

Highly Scalable Rough Set Reducts Generation

PAI-CHOU WANG

*Department of International Business
Southern Taiwan University of Technology
Tainan, 710 Taiwan*

Rough set theory is used to represent, analyze, and manipulate knowledge in information or decision tables. To remove superfluous attributes without changing the original knowledge, reduction is must in rough set. This paper introduces the pseudo decision table to replace the original table and two algorithms, *RGonCRS* and *SRGonCRS*, based on the current rules size, *CRS*, are presented to generate all reducts which ensure the lower approximation for each instance in the table with a minimal number of attributes. *RGonCRS* finds reducts by merging candidate attributes and *SRGonCRS* is a scalable version of *RGonCRS* which generates reducts for very large tables. Propositions and proofs are presented in this paper. Empirical tests are shown for *RGonCRS* using simulated information tables and UCI benchmark datasets and a preliminary test is generated for *SRGonCRS*. Results are compared to the well-known rough set software – *Rough Set Exploration System (RSES)*.

Keywords: reducts generation, scalable reducts generation, information reduction, knowledge reduction, rough set theory

1. INTRODUCTION

In the scientific computation, information and decision systems are used to store information and they are usually perceived as tables where rows and columns represent instances and attributes respectively. In order to utilize knowledge in the table, researchers develop different discovering mechanisms and rough set theory is one of them. Rough set theory was developed by Zdzislaw Pawlak [1] in the early 1980's. It uses a strict mathematical formalism to represent, analyze, and manipulate knowledge in a table and Pawlak [2] has shown the knowledge within the table can be categorized into indiscernible relations. When we try to represent and manipulate knowledge in the indiscernible relations, there is a longstanding tradition in science that simple theories are preferable to complex ones. This is known as *Occam's razor* after the medieval philosopher William of Occam. In rough set, reducts [2] meet this purpose. Reducts ensure some classification properties with a minimal number of attributes in a table and researchers can use reducts to represent knowledge without any superfluous attributes in the table. In regarding of reducts generation, many kinds of reducts [3] can be generated based on different classification properties in the table. This paper focuses on generating all reducts which ensure the lower approximation of each instance in the table [2]. Sometimes they are called certain reducts [4] or global reducts [5]. In this paper, we will use reducts to represent them.

Many methods from previous literatures [2, 4-19] have been proposed to generate reducts and the most popular ones are based on the discernibility matrix [6]. They create

Received February 14, 2006; revised May 19, 2006; accepted August 8, 2006.

Communicated by Tei-Wei Kuo.

a discernibility matrix from the table and a boolean function called discernibility function is generated from the discernibility matrix. Function is reduced using absorption law then reducts are generated by finding prime implicants in the function. Any of reducts generation takes $O(kn^2)$ in time complexity and $O(kn^2)$ in space complexity where n is the number of instances and k is the number of attributes [7]. When table becomes large (e.g. with over 10,000 instances), the methods based on discernibility matrix are not feasible [7] because it contains too many candidates.

Table 1. A simple decision table.

U	a_1	a_2	a_3	$decision$
x_1	1	1	2	1
x_2	2	1	2	2
x_3	3	4	4	1
x_4	3	4	3	2
x_5	3	5	3	1
x_6	3	5	4	2

Nguyen *et al.* [7] proposed efficient algorithms to compute short reducts by greedily merging attributes based on the reduction of current indiscernibility and they take $O(k^2n \log n)$ in time complexity and $O(kn)$ in space complexity. The operation of greedily merging attributes generates semi-minimal reducts and operation is needed to remove irrelevant attributes from semi-minimal reducts. If we extend Nguyen's algorithms to generate all reducts, they will suffer from creating enormous semi-minimal reducts. The other difficulty comes from generating all reducts based on the reduction of current indiscernibility. In some cases, attributes are candidates of reducts but they can not reduce the current indiscernibility through merging the attributes. If we look at Table 1, attribute a_2 or a_3 can not reduce the current indiscernibility for the attribute a_1 but they are candidates of a_1 . This indicates we have to try attributes combinations even they can not reduce the current indiscernibility. This will increase a large amount of computation time.

To generate reducts for very large tables, Kryszkiewicz [4] proposed a scalable method based on the *MNSR*, *MaximalNonSuperReducts*. The *MNSR* contains all maximal indiscernible conditional attribute sets by comparing each pair of instances in the dataset and any subset of attributes can not be a reduct if it has a superset in *MNSR*. The creation of k attributes reducts first generates all possible candidates from $k - 1$ attributes *non-super-reducts* then it removes false candidates and moves candidates which are subset of *MNSR* to k attributes *non-super-reducts*. The author does not mention the time and space complexity but the *MNSR* is generated from all maximal indiscernible conditional attribute sets among instances in the dataset. We suspect the time and space complexity to generate any reducts of the table should not be less than $O(kn^2)$ and the author does mention the generation of *MNSR* is the bottleneck. The other question came from the *non-super-reducts*. *Non-super-reducts* set must be kept in order to keep candidates of reducts. The size of *non-super-reducts* could increase dramatically if the total number of attributes gets bigger and bigger.

In order to develop a better solution to generate all reducts for different size of tables with or without all consistencies, two algorithms *RGonCRS*, *Reducts Generation on*

Current Rules Size, and *SRGonCRS*, *Scalable Reducts Generation on Current Rules Size*, are developed. *RGonCRS* finds reducts recursively by merging candidate attributes based on propositions of the current rules size. The current rules size is computed from the current inconsistent rules and the number of inconsistent rules keeps decreasing when new attributes are merged. *RGonCRS* has the same time and space complexity as algorithms proposed by Nguyen *et al.* [7] but *RGonCRS* has stronger conditions for the candidate attributes. This makes *RGonCRS* outperforms previous methods. *SRGonCRS* is a scalable version of *RGonCRS*. *SRGonCRS* finds reducts incrementally by first dividing table into sub tables using graph partitioning [20] then it generates reducts incrementally from merging sub tables. There are reasons to develop *SRGonCRS*. *SRGonCRS* is developed to generate reducts for very large table to avoid the memory limit and this paper believes a lot of real life datasets contain limited number of reducts. The new merged table can find reducts quicker through previous generated reducts and we also believe this will improve the overall running time for very large table.

Empirical results using simulated information tables and datasets from UCI Machine Learning Repository are presented for *RGonCRS*. Results are compared to the well known rough sets software- *Rough Set Exploration System*, *RSES*, [21-23] which is created by the research team supervised by Dr. Andrzej Skowron. *RSES* is not the only system this paper considered. *RSES* is chosen because it has a better performance than the other system- *Rosetta* [25, 26] and it contains no attributes or instances limitation. Results from simulated information tables show the larger size of attributes or instances is and the better results can be achieved from *RGonCRS*. For UCI benchmark datasets, results show *RGonCRS* runs in an average of 17 times faster than *RSES* for datasets containing at least 1000 instances. A preliminary test of *SRGonCRS* using the dermatology dataset is generated. The dermatology dataset is selected because it contains the longest execution time using *RGonCRS*. *SRGonCRS* takes 21226.34 ± 122.89 seconds to generate all reducts and it shows 6.42 times faster than *RGonCRS*.

The layout of this paper is as follows: Section 2 reviews the basic preliminaries for rough set and introduces the pseudo decision table. Section 3 defines the current rules size, *CRS*, and presents the propositions, proofs, and the algorithm for *RGonCRS*. Empirical results for *RGonCRS* are shown in section 4 using simulated information tables and datasets from UCI Machine Learning Repository. Section 5 presents propositions and the algorithm for *SRGonCRS*.

2. BASIC PRELIMINARIES

2.1 Information Tables

Information system is usually perceived as an information table where rows and columns represent instances and attributes respectively and the information table T can be denoted as $T = (U, A)$ where

$U = \{x_1, \dots, x_n\}$ is a nonempty finite set of instances called the universe.

$A = \{a_1, \dots, a_k\}$ is a nonempty finite sets called attributes.

Every primitive attribute associates a function $a: U \rightarrow V_a$ for $a \in A$ where V_a is the

value set of a . In order to represent, analyze, and manipulate knowledge in an information table, Pawlak [1, 2] has shown the knowledge can be referred as equivalence relations. Let $T = (U, A)$ be an information table. With any subset $B \subseteq A$, an equivalence relation, denoted by $IND(B)$ called the B -indiscernibility relation can be defined as $IND(B) = \{(x, y) \in U^2 : \forall a \in B(a(x) = a(y))\}$. Instances x, y satisfying relation $IND(B)$ are indiscernible by attributes from B . Indiscernibility relations divide the information table into equivalence categories and these categories represent the knowledge in the table.

In regarding of classification using the knowledge from the indiscernibility relations, approximation [2] is introduced. If $T = (U, A)$ is an information table, $B \subseteq A$ is a set of attributes and $X \subseteq U$ is a set of instances. The B -lower and B -upper approximation of X in T can be denoted as $\underline{B}X = \{x \in U: [x]_{IND(B)} \subseteq X\}$ and $\overline{B}X = \{x \in U: [x]_{IND(B)} \cap X \neq \emptyset\}$. By $[x]_{IND(B)}$ we denote equivalence category of $IND(B)$ defined by x . B -lower approximation of X , $\underline{B}X$, contains the indiscernible relations definitely define X and B -upper approximation of X , $\overline{B}X$, contains the indiscernible relations possibly define X . Table 2 shows a simple information table which has 6 instances and three attributes. Let $B = \{Color, Size\}$. It divides Table 2 into four indiscernible categories which are $\{x_1\}$, $\{x_2, x_4\}$, $\{x_3, x_5\}$, and $\{x_6\}$. The knowledge in category $\{x_2, x_4\}$ shows the color is yellow and the size is medium and the B -lower and B -upper approximation for $X = \{x_1, x_3\}$ are $\{x_1\}$ and $\{x_1, x_3, x_5\}$.

Table 2. A simple information table.

U	$Color$	$Size$	$Feel$
x_1	Red	Small	Soft
x_2	Yellow	Medium	Hard
x_3	Green	Small	Hard
x_4	Yellow	Medium	Moderate
x_5	Green	Small	Hard
x_6	Green	Big	Moderate

2.2 Decision Tables

Decision systems appear in varieties of applications and they act which decisions should be undertaken when some conditions are met. A decision system can be treated as an information system with decisions and it is also perceived as a table. A decision table T can be denoted as $T = (U, C, D)$ where U is a nonempty finite set of instances and C and D are condition and decision attributes. Let $P \subseteq C$. With every instance $x \in U$ it associates a function $f_x^P: A \rightarrow V$, where $f_x^P(a) = a(x)$ for every $a \in P \cup D$. Function f_x^P is referred as a decision rule in table T . $f_x^P|C$ and $f_x^P|D$ are denoted as the conditions and decisions of f_x^P respectively. A rule f_x^P is said consistent if every $y \in U$ and $y \neq x$, $f_x^P|C = f_y^P|C$ implies $f_x^P|D = f_y^P|D$. Otherwise, the rule is inconsistent. Information table is a special all consistent decision table if we add unique decision class to each instance in the information table. In this paper, we will use decision tables to represent both information and decision tables.

Rule consistency shows the indiscernible conditions must lead to the same decisions

and instances in a decision table can be divided into two categories, consistent and inconsistent categories, based on it. The consistent category contains instances with consistent rules and the inconsistent category contains instances with contradictive rules. In rough set, we use the *positive region* to represent the consistent category in the table. The positive region, $POS_P(D)$, for $P \subseteq C$ in a decision table T contains instances which are certainly classified based on the condition attributes P and $POS_P(D)$ is the lower approximation of each instance using the condition attributes P in table T . Rules are the knowledge in a decision table and a special property can be introduced for rules.

Property 1 Let $T = (U, C, D)$ be a decision table, $x \in U$, $P \subseteq C$, and $Q \subseteq \{C - P\}$. If f_x^P or f_x^Q is consistent, $f_x^{P \cup Q}$ is consistent.

Proof: Proposition can be shown by proof of the contradiction. If we assume f_x^P or f_x^Q is consistent and $f_x^{P \cup Q}$ is inconsistent, this indicates it exists an instance y such that $f_x^{P \cup Q} | C = f_y^{P \cup Q} | C$ and $f_x^{P \cup Q} | D \neq f_y^{P \cup Q} | D$ based on the rule consistency. If we eliminate the equal part of P or Q from the condition attributes, we have $f_x^Q | C = f_y^Q | C$ and $f_x^Q | D \neq f_y^Q | D$ or $f_x^P | C = f_y^P | C$ and $f_x^P | D \neq f_y^P | D$. This shows f_x^P or f_x^Q is inconsistent and this contradicts to the original assumption. We can prove the proposition.

Property 1 shows the consistency of a rule holds or the rule could become consistent when we merge new condition attributes. It ignites the initial thought in this paper. This paper generates reducts by searching and merging candidate attributes. Instead of searching all instances in the table, this paper ignores the current consistent instances and only looks at the current inconsistent instances. After the introduction of information and decision tables, this paper presents the pseudo decision table.

Procedure *convPDT*(T)
 ‘ T is a decision table
 Sort table T based on condition attributes;
 IF (inconsistencies exist in table T) THEN
 Replace decision classes for all inconsistent relations by the Max(decision classes in consistent rules) + 1;
 END

2.3 Pseudo Decision Table

Pseudo decision table is an all consistent decision table and it is used to cover the reducts generation for decision tables with or without all consistencies by converting them into all consistent tables. Procedure *convPDT* is used to convert a decision table into a pseudo decision table. It keeps the original positive region from the decision table then replaces the decision classes for all inconsistent categories by a new decision class if inconsistencies exist. An inconsistent category contains instances with contradictive rules. The time and space complexity of procedure *convPDT* are $O(kn \log(n))$ and $O(kn)$ where n is the total number of instances and k is the total number attributes.

Table 3. A simple decision table.

U	c_1	c_2	c_3	d
x_1	1	1	1	0
x_2	2	1	1	0
x_3	2	2	2	0
x_4	2	1	1	1
x_5	3	1	3	1
x_6	1	1	1	1

Table 4. A pseudo decision table from Table 2.

U	c_1	c_2	c_3	d
x_1	2	2	2	0
x_2	3	1	3	1
x_3	1	1	1	2
x_4	2	1	1	2

Tables 3 and 4 show a simple example for converting a decision table to a pseudo decision table. Table 3 is an inconsistent decision table which contains two consistent instances $\{x_3, x_5\}$ and two inconsistent categories $\{x_1, x_6\}$ and $\{x_2, x_4\}$. Its pseudo decision table is shown in Table 4 by keeping the consistent instances $\{x_3, x_5\}$ then assigning the inconsistent categories $\{x_1, x_6\}$ and $\{x_2, x_4\}$ to a new decision class. When we use the pseudo decision table to replace original table for reducts generation, we have to prove all reducts come from the pseudo decision table are reducts in the original table and nothing is more or less. This will be shown in proposition 1 in next section.

2.4 Attributes Dispensability and Reducts

Attributes dispensability finds dispensable attributes and is used to remove superfluous attributes in a table. This operation matches the longstanding tradition in science that simple theories are preferable to complex ones and is known as *Occam's razor* after the medieval philosopher William of Occam. Different kinds of attributes dispensability [3] can be defined. In this paper, we consider reducts [2] which ensure the lower approximation of each instance in the table and attributes dispensability can be defined as:

Definition 1 Given a decision table $T = (U, C, D)$ and $r \in C$. An attribute r is dispensable in C if $POS_{C-\{r\}}(D) = POS_C(D)$.

Definition 1 shows an attribute r is dispensable in an attributes set C if the positive region obtaining from C remains unchanged after the attribute r is removed. In Table 3, attribute c_1 is dispensable in $\{c_1, c_2, c_3\}$ because the positive region $\{x_3, x_5\}$ remains unchanged after we remove the attribute c_1 from $\{c_1, c_2, c_3\}$. Applying the Definition 1, reducts which preserve the lower approximation of each instance in the table with a minimal number of attributes can be defined.

Definition 2 Given a decision table $T = (U, C, D)$ and $P \subseteq C$. We say P is a reduct of C if $POS_P(D) = POS_C(D)$ and $\forall r \in P, POS_{P-\{r\}}(D) \neq POS_C(D)$.

Definition 2 shows a reduct of the attributes set C is a minimal subset of C which preserves the positive region from C and no attribute is dispensable in the reduct. A table may contain many reducts. In Table 3, $\{c_1, c_2\}$ is a reduct of $\{c_1, c_2, c_3\}$ because we can not find fewer attributes from $\{c_1, c_2\}$ which preserves the positive region $\{x_3, x_5\}$ from $\{c_1, c_2, c_3\}$ and there are two reducts $\{c_1, c_2\}$, and $\{c_3\}$ in Table 3. If we intersect all reducts, a special subset of attributes called *Core* is generated. *Core* exists in all reducts

and it is must in all reducts. The removal of any attribute from *Core* will change the positive region in the table. Function *genCore* is used to find *Core* and the time and space complexities are $O(k^2n \log(n))$ and $O(kn)$ where n is the total number of instances and k is the total number attributes.

Function *genCore* is the first operation of generating reducts because *Core* exists in all reducts. Before we go to next section and present the reducts generation algorithm using the pseudo decision table, a proposition needs to be shown that reducts generated from the pseudo decision table are the reducts in the original decision table and nothing is more or less.

Function *genCore*(*T*)

'*T* is a pseudo decision table

FOR (each condition attribute r in table *T*) DO

 Remove r from table *T* and sort table *T* based on the rest of condition attributes;

 IF (inconsistencies exist in table *T*) THEN

 Attribute r is a part of *Core*;

 END

END

Proposition 1 $T = (U, C, D)$ is a decision table and $T' = (U', C, D)$ is the pseudo decision table of T . We say $P \subseteq C$ and P is a reduct of C in T' if and only if P is a reduct of C in T .

Proof: To prove reducts exist in both tables, we have to show reducts from table T preserve the positive region in T' with a minimal number of attributes and vice versa. It is easy to show reducts of T are reducts of T' . Any reduct of T preserves the positive region of T' because T' is a pseudo decision table of T . The removal of attributes in the reduct of T changes the positive region of T and it also changes the positive region of T' . This proves the first part of the proposition.

Reducts of T' also preserve the positive region of T . If attributes are removed from reducts of T' , inconsistencies occur for instances from the positive region of T or from the positive region of T and new decision class of T' . Both cases change the positive region of T . This shows reducts of T' are reducts of T . Proposition 1 is proved.

Proposition 1 shows reducts of the original table can be generated from the pseudo decision table. Next, this paper presents the *RGonCRS* – Reducts Generation on Current Rules Size.

3. *RGonCRS* – REDUCTS GENERATION ON CURRENT RULES SIZE

In this section, this paper first defines the current rules size, **CRS**, for any subset of condition attributes then propositions are proposed to find candidates of reducts using **CRS** in the table. At the end of this section, new reducts generation algorithm is presented.

Not like the most widely used reducts generation method [6], *RGonCRS* finds reducts recursively by merging candidate attributes based on propositions of the current rules size, *CRS*. *CRS* is introduced to ensure possible candidate attributes for reducts and *CRS* finds candidates with an aid of instances. The calculation of *CRS* for any subset of condition attributes P is defined as:

Definition 3 Given a pseudo decision table $T = (U, C, D)$ and a subset of condition attributes $P \subseteq C$. The current rules size of P , $CRS(P)$, can be derived from adding the size of the positive region, $POS_P(D)$, and the number of inconsistent categories based on P .

CRS shows the total number of rules obtaining from P and the inconsistent rules are counted as one. If we look back to Table 4 for $P = \{c_1\}$, the $CRS(P)$ is 3 by adding two consistent rules $\{x_2, x_3\}$ and one inconsistent category $\{x_1, x_4\}$. The size of the positive region can be computed by subtracting the total number of inconsistent instances from the table size and Property 1 in section 2.2 shows the consistency of a rule holds when we merge new condition attributes. *CRS* value can be computed simply based on the current inconsistent instances instead of going over all instances in the table. In order to apply the *CRS* to generate reducts, a new reduct definition is defined.

Definition 4 Let $T = (U, C, D)$ be a pseudo decision table and $P \subseteq C$. We say P is a reduct of C if $CRS(P) = |T|$ and $CRS(P - \{r\}) \neq |T|$ for every $r \in P$.

Definition 4 shows another interpretation of the Definition 2 in section 2.4. Pseudo decision table is an all consistent decision table and the size of its positive region, $POS_C(D)$, is equal to $|T|$. If $P \subseteq C$, $CRS(P) = |T|$ and $CRS(P - \{r\}) \neq |T|$ for every $r \in P$, it shows P preserves the positive region of T and no attribute in P is dispensable. P is a reduct of C in T . Definition 4 redefines reducts of C using *CRS* in a pseudo decision table and it acts as a main theme in *RGonCRS*. *RGonCRS* finds and merges candidate attributes until the *CRS* of the current attributes set is equal to the table size. In order to show reducts can be generated by the way of merging attributes, proposition is proposed to show the *CRS* value for a given subset of attributes could increase when we merge new condition attributes into it.

Proposition 2 Let $T = (U, C, D)$ be a pseudo decision table, $P \subseteq C$, and $r \in \{C - P\}$. When we merge $\{r\}$ into P , we have $CRS(P \cup \{r\}) \geq \max(CRS(P), CRS(\{r\}))$.

Proof: When we merge P and $\{r\}$, we could encounter equal or unequal size of the positive regions between P and $\{r\}$ like Figs. 1 and 2. Fig. 2 is a special case of Fig. 1 based on Property 1 in section 2.2. If we sort the inconsistent instances for P or $\{r\}$ which has the larger number of inconsistent categories and merge the inconsistent instances from the other, the new inconsistent categories in $P \cup \{r\}$ can not be less than the largest number of inconsistent categories from P or $\{r\}$. This proves the Proposition 2.

Proposition 2 shows reducts could be found through merging attributes. In order to eliminate false candidates, this paper presents the Proposition 3.

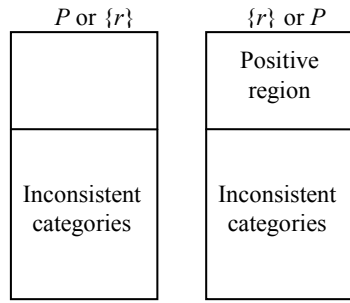


Fig. 1. Equal size of the positive region.

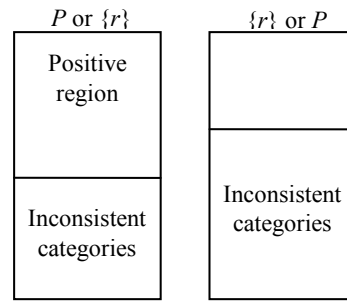


Fig. 2. Unequal size of the positive region.

Proposition 3 Let $T = (U, C, D)$ be a pseudo decision table and $P \subseteq C$. An attribute $r \in \{C - P\}$ can not be a candidate for reducts of C starting with P if $CRS(P \cup \{r\}) = CRS(P)$.

Proof: If there is a reduct starting with P and $\{r\}$, it means there exists a subset of attributes $S \subseteq \{C - P - \{r\}\}$ where instances in the inconsistent categories of $P \cup \{r\}$ will be converted to all consistent by merging S . If we have $CRS(P \cup \{r\}) = CRS(P)$, it shows all inconsistent categories using P remain unchanged in $P \cup \{r\}$ and this indicates the reduct starting with P and $\{r\}$ will still preserve the positive region of T when we remove the attribute r . This violates the definition of a reduct. Proposition 3 is proved.

Proposition 3 shows an attribute is excluded from reducts starting with a given subset of attributes if the CRS value keeps the same after we merge the attribute. Next, Proposition 4 is proposed to narrow the subsequent search for candidates.

Proposition 4 Let $T = (U, C, D)$ be a pseudo decision table and $P \subseteq C$. If Q is a set of candidate attributes derived from Proposition 3 for reducts of C starting with P and we merge a candidate $r \in Q$ into P , new candidate attributes for reducts starting with $P \cup \{r\}$ can be only found in $Q - \{r\}$.

The proof is trivial. If an attribute satisfies Proposition 3 for reducts starting with $P \cup \{r\}$, it should satisfy Proposition 3 for reducts starting with P because inconsistent categories in $P \cup \{r\}$ are also inconsistent in P . When reducts are generated simply based on Propositions 3 and 4, they will end up with semi-minimal reducts like [7]. In order to ensure the minimal criteria of reducts and stop the fake candidates in the middle of the operation, Proposition 5 is introduced.

Proposition 5 Let $T = (U, C, D)$ be a pseudo decision table and $P \subseteq C$. If attribute $r \in \{C - P\}$ satisfies Proposition 3 for reducts starting with P , we say r is not a candidate attribute if $\exists q \in P, CRS(\{P - \{q\}\} \cup \{r\}) = CRS(P \cup \{r\})$.

Proposition 5 is an extension of Proposition 3. If reducts contain $P \cup \{r\}$ and $\exists q \in P, CRS(\{P - \{q\}\} \cup \{r\}) = CRS(P \cup \{r\})$, this indicates attribute r is dispensable in reducts and this violates the definition of reducts. Proposition 5 enforces Proposition 3 and eliminates fake candidates in the middle of operation. The minimal criteria can be

ensured for reducts when attribute r satisfies Proposition 5 for $CRS(P) \neq |T|$ and $CRS(P \cup \{r\}) = |T|$. The operation of Proposition 5 does not generate a lot of computation because we can use information from previous candidate attributes to compute the new ones.

Function *findMerge*(*cList*, *candAttrs*)

' T' is a pseudo decision table

Find new candidates, *newCands*, for current attributes list, *cList*, from *CandAttrs* using proposition 3;

IF ($CRS(cList \cup newCands) \neq |T'|$) THEN

 Exit from the function;

END

Sort *newCands* based on CRS values in descending order;

FOR (each attribute r in *newCands*) DO

 IF (attribute r is a real candidate for *cList* based on proposition 5) THEN

 IF ($CRS(cList \cup \{r\}) = |T'|$) THEN

 Insert $cList \cup \{r\}$ into reducts set;

 ELSE

 Call *findMerge*($cList \cup \{r\}$, *newCands*- $\{r\}$);

 END

 END

END

Function *findMerge* uses previous propositions and finds reducts recursively by merging new candidates into current attributes list, *cList*, until the CRS value reaches table size $|T'|$. Operations of finding the CRS value for $cList \cup newCands \neq |T'|$ and sorting *newCands* based on CRS values are optimization processes. Function *findMerge* takes $O(k^2n \log n)$ in time complexity and $O(kn)$ in space complexity. The computation of this function will be getting shorter when the size of *cList* becomes larger. Next, algorithm of RGonCRS is presented. It starts converting the decision table to a pseudo decision table then finds *Core* using the function *genCore*. If CRS value of *Core* is equal to the table size, the table contains only one reduct, the *Core*. Otherwise, it sets the initial attributes list, *initList*, to *Core* and finds candidate attributes, *candAttrs*, for *initList*. Reducts are generating from merging attributes in *CandAttrs*. Any reducts of the table takes $O(k^2n \log n)$ in time complexity and $O(kn)$ in space complexity. Short reducts can also be found by applying greedy heuristics for candidate attributes with larger CRS value.

A simple example can be shown using Table 4 in section 2.3. There is no *Core* in Table 4 then the *initList* contains empty set and *candAttrs* contains $\{c_1, c_2, c_3\}$. $\{c_3\}$ is moved to reducts set because the $CRS(c_3)$ is equal to table size. Attribute c_2 is a candidate of attribute c_1 and $CRS(\{c_1, c_2\})$ is equal to the table size. $\{c_1, c_2\}$ is a reduct.

Algorithm *RGonCRS*

' $T = (U, C, D)$ is a decision table

$T' = ConvPDT(T)$;

Core = *genCore*(T');

```

IF ( $CRS(Core) = |T'|$ ) THEN
  Core is the only reduct;
ELSE
   $initList = Core$ ;
  IF isempty( $initList$ )
     $candAttrs$  contains all condition attributes;
  ELSE
    Find candidates,  $candAttrs$ , for  $initList$  using proposition 3 from  $\{C-Core\}$ ;
  END
  Remove attributes from  $candAttrs$  and insert them into reducts set if their  $CRS$  values
  are equal to  $|T|$  then sort  $candAttrs$  based on  $CRS$  values in descending order;
  While ( $CandAttrs$  is not empty) DO
    IF ( $CRS(initList \cup candAttrs) = |T'|$ ) THEN
      Let attribute  $r$  be the first attribute in  $CandAttrs$  and remove it from  $CandAttrs$ ;
      IF ( $CRS(initList \cup \{r\}) = |T'|$ ) THEN
        Insert  $initList \cup \{r\}$  into reducts set;
      ELSE
        Call  $findMerge(initList \cup \{r\}, candAttrs)$ ;
      END
    ELSE
      Exit from FOR loop;
    END
  END
END
END

```

4. EMPIRICAL RESULTS FROM *RGonCRS*

After propositions and algorithm are introduced, two parts of empirical tests are generated for *RGonCRS*. The first part contains tests for simulated information tables and the second part contains tests for decision tables come from UCI Machine Learning Repository and the traffic accident data in Taiwan, 2003. *RGonCRS* is implemented in an uncompiled Matlab code and tested in the Matlab 7. Results are compared to the well known rough set software – Rough Set Exploration System (*RSES*) [21-23] which is created by the research team supervised by Dr. Andrzej Skowron. *RSES* [5, 24] generates reducts by searching prime implicants in the discernibility function generated from the discernibility matrix. The search of prime implicants is only based on attributes operations and the minimal criteria for reducts come from prime implicants need to be verified. *RSES* is not the only system this paper considered. *RSES* is chosen because it has a better performance than another system – *Rosetta* [25, 26] and it contains no attributes or instances limitation. All tests are executed three times if the running time is less than 24 hours. Otherwise, it is executed once. The running time of *RGonCRS* covers all operations in the algorithm. A Windows 2003 based PC with a Pentium 4 2.8MHz CPU and 2G main memory is used for all tests.

Table 5 contains results for simulated information tables which are generated based on the design from Starzyk *et al.* [17, 27]. Tables contain attributes for 20, 25, and 30 and the number of instances varied from 50 to 150 in increment of 25 for each attribute. The value in each instance is an integer between 0 and 8 which is generated using a

Table 5. Test results for tables using uniformly random number generator.

Attributes	Instances	Reducts	<i>RGonCRS</i> (seconds)	<i>RSES</i> (seconds)
20	50	1730	8.85 ± 0.07	31.42 ± 0.24
20	75	2978	17.84 ± 0.06	256.92 ± 2.95
20	100	2730	25.75 ± 0.12	416.9 ± 5.28
20	125	2820	33.79 ± 0.14	680.98 ± 7.43
20	150	4795	51.88 ± 0.06	1261.06 ± 5.25
25	50	4180	20.05 ± 0.03	591.3 ± 18.5
25	75	7697	44.22 ± 0.12	3285.39 ± 83.9
25	100	7334	64.07 ± 0.15	8057.67 ± 64.93
25	125	9249	97.25 ± 0.1	18011 ± 407
25	150	13786	138.76 ± 0.08	36641.67 ± 381.16
30	50	9842	45.59 ± 0.1	3599.84 ± 62.82
30	75	15960	94.26 ± 0.16	20544.33 ± 24.85
30	100	17621	152.84 ± 0.35	113384
30	125	21908	220.09 ± 1.74	364085
30	150	39504	344.26 ± 0.1	*

Table 6. Test results for tables using normally random number generator ($\mu = 3$, $\sigma = 0.77$).

Attributes	Instances	Reducts	<i>RGonCRS</i> (seconds)	<i>RSES</i> (seconds)
20	50	10109	187.56 ± 0.83	10974 ± 68.2
25	50	48240	891.38 ± 1.1	*
30	50	191403	3483.5 ± 18.01	*

uniformly random number generator. The first three columns in Table 5 show total number of attributes, total number of instances, and total number of reducts in each information table and the last two columns contain the running time in seconds for *RGonCRS* and *RSES*. From Table 5, we see *RGonCRS* consistently outperforms *RSES* as the number of instances or attributes increase and an asterisk in *RSES* indicates we stop execution for no result over a week. This paper believes better results come from the aid of *CRS* which excludes false and fake candidates during reducts generation. In order to strengthen the belief, three additional tables based on normal distribution are generated. Normal distribution is applied because a lot of social behaviors are modeled by the normal distribution and this paper wants to see if similar improvements can be obtained from these tables. Each table contains 50 instances and each instance contains an integer between 0 and 5. Table 6 contains the results and *RGonCRS* still outperforms *RSES*. Two asterisks in *RSES* still indicate no results for over a week.

The second part of tests contains 19 decision tables and each table contains one decision attribute. Eighteen datasets are drawn from UCI Machine Learning Repository and the last dataset comes from the traffic accident data in Taiwan 2003 except three major cities Taipei, Taichung, and Kaohsiung. Instances with missing data are removed and all values are integers. Table 7 shows results for *RGonCRS* and *RSES* and *RGonCRS*

Table 7. Test results from decision tables.

Datasets	Attributes	Instances	Reducts	<i>RGonCRS</i> (seconds)	<i>RSES</i> (seconds)
Adult (train)	15	20162	2	157.01 ± 0.14	2010.85 ± 2.7
Australian (Statlog)	15	690	44	0.81 ± 0.01	2.9 ± 0.01
Balance Scale	5	625	1	0.07 ± 0.01	1.22 ± 0.02
Wisconsin Breast Cancer	10	699	20	0.41 ± 0.00	2.04 ± 0.07
Car Evaluation	7	1728	1	0.29 ± 0.01	5.8 ± 0.12
Chess King Rook vs King	7	28056	1	64.23 ± 0.04	1913.24 ± 2.41
Chess King Rook vs King Pawn	37	3196	4	5.21 ± 0.06	101.14 ± 0.85
Contraceptive Method Choice	10	1473	1	0.36 ± 0.01	5.92 ± 0.02
Dermatology	35	358	112708	136297.82	*
German (Statlog)	21	1000	846	104.98 ± 2.5	176.42 ± 0.27
Heart (Statlog)	14	270	109	3.05 ± 0.04	1.67 ± 0.03
Letter Recognition	17	20000	61	143.79 ± 1.02	3087 ± 72.44
Liver Disorder	7	345	9	0.17 ± 0.01	0.82 ± 0.06
Pima Indians Diabetes	9	768	28	0.89 ± 0.02	2.27 ± 0.03
Pen Based Recognition (train)	17	7494	2539	581.6 ± 1.73	6492.19 ± 48.32
Shuttle Control (train, Statlog)	10	43500	19	242.37 ± 0.66	5024.41 ± 24.89
SPECT (train)	23	80	70	1.75 ± 0.01	0.72 ± 0.06
Zoo	17	101	33	0.76 ± 0.01	0.72 ± 0.03
Traffic Accident Data (Taiwan, 2003)	23	171539	1	6570.1 ± 42.76	137363

still outperforms *RSES* in most cases. If we look at datasets which contains at least 1000 instances, the running time of *RGonCRS* achieves an average of 17 times faster than *RSES*. In dermatology dataset, Table 7 shows an asterisk in *RSES* where *RSES* shows not enough of memory and aborts the execution after it ran for almost a day. From both parts of tests, we know *RGonCRS* generates all reducts more efficiently than *RSES*. In the case of very large tables and limited memory, a scalable version of *RGonCRS* is introduced in the next section.

5. *SRGonCRS* – SCALABLE REDUCTS GENERATION ON CURRENT RULES SIZE

When tables become very large and there is limited memory, previous methods [2, 4-19] will have difficulty to generate all reducts either from keeping the discernibility matrix [6] or computing the *MNSR* [4], *MaximalNonSuperReducts*. *SRGonCRS*, *Scalable Reducts Generation on Current Rules Size*, is a scalable version of *RGonCRS* and it generates reducts incrementally. *SRGonCRS* is developed to generate reducts for very large table to avoid the memory limit and this paper believes a lot of real life datasets contain limited number of reducts. Reducts of the new merged table can be found quickly through previous generated reducts and we believe this will improve the overall running time for very large table.

Function *graphPartition(T)*

' $T = (U, C, D)$ is a pseudo decision table
Construct a $n \times n$ *edgeWeight* matrix where n is total number of instances;
subTables = \emptyset ;
WHILE ($|T| > threshold$)
 $[T_{left}, T_{right}] = 2\text{-way-graph-partitioning}(T)$;
 Insert T_{left} and T_{right} into subTables;
 IF ($|T_{left}| \leq |T_{right}|$) **THEN**
 $T = T_{left}$;
 Insert T_{right} into subTables;
 ELSE
 $T = T_{right}$;
 Insert T_{left} into subTables;
 END
END
Insert T into subTables;

The algorithm of *SRGonCRS* first divides the pseudo decision table into sub tables. Instead of dividing the table randomly, this paper applies a 2-way graph partitioning algorithm [20] to divide the table. The 2-way graph partitioning algorithm divides all instances in the table into two groups and the edge weights between two groups are minimized. Skowron *et al.* [6] showed reducts must have a non-empty intersection with elements in the discernibility matrix. In this paper, edge weight between instances is defined as square of the number of indiscernible attributes between them. Edge weight is defined like that because reducts generation is executed in a single machine and we hope reducts from a sub table could be reducts in the merged one.

Algorithm *SRGonCRS*

' $T = (U, C, D)$ is a decision table
 $T' = ConvPDT(T)$;
 $Core = genCore(T')$;
IF ($CRS(Core) = |T'|$) **THEN**
 $Core$ is the only reduct;
ELSE
 Generate sub tables using *graphPartition(T')* and move instances related to $Core$ to the first sub table, T_1 ;
 Generate all reducts, R , of T_1 using *RGonCRS*;
 WHILE (still have sub tables)
 Merge sub tables into T_1 ;
 $R_{new} = \emptyset$;
 FOR ($r \in R$) **DO**
 Generate reducts of T_1 using *RGonCRS* based on r and insert reducts into R_{new} ;
 END
 $R = R_{new}$;
 END
END

Function *graphPartition* is used to split the table and it generates sub tables. The sub table with smaller size is split recursively until the size of the sub table reaches a threshold. Threshold is used to prevent too many reducts generating from the first sub table. After sub tables are created, reducts of the smallest sub table is generated using *RGonCRS* then new table is created from merging sub tables and reducts are generated incrementally. Reducts generated from searching previous ones could encounter a problem where redundant searches could occur and they will waste a lot of computation time. This can be prevented by verifying previously applied reducts from the current one. Next, algorithm of *SRGonCRS* is introduced.

A preliminary test of *SRGonCRS* using the dermatology dataset is generated. The dermatology dataset is selected because it takes the longest execution time using *RGonCRS*. Dataset is split 4 times and the first sub table contains 20 instances. The first sub table generates 6,049 reducts and we generate all reducts by merging the rest of sub tables. The total running time does not include the time taken in the graph partitioning because it takes less than 60 seconds. Results are generated in the same platform like *RGonCRS* and they are executed for three times. *SRGonCRS* takes 21226.34 ± 122.89 seconds to generate all reducts and it shows 6.42 times faster than *RGonCRS*. The improvement can not be always fit into all datasets because overhead occurs for small datasets. After we present the *SRGonCRS*, proposition needs to show reducts from the original table can be generated incrementally and we will also explain why *SRGonCRS* can generate all reducts to avoid the memory limit.

Proposition 6 Let $T = (U, C, D)$, $T' = (U', C, D)$ be consistent decision tables and no inconsistencies occur when we merge T and T' . All reducts of C in $T \cup T'$ can be derived from the superset of all reducts of C in T or T' .

Proposition 6 shows all reducts of C in $T \cup T'$ can be derived from the superset of all reducts of C in T or T' . The proof is trivial. All reducts of C in T or T' preserve the positive region with a minimal number of attributes and all reducts of C in $T \cup T'$ also preserve the positive region of T and T' because no inconsistencies occur when we merge T and T' . This shows Proposition 6.

From *SRGonCRS*, reducts are generated incrementally from merged tables. Let $P \subseteq C$. If we merge two sub tables T and T' , the maximal number of inconsistent categories using P occurs when each instance in one table is contradictive to instances in the other table. This shows the total number of inconsistent categories can not be greater than $\min(|T|, |T'|)$ using P in $T \cup T'$. If we carefully select the size of sub table which is less than the memory limit in the machine, *SRGonCRS* can generate all reducts for very large table without the memory limit.

6. CONCLUSION

This paper presents two algorithms, *RGonCRS* and *SRGonCRS*, to generate all reducts based on the current rules size, *CRS*. Reducts which preserve the lower approximation of each instance in the table with a minimal number of attribute are consider in this paper. *RGonCRS* finds reducts by merging candidate attributes and *SRGonCRS* is a

scalable version of *RGonCRS* which finds reducts for very large tables. Propositions and proofs are shown in this paper. Empirical results are presented for *RGonCRS*. Simulated information tables and datasets from UCI Machine Learning Repository are tested and results are compared to the well-known rough set software – Rough Set Exploration System (*RSES*). *RGonCRS* consistently outperforms *RSES*. When the table becomes very large or the computation of *MNSR* [4] is infeasible, *SRGonCRS* is an extension of *RGonCRS* for very large table and it generates reducts incrementally by first dividing table into sub tables using graph partitioning and generating reducts incrementally from merging each sub table. A preliminary test is generated for *SRGonCRS* using the dataset with longest running time by *RGonCRS* and it shows 6.42 times faster than *RGonCRS*. In order to apply *SRGonCRS* to datasets with large table size or large attributes, the sub table size and the number of merged sub tables still need to pay more attention.

REFERENCES

1. Z. Pawlak, "Rough sets," *International Journal of Computer and Information Sciences*, Vol. 11, 1982, pp. 341-356.
2. Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer, Netherlands, 1991.
3. M. Kryszkiewicz, "Comparative study of alternative types of knowledge reduction in inconsistent systems," *International Journal of Intelligent Systems*, Vol. 16, 2001, pp. 105-120.
4. M. Kryszkiewicz and K. Cichon, "Toward scalable algorithms for discovering rough set reducts," in J. F. Peters, A. Skowron, J. W. Grzymala-Busse, B. Kostek, R. W. Swiniarski, and M. S. Szczuka, eds., *Transactions on Rough Sets I*, Springer, New York, 2004, pp. 120-143.
5. J. Bazan, H. S. Nguyen, S. H. Nguyen, P. Synak, and J. Wroblewski, "Rough set algorithms in classification problems," in L. Polkowski, S. Tsumoto, and T. Y. Lin, eds., *Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems*, Physical-Verlag, New York, 2000, pp. 49-88.
6. A. Skowron and C. Rauszer, "The discernibility matrices and functions in information systems," *Fundamenta Informaticae*, Vol. 15, 1991, pp. 331-362.
7. S. H. Nguyen and H. S. Nguyen, "Some efficient algorithms for rough set methods," in *Proceedings of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 1996, pp. 1451-1456.
8. A. Skowron and L. Polkowski, "Decision algorithms: a survey of rough set theoretic methods," *Fundamenta Informaticae*, Vol. 30, 1997, pp. 345-358.
9. A. Skowron and L. Polkowski, "Synthesis of decision systems from data tables," in T. Y. Lin, N. Cercone, eds., *Rough Sets and Data Mining, Analysis of Imprecise Data*, Kluwer, 1997, pp. 259-299.
10. J. Bazan, A. Skowron, and P. Synak, "Dynamic reducts as a tool for extracting laws from decision tables," in *Proceedings of International Symposium on Methodologies for Intelligent Systems*, 1994, pp. 346-355.
11. J. G. Bazan, "A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision table," in L. Polkowski and A. Skowron, eds., *Rough*

- Sets in Knowledge Discovery*, Physica-Verlag, Heidelberg, 1998, pp. 321-365.
12. D. Slezak, "Searching for frequential reducts in decision tables with uncertain objects," in *Proceedings of 1st International Conference on Rough Sets and Current Trends in Computing*, 1998, pp. 52-59.
 13. R. Slowinski, *Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory*, Kluwer Academic Publisher, 1992.
 14. S. Greco, B. Matarazzo, and B. Slowinski, "A new rough set approach to multicriteria and multiattribute classification," in *Proceedings of 1st International Conference on Rough Sets and Current Trends in Computing*, 1998, pp. 60-67.
 15. J. Stepaniuk, "Approximate spaces, reducts and representatives," in L. Polkowski, J. Kacprzyk, A. Skowron, eds., *Rough Sets in Knowledge Discovery 2: Applications, Case Studies, and Software Systems*, Physica-Verlag, Heidelberg, 1998, pp. 109-126.
 16. R. Susmaga, "Parallel computation of reducts," in *Proceedings of 1st International Conference on Rough Sets and Current Trends in Computing*, 1998, pp. 450-457.
 17. J. A. Starzyk, D. E. Nelson, and K. Sturtz, "A mathematical foundation for improved reduct generation in information systems," *Knowledge and Information Systems*, Vol. 2, 2000, pp. 131-146.
 18. M. Zhang, L. D. Xu, W. X. Zhang, and H. Z. Li, "A rough set approach to knowledge reduction based on inclusion degree and evidence reasoning theory," *Expert Systems*, Vol. 20, 2003, pp. 298-304.
 19. J. Wang and J. Wang, "Reduction algorithms based on discernibility matrix: the ordered attributes method," *Journal Computer Science & Technology*, Vol. 16, 2001, pp. 489-504.
 20. G. Karypis and V. Kumar, "A fast and highly quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, Vol. 20, 1999, pp. 359-392.
 21. J. G. Bazan and M. S. Szczuka, "RSES and RSESLib – a collection of tools for rough set computations," in *Proceedings of 2nd International Conference Rough Sets and Current Trends in Computing*, 2000, pp. 106-113.
 22. J. Bazan, M. S. Szczuka, and J. Wróblewski, "A new version of rough set exploration system," in *Proceedings of 3rd International Conference on Rough Sets and Current Trends in Computing*, 2002, pp. 397-404.
 23. A. Skowron, J. Bazan, M. S. Szczuka, and J. Wróblewski, *Rough Set Exploration System (version 2.2.1)*, <http://logic.mimuw.edu.pl/~rses/>, Access 25 June 2005.
 24. A. Skowron, J. Bazan, M. S. Szczuka, and J. Wróblewski, *RSES 2.2 Guide*, <http://logic.mimuw.edu.pl/~rses/>, Access 25 June 2005.
 25. J. Komorowski, A. Ohrn, and A. Skowron, *Handbook of Data Mining and Knowledge Discovery: The ROSETTA Rough Set Software System*, Oxford University Press, 2002, Ch. D.2.3.
 26. J. Komorowski, A. Ohrn, and A. Skowron, *Rosetta Software System (Version 1.4.41)*, <http://rosetta.lcb.uu.se/general/>, Access 25 June 2005.
 27. D. E. Nelson, "High range resolution radar target classification: a rough set approach," Ph.D. dissertation, School of Electrical Engineering and Computer Science, Ohio University, U.S.A., 2001.



Pai-Chou Wang (王派洲) received his Ph.D. and M.S. degrees in Computer and Information Science from Polytechnic University, New York, U.S.A., in 1996 and 1991. He is an associate professor of the Department of International Business in Southern Taiwan University of Technology, Tainan, Taiwan, since 1996. His research interests include distributed systems, genetic algorithms, and data mining.