On Reduct Construction Algorithms

Yiyu Yao[1], Yan Zhao[1], and Jue Wang[2]

[1] Department of Computer Science, University of Regina
Regina, Saskatchewan, Canada S4S 0A2
{yyao, yanzhao}@cs.uregina.ca

[2] Laboratory of Complex Systems and Intelligence Science, Institute of Automation
Chinese Academy of Sciences, Beijing, China 100080
jue.wang@mail.ia.ac.cn

Abstract. This paper critically analyzes reduct construction methods at two levels. At a high level, one can abstract commonalities from the existing algorithms, and classify them into three basic groups based on the underlying control structures. At a low level, by adopting different heuristics or fitness functions for attribute selection, one is able to derive most of the existing algorithms. The analysis brings new insights into the problem of reduct construction, and provides guidelines for the design of new algorithms.

Keywords: Reduct construction algorithms, deletion strategy, addition-deletion strategy, addition strategy, attribute selection heuristics.

1 Introduction

The theory of rough sets has been applied to data analysis, data mining and knowledge discovery. A fundamental notion supporting such applications is the concept of reducts, which has been studied extensively by many authors [14, 17, 21, 22, 25, 29, 30]. A reduct is a subset of attributes that is jointly sufficient and individually necessary for preserving the same information as provided by the entire set of attributes. It has been proved that finding a reduct with the minimal number of attributes is NP-hard [26]. Research efforts on reduct construction algorithms therefore mainly focus on designing search strategies and heuristics for finding a satisfactory reduct efficiently.

A review of the existing reduct construction algorithms shows that most of them tie together search strategies (i.e., control structures) and attribute selection heuristics. This leads to difficulties in analyzing, comparing, and classifying those algorithms, as well as the trend of introducing new algorithms constantly. With ample research results on this topic, it is perhaps the time for us to pause and to analyze critically those results, in order to gain more insights.

With a clear separation of control structures and attribute selection heuristics, we can critically analyze reduct construction algorithms with respect to the high level control strategies, and the low level attribute selection heuristics, respectively. This allows us to conclude that the differences between the existing algorithms lie more on the attribute selection heuristics than on the control strategies.

The rest of the paper is organized as follows. First of all, we discuss the connections between feature selection and reduct construction in Section 2. After that, basic concepts and notations of rough set theory are reviewed in Section 3. Three basic control structures are then presented in Section 4-6 by reformulating the existing algorithms, from which many variations can be generated easily. After these, Section 7 is the conclusion.

2 Feature Selection and Reduct Construction

Reduct computation is related to many disciplines. The same objective of simplifying the attribute domain has been studied in machine learning, pattern recognition, and feature selection in specific [3, 12, 13, 23].

Feature selection is a fundamental task in a number of different disciplines, such as pattern recognition, machine learning, concept learning and data mining. Feature selection is necessary for both description and prediction purposes. In the description process, it can be computationally complex to construct rules by using all available features; in the prediction process, the constructed high dimensional rules can be hard to test and evaluate for new coming instances. From a conceptual perspective, selection of relevant features, and elimination of irrelevant ones, are the main tasks of feature selection. From a theoretical perspective, it can be shown that an optimal feature selection requires an exhaustive search of all possible subsets of the entire feature set. If the cardinality of the entire feature set is large, this exhaustive method is impractical. For practical feature selection applications, the search is normally for a satisfactory set of features instead of an optimal set.

In the domain of feature selection, two methods, *forward selection* and *backward elimination*, have been extensively studied [3,12,13]. The forward selection strategy starts with the empty set and consecutively adds one attribute at a time until we obtain a satisfactory set of features. This strategy also can be called as an *addition* strategy for simplicity. On the contrary, the backward elimination strategy starts with the full set and consecutively deletes one attribute at a time until we obtain a satisfactory set of features. In this paper, this strategy also is called a *deletion* strategy. The forward strategy can be extended from the one-by-one sequential-add style to the "plus l - take away r" style. This kind of methods first enlarge the feature subset by l features, then delete r features as long as the remaining attribute set exhibits an improvement compared to the previous feature set. They avoid the nesting problem of feature subsets that are encountered in the sequential style, but need to set the values of l and r [4, 18]. The same idea can be applied to backward strategy variations.

In a consecutive forward selection or a backward elimination process, one can adopt different heuristics for feature selection. A heuristic decides and then adds the best feature, or deletes the worst feature, at each round. As a consequence, variations of the same algorithm can be derived.

The difference between reduct computation and feature selection is their halting strategies. For the purpose of feature selection, one might stop adding or deleting features when the information preservation is satisfied, the classification accuracy is not degrading, or the computation cost is affordable. For reduct construction, the algorithm does not stop until the *minimum* set of features that possesses some particular property is obtained. Reduct construction thus is a special case of feature selection. In fact, many feature selection algorithms can be viewed as performing a biased form of reduct computation. The results are not necessarily being reducts. Obviously, the extensive studies of feature selection, including the identification of relevant, irrelevant and redundant features, and the design, implementation and renovation of the filter and wrapper methods, affect the study of reduct computation.

By considering the properties of reducts, the deletion strategy always results in a reduct [7, 30]. On the other hand, algorithms based on a straightforward application of the addition strategy only produce a superset of a reduct [8, 10, 15, 16, 20]. In order to resolve this problem, many authors have considered a combined strategy by re-applying the deletion strategy on the superset of the reduct produced by the straightforward addition strategy [25]. An interesting question is whether there exists an addition-only strategy that can produce a reduct. A positive answer has been given by Zhao and Wang with an addition algorithm without further deletion [29].

According to the above discussion, we have three control strategies used by reduct construction algorithms. They are the deletion strategy, the addition-deletion strategy, and the addition strategy. We can classify reduct construction algorithms into the corresponding three groups.

3 Basic Concepts and Notations

The basic concepts, notations, and results related to the problem of reduct construction are briefly reviewed in this section.

3.1 Information Table and Attribute Lattice

Suppose data are represented by an information table, where a finite set of objects are described by a finite set of attributes [17].

Definition 1. *An information table S is the tuple:*

$$S = (U, At, \{V_a \mid a \in At\}, \{I_a \mid a \in At\}),$$

where U is a finite nonempty set of objects, At is a finite nonempty set of attributes, V_a is a nonempty set of values for an attribute $a \in At$, and $I_a : U \rightarrow V_a$ is an information function. For an object $x \in U$, an attribute $a \in At$, and a value $v \in V_a$, $I_a(x) = v$ means that the object x has the value v on attribute a.

Table 1. An information table

	a	b	c	d	e
o_1	0	0	0	1	1
o_2	0	1	2	0	0
o_3	0	1	1	1	0
o_4	1	2	0	0	1
o_5	0	2	2	1	0
o_6	0	3	1	0	2
o_7	0	3	1	1	1

Example 1. An information table is illustrated in Table 1, which has five attributes and seven objects.

The family of all attribute sets form an attribute lattice under the refinement order. Let $|At|$ denote the cardinality of At. An attribute lattice has $|At| + 1$ levels. The only node on the top level indicates the empty set $\emptyset$. The only node on the bottom level indicates the biggest attribute set At. Nodes on the second level stand for singleton attribute sets. There are $|At|$ nodes in the second level. For the n^{th} level and $n \geq 2$, there are

$$\frac{|At|(|At|-1)\ldots(|At|-n+2)}{(n-1)!}$$

nodes. There are $2^{|At|}$ attribute sets in the entire attribute lattice. An edge connecting a pair of nodes implies the refinement relationship between an attribute set and a subset or superset of the attribute set.

Example 2. Figure 1 illustrates the attribute lattice of the previous information Table 1. It is obvious that totally $2^5 = 32$ attribute sets can be defined for the universe.

3.2 Equivalence Relations

Definition 2. *Given an information table S, for any subset $A \subseteq At$ there is an associated equivalence relation $E_A \subseteq U \times U$, i.e.,*

$$E_A = \{(x, y) \in U \times U \mid \forall a \in A, I_a(x) = I_a(y)\},$$

which partitions U into disjoint subsets, called equivalence classes. An equivalence class containing any object $x \in U$ is defined as: $[x]_A = \{y \in U \mid \forall a \in A, I_a(x) = I_a(y)\}$. Such a partition of the universe is denoted by U/E_A, or U/A for simplicity.

A partition U/E_A is a refinement of another partition U/E_B, or equivalently, U/E_B is a coarsening of U/E_A, denoted by $U/E_A \preceq U/E_B$, if every equivalence class of U/E_A is contained in some equivalence class of U/E_B. The refinement relation is a partial order, i.e, it is reflexive, anti-symmetric and transitive.

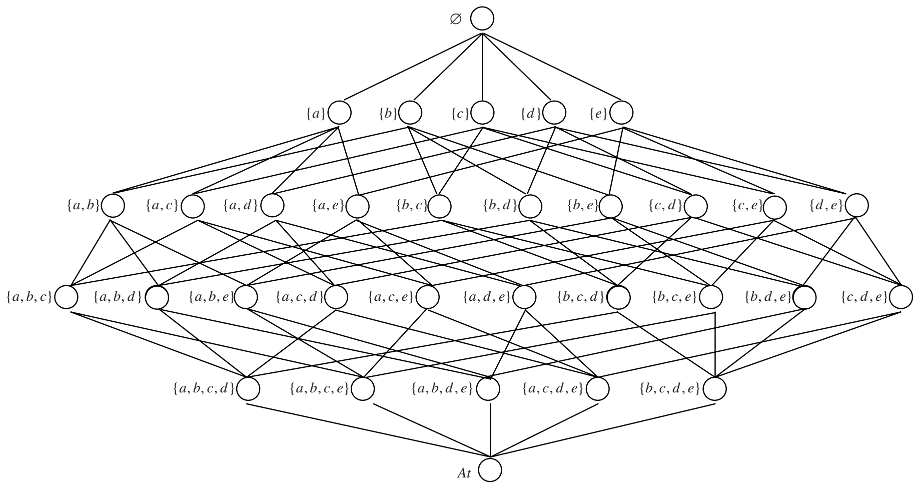


Fig. 1. The attribute lattice of the information Table 1

Given two partitions U/E_A and U/E_B, the meet of their equivalence classes, $U/E_A \wedge U/E_B$, is all nonempty intersections of an equivalence class from U/E_A and an equivalence class from U/E_B. The join of their equivalence classes, $U/E_A \vee U/E_B$, is all unions of an equivalence class from U/E_A and an equivalence class from U/E_B. The meet is the largest refinement partition of both U/E_A and U/E_B; the join is the smallest coarsening partition of both U/E_A and U/E_B. Clearly, $U/E_\emptyset$ is the coarsest partition, and U/E_{At} is the finest partition. For any $A \subseteq At$, we have $U/E_{At} \preceq U/E_A \preceq U/E_\emptyset$. The family of all partitions form a partition lattice under the refinement order.

3.3 Discernibility Matrices

Definition 3. *Given an information table S, for any two objects $(x, y) \in U \times U$, there is an associated discernibility relation $m_{x,y} \subseteq At$, i.e.,*

$$m_{x,y} = \{a \in At \mid I_a(x) \neq I_a(y)\}.$$

The physical meaning of $m_{x,y}$ is that objects x and y can be distinguished by any attribute in $m_{x,y}$.

The family of all discernibility relations can be conveniently stored in a $|U| \times |U|$ matrix, called a discernibility matrix M [21]. A discernibility matrix M is symmetric, i.e., $m_{x,y} = m_{y,x}$, and $m_{x,x} = \emptyset$. The family of all discernibility relations also can be expressed as a set M, collecting only the distinct nonempty elements, i.e., $M = \{m_{x,y} \mid m_{x,y} \neq \emptyset\}$.

Example 3. The discernibility matrix of information Table 1 is illustrated in Table 2. Since the discernibility matrix is symmetric, we only list its lower left half.

Table 2. The discernibility matrix of information Table 1

	o_1	o_2	o_3	o_4	o_5	o_6	o_7
o_1	-	-	-	-	-	-	-
o_2	$\{b,c,d,e\}$	-	-	-	-	-	-
o_3	$\{b,c,e\}$	$\{c,d\}$	-	-	-	-	-
o_4	$\{a,b,d\}$	$\{a,b,c,e\}$	At	-	-	-	-
o_5	$\{b,c,e\}$	$\{b,d\}$	$\{b,c\}$	$\{a,c,d,e\}$	-	-	-
o_6	$\{b,c,d,e\}$	$\{b,c,e\}$	$\{b,d,e\}$	$\{a,b,c,e\}$	$\{b,c,d,e\}$	-	-
o_7	$\{b,c\}$	$\{b,c,d,e\}$	$\{b,e\}$	$\{a,b,c,d\}$	$\{b,c,e\}$	$\{d,e\}$	-

The matrix also can be transformed to a set by collecting distinct elements and eliminating empty elements, such that:

$$\begin{aligned} M = \{&\{b,c\},\{b,d\},\{b,e\},\{c,d\},\{d,e\},\{a,b,d\},\\ &\{b,c,e\},\{b,d,e\},\{a,b,c,d\},\{a,b,c,e\},\\ &\{a,c,d,e\},\{b,c,d,e\},At\}. \end{aligned}$$

The difference of equivalence relations and discernibility relations is obvious. The equivalence relation E_A is based on an attribute set A, indicating all the object pairs that are indiscernible regarding A. The discernibility relation $m_{x,y}$ is based on an object pair (x,y), indicating all the attributes that any of them can distinguish x and y. The relationships between these two relations can be expressed as follows:

$$\begin{aligned} &(x,y) \notin E_{m_{x,y}};\\ &(x,y) \in E_A \Leftrightarrow A \cap m_{x,y} = \emptyset \text{ and } A \cup m_{x,y} = At. \end{aligned}$$

3.4 Reducts

Definition 4. *Given an information table S, a subset $R \subseteq At$ is called a ρ-reduct of At for the property ρ, if R satisfies the two conditions:*

(i). *R and At possess the same property ρ;*
(ii). *for any $a \in R$, $R - \{a\}$ cannot remain the property ρ.*

The first condition indicates the joint sufficiency of the attribute set R, and the second condition indicates that each attribute in R is individually necessary.

The property ρ can be interpreted in different ways. For example, considering the equivalence relations, the property ρ can be expressed as U/E_P, $[x]_P$ or the joint entropy of P, for any $P \subseteq At$ [17]. Also, regarding the family M of discernibility relations, the property ρ can be expressed as $\forall m \in M$, $m \cap P \neq \emptyset$.

According to different interpretations, condition (i) of the reduct definition can be written as:

- $U/E_R = U/E_{At}$,
- for all $x \in U$, $[x]_R = [x]_{At}$,

- $H(R) = H(At)$, where $H(.)$ denotes the joint entropy of the set, or
- $\forall m \in M, m \cap At \neq \emptyset$ and $m \cap R \neq \emptyset$.

It means that the equivalence relations of R and At define the same partition of the universe. For each object x in the universe, x has the same equivalence class defined by R and At. R and At provide the same information grain. Object pairs that can be distinguished by At also can be distinguished by R.

Given an information table, there may exist many reducts. The intersection of all reducts is called the Core.

Definition 5. *An attribute set $R' \subseteq At$ is called a super-reduct of a reduct R, if $R' \supseteq R$; an attribute set $R' \subset At$ $R' \neq \emptyset$ is called a partial reduct of a reduct R, if $R' \subset R$.*

Given a reduct, there exist many super-reducts and many partial reducts.

Figure 2 shows a very simple attribute lattice with 8 nodes in total. Suppose two reducts have been identified, and highlighted by stars on their corresponding nodes. If an attribute set is a reduct, then all its supersets are super-reducts. Here we shade their corresponding nodes in the lattice. At the same time, any subset of a reduct is a partial reduct. In the graph, we use circle with solid line to denote their corresponding nodes.

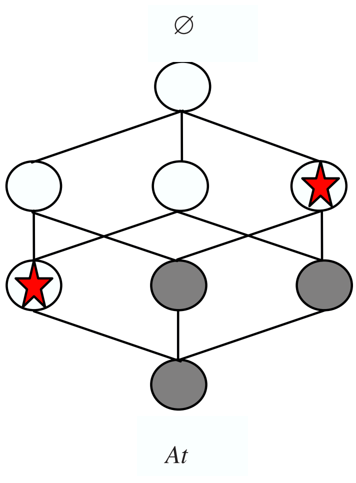


Fig. 2. An illustration of super- and partial reducts in a sample attribute lattice

Reduct computation can be understood as a search in the attribute lattice under the refinement relation. Both the deletion and addition strategies can be practised. The deletion strategy searches from At to $\emptyset$. As long as the condition (i) is met, a reduct or a super-reduct is obtained. If all of its subset are partial reducts, then it is identified as a reduct. A searching heuristic can facilitate the search process by deciding which attribute to be eliminated first, in order to move the search upward.

On the other hand, the addition strategy executes the search from $\emptyset$ to At. When the condition (i) is met, a reduct or a super-reduct is obtained, and the forward selection can be stopped. We need to eliminate the superfluous attributes from a super-reduct, if such is obtained. A searching heuristic decides which attribute to be added first, in order to move the search downward. An enhanced searching heuristic needs to prevent the search from leading to a proper superset of a reduct. By doing so, a backtrack elimination can be saved.

4 Reduct Construction by Deletion

4.1 Control Strategy

By a deletion method, we take At as a super-reduct, which is the largest super-reduct. Deletion methods can be described generally in Algorithm 1.

Algorithm 1. *The deletion method for computing a reduct*
Input: *An information table.*
Output: *A reduct R.*

```
(1) $R = At, CD = At$.
(2) While $CD \neq \emptyset$:
    (2.1) Compute fitness values of all the attributes in $CD$ regarding the
          property $\rho$ using a fitness function $\delta$;
    (2.2) Select an attribute $a$ according to its fitness, let $CD = CD - \{a\}$;
    (2.3) If $R - \{a\}$ is jointly sufficient, let $R = R - \{a\}$.
(3) Output $R$.
```

Many algorithms are proposed based on this simple deletion control strategy. For example, the algorithms proposed in [5,7,30] are implemented for computing a reduct based on information tables.

A deletion method starts with the trivial super-reduct, i.e., the entire attribute. It has to check all the attributes in At for deletion. It is not efficient in the cases when a reduct is short, and many attributes are eliminated from At after checking.

4.2 Attribute Selection Heuristics

The order of attributes for deletion is essential for reduct construction. Regarding a property ρ, different fitness functions may determine different orders of attributes, that may result in different reducts.

The attribute selection heuristic is given by a fitness function:

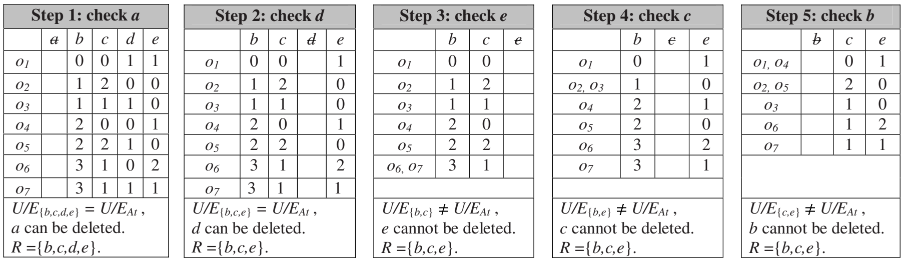
$$\delta : At \longrightarrow \Re, \quad (1)$$

where At is the set of attributes in the information table, and $\Re$ is the set of real numbers. The meaning of the function δ is determined by many semantic considerations. For example, it may be interpreted in terms of the cost of testing, the easiness of understanding, the actionability of an attribute, or the information gain an attribute produces.

Example 4. Suppose the fitness function δ is interpreted as an information entropy

$$\delta(a) = H(a) = -\sum_{v \in V_a} p(v) \log p(v). \tag{2}$$

This heuristic can be easily applied to information tables. For the information Table 1, we obtain $H(a) = 0.592$, $H(b) = 1.950$, $H(c) = 1.557$, $H(d) = 0.985$ and $H(e) = 1.449$, which yields an order $b \rightarrow c \rightarrow e \rightarrow d \rightarrow a$. According to this entropy-based order, the attribute a that contains least information is most likely to be deleted first, and the attributes d, e, c and b are then considered in turn. As a result, a reduct $\{b, c, e\}$ is computed. The iterative steps are illustrated in Figure 3.



Step 1: check *a*

	~~*a*~~	*b*	*c*	*d*	*e*
o_1		0	0	1	1
o_2		1	2	0	0
o_3		1	1	1	0
o_4		2	0	0	1
o_5		2	2	1	0
o_6		3	1	0	2
o_7		3	1	1	1

$U/E_{\{b,c,d,e\}} = U/E_{At}$, *a* can be deleted. $R = \{b,c,d,e\}$.

Step 2: check *d*

	b	*c*	~~*d*~~	*e*
o_1	0	0		1
o_2	1	2		0
o_3	1	1		0
o_4	2	0		1
o_5	2	2		0
o_6	3	1		2
o_7	3	1		1

$U/E_{\{b,c,e\}} = U/E_{At}$, *d* can be deleted. $R = \{b,c,e\}$.

Step 3: check *e*

	b	*c*	~~*e*~~
o_1	0	0	
o_2	1	2	
o_3	1	1	
o_4	2	0	
o_5	2	2	
o_6, o_7	3	1	

$U/E_{\{b,c\}} \neq U/E_{At}$, *e* cannot be deleted. $R = \{b,c,e\}$.

Step 4: check *c*

	b	~~*c*~~	*e*
o_1	0		1
o_2, o_3	1		0
o_4	2		1
o_5	2		0
o_6	3		2
o_7	3		1

$U/E_{\{b,e\}} \neq U/E_{At}$, *c* cannot be deleted. $R = \{b,c,e\}$.

Step 5: check *b*

	~~*b*~~	*c*	*e*
o_1, o_4		0	1
o_2, o_5		2	0
o_3		1	0
o_6		1	2
o_7		1	1

$U/E_{\{c,e\}} \neq U/E_{At}$, *b* cannot be deleted. $R = \{b,c,e\}$.

Fig. 3. An illustration of using a deletion strategy for the information Table 1

Suppose the fitness function δ is interpreted as the frequency that an attribute appears in any element of the discernibility matrix M, i.e.,

$$\delta(a) = |\{m \in M \mid a \in m\}|. \tag{3}$$

We attempt to first delete an attribute that differentiates a small number of objects. We can obtain a set of quantitative values for our sample discernibility matrix in Table 1, such that $\delta(a) = 6$, $\delta(b) = 18$, $\delta(c) = 16$, $\delta(d) = 12$, and $\delta(e) = 15$. The yielded order is consistent with the order yielded by the information gain. Consequently, the same reduct is computed.

Many algorithms use entropy-based heuristics, such as information gain, conditional entropy, and mutual information [2, 15, 16, 24, 27]. Some algorithms use frequency-based heuristics with respect to the discernibility matrix, such as [6, 17, 22, 25].

Besides a quantitative evaluation, the fitness function δ can be interpreted as a qualitative evaluation. A qualitative method relies on pairwise comparisons of attributes. For any two attributes $a, b \in At$, we assume that a user is able to state whether one is more important than, or is more preferred to, the other. Based on the user preference, usually the preferred attributes are intended to be kept, and the unfavourable attributes are intended to be deleted.

5 Reduct Construction by Addition-Deletion

5.1 Control Strategy

By an addition-deletion strategy, we start the construction from an empty set or the Core, and consequently add attributes until a super-reduct is obtained. The constructed super-reduct contains a reduct, but itself is not necessarily a reduct unless all the attributes in it are individually necessary. We need to delete the superfluous attributes in the super-reduct till a reduct is found [29, 30]. The addition-deletion methods can be described generally in Algorithm 2.

Algorithm 2. *The addition-deletion method for computing a reduct*
Input: *An information table.*
Output: *A reduct R.*

```
Addition:
(1) $R = \emptyset, CA = At$.
(2) While $R$ is not jointly sufficient and $CA \neq \emptyset$:
    (2.1) Compute fitness values of all the attributes in $CA$ regarding the
          property $\rho$ using a fitness function $\sigma$;
    (2.2) Select an attribute $a$ according to its fitness, let $CA = CA - \{a\}$;
    (2.3) Let $R = R \cup \{a\}$.

Deletion:
(3) $CD = R$.
(4) While $CD \neq \emptyset$:
    (4.1) Compute fitness values of all the attributes in $CD$ regarding the
          property $\rho$ using a fitness function $\delta$;
    (4.2) Select an attribute $a$ according to its fitness, let $CD = CD - \{a\}$;
    (4.3) If $R - \{a\}$ is jointly sufficient, let $R = R - \{a\}$.
(5) Output $R$.
```

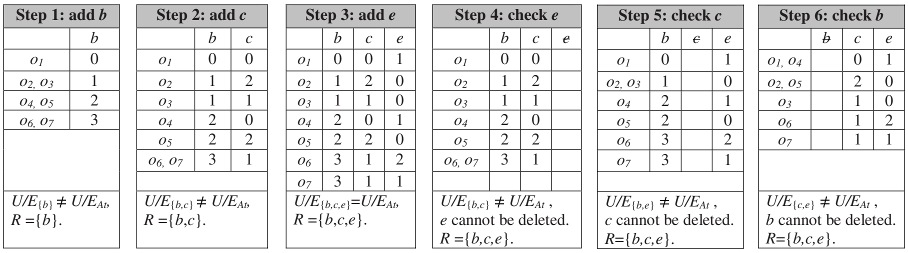
The addition-deletion strategy has been proposed and studied since the deletion strategy is not efficient, and the straightforward addition process can only find a super-reduct, but not a reduct. A lack of consideration of the latter problem has produced many incomplete reduct construction algorithms, such as the ones reported in [8, 10, 16, 20]. An addition-deletion algorithm based on the discernibility matrix has been proposed by Wang and Wang [25], which can construct the subset of attributes from At, and then reduce it to a reduct efficiently.

Due to the fact that an addition-deletion method computes a relatively precise super-reduct first, the deletion checking process is expected to be more efficient than a straightforward deletion-only method. This is true when regarding some orders, a super-reduct is constructed pretty fast. However, the process of computing a super-reduct itself is also time consuming, as well as the process of deleting the superfluous attributes from the constructed super-reduct.

5.2 Attribute Selection Heuristics

For the addition-deletion strategies, the orders of attributes for addition and deletion are both essential for the result reduct. Regarding a property ρ, by using the fitness function σ, we add the fit attributes to the empty set or the Core to form a super-reduct; by using the fitness function δ, we delete the superfluous attributes from the super-reduct in order to form a reduct. σ and δ can be two different heuristics, or the same heuristic. If one can order the attributes according to a fitness function δ from the most fit attribute to the least fit attribute, then this order can be used for adding them one by one until the sufficient condition is met, and the reversed order can be used for deleting the superfluous attributes. By this means, one heuristic determines two orders, and a reduct composed of more fit attributes is obtained.

Example 5. For the information table in Table 1, suppose the fitness function σ is interpreted as the frequency or information gain as we have defined for the fitness function δ in the previous section. A set of quantitative values are computed according to the chosen heuristic. The attribute b is mostly intended to be added, followed by attributes c, e, d and a. In this case, a super-reduct $\{b, c, e\}$ is computed. After using the reverse order to check the necessity, this super-reduct is identified as a reduct. The iterative steps are illustrated in Figure 4.



Step 1: add *b*

	b
o_1	0
o_2, o_3	1
o_4, o_5	2
o_6, o_7	3

$U/E_{\{b\}} \neq U/E_{At}$, $R = \{b\}$.

Step 2: add *c*

	b	c
o_1	0	0
o_2	1	2
o_3	1	1
o_4	2	0
o_5	2	2
o_6, o_7	3	1

$U/E_{\{b,c\}} \neq U/E_{At}$, $R = \{b,c\}$.

Step 3: add *e*

	b	c	e
o_1	0	0	1
o_2	1	2	0
o_3	1	1	0
o_4	2	0	1
o_5	2	2	0
o_6	3	1	2
o_7	3	1	1

$U/E_{\{b,c,e\}} = U/E_{At}$, $R = \{b,c,e\}$.

Step 4: check *e*

	b	c	~~e~~
o_1	0	0	
o_2	1	2	
o_3	1	1	
o_4	2	0	
o_5	2	2	
o_6, o_7	3	1	

$U/E_{\{b,c\}} \neq U/E_{At}$, *e* cannot be deleted. $R = \{b,c,e\}$.

Step 5: check *c*

	b	~~c~~	e
o_1	0		1
o_2, o_3	1		0
o_4	2		1
o_5	2		0
o_6	3		2
o_7	3		1

$U/E_{\{b,e\}} \neq U/E_{At}$, *c* cannot be deleted. $R = \{b,c,e\}$.

Step 6: check *b*

	~~b~~	c	e
o_1, o_4		0	1
o_2, o_5		2	0
o_3		1	0
o_6		1	2
o_7		1	1

$U/E_{\{c,e\}} \neq U/E_{At}$, *b* cannot be deleted. $R = \{b,c,e\}$.

Fig. 4. An illustration of using an addition-deletion strategy for the information Table 1

6 Reduct Construction by Addition

6.1 Control Strategy

By an addition method, we start the reduct construction process from an empty set or the Core, and consequently add attributes to it until it becomes a reduct. The essential difference between the addition method and the addition-deletion method is that, the addition method takes in one attribute if the constructed set is a partial reduct, while the addition-deletion method continuously adds attributes until a super-reduct is produced. In this case, superfluous attributes can be added by an addition-deletion method, and the deletion process is required to eliminate them. The addition methods can be described generally in Algorithm 3.

Algorithm 3. *The addition method for computing a reduct*
Input: *An information table S.*
Output: *A reduct R.*

```
(1) $R = \emptyset$, $CA = At$;
(2) While $CA \neq \emptyset$:
    (2.1) Compute fitness values of all the attributes in $CA$ regarding the
          property $\rho$ using a fitness function $\sigma$;
    (2.2) Select an attribute $a$ according to its fitness;
    (2.3) If $a$ is a core attribute, then let $R = R \cup \{a\}$ and $CA = CA - \{a\}$
          else
          (2.3.1) Compute fitness values of all the elements in $Group(a) =$
                  $\{m \in M \mid a \in m\}$ regarding the property $\rho$ using a fitness
                  function $\delta'$;
          (2.3.2) If $Group(a) = \emptyset$, let $CA = CA - \{a\}$ and go to Step (2),
                  else, select an element $m = \{a\} \cup A$ according to its fitness;
          (2.3.3) If $CA - A$ is jointly sufficient, let $R = R \cup \{a\}$ and
                  $CA = CA - m$, else, go to Step (2.3.2).
                  If $a$ cannot be made necessary regarding all $m \in Group(a)$,
                  let $CA = CA - \{a\}$.
(3) Output $R$.
```

For a selected attribute $a \in CA$, $Group(a) = \{m \in M \mid a \in m\}$ is the set of matrix elements that each indicates an object pair that can be distinguished by a. If a is a core attribute then it is individually necessary for constructing a reduct. If a is a non-core attribute, one can make a necessary by eliminating its associated attributes in an element $m \in Group(a)$ from further consideration. Suppose $Group(a) = \{m_1, m_2, \ldots, m_d\}$, $(m_i = A \cup \{a\}) \in Group(a)$ and $A \neq \emptyset$. It means that all the attributes in m can distinguish the object pair associated with m, and the attribute a is not individually necessary for such a task. We can make a necessary by eliminating all the attributes in A. If A is a superset of another element $m' \in M$, then A is necessary for distinguishing the object pair associated with m', which means that A cannot be eliminated. In other words, the attribute a cannot be made necessary regarding m. If a cannot be made necessary regarding all $m_i \in Group(a)$, then a cannot be added to the partial reduct.

Example 6. For our running example, suppose the non-core attribute a is selected. It is easy to obtain from the matrix in Table 2 that $Group(a) = \{m_{o_1,o_4}, m_{o_2,o_4}, m_{o_3,o_4}, m_{o_4,o_5}, m_{o_4,o_6}, m_{o_4,o_7}\}$. Suppose to make a necessary, the matrix element $m_{o_1,o_4} = \{a, b, d\}$ is selected, and thus the attributes in the set $m_{o_1,o_4} - \{a\} = \{b, d\}$ need to be eliminated. However, the elimination will cause the element $m_{o_2,o_5} = \{b, d\}$ becomes empty. In other words, the object pair (o_2, o_5) can no longer be distinguished. Therefore, attribute set $\{b, d\}$ cannot be eliminated, which means that attribute a cannot be added to the partial reduct regarding the matrix element m_{o_1,o_4}. We can easily verify that a cannot be added regarding any matrix element in $Group(a)$, thus a does not belong to the partial reduct.

6.2 Attribute Selection Heuristics

The addition algorithm requires the attributes added to the reduct are individually necessary. To ensure it, the associated attributes are eliminated for consideration. At the same time, the elimination should not change the joint sufficiency of the remaining attributes. Therefore, the general addition algorithm explicitly checks both the sufficiency condition and the necessity condition in Step (2.3.3). Its time complexity is higher than the general deletion algorithm.

Zhao and Wang suggested using a matrix absorption operation to simplify the checking process [29]. The matrix absorption operation is a sequence of all possible element absorption operations on pairs of elements whenever the following condition holds:

$$\emptyset \neq M(x', y') \subset M(x, y).$$

That is, the value of $M(x, y)$ is replaced by the value of $M(x', y')$ in the matrix. We also say $M(x, y)$ is absorbed by $M(x', y')$. The physical meaning of the absorption can be explained as follows. Suppose $M(x', y') \neq \emptyset$ and $M(x', y') \subset M(x, y)$. The set of attributes discerning both pairs (x', y') and (x, y) is given by $M(x, y) \cap M(x', y') = M(x', y')$. After absorption, $M(x, y)$ becomes $M(x', y')$. Attributes in $M(x', y')$ are sufficient to discern both object pairs (x', y') and (x, y). When an attribute from $M(x', y')$ is in a reduct, the same attribute can be used to discern (x, y). Thus, it is not necessary to consider attributes in $M(x, y) - M(x', y')$. After matrix absorption, no element in the matrix is a proper subset of another element.

By using the matrix absorption operation, the general addition algorithm can be much simplified. Let attribute a be selected in Steps (2.1) and (2.2), and $Group(a)$ store the elements contains a from the absorbed matrix. When an element $(m = \{a\} \cup A) \in Group(a)$ is selected in Steps (2.3.1) and (2.3.2), A can be eliminated immediately. Since A is not a proper subset of another element, thus is not necessary for distinguishing any object pair. Since $CA - A$ is ensured jointly sufficient, therefore, m can be eliminated after attribute a being made necessary and added to the partial reduct. We can set $CA = \{a \in At \mid Group(a) \neq \emptyset\}$, and apply the matrix absorption operation every time after the CA is updated.

Example 7. For our running Example 3, we can observe that the distinct matrix element $\{b, c\}$ can distinguish the object pairs (o_1, o_7) and (o_3, o_5) whose corresponding matrix elements equal to $\{b, c\}$, and also the object pairs (o_1, o_2), (o_1, o_3), (o_1, o_5), (o_1, o_6), (o_2, o_4), (o_2, o_6), (o_2, o_7), (o_3, o_4), (o_4, o_6), (o_4, o_7), (o_5, o_6) and (o_5, o_7) whose corresponding matrix elements contain $\{b, c\}$. By applying the matrix absorption operation we can obtain:

$\{d, e\}$ absorbs $\{b, d, e\}, \{a, c, d, e\}, \{b, c, d, e\}, At$;
$\{b, e\}$ absorbs $\{b, c, e\}, \{b, d, e\}, \{a, b, c, e\}, \{b, c, d, e\}, At$;
$\{b, c\}$ absorbs $\{b, c, e\}, \{a, b, c, d\}, \{a, b, c, e\}, \{b, c, d, e\}, At$;
$\{b, d\}$ absorbs $\{a, b, d\}, \{b, d, e\}, \{a, b, c, d\}, \{b, c, d, e\}, At$;
$\{c, d\}$ absorbs $\{a, b, c, d\}, \{a, c, d, e\}, \{b, c, d, e\}, At\}$.

As a result, the absorbed discernibility matrix contains the following distinct elements $\{b,c\}$, $\{b,d\}$, $\{b,e\}$, $\{c,d\}$, $\{d,e\}$. We can use $\widehat{M}$ denote the absorbed matrix in a set representation.

For the absorbed discernibility matrix, if we group the matrix elements, we obtain five overlapped sets:

$$\begin{aligned} Group(a) &= \emptyset, \\ Group(b) &= \{\{b,c\},\{b,d\},\{b,e\}\}, \\ Group(c) &= \{\{b,c\},\{c,d\}\}, \\ Group(d) &= \{\{b,d\},\{c,d\},\{d,e\}\}, \\ Group(e) &= \{\{b,e\},\{d,e\}\}. \end{aligned}$$

Attribute set $CA = \{b,c,d,e\}$.

The fitness function σ can be the one that we discussed in Sections 4 and 5. We need to discuss more about the fitness function δ'. We should note that the fitness function δ' of the proposed addition algorithm is different from the fitness function δ of the general deletion algorithm. That is because δ evaluates the fitness of one single attribute at a time, and δ' evaluates the fitness of a matrix element m, which is a set of attributes. Typically, δ' is the summation or the average fitness of all the included attributes.

Quantitatively, the selection of a matrix element for deletion can be described by a mapping:

$$\delta' : \{m_i \in Group(a)\} \longrightarrow \Re. \tag{4}$$

The meaning of the function δ' is determined by many semantic considerations as well.

Example 8. For example, a frequency-based heuristic can be defined as follows. For $m_i = \{a\} \cup A$,

$$\delta'(m_i) = |\{m \in \widehat{M} \mid m \cap A \neq \emptyset\}|. \tag{5}$$

For the running example, if the reduct attribute b is selected according to the information gain measure, we thus focus on $Group(b) = \{\{b,c\},\{b,d\},\{b,e\}\}$. Using the former heuristic, we obtain that $\delta'(\{b,c\}) = 2, \delta'(\{b,d\}) = 3$, and $\delta'(\{b,e\}) = 2$ in $\widehat{M}$. Suppose we therefore pick the element $\{b,d\}$. Consequently, a reduct $\{b,c,e\}$ can be computed. The iterative steps are illustrated in Figure 4.

We can also define the fitness function δ' as the information entropy, i.e., the joint entropy of all the attributes in the attribute set $m_i - \{a\}$. For example, if $m_i - \{a\} = \{b,c\}$, then

$$\begin{aligned} \delta'(m_i) &= H(m_i - \{a\}) \\ &= H(\{b,c\}) \\ &= -\sum_{x \in V_b} \sum_{y \in V_c} p(b,c) \log p(b,c). \end{aligned} \tag{6}$$

$\widehat{M} = \{\{b,c\}, \{b,d\}, \{b,e\}, \{c,d\}, \{d,e\}\}$

$CA = \{b, c, d, e\}$

Group$(b) = \{\{b,c\}, \{b,d\}, \{b,e\}\}$

Step 1: add b Delete $\{b, d\}$ from CA

	b	c	e
o_1	0	0	1
o_2	1	2	0
o_3	1	1	0
o_4	2	0	1
o_5	2	2	0
o_6	3	1	2
o_7	3	1	1

$U/E_{\{b,c,e\}} = U/E_{At}$, $R = \{b\}$.

$\widehat{M} = \{\{c\}, \{e\}\}$

$CA = \{c, e\}$

Group$(c) = \{\{c\}\}$

Step 2: add c Delete $\{c\}$ from CA

	b	c	e
o_1	0	0	1
o_2	1	2	0
o_3	1	1	0
o_4	2	0	1
o_5	2	2	0
o_6	3	1	2
o_7	3	1	1

$U/E_{\{b,c,e\}} = U/E_{At}$, $R = \{b,c\}$.

$\widehat{M} = \{\{e\}\}$

$CA = \{e\}$

Group$(e) = \{\{e\}\}$

Step 3: add e Delete $\{e\}$ from CA

	b	c	e
o_1	0	0	1
o_2	1	2	0
o_3	1	1	0
o_4	2	0	1
o_5	2	2	0
o_6	3	1	2
o_7	3	1	1

$U/E_{\{b,c,e\}} = U/E_{At}$, $R = \{b,c,e\}$.

Fig. 5. An illustration of using an addition strategy for the information Table 1

By applying this heuristic to the sample information Table 1, we obtain that $\delta'(\{b,c\}) = H(\{c\}) = 1.557$, $\delta'(\{b,e\}) = H(\{e\}) = 1.449$ and $\delta'(\{b,d\}) = H(\{d\}) = 0.985$. The reduct $\{b,c,e\}$ can be computed if the element $\{b,d\}$ is selected.

Similarly, qualitative evaluation can also be applied here for selecting a matrix element for deletion. This can be based on the user preference on the attribute set, that we have discussed in the previous sub-section. We usually select the most unfavourable matrix element for deletion.

Example 9. For our running example in Table 1, we only find two reducts according to the introduced heuristics. By applying different heuristics, we may be able to find the rest of reducts, like $\{c,d,e\}$ and $\{b,d\}$. The attribute lattice shown in Figure 6 highlights all the reducts by stars, and super-reducts by shadings, partial reducts by circles with solid lines.

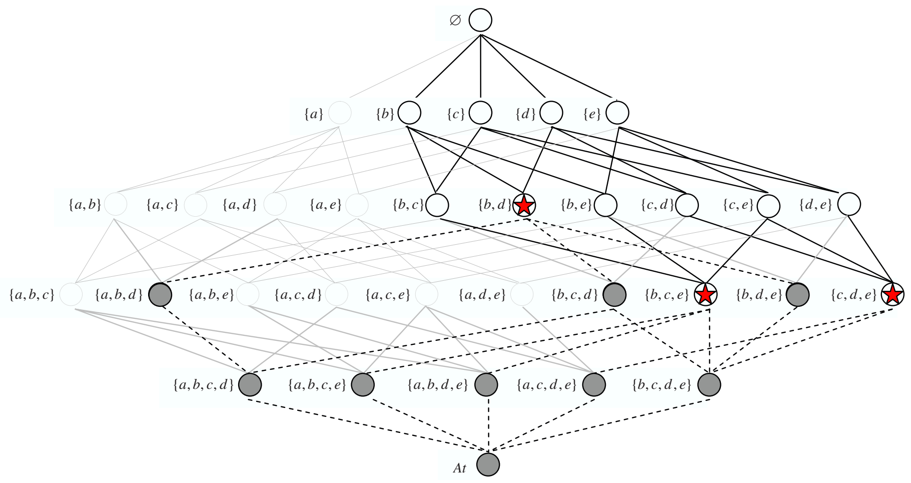


Fig. 6. An illustration of super- and partial reducts in the attribute lattice of Table 1

7 Time Complexity Analysis

Suppose the partition of the information table is chosen for the time complexity analysis. For an attribute $a \in At$, the execution of $U/E_{\{a\}}$ needs to compare each object pair regarding attribute a. It thus requires $\frac{|U|(|U|+1)}{2}$ comparisons, where $|U|$ is the cardinality of U. For an attribute set $A \subset At$, the execution of U/E_A needs to compare each object pair regarding all the attributes in A. It thus requires $\frac{|U|(|U|+1)}{2}|A|$ comparisons.

The attribute deletion operation of the deletion strategy is to check if the remaining attribute set is still jointly sufficient for each iteration. To check the necessity of attribute a_1, one needs to verify if $At - \{a_1\}$ produces the same partition as At does, and thus $\frac{|U|(|U|+1)}{2}(|At| - 1)$ comparisons are required. If a_1 is deleted after the checking, then to verify if $At - \{a_1\} - \{a_2\}$ produces the same partition as At does, one needs $\frac{|U|(|U|+1)}{2}(|At| - 2)$ comparisons. If a_1 is not deleted after the checking, then one still needs $\frac{|U|(|U|+1)}{2}(|At| - 1)$ comparisons, to check the necessity of a_2. Totally, $O(|U|^2|At|^2)$ comparisons are required to check the jointly sufficiency condition and the individual necessity condition for all attributes.

The addition-deletion strategy checks the joint sufficiency condition for a constructed super-reduct, and checks the individual necessity condition for all the attributes in the constructed super-reduct. To verify if $U/E_{\{a_1\}} = U/E_{At}$ one needs $\frac{|U|(|U|+1)}{2}$ comparisons. To verify if $U/E_{\{a_1,a_2\}} = U/E_{At}$ one needs $\frac{2|U|(|U|+1)}{2}$ comparisons, and so on. Totally, $O(|U|^2|At|^2)$ comparisons are required to construct a super-reduct. And same number of comparisons are required to check the necessity of all attributes.

The addition strategy picks an attribute to make it individually necessary by eliminating its associated attributes, at the same time, it ensures the elimination does not change the joint sufficiency of the remaining attributes. To ensure the necessity of attribute a_1, one needs to verify if $At - A_1$ produces the same partition as At does, where $m_{a_1} = \{a_1\} \cup A_1$. This requires $\frac{|U|(|U|+1)}{2}(|At| - |A_1|)$ comparisons. If a_1 is added after the checking, then one needs to verify if $At - m_{a_1} - A_2$ produces the same partition as At does, where $m_{a_2} = \{a_2\} \cup A_2$. This requires $\frac{|U|(|U|+1)}{2}(|At| - |m| - |A_2|)$ comparisons. If a_1 is not added after the checking, then one still needs $\frac{|U|(|U|+1)}{2}(|At| - |A_2| - 1)$ comparisons, to ensure the necessity of a_2. Totally, $O(|U|^2|At|^2)$ comparisons are required to check the jointly sufficiency condition and the individual necessity condition for all attributes.

This analysis is very rough. It should be noted that the addition-deletion algorithm normally does not need to add all attributes in At for a super-reduct. The addition algorithm relies on the absorption operation to simplify the matrix and generate the groups for all attributes. Normally, the addition algorithm is the most inefficient one comparing to the other two.

8 Conclusion

This paper provides a critical study of the existing reduct construction algorithms based on a two-level view: a high level view of control strategy and a low level view of attribute selection heuristics. Three groups of algorithms are discussed based on the deletion strategy, the addition-deletion strategy and the addition strategy.

We define the concepts of super-reducts and partial reducts besides the concept of reduct. A deletion strategy and an addition-deletion strategy strike to find a reduct from a super-reduct. An addition strategy strikes to find a reduct from a partial reduct.

This paper may be considered as an attempt to synthesize the results from existing studies into a general and easy to understand form, with an objective towards a more abstract theory. Any success in such a research will not only produce valuable insights into the problem, but also provide guidelines for the design of new reduct construction algorithms.

References

1. Bazan, J.G., Nguyen, H.S., Nguyen, S.H., Synak, P., Wroblewski, J.: Rough set algorithms in classification problem. In: Polkowski, L., Tsumoto, S., Lin, T.Y. (eds.) Rough Set Methods and Applications, pp. 49–88 (2000)
2. Beaubouef, T., Petry, F.E., Arora, G.: Information-theoretic measures of uncertainty for rough sets and rough relational databases. Information Sciences 109, 185–195 (1998)
3. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artificial Intelligence, 245–271 (1997)
4. Devijver, P.A., Kittler, J.: Pattern Recognition: A Statistical Approach. Prentice-Hall, New York (1982)
5. Grzymala-Busse, J.W.: LERS - A system for learning from examples based on rough sets. In: Slowinski, R. (ed.) Intelligent Decision Support, pp. 3–18. Kluwer Academic Publishers, Boston (1992)
6. Hoa, N.S., Son, N.H.: Some efficient algorithms for rough set methods. In: Proceedings of the Conference of Information Processing and Management of Uncertainty in Knowledge-based Systems, pp. 1451–1456 (1996)
7. Hu, X.: Using rough sets theory and database operations to construct a good ensemble of classifiers for data mining applications. In: Proceedings of ICDM, pp. 233–240 (2001)
8. Hu, X., Cercone, N.: Learning in relational databases: a rough set approach. International Journal of Computation Intelligence 11, 323–338 (1995)
9. Jain, A.K., Zongker, D.: Feature selection: evaluation, application and small sample performance. IEEE Transactions on Pattern Analysis and Machine Intelligence 19, 153–158 (1997)
10. Jenson, R., Shen, Q.: A rough set-aided system for sorting WWW bookmarks. In: Zhong, N., et al. (eds.) Web Intelligence: Research and Development, pp. 95–105 (2001)
11. John, G.H., Kohavi, R., Pfleger, K.: Irrelevant features and the subset selection problem. In: Proceedings of the Eleventh International Conference on Machine Learning, pp. 121–129 (1994)

12. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artificial Intelligence, 273–324 (1997)
13. Koller, D., Sahami, M.: Toward optimal feature selection. In: Proceedings of the Thirteenth International Conference of Machine Learning, pp. 284–292 (1996)
14. Mi, J.S., Wu, W.Z., Zhang, W.X.: Approaches to knowledge reduction based on variable precision rough set model. Information Sciences 159, 255–272 (2004)
15. Miao, D., Hou, L.: A comparison of rough set methods and representative inductive learning algorithms. Fundamenta Informaticae 59, 203–219 (2004)
16. Miao, D., Wang, J.: An information representation of the concepts and operations in rough set theory. Journal of Software 10, 113–116 (1999)
17. Pawlak, Z.: Rough Sets: Theoretical Aspects of Reasoning About Data. Kluwer, Boston (1991)
18. Portinale, L., Saitta, L.: Feature selection, Technical report, D14.1, University of Dortmund (2002)
19. Rauszer, C.: Reducts in information systems. Foundamenta Informaticae 15, 1–12 (1991)
20. Shen, Q., Chouchoulas, A.: A modular approach to generating fuzzy rules with reduced attributes for the monitoring of complex systems. Engineering Applications of Artificial Intelligence 13, 263–278 (2000)
21. Skowron, A., Rauszer, C.: The discernibility matrices and functions in information systems. In: Slowiński, R. (ed.) Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory. Kluwer, Dordrecht (1992)
22. Slezak, D.: Various approaches to reasoning with frequency based decision reducts: a survey. In: Polkowski, L., Tsumoto, S., Lin, T.Y. (eds.) Rough set methods and applications, pp. 235–285 (2000)
23. Swiniarski, R.W.: Rough sets methods in feature reduction and classification. International Journal of Applied Mathematics and Computer Science 11, 565–582 (2001)
24. Wang, G., Yu, H., Yang, D.: Decision table reduction based on conditional information entropy. Chinese Journal of Computers 25, 759–766 (2002)
25. Wang, J., Wang, J.: Reduction algorithms based on discernibility matrix: the ordered attributes method. Journal of Computer Science and Technology 16, 489–504 (2001)
26. Wong, S., Ziarko, W.: On optimal decision rules in decision tables. Bulletin of the Polish Academy of Sciences and Mathematics, 693–696 (1985)
27. Yu, H., Yang, D., Wu, Z., Li, H.: Rough set based attribute reduction algorithm. Computer Engineering and Applications 17, 22–47 (2001)
28. Zhao, M.: Data Description Based on Reduct Theory, Ph.D. Thesis, Institute of Automation, Chinese Academy of Sciences (2004)
29. Zhao, K., Wang, J.: A reduction algorithm meeting users' requirements. Journal of Computer Science and Technology 17, 578–593 (2002)
30. Ziarko, W.: Rough set approaches for discovering rules and attribute dependencies. In: Klösgen, W., Żytkow, J.M. (eds.) Handbook of Data Mining and Knowledge Discovery, pp. 328–339 (2002)