A Parallel Matrix-Based Method for Computing Approximations in Incomplete Information Systems

Junbo Zhang, Jian-Syuan Wong, Yi Pan, *Senior Member, IEEE*, and Tianrui Li, *Senior Member, IEEE*

Abstract—As the volume of data grows at an unprecedented rate, large-scale data mining and knowledge discovery present a tremendous challenge. Rough set theory, which has been used successfully in solving problems in pattern recognition, machine learning, and data mining, centers around the idea that a set of distinct objects may be approximated via a lower and upper bound. In order to obtain the benefits that rough sets can provide for data mining and related tasks, efficient computation of these approximations is vital. The recently introduced cloud computing model, MapReduce, has gained a lot of attention from the scientific community for its applicability to large-scale data analysis. In previous research, we proposed a MapReduce-based method for computing approximations in parallel, which can efficiently process complete data but fails in the case of missing (incomplete) data. To address this shortcoming, three different parallel matrix-based methods are introduced to process large-scale, incomplete data. All of them are built on MapReduce and implemented on Twister that is a lightweight MapReduce runtime system. The proposed parallel methods are then experimentally shown to be efficient for processing large-scale data.

Index Terms—Rough sets, data mining, MapReduce, matrix, incomplete information systems

1 Introduction

With the development of information technology, the data volume is growing at an unprecedented rate. Large-scale data mining and knowledge discovery has been a tremendous challenge. The parallel program model MapReduce supports large distributed data sets on clusters of computers [1], [2], which allows for an efficient analysis of large amounts of data. As one of the most important and effective cloud computing techniques, MapReduce has become a popular computing model for cloud platforms [3]. Many cloud platforms support the MapReduce framework. For instance, both Amazon EC2 and Microsoft Azure provide support for open source MapReduce Hadoop [4] and Twister [5] runtime systems via Amazon Elastic MapReduce [6] and Twister4Azure [7], respectively.

Rough set theory is a powerful mathematical tool that can be used to process inconsistent information in decision-making situations [8], [9], [10], [11]. It plays an important role in the fields of pattern recognition, machine learning and data mining during last decades [12], [13], [14], [15], [16], [17], [18], [19], [20], [21].

To our knowledge, the most current methods based on rough sets are sequential and can only run on a single computer to deal with small data sets. For example, ROSETTA is such a toolkit that can be used for data analysis, which can be downloaded at http://www.lcb.uu.se/tools/rosetta/. To expand the applications of rough sets to the field of large scale data mining and knowledge discovery, we proposed parallel methods based on MapReduce in our previous work [22], [23]. However, existing studies on parallel methods are mainly based on complete information systems. They do not consider the missing data in real databases, which frequently occur in decision-making problems. This motivates the need for an investigation of parallel data mining methods based on rough sets in incomplete information systems (IIS).

Our contributions in this paper include:

1) Traditional methods for computing approximations in IIS are based on the crisp set view, which are difficult to parallelize and fail to process large-scale incomplete data with rough set theory. To achieve this goal, a new matrix view of the rough set theory in IIS is presented and the relation matrix is then introduced for computing approximations.
2) A MapReduce-based parallel method to construct the relation matrix is designed for fast computing approximations. The key feature of this method is a Sub-Merge operation, which reduces the space requirement and accelerates the process of merging the relation matrices.
3) An incremental method is applied to the process of merging the relation matrices. With this feature, the relation matrix is updated in parallel and incrementally to efficiently accelerate the computational process.

- *J. Zhang is with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China and the Department of Computer Science, Georgia State University, Atlanta, GA 30303. E-mail: jbzhang@my.swjtu.edu.cn, jbzhang@cs.gsu.edu.*
- *J.-S. Wong and Y. Pan are with the Department of Computer Science, Georgia State University, Atlanta, GA 30303. E-mail: jwong9@student.gsu.edu, pan@cs.gsu.edu.*
- *T. Li is with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China. E-mail: trli@swjtu.edu.cn.*

Manuscript received 28 Oct. 2012; revised 11 Feb. 2014; accepted 12 May 2014. Date of publication 12 June 2014; date of current version 23 Dec. 2014. Recommended for acceptance by P. K. Chrysanthis.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2330821

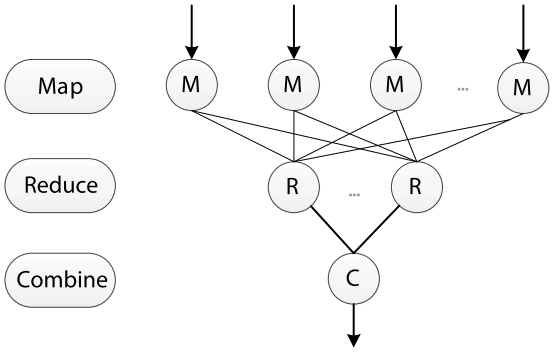


Fig. 1. The MapReduce framework in Twister.

4) A sparse matrix method is employed to optimize the proposed matrix-based method and further improve the performance of the algorithm.

The rest of the paper is organized as follows: we provide the basic concepts of the MapReduce model and rough sets under IIS in Section 2. We introduce the matrix representation of the approximations in IIS in Section 3. Three different parallel matrix-based methods are proposed for the incomplete decision table (IDT) in Section 4. Experimental analysis is given in Section 5. The paper ends with conclusions and further research topics in Section 6.

2 Preliminaries

The basic concepts, notation and results of the MapReduce model [1], [2] and rough sets as well as their extensions are briefly reviewed [9], [24], [25], [26], [27], [28].

2.1 The MapReduce Model and Twister

This section reviews some basic ideas of MapReduce and relative features of Twister. The MapReduce model, by Google, is designed to handle large-scale data sets in a distributed environment and has become a well-known parallel model in cloud computing [2]. It provides a convenient way to parallelize data analysis. The advantages of MapReduce include convenience, robustness, and scalability. Fig. 1 shows the MapReduce framework in Twister. The computation in MapReduce model includes two major phases, the Map phase and the Reduce phase. Each phase utilizes key/value pairs as input and output. The programmer can choose types of keys and values depending on the requirements. In addition, the Map and Reduce functions are also required for the Map phase and the Reduce phase, respectively. Input data is taken by the master node and divided into sub-program for the Map phase. A Map function can use a key/value pair (*K1*, *V1*), and also generate the output in another key/value pair (*K2*, *V2*) format. The intermediate results then are collected by the runtime system and used to produce the key/list values (*K2*, a list of (*V2*)) pairs as the input of a Reduce function. Once the Reduce function finishes its process, another key/list values (*K3*, a list of (*V3*)) pair is produced as the output.

After Google's work, various MapReduce platforms were developed. (1) Apache Hadoop [4] is one of the most popular run-time systems based on the MapReduce model; (2) Twister [5] is another run-time system especially designed for iterative MapReduce applications; (3) Phoenix [29] is a shared-memory implementation of MapReduce model for data-intensive processing tasks; (4) Mars [30] is a MapReduce runtime system for graphic processors. In this paper, the proposed algorithms are all implemented on Twister, which is reviewed as follows.

Twister, a lightweight MapReduce runtime system, is designed for facilitating iterative applications based on MapReduce model. In [5], Twister also demonstrates that it is a good fit for several iterative scientific applications such as Blast. Besides the support of iterative MapReduce, Twister also has other attractive features that are different to typical non-iterative MapReduce runtime systems: I. Intermediate results are stored in the distributed memory of worker role which is more efficient than writing and reading intermediate data in the local disk; II. Input file can be partitioned into several files as the input data of the Map phase; III. The publish/subscribe messaging infrastructure is applied as the network broker to make communications become more efficient. IV. The *Combine* phase in Twister is used to collect results from all the reduce outputs.

2.2 Rough Sets

Definition 1. *[9] An information system is defined as a pair $IS = (U, A)$ of non-empty, finite set U and A, where U is a universe of objects, and A is a set consisting of attributes,* i.e., *functions $a : U \to V_a$, where V_a is the set of values of attribute a, called the domain of a. Sometimes we distinguish in an information system $IS = (U, A)$ a partition of A into two classes $C, D \subseteq A$ of attributes, called condition and decision attributes, respectively. $DT = (U, C \cup D)$ is called a decision table.*

Definition 2. *[9] Let $B \subseteq A$ be a subset of attributes. The indiscernibility relation, denoted by I_B, is an equivalence relation defined as:*

$$\forall x, y, I_B(x, y) \Leftrightarrow \forall a \in B : a(x) = a(y), \tag{1}$$

where $a(x)$ denotes the value of attribute a of object x.

The classical rough set analysis depends on the indiscernibility relation that describes indistinguishability of objects. It is based on the equivalence relation. The objects in an equivalence class satisfy reflexive, symmetric and transitive. In addition, it is not able to handle missing data, and requires the information system should be complete. However, missing data appears frequently in real applications [25]. The classical rough set approach, based on complete information systems, cannot be directly applied in information systems with missing attribute values. To process missing data, some extended rough set models [25], [27], [28] were presented.

Definition 3. *[25] An information system (U, A) is called as an incomplete information system if there exists a in A and x in U that satisfy that the value $a(x)$ is unknown, denoted as "*", which is considered as an "everything is possible" value.*

Under this definition of IIS, the toleration and similarity relations are proposed respectively to deal with unknown data in [25], [27].

Definition 4. *[25] Let $B \subseteq A$ be a subset of attributes. The tolerance relation, denoted by T_B, is defined as:*

$$T_B = \{(x,y)|\forall a \in B, \\ (a(x) = a(y)) \vee (a(y) = *) \vee (a(x) = *)\}. \quad (2)$$

Obviously, the tolerance relation T is reflexive and symmetric, but not transitive.

Definition 5. *[27] Let $B \subseteq A$ be a subset of attributes. The similarity relation, denoted by S_B, is defined as*

$$S_B = \{(x,y)|\forall a \in B, (a(x) \neq *) \wedge (a(x) = a(y))\}. \quad (3)$$

Obviously, the similarity relation S is reflexive and transitive, but not symmetric.

In many cases, the information system is incomplete due to the following two reasons [28]. One is that the value of an attribute is lost for a specific case. As an example, the value is currently unavailable although it was known originally owing to a variety of reasons, e.g., it was recorded but erased later. Another reason is the value of an attribute is irrelevant or unimportant. For example, it is feasible to diagnose a patient in spite of the fact that some clinical test results are not taken. Such missing attribute values do not matter for the final outcome and are called as "do not care" conditions [28]. Under this assumption that missing values in IIS are lost or in the "do not care" condition, a new version of the IIS is given as follows.

Definition 6. *[28] $IIS = (U, A)$ is an incomplete information system if there exists a in A and x in U that satisfy that the value $a(x)$ is missing. All the missing values are denoted by "?" or "*", where the lost values are denoted by "?" and values in "do not care" condition are denoted by "*".*

Under this new definition of IIS, Grzymala-Busse integrated the tolerance relation [25] and the similarity relation [27] and put forward a characteristic relation [28].

Definition 7. *[28] In the incomplete information system $IIS = (U, A)$, for $B \subseteq A$, the characteristic relation K_B is defined as*

$$K_B = \{(x,y)|\forall a \in B, \\ (a(x) \neq ?) \wedge (a(x) = a(y) \vee a(x) = * \vee a(y) = *)\}. \quad (4)$$

where "?" and "" are missing values which mean the lost value and "do not care", respectively.*

When $(x,y) \in K_B$, we call x and y are indiscernible w.r.t. B. Let $K_B(x) = \{y|y \in U, (x,y) \in K_B\}$. We call $K_B(x)$ the characteristic class for x w.r.t. K_B.

The characteristic relation is reflexive, but not symmetric and transitive. It is a generalization of tolerance relation and similarity relation.

Table 1 shows a description of above binary relations.

Definition 8. *[24] Given an incomplete information system $IIS = (U, A)$, $\forall X \subseteq U$, $B \subseteq A$, the lower and upper approximations of X in terms of the characteristic relation K_B are defined as*

$$\underline{K_B}(X) = \{x \in U|K_B(x) \subseteq X\}, \quad (5)$$

TABLE 1
A Description of Relations

Relation	Reflexive	Symmetric	Transitive
Equivalence	√	√	√
Tolerance	√	√	×
Similarity	√	×	√
Characteristic	√	×	×

$$\overline{K_B}(X) = \{x \in U|K_B(x) \cap X \neq \emptyset\}. \quad (6)$$

Here, these two approximations divide the universe U into three disjoint regions: the positive region $POS_{K_B}(X)$, the boundary region $BND_{K_B}(X)$ and the negative region $NEG_{K_B}(X)$, respectively.

$$\begin{cases} POS_{K_B}(X) = \underline{K_B}(X) \\ BND_{K_B}(X) = \overline{K_B}(X) - \underline{K_B}(X) \\ NEG_{K_B}(X) = U - \overline{K_B}(X) \end{cases} \quad (7)$$

Definition 9. *[24] Given an incomplete decision table $IDT = (U, C \cup D)$, $B \subseteq C$. Let $U/D = \{D_1, D_2, \ldots, D_r\}$ be a partition over the decision D. Then the lower and upper approximations of the decision D with respect to attributes B are defined as*

$$\underline{K_B}(D) = \cup_{j=1}^{r} \underline{K}_B(D_j) \quad (8)$$

$$\overline{K_B}(D) = \cup_{j=1}^{r} \overline{K}_B(D_j), \quad (9)$$

where

$$\underline{K_B}(D_j) = \{x \in U|K_B(x) \subseteq D_j\} \quad (10)$$

$$\overline{K_B}(D_j) = \{x \in U|K_B(x) \cap D_j \neq \emptyset\}. \quad (11)$$

The positive region $POS_{K_B}(D)$, the boundary region $BND_{K_B}(D)$ and the negative region $NEG_{K_B}(D)$ of D with respect to the attribute subset B are defined as

$$\begin{cases} POS_{K_B}(D) = \underline{K_B}(D) \\ BND_{K_B}(D) = \overline{K_B}(D) - \underline{K_B}(D) \\ NEG_{K_B}(D) = U - \overline{K_B}(D) \end{cases} \quad (12)$$

3 Matrix Representation of the Approximations in Incomplete Information Systems

We here introduce the matrix representation of the lower and upper approximations in IIS.

From the matrix point of view, a set of axioms were constructed to characterize classical rough set upper approximation [31]. In our previous work, we defined a basic vector $H(X)$, which was induced from the relation matrix. Then four cut matrices of $H(X)$, denoted by $H^{[\mu,\nu]}(X)$, $H^{(\mu,\nu]}(X)$,

$H^{[\mu,\nu)}(X)$ and $H^{(\mu,\nu)}(X)$, were derived for the approximations, positive, boundary and negative regions intuitively in set-valued information systems [32]. In this paper, to process large-scale incomplete data effectively, we also introduce the matrix-based methods.

First, we review the vector representation of the subset X of the domain U.

Definition 10. *[32] Let $U = \{x_1, x_2, \ldots, x_n\}$, and X be a subset of U. The characteristic function $G(X) = (g_1, g_2, \ldots, g_n)^T$ (T denotes the transpose operation) is defined as*

$$g_i = \begin{cases} 1, & x_i \in X \\ 0, & x_i \notin X \end{cases} \tag{13}$$

where $G(X)$ assigns 1 to an element that belongs to X and 0 to an element that does not belong to X. For example, if $U = \{x_1, x_2, x_3\}$, $X = \{x_1, x_3\}$, then we have $G(X) = \{1, 0, 1\}$.

Therefore, we use the characteristic relation to construct the relation matrix.

Definition 11. *[33] Given an incomplete information system $IIS = (U, A)$. Let $B \subseteq A$ and K_B be an characteristic relation on U, $RM^{K_B}_{n\times n} = (m_{ij})_{n\times n}$ be an $n \times n$ matrix representing K_B, called the relation matrix w.r.t. B. Then*

$$m_{ij} = \begin{cases} 1, & (x_i, x_j) \in K_B \\ 0, & (x_i, x_j) \notin K_B. \end{cases} \tag{14}$$

According the above definitions, we obtain the following corollary easily.

Corollary 1. *Let $RM^{K_B}_{n\times n} = (m_{ij})_{n\times n}$ and K_B be a characteristic relation on U. Then $m_{ii} = 1, 1 \le i, j \le n$.*

The induced diagonal matrix of the relation matrix is given for the later calculation of the basic vector.

Definition 12. *[32] Let $B \subseteq A$, K_B be a characteristic relation on U, and $\Lambda^{K_B}_{n\times n}$ be an induced diagonal matrix of $RM = (m_{ij})_{n\times n}$. Then*

$$\begin{aligned} \Lambda^{K_B}_{n\times n} &= \operatorname{diag}\left(\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, \ldots, \frac{1}{\lambda_n}\right) \\ &= \operatorname{diag}\left(\frac{1}{\sum_{j=1}^{n} m_{1j}}, \frac{1}{\sum_{j=1}^{n} m_{2j}}, \ldots, \frac{1}{\sum_{j=1}^{n} m_{nj}}\right) \end{aligned} \tag{15}$$

where $\lambda_i = \sum_{j=1}^{n} m_{ij}, 1 \le i \le n$.

Corollary 2. $\Lambda^{K_B}_{n\times n} = \operatorname{diag}\left(\frac{1}{|K_B(x_1)|}, \frac{1}{|K_B(x_2)|}, \ldots, \frac{1}{|K_B(x_n)|}\right)$ *and* $1 \le |K_B(x_i)| \le n, 1 \le i \le n$.

To give the vector representation of rough set approximations, the positive region, the boundary region and the negative region, we need to introduce the basic vector as well as its four cut matrices.

Definition 13. *[32] The n-column vector called a basic vector, denoted by $H(X)$, is defined as:*

$$H(X) = \Lambda^{K_B}_{n\times n} \bullet M^{K_B}_{n\times n} \bullet G(X), \tag{16}$$

where $\bullet$ is the dot product of matrices.

Definition 14. *[32] Let $0 \le \mu \le \nu \le 1$. Four cut matrices of $H(X)$, denoted by $H^{[\mu,\nu]}(X)$, $H^{(\mu,\nu]}(X)$, $H^{[\mu,\nu)}(X)$ and $H^{(\mu,\nu)}(X)$, are defined as follows:*

1) $H^{[\mu,\nu]}(X) = (h'_i)_{n\times 1}$

$$h'_i = \begin{cases} 1, & \mu \le h_i \le \nu \\ 0, & else. \end{cases} \tag{17}$$

2) $H^{(\mu,\nu]}(X) = (h'_i)_{n\times 1}$

$$h'_i = \begin{cases} 1, & \mu < h_i \le \nu \\ 0, & else. \end{cases} \tag{18}$$

3) $H^{[\mu,\nu)}(X) = (h'_i)_{n\times 1}$

$$h'_i = \begin{cases} 1, & \mu \le h_i < \nu \\ 0, & else. \end{cases} \tag{19}$$

4) $H^{(\mu,\nu)}(X) = (h'_i)_{n\times 1}$

$$h'_i = \begin{cases} 1, & \mu < h_i < \nu \\ 0, & else. \end{cases} \tag{20}$$

Remark 1. These four cut matrices are Boolean matrices.

According to the cut matrices of the basic vector, we present the vector representation of rough set approximations as well as the positive region, the boundary region and the negative region.

Corollary 3. *Given any subset $X \subseteq U$ in an incomplete information system $IIS = (U, A)$, where $U = \{x_1, x_2, \ldots, x_n\}$. $B \subseteq A$ and K_B is a characteristic relation on U. $H(X) = (h_1, h_2, \ldots, h_n)^T$ is the basic vector. Then the lower and upper approximations of X in IIS can be computed from the cut matrix of $H(X)$ as follows.*

(1) The n-column boolean vector $G(\underline{K_B}(X))$ of the lower approximation $\underline{K_B}(X)$:

$$G(\underline{K_B}(X)) = H^{[1,1]}(X) \tag{21}$$

(2) The n-column boolean vector $G(\overline{K_B}(X))$ of the upper approximation $\overline{K_B}(X)$:

$$G(\overline{K_B}(X)) = H^{(0,1]}(X) \tag{22}$$

Corollary 4. *The positive region $POS_{K_B}(X)$, the boundary region $BND_{K_B}(X)$, and the negative region $NEG_{K_B}(X)$ can also be generated from the cut matrix of $H(X)$, respectively as follows.*

(1) The n-column boolean vector $G(POS_{K_B}(X))$ of the positive region:

$$G(POS_{K_B}(X)) = H^{[1,1]}(X) \tag{23}$$

(2) The n-column boolean vector $G(BND_{K_B}(X))$ of the boundary region:

$$G(BND_{K_B}(X)) = H^{(0,1)}(X) \tag{24}$$

(3) The n-column boolean vector $G(NEG_{K_B}(X))$ of the negative region:

$$G(NEG_{K_B}(X)) = H^{[0,0]}(X) \tag{25}$$

3.1 Matrix-Based Method in the Incomplete Decision Table

We have introduced the vector representation of rough set approximations of the subset X of the domain U. However, in IDT, the domain U is partitioned by the decision D. Therefore, we need to give the representation of rough set approximations of all subsets of the domain U. That is the matrix representation of rough set approximations.

Definition 15. *Given an incomplete decision table $IDT = (U, C \cup D)$, $B \subseteq C$, let $U/D = \{D_1, D_2, \ldots, D_r\}$ be a partition over the decision D. $\forall D_j \in U/D$, $G(D_j)$ is an n-column boolean vector of D_j. Let $DM_{n\times r} = (G(D_1), G(D_2), \ldots, G(D_r))$ be an $n \times r$ boolean matrix, called a decision matrix.*

Similarly, we combine all the basic vectors as the basic matrix and give its computational method as follows.

Corollary 5. *Given an incomplete decision table $IDT = (U, C \cup D)$, $B \subseteq C$. Let $U/D = \{D_1, D_2, \ldots, D_r\}$ be a partition over the decision D. $\forall D_j \in U/D$, $H(D_j)$ is the basic vector. Let $HD = (H(D_1), H(D_2), \ldots, H(D_r))$ be an $n \times r$ matrix, called a basic matrix. Then, HD can be computed as follows.*

$$HD = \Lambda^{K_B}_{n\times n} \bullet RM^{K_B}_{n\times n} \bullet DM_{n\times r} \tag{26}$$

where $\bullet$ is the dot product of matrices.

Therefore, we present the vector representation of rough set approximations of each subset as well as the positive region, the boundary region and the negative region of the decision D.

Corollary 6. *Given an incomplete decision table $IDT = (U, C \cup D)$, $B \subseteq C$. Let $U/D = \{D_1, D_2, \ldots, D_r\}$ be a partition over the decision D. $HD = (H(D_1), H(D_2), \ldots, H(D_r))$ is a basic matrix. $\forall j = 1, 2, \ldots, r$, the upper and lower approximations of D_j in IIS can be computed from the cut matrix of HD as follows.*

(1) The n-column boolean vector $G(\underline{K_B}(D_j))$ of the lower approximation $\underline{K_B}(D_j)$:

$$G(\underline{K_B}(D_j)) = H^{[1,1]}(D_j) \tag{27}$$

(2) The n-column boolean vector $G(\overline{K_B}(D_j))$ of the upper approximation $\overline{K_B}(D_j)$:

$$G(\overline{K_B}(D_j)) = H^{(0,1]}(D_j) \tag{28}$$

The positive region $POS_{K_B}(D)$, the boundary region $BND_{K_B}(D)$, and the negative region $NEG_{K_B}(D)$ can also be generated from the cut matrix of HD, respectively as follows.

(3) The n-column boolean vector $G(POS_{K_B}(D))$ of the positive region:

$$G(POS_{K_B}(D)) = \sum\nolimits_{j=1}^{r} H^{[1,1]}(D_j), \tag{29}$$

where $\sum$ is the boolean addition (logical operation OR) of boolean vectors.

(4) The n-column boolean vector $G(BND_{K_B}(D))$ of the boundary region:

$$\begin{aligned} G(BND_{K_B}(D)) &= \sum_{j=1}^{r} H^{(0,1]}(D_j) - \sum_{j=1}^{r} H^{[1,1]}(D_j) \\ &= \sum_{j=1}^{r} H^{(0,1)}(D_j). \end{aligned} \tag{30}$$

(5) The n-column boolean vector $G(NEG_{K_B}(D))$ of the negative region:

$$\begin{aligned} G(NEG_{K_B}(D)) &= J - \sum_{j=1}^{r} H^{(0,1]}(D_j) \\ &= \left(\sum_{j=1}^{r} H^{(0,1]}(D_j) \right)^{[0,0]} \end{aligned} \tag{31}$$

where $J = (1, 1, \ldots, 1)^T$ is an $n \times 1$ vector of ones.

3.2 Sequential Matrix-Based Algorithm (SMA) in Incomplete Decision Table

From the view of the matrix, according Definitions 11 and 15 and Corollaries 5 and 6, we present a sequential method for computing rough set approximations in IDT and its corresponding naive Sequential Matrix-based Algorithm (Fig. 2 and Algorithm 1).

Algorithm 1: A Naive Sequential Matrix-Based Algorithm (SMA) for Computation of Rough Set Approximations

Input: The incomplete decision table $IDT = (U, C \cup D)$ and the condition attribute subset $B \subseteq C$.
Output: The approximations of the decision: $\underline{K}_B(D_i)$ and $\overline{K}_B(D_i)$, $i = 1, 2, \ldots, r$; the positive region $POS_{K_B}(D)$, the boundary region $BND_{K_B}(D)$, and the negative region $NEG_{K_B}(D)$.

1 **begin**
2 **Construct** the relation matrix $RM^{K_B}_{n\times n} = (m_{ij})_{n\times n}$ in terms of the condition attributes B by using the characteristic relation K_B;
3 **Compute** the induced diagonal matrix $\Lambda^{K_B}_{n\times n} = \mathrm{diag}\left(\frac{1}{\sum_{j=1}^{n} m_{1j}}, \frac{1}{\sum_{j=1}^{n} m_{2j}}, \ldots, \frac{1}{\sum_{j=1}^{n} m_{nj}}\right)$;
4 **Construct** the decision matrix $DM_{n\times r} = (G(D_1), G(D_2), \ldots, G(D_r))$ in terms of the decision attribute D;
5 **Compute** the basic matrix $HD = \Lambda^{K_B}_{n\times n} \bullet RM^{K_B}_{n\times n} \bullet DM_{n\times r}$;
6 **Compute** and **Output** the approximations of the decision, the positive region, the boundary region and the negative region by using the cut matrices of HD.
7 **end**

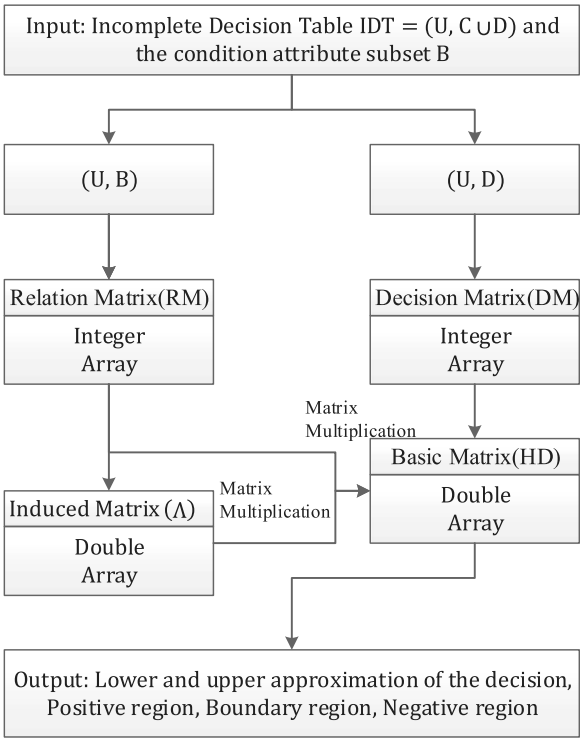


Fig. 2. The sequential method for computing rough set approximations in IDT. In order to facilitate the operations of matrix multiplication, we use integer arrays to construct the relation matrix and the decision matrix in our implementation.

Algorithm SMA (see Algorithm 1) is a sequential matrix-based algorithm for computing approximations of the decision, the positive region, the boundary region and the negative region. Step 2 is to construct the relation matrix according to Definition 11, whose time complexity is $O(|U|^2|C|)$. By Definition 12, Step 3 is to compute the induced diagonal matrix. Its time complexity is $O(|U|^2)$. Step 4 is to construct the decision matrix according to Definition 15, which costs $O(|U||D|)$. Step 5 is to compute the basic matrix HD by Corollary 5 with time complexity $O(|U|^2 r)$. Finally, Step 6 is to compute and output the approximations of the decision, the positive region, the boundary region and the negative region according to Corollary 6 with time complexity $O(|U|r)$. Therefore, the total complexity of Algorithm 2 is $O(|U|^2(|C|+r))$. In many real-applications, *e.g.*, gene selection [34] which includes thousands of condition attributes, there is only one decision attribute and a few decision classes. It means $|C| \gg |D|$ and $|C| \gg r$. Thus the most intensive computation to occur is the construction of the relation matrix. In following sections, we will discuss how to construct the relation matrix in parallel.

4 Parallel Matrix-Based Methods in Incomplete Decision Table

As mentioned above, the most intensive computation to occur is the construction of the relation matrix. To accelerate the process of the construction of the relation matrix, we use the MapReduce framework to parallelize this process.

4.1 S1: A Parallel Strategy Based on MapReduce

In this section, we give a simple parallel strategy based on MapReduce, called S1.

We first give the definition of the partition of IDT with regard to the attribute.

Definition 16. *Given an incomplete decision table* $IDT = (U, C \cup D)$. *Let* $S_t = (U, c_t)$, $t \in \{1, 2, \ldots, |C|\}$ *and* $S_D = (U, D)$. *It means that IDT is divided into* $|C|+1$ *sub-decision tables.*

Remark 2. The sub-decision table S_D is used to construct the decision matrix in sequential according to Definition 15. In order to show the merge operation of the matrices, we need to introduce the logical conjunction of two matrices firstly.

Definition 17. *Given two* $\mu \times \nu$ *boolean matrices* $Y = (y_{ij})_{\mu\times\nu}$ *and* $Z = (z_{ij})_{\mu\times\nu}$, *the logical conjunction* $\wedge$ *of two matrices is defined as*

$$Y \wedge Z = (y_{ij} \wedge z_{ij})_{\mu\times\nu}. \tag{32}$$

Actually, we partition IDT with regard to the attribute. Each attribute in IDT can generate a relation matrix. Here, we give the following lemma to show how to merge two relation matrices.

Lemma 1. *Let* $b_1, b_2 \in C$ *and* $K_{\{b_1,b_2\}}$, K_{b_1}, K_{b_2} *be the characteristic relation on* U. *Then*

$$RM_{n\times n}^{K_{\{b_1,b_2\}}} = RM_{n\times n}^{K_{b_1}} \wedge RM_{n\times n}^{K_{b_2}}. \tag{33}$$

Proof. Suppose $RM_{n\times n}^{K_{\{b_1,b_2\}}} = (m_{ij})_{n\times n}$, $RM_{n\times n}^{K_{b_1}} = (\varphi_{ij}^{b_1})_{n\times n}$ and $RM_{n\times n}^{K_{b_2}} = (\varphi_{ij}^{b_2})_{n\times n}$.

(a) If $(x_i, x_j) \in K_{\{b_1,b_2\}}$, then $m_{ij} = 1$, that is, x_i and x_j are indiscernible w.r.t. $\{b_1, b_2\}$. By Definition 7, we have $x_i K_{b_1} x_j$ and $x_i K_{b_2} x_j$. Therefore, $\varphi_{ij}^{b_1} = 1$ and $\varphi_{ij}^{b_2} = 1$. Hence, $m_{ij} = \varphi_{ij}^{b_1} \wedge \varphi_{ij}^{b_2} = 1$.

(b) If $(x_i, x_j) \notin K_{\{b_1,b_2\}}$, then $m_{ij} = 0$, that is, x_i and x_j are discernible w.r.t. $\{b_1, b_2\}$. By Definition 7, we have $(x_i, x_j) \notin K_{b_1}$ or $(x_i, x_j) \notin K_{b_2}$. Therefore, $\varphi_{ij}^{b_1} = 0$ or $\varphi_{ij}^{b_2} = 0$. Hence, $m_{ij} = \varphi_{ij}^{b_1} \wedge \varphi_{ij}^{b_2} = 0$.

To sum up, $\forall i, j \in \{1, 2, \ldots, n\}$, $m_{ij} = \varphi_{ij}^{b_1} \wedge \varphi_{ij}^{b_2}$, namely, $m_{n\times n}^{K_{b_1,b_2}} = m_{n\times n}^{K_{b_1}} \wedge m_{n\times n}^{K_{b_2}}$. □

Similarly, multiple relation matrices generated by different attributes can be merged according to the following corollary.

Corollary 7. *Let* $B \subseteq C$, K_B *be the characteristic relation on* U. *Then*

$$RM_{n\times n}^{K_B} = \bigwedge_{b\in B} RM_{n\times n}^{K_b} \tag{34}$$

According to Lemma 1, each condition attribute will generate a sub-relation matrix. For $IDT = (U, C \cup D)$, it will cost the space $|U|^2|C|$ to construct the relation matrix. As an example, there exists a data set with 10,000 objects each contains 1,000 attributes. It means the required memory space for storing an intermediate matrix of an attribute

is about 95 MB $(10,000^2/(2^{20}) \approx 95.36)$. If we apply it to analyze this data, then about 93 GB memory space (1,000 attributes $\times$ 95 MB) is required to save all the intermediate results. To reduce the required memory, we introduce a local merge operation, called Sub-Merge, which can be viewed as a local Reduce. With the example mentioned above, if eight compute nodes are utilized for the analysis, then 125 attributes $(1{,}000/8)$ would be assigned to each node. Each node processes these 125 attributes one by one. Before passing intermediate matrices to the Reduce phase, the proposed Sub-Merge operation merges these intermediate matrices. Each node only generates one intermediate matrix with the Sub-Merge, but generates 125 intermediate matrices without the Sub-Merge. With this operation, only 760 MB memory space (8 $\times$ 95 MB) is required to save all the intermediate results. Moreover, it requires only 95 MB memory space for each node. Without the Sub-Merge operation, each node needs 11875 MB memory space (125 $\times$ 95 MB), which is always greater than the memory of the node. To verify this viewpoint, we have conducted the experiment to test the algorithm without the Sub-Merge operation. Unfortunately, we always receive the error message "Out of Memory" since the requirement of memory exceeds the space we have in each node. Besides, the relation matrix is asymmetric only when processing the missing data "?" according to Definition 7. It means that the missing data "?" can be processed separately when there are only a few "?". In this situation, it can reduce about half of the space. In the later implementation, we only use the (a)symmetry property to construct the relation matrix; however, a full matrix is still stored since the improvement of its performance may not be significant when the memory space is enough and there are many missing data "?".

The Sub-Merge operation is based on the merge of relation matrices of different attribute sets. Therefore, we give a new definition of the partition of IDT with regard to the attribute set.

Definition 18. *Given an incomplete decision table* $IDT = (U, C \cup D)$. *Let* $S_t = (U, C_t)$, $t \in \{1, 2, \ldots, f\}$ *and* $S_D = (U, D)$. *It satisfies: (1)* $C = \bigcup_{t=1}^{f} C_t$; *(2)* $C_j \cap C_k = \emptyset$, $\forall j, k \in \{1, 2, \ldots, f\}$ *and* $j \neq k$. *It means that IDT is divided into* $f + 1$ *sub-decision tables.*

Be different from Definition 16, this definition shows that each sub-decision table includes an attribute set rather than a single condition attribute.

Lemma 2. *Let* $B_1, B_2 \subseteq C, B_1 \cap B_2 = \emptyset$ *and* $K_{\{B_1 \cup B_2\}}$, *where* K_{B_1}, K_{B_2} *are two characteristic relations on* U. *Then*

$$RM_{n\times n}^{K_{\{B_1\cup B_2\}}} = RM_{n\times n}^{K_{B_1}} \wedge RM_{n\times n}^{K_{B_2}}. \tag{35}$$

Proof. The proof is similar to that of Lemma 1. □

Corollary 8. *Let* $B = \bigcup_{t=1}^{f} B_t \subseteq C$, K_B *be the characteristic relation on* U. *Then*

$$RM_{n\times n}^{K_B} = \bigwedge_{t=1}^{f} RM_{n\times n}^{K_{B_t}} \tag{36}$$

Algorithm 2: A Parallel Matrix-based Algorithm (PMA) for Computation of Rough Set Approximations

1 **Call** the MapReduce module to construct the relation matrix;
2 **Construct** the decision matrix;
3 **Split** the relation matrix into a set of row blocks and the decision matrix into a set of column blocks, and **Send** the data blocks to different nodes;
4 **Call** the MapReduce based matrix operations module to compute the basic matrix;
5 **Output** the approximations of the decision, the positive region, the boundary region and the negative region;

According to Lemma 2, each condition attribute subset can generate a sub-relation matrix independently. And, the relation matrix can be constructed through the logical conjunction of sub-relation matrices according to Corollary 8. It is easy to know that the elements in the relation matrix are independent; thus, they can be computed in parallel. Fig. 3 gives the parallel strategy based on MapReduce. Intuitively, we also present the parallel algorithm (see Algorithm 2). Assume that the input data are split and stored in different nodes. In each Map, the sub-relation

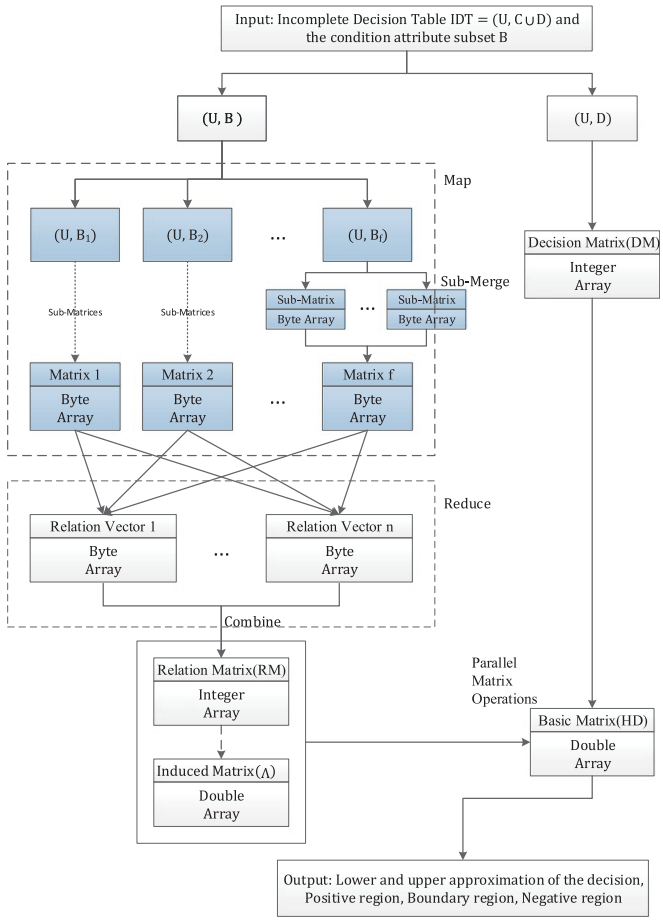


Fig. 3. A parallel strategy based on MapReduce. To reduce space complexity, we use byte arrays to storage the sub-relation matrices.

matrix is constructed. Its size is $n \times n$, where $n = |U|$. Simply, the row number of sub-relation matrix is set as the key of output in the Map phase. The key domain is $\{1, 2, \ldots, n\}$. It means that there are n tasks in the Reduce phase. Preferably, we can use a parameter η to control the number of tasks in the Reduce phase. Namely, the key domain is $\{1, 2, \ldots, \eta\}$. Specifically, (1) If $\eta = 1$, it is the coarsest level of granularity. Thus there is only one task in the Reduce phase, which processes $n \times n$ elements. (2) If $\eta = n \times n$, it is the finest level of granularity. Thus there are $n \times n$ tasks in the Reduce phase, each of which processes only one element. The Map and Reduce functions in S1 are outlined in Algorithms 3 and 4, respectively. Specially, the process Sub-Merge in the Map function is used to merge sub-relation matrices. After the Reduce phase, we use *Combine* to merge the relation vectors and construct the relation matrix.

Algorithm 3: S1-Map($key, value$)

Input:
key: The condition attribute B_t
$value$: The sub-decision table $S_t = (U, B_t)$
Output:
key': Row number in sub-relation matrix
$value'$: A byte array

```
1  begin
2  |  Let byte [][] RM^{K_{B_t}}_{n×n} = {1}; // n × n matrix
3  |  For each b ∈ B_t do
4  |  |  Construct the sub-relation matrix RM^{K_b}_{n×n}
5  |  |  RM^{K_{B_t}}_{n×n} = RM^{K_{B_t}}_{n×n} ∧ RM^{K_b}_{n×n};
6  |  end
7  |  For i = 1 to n do
8  |  |  key' = i;
9  |  |  value = RM^{K_{B_t}}_{n×n}(i); // the ith row of RM^{K_{b_t}}_{n×n}
10 |  |  output.collect(key', value');
11 |  end
12 end
```

Algorithm 4: S1-Reduce($key, value$)

Input:
key: Row number in sub-relation matrix
$value$: A list of byte array V
Output:
key': Row number in sub-relation matrix
$value'$: A byte array

```
1  begin
2  |  key' = key;
3  |  Let byte [] value' = {1}; // 1 × n vector
4  |  for each v ∈ V do
5  |  |  value' = value' & v; // & is And-operation
   |  |  of vectors
6  |  end
7  |  output.collect(key', value');
8  end
```

The only sequential step is the construction of the decision matrix, which is same to that decision matrix of Algorithm 1. As for the remaining matrix multiplications, we employ the MapReduce based matrix multiplication module that has been implemented in the runtime system Twister [5]. Before it, we need to split the relation matrix and the decision matrix. It is based on the column/row decomposition strategy.[1] Be different to the original matrix multiplication in Twister, we combine two matrix multiplications as one that can accelerate the process of the computation. According to Corollary 5, given the relation matrix $RM^{K_B}_{n\times n} = (m_{ik})_{n\times n}$ and the decision matrix $DM = (d_{kj})_{n\times r}$, one can write the basic matrix with the alternative form as

$$\begin{aligned} HD &= \Lambda^{K_B}_{n\times n} \bullet \left(RM^{K_B}_{n\times n} \bullet DM_{n\times r}\right) \\ &= \operatorname{diag}\left(\frac{1}{\sum_{k=1}^{n} m_{1k}}, \frac{1}{\sum_{k=1}^{n} m_{2k}}, \ldots, \frac{1}{\sum_{k=1}^{n} m_{nk}}\right) \\ &\quad \bullet \begin{pmatrix} \sum_{k=1}^{n} m_{1k}d_{k1} & \cdots & \sum_{k=1}^{n} m_{1k}d_{kr} \\ \sum_{k=1}^{n} m_{2k}d_{k1} & \cdots & \sum_{k=1}^{n} m_{2k}d_{kr} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} m_{nk}d_{k1} & \cdots & \sum_{k=1}^{n} m_{nk}d_{kr} \end{pmatrix} \\ &= \begin{pmatrix} \frac{\sum_{k=1}^{n} m_{1k}d_{k1}}{\sum_{k=1}^{n} m_{1k}} & \cdots & \frac{\sum_{k=1}^{n} m_{1k}d_{kr}}{\sum_{k=1}^{n} m_{1k}} \\ \frac{\sum_{k=1}^{n} m_{2k}d_{k1}}{\sum_{k=1}^{n} m_{2k}} & \cdots & \frac{\sum_{k=1}^{n} m_{2k}d_{kr}}{\sum_{k=1}^{n} m_{2k}} \\ \vdots & \ddots & \vdots \\ \frac{\sum_{k=1}^{n} m_{nk}d_{k1}}{\sum_{k=1}^{n} m_{nk}} & \cdots & \frac{\sum_{k=1}^{n} m_{nk}d_{kr}}{\sum_{k=1}^{n} m_{nk}} \end{pmatrix}. \end{aligned}$$

Therefore, after finishing the computations of the relation matrix and the decision matrix, the basic matrix HD can be computed directly. The alternative form also shows that the one-step matrix operations for computing HD has the similar form with the matrix multiplication. The column/row decomposition strategy can also be used here.

4.2 S2: An Incremental Parallel Strategy Based on MapReduce

To accelerate the process of computation of rough set approximations, Zhang et al. proposed an incremental algorithm to update the relation matrix when adding or deleting attributes in set-valued information systems [32]. In S1, we know that each condition attribute generates a sub-relation matrix each time and they are merged in the Sub-Merge of Map phase and the Reduce phase. The process of merging can be viewed as a process of adding attributes one by one. It is a typical incremental process. Based on these analysis, we present the incremental parallel strategy based on MapReduce, called S2.

Corollary 9. *[32] Let $P, Q \subseteq C$ and $Q \cap P = \emptyset$. Suppose $M^{K_{P\cup Q}}_{n\times n} = (m^{\uparrow}_{ij})_{n\times n}$ is a relation matrix representing $K_{P\cup Q}$, and $M^{K_P}_{n\times n} = (m_{ij})_{n\times n}$ is a relation matrix representing K_P. The relation matrix by adding Q to P can be updated as:*

$$m^{\uparrow}_{ij} = \begin{cases} 0, & m_{ij} = 0 \vee (x_i, x_j) \notin K_Q \\ 1, & m_{ij} = 1 \wedge (x_i, x_j) \in K_Q. \end{cases} \tag{37}$$

1. http://www.iterativemapreduce.org/samples.html#Matrix_Multiplication.

Remark 3. When adding Q to P, m_{ij} keeps constant while $m_{ij} = 0$.

Here, the flow chart of S2 is same to S1. The difference between S2 and S1 is that we use an incremental technique to accelerate the construction of the relation matrix in the Map phase by Corollary 9 as outlined in Algorithm 5. The other steps are same to those in S1.

Algorithm 5: S2-Map($key, value$)

Input:
key: The condition attribute B_t
value: The sub-decision table $S_t = (U, B_t)$
Output:
key′: Row number in sub-relation matrix
value′: A byte array

```
1  begin
2    Let byte [][] RM^{K_{B_t}}_{n×n} = {m_ij}= 1; // n × n matrix
3    for each b ∈ B_t do
4      for i = 1 to n do
5        for j = 1 to n do
6          if m_ij == 1 and (x_i, x_j) ∉ K_b then
           m_ij = 0;
7        end
8      end
9    end
10   for i = 1 to n do
11     key′ = i;
12     value = RM^{K_{B_t}}_{n×n}(i); // the ith row of RM^{K_{b_t}}_{n×n}
13     output.collect(key′, value′);
14   end
15 end
```

4.3 S3: An Incremental Parallel Strategy Using the Sparse Matrix Based on MapReduce

A sparse matrix is a matrix populated primarily with zeros as elements of the table [35]. The native data structure for a matrix is a two-dimensional array, which is used in S1 and S2. Each entry in the array represents an element m_{ij} of the matrix and can be accessed by the two indices *i* and *j*. In tradition, *i* represents the row number (top-to-bottom), while *j* represents the column number (left-to-right) of each element in the table. For an $m \times n$ matrix, enough memory to store up to $(m \times n)$ entries to represent the matrix is required. As the number of condition attributes increases, there are more and more zero entries in the relation matrix. Reduction of substantial memory requirement can be accomplished by storing only the non-zero entries. Depending on the number and distribution of the non-zero entries, different data structures can be used to save significant amount of memory when compared to the native method.

Storing formats of the sparse matrix can be divided into two groups: (1) those that support efficient matrix operations and (2) those that support efficient modification. The incremental construction of the relation matrix in S2 shows the most operations are matrix update (modification). Thus, we select the latter format. The efficient modification group includes List of lists (LIL), Dictionary of keys (DOK) and Coordinate list (COO), which is typically used to construct the matrix. In our implementation, we use LIL to construct the relation matrix in parallel. LIL stores one list per row, where each entry stores a column index as the value. Typically, these entries are sorted by the column index for faster lookup. This is another format which is good for incremental matrix construction. Since our relation matrix is $(0, 1)$ matrix, we use a simplified LIL that also stores one list per row but each entry only stores a column index. We here define the sparse degree of the relation matrix as follows.

Algorithm 6: S3-Map($key, value$)

Input:
key: The condition attribute B_t
value: The sub-decision table $S_t = (U, B_t)$
Output:
key′: Row number in sub-relation matrix
value′: A list

```
1  begin
2    for i = 1 to n do
3      Let LIL(i) = Null; // LIL is list of list, LIL(i) is
       the ith list in LIL
4    end
5    Select an attribute b′ from B;
6    for i = 1 to n do
7      for j = 1 to n do
8        if (x_i, x_j) ∈ K_{b′} then LIL(i).add(j);
9      end
10   end
11   for each b ∈ {B_t − b′} do
12     for i = 1 to n do
13       for j = 1 to n do
14         if(x_i, x_j) ∉ K_b then
           LIL(i).remove(j);
15       end
16     end
17   end
18   for i = 1 to n do
19     key′ = i;
20     value = LIL(i);
21     output.collect(key′, value′);
22   end
23 end
```

Definition 19. *Given a relation matrix $RM^{K_B}_{n\times n} = (m_{ij})_{n\times n}$. K_B is a characteristic relation on U. The sparse degree of the relation matrix, denoted by SD_B, is defined as*

$$SD_B = 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij}}{n \times n}. \tag{38}$$

Lemma 3. *Let $P \subseteq Q \subseteq C$, $M^{K_P}_{n\times n} = (m_{ij})_{n\times n}$ and $M^{K_Q}_{n\times n} = (m'_{ij})_{n\times n}$ be relation matrices representing K_P and K_Q, respectively. Their sparse degrees are SD_P and SD_Q, respectively. Then*

$$SD_P \leq SD_Q. \tag{39}$$

Proof. According to Corollary 9, when adding attributes, m_{ij} keeps constant while $m_{ij} = 0$. Since $P \subseteq Q$, we have

$m_{ij} \geq m'_{ij}, 1 \leq i, j \leq n$. Hence, $\sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij} \geq \sum_{i=1}^{n}\sum_{j=1}^{n} m'_{ij}$. Therefore, $1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij}}{n \times n} \leq 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} m'_{ij}}{n \times n}$, in other words, $SD_P \leq SD_Q$. □

Remark 4. When there are more attributes in IIS, the sparse degree of the relation matrix would be higher, in other words, the relation matrix would be sparser.

According to these observations, we propose the incremental parallel strategy using the sparse matrix based on MapReduce, called S3. We here use LIL to redesign our Map and Reduce function, as outlined in Algorithms 6 and 7, respectively. The other steps are similar to those in S1. The most difference is that we use the sparse matrix to replace the matrix and some matrix operations, e.g., matrix multiplication, are replaced as the sparse matrix operations. The column/row decomposition strategy is also used here.

Algorithm 7: S3-Reduce($key, value$)

Input:
key: Row number in sub-relation matrix
$value$: A list of list V
Output:
key': Row number in sub-relation matrix
$value'$: A list

```
1 begin
2   key' = key;
3   Let value' = ALL = {1, 2, ..., n};
4   for each v ∈ V do
5     value' = value' ∩ v; // ∩ is the intersection
        operation of two lists
6   end
7   output.collect(key', value');
8 end
```

5 Experimental Analysis

Our experiments run on the representative MapReduce runtime system Twister [5]. Its version is 0.9. And we code our algorithms (a pure sequential version and a parallel Twister version) in JAVA and compile the programs with JDK1.7.0_05. The experiments on Twister are carried out on Cheetah cluster [36] using nine nodes, which contains one head node and eight compute nodes. Four compute nodes have 16 GB main memory and use AMD Opteron(TM) Processor 2376 CPUs (8 cores in all, each has a clock frequency of 2.3 GHz). Another four nodes have 8 GB main memory and use Intel Xeon CPU E5410, comprising two Quad-Core CPUs (eight cores in all, each has a clock frequency of 2.33 GHz). We here compare the performance using 1, 2, 4, 8, 16 and 32 cores.[2]

We download two data sets from machine learning data repository, University of California at Irvine [37]. They are "Balance Scale" and "Mushroom". In data set "Balance Scale", there are 625 samples, one decision attribute and four condition attributes. Another data set "Mushroom" consists of 8,124 samples. Each sample consists of one decision attribute and 22 condition attributes, which includes missing values. We only test the performance in this paper. Hence, we set half of missing values as "*", and the other half as "?". To test the performance of parallel methods, the data sets are used as input many times to stimulate the large input condition. Take the data set "Mushroom" as an example, we keep the number of sample constant and input the condition attributes repeatedly. If we repeat 32 times, it would have 704 (22×32) attributes. We call this data set S8k-A0.7k. All simulated data sets are outlined in Table 2.

TABLE 2
A Description of Data Sets

Data Sets	Rename	Samples	Attributes	Classes	Size
Balance Scale	S625-A8k	625	8192	3	16MB
	S625-A82k	625	81920	3	158MB
	S625-A819k	625	819200	3	1.37GB
	S625-A4096k	625	4096000	3	5.43GB
Mushroom	S8k-A0.7k	8124	704	2	11MB
	S8k-A2.8k	8124	2816	2	45MB
	S8k-A23k	8124	22528	2	357MB
	S8k-A45k	8124	45056	2	715MB

Note: S625-A8k means 625 samples and about 8k attributes.

5.1 Performance

The aforementioned three parallel strategies (see Section 4) are executed on the MapReduce runtime system Twister.

In what follows, we first test the performance of Strategy 1, which is a parallel method without the incremental technique. We examine its speedup characteristics. It is defined by the following formula [38]:

$$Speedup(p) = \frac{T_1}{T_p}, \tag{40}$$

where p is the number of cores, T_1 is the execution time on one core, and T_p is the execution time on p cores.

The linear speedup is a system with p computers yields a speedup of p. It is ideal but difficult to achieve because the communication cost increases when the cluster becomes large.

We perform the execution time and the speedup evaluation on data sets with various sizes and structures. The number of cores varies from 1 to 32. According to the parallel method of Section 4.1, the only sequential portion in Strategy 1 is the construction of the decision matrix. Therefore, we recorded the execution time of the construction of the decision matrix and other steps separately when testing the pure sequential version. Then, the expected speedup number can be obtained by using Amdahl's Law with the following formula [39]:

$$Expected\ Speedup(p) = \frac{1}{R_S + \frac{R_P}{p}}, \tag{41}$$

where $R_S + R_P = 1$, R_S represents the ratio of the sequential portion in the program.

2. The single core is used to test the performance of the pure sequential version.

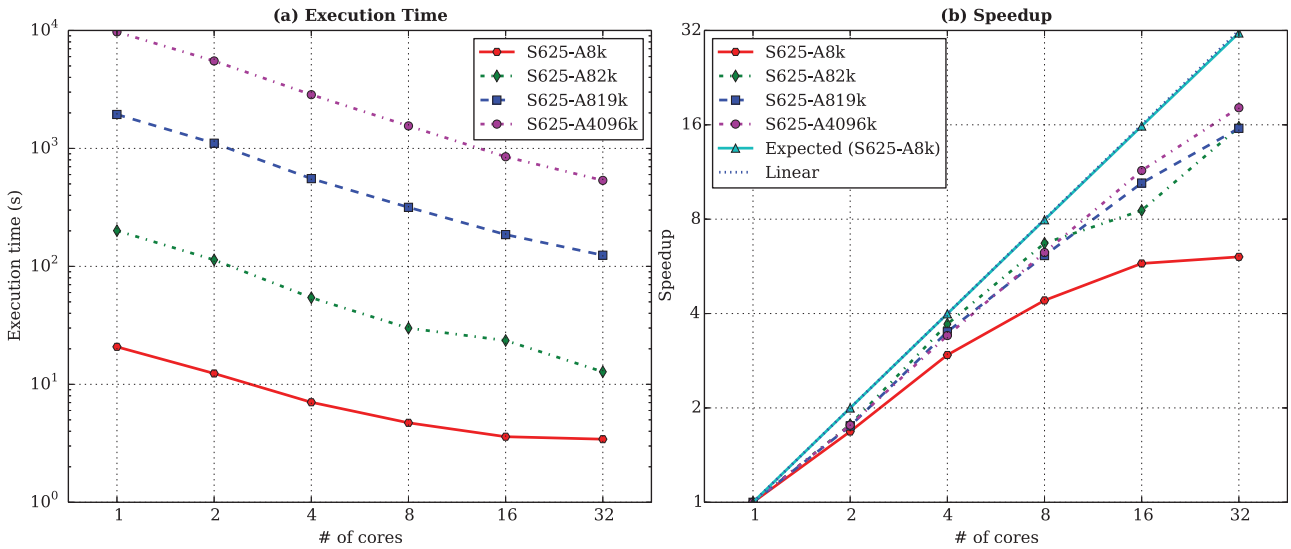


Fig. 4. Execution Time and Speedup plots using parallel Strategy 1 on simulated data sets of Balance Scale.

Figs. 4 and 5 show the execution time and the speedup for all these data sets. The expected speedup of S625-A8k (Balance Scale) and S8k-A0.7k (Mushroom) are shown in Figs. 4b and 5b, respectively. Both are very close to the linear speedup because the only sequential portion (the construction of the decision matrix) can be computed efficiently. The expected speedup of the other three simulated data sets of Balance Scale and Mushroom are not shown here because it is obvious that they are better than S625-A8k and S8k-A0.7k, respectively. As the results show, the proposed parallel algorithms have a great speedup performance. As the size of the data set increases, the speedup becomes better. For example, the size of the data set S625-A4096k is 5.43 GB. Its speedup on 32 core is greater than 18. Therefore, the parallel Strategy 1 can process large-scale data efficiently.

5.2 A Comparison of Different Parallel Strategies

In this section, we show a comparison of different parallel strategies. We here only compare the execution time of different parallel strategies rather than speedup because Strategies 2 and 3 (S2, in Section 4.2 and S3, in Section 4.3) use an incremental technique to accelerate the update process, whose time complexity depends on the characteristics of the testing data. Fig. 6 shows the execution time of Strategies 1, 2 and 3 versus the number of cores. It is obvious that S2 and S3 always have the better performance than S1. That is because we utilize the incremental method in S2 and S3 to facilitate the Sub-Merge operation. With the incremental method, further processes of merge (update) operation are only executed while the values in temporary result matrix are 1. Instead, it always executes $n \times n$ (n is the numbers of objects) operations in S1. Therefore, many redundant processes can be skipped in S2 and S3.

In S3, we use the sparse matrix to optimize the storing space, which is different from the data structure used in S1 and S2, since the relation matrix constructed from large data is usually sparse. Thus, we implement the sparse relation matrix with the data structure LIL in S3, which allows us to change the size of a sparse matrix dynamically, and the space requirement can be decreased as well.

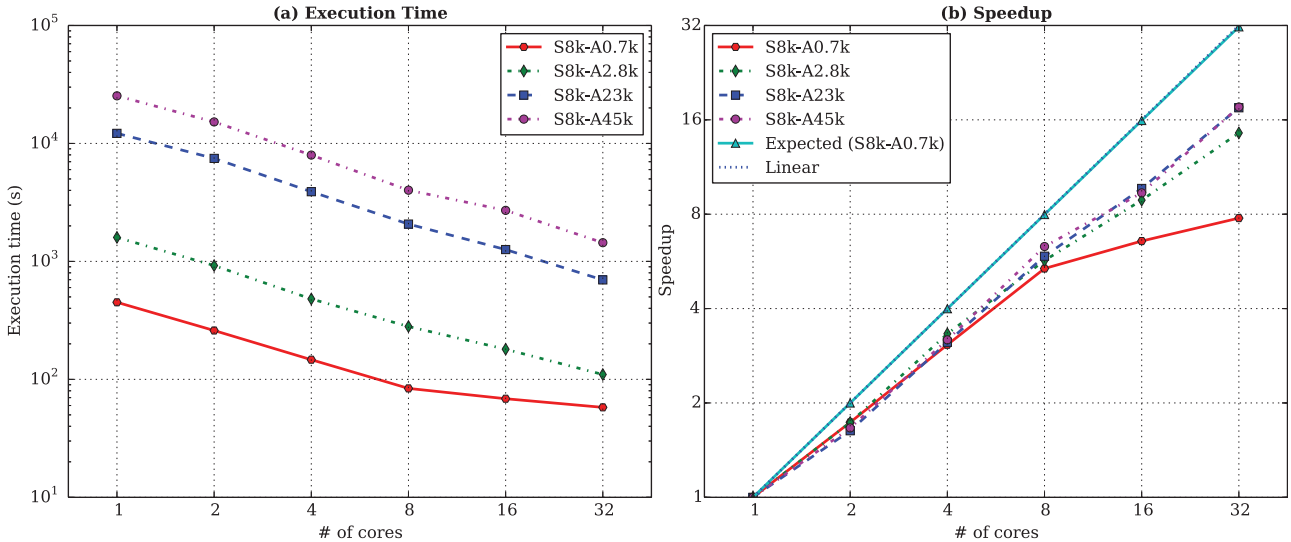


Fig. 5. Execution Time and Speedup plots using parallel Strategy 1 on simulated data sets of Mushroom.

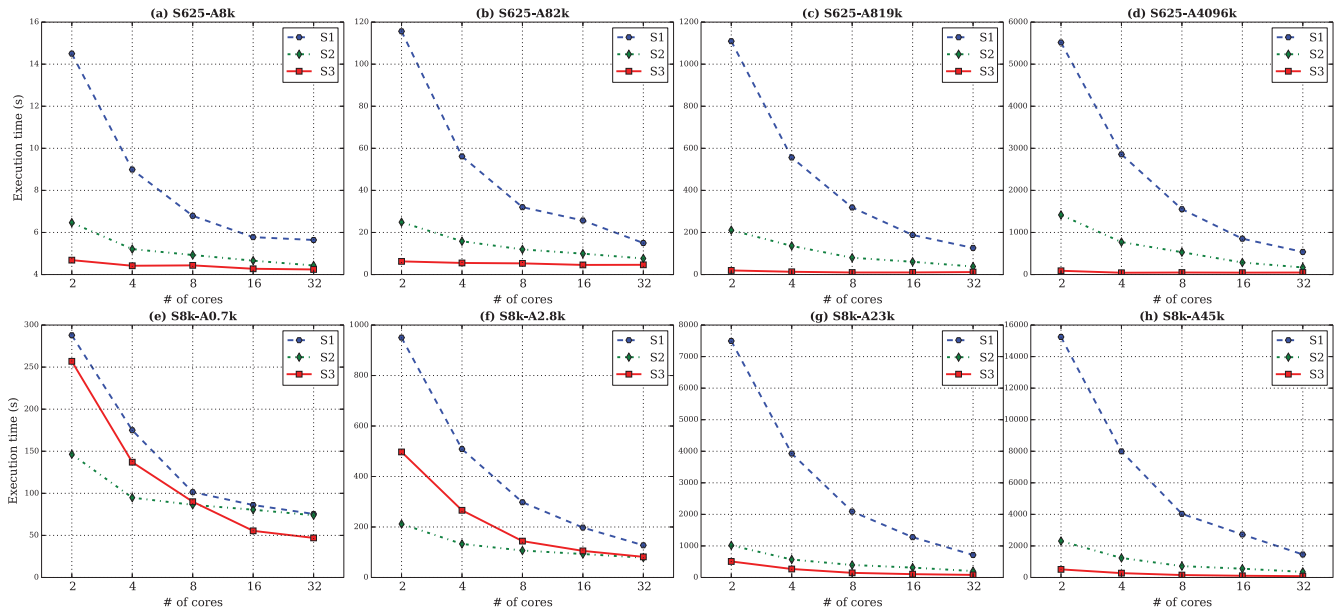


Fig. 6. Execution time of different parallel strategies versus the number of cores. S1: Strategy 1; S2: Strategy 2; S3: Strategy 3.

From Fig. 6, even though the execution time using S3 is faster than that using S2 in most conditions, S2 still has the better performance than S3 while the testing data contains huge numbers of objects and only a few attributes. Since the matrix structure is used in S2, a traverse process in an $n \times n$ relation matrix requires $n \times n$ elements to be examined. However, LIL structure traversal in S3 only needs to look over the elements remaining in the structure. Owing to the feature of LIL structure, the sparse degree of the relation matrix has a significant effect on the performance of S3 which means the higher sparse degree of the LIL structure has, the fewer elements need to be traversed. In this condition, S3 performs better than S2. But, in some cases, S2 is superior to S3 such as the result shown in Fig. 6f. Two possible reasons are given as follows: 1) S3 takes much more time to initialize the matrix while the size of objects increases significantly; 2) The size of generated intermediate matrix will not be shrunk significantly because of only few attributes using S3 which causes the communication overhead.

Based on these observations, we have the following qualitative results: 1) S3 performs better than S2 when the sparse degree of the relation matrix is higher that may be caused by a large number of attributes; 2) S2 performs better than S3 when the sparse degree of the relation matrix is lower. In real applications, we can choose the incremental strategies S2 or S3 to accelerate processes of execution according to different sparse degrees.

6 Conclusions

In this paper, to process large-scale incomplete data with rough set theory, we first introduced the matrix representations of lower and upper approximations in IIS. According to the characteristics of the matrix, we proposed three parallel strategies based on MapReduce to compute approximations. All of them were implemented on the MapReduce runtime system Twister. The results of testing speedup of S1 show that the parallel strategy S1 has a good performance on processing large-scale data. In examining an incremental method to accelerate the processing procedures in S2 and S3, we found that S2 and S3 always have better performance than S1, and, in most cases, S3 outperforms S2. But, if the input data contains huge quantities of objects and only a few attributes, S2 would do better than S3. Our extensive experimental evaluation demonstrates that our proposed parallel methods are more efficient in analyzing data with large amount of attributes. As we can see, computing lower and upper approximations is the key to rule induction and feature selection when utilizing rough set-based methods. We plan in our future work to further investigate the knowledge discovered and features selection from large-scale, incomplete data.

Acknowledgments

The authors would like to thank Alexander G. Ororbia II and all anonymous reviewers for their constructive comments, which helped improve the presentation of this paper. This work is supported by the National Science Foundation of China (Nos. 60873108, 61175047, 61100117) and NSAF (No. U1230117), the Fundamental Research Funds for the Central Universities (No. SWJTU11ZT08), the Research Fund of Traction Power State Key Laboratory, Southwest Jiaotong University (No. 2012TPL_T15), the Science and Technology Planning Project of Sichuan Province (No. 2012RZ0009), China, and the Fostering Foundation for the Excellent Ph.D. Dissertation of Southwest Jiaotong University (No. 2012ZJB), China. T. Li is the corresponding author.

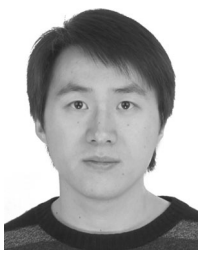
References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Oper. Syst. Des. Implementation*, 2004, pp. 10–10.

[2] J. Dean, and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[3] J. B. Zhang, D. Xiang, T. R. Li, and Y. Pan, "M2M: A simple Matlab-to-MapReduce translator for cloud computing," *Tsinghua Sci. Technol.*, vol. 18, no. 1, pp. 1–9, Feb. 2013.

[4] T. White, *Hadoop: The Definitive Guide*, 2nd ed. O'Reilly Media/ Yahoo Press, Sebastopol, CA, USA, 2010.
[5] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative MapReduce," in *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.*, 2010, pp. 810–818.
[6] *Amazon Elastic MapReduce*. (2013). [Online]. Available: http://aws.amazon.com/elasticmapreduce/
[7] *Twister4Azure*. (2013). [Online]. Available: http://salsahpc.indiana.edu/twister4azure/index.html
[8] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data, System Theory, Knowledge Engineering and Problem Solving*, vol. 9. Norwell, MA, USA: Kluwer, 1991.
[9] Z. Pawlak and A. Skowron, "Rudiments of rough sets," *Inf. Sci.*, vol. 177, no. 1, pp. 3–27, 2007.
[10] Z. Pawlak and A. Skowron, "Rough sets: Some extensions," *Inf. Sci.*, vol. 177, no. 1, pp. 28–40, 2007.
[11] Z. Pawlak and A. Skowron, "Rough sets and boolean reasoning," *Inf. Sci.*, vol. 177, no. 1, pp. 41–73, 2007.
[12] J. Y. Liang, F. Wang, C. Y. Dang, and Y. H. Qian, "A group incremental approach to feature selection applying rough set technique," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 2, pp. 294–308, Feb. 2014.
[13] J. W. Grzymala-Busse, and W. Ziarko, "Data mining and rough set theory," *Commun. ACM*, vol. 43, no. 4, pp. 108–109, Apr. 2000.
[14] W. Ziarko, "Discovery through rough set theory," *Commun. ACM*, vol. 42, no. 11, pp. 54–57, Nov. 1999.
[15] Q. H. Hu, W. Pedrycz, D. R. Yu, and J. Lang, "Selecting discrete and continuous features based on neighborhood decision error minimization," *IEEE Trans. Syst., Man, Cybern., Part B: Cybern.*, vol. 40, no. 1, pp. 137–150, Feb. 2010.
[16] Q. H. Hu, Z. X. Xie, and D. R. Yu, "Hybrid attribute reduction based on a novel fuzzy-rough model and information granulation," *Pattern Recognit.*, vol. 40, no. 12, pp. 3509–3521, Dec. 2007.
[17] Q. H. Hu, D. R. Yu, J. F. Liu, and C. X. Wu, "Neighborhood rough set based heterogeneous feature subset selection," *Inf. Sci.*, vol. 178, no. 18, pp. 3577–3594, Sep. 2008.
[18] Y. H. Qian, C. Y. Dang, J. Y. Liang, and D. Tang, "Set-valued ordered information systems," *Inf. Sci.*, vol. 179, no. 16, pp. 2809–2832, Jul. 2009.
[19] Y. H. Qian, J. Y. Liang, W. Pedrycz, and C. Y. Dang, "An efficient accelerator for attribute reduction from incomplete data in rough set framework," *Pattern Recognit.*, vol. 44, no. 8, pp. 1658–1670, Aug. 2011.
[20] Y. H. Qian, J. Y. Liang, W. Pedrycz, and C. Y. Dang, "Positive approximation: An accelerator for attribute reduction in rough set theory," *Artif. Intell.*, vol. 174, no. 9/10, pp. 597–618, Jun. 2010.
[21] J. B. Zhang, T. R. Li, D. Ruan, and D. Liu, "Neighborhood rough sets for dynamic data mining," *Int. J. Intell. Syst.*, vol. 27, no. 4, pp. 317–342, 2012.
[22] J. B. Zhang, T. R. Li, D. Ruan, Z. Z. Gao, and C. B. Zhao, "A parallel method for computing rough set approximations," *Inf. Sci.*, vol. 194, pp. 209–223, Jul. 2012.
[23] J. B. Zhang, J.-S. Wong, T. R. Li, and Y. Pan, "A comparison of parallel large-scale knowledge acquisition using rough set theory on different MapReduce runtime systems," *Int. J. Approximate Reasoning*, vol. 55, no. 3, pp. 896–907, Mar. 2014.
[24] T. R. Li, D. Ruan, W. Geert, J. Song, and Y. Xu, "A rough sets based characteristic relation approach for dynamic attribute generalization in data mining," *Knowl.-Based Syst.*, vol. 20, no. 5, pp. 485–494, Jun. 2007.
[25] M. Kryszkiewicz, "Rough set approach to incomplete information systems," *Inf. Sci.*, vol. 112, no. 1–4, pp. 39–49, Dec. 1998.
[26] M. Kryszkiewicz, "Rules in incomplete information systems," *Inf. Sci.*, vol. 113, no. 3/4, pp. 271–292, Feb. 1999.
[27] J. Stefanowski, and A. Tsoukiàs, "On the extension of rough sets under incomplete information," in *Proc. 7th Int. Workshop New Directions Rough Sets, Data Min., Granular-Soft Comput.*, 1999, pp. 73–81.
[28] J. W. Grzymala-Busse, "Characteristic relations for incomplete data: A generalization of the indiscernibility relation," in *Trans. Rough Sets IV*, vol. 3700, ser. Lecture Notes in Computer Science, J. Peters, and A. Skowron, Eds. Berlin, Germany: Springer, 2005, pp. 58–68.
[29] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *Proc. IEEE 13th Int. Symp. High Perform. Comput. Archit.*, 2007, pp. 13–24.
[30] B. S. He, W. B. Fang, Q. Luo, N. K. Govindaraju, and T. Y. Wang, "Mars: A MapReduce framework on graphics processors," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Technol.*, 2008, pp. 260–269.
[31] G. L. Liu, "The axiomatization of the rough set upper approximation operations," *Fundamenta Informat.*, vol. 69, no. 3, pp. 331–342, 2006.
[32] J. B. Zhang, T. R. Li, D. Ruan, and D. Liu, "Rough sets based matrix approaches with dynamic attribute variation in set-valued information systems," *Int. J. Approximate Reasoning*, vol. 53, no. 4, pp. 620–635, 2012.
[33] G. L. Liu, "Axiomatic systems for rough sets and fuzzy rough sets," *Int. J. Approximate Reasoning*, vol. 48, no. 3, pp. 857–867, Aug. 2008.
[34] M.-L. Hou, S.-L. Wang, X.-L. Li, and Y.-K. Lei, "Neighborhood rough set reduction-based gene selection and prioritization for gene expression profile analysis and molecular cancer classification," *J. Biomed. Biotechnol.*, vol. 2010, p. 726413, 2010.
[35] J. Stoer, and R. Bulirsch, *Introduction to Numerical Analysis*, 3rd ed. New York, NY, USA: Springer-Verlag, 2002.
[36] *Cheetah cluster in Georgia State University*. (2013). [Online]. Availablel: http://cs.gsu.edu/?q=cheetah
[37] D. Newman, S. Hettich, C. Blake, and C. Merz. (1998). UCI repository of machine learning databases. Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, USA. [Online]. Available: http://archive.ics.uci.edu/ml/
[38] X. W. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Min. Knowl. Discov.*, vol. 3, no. 3, pp. 263–290, 1999.
[39] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities." in *Proc. Spring Joint Comput. Conf.*, 1967, pp. 483–485.



Junbo Zhang received the BEng degree from Southwest Jiaotong University, China, in 2009. He is currently working toward the PhD degree at the School of Information Science and Technology, Southwest Jiaotong University, China. From February 2012 to February 2013, he was a visiting PhD student at the Department of Computer Science, Georgia State University. His research interests include deep learning, big data mining, cloud computing, and rough sets. He is currently a student member of the ACM and CCF.

Jian-Syuan Wong received the BS degree from National Sun Yat-Sen University, Taiwan, in 2008, and the MS degree from Georgia State University, in 2012. He is currently working toward the PhD degree at the College of Information Sciences and Technology, Pennsylvania State University. He was a research assistant at the Morehouse School of Medicine. His research interests include cloud computing, parallel and distributed systems, and data mining.

Yi Pan received the BEng and MEng degrees in computer engineering from Tsinghua University, China, in 1982 and 1984, respectively, and the PhD degree in computer science from the University of Pittsburgh in 1991. He is a Distinguished university professor in the Department of Computer Science and interim associate dean of arts and sciences at Georgia State University. His research interests include parallel and cloud computing, wireless networks, optical networks, algorithms, and bioinformatics. He has published more than 330 papers including over 150 SCI journal papers and 50 IEEE Transactions papers. In addition, he has edited/authored 39 books. He has served as an editor-in-chief or editorial board member for 15 journals including seven IEEE Transactions and a guest editor for 12 special issues for 10 journals including two IEEE Transactions. He has organized numerous international conferences and workshops and has delivered more than 40 keynote speeches at international conferences around the world. He received many awards including IEEE Transactions Best Paper Award, IBM Faculty Award, MSN Best Paper Award, JSPS Senior Invitation Fellowship, IEEE BIBE Outstanding Achievement Award, NSF Research Opportunity Award, and AFOSR Summer Faculty Research Fellowship. He is a senior member of the IEEE.

Tianrui Li (SM'10) received the BS, MS, and PhD degrees from Southwest Jiaotong University, Chengdu, China, in 1992, 1995, and 2002, respectively. He was a postdoctoral researcher with SCK•CEN, Belgium, from 2005 to 2006, and a visiting professor with Hasselt University, Belgium, in 2008, the University of Technology, Sydney, Australia, in 2009, and the University of Regina, Canada, in 2014. He is currently a professor and the director of the Key Laboratory of Cloud Computing and Intelligent Techniques, Southwest Jiaotong University. He has authored or coauthored more than 100 research papers in refereed journals and conferences. His research interests include big data, cloud computing, data mining, granular computing, and rough sets. He is a senior member of the IEEE.