Generating Core Based on Discernibility Measure and MapReduce

Michal Czolombitko(✉) and Jaroslaw Stepaniuk

Faculty of Computer Science, Bialystok University of Technology,
Wiejska 45A, 15-351 Bialystok, Poland
{m.czolombitko,j.stepaniuk}@pb.edu.pl
http://www.wi.pb.edu.pl

Abstract. In this paper we propose a parallel method for generating attribute core based on distributed programming model MapReduce and rough set theory. The results of the experiments on real dataset show that the proposed method is effective for big data.

Keywords: Rough sets · MapReduce · Core

1 Introduction

Since the massive data could be stored in cloud platforms, data mining for the large datasets is hot topic. Parallel methods of computing are alternative for large datasets processing and knowledge discovery for large data. MapReduce is a distributed programming model, proposed by Google for processing large datasets, so called Big Data. Users specify the required functions Map and Reduce and optional function Combine. Every step of computation takes as input pairs $< key, values >$ and produces another output pairs $< key', values' >$. In the first step, the Map function reads the input as a set $< key, values >$ pairs and applies user defined function to each pair. The result is a second set of the intermediate pairs $< key', values' >$, sent to Combine or Reduce function. Combine function is a local Reduce, which can help to reduce final computation. It applies second user defined function to each intermediate key with all its associated values to merge and group data. Results are sorted, shuffled and sent to the Reduce function. Reduce function merges and groups all values to each key and produces zero or more outputs.

Rough set theory is mathematical tool for dealing with incomplete and uncertain information [4]. The notion of core is very important in the rough set theory. In decision table some of the condition attributes may be superfluous (redundant in other words). This means that their removal cannot worsen the classification. The set of all indispensable condition attributes is called the core. One can also observe that the core is the intersection of all decision reducts – each element of the core belongs to every reduct. Thus, in a sense, the core is the most important subset of condition attributes. None of its elements can be removed without affecting the classification power of all condition attributes. A much

M. Kryszkiewicz et al. (Eds.): PReMI 2015, LNCS 9124, pp. 367–376, 2015.
DOI: 10.1007/978-3-319-19941-2_35

more detailed description of the concept of the core can be found, for example, in the book [6] or in the article [4].

There are some research works combining MapReduce and rough set theory. In [10] parallel method for computing rough set approximations was proposed. In [9] method for computing core based on finding positive region was proposed. They also presented parallel algorithm of attribute reduction in [8]. In [2] is proposed a design of a Patient-customized Healthcare System based on the Hadoop with Text Mining for an efficient Disease Management and Prediction.

In this paper we propose parallel method for generating attribute core based on distributed programming model MapReduce and rough set theory. This paper is organized as follows. Section 1 includes background introduction to rough sets and two algorithms of generating core. Parallel algorithm based on discernibility measure of a set and MapReduce is proposed in Sect. 3. Results of experiments and analysis is presented in Sect. 4. Conclusions and future work are drawn in the last Section.

2 Pseudocode for Generating Core

2.1 Pseudocode for Generating Core Using Discernibility Matrix

In order to compute the core we can use discernibility matrix introduced by Prof. Andrzej Skowron (see e.g. [4,6]). The core is the set of all single element entries of the discernibility matrix.

Let $DT = (U, A \cup \{d\})$ be a decision table, where U is a set of objects, A is a set of condition attributes and d is a decision attribute. Below one can find pseudocode for an algorithm of calculating core $C \subseteq A$ using discernibility matrix $[DM(x,y)]_{x,y \in U}$, where $DM(x,y) = \{a \in A : a(x) \neq a(y) \text{ and } d(x) \neq d(y)\}$.

INPUT: discernibility matrix $[DM(x,y)]_{x,y \in U}$
OUTPUT: core $C \subseteq A$

```
1: C ← ∅
2: for x ∈ U do
3:    for y ∈ U do
4:       if cardinality(DM(x,y)) = 1 and DM(x,y) ⊈ C then
5:          C ← C ∪ DM(x,y)
6:       end if
7:    end for
8: end for
```

The main concept of this algorithm is based on a property of singletons i.e. cells from discernibility matrix consisted of the only one attribute. This property tells that any singleton cannot be removed without affecting the classification power. Input for the algorithm is the discernibility matrix DM. Output is core C consisting of a subset of condition attributes set denoted as A. Core is initialized as empty set in line 1. Two loops in lines 2 and 3 are responsible for iteration over all objects in discernibility matrix. In the condition instruction in line 4 it

is checked if a matrix cell contains only one attribute. If so, then this attribute is added to the core C.

The main disadvantage of using discernibility matrix for big datasets is its size. Memory complexity of creating this type of square matrix is equal to $(cardinality(U))^2 * cardinality(A)$. This makes the algorithm showed in this subsection infeasible for big data. In the next subsection we present an approach more feasible for big data.

2.2 Pseudocode for Generating Core Based on Discernibility Measure of a Set of Attributes

Generating core based on discernibility measure was discussed in [3]. Counting table CT is a two-dimensional array indexed by values of information vectors (vector of all values of an attribute set $B \subseteq A$) and decisions values, where

$$CT(i,j) = cardinality(\{x \in U : \vec{x}_B = i \text{ and } d(x) = j\})$$

Pessimistic memory complexity of creating this type of matrix is equal to $(caridinality(U) * cardinality(V_d))$, where V_d is a set of all decisions.
The discernibility measure $disc(B)$ of set of attributes $B \subseteq A$ can be calculated from the counting table as follows:

$$disc(B) = \frac{1}{2} \sum_{i,j} \sum_{k,l} CT(i,j) \cdot CT(k,l), \text{if } i \neq k \text{ and } j \neq l$$

Below is the pseudocode for this algorithm:

INPUT: decision table $DT = (U, A \cup \{d\})$
OUTPUT: core $C \subseteq A$

$C \leftarrow \emptyset$ {C is equal to empty set}
$CT \leftarrow 0$ {All values in counting table CT are equal to 0}
for $x \in U$ **do**
  $CT(\vec{x}_A, d(x)) \leftarrow CT(\vec{x}_A, d(x)) + 1$
end for
$disc(A) \leftarrow 0$
for $[x]_A \in U/IND(A)$ {$U/IND(A)$ is partition of U defined by A} **do**
  for $[y]_A \in U/IND(A)$ **do**
    if $\vec{x}_A \neq \vec{y}_A$ **and** $d(x) \neq d(y)$ **then**
      $disc(A) \leftarrow disc(A) + CT(\vec{x}_A, d(x)) \cdot CT(\vec{y}_A, d(y))$
    end if
  end for
end for
$CT \leftarrow 0$
for $a \in A$ **do**
  $B \leftarrow A - \{a\}$
  $disc(B) \leftarrow 0$
  for $x \in U$ **do**

19: $CT(\vec{x}_B, d(x)) \leftarrow CT(\vec{x}_B, d(x)) + 1$
20: **end for**
21: **for** $[x]_B \in U/IND(A)$ **do**
22: **for** $[y]_B \in U/IND(A)$ **do**
23: **if** $\vec{x}_B \neq \vec{y}_B$ **and** $d(x) \neq d(y)$ **then**
24: $disc(B) \leftarrow disc(B) + CT(\vec{x}_B, d(x)) \cdot CT(\vec{y}_B, d(y))$
25: **end if**
26: **end for**
27: **end for**
28: **if** $disc(B) < disc(A)$ **then**
29: $C \leftarrow C \cup \{a\}$
30: **end if**
31: **end for**

Input to the algorithm is a decision table DT, and output is the core C of DT. In the beginning core C is initialized as empty set and all values in the counting table are set to zero. First loop in line 3 generate counting table for set of all conditional attributes. For each object in decision table, value in array is increased by one. Indexes of value are information vector of object and its value of decision. In the line 6 value of discernibility measure of set of all condition attributes, $disc(A)$, is initialized as zero. Next two loops in lines 7 and 8 take subsequent equivalence classes to comparison. If information vectors of these classes are not equal, $disc(A)$ is increased by product of two values from the counting tables where indexes are values information vectors and different decisions. Next step is computing the discernibility measure of set of attributes after removing one of them. In line 14 all values in the counting table are set to zero. Loop in line 15 takes attribute a from set of all condition attributes A. Set B is initialized as set of all condition attributes after removing this attribute. Similarly as above, in lines 15–27 is calculated $disc(B)$. Finally, values of the discernibility measure of set of all attributes and after removing attribute a are compared in the line 28. In case of difference, this attribute is added to the core.

3 MapReduce Implementation

The main concept of the proposed algorithm is parallel computation of counting tables. The proposed algorithm consists of the four steps: Map, Combine, Reduce and Compute Core.

Step 1. Map

INPUT: key : subtable id, $value$: decision subtable $DT_i = (U_i, A \cup \{d\})$
OUTPUT: $< key', value' >$ pair where $key' : (\vec{x}_B, d(x))$ and $value'$ is id of the object x

1: **for** $x \in U_i$ **do**
2: $key' \leftarrow (\vec{x}_A, d(x))$
3: $value' \leftarrow$ id of the object x
4: $emit(< key', value' >)$

```
5:    for a ∈ A do
6:       B ← A − {a}
7:       key′ ← (x⃗_B, d(x))
8:       value′ ← id of the object x
9:       emit(< key′, value′ >)
10:   end for
11: end for
```

Input to function Map are: *key* is a subtable id stored in HDFS, and *value* is decision subtable $DT_i = (U_i, A \cup \{d\})$. First loop in line 1 takes an object x from decision subtable DT_i and emits pair $< key', value' >$, where key' is information vector with respect to set of all condition attributes and decision of the object x, and *value* is id of the this object. Loop in line 5 takes attribute a from set of all condition attributes A. Set B is initialized as set A after removing this attribute. Next step is emitting pair $< key', value' >$, where key' is information vector with respect to set B and decision of the object x, and $value'$ is id of the this object.

Step 2. Combine

INPUT: $< key, value >$ where $key : (\vec{x}_B, d(x))$ and *value* is id of the object x.

OUTPUT: $< key', value' >$ where $key : (\vec{x}_B, d(x))$ and $value'$ is the number of objects belonging to equivalence class $[x]_B$ with equal decision values, from decision subtable DT_i.

```
1: key′ ← key
2: value′ ← 0
3: for each value do
4:    value′ ← value′ + 1
5: end for
6: emit(< key′, value′ >)
```

Input to function Combine are $< key, value >$ pairs where $key : (\vec{x}_B, d(x))$ and *value* is id of the object. This function accepts a key and a set of values associated with this key from the local Map. Function emits pairs $< key', value' >$, where key' is $(\vec{x}_B, d(x))$ and $value'$ is the number of objects associated with this key from decision subtable DT_i.

Step 3. Reduce

INPUT: $< key, value >$ where $key : (\vec{x}_B, d(x))$ and *value* is the number of objects belonging to equivalence class $[x]_B$ with equal decision values from decision subtable DT_i.

OUTPUT: the files containing pairs $< key', value' >$ where $key : (\vec{x}_B, d(x))$ and $value'$ is the number of objects belonging to equivalence class $[x]_B$ with equal decision values from decision table DT.

```
1: key′ ← key
2: value′ ← 0
3: for each value do
```

4: $value' \leftarrow value' + value$
5: **end for**
6: $saveToFile('A - B'.dat, < key', value' >)$

Input to function Reduce are $< key, value >$ pairs where $key : (\vec{x}_B, d(x))$ and $value$ is the number of objects belonging to equivalence class $[x]_B$ with the same decisions from decision subtable DT_i. Function emits pairs $< key', value' >$, where key' is $(\vec{x}_B, d(x))$ and $value'$ is a number of the objects associated with this key from decision table DT. Each pair is saved to file, which name based on index removed attribute from the set of all condition attributes A.

Step 4. Compute Core

INPUT: the files containing the pairs $< key, value >$ where $key : (\vec{x}_B, d(x))$ and $value$ is number of the objects belong to $[x]_B$ with the same decision value.

OUTPUT: core $C \subseteq A$

1: $C \leftarrow \emptyset$
2: $disc(A) \leftarrow 0$
3: **for** $x \in \emptyset.dat$ **do**
4: **for** $y \in \emptyset.dat$ **do**
5: **if** $\vec{x}_B \neq \vec{y}_B$ **and** $d(x) \neq d(y)$ **then**
6: $disc(A) \leftarrow disc(A) + value(x) \cdot value(y)$
7: **end if**
8: **end for**
9: **end for**
10: **for** $a \in A$ **do**
11: $disc(A - \{a\}) \leftarrow 0$
12: **for** $x \in a.dat$ **do**
13: **for** $y \in a.dat$ **do**
14: **if** $\vec{x}_B \neq \vec{y}_B$ **and** $d(x) \neq d(y)$ **then**
15: $disc(A - \{a\}) \leftarrow disc(A - \{a\}) + value(x) \cdot value(y)$
16: **end if**
17: **end for**
18: **end for**
19: **if** $disc(A - \{a\}) < disc(A)$ **then**
20: $C \leftarrow C \cup \{a\}$
21: **end if**
22: **end for**

Input to the function Compute Core is directory contains files, which names based on index of removed attribute from set of all condition attributes, and output is core. In the beginning core C is initialized as empty set and discernibility measure of set of all attributes is set to zero. First two loops in lines 3 and 4 take subsequent lines from file with counting table for all condition attributes to comparison. If these two lines contains information about two different information vectors and decisions, $disc(A)$ is increased by product of two values these lines.

Similarly operations are repeated for each file contains $< key, value >$ pairs. Name of these files based on removed attribute from original set of condition attributes. If computed discernibility measure of set of attributes without this attribute is less than for all attributes, this attribute is added to the core.

4 Experimental Results

The algorithm was running on the Hadoop MapReduce platform [1], where Hadoop MapReduce is a YARN-based system for parallel processing of big datasets. In the experiments, Hadoop 2.5.1 version was used. Cluster of compute nodes contains 19 PC's. All the PC's has four 3.4 GHz cores and 8 GB of memory.

In this paper, we present the results of the conducted experiments using data about children with insulin-dependent diabetes mellitus (type 1). Diabetes mellitus is a chronic disease of the body's metabolism characterized by an inability to produce enough insulin to process carbohydrates, fat, and protein efficiently. Treatment requires injections of insulin. Twelve condition attributes, which include the results of physical and laboratory examinations and one decision attribute (microalbuminuria) describe the database used in our experiments. The data collection so far consists of 107 cases. The database is shown at the end of the paper [5]. A detailed analysis of the above data is in Chap. 6 of the book [6].

This database was used for generating bigger datasets consisting of $0.5 \cdot 10^6$ to $30 \cdot 10^6$ of objects. New datasets were created by randomly multiplying the rows of original dataset. Numerical values were discretized and each attributes value was encoded using two digits.

4.1 Speedup

In speedup tests, the dataset size is constant and the number of nodes grows in each step of experiment. To measure speedup, we used dataset contains $30 \cdot 10^6$ objects. The speedup given by the n-times larger system is measured as [7]:

$$Speedup(n) = \frac{t_n}{t_1}$$

where n is number of the nodes in cluster, t_1 is the computation time on one node, and t_n is the computation time on n nodes.

The ideal parallel system with n nodes provides n times speedup. The linear speedup is difficult to achieve because of the I/O operations data from HDFS and the communications cost between nodes. Table 1 shows the computational time of generating core from dataset containing $30 \cdot 10^6$ objects. The number of nodes varied from 1 to 19. Figure 1 shows, the proposed algorithm has a good performance until the number of nodes is less than 14. Mapper doesn't always work on node where is stored block of file in HDFS. In this case block of file with data is sending from node, where is stored to another, where is processing. The main reason why speedup isn't linear is overhead read and write operations. Generally, if the number of node is bigger, the speed performs better.

Table 1. Speedup experiment results

Number of the nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Time (s)	415	300	191	131	104	89	77	73	68	62	60	54	52	52	54	47	56	46	45

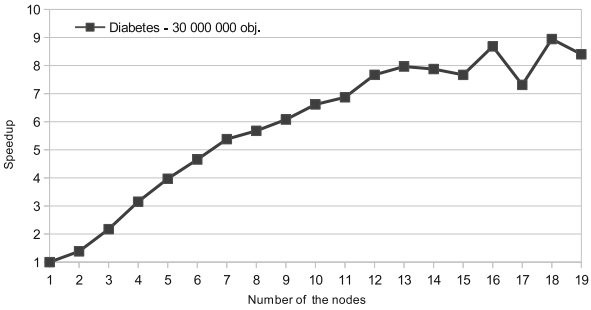

Fig. 1. Speedup

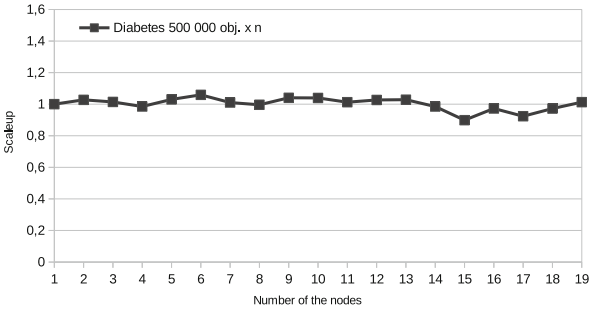

Fig. 2. Scaleup

4.2 Scaleup

Scaleup analysis measures stability system when system and dataset size grow in each step of experiment. The scaleup coefficient is defined as follows [7]:

$$Scaleup(DT, n) = \frac{t_{DT_{1,1}}}{t_{DT_n,n}}$$

where $t_{DT_{1,1}}$ is the computational time for dataset DT on one node, and $t_{DT_n,n}$ is the computational time for n-larger dataset DT on n nodes.

We demonstrate how good the proposed parallel algorithm handles larger datasets when more nodes are available. In scaleup experiments there is linear relationship between number of nodes and dataset size. Core is generated for the dataset consisting of $0.5 \cdot 10^6$ objects on one node and for $9.5 \cdot 10^6$ objects on 19 nodes. Clearly, Fig. 2 shows that the proposed algorithm is scalable.

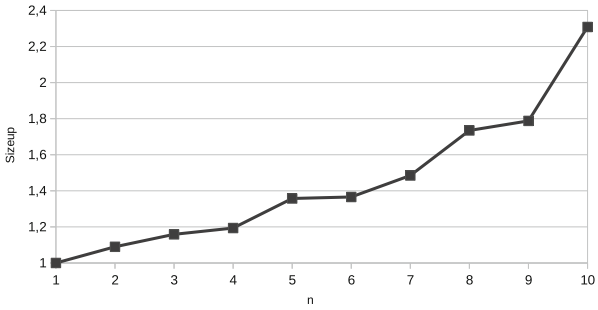

Fig. 3. Sizeup

4.3 Sizeup

In sizeup tests, the number of nodes is constant, and the dataset size grows in each step of experiment. Sizeup measures how much time is needed for calculations when the size of dataset n-larger than the original dataset. Sizeup is defined as follows [7]:

$$Sizeup(DT) = \frac{t_{DT_n}}{t_{DT_1}}$$

where t_{DT_1} is execution time for a given dataset DT, and t_{DT_n} is execution time n-larger dataset than DT. Figure 3 shows the sizeup experiments results on ten nodes. Results shows that the proposed algorithm has a very good sizeup performance. When dataset grows ten times, the computational time grows 2.3 times.

5 Conclusions and Future Research

In this paper, generating core for large datasets based on rough set theory is studied. A parallel attribute reduction algorithm is proposed, which is based on the distributed programming model of MapReduce and the core computation algorithm using discernibility measure of a set of attributes. It is worth noting that a very interesting element of the paper is an usage of a counting table instead of a discernibility matrix. The results of the experiments show that the proposed method is efficient for large data, and it is a useful method for data mining and knowledge discovery for big datasets. Our future research work will focus on optimizing data placement in Hadoop to improve efficiency and on applications of distributed in-memory computing for attribute reduction.

Acknowledgements. The research is supported by the Polish National Science Centre under the grant 2012/07/B/ST6/01504 (Jaroslaw Stepaniuk) and by the scientific grant S/WI/3/2013 (Michal Czolombitko).

The experiments were performed using resources co-financed with the European Union funds as a part of the "Centre for Modern Education of the Bialystok University of Technology" project (Operational Programme Development of Eastern Poland)

References

1. Hadoop MapReduce. http://hadoop.apache.org
2. Lee, B.K., Jeong, E.H.: A design of a patient-customized healthcare system based on the hadoop with text mining (PHSHT) for an efficient disease management and prediction. Int. J. Softw. Eng. Appl. **8**(8), 131–150 (2014)
3. Nguyen, H.S.: Approximate boolean reasoning: foundations and applications in data mining. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets V. LNCS, vol. 4100, pp. 334–506. Springer, Heidelberg (2006)
4. Pawlak, Z., Skowron, A.: Rudiments of rough sets. Inf. Sci. **177**(1), 3–27 (2007)
5. Stepaniuk, J.: Knowledge discovery by application of rough set models. In: Polkowski, L., Tsumoto, S., Lin, T.Y. (eds.) Rough Set Methods and Applications. STUDFUZZ, vol. 56, pp. 137–233. Springer, Heidelberg (2000)
6. Stepaniuk, J.: Rough-Granular Computing in Knowledge Discovery and Data Mining. SCI, vol. 152. Springer, Heidelberg (2008)
7. Xu, X., Jäger, J., Kriegel, H.P.: A fast parallel clustering algorithm for large spatial databases. Data Min. Knowl. Discov. **3**, 263–290 (1999)
8. Yang, Y., Chen, Z., Liang, Z., Wang, G.: Attribute reduction for massive data based on rough set theory and MapReduce. In: Yu, J., Greco, S., Lingras, P., Wang, G., Skowron, A. (eds.) RSKT 2010. LNCS, vol. 6401, pp. 672–678. Springer, Heidelberg (2010)
9. Yang, Y., Chen, Z.: Parallelized computing of attribute core based on rough set theory and MapReduce. In: Li, T., Nguyen, H.S., Wang, G., Grzymala-Busse, J., Janicki, R., Hassanien, A.E., Yu, H. (eds.) RSKT 2012. LNCS, vol. 7414, pp. 155–160. Springer, Heidelberg (2012)
10. Zhang, J., Li, T., Ruan, D., Gao, Z., Zhao, C.: A parallel method for computing rough set approximations. Inf. Sci. **194**, 209–223 (2012)