Quick Attribute Reduction Algorithm Based on Consistent Degree

Biqing Wang

College of Mathematics and Computer, Tongling University, Tongling, Anhui 244000, China

Abstract — As is well known, attribute reduction is one of the basic concepts in rough set theory. In this paper, a new algorithm for computing equivalence classes based on segment quick sort is provided and its time complexity is cut down to O(|C||U|) compared with traditional algorithms. Then a method for computing object number of positive region without computing positive region is given. On this basis, the concept of consistent degree is introduced and its properties are studied. Furthermore, definitions of core attributes and attribute importance are proposed. The algorithm for computing core attributes and attribute importance are designed. Finally, a quick attribute reduction algorithm which uses core attributes as initial reduction and attribute importance as heuristic information is presented. In the worst case, its time complexity is O(|C|^2|U|). Example analyses and experimental results show that the quick attribute reduction algorithm of this paper is feasible and efficient.

Keywords - Rough set; decision table; attribute reduction; positive region; consistent degree

I. INTRODUCTION

Rough set theory [1] was proposed by Pawlak in 1982. It is a mathematical tool which processes fuzzy and uncertain knowledge. It can find out hidden information from massive data and has been successfully applied in the fields of data mining, machine learning, artificial intelligence, pattern recognition and so on [2-3]. At present, researches on rough set algorithm include how to compute upper approximation, lower approximation, equivalence class, positive region, reduction, core, etc [4-6]. Because decision table is the main object of study, researches on rough set algorithms focus on attribute reduction of decision table.

Attribute reduction is defined as a process of deleting redundant attributes from larger set of condition attributes. Obtaining a minimal reduction or all the reductions is a NP-hard problem [7]. This has been proven. Generally speaking, we do not need to get a minimal reduction or all the reduction. Sometimes we just need an reduction available. To solve this problem, heuristic algorithms are usually used. In recent years, many different heuristic reduction algorithms for finding reducts are proposed, such as discernibility matrix strategy, positive region, conjunctive term in the minimal disjunctive normal form and so on [2-3]. At present, researches on rough set algorithms focus on attribute reduction of decision table.

Discernibility matrix is a effective tool for finding out reducts, which was introduced by Skowron [8]. An element m_{i,j} of the matrix is the set of all attributes on which the corresponding two objects x_i and x_j have different values. First, discernibility matrix of data set is computed. Then, the minimal disjunctive normal form of all the element in discernibility matrix is computed. Thus, every conjunctive term in the minimal disjunctive normal form corresponds to a reduction and all the reductions can be obtained. The merit of discernibility matrix strategy is intuitively clear, easy to be understood and all the reducts can be conveniently obtained. But, it has some shortcomings, as there are a lot of repeated elements in discernibility matrix and it greatly decreases computational efficiency of reduction algorithm. Now, many researchers study attribute reduction algorithms based on discernibility matrix. Hu and his colleagues [9] introduced a method and its time complexity was O(|C|^2|U|). Because the inconsistent decision tables are not taken into account, Hu’s algorithm can not assure to obtain correct reductions for inconsistent decision tables. Liu and his colleagues [10] put forward a algorithm based on improved discernibility matrix. It can process inconsistent decision tables well and its time complexity was the same as Hu’s algorithm. Cai et al. [11] further improved discernibility matrix and offered an algorithm whose time complexity was O(|C|^2|U|). Xu and his colleagues [14] proposed an algorithm for computing equivalence class partition based on quick sort and its time complexity was O(|C||U|). Ye [12] made an improvement to Jelonek’s algorithm and time complexity was O(|C|^2|U|) in the worst case. Liu et al. [13] put forward an algorithm for computing equivalence class partition based on quick sort and its time complexity was O(|C|^2|U|). On this basis, time complexity of reduction algorithm was O(|C|^2|U|) and its time complexity was cut down to O(|C||U|). Time complexity of reduction algorithm was O(|C|^2|U|) accordingly. But the algorithm was incomplete in some cases.

DOI 10.5013/IJSSST.a.17.37.25 25.1 ISSN: 1473-804x online, 1473-8031 print
All these methods mentioned above have heavy computing load during process of big data sets and some of them are incomplete algorithms. With the above analysis, in this paper, a complete algorithm of quick attribute reduction based on consistent degree is proposed. As equivalence class partition is the basic step in this algorithm, an algorithm for computing equivalence class based on segment quick sort is provided first and its time complexity is cut down to $O(C|U|)$. Then, a method for computing object number of positive region without computing positive region is given. On this basis, the concept of consistent degree and its properties are studied. Furthermore, definitions and algorithms of core attributes and attribute importance are proposed. Finally, a quick and complete attribute reduction algorithm which uses core attributes as initial reduction and attribute importance as heuristic information is presented. In the worst case, its time complexity is $O(C^2|U|)$. Example analyses and experimental results show that the attribute reduction algorithm of this paper is feasible and efficient.

The paper is organized in the following manner. Section II introduces the basic concept of rough set. Section III presents an algorithm for computing equivalence classes based on segment quick sort. In Section IV, definition and algorithm of consistent degree are given. In Section V, an efficient algorithm for computing core is put forward. In Section VI, definition and algorithm of attribute importance are proposed. In Section VII, a quick attribute reduction algorithm is presented. Section VIII analyzes some experimental results. Section IX provides some brief conclusions.

II. PRELIMINARIES

In this section, we review some basic concepts in rough set [1], [16] to be used in this paper.

**Definition 1.** An information system is defined as $S=(U,A,V,f)$, where $U$ is the set of objects; $A$ is the set of attributes; $V=\bigcup_{a\in A}V_a$, where $V_a$ is the set of values of attribute $a$; $f:U\times A\rightarrow V$ is an information function, which determines values of attribute of every object $u$, namely, $f(u,a)\in V_a$ for every $u\in U$ and $a\in A$.

If set of attributes can be divided into condition attributes set $C$ and decision attributes set $D$, namely, $C\cup D=A$, $C\cap D=\emptyset$, the information system is called decision system or decision table, where $D$ has only one attribute.

**Definition 2.** In decision table $S$, if $f(u,C)=f(u,C)$ and $f(u,D)\neq f(u,D)$ for $u,R\subseteq U$ and $i\neq j$, $S$ is called an inconsistent decision table. Otherwise $S$ is called a consistent decision table.

**Definition 3.** In decision table $S$, for $\forall R\subseteq A$, $\forall X\subseteq U$, $U/R=\{R_1, R_2, ..., R_l\}$, lower approximation of $X$ with respect to $R$ is defined as $\overline{X}_R=\bigcup\{R_i|R_i\subseteq U/R, R_i\cap X\neq \emptyset\}$.

**Definition 4.** In decision table $S$, for $P\subseteq A$, an indiscernibility relation $IND(P)$ is defined as $IND(P)\equiv \{(x, y)|x\in U\times U|\forall a\in P, f(x,a)=f(y,a)\}$.

$IND(P)$ is an equivalence relation. Equivalence class of $x$ for set of attributes $P$ is $[x]_{IND(P)}=\{y|x\in U, y\in IND(P)x\}$.

Relation $IND(P)$ determines a division, which is expressed by $U/IND(P)$. For convenience, $P$ is used to substitute $IND(P)$. So, $U/IND(P)$ can be denoted by $U/P$.

**Definition 5.** In decision table $S$, for $\forall P,Q\subseteq A$, positive region of $Q$ with respect to $P$ is defined as following:

$$POS_Q(P)=\bigcup_{X\subseteq U/Q}PX$$

where $P$ is lower approximation of $X$ with respect to $P$.

**Definition 6.** In decision table $S$, for $a\in C$, $a$ is unnecessary if $POS_C(D)=POS_C-{a}(D)$. Otherwise $a$ is necessary. The set of all the necessary attributes is called the core of $C$ with respect to $D$ and defined as $CORE(C)$.

**Definition 7.** In decision table $S$, for $R\subseteq C$, $R$ is called a reduction of $C$ if $POS_R(Q)=POS_C(D)$ and all the attributes in $R$ are necessary.

**Definition 8.** In decision table $S$, for $\forall P,Q\subseteq A$, let $U/P=\{P_1, P_2, ..., P_l\}$ and $U/Q=\{Q_1, Q_2, ..., Q_S\}$. If $\forall P_i\subseteq U/P$, there exists $Q_j\subseteq U/Q$ which makes $P_i\subseteq Q_j$, $U/P$ is called a refinement of $U/Q$, denoted by $U/P\leq U/Q$. If $U/P\leq U/Q$ and $U/Q\leq U/P$, $U/P=U/Q$.

III. NEW ALGORITHM FOR COMPUTING U/C

In quick attribute reduction algorithm based on consistent degree, equivalence class partition is an important step. It lays a foundation for computing consistent degree, core, attribute importance and directly affect time complexity of reduction algorithm. Time complexity of traditional algorithms for computing $U/C$ was $O(|C||U|)$.

Reference [13] put forward an algorithm for computing $U/C$ based on quick sort and its time complexity was cut down to $O(|C||U|\log|U|)$. Reference [14] proposed an algorithm for computing equivalence class partition based on radix sort and its time complexity was cut down to $O(|C||U|\log|U|)$.

After a thorough study on previous literatures, an algorithm for computing $U/C$ based on segment quick sort is provided in this paper and its time complexity is also $O(|C||U|)$.

Let us review the quick sort algorithm firstly. It arranges data anew to divide awaiting processing sequence into two subsequences and make all the data of the first subsequence are less than all the data of the second subsequence. Then, every subsequence is sorted recursively. Thus, the whole sequence can be sorted. In general case, a datum need $\log N$ times move to get its final position. So, time complexity for quick sort is $O(N\log N)$. This paper generalizes the quick sort to segment quick sort, which is used for computing $U/C$ and has higher efficiency.
Based on the concept of quick sort, an outline of the method for computing U/C is given. Sorting decision table by attribute subset C means sorting decision table by every attribute in turn. For every attribute, we find out its minimum and maximum firstly. Then, approximate location of each datum in ordered sequence is computed according to the proportion relationship among the datum, minimum and maximum. Next, size of each subsequence can be counted via datum’s location. Hence, we may arrange data anew to divide awaiting processing sequence into some subsequences and make data of the anterior subsequence are less than data of the posterior subsequence. Afterwards, subsequence whose size is greater than 1 is sorted by quick sort. Thus, the whole sequence can be sorted. Last, we sort decision table by other attributes in turn and analyse sorted decision table to divide equivalence classes.

Let $S=(U, A, V, f)$ be an awaiting processing decision table. A row in decision table is a record and denoted by $R_j(j=1, 2, ..., n)$. $s'$ [$n$] is a array of structure which is used to store sorted decision table. $g$ is the number of subsequences and its value may be taken flexibly. In general case, $g$ can be $\lceil n/2 \rceil$. loc is the segment number of a record in $s'$ [$n$] after segmentation. $f$ [$g$] is a array which stores sizes of every subsequence. $h$ [$n$] is a array which stores “head information” of the subsequences whose size are greater than 1; $t$ [$m$] stores subscript of the first element of the $(m+1)$-th subsequence whose size is greater than 1 in $s'$ [$n$]. $t$ [$n$] is a array which stores “tail information” of the subsequences whose size is greater than 1; $t$ [$m$] stores subscript of the last element of the $(m+1)$-th subsequence whose size is greater than 1 in $s'$ [$n$]. So, this algorithm may be described as follows.

Algorithm 1: Algorithm for computing U/C

Input: $S=(U, C \cup D, V, f)$, $U= \{u_1, u_2, ..., u_n\}$, $C= \{a_1, a_2, ..., a_k\}$.

Output: U/C.

Step1. Sort S by $a_i(i=1,2,\ldots,k)$ in turn:

```plaintext
for(i=1;i<=k;i++)
if(f[0]≥2)
{h[0]=f[0];
t[0]=f[0];
m=m+1;}
```

Step1.1 Let $f$ [$g$] be zero:

```plaintext
for(r=0;r<g;r++) f[r]=0;
```

Step1.2 For every $a_i(i=1,2,\ldots,k)$, find out minimum and maximum of $f(u_j, a_i)(j=1,2,\ldots,n)$, denoted by $\min$ and $\max$:

```plaintext
mini=f(u_1, a_i); maxi=f(u_n, a_i);
for(j=1;j<n+1;j++)
if(f(u_j, a_i)<mini) mini=f(u_j, a_i);
if(f(u_j, a_i)>maxi) maxi=f(u_j, a_i);
```

Step1.3 Compute size of every subsequence:

```plaintext
loc=(int)((g-1)/(f(u_i, a_i)−\min))/(\max–\min);
```

Step1.4 Store locations of subsequences whose sizes are greater than or equal to 2:

```plaintext
m=0;
\{if(0)≥2
{h[0]=0;
\{t[0]=0;
\}f[0]−1;
\}m=m+1;\}
```

for(r=1;r<=g−1;r++)

```plaintext
for(j=1;j<n+1;j++)
{loc=(int)((g-1)/(f(u_i, a_i)−\min))/(\max–\min));
```

Step1.5 Place every record anew:

```plaintext
\{if(f[r]≥2
\{h[m]=f[r−1];
t[m]=f[r−1]+f[r]−1;
m=m+1;
\}f[r]=f[r−1]+f[r];
```

Step1.6 Sort subsequences whose sizes are greater than or equal to 2 with quick sort:

```plaintext
while(m=0)
{sort s'[h[m], t[m]] with quick sort;
m=m–1;
```

Step2 Get equivalence classes:

```plaintext
d=1;L_1=\{ u_1' \};
\{for(j=2;j<n+1;j++)
if(f(u_j, a_i)=f(u_j, a_i) for \forall a_i)
L_d=L_d \cup \{ u_j' \};
else {d=d+1;L_d=\{ u_j' \};
```

Step3 Return L.

To analyse time complexity of Algorithm 1 more distinctly, Theorem 1 is given first.

**Theorem 1.** For $S=(U, A, V, f)$, $U=\{u_1, u_2, ..., u_n\}$, $C=\{a_1, ..., a_i, ..., a_k\}$, if $f(u_1, a_1)$, $f(u_2, a_2)$, ..., $f(u_n, a_i)$ are n independent observed values of arbitrary continuous random variable $\xi$, time complexity of segment quick sort for $f(u_1, a_1)$, $f(u_2, a_2)$, ..., $f(u_n, a_i)$ is $\Omega(|U|)$.

**Proof.** Let $F(x)$ be distribution function of $\xi$ and $F^{-1}(x)$ be inverse function of $F(x)$. The distribution of random variable $0=F(\xi)$ is discussed as follows.

\[ P(\theta<x)=P(F(\xi)<x)=P(\xi<F^{-1}(x))=F(F^{-1}(x))=x \]

(2) Because $\xi=F(\xi)$ obeys uniform distribution in the closed interval $[0, 1]$. $F(f(u_1, a_1)$, $F(f(u_2, a_2)$, ..., $F(f(u_n, a_i)$ are n independent observed values of $\theta$. Obviously, time complexity of map from $f(u_1, a_1)$, $f(u_2, a_2)$, ..., $f(u_n, a_i)$ to $F(f(u_1, a_1)$, $F(f(u_2, a_2)$, ..., $F(f(u_n, a_i)$ is $\Omega(|U|)$. Then, we use segment quick sort to sort $F(f(u_1, a_1)$, $F(f(u_2, a_2)$, ..., $F(f(u_n, a_i)$.

Since $g=[n/2]$, the number of observed values in the i-th segment is $\leq g$. Thus, average time of quick sort for the i-th segment $T_i=\sum_{i=1}^{g} C_{n_i}$, the total time of quick sort for all the segments $T=\sum_{i=1}^{g} T_i \leq \sum_{i=1}^{g} C_{n_i}$, where $C=\max \{C_i\}$, namely, $T=O(n)=O(|U|)$. So, $F(f(u_1, a_1)$, $F(f(u_2, a_2)$, ..., $F(f(u_n, a_i)$ are sorted. When $F(f(u_1, a_1)$, $F(f(u_2, a_2)$, ..., $F(f(u_n, a_i)$ are mapped to $f(u_1, a_1)$, $f(u_2, a_2)$, ..., $f(u_n, a_i)$, $A_1, A_2, ..., A_n$ are also sorted because $F^{-1}(x)$ is a monotone function.
increasing function. Time complexity for mapping is $O(|U|)$. Therefore, time complexity of segment quick sort for $f(u_i, a_j)$, $f(u_i, a_j)$, ..., $f(u_i, a_j)$ is $O(|U|)$. □

According to Theorem 1, time complexity of loop body in step 1 is $O(|U|)$. As cycle number is $O(|C|)$, time complexity of step 1 is $O(|C||U|)$. Time complexity of step 2 is also $O(|C||U|)$. Thus, time complexity of Algorithm 1 is $O(C||U|)$. For example, we can compute $U/\{a,b,c\}$ for Table I by Algorithm 1. Let $g=3$. First, decision table is sorted by attribute $a$. $U$ is divided into three subsequences: $\{u_4\}$, $\{u_2, u_3, u_5\}$ and $\{u_1, u_6\}$. Then, quick sort is used for the second and the third subsequence. Sorted by attribute $a$, $U$ is $\{u_5, u_3, u_2, u_1, u_6\}$. Similarly, we can sort Table I by other attributes in turn. Sorted by attribute $b$, $U$ is $\{u_1, u_2, u_3, u_4, u_5, u_6\}$. So, by step 2, $U/\{a,b,c\}$ is $\{\{u_1\}, \{u_2, u_3\}, \{u_4\}, \{u_5\}, \{u_6\}\}$. So, we need not to compute positive region before attribute reduction. During the process of equivalence class partition, universe is compressed gradually according to information about consistent equivalence classes. Finally, attribute reduction is found without computing positive region. Definition of consistent degree is given to describe this method more distinctly.

**Definition 9.** In decision table $S$, for $P \subseteq C$, consistent degree of $D$ with respect to $P$ is defined as following:

$$
CONC(D)=CONP(D)+\sum|E_i|
$$

(3)

where $E_i$ are the first two kinds of equivalence classes in $(U-POS_{P\cup\{a\}}(D))/U$. $|E_i|$ is the number of objects in $E_i$. Theorem 2.

Theorem 2. In the process of computing $U/C$, three kinds of equivalence classes may be produced. These equivalence classes is respectively: equivalence class which contains one object, consistent equivalence class, inconsistent equivalence class. Among them, the first kind and the second kind belong to $POS_C(D)$ definitely. The third kind need to be divided by remaining attributes to find whether there is a proper subset which belongs to $POS_C(D)$ or not. □

**Proof.** According to Definition 8, if the first kind of equivalence classes are divided by remaining attributes, they are also the first kind which contain one object. Obviously, they belong to $POS_C(D)$. If the second kind of equivalence classes are divided by remaining attributes, they are divided into more consistent equivalence classes which contain less objects and belong to $POS_C(D)$. If the third kind of equivalence classes are divided by remaining attributes, some consistent equivalence classes may be produced and they belong to $POS_C(D)$. So, we need not to compute positive region before attribute reduction. During the process of equivalence class partition, universe is compressed gradually according to information about consistent equivalence classes. Finally, attribute reduction is found without computing positive region. Definition of consistent degree is given to describe this method more distinctly.

**IV. DEFINITION AND ALGORITHM OF CONSISTENT DEGREE**

In this section, we will research the consistent degree. Firstly, Theorem 2 is given.

**Theorem 2.** In the process of computing $U/C$, three kinds of equivalence classes may be produced. These equivalence classes is respectively: equivalence class which contains one object, consistent equivalence class, inconsistent equivalence class. Among them, the first kind and the second kind belong to $POS_C(D)$ definitely. The third kind need to be divided by remaining attributes to find whether there is a proper subset which belongs to $POS_C(D)$ or not.

**Proof.** According to Definition 8, if the first kind of equivalence classes are divided by remaining attributes, they are also the first kind which contain one object. Obviously, they belong to $POS_C(D)$. If the second kind of equivalence classes are divided by remaining attributes, they are divided into more consistent equivalence classes which contain less objects and belong to $POS_C(D)$. If the third kind of equivalence classes are divided by remaining attributes, some consistent equivalence classes may be produced and they belong to $POS_C(D)$. So, we need not to compute positive region before attribute reduction. During the process of equivalence class partition, universe is compressed gradually according to information about consistent equivalence classes. Finally, attribute reduction is found without computing positive region. Definition of consistent degree is given to describe this method more distinctly.

**Definition 9.** In decision table $S$, for $P \subseteq C$, consistent degree of $D$ with respect to $P$ is defined as following:

$$
CONC(D)=CONP(D)+\sum|E_i|
$$

(3)

where $E_i$ are the first two kinds of equivalence classes in $(U-POS_{P\cup\{a\}}(D))/U$. $|E_i|$ is the number of objects in $E_i$. From the final result, $CONC(D)$ seems to be just the number of elements in $POS_C(D)$. In fact, difference between them lies in solving process, which we can see from the Definition and the algorithm of consistent degree. The computation of consistent degree need not to obtain positive region firstly and count the number of objects in the positive region. Contrarily, the number of objects in the first and the second kind of equivalence classes are added up continually during the process of sorting decision table by every attribute in turn. After each summation, the accumulated objects are deleted from universe. Thus, universe probably shrinks. The computation may finishes without sorting decision table by the last few attributes. Therefore, this method can greatly shorten the time of calculation and upgrade efficiency.

The followed is the solution algorithm of consistent degree.

**Algorithm 2: Algorithm for computing consistent degree**

**Input:** $S=\{U, C \cup D, V, f\}$, $C=\{a_1, a_2, ..., a_k\}$.

**Output:** $CONC(D)$.

Step 1 $CONC(D)=0$.

Step 2 Scan values of decision attribute $D$. If they are not the same, decision table is sorted by attribute $a$ first. $U$ is divided into three equivalence classes: $\{u_4\}$, $\{u_2, u_3, u_5\}$ and $\{u_1, u_6\}$. Then, quick sort is used for the second subsequence. Sorted by attribute $b$, $U$ is $\{u_1, u_2, u_3, u_4, u_5, u_6\}$. So, by step 2, $U/\{a,b,c\}$ is $\{\{u_1\}, \{u_2, u_3\}, \{u_4\}, \{u_5\}, \{u_6\}\}$. So, we need not to compute positive region before attribute reduction. During the process of equivalence class partition, universe is compressed gradually according to information about consistent equivalence classes. Finally, attribute reduction is found without computing positive region. Definition of consistent degree is given to describe this method more distinctly.

**Definition 9.** In decision table $S$, for $P \subseteq C$, consistent degree of $D$ with respect to $P$ is defined as following:

$$
CONC(D)=CONP(D)+\sum|E_i|
$$

(3)

where $E_i$ are the first two kinds of equivalence classes in $(U-POS_{P\cup\{a\}}(D))/U$. $|E_i|$ is the number of objects in $E_i$. From the final result, $CONC(D)$ seems to be just the number of elements in $POS_C(D)$. In fact, difference between them lies in solving process, which we can see from the Definition and the algorithm of consistent degree. The computation of consistent degree need not to obtain positive region firstly and count the number of objects in the positive region. Contrarily, the number of objects in the first and the second kind of equivalence classes are added up continually during the process of sorting decision table by every attribute in turn. After each summation, the accumulated objects are deleted from universe. Thus, universe probably shrinks. The computation may finish without sorting decision table by the last few attributes. Therefore, this method can greatly shorten the time of calculation and upgrade efficiency.

The followed is the solution algorithm of consistent degree.

**Algorithm 2: Algorithm for computing consistent degree**

**Input:** $S=\{U, C \cup D, V, f\}$, $C=\{a_1, a_2, ..., a_k\}$.

**Output:** $CONC(D)$.

**Step 1** $CONC(D)=0$.

**Step 2** Scan values of decision attribute $D$. If they are not the same, go to Step 3.

**Step 3** for $i=1; i<=k; i++$

If universe is not an empty set, sort objects in the universe by the $i$-th attribute on the basis of the last sort according to the Algorithm 1. Then scan values of decision attribute $D$ of every equivalence class, if it is the first or the second kind of equivalence class, add the number of objects of it to $CONC(D)$ and delete it from universe.

Otherwise, go to Step 4.

**Step 4** Return $CONC(D)$.

In Algorithm 2, time complexity of step 2 is $O(|U|)$. Time complexity of step 3 is $O(|C||U|)$. So, time complexity of Algorithm 2 is $O(|C||U|)$. We give the following example which computes consistent degree of Table I to illustrate Algorithm 2. Let $CONC(D)$ be zero. Because values of decision attribute $D$ are not the same, decision table is sorted by attribute $a$ first. $U$ is divided into three equivalence classes: $\{u_4\}$, $\{u_2, u_3, u_5\}$ and $\{u_1, u_6\}$. Among them, $\{u_4\}$ is the first kind of equivalence class and $\{u_1, u_6\}$ is the second kind of equivalence class. So, 1 and 2 are added to $CONC(D)$ and
{u_4, u_5, u_6} are deleted from universe. \( \text{CON}_{C}(D) \) is 3 now. Then, \{u_4, u_5, u_6\} is sorted by attribute b and divided into two equivalence classes: \{u_2, u_3\} and \{u_4\}. \{u_2, u_3\} is the first kind of equivalence class. Thus, 1 is added to \( \text{CON}_{C}(D) \) and \( \text{CON}_{C}(D) \) is 4 now. Last, \{u_2, u_3\} is deleted from universe. \( \text{CON}_{C}(D) \) is 4 now. Last, \{u_2, u_3\} is deleted from universe. \( \text{CON}_{C}(D) \) is 3 now.

To get a attribute reduction by heuristic algorithms, the core is usually used as initial reduction. Thus, definition of core in our algorithm is given first.

\textbf{Definition 10}. In \( S=(U,C \cup \{D\}, f) \), for \( a \in C \), \( \text{CORE}_{a}(C) \) is defined as following:

\[
\text{CORE}_{a}(C)=\{a|a \in C \land \text{CON}_{C}\{a\}(D)<\text{CON}_{C}(D)\}
\]

(4)

\text{Definition 10} indicates that a is a core attribute if consistent degree decreases when a is deleted. Otherwise, a is not core attribute. This method avoids tedious process in positive region strategy and heavy computing load in discernibility matrix strategy.

\textbf{Algorithm 3}: Algorithm for computing \( \text{CORE}_{a}(C) \)

\textbf{Input}: \( S=(U,C \cup \{D\}, f) \), \( R \subseteq C \land R \subseteq C \land R \), \( a \in C \land \text{CORE}_{a}(C) \) is \( \emptyset \).

\textbf{Output}: \( \text{CORE}_{a}(C) \).

\text{Step1} Compute \( \text{CON}_{C}(D) \) according to Algorithm 2.

\text{Step2} Compute \( \text{CON}_{C}\{a\}(D) \) according to Algorithm 2.

\text{Step3}\text{.1} if(\( \text{CON}_{C}\{a\}(D)<\text{CON}_{C}(D) \))

\text{Step3}\text{.2} Compute \( \text{CORE}_{a}(C) \) according to Algorithm 2.

\text{Step4} Return \( \text{CORE}_{a}(C) \).

In \text{Algorithm 3}, time complexity of \text{Step 2} is \( O(|C||U|) \). As cycle number is \( O(|C|) \) and time complexity of \text{Step3.1} is \( O(|C||U|) \), time complexity of \text{Algorithm 3} is \( O(|C||U|) \). Thus, time complexity of \text{Algorithm 3} is \( O(|C||U|)+O(|C|^2|U|)=O(|C|^2|U|) \).

For instance, we can get core of Table 1 by \text{Algorithm 3}. First, \( \text{CON}_{C}(D) \) is 4 according to \text{Algorithm 2}. Then, \( \text{CON}_{C}\{a\}(D) \) is 1. \( \text{CON}_{C}\{b\}(D) \) is 4 too. So, \( \text{CORE}_{a}(C) \) is \{a\}.

\textbf{V. EFFICIENT ALGORITHM FOR COMPUTING CORE}

To get a attribute reduction by heuristic algorithms, the core is usually used as initial reduction. Thus, definition of core in our algorithm is given first.

\textbf{Definition 11}. In \( S=(U,C \cup \{D\}, f) \), for \( a \in C \), \( \text{CORE}_{a}(C) \) is defined as following:

\[
\text{CORE}_{a}(C)=\{a|a \in C \land \text{CON}_{C}\{a\}(D)<\text{CON}_{C}(D)\}
\]

\text{Definition 11} indicates that a is core attribute if consistent degree decreases when a is deleted. Otherwise, a is not core attribute. This method avoids tedious process in positive region strategy and heavy computing load in discernibility matrix strategy.

\textbf{Algorithm 3}: Algorithm for computing \( \text{CORE}_{a}(C) \)

\textbf{Input}: \( S=(U,C \cup \{D\}, f) \), \( R \subseteq C \land R \subseteq C \land R \), \( a \in C \land \text{CORE}_{a}(C) \) is \( \emptyset \).

\textbf{Output}: \( \text{CORE}_{a}(C) \).

\text{Step1} Compute \( \text{CON}_{R \cup \{a\}}(D) \) according to \text{Algorithm 2}.

\text{Step2} Compute \( \text{IMP}_{C-R}(a) \) according to \text{Definition 11}.

\text{Step3} Return \( \text{IMP}_{C-R}(a) \).

Because time complexity of \text{Algorithm 2} is \( O(|C||U|) \), time complexity of \text{Step 1} in \text{Algorithm 4} is \( O(|U|) \). So, time complexity of \text{Algorithm 4} is \( O(|U|) \). For Table 1, let \( R=\text{CORE}_{a}(C) \). \( \text{CON}_{R}(D) \) is 3. \( \text{IMP}_{C-R}(b) \) is 1. \( \text{IMP}_{C-R}(c) \) is 1 too.

\textbf{VII. QUICK ATTRIBUTE REDUCTION ALGORITHM BASED ON CONSISTENT DEGREE}

In this section, we let the initial reduction be the core [15]. Then a non-core attribute which has the maximum importance is added to initial reduction. When consistent degree of current reduction is equal to that of all the decision attributes, the computation finishes.

\textbf{Theorem 3}. In \( S=(U,C \cup \{D\}, f) \), for \( R \subseteq C \land R \), \( a \) is a attribute reduction if and only if \( \text{CON}_{R}(D)=\text{CON}_{C}(D) \).

\textbf{Proof}. \( \text{POS}_{R}(D)=\text{POS}_{C}(D) \Leftrightarrow \text{CON}_{R}(D)=\text{CON}_{C}(D) \) needs to be proved.

(1) \( \text{POS}_{R}(D)=\text{POS}_{C}(D) \Leftrightarrow \text{CON}_{R}(D)=\text{CON}_{C}(D) \) can be proved first.

Because \( \text{POS}_{R}(D)=\text{POS}_{C}(D) \), \( |\text{POS}_{R}(D)|=|\text{POS}_{C}(D)| \).

Then \( \text{CON}_{R}(D)=\text{CON}_{C}(D) \).

(2) \( \text{CON}_{R}(D)=\text{CON}_{C}(D) \Rightarrow \text{POS}_{R}(D)=\text{POS}_{C}(D) \) can be proved next.

① Since \( R \subseteq C \land R \subseteq \text{POS}_{C}(D) \).

② Suppose \( \text{POS}_{R}(D) \subset \text{POS}_{C}(D) \), we can see that \( |\text{POS}_{R}(D)|<|\text{POS}_{C}(D)| \). So we can obtain that \( \text{CON}_{R}(D)<\text{CON}_{C}(D) \). This result is in contradiction with \( \text{CON}_{R}(D)=\text{CON}_{C}(D) \). Therefore, \( \text{POS}_{R}(D) \subseteq \text{POS}_{C}(D) \).

According to ① and ②, \( \text{POS}_{R}(D)=\text{POS}_{C}(D) \).

According to (1) and (2), \text{Theorem 3} is proved. □

According to Theorem 3 and Algorithms mentioned above, quick attribute reduction algorithm is given.

\textbf{Algorithm 5}: Algorithm for computing attribute reduction

\textbf{Input}: \( S=(U,C \cup \{D\}, f) \).

\textbf{Output}: attribute reduction R.

\text{Step1} Compute \( \text{CON}_{R}(D) \) according to \text{Algorithm 2}.

\text{Compute \( \text{CORE}_{a}(C) \) according to \text{Algorithm 3}. Let \( R=\text{CORE}_{a}(C) \). \text{Compute \( \text{CON}_{R}(D) \) according to \text{Algorithm 2}.}

\text{Step2} while(\( \text{CON}_{R}(D)<\text{CON}_{C}(D) \))

\text{Step2.1 if(i=1;i<=|C-R|;i++)}

\text{Compute IMP}_{C-R}(a_{i}) according to \text{Algorithm 4}.

\text{Step2.2 Choose a_{i} which has the maximum IMP}_{C-R}(a_{i}).}

\text{If there are several such a_{i}, pick one at random.}

\text{Step2.3 R=R \cup \{a_{i}\}.}

\text{Step2.4 Recalculate \text{CON}_{R}(D).}

\}

\text{Step3} Return R.

DOI 10.5013/IJSSST.a.17.37.25 25.5 ISSN: 1473-804x online, 1473-8031 print
In Algorithm 5, time complexity of Step 1 is $O(|C|^2|U|)$. Cycle number of Step 2 is $|C-R|$ at most. Time complexity of Step 2.1 is $O(|C-R||U|)$. Time complexity of Step 2.2 is $O(|C-R||U|)$. So, time complexity of Step 2 is $O(|C|^2|U|)$. Therefore, time complexity of Algorithm 5 is $O(|C|^2|U|)$.

**Theorem 4.** Algorithm 5 is a complete attribute reduction algorithm.

**Proof.** According to [1], an algorithm is called a complete attribute reduction algorithm if attributes set $R \subseteq C$ obtained by the algorithm can satisfy the following two conditions.

1. $POSC(D)=POSC(R(D))$;
2. For any $a \in R$, $POSC_{[a]}(D) \neq POSC(D)$.

According to Theorem 3, Algorithm 5 satisfies the first condition obviously. If any attribute $a$ is deleted from reduction obtained by Algorithm 5, $CON_{R_{[a]}(D)} < CON_{C(D)}$. That is to say, Algorithm 5 satisfies the second condition. Therefore, Algorithm 5 is a complete attribute reduction algorithm.

Now, we can give attribute reduction of Table I according to Algorithm 2. $CON_{C(D)}$ is 4. According to Algorithm 3, $CORE_{[a]}(C)$ is $\{a\}$. Let $R=\{a\}$, $CON_{R}(D)$ is 3. Because $CON_{R}(D)$ is less than $CON_{C}(D)$, $IMP_{C-[a]}^{D}(b)$ and $IMP_{C-[a]}^{D}(c)$ are computed. $IMP_{C-[a]}^{D}(b)$ is 1. $IMP_{C-[a]}^{D}(c)$ is 1 too. So, attribute $b$ is chosen randomly and added to $\{a\}$. Thus, $R=\{a, b\}$. Since $CON_{R}(D)$ is 4 and equal to $CON_{C}(D)$, Algorithm 5 finishes. Finally a attribute reduction is $\{a, b\}$.

**VIII. EXPERIMENTAL RESULTS AND ANALYSIS**

We use six decision tables from UCI Repository of Machine Learning Databases to test the performances of the Algorithms mentioned above. All the experiments have been carried out on a personal computer with Windows 7, Inter Core i3 CPU M380(2.53 GHz) and 4 GB memory. We compare the time for computing reduction of each decision table by algorithm in [13], algorithm in [14] and Algorithm 5 in this paper. The above three algorithms are denoted by Table by algorithm in [13], algorithm in [14] and Algorithm 5 respectively. Experimental data are shown in Table II. Experimental results can be seen from Table III.

**REFERENCES**


