Attribute Reduction Based on Sorting and Incremental Method

Wang Biqing*

College of Mathematics and Computer, Tongling University, Tongling, Anhui, 244000, P.R. China

Abstract: Positive region is one of the core concepts in rough set theory. Time complexity of computing the positive region can directly affect other algorithms. In this paper, a new algorithm for computing equivalence classes based on generalized quick sort and insertion sort is provided and its time complexity for computing U/C is cut down to O(|C||U|) compared with other traditional algorithms. On this basis, an algorithm for fast computing positive region by adding identifier atttibutes to sorted decision tables is proposed and its time complexity for computing $POS_C(D)$ is cut down to O(|C||U|) accordingly. By using this foundation, two incremental algorithms for fast computing positive region are designed. They make full use of equivalence classes and positive region which already exist to compute positive region incrementally. Thus, these algorithms can reduce computational work signicantly and get higher efficiency. Among them, the algorithm based on multiway tree is the most efficient. Its time complexity for computing $POS_C(D)$ is far less than O(|C||U/C|). Finally, a attribute reduction algorithm based on incremental method is given and its time complexity is $O(|C|^2|U/C|)$. The example analysis and experimental result show that the attribute reduction algorithm of this paper are feasible and efficient.

Keywords: Attribute reduction, incremental algorithm, positive region, rough set, sorting.

1. INTRODUCTION

Rough set theory [1] is a mathematical tool which processes fuzzy and uncertain knowledge. It is widely used in artificial intelligence, machine learning, decision analysis, process control, pattern recognition, data mining and other fields. At present, researches on rough set algorithm include how to compute upper approximation, lower approximation, equivalence class, positive region, reduction, core, and so on.

As reduction [2] is one of the basic concepts in the rough set theory, researches on rough set algorithm focus on attribute reduction mainly, including attribute in consistent decision tables and inconsistent decision table. Among them, many are based on positive region [3]. In rough set theory, positive region is also a very important concept. Computation of equivalence classes and positive region directly affect other algorithms's time complexity. So, how to compute positive region efficiently is the key to increasing the efficiency of its correlation algorithm.

For attribute reduction based on positive region, U/C must be computed either in [4] which uses analysis method or in [5] which uses differential matrix. So U/C must be computed first in attribute reduction algorithms based on the positive region. Time complexity of traditional algorithms for computing U/C is $O(|C||U|^2)$ and it is not satisfactory. Reference [6] puts forward an algorithm for computing U/C based on quick sort and its time complexity is O(|C||U|log|U|). On this basis, an algorithm for computing U/C based on radix sort is presented in [7] and its time complexity is cut down to O(|C||U|). But it is obtained on the basis that attribute value is one digit data. As time complexity of radix sort is O(d|U|) where d is digit capacity of attribute value, time complexity of the algorithm in [7] is O(d|C||U|) actually. In this paper, an algorithm for computing U/C based on generalized quick sort and insertion sort is provided and its time complexity is cut down to O(|C||U|).

Although computation of positive region is very important, relatively few researches on algorithm of positive region are done. Traditional algorithms [8] for computing $POS_C(D)$ judge whether every object in U belongs to existing equivalence classes according to the value of condition attributes C, so as to compute positive region. The amount of calculation is great and its time complexity is $O(|C||U|^2)$. Reference [9] judges whether unclassified objects in U have the same values of every attribute in condition attributes C. If so, they belong to a same equivalence class. In the worst case, its time complexity is $O(|C||U|^2)$ like the traditional algorithms for computing $POS_C(D)$. Reference [10] gives several new properties of positive region, including record filtering property, substitution property and decomposition property etc. These properties are useful for simplifying the dynamical computation of positive region and its time complexity is O(|C||U|log|U|). In [11], an algorithm for computing the equivalence classes based on radix sort by using distributing counting is designed. Besides, time complexity for computing $POS_C(D)$ using the algorithm is also O(|C||U|log|U|). To improve efficiency of the algorithms for computing positive region, in this paper, an algorithm for fast computing positive region by adding identifier atttibutes to sorted decision tables is proposed on the basis of an in-depth study of rough set theory. The algorithm's time complexity is O(|C||U|). The detailed description of the algorithm and time complexity analysis are given. The theoretical analysis and experimental results indicate that this algorithm can decrease the computational complexity effectively.

In recent years, most researches on positive region are focused on static information system. When the amount of objects increases, recomputation of positive region will waste many time and space resources. So, more effective measures must be taken. Thus, two incremental algorithms for fast computing positive region are presented in this paper. The first one is by the attributes increased. The second one is by the objects increased based on multiway tree. They all obtain positive region incrementally on the basis of equivalence classes and positive region which already exist. So, these algorithms can reduce the computational work signicantly and get higher efficiency. Time complexity of the second incremental algorithm is far less than O(|C||U/C|). Finally, a attribute reduction algorithm based on the second incremental method is proposed and its time complexity is $O(|C|^2|U/C|)$.

2. PRELIMINARIES

In this section, we review some basic concepts in rough set [1], [12] 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 Va is the set of values of attribute a; f:U×A→V is an information function, which determines values of attribute of every object u, namely, $f(u,a)\in Va$ for every $u\in U$ and $a\in A$.

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

Definition 2. In information system S, for every attribute subset $P\subseteq A$, an indiscernibility relation IND(P) is defined as $IND(P)=\{(x,y)\in U\times U: \forall a\in P, f(x,a)=f(y,a)\}$.

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

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 3. In information system S, for $\forall R\subseteq A$, $\forall X\subseteq U$, $U/R=\{R_1,R_2,\ldots,R_l\}$, lower approximation is defined as $\underline{R}X=U\{R_i|R_i\in U/R,R_i\in X\}$, upper approximations is defined as $\overline{R}(X)=U\{R_i|R_i\in U/R,R_i\cap X\neq\varnothing\}$.

Definition 4. In decision table S, if $f(u_i,C)=f(u_j,C)$ and $f(u_i,D)\neq f(u_j,D)$ for $u_i,u_j\in U$ and $i\neq j$, S is called an inconsistent decision table. Otherwise S is called a consistent decision table.

Definition 5. In information system S, for $\forall P,Q\subseteq A$, positive region $POS_P(Q)$ is defined as $POS_P(Q)=\bigcup_{X\in U/Q}\underline{P}X$, where $\underline{P}X$ is P lower approximation of X.

Definition 6. In decision table S, for $\forall P,Q\subseteq A$, let $U/P=\{P_1,P_2,\ldots,P_t\}$ and $U/Q=\{Q_1,Q_2,\ldots,Q_S\}$. If $\forall P_i\in U/P$, there exists $Q_j\in U/Q$ 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.

Definition 7. 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 and defined as $CORE_D(C)$.

Definition 8. In decision table S, for $R\subseteq C$, R is called a attribute reduction of C if $POS_R(Q)=POS_C(D)$ and all the attributes in R are necessary.

3. A NEW ALGORITHM FOR COMPUTING U/P BASED ON GENERALIZED QUICK SORT AND INSERTION SORT

After a thorough study on algorithms for computing U/P in previous literatures, a new algorithm is given. We can sort decision table by attribute subset P using idea of generalized quick sort and insertion sort. Then, sorted decision table is analysed and equivalence classes are divided. Time complexity for computing U/P is O(|P||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 logN times move to get its final position. So, time complexity for quick sort is O(NlogN). It can be generalized to present following algorithm 1. To describe more distinctly, Theorem 1 and Theorem 2 are given first.

Theorem 1. In decision table S, two objects $u_i,u_j\in U$ belong to one equivalence class for set of attributes P if and only if $f(u_i,a)=f(u_j,a)$ for $\forall a\in P$.

Proof. According to definition of equivalence class in Definition 2, if u_i,u_j belong to one equivalence class for set of attributes P, u_iPu_j. According to definition of indiscernibility relation in Definition 2, $f(u_i,a)=f(u_j,a)$ for $\forall a\in P$.

Theorem 2. Let min_i and max_i be minimum and maximum of $f(u_j,a_i)(j=1,2,\ldots,n)$ respectively, $f(u_j,a_i)(j=1, 2,\ldots,n)$ can be divided into g subsequences by equation (1)

$$loc=(int)((g - 1)(f(u_j,a_i) - min_i)/(max_i - min_i)) \qquad (1)$$

where loc denotes subsequence number of $f(u_j,a_i)$.

Proof. If $f(u_j,a_i)=min_i$, loc=0. If $f(u_j,a_i)=max_i$, loc=g - 1. If $f(u_j,a_i)$ takes other value, loc is between 0 and g - 1. So, there are g subsequences in all.

Based on Theorem 1 and Theorem 2, concepts of quick sort and insertion sort, skeleton of computing U/P is given.

Sorting decision table by attribute subset P 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 relation- ship among the datum, minimum and maximum. Next, size of each subsequence can be counted *via* datum's location. Hence, we may arranges 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, every subsequence is sorted recursively. Thus, the whole sequence can be sorted. In orde to improve the computational efficiency better, other sorting method can be used when size of subsequence is small. We suggest using insertion sort to sort subsequence. When subsequence is not very large, insertion sort is faster relatively. Then, 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 $R_j(j=1, 2,\cdots,n)$. $s'[n]$ is a array of structure which is used to store sorted decision table. g is the number of subsequences and f[g] is a array which stores sizes of every subsequence. *l* is the max size of subsequence which may be sorted by insertion sort. h[n] is a array which stores "head information" of subsequence whose size is greater than *l*; h[m] stores subscript of the first element of the (m+1)-th subsequence whose size is greater than *l* in $s'[n]$. t[n] is a array which stores "tail information" of subsequence whose size is greater than *l*; t[m] stores subscript of the last element of the (m+1)-th subsequence whose size is greater than *l* in $s'[n]$. $h'[n]$ is a array which stores "head information" of subsequence whose size is not greater than *l*; $h'[m']$ stores the subscript of the first element of the $(m'+1)$-th subsequence whose size is not greater than *l* in $s'[n]$. $t'[n]$ is a array which stores "tail information" of subsequence whose size is not greater than *l*; $t'[m']$ stores subscript of the last element of the $(m'+1)$-th subsequence whose size is not greater than *l* in $s'[n]$. Let $s'[b..e]$ be the subsequence which need to be sorted by insertion sort. key is a structure variable which is used temporarily. Let $\{u'_1, u'_2, \cdots, u'_n\}$ be objects series obtained through sorting decision table by attribute subset P. The i-th equivalence class is denoted by L_i and the set of L_i is L. So, this algorithm (Algorithm **1**) may be described as follows.

Algorithm 1: Algorithm for computing U/P

Input: S=(U,A,V,f), U={$u_1, u_2, \ldots, u_n$}, $P \subseteq C$ and P={$a_1, a_2, \ldots, a_k$}.

Output: U/P.

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

```
for(i=1;i<k+1;i++)
```

Step 1.1. Let f[g] be set zero**:**

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

Step 1.2. 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_i and max_i**:**

```
min_i=f(u_1,a_i);max_i=f(u_1,a_i);
for(j=1;i<n+1;j++)
  {if(f(u_j,a_i)<min_i) min_i=f(u_j,a_i);
   if(f(u_j,a_i)>max_i) max_i=f(u_j,a_i);}
```

Step 1.3. Compute size of every subsequence**:**

```
for(j=1;i<n+1;j++)
  {loc=(int)((g - 1)(f(u_j,a_i) - min_i)/ (max_i - min_i));
   f[loc]=f[loc]+1;}
```

Step 1.4. Store location of each subsequence in s':

```
m=0; m'=0
if (f[0]> l)
  {h[0]=0;
   t[0]= f[0] - 1;
   m=m+l;}
 else
   {h'[0]=0;
   t'[0]=f[0] - 1;
   m'=m'+1;}
for(r=1;r<g;r++)
  {if (f[r]> l)
    {h[m]=f[r - 1];
     t[m]=f[r - l]+f[r] - 1;
     m=m+l; }
    else
      {h'[m']=f[r - 1];
     t'[m']=f[r - l]+f[r] - 1;
     m'=m'+1;}
   f[r]=f[r - l]+f[r];}
```

Step 1.5. Place every record anew**:**

```
for(j=1;i<n+1;j++)
    {loc=(int)((g - 1)(f(u_j,a_i) - min_i)/ (max_i - min_i));
    s'[f(loc) - 1]=R_j;
    f[loc]=f[loc] - 1;}
```

Step1.6. Divide subsequences whose size is greater than *l* into g subsequences recursively**:**

```
While(m>0)
    {Divide s'[h[m]..t[m]] into g subsequences re-
     cursively;
     m=m - 1;}
```

Table 1. Decision table.

U	a	b	c	d	D
u_1	2	2	3	3	0
u_2	5	2	3	1	1
u_3	4	2	2	2	2
u_4	1	4	2	1	3
u_5	2	2	3	3	4
u_6	1	4	2	1	3
u_7	4	4	4	1	0
u_8	3	3	1	1	1
u_9	3	1	4	2	4
u_{10}	5	3	1	3	4
u_{11}	4	2	2	2	3
u_{12}	2	4	5	4	1
u_{13}	5	3	1	3	0
u_{14}	4	4	4	1	0
u_{15}	1	3	1	2	4

Step 1.7. Use insertion sort to sort subsequences:

```
for(j=b+1;i<e+1;j++)
  {key=$s'$[j]
   w=j - 1;
   while(w>(b - 1)&& $s'$[w].$a_i$>key.$a_i$)
     {$s'$[w+1]=$s'$[w];
      w=w - 1;}
   $s'$[w+1]= key;}
```

Step 2. Get equivalence classes:

```
d=1;$L_1$={$u'_1$};
for (j=2;j<n+1;j++)
  if(f($u'_j$,$a_i$)=f($u'_{j-1}$,$a_i$) for $\forall a_i$) $L_d=L_d \cup \{u'_j\}$;
  else {d=d+1;$L_d$={$u'_j$};}
```

Step 3. Return L.

In normal conditions, an attribute sequence only need several finite times division to get small subsequences for insertion sort. Therefore, time complexity of the first six step in step 1 is O(|U|) obviously. In the seventh step in step 1, size of every subsequence is l at most. In the worst case, subsequentce is nonincreasing. Thus, comparison times is $\sum_{i=1}^{l-1} i = l(l-1)/2$ by using insertion sort for one subsequence and total comparison times is $(|U|/l)l(l-1)/2$. Because l is a constant, time complexity of the seventh step in step 1 is O(|U|) too. As cycle number is O(|P|), time complexity of step 1 is O(|P||U|). Time complexity of step 2 is also O(|P||U|). Thus, time complexity of Algorithm 1 is O(|P||U|). If P=C, time complexity for computing U/C is O(|C||U|). To illustrate the above algorithm, let us consider the following example.

Table **1** presents a decision table. We can compute U/{a,b, c,d} by using Algorithm 1. Let g=3, l=4.

First, decision table is sorted by attribute a. U is divided into three subsequences: $\{u_1,u_4,u_5,u_6,u_{12},u_{15}\}$, $\{u_3,u_7,u_8,u_9,u_{11}, u_{14}\}$ and $\{u_2,u_{10},u_{13}\}$. Among them, size of the third subsequence is less then 4. So, it doesn't need further division. Then, the first subsequence is divided into three subsequences: $\{u_4,u_6,u_{15}\}$, $\varnothing$ and $\{u_1,u_5,u_{12}\}$. The second subsequence is divided into three subsequences: $\{u_8, u_9\}$, $\varnothing$ and $\{u_3,u_7,u_{11}, u_{14}\}$. Now, all the subsequences are not greater than 4 and division finish. Next, insertion sort is used for each subsequence. In this way, sequence of U obtained is $\{u_{15},u_6, u_4,u_{12},u_5,u_1,u_9,u_8,u_{14},u_{11},u_7,u_3,u_{13},u_{10},u_2\}$.

Similarly, we can sort decision table by other attributes in turn. Thus, sequence of U obtained through sorting decision table by attribute b is $\{u_9,u_2,u_3,u_{11},u_1,u_5,u_{10},u_{13},u_8,u_{15},u_7,u_{14}, u_{12},u_4,u_6\}$. Sequence of U obtained through sorting decision table by attribute c is $\{u_{15},u_8,u_{13},u_{10},u_6,u_4,u_{11},u_3, u_5,u_1,u_2,u_{14}, u_7,u_9,u_{12}\}$. Sequence of U obtained through sorting decision table by attribute d is $\{u_7,u_{14},u_2,u_4,u_6, u_8,u_9,u_3,u_{11},u_{15}, u_1,u_5,u_{10}, u_{13},u_{12}\}$.

Finally, by step 3, U/{a,b,c,d} is $\{\{u_7,u_{14}\},\{u_2\},\{u_4,u_6\},\{u_8\},\{u_9\},\{u_3,u_{11}\},\{u_{15}\},\{u_1,u_5\},\{u_{10},u_{13}\},\{u_{12}\}\}$.

4. AN ALGORITHM FOR FAST COMPUTING $POS_P(Q)$ BY ADDING IDENTIFIER ATTTIBUTES TO SORTED DECISION TABLES

On the basis of Algorithm 1, we present a fast algorithm for computing $POS_P(Q)$. Firstly, idea of identifier is introduced to distinguish equivalence classes. Every equivalence class can be identified by an identifier. Identifier atttibutes are added to sorted decision tables and their values are every object's identifier of equivalence class. In order to describe the method for computing $POS_P(Q)$ more distinctly, Theorem 3 and Theorem 4 are given first. They are theoretical basis of Algorithm 2. Let I be an identifier atttibute.

Theorem 3. If $Y\in U/P$ and there exists $f(u_i,I)\neq f(u_j,I)$, $Y\not\subset POS_P(Q)$ for $u_i,u_j\in Y$ and $i\neq j$.

Proof. As $f(u_i,I)\neq f(u_j,I)$, $|Y/Q|\neq 1$. So, $Y\not\subset \underline{P}X$ $(X\in U/Q)$. Therefore, $Y\not\subset POS_P(Q)$. □

Theorem 4. $POS_P(Q)=\cup\{[u_i]_P|[u_i]_P=[u_j]_P\wedge f(u_i, I)=f(u_j, I)\}$ for u_i, $u_j\in U$ and $i\neq j$.

Proof. Let $U/P=\{[u]_P|u\in U\}$ and let $U/Q=\{[u]_Q|\ u\in U\}$. Suppose $u\in\cup\{[u_i]_P|[u_i]_P=[u_j]_P\ \wedge f(u_i, I)= f(u_j, I)\}$. Then there exists $[u_s]_P\in\{[u]_P|u\in U\}$ and $|[u_s]_P/Q|=1$ such that $u\in[u_s]_P$. So, there exists $[u_t]_Q\in U/Q$ such that $[u_s]_P\subseteq[u_t]_Q$. According to properties of lower approximation, $\underline{P}[u_s]_P=[u_s]_P$ and $\underline{P}[u_s]_P\subseteq\underline{P}[u_t]_Q$. Thus, we can get $[u_s]_P\subseteq\underline{P}[u_t]_Q\subseteq POS_P(Q)$. Thus, $u\in POS_P(Q)$. Therefore, $\cup\{[u_i]_P|[u_i]_P=[u_j]_P\wedge f(u_i,I)=f(u_j,I)\}\subseteq POS_P(Q)$.

Suppose $u\in POS_P(Q)$. As $POS_P(Q)=\bigcup_{X\in U/Q}\underline{P}X=\bigcup_{u\in U}\underline{P}[u]_Q$, there exists $[u_t]_Q\in\{[u]_Q|u\in U\}$ such that $u\in\underline{P}[u_t]_Q$. Because $\underline{P}[u_t]_Q=\cup\{[u]_P|[u]_P\subseteq[u_t]_Q\}$, there exists $[u_s]_P\in\cup\{[u]_P|[u]_P\subseteq[u_t]_Q\}$ such that $u\in[u_s]_P$. As $[u_s]_P\subseteq[u_t]_Q$, every objects in $[u_s]_P$ has the same value of attribute I. So, we get $[u_s]_P\subseteq\cup\{[u_i]_P|[u_i]_P=[u_j]_P\wedge f(u_i,I)=f(u_j,I)\}$, namely, $u\in\cup\{[u_i]_P|[u_i]_P=[u_j]_P\wedge f(u_i, I)=f(u_j, I)\}$. Therefore, $POS_P(Q)\subseteq\cup\{[u_i]_P|[u_i]_P=[u_j]_P\wedge f(u_i,I)=f(u_j,I)\}$.

On the basis of Theorem 3 and Theorem 4, Algorithm **2** may be described as follows.

Algorithm 2: Algorithm for computing $POS_P(Q)$ by adding identifier atttibutes

Input: S=(U,A,V,f) and $P,Q\subseteq A$.

Output: $POS_P(Q)$.

Step 1. Get decision table S_P sorted by attribute subset P and S_Q sorted by attribute subset Q according to the Algorithm 1. Add attribute PI and attribute QI to S_P. Add attribute QI to S_Q. Fill attribute PI of S_P with identifiers of equivalence classes of U/P. Fill attribute QI of S_Q with identifiers of equivalence classes of U/Q.

Step 2. Sort S_Q by attribute subset P according to the Algori- thm 1 and then copy attribute QI of S_Q to attribute QI of S_P.

Step 3. Scan S_P and if all the objects in one equivalence class of U/P have the same identifier of equivalence class of U/Q, copy out the objects into $POS_P(Q)$. Otherwise, scan the next equivalence class of U/P. When S_P is scanned completely, algorithm finish.

Step 4. Return $POS_P(Q)$.

Step 1 of Algorithm 2 gets S_P and S_Q and its time complexity is $O(|A||U|)$. Time complexity of step 2 is also $O(|A||U|)$. Time complexity of step3 is $O(|U|)$. Therefor, time complexity of Algorithm 2 is $O(|A||U|)$.

According to Algorithm 2, we compute $POS_P(Q)$ of the following example. Let $\{\{u_1,u_5\},\{u_3,u_4\},\{u_2,u_8\},\{u_6\},\{u_7\}\}$ be U/P of a certain decision table. Identifier of every equivalence class is a,b,c,d,e respectively. Let $\{\{u_1,u_5,u_6\},\{u_3,u_4\},\{u_2,u_7\},\{u_8\}\}$ be U/Q of the decision table. Identifier of every equivalence class is 1,2,3,4 respectively. From PI and QI in S_P(as shown in Table **2**),we can judge that all the objects in $\{u_2,u_8\}$ do not have the same identifiers of equivalence classes of U/Q. So, they do not belong to $POS_P(Q)$. There- fore,$POS_P(Q)=\{u_1,u_5\}\cup\{u_3,u_4\}\cup\{u_6\}\cup\{u_7\}=\{u_1,u_3,u_4,u_5,u_6,u_7\}$.

Table 2. An example of algorithm 2.

U/P	PI	QI
u_1	a	1
u_5	a	1
u_3	b	2
u_4	b	2
u_2	c	3
u_8	c	4
u_6	d	1
u_7	e	3

If P=C and Q=D, Algorithm 2 can be used to compute $POS_C(D)$. In this case, every object's identifier is its value of decision attribute D, and time complexity is $O(|C||U|)$. Therefore, $POS_C(D)$ of Table **1** is $\{u_7,u_{14},u_2,u_8,u_{12},u_4,u_6,u_9,u_{15}\}$ computed by Algorithm 2.

5. AN INCREMENTAL ALGORITHM FOR FAST COMPUTING $POS_P(Q)$ BY THE ATTRIBUTES INCREASED

In the foregoing Algorithm 2, we compute U/P first, then obtain $POS_P(Q)$ on the basis of U/P. In further research, we find that $POS_P(Q)$ can be got step by step in the process of

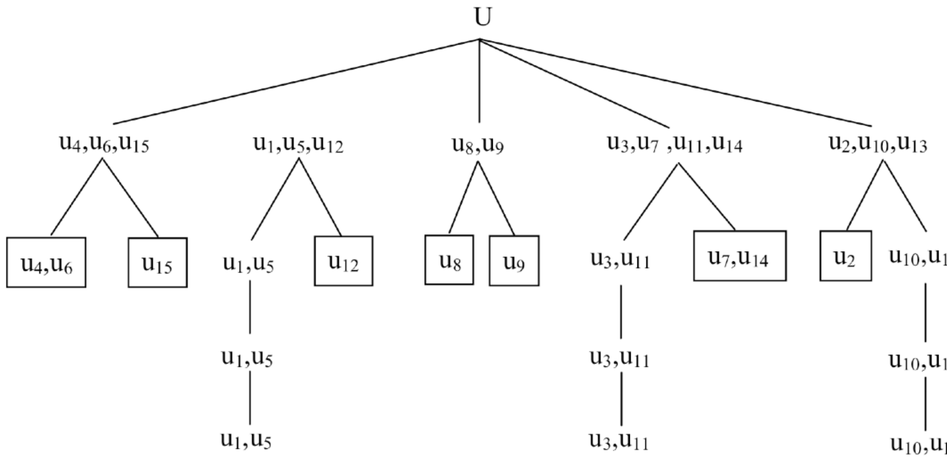

Fig. (1). An example of algorithm **3**.

computing U/P by the attributes increased. To explicate this idea, Theorem 5 is given first.

Theorem 5. In the process of computing U/P, 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 $POS_P(Q)$ definitely. The third kind need to be divided by remaining attributes to find whether there is a proper subset which belongs to $POS_P(Q)$ or not.

Proof. According to Definition 6, 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 $POS_P(Q)$. 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 $POS_P(Q)$. If the third kind of equivalence classes are divided by remaining attributes, some consistent equivalence classes may be produced and they belong to $POS_P(Q)$.

Now, we can present an increment algorithm for fast computing $POS_P(Q)$ by the attributes increased. In the process of dividing equivalence classes by every attribute in turn, the first kind and the second kind are added to $POS_P(Q)$ and removed from universe. Thus, universe is compressed gradually. Then universe is divided by next attribute. $POS_P(Q)$ is obtained after universe is divided by the last attribute.

The above algorithm uses multiway tree logic structure. Root node is U. Other node contains an equivalence class, its value of a_i and kind identifier. Nodes in the same level have the same attribute but have different values. Each leaf node either belongs to $POS_P(Q)$ or dosen't belong to $POS_P(Q)$ definitely.

It is noteworthy that universe is compressed gradually in the above computing process. So, computation may finish without dividing equivalence classes by last attributes. Therefore, it can shorten the time of calculation and upgrade efficiency.

Algorithm **3** may be described as follows.

Algorithm 3: Increment algorithm for computing $POS_P(Q)$ by the attributes increased

Input: S=(U,A,V,f) and $P,Q \subseteq A$.

Output: $POS_P(Q)$.

Step 1. $POS_P(Q)=\varnothing$, RootNode=U.

Step 2. for(i=1;i< k+1;i++)

If universe is not empty, divide equivalence class of each node by the i-th attribute and build the (i+1)-th level nodes. Then, divide equivalence class of each node by attributes in Q and mark kind identifier. If it is the first kind or the second kind of equivalence class, add it to $POS_P(Q)$ and remove it from universe.

Else, go to Step 3.

Step 3. Return $POS_P(Q)$.

In Algorithm **3**, let $R \subset P$. According to Algorithm 1, time complexity for computing $POS_P(Q)$ is $O((|P - R|+|Q|)|U - POS_R(Q)|)=O((|A||U - POS_R(Q)|)$ on the basis of $POS_R(Q)$. Obviously, it is more efficient than Algorithm 2. This is because Algorithm 3 makes full use of the existing $POS_R(Q)$ when it computes $POS_P(Q)$. So, the amount of calculation is reduced. In fact, the above time complexity is estimated for |P - R| attributes in one step. If time complexity is strictly estimated step by step according to Algorithm 3, it will be smaller. We give the following example which computes positive region of Table **1** to illustrate Algorithm 3 in Fig. (**1**). To make the Fig. (**1**) more distinct, values of a_i and kind identifiers are not marked out. First, root node U is build. Then, U is divided by attribute a and the second level is got. Afterwards, each node in the second level is divided by attribute b and the third level is got. Among them, $\{u_{15}\}$, $\{u_{12}\}$, $\{u_8\}$, $\{u_9\}$, $\{u_2\}$ are the first kind of equivalence classes and $\{u_4,u_6\}$, $\{u_7,u_{14}\}$ are the second kind of equivalence classes. They both belong to positive region. Remaining equivalence classes are divided by attribute c and d. Thus, the fourth level and the fifth level are got. But, there are no other objects which belong to positive region. Therefore, $POS_C(D)$ of Table **1** is $\{u_7, u_{14}, u_2, u_8, u_{12}, u_4, u_6, u_9, u_{15}\}$. The result is the same as the result which got by Algorithm 2. But, Algorithm 3 is more efficient.

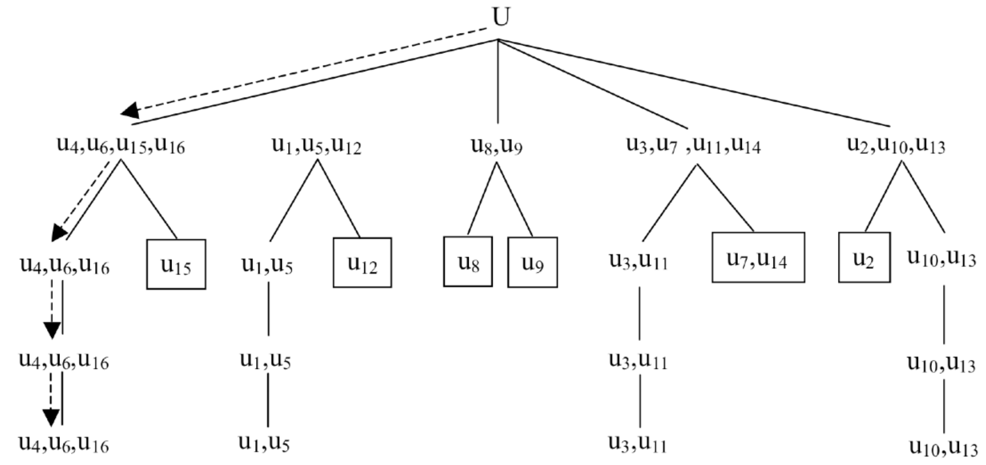

Fig. (2). An example of algorithm **4**.

6. AN INCREMENT ALGORITHM FOR FAST COMPUTING $POS_P(Q)$ BY THE OBJECTS INCREASED BASED ON MULTIWAY TREE

Further, we can put forward another increment algorithm for fast computing $POS_P(Q)$ by the objects increased which is based on Algorithm **3**.

Algorithm **4** may be described as follows.

Algorithm 4: Increment algorithm for computing $POS_P(Q)$ by the objects increased based on multiway tree

Input: S=(U,A,V,f), $P,Q \subseteq A$, multiway tree, $POS_P(Q)$, u.

Output: new $POS_P(Q)$.

Step 1. Search the multiway tree from root node by values of attributes for u.

If u dosen't belong to any leaf node, a new leaf node which contains u is build under current node and u is added to $POS_P(Q)$. Go to Step 3.

Else, go to Step 2.

Step 2. If the already existing objects in the leaf node are in an inconsistent equivalence class, u dosen't belong to $POS_P(Q)$. Go to Step 3.

If the already existing objects in the leaf node are in the first or the second kind of equivalence class, compare their values of attributes in Q with u.

Step 2.1. If they are the same, u is added to $POS_P(Q)$. Go to Step3.

Else, Algorithm 3 is used to divide the leaf node. Kind identifier and $POS_P(Q)$ are updated according to the final result.

Step 3. Return new $POS_P(Q)$.

In Algorithm **4**, calculation amount of Step 1 is $|U/a|+|U'/b|+|U''/c|+\ldots<|A||U/P|$, where U', U'' are nodes in search path. Calculation amount of Step 2 is $|U/a|+|U'/b|+|U''/c|+\ldots+|A||U_0|<|A||U/P|+|A||U_0|=|A|(|U/P|+|U_0|)$, where U_0 is the node which u belongs to. According to data sets from UCI Repository of Machine Learning Databases, including Haberman's Survival, mushroom, letter-recognition, $|U_0|<<|U/P|$ generally. So, calculation amount of Step 2 is far less than $O(|A|(|U/P|+|U_0|))=O(|A||U/P|)$. If P=C and Q=D, time complexity of Algorithm **4** is far less than $O(|C||U/C|)$. In addition, compared with Algorithm **3**, Algorithm 4 is more efficient. Let u belong to the j-th node of the i-th level. For Algorithm **4**, calculation amount for searching from the first level to the (i - 1)-th level is $|U/a|+|U'/b|+|U''/c|+\ldots$. Clearly, it is less than $|U|+|U|+|U|+\ldots$ of Algorithm **3**. Calculation amounts for the j-th node of the i-th level are the same for two algorithms. But Algorithm **3** needs compute other nodes of the i-th level. Therefor, Algorithm 4 gets higher efficiency.

For instance, object u_{16} is added to Table **1** in Fig. (**2**). Its values of attributes are {1, 4, 2, 1, 0}. Search path is shown as dotted line. Finally, because $\{u_4, u_6, u_{16}\}$ is a inconsistent equivalence class, u_4 and u_6 are deleted from $POS_P(Q)$. The updating is accomplished.

7. ATTRIBUTE REDUCTION ALGORITHM

The Algorithm **4** mentioned above can be used to compute attribute reduction directly. A attribute reduction algorithm based on incremental method is given as follows.

Algorithm 5: Attribute reduction algorithm based on incremental method

Input: $S=(U, C\cup D,V,f)$.

Output: a attribute reduction R.

Step 1. Let $R=\varnothing$.

Step 2. for(i=1;i<=k;i++)

Step 2.1. $C=C-\{a_i\}$;

Step 2.2. Compute $POS_C(D)$ and $POS_{C-\{ai\}}(D)$ according to the Algorithm **4**. If $POS_C(D)\neq POS_{C+\{R\}}(D)$, $R=R+\{a_i\}$.

Step 3 Return R.

Algorithm **5** judges whether a_i is necessary in turn. In fact, it can be done randomly. As we can see from the steps

Table 3. Comparison of four algorithms for computing $POS_C(D)$.

Decision Table	Number of Objects	Number of Condition Attributes	Number of Decision Attributes	Computational Time (ms)			
				A1	A2	A3	A4
Monkey	556	17	1	243.109	21.156	2.312	0.332
Balance	625	4	1	11.227	3.184	0.309	0.172
Cancer	683	9	1	223.171	12.597	1.460	0.207
Car	1728	6	1	1432.138	19.145	1.794	1.002

of Algorithm **5**, the algorithm consist of one layer of cycle and number of cycles should be not more than |C| times. The cost of every cycle is used to compute positive region. Because time complexity of Algorithm 4 is O(|C||U/C|), time complexity of Algorithm 5 is O($|C|^2$|U/C|) accordingly. For example, attribute reduction of Table **1** is {d} according to the Algorithm **5**.

8. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, to test the performances of the above-mentioned Algorithms, we use four decision tables from UCI Repository of Machine Learning Databases. All the experiments have been carried out on a personal computer with Windows 7, Intel Pentium(R) Dual CPU(1.86 GHz) and 2 GB memory.

First, we compare the time for computing $POS_C(D)$ of each decision table by traditional algorithm in [8], algorithm in [9], Algorithm **3** and Algorithm **4** in this paper. The above four algorithms are denoted by A1, A2, A3, A4 respectively. We randomly choose 85% of the objects from each decision table to be base decision tables. The remaining 15% are as new objects. Experimental results can be seen from Table **3**.

From Table **3**, we can see that Algorithm **3** and Algorithm **4** are more efficient than other algorithms. The more objects there are, the more efficient Algorithm **3** and Algorithm **4** are.

Then, Algorithm **3** and Algorithm **4** are compared when we add different number of new objects. We randomly choose 691 objects from car decision table to be the base decision table and randomly choose 173, 259, 346, 432, 519, 605, 691 objects in turn from the remaining 1037 objects to be increment. Experimental results can be seen from Fig. (**3**).

By Fig. (**3**), the computational time of two algorithms increases with the increasing of the size of data sets. When increment are less than 432, we can see that the computational time of Algorithm 4 is obviously faster than Algorithm 3. However, when increment are greater than 432, the gap between computational time of two algorithms narrows gradually. This experimental result shows that efficiency of Algorithm 4 decreases with the increasing of the size of incremental data sets. For this reason, time interval of the incremental algorithm for computing positive region should be relatively short in order to reduce the number of new objects and enhance the update efficiency.

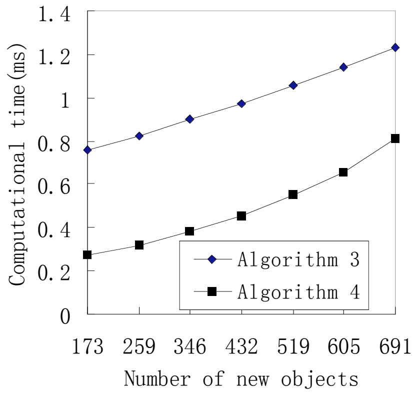

Fig. (3). Comparison of algorithm **3** and algorithm **4** when adding different number of new objects.

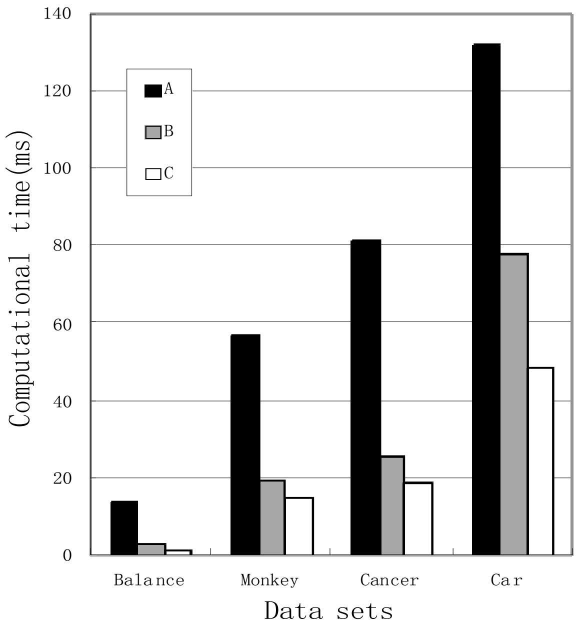

Fig. (4). Comparison of algorithm A, B and C.

To test the performances of Algorithms 5, we compare three attribute reduction algorithms by the datasets mentioned above. Algorithm in [6], algorithm in [7], Algorithm 5 in this paper are denoted by A, B, C respectively. Experimental results can be seen from Fig. (**4**).

From Fig. (**4**), we can see that Algorithm 5 is more efficient than other algorithms. The advantage is more obvious with datasets' enlargement.

9. CONCLUSION

In this paper, rough set theory is deeply researched first. Then aiming at the shortcomings of traditional algorithms for computing positive region and centering on the important concept of positive region, some new algorithms including incremental algorithms for fast computing positive region are proposed. Especially, the algorithm based on multiway tree is the most efficient. Its time complexity for computing $POS_C(D)$ is far less than $O(|C||U/C|)$. Then, a attribute reduction algorithm based on it is introduced and its time complexity is $O(|C|^2|U/C|)$. Theoretical analysis and experimental results indicate that it is superior to other traditional algorithms. As computation of attribute reduction is basic, further researches should consider other significant algorithms for rough set based on the work of this paper and its application to decision support combined with other uncertainty methods. Another interesting trend of research can be focused on application to pattern recognition.

CONFLICT OF INTEREST

The author confirms that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] Z. Pawlak, "Rough sets," *International Journal of Computer and Information Science*, vol. 11, no. 5, pp. 341-356, 1982.

[2] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data*, Dordrecht: Kluwer Academic Publishers, 1991.

[3] J. Jelonek, K. Krawiec, and R. Slowinski, "Rough set reduction of attributes and their domains for neural networks," *International Journal of Computational Intelligence*, vol. 11, no. 2, pp. 339-347, 1995.

[4] J. L. Du, Z. X. Chi, and W. Zhai, "An improved algorithm for reduction of knowledge based on significance of attribution," *Mini Micro System*, vol. 24, no. 6, pp. 976-978, 2003.

[5] W. J. Liu, Y. D. Gu, Y. B. Feng, and J. Y. Wang, "An improved attribute reduction algorithm of decisiontable," *Pattern Recognition and Arti- ficial Intelligence*, vol. 17, no. 1, pp. 119-123, 2004.

[6] S. H. Liu, Q. J. Sheng, and B. Wu, "Research on efficient algorithm for rough set method," *Chinese Journal of Computers*, vol. 26, no. 5, pp. 524-529, 2003.

[7] Z. Y. Xu, Z. P. Liu, and B. R. Yang, "A quick attribute reduction algorithm with complexity of max($O(|C||U|)$, $O(|C|^2|U/C|)$)," *Chinese Journal of Computers*, vol. 29, no. 3, pp. 391-399, 2006.

[8] G. Y. Wang, *Rough Set Theory and Knowledge Acquisition*, Xi'an: Xi'an Jiaotong University Press, 2001.

[9] Q. Liu, *Rough Set and Rough Reasoning*, Science Press, Beijing, 2001.

[10] D. Y. Ye, and Z. J. Chen, "Some properties of positive regions in Rough Set," *Journal of Fuzhou University*, vol. 30, no. 5, pp. 521-523, 2002.

[11] H. Ge, L. S. Li, and C. J. Yang, "Quick algorithm for computing core attribute," *Control and Decision*, vol. 24, no. 5, pp. 738-742, 2009.

[12] W. X. Zhang, W. Z. Wu, and J. Y. Liang, *Rough Set Theory And Methods,* Science Press, Beijing, 2001.

Received: June 10, 2015 Revised: July 29, 2015 Accepted: August 15, 2015