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 $\langle key, value\rangle$ 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 ith mapper, the granules of $U/(R \cup \{a\}),\ \forall a \in (C - R)$ are utilized for the computation of $|POS_{R\cup\{a\}}(\{d\})|,\ \forall a \in (C - R)$. In order to optimize the memory utilization, the memory occupied by $U/(R \cup \{a\}),\ \forall a \in (C - R)$ 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.

4.4.3. Simplification of shuffle and sort phase

Without loss of generality, consider a decision system having k distinct values for each attribute. From Section 3.2.1, 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 $k^{|R\cup\{a\}|},\ \forall a \in (C - R)$. As p denotes the number of data partitions, then a total $p*|C - R|*k^{(|R|+1)}$ size of $\langle key, value\rangle$ 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 our proposed design, as given in Section 4.3.1 of mapper phase, each mapper produces a single $\langle key, value\rangle$ pair corresponding to the local best attribute, and the local best attribute's positive count. This results in a total p size of $\langle key, value\rangle$ 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.

4.4.4. 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 $\bigcirc(|C|^2|U|log|U|)$, and the complexity of an iteration is $\bigcirc(|C - R||U|log|U|)$. In the vertical partitioning scheme, the time complexity of an iteration in mapper is $\bigcirc(\frac{|C-R|}{p}|U|log|U|)$, where p is the number of data partitions. Therefore, for very large object space datasets 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 datasets while being scalable to very large attribute space datasets as the approach is horizontally scalable in attribute space, i.e., very large-scale high dimensional datasets can be handled by addition of nodes to the cluster. Theoretical explanation of this section is validated experimentally in Section 5.4.

Table 1
Cluster configuration of different algorithms.

	MR_IQRA_VP	*PLAR*[a]	*PFSPA*[b]	*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

[a]Cluster configuration as reported in [28].
[b]Cluster configuration as reported in [33].

Table 2
Datasets used in the experiments.

S.No	Dataset	Objects	Attributes	Classes
1	Gisette	6000	5000	2
2	Basehock	1993	4864	2
3	Gene expression Cancer RNA-Seq	801	20561	5
4	Semeion Handwritten Digit	1593	256	10

5. Experimental results

In this section, the efficiency of MR_IQRA_VP algorithm is given with detailed comparative analysis. And, the performance of the proposed algorithm is evaluated experimentally with various metrics. Finally, the relevance of MR_IQRA_VP algorithm is discussed in the last subsection.

5.1. Experimental set up

Apache Spark MapReduce framework has reliable fault tolerance, and support for in-memory computations that are required to run iterative algorithms [45,46]. Thus, the proposed work is implemented on the Apache Spark framework. The proposed algorithm has been executed on a cluster of six nodes, where one node is set as master, and the rest are set as slaves. Each node uses Intel (R) Core (TM) i3-7100 CPU@3.90 GHz processor with four cores, and 8 GB of main memory. All the nodes run on Ubuntu 18.04 LTS operating system. And all the machines are connected via Ethernet (with 1000Mbps speed). Here, each node is installed with Java 1.8.0_171, Apache Spark 2.3.1 and Scala 2.11.4.

The proposed algorithm is compared with the existing MapReduce based distributed/parallel reduct computation algorithms, PLAR [28], PFSPA [33], and Apache Spark version of IN_MRIQRA_IG algorithm [30]. The cluster configuration of these algorithms as reported in the respective publication sources and the configuration of the proposed algorithm are given in Table 1.

In Section 4.4.4, we have described that the proposed algorithm is suitable for the datasets having a moderate size of

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

Table 3
The obtained reduct of MR_IQRA_VP for different datasets.

S.No	Dataset	Reduct	γ_C	γ_R
1	Gisette	{3058, 1523, 4734, 1923, 4165, 4272, 1555, 3708, 4159, 3354, 4694, 1408, 3470, 3166, 4958}	1.0	1.0
2	Gene expression Cancer RNA-Seq	{18493, 15864, 15442, 16327, 6917, 17651, 19940}	1.0	1.0
3	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
4	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

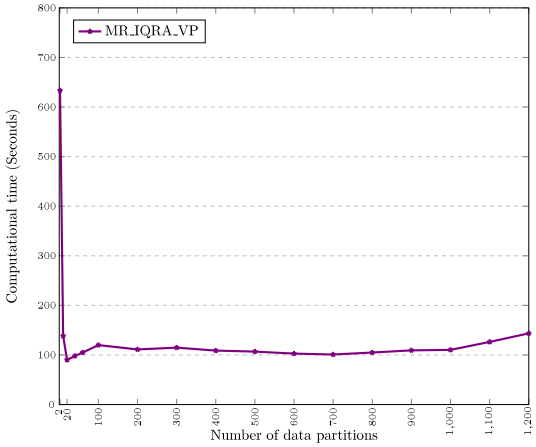


Fig. 3. MR_IQRA_VP behavior with different partitions on Gisette dataset.

objects with any number of attributes, and the same would be demonstrated empirically along with theoretical validation in Section 5.4. Accordingly, for comparative analysis, datasets have been considered to meet this criterion. A series of experiments have been conducted on the benchmark datasets such as Gisette, Gene expression Cancer RNA-Seq, Basehock and Semeion Handwritten Digit datasets. The Basehock dataset is available in Arizona State University feature selection dataset repository [47], and the rest of the datasets are available in UCI dataset repository [48]. Details of the datasets are given in Table 2.

5.1.1. Selection of the number of 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. 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 framework. Even though we have different theories in the literature [36,42,45,46] to select the number of data partitions in MapReduce framework, we have conducted a trial experiment invoking the proposed algorithm MR_IQRA_VP on Gisette dataset for different number of data partitions in the given cluster configuration (refer Table 1). The number of partitions are taken in the range of 2–1000, and the resulting computational times are plotted in Fig. 3. From the results, we can observe that, until 24 data partitions (which is equal to number of cores in the cluster) the computational time is inversely proportional to number of partitions. And as we can see in the figure, we have not found much difference in computational time from 24 to 1000 partitions. As given in the literature [45,46], the number of data partitions should be equal to the number of cores (some times may be 2 *or* 3 $\times$ (*number of cores*)) in the cluster to achieve the maximum parallelism, and the same is observed in the experiment.

Table 4
Comparative results of MR_IQRA_VP with PLAR_SCE (Time: Seconds).

Iteration	*PLAR_SCE*[a]	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[b]	89.02

[a]Results as reported in [28].
[b]Indicates the estimated time of PLAR_SCE algorithm.

As mentioned in Section 4.4.4, in the proposed algorithm, the time complexity of an iteration in mapper is $\bigcirc(\frac{|C-R|}{p}|U|log|U|)$, where p is the number of data partitions. The proposed algorithm assumes that, cluster has p number of cores so that it achieves the expected computational gains. But, as illustrated in the above experiment, practically realizable parallelism is restricted by the number of available cores in the cluster. Hence, in all our experiments the number of data partitions is set to the number of available cores (i.e., 24).

5.2. Efficiency of MR_IQRA_VP

The efficiency of the proposed algorithm is shown by comparing the results with existing MapReduce based distributed/parallel reduct computation algorithms on different datasets. For reproducible research, obtained reducts and their respective γ_C and γ_R values of proposed MR_IQRA_VP algorithm for different datasets are given in Table 3.

5.2.1. MR_IQRA_VP comparison with PLAR

The authors of Parallel Large scale Attribute Reduction (PLAR) [28], 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 dataset obtained by PLAR_SCE are reported in [28]. 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 dataset. For this reason, MR_IQRA_VP is compared with PLAR_SCE in terms of iterations. The comparison results are shown in Table 4. The proposed algorithm computed the reduct with the length 15 (refer Table 3). Accordingly, the estimated computational time of PLAR_SCE for 15 iterations is calculated as 5187 s.

MR_IQRA_VP completed first five iterations in 72.12 s against 1729 s incurred in PLAR_SCE algorithm. The MR_IQRA_VP algorithm achieved a significant computational gain of 95.82%.

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

MR_IQRA_VP obtained the complete reduct in 89.02 s against estimated computational time of 5187 s 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 1), 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 scheme for Gisette kind of datasets with larger attribute space.

5.2.2. MR_IQRA_VP comparison with PFSPA

Qing He et al. [33] 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 Semeion Handwritten Digit dataset. From the experimental design in [33], dataset is replicated several times, and experiments were conducted on the replicated dataset. The comparative results are shown in Table 5.

Both PFSPA [33] and MR_IQRA_VP obtained similar length reducts with different sizes of Semeion Handwritten Digit dataset. 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.

5.2.3. MR_IQRA_VP comparison with IN_MRIQRA_IG

Sai Prasad et al. developed a parallel version of IQRA_IG (IN_MRIQRA_IG) [30] 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 on the Gisette, Gene expression Cancer RNA-Seq, and Basehock datasets. The comparison results are shown in Table 6.

MR_IQRA_VP achieved 52.75% computational gain in Basehock dataset, 66.61% in Gene expression Cancer RNA-Seq dataset, and 27.99% in Gisette dataset 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 Gene expression Cancer RNA-Seq dataset having smaller object space with huge attribute space.

5.3. Performance evaluation of MR_IQRA_VP

The performance of MR_IQRA_VP has been evaluated based on Speedup, Scaleup and Sizeup measurements along with IN_MRIQRA_IG. These are the scalability metrics for parallel systems, and they are used to predict the performance of a parallel algorithm.

5.3.1. Speedup

Speedup can be measured by increasing the number of machines in the parallel system while keeping the dataset 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 datasets with different varied nodes from 1 to 6. Fig. 4 shows the speedup performance results of MR_IQRA_VP and IN_MRIQRA_IG algorithms for different datasets.

As shown in Fig. 4, MR_IQRA_VP has achieved better speedup performance (nearing expected linear speedup for Gisette dataset) over IN_MRIQRA_IG, this is mainly due to simplified shuffle and sort phase in MR_IQRA_VP as described in Section 4.4.3. IN_MRIQRA_IG and other horizontal partitioning based algorithms are effected by elaborate shuffle and sort phase which hampered the speedup performance.

5.3.2. Scaleup

Scaleup is defined as the ability of an n-times larger cluster to perform n-times larger dataset in the same run time as the original system. It is given as: $Scaleup(n) = \frac{Time\ of\ one\ data\ on\ one\ computer}{Time\ of\ n\ data\ on\ n\ computers}$. To find the scaleup of the proposed algorithm, we increased the size of the dataset in proportion to the number of computers in the cluster. Here each dataset size started from 20%, 40%, 60%, 80%, and 100% of attributes in the dataset (that is, dataset 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. Fig. 5 shows the scaleup performance results of MR_IQRA_VP and IN_MRIQRA_IG algorithms for different datasets.

The scaleup analysis results shown in Fig. 5 demonstrates that, increasing the attribute space results in less computational time for MR_IQRA_VP, and more computational time for IN_MRIQRA_IG. Thus, in comparison with IN_MRIQRA_IG algorithm, our proposed algorithm exhibited better scaleup for the datasets that have moderate object space and larger attribute space.

5.3.3. Sizeup

Sizeup measures the time it takes on a given system when the dataset is n-times larger than the original dataset. It calculates the increase in computational time based on the size of datasets. The sizeup is specified as: $Sizeup(n) = \frac{Time\ for\ processing\ n\ data}{Time\ for\ processing\ one\ data}$. To find the sizeup performance of the proposed algorithm, we kept the number of computers as constant, and changed the size of the dataset. The number of computers were kept as six nodes. Each dataset size was increased with 20%, 40%, 60%, 80%, and 100% of attributes in the dataset. Fig. 6 shows the sizeup performance results of MR_IQRA_VP and IN_MRIQRA_IG algorithms for different datasets with varying sizes of attribute space.

As mentioned earlier, MR_IQRA_VP uses vertical partitioning scheme, the computational load in mappers increase in linear proportionality with an increase in the size of object space of the dataset. The Fig. 6 shows that, MR_IQRA_VP gives better sizeup than IN_MRIQRA_IG algorithm for the datasets having smaller object space and larger attribute space.

5.4. Relevance of MR_IQRA_VP

The experimental results have suggested that MR_IQRA_VP algorithm is suitable for datasets with moderate object space and larger attribute space. Considering IN_MRIQRA_IG as the representation of 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 datasets relevant for horizontal partitioning based reduct algorithms and vertical partitioning based reduct algorithms.

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.

Table 5
Comparative results of MR_IQRA_VP with PFSPA.

Dataset	Objects	Attributes	*PFSPA*[a]		MR_IQRA_VP	
			Time (Seconds)	Reduct length	Time (Seconds)	Reduct length
Handwritten	1 593	256	1086	22	10.04	22
Handwritten 3 times	4 779	256	1205	22	15.34	22
Handwritten 6 times	9 558	256	1282	22	23.84	21
Handwritten 12 times	19 116	256	1596	22	67.58	21
Handwritten 24 times	38 232	256	2062	22	307.88	22

[a]Results as reported in [33].

Table 6
Comparative results of MR_IQRA_VP with IN_MRIQRA_IG.

Dataset	Attributes	IN_MRIQRA_IG		MR_IQRA_VP	
		Time (Seconds)	Reduct length	Time (Seconds)	Reduct length
Basehock	4 864	76.68	45	36.23	45
Gene expression Cancer RNA-Seq	20 561	36.06	07	12.04	07
Gisette	5 000	123.63	16	89.02	15

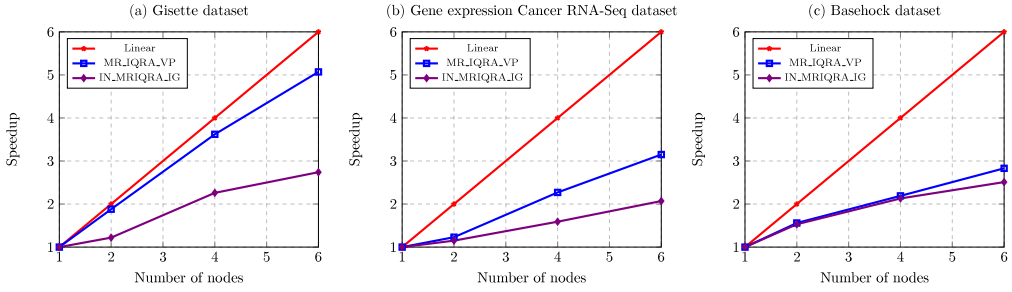


Fig. 4. Speedup of MR_IQRA_VP and IN_MRIQRA_IG for different datasets.

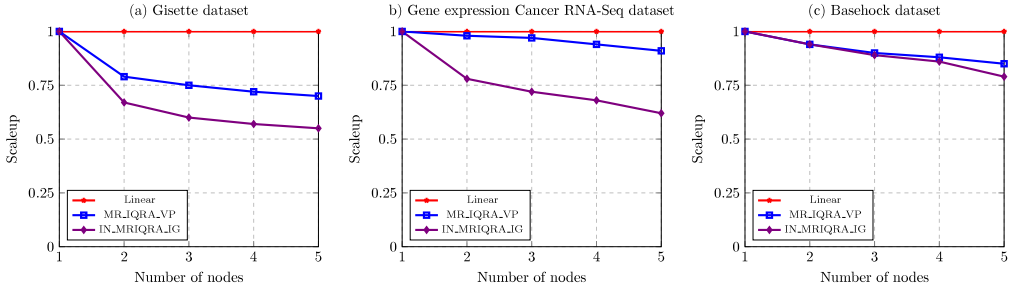


Fig. 5. Scaleup of MR_IQRA_VP and IN_MRIQRA_IG for different datasets.

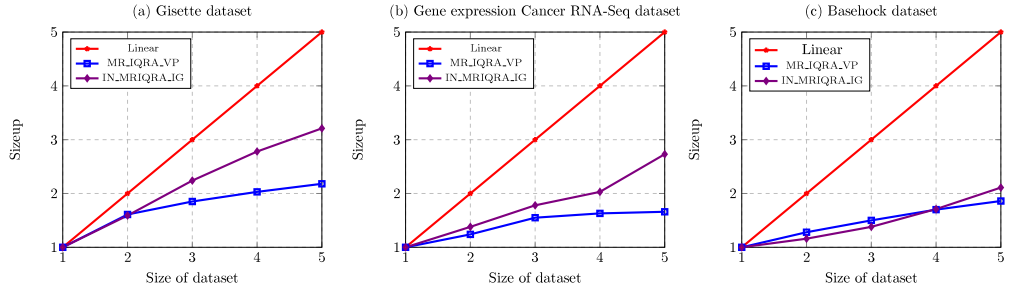
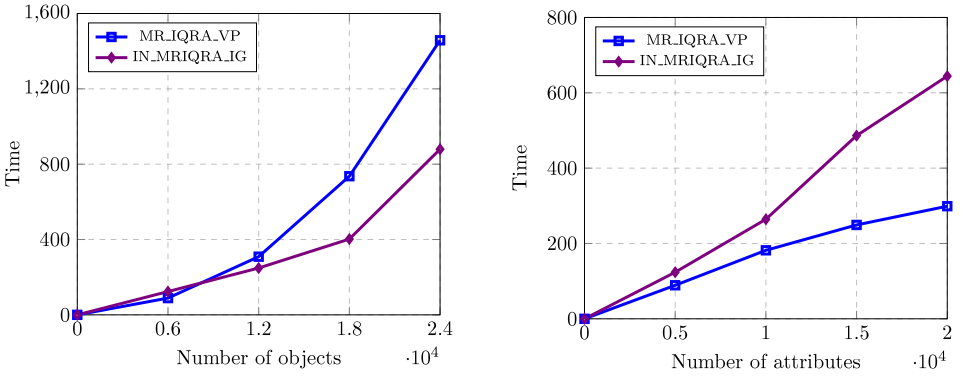


Fig. 6. Sizeup of MR_IQRA_VP and IN_MRIQRA_IG for different datasets.

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

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

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.



(a) Scalability in the object space (b) Scalability in the attribute space

Fig. 7. Behavior of MR_IQRA_VP and IN_MRIQRA_IG for varying objects and attributes of Gisette dataset.

Table 7
Comparison of MR_IQRA_VP, IN_MRIQRA_IG for varying objects and attributes of Gisette dataset (Time: Seconds).

S.No	Objects	Attributes	IN_MRIQRA_IG		MR_IQRA_VP	
			Time	Sizeup	Time	Sizeup
1	6 000	5 000	123.63	1.00	89.02	1.00
2	12 000	5 000	248.61	2.01	308.90	3.47
3	18 000	5 000	402.35	3.25	736.12	8.26
4	24 000	5 000	879.41	7.11	1457.87	16.37
5	6 000	10 000	264.27	2.13	181.83	2.04
6	6 000	15 000	486.32	3.93	249.18	2.79
7	6 000	20 000	644.22	5.21	298.98	3.35

IN_MRIQRA_IG. The sizeup coefficients are reported in Table 7, and the details of original Gisette dataset are reported in serial number 1. The sizeup analysis shows that MR_IQRA_VP is a highly scalable algorithm for scalability in attribute space. But it is not recommended for datasets 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 massive attribute space datasets with moderate object space frequently found in the areas of Bioinformatics and Web mining.

6. Conclusion

We have proposed and implemented a highly scalable MapReduce based reduct computation algorithm MR_IQRA_VP using vertical partitioning scheme. With this scheme 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 datasets having moderate size object space and larger size attribute space such as microarray datasets in Bioinformatics. In future, MR_IQRA_VP algorithm will be improved further by reducing the serial computation involved in the mapper phase. Vertical partitioning scheme will also be applied in designing MapReduce based reduct computation algorithms for large scale incomplete and hybrid decision systems using extensions of rough sets. We will also investigate for suitable MapReduce approaches that can scale both in object space as well as attribute space.

Acknowledgments

Authors are grateful to the four anonymous reviewers for their valuable comments and suggestions which helped in improving the quality of this manuscript. The first author acknowledges the financial support received from the Digital India Corporation, Government of India under Visvesvaraya Ph.D. Scheme. The second author acknowledges the financial support of AICTE, Government of India under RPS project [grant number: File no. 8-47/RIFD/RPS/POLICY-1/2016-17].

References

[1] T. Li, C. Luo, H. Chen, J. Zhang, PICKT: a solution for big data analysis, in: International Conference on Rough Sets and Knowledge Technology, Springer, 2015, pp. 15–25.
[2] S. Gunelius, The data explosion in 2014 minute by minute–infographic, 2014, ACI. Retrieved July, 29, p. 2015.
[3] Q. Hu, Z. Xie, D. Yu, Hybrid attribute reduction based on a novel fuzzy-rough model and information granulation, Pattern Recognit. 40 (12) (2007) 3509–3521.
[4] M. Dash, H. Liu, Consistency-based search in feature selection, Artif. Intell. 151 (1–2) (2003) 155–176.
[5] Q. Hu, W. Pedrycz, D. Yu, J. Lang, Selecting discrete and continuous features based on neighborhood decision error minimization, IEEE Trans. Syst. Man Cybern. B 40 (1) (2010) 137–150.
[6] Y. Yang, Z. Chen, Z. Liang, G. Wang, Attribute reduction for massive data based on rough set theory and mapreduce, in: International Conference on Rough Sets and Knowledge Technology, Springer, 2010, pp. 672–678.
[7] Z. Pawlak, Rough sets, Int. J. Comput. Inf. Sci. 11 (5) (1982) 341–356.
[8] Z. Pawlak, Rough Sets: Theoretical Aspects of Reasoning About Data, Kluwer, Netherlands, 1991, p. 224.
[9] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (Mar) (2003) 1157–1182.
[10] Z. Pawlak, A. Skowron, Rudiments of rough sets, Inform. Sci. 177 (1) (2007) 3–27.
[11] Z. Pawlak, A. Skowron, Rough sets: some extensions, Inform. Sci. 177 (1) (2007) 28–40.
[12] Q. Hu, D. Yu, J. Liu, C. Wu, Neighborhood rough set based heterogeneous feature subset selection, Inform. Sci. 178 (18) (2008) 3577–3594.
[13] F. Min, H. He, Y. Qian, W. Zhu, Test-cost-sensitive attribute reduction, Inform. Sci. 181 (22) (2011) 4928–4942.
[14] D.-Q. Miao, G.-R. Hu, A heuristic algorithm for reduction of knowledge, J. Comput. Res. Dev. 36 (6) (1999) 681–684.
[15] D. Miao, Y. Zhao, Y. Yao, H. Li, F. Xu, Relative reducts in consistent and inconsistent decision tables of the pawlak rough set model, Inform. Sci. 179 (24) (2009) 4140–4150.
[16] A. Chouchoulas, Q. Shen, Rough set-aided keyword reduction for text categorization, Appl. Artif. Intell. 15 (9) (2001) 843–873.
[17] P.S.V.S. Sai Prasad, C.R. Rao, Extensions to iquickreduct, in: International Workshop on Multi-Disciplinary Trends in Artificial Intelligence, Springer, 2011, pp. 351–362.
[18] Y. Qian, J. Liang, W. Pedrycz, C. Dang, Positive approximation: an accelerator for attribute reduction in rough set theory, Artificial Intelligence 174 (9–10) (2010) 597–618.

[19] J. Zhang, T. Li, D. Ruan, D. Liu, Rough sets based matrix approaches with dynamic attribute variation in set-valued information systems, Internat. J. Approx. Reason. 53 (4) (2012) 620–635.
[20] Y. Cheng, The incremental method for fast computing the rough fuzzy approximations, Data Knowl. Eng. 70 (1) (2011) 84–100.
[21] T. Li, D. Ruan, W. Geert, J. Song, Y. Xu, A rough sets based characteristic relation approach for dynamic attribute generalization in data mining, Knowl.-Based Syst. 20 (5) (2007) 485–494.
[22] J. Liang, F. Wang, C. Dang, Y. Qian, A group incremental approach to feature selection applying rough set technique., IEEE Trans. Knowl. Data Eng. 26 (2) (2014) 294–308.
[23] Y. Qian, J. Liang, W. Pedrycz, C. Dang, An efficient accelerator for attribute reduction from incomplete data in rough set framework, Pattern Recognit. 44 (8) (2011) 1658–1670.
[24] B. Zhou, H. Cho, X. Zhang, Scalable implementations of rough set algorithms: A survey, in: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Springer, 2018, pp. 648–660.
[25] M.S. Raza, U. Qamar, A parallel rough set based dependency calculation method for efficient feature selection, Appl. Soft Comput. 71 (2018) 1020–1034.
[26] J. Zhang, T. Li, D. Ruan, Z. Gao, C. Zhao, A parallel method for computing rough set approximations, Inform. Sci. 194 (2012) 209–223.
[27] J. Qian, D. Miao, Z. Zhang, X. Yue, Parallel attribute reduction algorithms using mapreduce, Inform. Sci. 279 (2014) 671–690.
[28] J. Zhang, T. Li, Y. Pan, Parallel large-scale attribute reduction on cloud systems, 2016, arXiv preprint arXiv:1610.01807.
[29] M. Czolombitko, J. Stepaniuk, Attribute reduction based on mapreduce model and discernibility measure, in: IFIP International Conference on Computer Information Systems and Industrial Management, Springer, 2016, pp. 55–66.
[30] P.S.V.S. Sai Prasad, H.B. Subrahmanyam, P.K. Singh, Scalable iqra_ig algorithm: an iterative mapreduce approach for reduct computation, in: International Conference on Distributed Computing and Internet Technology, Springer, 2017, pp. 58–69.
[31] P.K. Singh, P.S.V.S. Sai Prasad, Scalable quick reduct algorithm: iterative mapreduce approach, in: Proceedings of the 3rd IKDD Conference on Data Science, 2016, ACM, 2016, p. 25.
[32] U.V. Divya, P.S.V.S. Sai Prasad, Hashing supported iterative mapreduce based scalable SBE reduct computation, in: International Conference on Distributed Computing and Internet Technology, Springer, 2018, pp. 163–170.
[33] Q. He, X. Cheng, F. Zhuang, Z. Shi, Parallel feature selection using positive approximation based on mapreduce, in: Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on, IEEE, 2014, pp. 397–402.
[34] M. Chen, J. Yuan, L. Li, D. Liu, T. Li, A fast heuristic attribute reduction algorithm using spark, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2017, pp. 2393–2398.
[35] T. White, Hadoop: The Definitive Guide, " O'Reilly Media, Inc.", 2012.
[36] A. Spark, Apache spark: Lightning-fast cluster computing, 2016, URL http://spark.apache.org.
[37] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, G. Fox, Twister: a runtime for iterative mapreduce, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM, 2010, pp. 810–818.
[38] S. Ramírez-Gallego, H. Mouriño-Talín, D. Martínez-Rego, V. Bolón-Canedo, J.M. Benítez, A. Alonso-Betanzos, F. Herrera, An information theory-based feature selection framework for big data under apache spark, IEEE Trans. Syst. Man Cybern. A 48 (9) (2018) 1441–1453.
[39] Y. Yao, The two sides of the theory of rough sets, Knowl.-Based Syst. 80 (2015) 67–77.
[40] A. Skowron, C. Rauszer, The discernibility matrices and functions in information systems, in: Intelligent Decision Support, Springer, 1992, pp. 331–362.
[41] J.R. Anaraki, M. Eftekhari, Rough set based feature selection: a review, in: Information and Knowledge Technology (IKT), 2013 5th Conference on, IEEE, 2013, pp. 301–306.
[42] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[43] Gene expression data, National Center for Biotechnology Information, URL https://www.ncbi.nlm.nih.gov/gene.
[44] HashMap, Scala 2.11.4 library, URL https://www.scala-lang.org/api/2.11.4/#scala.collection.mutable.HashMap.
[45] P. Jakovits, S.N. Srirama, Evaluating mapreduce frameworks for iterative scientific computing applications, in: High Performance Computing & Simulation (HPCS), 2014 International Conference on, IEEE, 2014, pp. 226–233.
[46] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, E.M. Nguifo, An experimental survey on big data frameworks, Future Gener. Comput. Syst. (2018) 546–554.
[47] A.S. University, Feature selection data sets: http://featureselection.asu.edu/datasets.php,
[48] M. Lichman, et al., UCI machine learning repository, 2013.

Please cite this article as: P. Sowkuntla and P.S.V.S. Sai Prasad, MapReduce based improved quick reduct algorithm with granular refinement using vertical partitioning scheme, Knowledge-Based Systems (2019) 105104, https://doi.org/10.1016/j.knosys.2019.105104.