An Efficient Method for Computing All Reducts

Yongguang Bao  
Aichi Information System  
baoyg@yahoo.com.cn

Xiaoyong Du  
School of Information, Renmin University of China  
duyong@mail.ruc.edu.cn

Mingrong Deng  
School of Management, Zhejiang University, China  
Dengmr@zju.edu.cn

Naohiro Ishii  
Department of Information Networking Eng., Aichi Institute of Technology  
ishii@in.aitech.ac.jp

Keywords: machine learning, rough sets, attribute reduct

Summary

In the process of data mining of decision table using Rough Sets methodology, the main computational effort is associated with the determination of the reducts. Computing all reducts is a combinatorial NP-hard computational problem. Therefore the only way to achieve its faster execution is by providing an algorithm, with a better constant factor, which may solve this problem in reasonable time for real-life data sets. The purpose of this presentation is to propose two new efficient algorithms to compute reducts in information systems. The proposed algorithms are based on the proposition of reduct and the relation between the reduct and discernibility matrix. Experiments have been conducted on some real world domains in execution time. The results show it improves the execution time when compared with the other methods. In real application, we can combine the two proposed algorithms.

1. Introduction

Data mining from databases aims at extraction of implicit knowledge from large databases. Rough Sets theory firstly introduced by Pawlak [Pawlak 82] provides us a new approach to do data analysis in practice [Pawlak 91, Skowron 92, Ziarko 96]. It has been used successfully and widely in machine learning and data mining. An important task in rough set based data analysis is computation of the attribute reduct. Attribute reduction in information system is probably most closely related to feature selection, which has been studied thoroughly in numerous papers (e.g. [Liu 98]).

A reduct of an information system is a minimal subset of the attributes of the system that has the same power as the whole attribute set in discerning objects in the information system. Like the feature selection, attribute reduction can help in the processing of data mining. It selects features relevant to a particular application, removes irrelevant and/or redundant ones, evaluates the relative importance of features, improves the data quality, makes mining algorithms work faster on larger sized data, and enhances the comprehensibility of mined results. In conclusion, attribute reduction is useful in the processing of data mining.

Computing reducts is a computationally complex task. The generation of a minimal reduct has been proved to be NP-hard in [Skowron 92]. As a result, two groups of reduct computing algorithms are used for solving real-life problem: exact algorithms and approximate algorithms. Many researches focus on proposing approximate algorithm (e.g. [Pawlak 91, Pawlak 94, Hu 97, Wroblewski 95, Ziarko 95, Jenssen 97]). However, in many real-life applications, there is necessary and possible to compute all reducts for an information system. For example, we may need to compute the reduct with minimal cardinality or the reduct with the minimal rules or multiple knowledge or population of reducts etc. In these cases, an exact algorithm to generate all reducts is necessary. Moreover, as can be seen in [Hu 97], DBMaxi (which generates all the minimal decision rules based on all
reducts) represented the upper bound of the classification accuracy when compared with classification methods based on single reduct or multi reducts. So, it is valuable to study the computation of all reducts. Since computing all reducts is an expensive operation, especially changing on the decision table will cause re-computation of the reducts, incremental computing method is also required in dynamic information systems.

This paper is concerned with the exact algorithms for reducts computation in information systems and their practical evaluation. This paper proposes a new efficient non-incremental algorithm and a new efficient incremental algorithm to improve the execution time when compared with the other methods.

The remains of this paper are organized as follows: Section 2 introduces the basic definitions and notions that are relevant to reduct in rough sets. Section 3 describes the computation of reducts and the original algorithm, and presents our new algorithm. Section 4 reports on the results of the experiment. Finally, we conclude the paper in Section 5.

2. Information Systems and Rough Sets

In this section we present some basic notions related to information system and Rough Sets.

An information system is pair of $S = < U, Q >$, where $U$ - a nonempty, finite set called the universe and $Q$ - a nonempty finite set of attributes i.e. $q: U \rightarrow V_q$ for $q \in Q$, where $V_q$ is called the value set of $q$.

With every subset of attributes $B \subseteq Q$, we associate a binary relation $IND(B)$, called $B$-indiscernibility relation, and defined as follows:

$IND(B) = \{(x, y) \in U \times U : \forall q \in B, q(x) = q(y)\}$.

Obviously $IND(B)$ is an equivalence relation. The object $x, y$ satisfying the relation $IND(B)$, are indiscernible with respect to subset B of attributes. In other words, we cannot distinguish $x$ from $y$ in terms of attributes in $B$. By $[x]_B$ we denoted the equivalence class of $IND(B)$ containing $x$, i.e. the set $\{y \in U : xIND(B)y\}$.

Core and reduct are two fundamental concepts of rough sets. A reduct is the essential part of an information system that can discern all objects discernible by the original information system. The core is the common parts of all reducts.

An attribute $q \in B \subseteq Q$ is dispensable in $B$, if $IND(B) = IND(B - \{q\})$, otherwise $q$ is indispensable in $B$. If $q$ is an indispensable feature, deleting it from $S$ will cause $S$ to be inconsistent. Otherwise, $q$ can be deleted from $S$.

The set of all indispensable attributes in $Q$ is called the core of $S$, and is denoted by $CORE(S)$. We say that $B \subseteq Q$ is independent in $S$ if every attribute of $B$ is indispensable in $S$. Otherwise the set $B$ is said to be dependent in $S$.

A set $B \subseteq Q$ is called a reduct in $S$ if $B$ is independent in $S$ and $IND(B) = IND(Q)$. The set of all reducts in $S$ is denoted by $RED(S)$.

It follows from the definition that $B \in RED(S)$ iff $B$ is minimal (with respect to set theoretical inclusion) subset of $Q$ such $IND(B) \subseteq IND(Q)$. And $CORE(S) = \cap RED(S)$.

3. Computation of Reducts

This section is concerned with the exact algorithms for reducts computation. The exact algorithms are capable of generating all reducts from information systems. Their main disadvantage is the computational complexity that may grow non-polynomially with the number of features in data set. However, only some naive approaches perform the search by scanning through all possible elements of the search space. Advance algorithms seek to utilize various properties of reduct definition, the decision table or the problem formulation to produce the results more efficiently. Despite the theoretical complexity, these algorithms may perform well for some real-life data sets.

Now, we discuss the non-incremental exact algorithms firstly. A. Skowron and C. Rauszer [Skowron 92] proposed an efficient exact algorithm RGA. It is based on the notion of the discernibility matrix, which is defined as follows, and takes advantage of some properties of Boolean function. Let $S = < U, Q >$ be an information system and let us assume that $Q = \{q_1, \ldots, q_m\}, U = \{x_1, \ldots, x_n\}$. By $M(S)$ we denote an $n \times n$ matrix ($c_{ij}$), called the discernibility matrix of $S$, such that $c_{ij} = \{q \in Q : q(x_i) \neq q(x_j)\}$ for $i, j = 1, 2, \ldots, n$.

Since $M(S)$ is symmetric and $c_{ii} = \emptyset$ for $i = 1, 2, \ldots, n$, we represent $M(S)$ only by elements in the lower triangle of $M(S)$, i.e. the $c_{ij}$ is with $1 \leq j < i \leq n$.

The discernibility matrix is computed for pairs of objects and stores the ‘differences’ between all possible pairs of objects that must remain discernible. The main idea of RGA is that if a set of attributes is to
satisfy the consistency criterion, (i.e. be sufficient to discern all the required objects), it must have a non-empty intersection with non-empty elements of the discernibility matrix. As a result, the set of all reducts may be considered as a family of minimal sets having non-empty intersection with non-empty elements of the discernibility matrix. A. Skowron and C. Rauszer [Skowron 92] showed that this problem has a counterpart in the theory of Boolean functions—it corresponds to the problem of transforming a Boolean function from its disjunctive form (DNF) to the non-redundant conjunctive form (CNF). The main technique in the theory of Boolean functions is searching for the prime implicants of the function, which is equivalent to finding independent subsets of attributes.

RGA consists of two main phases: the first phase creates and transforms the discernibility matrix, the second one generate reducts using the created matrix. R. Susmaga [Susmaga 98] gave the modification of RGA to improve executing time, and called it MRGA. It is based on the observation that the computing time of Phase II (generating all the minimal sets) may be improved if the created absorbed matrix were sorted in the ascending order of the cardinality of its elements before Phase II is started. The experiment results in [Susmaga 98] showed MRGA improved the execution time better than RGA, and it was the best non-incremental exact algorithm. The algorithm MRGA is presented in Fig. 1.

In Fig. 1, Min performs an operation that is analogous to checking for prime implications of Boolean function. The returned value is true if the argument R does not contain redundant attributes.

\[
\text{Min}(R, \text{ADL}, i) = \begin{cases} 
\text{True} : & \text{if for each } a \in R \text{ there exists } C_j \in \text{ADL}_{1\ldots i}, \text{such that } R \cap C_j = a \\
\text{False} : & \text{otherwise}
\end{cases}
\]

\[
\text{ADL}_{1\ldots i} \text{ is a list consisting of } i \text{ initial element of } \text{ADL}: \text{ADL}_{1\ldots i} = C_1, C_2, \ldots, C_i
\]

In real application, data gathering is very often a dynamic process. This means that the data set is not always available on the whole from the beginning. In fact, in many cases the set is slowly increasing—it grows larger every time when new data arrives. The problem specific to the definition of reducts is that after increasing the number of objects in decision table the set of reducts computed using the exact non-incremental algorithm for the previous version of the table is no longer valid, because that there is no guarantee that a given reduct computed for the previous version of the decision table remains a reduct in the increased table, and there is no guarantee that the set of reducts computed for the previous version of the decision table contains all the reducts of the increased table.

As a consequence, if a regular non-incremental exact algorithm is used for computing reducts and the size of the decision increases with time, the algorithm must be invoked anew after every change of the information system, which may turn out to be extremely inefficient. The worst case occurs when the size of the table increases by one object at a time. Such a situation may cause an intolerable computational load and eliminate the possibility of storing an up-to-date set of reducts for the current state of the data table. A solution to this problem is an incremental algorithm, that is algorithm which instead of generating a new set of reducts in each run, tries to update the existing set of reducts according to the increased contents of the information systems.

E. Orlowska and W. Orlowski [Orlowska 92] proposed an incremental algorithm RMA for reduct maintenance. But it is not good enough to use, R. Susmaga [Susmaga 98] modified RMA by introducing the idea of the discernibility matrix into RMA to improve the execution time, and called it MRA-DL. And based

---

**Fig. 1** The Modified Reduct Generation Algorithm (MRGA)
on the idea of incremental modification of discernibility matrix by W. Ziarko and N. Shan [Ziarko 94]. R. Susmaga [Susmaga 98] gave the incremental version IRGA of the algorithm RGA, that is IRGA. It employs the same mechanisms for generating the minimal subset of attributes. The experiment results on some real-life data sets showed that IRGA was the best incremental algorithm for computing reducts. The algorithm IRGA is presented in Fig. 2.

![Image](51x222 to 282x623)

**Input**: A number of objects being added successively to the set \( U \);

The objects are described by values of attributes from the set \( Q \).

**Output**: The set \( K \) of all reducts for the current contents of the set \( Q \).

The set of objects is updated after every arrival of a new object.

**Initial Step**

Start with a one-element set \( U \) (the first object).

All single attribute sets are reducts: \( K = \{ \{ q \} \text{ for each } q \in Q \} \)

The absorbed discernibility list is empty: \( ADL = \emptyset \).

For every object \( x (x \notin U) \):

**PHASE I**

**Step 1**

Establish and absorb the list of current differences:

\[ CDL = \{ C_i \in CDL : C_i \neq \emptyset \text{ and for no } C \in ADL \cap CDL : C \subseteq C_i \} \]

The resulting absorbed list of current differences contains elements \( C_1, C_2, \ldots, C_d \).

**Step 2**

Update the absorbed discernibility list:

\[ ADL = \{ C_i \in ADL \cap CDL : C_i \neq \emptyset \text{ and for no } C \in CDL : C \subseteq C_i \} \]

**PHASE II**

**Step 1**

\( R_0 := K \)

**Step 2**

For every \( i = 1, 2, \ldots, d \) compute:

\[ S_i := \{ R \in R_{i-1} : R \cap C_i \neq \emptyset \} \]

\[ T_i = \bigcup_{a \in C_i} R \in R_{i-1} \setminus R \cap C_i \neq \emptyset \]

\[ MIN_i := \{ R \in T_i : \text{Min}(R, \overline{ADL}, [ADL]) = \text{True} \} \]

\( R_i := S_i \cup MIN_i \)

**Step 3**

\( K := R_d \)

\( U := U \setminus \{ x \} \)

The final result is the set \( K \)

![Fig. 2 Incremental Reduct Generation Algorithm (IRGA)](51x222 to 282x623)

In the following, a new non-incremental algorithm and an incremental algorithm will be proposed. In the algorithm MRGA, the discernibility matrix serves two purposes. Firstly, it is used to check the consistency criterion and, secondly, it is used directly in the reduction generation phase, in which the reducts are computed as the minimal sets having non-empty intersections with the elements of the discernibility matrix. By analysis, we can see that the Min operation in MRGA would cost more time when \( |R| \) is big or \( |ADL| \) is big, especially in the case of return False. In algorithm MRGA, from the proposition of reduct and discernibility matrix, we can see that \( R_i \subseteq S_i \cup T_i \), and the each element of \( S_i \cup T_i \) must be a reduct or a subset of one reduct. Following this, it is only needed to check the minimal property in \( S_i \cup T_i \).

On the other hand, in the point of Boolean function, absorption low can be used in the process of transformation to its conjunctive form. Based the above discussion we have the new exact algorithms for reducts computation in the Fig. 3. In our algorithm, we will not use the discernibility matrix in the reduce generation phase to check the minimal sets. We use an effective checking process based on the reduct proposition, to decrease the checking cost, then to improve the execution time.

**Input**: A set of objects \( U(|U| = N) \).

The objects are described by values of attributes from the set \( Q \).

**Output**: The set \( K \) of all reducts for the set \( U \).

**PHASE I**

**Step 1**

Create the absorbed discernibility list \( ADL \) for \( i = 1, \ldots, N \).

Eliminating empty and non-minimal elements:

\[ ADL := \{ C_{ij} : C_{ij} \neq \emptyset \text{ and for no } C \in ADL : C \subseteq C_{ij} \} \]

where \( c_{ij} = [q \in Q : q(x_j) \neq q(x_i)] . i, j = 1, 2, \ldots, |U| \)

The resulting absorbed list of current differences contains elements \( C_1, C_2, \ldots, C_d \), where \( d \leq \lfloor N(N-1)/2 \rfloor \).

**Step 2**

Sort the \( ADL \) in the ascending order of the cardinality of its elements.

**PHASE II**

**Step 1**

\( R_0 := \emptyset \)

**Step 2**

For every \( i = 1, 2, \ldots, d \) compute:

\[ S_i := \{ R \in R_{i-1} : R \cap C_i \neq \emptyset \} \]

\[ T_i = \bigcup_{a \in C_i} R \in R_{i-1} \setminus R \cap C_i \neq \emptyset \]

\[ MIN_i := \{ R \in R_i : \text{Min}(R, ADL, [ADL]) = \text{True} \} \]

\( R_i := S_i \cup MIN_i \)

The final result is the set \( K \)

![Fig. 3 Reduct Computation Algorithm (RCA)](51x222 to 282x623)

As same as the algorithm IRGA, we can give the incremental version of algorithm RCA, the Incremental Reduct Computation Algorithm (IRCA)[Bao 01]. It only uses \( MIN_i := \{ R \in T_i : \text{for no } S \in S_i \text{ s.t. } S \subseteq R \} \) instead of \( MIN_i := \{ R \in T_i : \text{Min}(R, ADL, [ADL]) = \text{True} \} \) to check minimal sets in algorithm IRGA. Because of the repetition, we omit it in this presentation.

For the sake of illustration of the effectiveness of
our algorithm, we do some comparison about IRCA and IRGA in the following, let us consider the simple example in the following.

**Example**

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

When using the Algorithm IRGA, the initial step is simple, we have $ADL = \emptyset$ and $K = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. For the third object, we have $CDL = ADL = \{\{abcde\}\}$ and $R_0 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. Then we get $S_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$ and $T_1 = \emptyset$, so we get $K = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. The third object can be processed.

For the third object, we have $CDL = ADL = \{\{ab\}, \{cde\}\}$ and $R_0 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. Then we get $S_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$ and $T_1 = \emptyset$. Then go to $i = 2$, have $S_i = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$ and after being checked, have $R_i = S_i = \{\{a\}, \{b\}\}$. Then go to $i = 2$, have $S_i = \{\{a\}, \{b\}\}$ and then after being checked, have $R_i = \{\{a\}, \{b\}\}$. So get $K = \{\{a\}, \{b\}\}$. The fourth object can be processed.

For the fourth object, we have $CDL = ADL = \{\{ab\}, \{be\}\}$ and $R_0 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. Then we get $S_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$ and $T_1 = \emptyset$. Then go to $i = 2$, have $S_i = \{\{a\}, \{b\}\}$ and after being checked, have $R_i = \{\{a\}, \{b\}\}$. So get $K = \{\{a\}, \{b\}\}$. The fourth object can be processed.

When considering the upper example using IRCA.

For the third object, we have $CDL = ADL = \{\{abcde\}\}$ and $R_0 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. Then we get $S_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$ and $T_1 = \emptyset$. Then go to $i = 2$, have $S_i = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$ and after being checked, have $R_i = \{\{a\}, \{b\}\}$. Then go to $i = 2$, have $S_i = \{\{a\}, \{b\}\}$ and then after being checked, have $R_i = \{\{a\}, \{b\}\}$. So get $K = \{\{a\}, \{b\}\}$. The fourth object can be processed.

Compare the processing with algorithm IRGA, obviously IRCA decreases the compute content and times and get some reduces. For example, to the third object, as you see IRCA generates candidates:

<table>
<thead>
<tr>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a}$, ${b}$, ${c}$, ${d}$, ${e}$;</td>
</tr>
<tr>
<td>${a}, {b}, {c}, {d}, {e}$;</td>
</tr>
<tr>
<td>${a}, {b}$;</td>
</tr>
<tr>
<td>${a}, {b}, {c}, {d}$;</td>
</tr>
<tr>
<td>${a}, {b}, {c}, {d}, {e}$;</td>
</tr>
<tr>
<td>${a}, {b}$;</td>
</tr>
</tbody>
</table>

But algorithm IRGA generates candidates:

<table>
<thead>
<tr>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a}, {b}, {c}, {d}, {e}$;</td>
</tr>
<tr>
<td>${a}, {b}, {c}, {d}, {e}$;</td>
</tr>
<tr>
<td>${a}, {b}, {c}, {d}, {e}$;</td>
</tr>
</tbody>
</table>

Obviously the checking process of IRGA is simpler than IRGA.

### 4. Experiment Results

For evaluating the effectiveness of our new algorithm, IRCA/IRGA algorithm was compared with MRGA/IRGA on some data sets, because MRGA/IRGA is the best non-incremental/incremental algorithm till now [Susmaga 98]. The computation may be performed in either incremental or non-incremental way. Depending on this, two kinds of computing times are being observed: absolute computing times and relative (per object) computing times.
Absolute computing times required for computing all exact reducts for a data set. This mainly concerns the non-incremental algorithms, but incremental algorithms may also be evaluated in that way to test their ability to work as regular algorithms for reduct computation. In that case only the final set of reducts is considered to be the result of running the algorithm. Relative computing times required for updating the set of reducts after arrival of each object. This mainly concerns the incremental algorithms, but non-incremental algorithms may be also evaluated in that way to test their ability to work as incremental algorithms, in which case they have to be simply invoked to run anew after arrival of every new object. It must be stressed that in this case the algorithms produce a very voluminous output (the sets of reducts being updated after arrival of each new object).

The data sets used in this experiment are downloaded from the UCI Machine Learning Repository [Merz 98]. For the missing value we just take it as a new value. The basic characteristic of the data sets is showed in the Table 2. It contains Attr., Rec., min, max, Reducts, which are denoted the cardinality of condition attributes, the number of the objects, the minimal cardinality of all reducts, the maximal cardinality of all reducts, and the number of the all reducts respectively.

The computing platform was personal computer (Windows 98,64MB, 233MHz). If a given algorithm failed to produce results within a specified time period (namely 200000 seconds), the program was terminated and the result (the actual computing time) remained unknown. Such a situation is marked by the entries "####" in the result table and the lack of bars in the corresponding charts. The experiments were performed as follow. All the algorithms being tested were invoked for the same collection of data sets and their actual computing times were observed. The tests included the two non-incremental algorithms MRGA, RCA and the two incremental algorithms IRGA and IRCA. In the incremental tests the non-incremental algorithms were invoked in their Multi-Run versions while in the non-incremental tests only the final results of the incremental algorithms were taken into account.

### 4.1 Absolute Computing Times

The first experiment was of non-incremental character. In this case the result is regarded as the absolute computing time of the algorithm. And because of the sameness of RCA and MRGA (IRCA and IRGA) in Phase I, we give the computing times of Phase I and total times respectively. The absolute computing times for the algorithms MRGA, RCA, IRGA and IRCA are presented in Table 3. The columns "Num", "Min", "Max" and "times(s)" of ADL show the number of the absorbed discernibility list (ADL), the minimal cardinality of ADL, the maximal cardinality of ADL, and computing times of creating ADL respectively. The “RCA”, “MRGA”, “IRCA” and “IRGA” columns show the total computing times of algorithms RCA, MRGA, IRCA and IRGA respectively.

As it can be seen, the absolute computing times for most of the data sets are considerable. And obviously, the non-incremental algorithms are better than the incremental algorithms. RCA is modification of MRGA, and although the change only concerns the processing of the checking minimal sets in Phase II, a steady improvement over the original MRGA may be observed. MRGA failed in Lsd200 and RCA does not. If we view failed cases as the worst results, then IRGA produced the worst results in all cases, and RCA is the best in all cases. Especially, it improves MRGA more when the number of all reducts is big. IRCA is evidently better than IRGA in the sense of incremental character. What may be concluded from the above table is that the RCA, developed in this paper, turns out to be the best approach in absolute computing times.
4.2 Relative Computing Times

The second experiment was of incremental character. In this case the result regarded as the relative computing time of the algorithm. The relative (per object) computing times for the algorithms MRGA, RCA, IRGA and IRCA are presented in Table 4.

From the experimental results in table 4, we can see that the IRCA is certainly the best. It did not fail in all cases and others algorithms failed in two data sets. If we view failed cases as the worst results, then MRGA produced the worst results in all cases in the sense of incremental character, and IRCA is the best in all cases. A conclusion concerning this part is that the best incremental algorithm is evidently the IRCA. The relative computing times produced by the IRCA are promising even for the most difficult data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>MRGA</th>
<th>RCA</th>
<th>IRGA</th>
<th>IRCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-Tac-Toe</td>
<td>2.01E-01</td>
<td>2.01E-01</td>
<td>8.87E-04</td>
<td>8.56E-04</td>
</tr>
<tr>
<td>Voting</td>
<td>1.45E-01</td>
<td>1.15E-01</td>
<td>2.46E-03</td>
<td>1.10E-03</td>
</tr>
<tr>
<td>Mushroom</td>
<td>3.74E+02</td>
<td>3.73E+01</td>
<td>1.61E-02</td>
<td>1.58E-02</td>
</tr>
<tr>
<td>Chess2</td>
<td>8.15E+00</td>
<td>8.15E+00</td>
<td>8.30E-03</td>
<td>8.22E-03</td>
</tr>
<tr>
<td>Connect</td>
<td>3.87E+01</td>
<td>3.84E+01</td>
<td>2.13E-02</td>
<td>2.11E-02</td>
</tr>
<tr>
<td>Zoo</td>
<td>2.67E+01</td>
<td>1.33E+01</td>
<td>4.06E-02</td>
<td>4.00E-02</td>
</tr>
<tr>
<td>Sed047</td>
<td>9.83E-01</td>
<td>2.92E-00</td>
<td>1.75E-01</td>
<td>4.40E-02</td>
</tr>
<tr>
<td>Lsd200</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Lsd307</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>Letter260</td>
<td>2.56E+01</td>
<td>1.42E+00</td>
<td>5.77E-01</td>
<td>3.23E-02</td>
</tr>
</tbody>
</table>

5. Conclusions

Computation of reduction of a decision table is of primary importance in Rough Sets theory. Because computing all reducts is NP-hard (in respect of the number of attributes), one cannot expect any algorithm can improve asymptotic computational complexity in worst case. The only way to solve the problem more efficiently is to propose an algorithm that could be better by a polynomial factor. In this paper, we propose a new effective way to compute all reducts. The presented non-incremental/incremental algorithm is based on the proposition of reduce and the relation between reduce and discernibility matrix. It gives some improvement over others (in the real execution time). Simulation results show that RCA and IRCA are efficient algorithms for computing the all reducts. And in the practice, we can combine the algorithm RCA with the algorithm IRCA. It can also be successfully used to find the all relative reducts in decision table.

Acknowledgments

The authors would like to express appreciation to the reviewers for their helpful comments and valuable suggestions towards the improvement of this paper.

References


Susmaga 98] R. Susmaga, “Experiments in Incremental computation of Reducts”, in Methodology and applications, (Studies in fuzziness and soft computing; v.18. Rough sets
An Efficient Method for Computing All Reducts


Received October 18, 2002.

---Author’s Profile---

Bao, Yongguang
Received the M.S. degree in mathematics from Zhejiang University, China in 1993. Since 1997, he has been a lecturer of the Institute of Decision-making and Optimization in Zhejiang University. He received the Doctor of Engineer in computer science from Nagoya Institute of Technology in 2003. His current research interests include algorithm design and analysis, data mining, rough set theory and artificial intelligence.

Du, Xiaoyong
Received the B.S. degree in computational mathematics from Hangzhou University, Zhejiang, China in 1983, and the M.E. degree in information and computer science from Renmin University of China, Beijing, in 1988, and the Doctor of Engineer in computer science from Nagoya Institute of Technology in 1997. Since 1999, he has been a professor of the Institute of Data and Knowledge Engineering in Renmin University of China. His current research interests include databases, data warehouse and data mining, algorithm design and analysis, intelligent information retrieval and artificial intelligence. He is a member of IEEE CS, CCF and IPSJ.

Deng, Mingrong
Received a B.Sc. degree in Applied Mathematics from Huazhong University of Science and Technology, a M.Sc. degree in Operations Research from Dalian University of Technology, and a Ph.D. degree in Computer Science from Zhejiang University. Since 1999, he has been an associate professor in School of Management at Zhejiang University, China. His current research interests include quantitative analysis, logistics, supply chain management and data mining.

Ishii, Naohiro (Member)
Received the B.E. and M.E. degrees and the Doctor of Engineering degree in Electrical and Communication Engineering from Tohoku University, Sendai, in 1963, 1965 and 1968, respectively. From 1968 to 1974 he was at school of Medicine in Tohoku University, where he worked on information processing in central nervous system. Since 1975 he has been with the Nagoya Institute of Technology where he is a professor in the Department of Intelligence and Computer Science. His current research interests include databases, software engineering, algorithm design and analysis, nonlinear analysis of neural network and artificial intelligence. Since 2003, he has been a professor of Department of Information Networking Eng, Aichi Institute of Technology. He is a member of IPSJ, IEICE, ACM and IEEE.