Towards Scalable Algorithms for Discovering Rough Set Reducts

Marzena Kryszkiewicz[1] and Katarzyna Cichoń[1,2]

[1] Institute of Computer Science, Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
mkr@ii.pw.edu.pl
[2] Institute of Electrical Apparatus, Technical University of Lodz
Stefanowskiego 18/22, 90-924 Lodz, Poland
cichon@p.lodz.pl

Abstract. Rough set theory allows one to find reducts from a decision table, which are minimal sets of attributes preserving the required quality of classification. In this article, we propose a number of algorithms for discovering all generalized reducts (preserving generalized decisions), all possible reducts (preserving upper approximations) and certain reducts (preserving lower approximations). The new *RAD* and *CoreRAD* algorithms, we propose, discover exact reducts. They require, however, the determination of all maximal attribute sets that are not supersets of reducts. In the case, when their determination is infeasible, we propose *GRA* and *CoreGRA* algorithms, which search approximate reducts. These two algorithms are well suited to the discovery of supersets of reducts from very large decision tables.

1 Introduction

Rough set theory has been conceived as a non-statistical tool for analysis of imperfect data [17]. Rough set methodology allows one to discover interesting data dependencies, decision rules, repetitive data patterns and to analyse conflict situations [24]. The reasoning in the rough set approach is based solely on available information. Objects are perceived as indiscernible if they have the same description in the system. This may be a reason for uncertainty. Two or more objects identically described in the system may belong to different classes (concepts). Such concepts, though vague, can be defined roughly by means of a pair of crisp sets: lower approximation and upper approximation. Lower approximation of a concept is a set of objects that surely belong to that concept, whereas upper approximation is a set of objects that possibly belong to that concept.

Rough set theory allows one to find reducts from a decision table, which are minimal sets of attributes preserving the required quality of classification. For example, a reduct may preserve lower approximations of decision classes, or upper approximations of decision classes, or both. A number of methods for discovering reducts have already been proposed in the literature [2-8, 11, 15-17, 20-31]. The most popular

J.F. Peters et al. (Eds.): Transactions on Rough Sets I, LNCS 3100, pp. 120–143, 2004.

methods are based on discernibility matrices [20]. Other methods are based, e.g., on the theory of cones and fences [7, 19]. Unfortunately, the existing methods are not capable to discover all reducts from very large decision tables, although research on discovering rough set decision rules in large data sets started a few years ago (see e.g., [9-10, 14]). One may try to overcome this problem either by applying heuristics or data sampling or both, or by restricting search to looking for some reducts instead of all of them.

Recently, we have proposed the *GRA*-like (*GeneralizedReductsApriori*) algorithms for discovering approximate generalized, possible and certain reducts from very large decision tables [13]. This article extends the results obtained in [13]. Here, we propose new algorithms - *RAD* and *CoreRAD* - for discovering exact generalized, possible and certain reducts. *CoreRAD* is a variation of *RAD*, which uses information on the so-called core in order to restrict the number of candidates for reducts and the number of scans of the decision table. The new algorithms require the determination of all maximal sets that are not supersets of reducts ($\mathcal{MNSR}$). The knowledge of $\mathcal{MNSR}$ is sufficient to evaluate candidates for reducts correctly. The method of creating and pruning candidates is very similar to the one proposed in *GRA* [13]. In the case, when the calculation of $\mathcal{MNSR}$ is infeasible, we advocate to search approximate reducts. In the article, we first introduce the theory behind approximate reducts and then present in detail respective algorithms (*GRA* and *CoreGRA*).

The layout of the article is as follows: In Section 2, we remind basic rough set notions and prove some of their properties that will be applied in the proposed algorithms. In Section 3, we propose the *RAD* algorithm for discovering generalized and possible reducts. A number of optimizations of the basic algorithm are discussed as well. The *CoreRAD* algorithm, which calculates both the core and the reducts, is offered in Section 4. In Section 5, we discuss briefly how to adapt *RAD* and *CoreRAD* for the discovery of certain reducts. The notions of approximate reducts are introduced in Section 6. We prove that approximate reducts are supersets of exact reducts. The properties of approximate generalized reducts are used in the construction of the *GRA* algorithm, which is presented in Section 7. In Section 8, we discuss the *CoreGRA* algorithm, which calculates both the approximate generalized reducts and the approximate core. In Section 9, we propose simple modifications of *GRA* and *CoreGRA* that enable the usage of these algorithms for discovering approximate certain reducts. Section 10 concludes the results indicating that the proposed solutions can be applied in the case of incomplete decision tables as well.

2 Basic Notions

2.1 Information Systems

An *information system* (*IS*) is a pair S = (O, *AT*), where O is a non-empty finite set of *objects* and *AT* is a non-empty finite set of *attributes*, such that a: O $\rightarrow V_a$ for any $a \in AT$, where V_a is called *domain* of the attribute a.

An attribute-value pair (a,v), where $a \in AT$ and $v \in V_a$, is called an *atomic descriptor*. An *atomic descriptor* or its conjunction is called a *descriptor* [20]. A conjunction of atomic descriptors for attributes $A \subseteq AT$ is called *A-descriptor*.

Let S = (O, *AT*). Each subset of attributes $A \subseteq AT$ determines a binary *indiscernibility relation IND*(*A*), $IND(A) = \{(x,y) \in O \times O \mid \forall a \in A, a(x) = a(y)\}$. The relation *IND*(*A*), $A \subseteq AT$, is an equivalence relation and constitutes a partition of O. Objects indiscernible with regard to their description on attribute set *A* in the system will be denoted by $I_A(x)$; that is, $I_A(x) = \{y \in O \mid (x,y) \in IND(A)\}$.

Property 1 [9]. Let $A, B \subseteq AT$.

a) If $A \subseteq B$, then $I_B(x) \subseteq I_A(x)$.

b) $I_{A \cup B}(x) = I_A(x) \cap I_B(x)$.

c) $I_A(x) = \bigcap_{a \in A} I_a(x)$.

Let $X \subseteq O$ and $A \subseteq AT$. $\underline{A}X$ is defined as a *lower approximation* of *X* iff $\underline{A}X = \{x \in O \mid I_A(x) \subseteq X\} = \{x \in X \mid I_A(x) \subseteq X\}$. $\overline{A}X$ is defined as an *upper approximation* of *X* iff $\overline{A}X = \{x \in O \mid I_A(x) \cap X \neq \emptyset\} = \bigcup \{I_A(x) \mid x \in X\}$. $\underline{A}X$ is the set of objects that belong to *X* with certainty, while $\overline{A}X$ is the set of objects that possibly belong to *X*.

2.2 Decision Tables

A *decision table* is an information system $DT = (O, AT \cup \{d\})$, where $d \notin AT$ is a distinguished attribute called the *decision*, and the elements of *AT* are called *conditions*. The set of all objects whose decision value equals *k*, $k \in V_d$, will be denoted by X_k. Let us define the function $\partial_A: O \rightarrow P(V_d)$, $A \subseteq AT$, as follows [18]:

$$\partial_A(x) = \{d(y) \mid y \in I_A(x)\}.$$

∂_A will be called *A-generalized decision* in *DT*. For *A* = *AT*, an *A-generalized decision* will be also called briefly a *generalized decision*.

Table 1. $DT = (O, AT \cup \{f\})$ extended by generalized decision ∂_{AT}.

$x \in O$	*a*	*b*	*c*	*D*	*e*	*f*	∂_{AT}
1	1	0	0	1	1	1	{1}
2	1	1	1	1	2	1	{1}
3	0	1	1	0	3	1	{1,2}
4	0	1	1	0	3	2	{1,2}
5	0	1	1	2	2	2	{2}
6	1	1	0	2	2	2	{2,3}
7	1	1	0	2	2	3	{2,3}
8	1	1	0	3	2	3	{3}
9	1	0	0	3	2	3	{3}

Table 2. $DT' = (O, AT \cup \{\partial_{AT}\})$ – sorted and reduced version of *DT* from Table 1.

$x \in O$ in *DT'* ($x \in O$ in *DT*)	*a*	*b*	*c*	*d*	*e*	∂_{AT}
1 (3,4)	0	1	1	0	3	{1,2}
2 (5)	0	1	1	2	2	{2}
3 (1)	1	0	0	1	1	{1}
4 (9)	1	0	0	3	2	{3}
5 (6,7)	1	1	0	2	2	{2,3}
6 (8)	1	1	0	3	2	{3}
7 (2)	1	1	1	1	2	{1}

Example 1. Table 1 describes a sample decision table *DT*. The conditional attributes are as follows: *AT* = {*a, b, c, d, e*}. The decision attribute is *f*. One may note that objects 3 and 4 are indiscernible with respect to the conditional attributes in *AT*.

Hence, ∂_{AT} for object 3 contains both the decision 1 for object 3, as well as the decision 2 for object 4. Analogously, ∂_{AT} for object 4 contains both its own decision (2), as well as the decision of object 3 (1). Please see the last column in Table 1 for generalized decision ∂_{AT} for all objects in *DT*. Let X_1 be the class of objects determined by decision 1; that is, $X_1 = \{1,2,3\}$. The lower and upper approximations of X_1 are as follows: $\underline{AT}X_1 = \{1,2\}$ and $\overline{AT}X_1 = \{1,2,3,4\}$. □

Property 2 shows that the approximations of decision classes can be expressed by means of an *A*-generalized decision.

Property 2 [9-11]. Let $X_i \subseteq$ O and $A \subseteq AT$.

a) $I_A(x) \subseteq X_i$ iff $\partial_A(x) = \{i\}$.
b) $I_A(x) \cap X_i \neq \emptyset$ iff $i \in \partial_A(x)$.
c) $\underline{A}X_i = \{x \in \mathrm{O} \mid \partial_A(x) = \{i\}\}$.
d) $\overline{A}X_i = \{x \in \mathrm{O} \mid i \in \partial_A(x)\}$.
e) $\partial_A(x) = \partial_A(y)$ for any $(x,y) \in IND(A)$.

By Property 2e, objects having the same *A*-descriptor have also the same *A*-generalized decision value; that is, the *A*-descriptor uniquely determines the *A*-generalized decision value for all objects satisfying this descriptor. In the sequel, the *A*-generalized decision value determined by *A*-descriptor *t*, such that *t* is satisfied by at least one object in the system, will be denoted by ∂_t. Table 2 shows the generalized decision values determined by atomic descriptors that occur in Table 1.

Table 3. Generalized decision values $\partial_{(a,v)}$ determined by atomic descriptors (a,v), where $a \in AT$, $v \in V_a$, supported by *DT* from Table 1.

(a,v)	$(a,0)$	$(a,1)$	$(b,0)$	$(b,1)$	$(c,0)$	$(c,1)$	$(d,0)$	$(d,1)$	$(d,2)$	$(d,3)$	$(e,1)$	$(e,2)$	$(e,3)$
$\partial_{(a,v)}$	{1,2}	{1,2,3}	{1,3}	{1,2,3}	{1,2,3}	{1,2}	{1,2}	{1}	{2,3}	{3}	{1}	{1,2,3}	{1,2}

We note that the *A*- and *B*-generalized decision values for object *x* provide an upper bound on the $A \cup B$-generalized decision value for *x*.

Property 3 [13]. Let $A,B \subseteq AT$, $x \in DT$. $\partial_{A \cup B}(x) \subseteq \partial_A(x) \cap \partial_B(x)$.

Proof: $\partial_{A \cup B}(x) = \{d(y) \mid y \in I_{A \cup B}(x)\}$ = /* by Property 1b */ = $\{d(y) \mid y \in (I_A(x) \cap I_B(x))\}$ $\subseteq \{d(y) \mid y \in I_A(x)\} \cap \{d(y) \mid y \in I_B(x)\} = \partial_A(x) \cap \partial_B(x)$. □

We conclude further that the elementary *a*-generalized decision values for *x*, $a \in A$, can be used for calculating an upper bound on the *A*-generalized decision value for *x*.

Corollary 1. Let $A \subseteq AT$ and $x \in DT$. $\partial_A(x) \subseteq \bigcap_{a \in A} \partial_a(x) = \bigcap_{a \in A} \partial_{(a,\, a(x))}$.

Example 2. The $\{ce\}$-generalized decision value calculated from *DT* in Table 1 for object 5 ($\partial_{\{ce\}}(5) = \{1,2\}$) equals its upper bound $\partial_c(5) \cap \partial_e(5) = \partial_{(c,1)} \cap \partial_{(e,2)} = \{1,2\}$ $\cap \{1,2,3\} = \{1,2\}$. On the other hand, the $\{ce\}$-generalized decision value for object 6 ($\partial_{\{ce\}}(6) = \{2,3\}$) is a proper subset of its upper bound $\partial_c(6) \cap \partial_e(6) = \partial_{(c,0)} \cap \partial_{(e,2)}$ $= \{1,2,3\} \cap \{1,2,3\} = \{1,2,3\}$. □

Corollary 2. Let $A \subseteq B \subseteq AT$, $x \in DT$. $\partial_B(x) \subseteq \partial_A(x)$.

Proof: By Property 3, $\partial_B(x) \subseteq \partial_A(x) \cap \partial_{B \setminus A}(x)$. Hence, $\partial_B(x) \subseteq \partial_A(x)$. □

Finally, we observe that A- and B-generalized decision values for object x, where $A \subseteq B \subseteq AT$, are identical when their cardinalities are identical.

Proposition 1. Let $A \subseteq B \subseteq AT$ and $x \in DT$. $\partial_A(x) = \partial_B(x)$ iff $|\partial_A(x)| = |\partial_B(x)|$.

Proof: ($\Rightarrow$) Straightforward.

($\Leftarrow$) Let $|\partial_A(x)| = |\partial_B(x)|$ (*). Since, $A \subseteq B$, then by Corollary 2, $\partial_A(x) \supseteq \partial_B(x)$. Taking into account (*), we conclude $\partial_A(x) = \partial_B(x)$. □

2.3 Reducts for Decision Tables

Reducts for decision tables are minimal sets of conditional attributes that preserve the required properties of classification. In what follows, we provide definitions of reducts preserving lower and upper approximations of decision classes and objects' generalized decisions, respectively.

Let $\emptyset \neq A \subseteq AT$. A is a *certain reduct* (*c-reduct*) of DT iff A is a minimal attribute set such that

$$\forall x \in \mathrm{O},\ x \in \underline{AT}X_{d(x)} \Rightarrow I_A(x) \subseteq X_{d(x)} \quad \text{(c)}$$

A certain reduct is a set of attributes that allows us to distinguish each object x belonging to the lower approximation of its decision class in DT from the objects that do not belong to this approximation.

A is a *possible reduct* (*p-reduct*) of DT iff A is a minimal attribute set such that

$$\forall x \in \mathrm{O},\ I_A(x) \subseteq \overline{AT}X_{d(x)} \quad \text{(p)}$$

A possible reduct is a set of attributes that allows us to distinguish each object x in DT from objects that do not belong to the upper approximation of its decision class.

A is a *generalized decision reduct* (*g-reduct*) of DT iff A is a minimal set such that

$$\forall x \in \mathrm{O},\ \partial_A(x) = \partial_{AT}(x) \quad \text{(g)}$$

A generalized decision reduct is a set of attributes that preserves the generalized decision value for each object x in DT. In the sequel, a superset of a t-reduct, where $t \in \{c, p, g\}$, will be called a *t-super-reduct*.

Corollary 3. AT is a superset of all c-reducts, p-reducts, and g-reducts for any DT.

Proposition 2. Let $A \subseteq AT$.

a) If A satisfies property (**c**), then all of its supersets satisfy property (**c**).
b) If A does not satisfy property (**c**), then all of its subsets do not satisfy (**c**).
c) If A satisfies property (**p**), then all of its supersets satisfy property (**p**).
d) If A does not satisfy property (**p**), then all of its subsets do not satisfy (**p**).
e) If A satisfies property (**g**), then all of its supersets satisfy property (**g**).
f) If A does not satisfy property (**g**), then all of its subsets do not satisfy (**g**).

Proof: Let $A \subseteq B \subseteq AT$ and $x \in \mathrm{O}$.

Ad a) Let A satisfy property (**c**) and $x \in \underline{AT}X_{d(x)}$. We are to prove that $I_B(x) \subseteq X_{d(x)}$. Since A satisfies property (**c**), then $I_A(x) \subseteq X_{d(x)}$ (*). By Property 1a, $I_B(x) \subseteq I_A(x)$ (**). By (*) and (**), $I_B(x) \subseteq X_{d(x)}$.
Ad b) Analogous to a).
Ad c) Let A satisfy property (**g**). We are to prove that $\partial_B(x) = \partial_{AT}(x)$. Since A satisfies property (**g**), then $\partial_A(x) = \partial_{AT}(x)$ (*). By Corollary 2, $\partial_{AT}(x) \subseteq \partial_B(x) \subseteq \partial_A(x)$ (**). By (*) and (**), $\partial_B(x) = \partial_{AT}(x)$.
Ad b, d, f) Follow immediately from Proposition 2a, b, c, respectively. □

Corollary 4.
a) c-super-reducts are all and the only attribute sets that satisfy property (**c**).
b) p-super-reducts are all and the only attribute sets that satisfy property (**p**).
c) g-super-reducts are all and the only attribute sets that satisfy property (**g**).

Proof: By definition of reducts and Proposition 2. □

Interestingly, not only g-reducts, but also p-reducts and c-reducts, can be determined by examining generalized decisions.

Theorem 1 [11]. The set of all generalized decision reducts of DT equals the set of all possible reducts of DT.

Lemma 1 [13]. $A \subseteq AT$ is a c-reduct of DT iff A is a minimal set such that $\forall x \in \mathrm{O}$, $\partial_{AT}(x) = \{d(x)\} \Rightarrow \partial_A(x) = \{d(x)\}$.

Proof: By Property 2a,c. □

Corollary 5 [13]. $A \subseteq AT$ is a *c-reduct* of DT iff A is a minimal set such that $\forall x \in \mathrm{O}$, $\partial_{AT}(x) = \{d(x)\} \Rightarrow \partial_A(x) = \partial_{AT}(x)$.

2.4 Core

The notion of a *core* is meant to be the greatest set of attributes without which an attribute set does not satisfy the required classification property (i.e. is not a super-reduct). The generic notion of a *t-core*, $t \in \{c, p, g\}$, corresponding to c-reducts, p-reducts and g-reducts, respectively, is defined as follows:

$$t\text{-}core = \{a \in AT \mid AT \setminus \{a\} \text{ is not a } t\text{-super-reduct}\}.$$

Clearly, the p-core and g-core are the same.

Proposition 3. Let $\mathcal{R}$ be all reducts of the same type t, where $t \in \{c, p, g\}$.

$$t\text{-core} = \bigcap \mathcal{R}.$$

Proof: Let us consider the case when $\mathcal{R}$ is the set of all c-reducts. Let $b \in c$-core. Hence b is an attribute in AT such that $AT \setminus \{b\}$ is not a superset of c-reduct. By Corollary 4a and Proposition 2b, no attribute set without b satisfies property (**c**). Hence, no attribute set without b is a c-reduct. Thus, all c-reducts contain b; that is, $\bigcap \mathcal{R} \supseteq \{b\}$. Generalizing this observation, $\bigcap \mathcal{R} \supseteq c$-core.

Now, we will prove by contradiction that $\bigcap\mathcal{R} \setminus c$-core is an empty set. Let $d \in \bigcap\mathcal{R}$ and $d \notin c$-core. Since $d \notin c$-core, then, by definition of a core, $AT\setminus\{d\}$ is a superset of some c-reduct, say B. Since B is a subset of $AT\setminus\{d\}$, then B does not contain d either. This means that among c-reducts, there is an attribute set (B), which does not contain d. Therefore, $d \notin \bigcap\mathcal{R}$, which contradicts the assumption.

The cases when $\mathcal{R}$ is the set of all p-reducts or g-reducts can be proved analogously from Corollary 4b,c and Proposition 2d,f, respectively. □

3 Discovering Generalized Reducts

3.1 Main Algorithm

Notation for *RAD*

- $\mathcal{R}_k$ – candidate k attribute sets (potential g-reducts);
- $\mathcal{A}_k$ – k attribute sets that are not g-super-reducts;
- $\mathcal{MNSR}$ – all maximal conditional attribute sets that are not g-super-reducts;
- $\mathcal{MNSR}_k$ – k attribute sets in $\mathcal{MNSR}$;
- *DT'* – reduced *DT*;
- $x.a$ – the value of an attribute a for object x;
- $x.\partial_{AT}$ – the generalized decision value for object x.

Algorithm. *RAD*;

DT' = *GenDecRepresentation-of-DT*(*DT*);
$\mathcal{MNSR}$ = *MaximalNonSuperReducts*(*DT'*);
/* search g-reducts - note: g-reducts are all attribute sets that are not subsets of any set in $\mathcal{MNSR}$ */
if $|\mathcal{MNSR}_{|AT|-1}| = |AT|$ **then return** AT**;** // optional optimizing step 1
$\mathcal{R}_1 = \{\{a\} | a \in AT\}$; $\mathcal{A}_1 = \{\}$; // initialize 1 attribute candidates for g-reducts
forall $B \in \mathcal{MNSR}$ **do** move subsets of B from $\mathcal{R}_1$ to $\mathcal{A}_1$; // subsets of non-super-reducts are not reducts
for ($k = 1$; $\mathcal{A}_k \neq \{\}$; k++) **do begin**
  if $|\mathcal{MNSR}| = 1$ **then return** $\cup_k \mathcal{R}_k$; // optional optimizing step 2
  $\mathcal{MNSR} = \mathcal{MNSR} \setminus \mathcal{MNSR}_k$; // $\mathcal{MNSR}_k$ is not useful any more – optional optimizing step 3
  /* create k+1 attribute g-reducts $\mathcal{R}_{k+1}$ and non-g-super-reducts $\mathcal{A}_{k+1}$ from $\mathcal{A}_k$ and $\mathcal{MNSR}$ */
  RADGen($\mathcal{R}_{k+1}$, $\mathcal{A}_{k+1}$, $\mathcal{A}_k$, $\mathcal{MNSR}$);
endfor;
return $\cup_k \mathcal{R}_k$;

The *RAD* (*ReductsAprioriDiscovery*) algorithm we propose starts by determining the reduced decision table *DT'* that stores only conditional attributes *AT* and the *AT*-generalized decision for each object in *DT* instead of the original decision (see Section 3.2 for the description of the *GenDecRepresentation-of-DT* function). Each class of objects indiscernible w.r.t. $AT \cup \{\partial_{AT}\}$ in *DT* (see Table 1) is represented by one object in *DT'* (see Table 2). Next, *DT'* is examined in order to find all maximal attribute sets $\mathcal{MNSR}$ that are not g-super-reducts (see Section 3.3 for the description of the *MaximalNonSuperReducts* function). The information on $\mathcal{MNSR}$ is sufficient to derive all g-reducts; namely, g-reducts are these sets each of which has no superset in $\mathcal{MNSR}$ (i.e., is a g-super-reduct), but all proper subsets of which have supersets in $\mathcal{MNSR}$ (i.e., are not g-reducts).

Now, *RAD* creates initial candidates for *g*-reducts that are singleton sets and are stored in $\mathcal{R}_1$. The candidates in $\mathcal{R}_1$ that are subsets of $\mathcal{MNSR}$ are moved to 1 attribute non-*g*-super-reducts $\mathcal{A}_1$. The main loop starts. In each *k*-th iteration, $k \geq 1$, *k*+1 attribute candidates $\mathcal{R}_{k+1}$ are created from *k* attribute sets in $\mathcal{A}_k$, which are not *g*-super-reducts (see Section 3.4 for the description of the *RADGen* procedure). The information on non-*g*-super-reducts $\mathcal{MNSR}$ is used to prune candidates in $\mathcal{R}_{k+1}$. Namely, each candidate in $\mathcal{R}_{k+1}$ that has a superset in $\mathcal{MNSR}$ is not a *g*-super-reduct. Therefore it is moved from $\mathcal{R}_{k+1}$ to $\mathcal{A}_{k+1}$. The algorithm stops when $\mathcal{A}_k = \{\}$. Optional optimizing steps in *RAD* are discussed in Section 3.5.

3.2 Determining Generalized Decision Representation of Decision Table

The *GenDecRepresentation-of-DT* function starts with sorting the given decision table *DT* w.r.t. the set of all conditional attributes and (optionally) the decision attribute. The sorting enables fast determination of the generalized decision values for all classes of objects indiscernible w.r.t. *AT*. Each such class will be represented by one object in the new decision table $DT' = (AT, \{\partial_{AT}\})$, where the decision attribute is replaced by the generalized decision.

function *GenDecRepresentation-of-DT*(decision table *DT*);

```
  DT' = {};
  sort DT with respect to AT and d;        // apply any ordering of attributes in AT, e.g. lexicographical
  x = first object in DT;                  // or null if DT is empty
  while x is not null do begin
    forall a∈AT do x'.a = x.a;   x'.∂_AT = {d(y)| y∈I_AT(x)};   add x' to DT';
    x = the first object located just after I_AT(x) in DT;
  endwhile;
return DT';
```

3.3 Calculating Maximal Non-super-reducts

The purpose of the *MaximalNonSuperReducts* function is to determine all maximal conditional attribute sets that are not *g*-super-reducts. To this end, each object in the reduced decision table *DT'* is compared with all other objects from different generalized decision classes. The result of the comparison of two objects, say *x* and *y*, belonging to different classes is the set of all attributes on which *x* and *y* are indiscernible. Clearly, such a resulting set is not a *g*-super-reduct, since it does not discern at least one pair of objects belonging to different generalized decision classes. The comparison results, which are non-*g*-super-reducts, are stored in the $\mathcal{NSR}$ variable. After the comparison of objects is accomplished, $\mathcal{NSR}$ contains a superset of all maximal non-*g*-super-reducts. The function returns MAX($\mathcal{NSR}$), which can be calculated as the final step or on the fly. For *DT'* from Table 2, *MaximalNonSuperReducts* will find $\mathcal{NSR} = \{abc, b, bc, e, bde, be, bce, ac, ace, ae, abce, abe\}$, and eventually will return MAX($\mathcal{NSR}$) = $\{abce, bde\}$.

function *MaximalNonSuperReductsNew*(reduced decision table *DT'*);
  $\mathcal{NSR}$ = {};
  forall objects *x* in *DT'* **do**
    forall objects *y* following *x* in *DT'* **do**
      if $x.\partial_{AT} \neq y.\partial_{AT}$ **then**
      /* objects *x* and *y* should be distinguishable as they belong to different generalized decision classes; */
      /* the set $\{a \in AT \mid x.a = y.a\}$ is not a *g*-super-reduct since it does not distinguish between *x* and *y* */
        insert in $\{a \in AT \mid x.a = y.a\}$, if non-empty, to $\mathcal{NSR}$;
return MAX($\mathcal{NSR}$); // note: MAX($\mathcal{NSR}$) contains all maximal non-*g*-super-reducts

3.4 Generating Candidates for Reducts

The *RADGen* procedure has 4 arguments. Two of them are input ones: *k* attribute non-*g*-super-reducts $\mathcal{A}_k$ and the maximal non-*g*-super-reducts $\mathcal{MNSR}$. The two remaining candidates $\mathcal{R}_{k+1}$ and $\mathcal{A}_{k+1}$ are output ones. After the completion of the function, $\mathcal{R}_{k+1}$ contains *k*+1 attribute *g*-reducts and $\mathcal{A}_{k+1}$ contains *k*+1 attribute non-*g*-super-reducts. During the first phase of the procedure, new *k*+1 attribute candidates are created by merging *k* attribute non-*g*-super-reducts in $\mathcal{A}_k$ that differ only in their final attributes. The characteristic feature of such a method of creating candidates is that no candidate that is likely to be a solution (here: *g*-reduct) is missed and that no candidate is generated twice (please, see the detailed description of the *Apriori* algorithm [1] for justification). In the second phase, it is checked for each newly obtained *k*+1 attribute candidate whether all its proper *k* attribute subsets are contained in non-*g*-super-reducts $\mathcal{A}_k$. If yes, then a candidate remains in $\mathcal{R}_{k+1}$; otherwise it is pruned as a proper superset of some *g*-super-reduct. Finally, all candidates in $\mathcal{R}_{k+1}$ that are subsets of maximal non-*g*-super-reducts $\mathcal{MNSR}$ are found non-*g*-super-reducts too, and thus are moved to $\mathcal{A}_{k+1}$.

procedure *RADGen*(**var** $\mathcal{R}_{k+1}$, **var** $\mathcal{A}_{k+1}$, **in** $\mathcal{A}_k$, **in** $\mathcal{MNSR}$);
  forall $B, C \in \mathcal{A}_k$ **do** /* Merging */
    if $B[1] = C[1] \wedge ... \wedge B[k\text{-}1] = C[k\text{-}1] \wedge B[k] < C[k]$ **then begin**
      $A = B[1] \bullet B[2] \bullet ... \bullet B[k] \bullet C[k]$; add *A* to $\mathcal{R}_{k+1}$;
    endif;
  forall $A \in \mathcal{R}_{k+1}$ **do** /* Pruning */
    forall *k* attribute sets $B \subset A$ **do**
      if $B \notin \mathcal{A}_k$ **then** delete *A* from $\mathcal{R}_{k+1}$; // *A* is a proper superset of *g*-super-reduct *B*
  forall $B \in \mathcal{MNSR}$ **do** move subsets of *B* from $\mathcal{R}_{k+1}$ to $\mathcal{A}_{k+1}$; /* Removing subsets of non-*g*-super-reducts */
return;

3.5 Optimizing Steps in *RAD*

In the main algorithm, we offer an optimization that may speed up checking which candidates are not *g*-reducts (optimizing step 3) and two optimizations for reducing the number of useless iterations (optimizing steps 1 and 2).

In step 3, *k* attribute sets are deleted from $\mathcal{MNSR}$ since they are useless for identifying non-*g*-superset-reducts among *l* attribute candidates, where $l > k$.

Optimizing step 1 is based on the following observation: the condition $|\mathcal{MNSR}_{|AT|-1}| = |AT|$ implies that all $AT\backslash\{a\}$ sets are not g-super-reducts. Hence, AT is the only g-reduct for DT and thus the algorithm can be stopped.

Optimizing step 2 can be applied when $|\mathcal{MNSR}| = 1$. This condition implies that all sets in $\mathcal{A}_k$, which are not g-super-reducts, have exactly one - the same superset, say B, in maximal non-g-super-reducts $\mathcal{MNSR}$. If one continues the creation of k+1 attribute candidates $\mathcal{R}_{k+1}$ by merging sets in $\mathcal{A}_k$, then the new k+1 attribute candidates would be still subsets of B. Hence, they would be pruned by the *RADGen* procedure from $\mathcal{R}_{k+1}$ to $\mathcal{A}_{k+1}$. As a result, one would obtain $\mathcal{R}_{k+1} = \{\}$ and $|\mathcal{MNSR}| = 1$. Such a scenario would continue when creating longer candidates until $\mathcal{A}_l = \{B\}$, $l > k$. Then, *RADGen* will produce empty $\mathcal{R}_{l+1}$ and empty $\mathcal{A}_{l+1}$; that is, the condition, which stops the *RAD* algorithm. In conclusion, the condition $|\mathcal{MNSR}| = 1$ implies that no more g-reducts will be discovered, so the algorithm can be stopped.

3.6 Illustration of *RAD*

Let us illustrate now the discovery of g-reducts of DT from Table 1. We assume that maximal non-g-super-reducts $\mathcal{MNSR}$ are already found and are equal to $\{\{abce\}, \{bde\}\}$. Table 4 shows how candidates for g-reducts change in each iteration.

Table 4. $\mathcal{R}_k$ and $\mathcal{A}_k$ after verification w.r.t. $\mathcal{MNSR}$ in subsequent iterations of *New*.

k	$\mathcal{A}_k$ (each X in $\mathcal{A}_k$ has a superset in $\mathcal{MNSR}$)	$\mathcal{R}_k$ (each X in $\mathcal{R}_k$ has no superset in $\mathcal{MNSR}$)
1	$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}$	
2	$\{ab\}, \{ac\}, \{ae\}, \{bc\}, \{bd\}, \{be\}, \{ce\}, \{de\}$	$\{ad\}, \{cd\}$
3	$\{abc\}, \{abe\}, \{ace\}, \{bce\}, \{bde\}$	
4	$\{abce\}$	

4 Core-Oriented Discovery of Generalized Reducts

4.1 Main Algorithm

In this section, we offer the *CoreRAD* procedure, which finds not only g-reducts, but also their core. The layout of *CoreRAD* reminds that of *RAD*. *CoreRAD*, however, differs from *RAD* in that it first checks if the set of all maximal non-g-super-reducts $\mathcal{MNSR}$ is empty. If yes, then each single conditional attribute is a g-reduct, so *CoreRAD* returns $\{\{a\} | a \in AT\}$ as the set of all g-reducts and $\bigcap_{a \in AT} \{a\} = \emptyset$ as the g-core (by Proposition 3). Otherwise, *CoreRAD* determines the g-core by definition from all maximal $|AT|$-1 non-g-super-reducts in $\mathcal{MNSR}$. All sets in $\mathcal{MNSR}$ that are not supersets of the g-core are deleted, since the only candidates considered in *CoreRAD* will be the g-core and its supersets. If the reduced $\mathcal{MNSR}$ is an empty set, then the g-core does not have subsets in $\mathcal{MNSR}$ and thus it is the only g-reduct. Otherwise, the g-core is not a g-reduct, and the new candidates $\mathcal{R}_{|core|+1}$ are created by merging the g-core with the remaining attributes in AT. Clearly, the new candidates

which have supersets in maximal non-*g*-super-reducts $\mathcal{MNSR}$ are not *g*-reducts either, and hence are moved from $\mathcal{R}_{|core|+1}$ to $\mathcal{A}_{|core|+1}$. From now on, *CoreRAD* is performed in the same way as *RAD*.

Algorithm. *CoreRAD*;

DT' = *GenDecRepresentation-of-DT*(*DT*);
$\mathcal{MNSR}$ = *MaximalNonSuperReducts*(*DT'*);
if $\mathcal{MNSR} = \{\}$ **then return** $(\emptyset, \{\{a\} | a \in AT\})$; // each conditional attribute is a *g*-reduct
core = $\emptyset$;
forall $A \in \mathcal{MNSR}_{|AT|-1}$ **do begin** $\{a\} = AT \setminus A$; *core* = *core* $\cup \{a\}$ **endfor**;
if $|\mathcal{MNSR}_{|AT|-1}| = |AT|$ **then return** (AT, AT); // or **if** *core* = *AT* **then** - optional optimizing step 1
$\mathcal{MNSR} = \{B \in \mathcal{MNSR} | B \supseteq core\}$; // *g*-reducts are supersets of the *g*-core
if $\mathcal{MNSR} = \{\}$ **then return** $(core, \{core\})$; // *g*-core is a *g*-reduct as there is no its superset in $\mathcal{MNSR}$
$\mathcal{MNSR} = \mathcal{MNSR} \setminus \mathcal{MNSR}_{|core|}$; // or equivalently $\mathcal{MNSR} = \mathcal{MNSR} \setminus \{core\}$;
/* initialize candidate for reducts as *g*-core's supersets */
startLevel = |*core*| + 1; $\mathcal{R}_{startLevel} = \{\}$; $\mathcal{A}_{startLevel} = \{\}$;
forall $a \in AT \setminus core$ **do begin** $A = core \cup \{a\}$; $\mathcal{R}_{startLevel} = \mathcal{R}_{startLevel} \cup \{A\}$ **endfor**;
forall $B \in \mathcal{MNSR}$ **do** move subsets of *B* from $\mathcal{R}_{startLevel}$ to $\mathcal{A}_{startLevel}$;
for (*k* = *startLevel*; $\mathcal{A}_k \neq \{\}$; *k*++) **do begin**
  if $|\mathcal{MNSR}| = 1$ **then return** $(core, \cup_k \mathcal{R}_k)$; // optional optimizing step 2
  $\mathcal{MNSR} = \mathcal{MNSR} \setminus \mathcal{MNSR}_k$; // $\mathcal{MNSR}_k$ is not useful any more – optional optimizing step 3
  /* create *k*+1 attribute *g*-reducts $\mathcal{R}_{k+1}$ and non-*g*-super-reducts $\mathcal{A}_{k+1}$ from $\mathcal{A}_k$ and $\mathcal{MNSR}$ */
  GRAGen($\mathcal{R}_{k+1}$, $\mathcal{A}_{k+1}$, $\mathcal{A}_k$, $\mathcal{MNSR}$);
endfor;
return $(core, \cup_k \mathcal{R}_k)$;

4.2 Illustration of *CoreRAD*

We will illustrate now the core-oriented discovery of *g*-reducts of *DT* from Table 1. We assume that $\mathcal{MNSR}$ has already been calculated and equals $\{\{abce\}, \{bde\}\}$. Hence, *core* = *AT* / $\{abce\}$ = $\{d\}$. Now, we leave only the supersets of the *core* in $\mathcal{MNSR}$; thus $\mathcal{MNSR}$ becomes equal to $\{\{bde\}\}$. Table 5 shows how candidates for *g*-reducts change in each iteration (here: only 1 iteration was sufficient).

Table 5. $\mathcal{R}_k$ and $\mathcal{A}_k$ after verification w.r.t. $\mathcal{MNSR}$ in subsequent iterations of *CoreRAD*.

K	$\mathcal{A}_k$ (each *X* in $\mathcal{A}_k$ has a superset in $\mathcal{MNSR}$)	$\mathcal{R}_k$ (each *X* in $\mathcal{R}_k$ has no superset in $\mathcal{MNSR}$)
2	$\{bd\}, \{de\}$	$\{ad\}, \{cd\}$

5 Discovering Certain Reducts

RAD and *CoreRAD* can easily be adapted for the discovery of certain reducts. It suffices to modify line 4 of the *MaximalNonSuperReducts* function as follows:

if $(x.\partial_{AT} \neq y.\partial_{AT})$ **and** $(|x.\partial_{AT}| = 1$ **or** $|y.\partial_{AT}| = 1)$ **then**

This modification guarantees that all objects from lower approximations of all decision classes, which have singleton generalized decisions, will be compared with all objects not belonging to the lower approximations of their decision classes.

6 Approximate Attribute Reduction

6.1 Approximate Reducts for Decision Table

The discovery of reducts may be very time consuming. Therefore, one may resign from calculating strict reducts and search more efficiently for approximate reducts, which however, should be supersets of exact reducts and subsets of *AT*. In this section, we introduce the notion of such approximate reducts based on the observation that for any object x in O: $\bigcap_{a\in A} \partial_a(x) \supseteq \partial_A(x)$ (by Corollary 1).

Let $\emptyset \neq A \subseteq AT$. AT is defined an approximate generalized decision reduct (ag-reduct) of DT iff $\exists x \in O$, $\bigcap_{a\in AT} \partial_a(x) \supset \partial_{AT}(x)$. Otherwise, A is an approximate generalized decision reduct (g-reduct) of DT iff A is a minimal set such that

$$\forall x \in O,\ \bigcap_{a\in A} \partial_a(x) = \partial_{AT}(x) \qquad \text{(ag)}$$

Corollary 5 specifies properties of certain decision reducts in terms of generalized decisions. By analogy to this corollary, we define an *approximate certain decision reduct* as follows:

AT is defined an *approximate certain decision reduct* (*ac-reduct*) of *DT* iff $\exists x \in O$, $\partial_{AT}(x) = \{d(x)\} \Rightarrow \bigcap_{a\in AT} \partial_a(x) \supset \partial_{AT}(x)$. Otherwise, *A* is defined an *approximate certain reduct* (*ac-reduct*) of *DT* iff *A* is a minimal attribute set such that

$$\forall x \in O,\ \partial_{AT}(x) = \{d(x)\} \Rightarrow \bigcap_{a\in A} \partial_a(x) = \partial_{AT}(x) \qquad \text{(ac)}$$

In the sequel, a superset of a *t*-reduct, $t \in \{ac, ag\}$, will be called a *t-super-reduct*.

Corollary 6. *AT* is a superset of all *ac*-reducts and *ag*-reducts for any *DT*.

Proposition 4. Let $x \in O$ and $A \subseteq AT$. If $\bigcap_{a\in A} \partial_a(x) = \partial_{AT}(x)$, then:

a) $\bigcap_{a\in A} \partial_a(x) = \partial_A(x) = \partial_{AT}(x)$.

b) $\forall B \subseteq AT, B \supset A \Rightarrow \bigcap_{a\in B} \partial_a(x) = \partial_B(x) = \partial_{AT}(x)$.

Proof: Let $\bigcap_{a\in A} \partial_a(x) = \partial_{AT}(x)$ (*).

Ad a) By Corollaries 1-2, $\bigcap_{a\in A} \partial_a(x) \supseteq \partial_A(x) \supseteq \partial_{AT}(x)$. Taking into account (*), $\bigcap_{a\in A} \partial_a(x) = \partial_A(x) = \partial_{AT}(x)$.

Ad b) Let $B \subseteq AT$, $B \supset A$. By Corollary 2, $\partial_A(x) \supseteq \partial_B(x) \supseteq \partial_{AT}(x)$. Taking into account Proposition 4a, $\bigcap_{a\in A} \partial_a(x) = \partial_A(x) = \partial_B(x) = \partial_{AT}(x)$ (**). Clearly, $\bigcap_{a\in A} \partial_a(x) \supseteq \bigcap_{a\in B} \partial_a(x) \supseteq \bigcap_{a\in AT} \partial_a(x)$. Taking into account (**), $\partial_B(x) = \partial_{AT}(x) = \bigcap_{a\in A} \partial_a(x) \supseteq \bigcap_{a\in B} \partial_a(x) \supseteq \bigcap_{a\in AT} \partial_a(x) \supseteq \partial_{AT}(x)$. Hence, $\bigcap_{a\in B} \partial_a(x) = \partial_B(x) = \partial_{AT}(x)$. □

Corollary 7.

a) An *ag*-reduct is a *g*-super-reduct.
b) An *ag*-reduct is a *p*-super-reduct.
c) An *ac*-reduct is a *c*-super-reduct.

Proof: Ad a) Let A be an *ag*-reduct. If $\exists x \in O$, $\bigcap_{a \in AT} \partial_a(x) \supset \partial_{AT}(x)$, then $A = AT$, which by Corollary 3 is a *g*-super-reduct. Otherwise, by definition of an *ag*-reduct and Proposition 4a, $\forall x \in O$, $\bigcap_{a \in A} \partial_a(x) = \partial_A(x) = \partial_{AT}(x)$. Thus A satisfies property (**g**). Hence, by Corollary 4c, A is a *g*-super-reduct.
Ad b) Follows from Theorem 1 and Corollary 7a.
Ad c) Analogous, to the proof of Corollary 7a. Follows from the definition of an *ac*-reduct, Corollary 3, Corollary 5, Corollary 4a and Proposition 4a.

Proposition 5. Let $A \subseteq AT$.
a) If A satisfies property (**ag**), then all of its supersets satisfy property (**ag**).
b) If A does not satisfy property (**ag**), then all of its subsets do not satisfy (**ag**).
c) If A satisfies property (**ac**), then all of its supersets satisfy property (**ac**).
d) If A does not satisfy property (**ac**), then all of its subsets do not satisfy (**ac**).

Proof: Ad a,c) Follow from Proposition 4.
Ad b, d) Follow immediately from Proposition 5a, c, respectively. □

Corollary 8.
a) *ag*-super-reducts are all and the only attribute sets that satisfy property (**ag**).
b) *ac*-super-reducts are all and the only attribute sets that satisfy property (**ac**).

Proof: By definition of respective approximate reducts and Proposition 5. □

6.2 Approximate Core

An *approximate core* will be defined in usual way; that is,

$$t\text{-}core = \{a \in AT \mid AT \setminus \{a\} \text{ is not a } t\text{-super-reduct}\}, \text{ where } t \in \{ac, ag\}.$$

Proposition 6. Let $\mathcal{R}$ be all approximate reducts of the same type t, $t \in \{ac, ag\}$.

$$t\text{-core} = \bigcap \mathcal{R}.$$

Proof: Follows from Corollary 8 and Proposition 5, and is analogous to the proof of Proposition 3. □

7 Discovering Approximate Generalized Reducts

7.1 Main Algorithm

The *GRA* (*GeneralizedReductsApriori*) algorithm, we have recently introduced in [13], finds all *ag*-reducts from the decision table *DT*. Unlike in *RAD*, *GRA*, does not need to store all maximal non-*g*-super-reducts $\mathcal{MNSR}$. On the other hand, *GRA* requires the candidates for reducts to be evaluated against the decision table. The validation of the candidate solution against the decision table *DT* in our algorithm consists in checking if the candidate satisfies property (**ag**); that is, if the intersection of the elementary generalized decisions of the attributes in the candidate set determines the same generalized decision value as the set of all conditional attributes *AT* does for each object in *DT*. We will use the following properties in the process of searching reducts in order to prune the search space efficiently:

- Proper supersets of *ag*-reducts are not *ag*-reducts, and hence such sets shall not be evaluated against the decision table.
- Subsets of attribute sets that are not *ag*-super-reducts are not *ag*-reducts, and thus such sets shall not be evaluated against the decision table.
- An attribute set whose all proper subsets are not *ag*-super-reducts may or may not be an *ag*-reduct, and hence should be evaluated against the decision table.

Since our algorithm is to work with very large decision tables, we propose to restrict the number of decision table objects against which a candidate should be evaluated. Our proposal is based on the following observation:

- If an attribute set A satisfies property (**ag**) for the first n objects in DT (or reduced DT') and does not satisfy it for object $n+1$, then A is certainly not an *ag*-reduct and thus evaluating it against the remaining objects in DT (DT') is useless.
- If an attribute set A satisfies property (**ag**) for the first n objects in DT (or DT'), then property (**ag**) will be satisfied for these objects for all supersets of A. Hence, the evaluation of the first n objects should be skipped for a candidate that is a proper superset of A.

The *GRA* algorithm starts with building the reduced version DT' of decision table DT (see Section 3.2 for the description of the *GenDecRepresentation-of-DT* function). DT' stores only the AT-generalized decisions instead of the original decisions. Next, the a-generalized decision value for each atomic descriptor (a,v) occurring in DT (or in DT') is calculated as the set of the decisions (or the union of the AT-generalized decisions) of the objects supporting (a,v) in DT (or in DT'). Each pair: (*atomic descriptor*, *its generalized decision*) is stored in Γ. Now *GRA* creates initial candidates for *ag*-reducts. The initial candidates are singleton sets and are stored in $\mathcal{R}_1$. The set of 1 attribute non-*ag*-super-reducts $\mathcal{A}_1$, as well as known maximal non-*ag*-super-reducts $\mathcal{NSR}$, are initialized to an empty set. The main loop starts. In each k-th iteration, $k \geq 1$, the k attribute candidates $\mathcal{R}_k$ are evaluated during one pass over DT' (see Section 7.2 for the description of the *EvaluateCandidates* procedure). As a side effect of evaluating of $\mathcal{R}_k$, all k attribute non-*ag*-super-reducts $\mathcal{A}_k$ are found and known maximal non-*ag*-super-reducts $\mathcal{NSR}$ are updated. The case when $\mathcal{NSR}_{|AT|} = AT$ indicates that AT does not satisfy property (**ag**) for some object. Hence, by definition AT is the only *ag*-reduct and the algorithms stops. Otherwise, $k+1$ attribute candidates $\mathcal{R}_{k+1}$ are created from k attribute sets in $\mathcal{A}_k$, which turned out not to be *ag*-super-reducts (see Section 7.4 for the description of the *GRAGen* procedure). The information on non-*ag*-super-reducts $\mathcal{NSR}$ is used to prune the candidates in $\mathcal{R}_{k+1}$. Namely, each candidate in $\mathcal{R}_{k+1}$ that has a superset in $\mathcal{NSR}$ is known a priori not to be an *ag*-reduct. Therefore it is moved from $\mathcal{R}_{k+1}$ to $\mathcal{A}_{k+1}$. The algorithm stops when $\mathcal{R}_k = \mathcal{A}_k = \{\}$. Optimizations steps 1-2 in *GRA* are analogous to steps 1-2 in *RAD*, which were discussed in Section 3.5.

Modified or additional notation for *GRA*

- $\mathcal{R}_k$ – candidate k attribute sets (potential *ag*-reducts);
- $\mathcal{A}_k$ – k attribute sets that are not *ag*-super-reducts;
- $A.id$ – the identifier of the object against which attribute set A should be evaluated;
- $\mathcal{NSR}$ – quasi maximal attribute sets found not to be *ag*-super-reducts;
- $\mathcal{NSR}_k$ – k attribute sets in $\mathcal{NSR}$;
- $x.identifier$ – the identifier of object x;
- Γ - the set containing generalized decision values determined by atomic descriptors supported by objects in DT (DT'); that is: $\Gamma = \cup_{a\in AT, v\in V_a} \{\{(a,v), \partial_{(a,v)})\}$.

Algorithm. *GRA*;

DT' = *GenDecRepresentation-of-DT*(DT);
/* calculate *a*-generalized decision value for each atomic descriptor (a,v) supported by DT (or DT') */
for each conditional attribute $a\in AT$ **do**
for each domain value $v\in V_a$ **do begin** compute $\partial_{(a,v)}$; store $((a,v), \partial_{(a,v)})$ in Γ **endfor**;
/* initialize 1 attribute candidates */
$\mathcal{R}_1 = \{\{a\} | a\in AT\}$; $\mathcal{A}_1 = \{\}$; $\mathcal{NSR} = \{\}$; // conditional attributes are candidates for *ag*-reducts
for each A in $\mathcal{R}_1$ **do** $A.id = 1$; // the evaluation of candidate A should start from object 1 in DT'
/* search reducts */
for ($k = 1$; $\mathcal{A}_k \neq \{\} \vee \mathcal{R}_k \neq \{\}$; k++) **do begin**
if $\mathcal{R}_k \neq \{\}$ **then begin**
/* find and move non-*ag*-reducts from $\mathcal{R}_k$ to $\mathcal{A}_k$ and determine maximal non-*ag*-super-reducts $\mathcal{NSR}$ */
EvaluateCandidates($\mathcal{R}_k$, $\mathcal{A}_k$, Γ, $\mathcal{NSR}$);
if $|\mathcal{NSR}_{|AT|}| = 1$ **then return** AT; // or equivalently, **if** $\mathcal{NSR}_{|AT|} = AT$ **then**
if $|\mathcal{NSR}_{|AT|-1}| = |AT|$ **then return** AT; // optional optimizing step 1
elseif $|\mathcal{NSR}| = 1$ **then return** $\cup_k \mathcal{R}_k$; // optional optimizing step 2
endif;
/* create k+1 attribute candidates $\mathcal{R}_{k+1}$ and non-*ag*-super-reducts $\mathcal{A}_{k+1}$ from $\mathcal{A}_k$ and $\mathcal{NSR}$ */
GRAGen($\mathcal{R}_{k+1}$, $\mathcal{A}_{k+1}$, $\mathcal{A}_k$, $\mathcal{NSR}$);
endfor;
return $\cup_k \mathcal{R}_k$;

A characteristic feature of our algorithm, which is shared by all *Apriori*-like algorithms (see [1] for the *Apriori* algorithm), is that the evaluation of candidates requires no more than n scans of the data set (decision table), where n is the length of a longest candidate (here: $n \leq |AT|$).

GRA, however, differs from *Apriori* in several ways. First of all, our candidates are sets of attributes instead of descriptors. Next, we evaluate candidates whether they satisfy property (**ag**), while the evaluation in *Apriori* consists in calculating the number of objects satisfying candidate descriptors. Additionally, our algorithm uses dynamically obtained information on non-*ag*-super-reducts to restrict the search space as quickly as possible. Another distinct optimizing feature of our algorithm is that the majority of candidates is evaluated against a fraction of the decision table instead of the entire decision table (see Section 7.2). Namely, having found that a candidate A does not satisfy the required property (**ag**) for some object x, the next objects are not considered for evaluating this candidate at all. In addition, the evaluation of candidates that are proper supersets of the invalidated candidate A starts from object x. These two optimizations may speed up the evaluation process considerably.

7.2 Evaluating Candidates for Approximate Reductions

The *EvaluateCandidates* procedure takes 4 arguments: k attribute candidates for ag-reducts $\mathcal{R}_k$, k attribute sets that are known not to be ag-super-reducts $\mathcal{A}_k$, the generalized decisions determined by atomic descriptors Γ, and known maximal non-ag-approximate-super-reducts $\mathcal{NSR}$. For each object read from *DT'*, the candidates in $\mathcal{R}_k$ that should be evaluated against this object are identified. These are candidates A such that $A.id$ equals the *identifier* of the object. Let x be the object under consideration and A be a candidate such that $A.id = x.identifier$. The upper bound ∂ on $\partial_A(x)$ is calculated from the generalized decisions determined by the atomic descriptors stored in Γ. If ∂ equals $x.\partial_{AT}$, then A satisfies property (**ag**) for object x and still has a chance to be an ag-reduct. Hence, $A.id$ is incremented to indicate that A should be evaluated against the next object after x in *DT'* too. Otherwise, if $\partial \neq x.\partial_{AT}$, then A is certainly not an ag-reduct and thus is moved from candidates $\mathcal{R}_k$ to non-ag-super-reducts $\mathcal{A}_k$. Additionally, the *MaximalNonAGSuperReduct* procedure (see Section 7.3) is called to determine a quasi maximal superset (*nsr*) of A that does not satisfy property (**ag**) for object x either. If *nsr* obtains the maximal possible length (i.e. $|nsr| = |AT|$), AT is returned as the maximal set the approximate generalized decision of which differs from the real AT-generalized decision, and the procedure stops. Otherwise, the found non-ag-super-reduct is stored in $\mathcal{NSR}$'. Since the evaluation of candidates against objects may result in moving all candidates from $\mathcal{R}_k$ to $\mathcal{A}_k$, scanning of *DT'* is stopped as soon as all candidates turned out false ones.

The last step of the *EvaluateCandidates* procedure consists in updating maximal non-ag-super-reducts $\mathcal{NSR}$ with $\mathcal{NSR}$'. Please note that k attribute sets are not stored in the final $\mathcal{NSR}$ since they are useless for identifying non-super-reducts among l attribute candidates, where $l > k$.

```
procedure EvaluateCandidates(var R_k, var A_k, in Γ, var NSR);
/* assert: Γ = ∪_{a∈AT, v∈Va} {{(a,v), ∂_(a,v)(a,v)}} */
   NSR' = {};
  for each object x in DT' do begin
     for each candidate A in R_k do
       if A.id = x.identifier then begin
         ∂ = ∩_{a∈A} ∂_(a, x.a);                          // note: each ((a, x.a), ∂_(a, x.a)) ∈ Γ
         if ∂ ≠ x.∂_AT then begin                         // or equivalently: if |∂| ≠ |x.∂_AT| then
           move A from R_k to A_k;
           nsr = MaximalNonAGSuperReduct(A, x, ∂, Γ);  // find a quasi maximal non-ag-super-reduct
           if nsr = AT then begin NSR = {AT}; return endif; // or equivalently: if |nsr| = |AT| then
           add nsr to NSR';
         else A.id = x.identifier + 1                     // A should be evaluated against the next object
         endif
       endif;
     if R_k = {} then break;
   endfor;
   NSR = MAX((NSR' \ NSR_k') ∪ (NSR \ NSR_k));
return;
```

7.3 Calculating Quasi Maximal Non-approximate Generalized Super-reducts

The *MaximalNonAGSuperReduct* function is called whenever a candidate, say A, does not satisfy property (**ag**) for some object x. This function returns a quasi maximal superset of A that does not satisfy property (**ag**) for x. Clearly, there may be many such supersets of A; however the function creates and evaluates supersets of A in a specific order. Namely, *nsr* variable, which initially equals A, is extended in each iteration with one attribute (assigned to variable a) that is next after the one recently added to *nsr*. Please note that the first attribute in AT is assumed to be next to the last attribute in AT. The creation of supersets stops when an evaluated attribute $nsr \cup \{a\}$ satisfies property (**ag**) for object x. Then, *MaximalNonAGSuperReduct* returns *nsr* as a known maximal superset of A, which is not an *ag*-super-reduct.

function *MaximalNonAGSuperReduct*(**in** A, **in** x, **in** ∂, **in** Γ);

/* assert: $\partial \neq x.\partial_{AT}$ */
  $nsr = A$; $\partial_{nsr} = \partial$; *previous_a* = last attribute in A;
  for (i=1; i <= $|AT|$; i++) **do**
    if *previous_a* = last attribute in AT **then** a = first attribute in AT
    else a = next attribute after *previous_a* in AT;
    previous_a = a; $\partial_{nsr} = \partial_{nsr} \cap \partial_{(a, x.a)}$; // note: each $((a, x.a), \partial_{(a, x.a)}) \in \Gamma$
    if $\partial_{nsr} = x.\partial_{AT}$ **then break else** add a to *nsr*;
  endfor;
return *nsr*;

7.4 Generating Candidates for Reductions

The *GRAGen* procedure differs from *RADGen* only in the pruning phase in that it determines the *id* field of each k+1 attribute candidate, say A, as a side effect of checking if A has all its k attribute subsets in $\mathcal{A}_k$. Namely, $A.id$ is assigned the maximum of the *id* fields of A's subsets in $\mathcal{A}_k$. Such value of $A.id$ field means that there was a subset of A in $\mathcal{A}_k$ that satisfied property (**ag**) for $A.id$-1 objects. Hence, A is known a priori to satisfy this property for $A.id$-1 objects and the first object against which it should be evaluated has identifier equal to $A.id$.

procedure *GRAGen*(**var** $\mathcal{R}_{k+1}$, **var** $\mathcal{A}_{k+1}$, **in** $\mathcal{A}_k$, **in** $\mathcal{NSR}$);

  forall $B, C \in \mathcal{A}_k$ **do** /* Merging */
    if $B[1] = C[1] \wedge ... \wedge B[k\text{-}1] = C[k\text{-}1] \wedge B[k] < C[k]$ **then begin**
      $A = B[1] \bullet B[2] \bullet ... \bullet B[k] \bullet C[k]$; add A to $\mathcal{R}_{k+1}$; $A.id = 1$;
    endif;
  forall $A \in \mathcal{R}_{k+1}$ **do** /* Pruning */
    forall k attribute sets $B \subset A$ **do**
      if $B \in \mathcal{A}_k$ **then** $A.id = \max(A.id, B.id)$
      else delete A from $\mathcal{R}_{k+1}$; // A is a proper superset of super-reduct B
  forall $B \in \mathcal{NSR}$ **do** move subsets of B from $\mathcal{R}_{k+1}$ to $\mathcal{A}_{k+1}$; /* Removing subsets of non-super-reducts */
return;

Example 3. We will illustrate *GRAGen* by showing how the candidates of size 3 are created. Let $\mathcal{A}_2 = \{\{ab\}_{[id:2]}, \{ac\}_{[id:3]}, \{ae\}_{[id:2]}, \{bc\}_{[id:3]}, \{bd\}_{[id:2]}, \{be\}_{[id:3]},$

$\{ce\}_{[id:3]}$, $\{de\}_{[id:2]}\}$ (the indices provide information on identifiers of objects recently evaluated for respective attribute sets) and $\mathcal{NSR} = \{\{abce\}, \{bde\}\}$.

The first phase of the procedure consists in creating candidates $\mathcal{R}_3$ from pairs of sets in $\mathcal{A}_2$ that differ only in their final attributes. Thus, we receive the following candidates: $\mathcal{R}_3 = \{\{abc\}_{[id:1]}, \{abe\}_{[id:1]}, \{ace\}_{[id:1]}, \{bcd\}_{[id:1]}, \{bce\}_{[id:1]}, \{bde\}_{[id:1]}\}$.

The pruning phase deletes these candidates from $\mathcal{R}_3$ that do not have at least one of their 2 attribute subsets in $\mathcal{A}_2$. In addition, the field *id* of each candidate is set to maximum of *id* values of all proper subsets of the candidates in $\mathcal{A}_2$. The only candidate in $\mathcal{R}_3$ that does not have some of its 2 attribute subsets in $\mathcal{A}_2$ is $\{bcd\}$. Namely, $\{cd\}$ is such a subset of $\{bcd\}$, which does not belong to $\mathcal{A}_2$. The fact that $\{cd\} \notin \mathcal{A}_2$ means that $\{cd\}$ is an *ag*-super-reduct. Hence, $\{bcd\}$ is known a priori to be a proper superset of an *ag*-reduct. Thus, this candidate is pruned from candidates $\mathcal{R}_3$. As a result, $\mathcal{R}_3$ becomes equal to $\{\{abc\}_{[id:3]}, \{abe\}_{[id:3]}, \{ace\}_{[id:3]}, \{bce\}_{[id:3]}, \{bde\}_{[id:3]}\}$.

The final phase determines candidates that are certainly not *ag*-reducts as they are subsets of previously found non-*ag*-super-reducts $\mathcal{NSR}$. Such candidates are moved from $\mathcal{R}_3$ to $\mathcal{A}_3$. Eventually, $\mathcal{A}_3 = \{\{abc\}_{[id:3]}, \{abe\}_{[id:3]}, \{ace\}_{[id:3]}, \{bce\}_{[id:3]}, \{bde\}_{[id:3]}\}$ and $\mathcal{R}_3 = \{\}$. Hence, no 3 attribute candidates should be evaluated against the decision table. □

7.5 Illustration of *GRA*

In this section, we illustrate the discovery of *ag*-reducts for *DT* from Table 1. We assume that the reduced decision table *DT'* (see Table 2) has already been determined. Table 6 shows how candidates change in each iteration before and after validation (if any) against *DT'*. In this process, the reduced decision table was scanned twice in order to evaluate the candidate sets. Only 8 candidates were evaluated against *DT'*, although 21 attribute sets were enumerated (that is, occurred in $\mathcal{R}$ or $\mathcal{A}$). As a result, 2 approximate *ag*-reducts were found; namely, $\{ad\}$ and $\{cd\}$, which are exact *g*-reducts (see Section 3.6).

Table 6. $\mathcal{R}_k$, $\mathcal{A}_k$, and $\mathcal{NSR}$ in subsequent iterations of *GRA*.

k	$\mathcal{R}_k$ before validation	$\mathcal{A}_k$ before validation	$\mathcal{R}_k$ after validation	$\mathcal{A}_k$ after validation	$\mathcal{NSR}'$	$\mathcal{NSR}$
1	$\{a\}_{[id:1]}$, $\{b\}_{[id:1]}$, $\{c\}_{[id:1]}$, $\{d\}_{[id:1]}$, $\{e\}_{[id:1]}$			$\{a\}_{[id:2]}$, $\{b\}_{[id:1]}$, $\{c\}_{[id:3]}$, $\{d\}_{[id:2]}$, $\{e\}_{[id:2]}$	$\{abc\}$, $\{b\}$, $\{c\}$, $\{de\}$, $\{abce\}$	$\{abce\}$, $\{de\}$
2	$\{ad\}_{[id:2]}$, $\{bd\}_{[id:2]}$, $\{cd\}_{[id:3]}$	$\{ab\}_{[id:2]}$, $\{ac\}_{[id:3]}$, $\{ae\}_{[id:2]}$, $\{bc\}_{[id:3]}$, $\{be\}_{[id:2]}$, $\{ce\}_{[id:3]}$, $\{de\}_{[id:2]}\}$	$\{ad\}_{[id:8]}$, $\{cd\}_{[id:8]}$	$\{ab\}_{[id:2]}$, $\{ac\}_{[id:3]}$, $\{ae\}_{[id:2]}$, $\{bc\}_{[id:3]}$, $\{bd\}_{[id:2]}$, $\{be\}_{[id:3]}$, $\{ce\}_{[id:3]}$, $\{de\}_{[id:2]}$	$\{bde\}$	$\{abce\}$, $\{bde\}$
3		$\{abc\}_{[id:3]}$, $\{abe\}_{[id:3]}$, $\{ace\}_{[id:3]}$, $\{bce\}_{[id:3]}$, $\{bde\}_{[id:3]}$				$\{abce\}$
4		$\{abce\}_{[id:3]}$				

8 Core-Oriented Discovery of Approximate Generalized Reducts

8.1 Main Algorithm

Algorithm. *CoreGRA*;

DT' = *GenDecRepresentation-of-DT*(*DT*);
for each conditional attribute $a \in AT$ **do**
  for each domain value $v \in V_a$ **do begin** compute $\partial_{(a,v)}$; store $((a,v), \partial_{(a,v)})$ in Γ **endfor**;
/*initialize 1 attribute candidates */
$\mathcal{R}_1 = \{\{a\} | a \in AT\}$; $\mathcal{A}_1 = \{\}$; $\mathcal{NSR} = \{\}$; // conditional attributes are candidates for *ag*-reducts
for each A in $\mathcal{R}_1$ **do begin** $A.id = 1$; $A.nsr = A$ **endfor**;
/* find and move non-reducts from $\mathcal{R}_1$ to $\mathcal{A}_1$; determine maximal non-reducts */
EvaluateCandidates1($\mathcal{R}_1$, $\mathcal{A}_1$, Γ, $\mathcal{NSR}$);
if $|\mathcal{NSR}_{|AT|}| = 1$ **then return** (AT, AT); // or equivalently, **if** $\mathcal{NSR}_{|AT|} = AT$ **then**
/* determine core */
$core = \emptyset$; $core.id = 1$;
forall $A \in \mathcal{NSR}_{|AT|-1}$ **do begin** $\{a\} = AT \setminus A$; $core = core \cup \{a\}$; $core.id = \max(core.id, \{a\}.id)$ **endfor**;
/* create candidate for reducts as core's supersets */
if $core = \emptyset$ **then begin**
  startLevel = 2;
  ReductsAprioriGen($\mathcal{R}_2$, $\mathcal{A}_2$, $\mathcal{A}_1$, $\mathcal{NSR}$); //create 2 attribute candidates from 1 attribute non-*ag*-reducts
else begin
  $\mathcal{NSR} = \{B \in \mathcal{NSR} | B \supseteq core\}$; // *ag*-reducts are supersets of *ag*-core
  if $|core| > 1$ **then**
    if $\mathcal{NSR} \neq \{\}$ **then** // *ag*-core is not an *ag*-reduct as there is its superset in $\mathcal{NSR}$
      $\mathcal{NSR} = \mathcal{NSR} \setminus \mathcal{NSR}_{|core|}$ // or equivalently $\mathcal{NSR} = \mathcal{NSR} \setminus \{core\}$;
    else begin
      $\mathcal{R}_{|core|} = \{core\}$; $\mathcal{A}_{|core|} = \{\}$; *EvaluateCandidates*($\mathcal{R}_{|core|}$, $\mathcal{A}_{|core|}$, Γ, $\mathcal{NSR}$);
      if $|\mathcal{NSR}_{|AT|}| = 1$ **then return** (AT, AT);
    endif;
  if $\mathcal{R}_{|core|} = \{core\}$ **then return**($core$, $\mathcal{R}_{|core|}$) // or equivalently **if** $|\mathcal{R}_{|core|}| = 1$ **then**
  else begin
    $startLevel = |core| + 1$; $\mathcal{R}_{startLevel} = \{\}$; $\mathcal{A}_{startLevel} = \{\}$;
    forall $\{a\} \in \mathcal{A}_1$ such that $a \notin core$ **do begin**
      $A = core \cup \{a\}$; $A.id = \max(core.id, \{a\}.id)$; // candidates should contain *ag*-core
      $\mathcal{R}_{startLevel} = \mathcal{R}_{startLevel} \cup \{A\}$
    endfor;
    forall $B \in \mathcal{NSR}$ **do** move subsets of B from $\mathcal{R}_{startLevel}$ to $\mathcal{A}_{startLevel}$;
  endif
endif;
for ($k = startLevel$; $\mathcal{A}_k \neq \{\} \vee \mathcal{R}_k \neq \{\}$; k++) **do begin** /* *ag*-reducts' regular search */
  if $\mathcal{R}_k \neq \{\}$ **then begin**
  /* find and move non-*ag*-reducts from $\mathcal{R}_k$ to $\mathcal{A}_k$ and determine maximal non-*ag*-super-reducts $\mathcal{NSR}$ */
    EvaluateCandidates($\mathcal{R}_k$, $\mathcal{A}_k$, Γ, $\mathcal{NSR}$);
    if $|\mathcal{NSR}_{|AT|}| = 1$ **then return** (AT, AT) **endif;**
  elseif $|\mathcal{NSR}| = 1$ **then return** ($core$; $\cup_k \mathcal{R}_k$); // optional optimizing step
  endif;
  GRAGen($\mathcal{R}_{k+1}$, $\mathcal{A}_{k+1}$, $\mathcal{A}_k$, $\mathcal{NSR}$); // create (k+1)-candidates from k attribute non-*ag*-reducts
endfor;
return ($core$; $\cup_k \mathcal{R}_k$);

The *CoreGRA* algorithm, we propose, finds not only *ag*-reducts, but also their core. The layout of *CoreGRA* reminds that of *GRA*. *CoreGRA*, however, differs from *GRA*, in that it evaluates 1 attribute candidates in special way that provides sufficient information to determine the *ag*-core, and next creates subsequent candidates only as supersets of the found *ag*-core. *CoreGRA* calls the *EvaluateCandidate1* procedure (see Section 8.2) in order to evaluate 1 attribute candidates. Unlike the *EvaluateCandidate* procedure, *EvaluateCandidate1* guarantees that all maximal $|AT|$-1 non-*ag*-super-reducts will be determined and returned in $\mathcal{NSR}$. Using this information, the *ag*-core will then be calculated according to its definition.

If the *ag*-core is an empty set, then 2 attribute and longer candidates are created and evaluated as in *GRA*. Otherwise, all sets in $\mathcal{NSR}$ that are not supersets of the *ag*-core are deleted, since the only candidates considered in *CoreGRA* will be the *ag*-core and its supersets. If the *ag*-core contains only one attribute, it is not evaluated because singleton attributes were already evaluated. The *ag*-core is not evaluated also in the case, when $\mathcal{NSR}$, already restricted to non-*ag*-super-reducts being the core's supersets, is not empty. In this case, the *ag*-core is also a non-*ag*-super-reduct as a subset of some non-*ag*-super-reduct in $\mathcal{NSR}$. Otherwise, the *ag*-core is evaluated. Provided the *ag*-core is found an *ag*-reduct, it is returned as the only *ag*-reduct. If the *ag*-core is not a reduct, the new candidates $\mathcal{R}_{|core|+1}$ are created by merging the core with the remaining attributes in *AT*. Clearly, the new candidates which have supersets in maximal known non-*ag*-super-reducts $\mathcal{NSR}$, are not *ag*-reducts either, and hence are moved from $\mathcal{R}_{|core|+1}$ to $\mathcal{A}_{|core|+1}$. From now on, *CoreGRA* is performed in the same way as *GRA*.

It is expected that *CoreGRA* should perform better than *GRA*, when the *ag*-core consists of a sufficient number of attributes. Then fewer iterations should be performed and probably fewer candidates will be evaluated. Nevertheless, when the number of attributes in the *ag*-core is small, *CoreGRA* may be less effective than *GRA* because of the more exhaustive evaluation of 1 attribute candidates (their *nsr* fields are likely to be evaluated against the entire decision table in *CoreGRA*).

8.2 Evaluating Singleton Candidates

Below we describe the *EvaluateCandidates1* procedure, which is primarily intended to be applied only to 1 attribute candidates in *CoreGRA*, although it can be applied for evaluating candidates of any length. It is assumed that an additional field *nsr* is associated with each *k* attribute candidate *A* in $\mathcal{R}_k$.

The *EvaluateCandidates1* procedure differs from *EvaluateCandidates* in that after discovering that a candidate *A* is not an *ag*-reduct, it is not removed from $\mathcal{R}_k$ immediately. Nevertheless, *EvaluateCandidates1* stops advancing *A.id* field as soon as the first object invalidating *A* is found (like *EvaluateCandidates* does). In such a case, instead of evaluating *A*, its *nsr* field is extended and evaluated against the remaining objects in the decision table as long as *nsr* obtains the maximal possible length (i.e. $|nsr| = |AT|$) or the end of the decision table is reached. In the former case, *AT* is returned as the maximal set the approximate generalized decision of which differs from

the real *AT*-generalized decision, and the procedure stops. In the latter case, the remaining candidates A in $\mathcal{R}_k$ that turned out not *ag*-reducts (i.e. such that $A.id \neq |DT|+1$), are moved to $\mathcal{A}_k$ and $\mathcal{NSR}'$ is updated with their *nsr* fields.

procedure *EvaluateCandidates1*(**var** $\mathcal{R}_k$, **var** $\mathcal{A}_k$, **in** Γ, **var** $\mathcal{NSR}$);

```
NSR' = {};
for each object x in DT do begin
  for each candidate A in R_k do begin
    ∂ = ∩_{a∈A.nsr} ∂_{(a,x.a)};                    // note: each ((a,x.a), ∂_{(a,x.a)}) ∈ Γ
    if ∂ ≠ x.∂_AT then begin                        // or equivalently: if |∂| = |x.∂_AT| then
      A.nsr = MaximalNonAGSuperReduct(A.nsr, x, ∂, Γ);   // find a maximal non-ag-super-reduct
      if A.nsr = AT then begin NSR = {AT}; return endif  // or equivalently: if |A.nsr| = |AT| then
    elseif A.id = x.identifier then A.id = x.identifier + 1   // evaluate A's supersets against the next object
    endif;
  endfor;
  if R_k = {} then break;
endfor;
for each candidate A in R_k do
  if A.id ≠ |DT|+1 then move A from R_k to A_k;  add A.nsr to NSR' endif;      // A is not an ag-reduct
  NSR = MAX(NSR' \ NSR_k');// NSR = MAX((NSR' \ NSR_k') ∪ (NSR \ NSR_k)) for k > 1
return;
```

8.3 Illustration of *CoreGRA*

In this section, we illustrate how *CoreGRA* searches *ag*-reducts in the decision table *DT* from Table 1. Table 7 shows how candidates change in each iteration before and after validation against the reduced decision table *DT'* from Table 2.

After 1 attribute candidates were evaluated by *EvaluateCandidates1*, $\mathcal{NSR}$ became equal to $\{\{abce\}, \{de\}\}$. Thus, $\{abce\}$ was the only set in $\mathcal{NSR}$ the length of which was equal to $|AT|$-1. Hence, the *ag*-core was determined as $AT\backslash\{abce\} = \{d\}$. Since the new candidates were to be supersets of the *ag*-core, all sets from $\mathcal{NSR}$ that were not supersets of the *ag*-core were pruned and $\mathcal{NSR}$ became equal to $\{\{de\}\}$. The *ag*-core $\{d\}$ is not an *ag*-reduct, as it was not present in the set of the positively evaluated candidates $\mathcal{R}_1$ (here: $\mathcal{R}_1 = \varnothing$).

New candidates were created by merging the *ag*-core with the remaining attributes in *AT* resulting in the following 4 attribute candidates: $\{ad\}$, $\{bd\}$, $\{cd\}$, $\{de\}$. One of them ($\{de\}$) was known a priori not to be an *ag*-reduct as a subset of the known non-*ag*-super-reduct $\{de\}$ in $\mathcal{NSR}$. From now on, *CoreGRA* proceeded as *GRA*. The execution of the *CoreGRA* algorithm resulted in enumeration of 9 attribute sets instead of 21 (see Section 7.5).

Table 7. $\mathcal{R}_k$, $\mathcal{A}_k$, and $\mathcal{NSR}$ in subsequent iterations of *CoreGRA*.

k	$\mathcal{R}_k$ before validation	$\mathcal{A}_k$ before validation	$\mathcal{R}_k$ after validation	$\mathcal{A}_k$ after validation	$\mathcal{NSR}'$	$\mathcal{NSR}$
1	$\{a\}_{[id:1]}$, $\{b\}_{[id:1]}$, $\{c\}_{[id:1]}$, $\{d\}_{[id:1]}$, $\{e\}_{[id:1]}$			$\{a\}_{[id:2]}$, $\{b\}_{[id:1]}$, $\{c\}_{[id:3]}$, $\{d\}_{[id:2]}$, $\{e\}_{[id:2]}$	$\{abc\}$, $\{bc\}$, $\{c\}$, $\{de\}$, $\{abce\}$	~~$\{abce\}$~~, $\{de\}$
2	$\{ad\}_{[id:2]}$, $\{bd\}_{[id:2]}$, $\{cd\}_{[id:3]}$	$\{de\}_{[id:2]}$	$\{ad\}_{[id:8]}$, $\{cd\}_{[id:8]}$	$\{bd\}_{[id:2]}$, $\{de\}_{[id:2]}$	$\{bde\}$	$\{bde\}$

9 Discovering Approximate Certain Reducts

Approximate certain reducts of *DT* are defined by means of generalized decisions only of objects in *DT* with singleton *AT*-generalized decisions. This observation suggests that the *GRA* and *CoreGRA* algorithms shall calculate *ac*-reducts of *DT* correctly, provided the candidate attribute sets are evaluated only against the objects in *DT* with singleton *AT*-generalized decisions. This can be achieved in two ways:

a) either the initialization of candidates in the *GRA* procedure should be preceded by an additional operation that removes all objects from *DT* (or *DT'*) that have non-singleton *AT*-generalized decisions and renumbers the remaining objects;
b) or the evaluation of candidates should be modified so that to ignore objects with non-singleton *AT*-generalized decisions safely (please, see [13]).

10 Conclusion

In the article, we have offered two new algorithms: *RAD* and *CoreRAD* for discovering all exact generalized (and by this also possible) and certain reducts from decision tables. In addition, *CoreRAD* determines the core. Both algorithms require the calculation of all maximal attribute sets $\mathcal{MNSR}$ that are not super-reducts. An *Apriori*-like method of determining reducts based on $\mathcal{MNSR}$ was proposed. Our method of determining $\mathcal{MNSR}$ is orthogonal to the methods that determine a discernibility matrix ($\mathcal{DM}$), which stores information on sets of attributes each of which discerns at least one pair of objects that should be discerned, and return the family of all such minimal sets ($\mathcal{MDM}$). The reducts are then found from $\mathcal{MDM}$ by applying Boolean reasoning.

The calculation of $\mathcal{MNSR}$ (as well as $\mathcal{MDM}$) requires comparing each pair of objects in the decision table and finding maximal (minimal) attribute sets among those that are the result of the objects' comparison. This operation is very costly when the number of objects in a decision table is large. In order to overcome this problem one may use a reduced table $(AT, \{\partial_{AT}\})$, which stores one object instead of many original objects that are indiscernible on *AT* and ∂_{AT}. Nevertheless, when the number of objects in the reduced table is still large or the number of $\mathcal{MNSR}$ ($\mathcal{MDM}$) is large, the calculation of reducts may be infeasible. Our preliminary experiments indicate that the determination of $\mathcal{MNSR}$ is a bottleneck of the proposed *RAD*-like algorithms in such cases. To the contrary, the proposed *Apriori*-like method of determining reducts based on $\mathcal{MNSR}$ is very efficient.

In the case, when the determination of $\mathcal{MNSR}$ is infeasible, we advocate to search approximate reducts. In the article, we have defined such approximate reducts based on the properties of a generalized decision function. We have shown that for each *A*-generalized decision one may derive its upper bound (*A*-approximate generalized decision) from elementary *a*-generalized decisions, where $a \in A$. Whereas exact generalized (certain) reducts preserve the *AT*-generalized decision for all objects (for objects with singleton generalized decisions), each approximate generalized (certain) reduct *A* guarantees that *A*-approximate generalized decision is equal to the

AT-generalized decision for all objects (for objects with singleton generalized decisions). An exception to the rule is the case, when there is an object for which the approximate *AT*-generalized decision differs from the actual *AT*-generalized decision. Then the entire set of conditional attributes *AT* is defined as a reduct. We have proved that approximate generalized and certain reducts are supersets of exact reducts of respective types. In addition, approximate generalized reducts are supersets of exact possible reducts.

We have presented *GRA* and *CoreGRA* algorithms for discovering approximate generalized (and by this also possible) reducts and certain reducts from very large decision tables. The experiments we have carried out and reported in [13] prove that the *GRA*-like algorithms are scalable with respect to the number of objects in a decision table and that *CoreGRA* tends to outperform *GRA* with increasing number of conditional attributes. For a few conditional attributes, however, *GRA* may find reducts faster. Nevertheless, the experiments need to be continued to fully recognize the performance characteristics of particular *GRA*-like algorithms.

Finally, we note that all the proposed algorithms are capable to discover all discussed types of reducts from incomplete decision tables as well. The only difference consists in a slightly different determination of generalized decision value for atomic descriptors, namely $\partial_A(x) = \{d(y) \mid y \in S_A(x)\}$, where $S_A(x) = \{y \in O \mid \forall a \in A, (a(x) = a(y)) \vee (a(x) \text{ is NULL}) \vee (a(y) \text{ is NULL})\}$ (see e.g. [12]). In the future, we intend to develop scalable algorithms for discovering all exact reducts.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast Discovery of Association Rules. In: Advances in KDD. AAAI, Menlo Park, California (1996) 307-328
2. Bazan, J., Skowron, A., Synak, P.: Dynamic Reducts as a Tool for Extracting Laws from Decision Tables. In: Proc. of ISMIS '94, Charlotte, USA. LNAI, Vol. 869, Springer-Verlag, (1994) 346–355
3. Bazan, J., Nguyen, H.S., Nguyen, S.H., Synak, P., Wróblewski, J.: Rough Set Algorithms in Classification Problem. In: L. Polkowski, S. Tsumoto and T.Y. Lin (eds.): Rough Set Methods and Applications. Physica-Verlag, Heidelberg, New York (2000) 49 - 88
4. Jelonek, J., Krawiec, K., Stefanowski, J.: Comparative Study of Feature Subset Selection Techniques for Machine Learning Tasks. Proc. of IIS '98, Malbork, Poland (1998) 68–77
5. John, H.G., Kohavi, R., Pfleger, K.: Irrelevant Features and the Subset Selection Problem. In: Machine Learning: Proc. of the Eleventh International Conference, Morgan Kaufmann Publishers, San Francisco, CA, (1994) 121–129
6. Kohavi, R., Frasca, B.: Useful Feature Subsets and Rough Set Reducts. In: Proc. of the Third International Workshop on Rough Sets and Soft Computing, San Jose, CA (1994)
7. Kryszkiewicz, M.: The Algorithms of Knowledge Reduction in Information Systems, Ph.D. Thesis, Warsaw University of Technology, Institute of Computer Science (1994)
8. Kryszkiewicz, M., Rybinski, H.: Finding Reducts in Composed Information Systems. Fundamenta Informaticae Vol. 27, No. 2–3 (1996) 183–196
9. Kryszkiewicz, M.: Strong Rules in Large Databases. In: Proc. of IPMU' 98, Paris, France, Vol. 2 (1998) 1520–1527
10. Kryszkiewicz M., Rybinski H.: Knowledge Discovery from Large Databases using Rough Sets. In: Proc. of EUFIT '98, Aachen, Germany, Vol. 1 (1998) 85-89

11. Kryszkiewicz, M.: Comparative Study of Alternative Types of Knowledge Reduction in Inconsistent Systems. International Journal of Intelligent Systems, Wiley, Vol. 16, No. 1 (2001) 105–120
12. Kryszkiewicz, M.: Rough Set Approach to Rules Generation from Incomplete Information Systems. In: The Encyclopedia of Computer Science and Technology. Marcel Dekker, Inc., New York, Vol. 44 (2001) 319-346
13. Kryszkiewicz, M., Cichoń K.: Scalable Methods of Discovering Rough Sets Reducts. ICS Research Report 28/2003, Warsaw University of Technology (2003)
14. Lin, T.Y.: Rough Set Theory in Very Large Databases. In: Proc. of CESA IMACS '96, Lille, France Vol. 2 (1996) 936-941
15. Modrzejewski, M.: Feature Selection using Rough Sets Theory. In: Proc. of the European Conference on Machine Learning (1993) 213–226
16. Nguyen, S.H., Skowron, A., Synak, P., Wróblewski, J.: Knowledge Discovery in Databases: Rough Set Approach. In: Proc. of IFSA '97, Prague, Vol. II (1997) 204-209
17. Pawlak, Z.: Rough Sets: Theoretical Aspects of Reasoning about Data, Kluwer Academic Publishers, Vol. 9 (1991)
18. Pawlak, Z., Skowron, A.: A Rough Set Approach to Decision Rules Generation, ICS Research Report 23/93, Warsaw University of Technology (1993)
19. Romanski, S., Operations on Families of Sets for Exhaustive Search, Given a Monotonic Boolean Function. In: Proc. of Intl' Conf. on Data and Knowledge Bases, Israel (1988)
20. Skowron, A., Rauszer, C.: The Discernibility Matrices and Functions in Information Systems. In: Intelligent Decision Support: Handbook of Applications and Advances of Rough Sets Theory. Kluwer Academic Publishers (1992) 331-362
21. Skowron, A., Swiniarski, R.W.: Information Granulation and Pattern Recognition. In: S.K. Pal, L. Polkowski, A. Skowron (eds.): Rough-Neural Computing. Techniques for Computing with Words. Heidelberg: Springer-Verlag (2004)
22. Slezak, D.: Approximate Reducts in Decision Tables. In: Proc. of IPMU '96, Granada, Spain, Vol. 3 (1996) 1159-1164
23. Slezak, D.: Searching for Frequential Reducts in Decision Tables with Uncertain Objects. In: Proc. of RSCTC '98, Warsaw. Springer-Verlag, Berlin (1998) 52–59
24. Slowiński, R. (ed.): Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory. Kluwer Academic Publishers, Vol 11 (1992)
25. Stepaniuk, J.: Approximation Spaces, Reducts and Representatives. In: Skowron, A., Polkowski, L. (eds.): Rough Sets in Data Mining and Knowledge Discovery, Springer-Verlag, Berlin (1998)
26. Susmaga, R.: Experiments in Incremental Computation of Reducts. In: Skowron, A., Polkowski, L., (eds.): Rough Sets in Data Mining and Knowledge Discovery, Springer-Verlag, Berlin (1998)
27. Susmaga, R.: Parallel Computation of Reducts. In: Proc. of RSCTC '98, Warsaw. Springer-Verlag, Berlin (1998) 450–457
28. Susmaga, R.: Computation of Shortest Reducts. In: Foundations of Computing and Decision Sciences, Poznan, Poland, Vol. 2, No. 23 (1998)
29. Susmaga, R.: Effective Tests for Inclusion Minimality in Reduct Generation. In: Foundations of Computing and Decision Sciences, Vol. 4, No. 23 (1998) 219–240
30. Tannhäuser, M.: Efficient Reduct Computation. M.Sc. Thesis, Institute of Mathematics, Warsaw University, Warsaw (1994)
31. Wroblewski, J.: Finding Minimal Reducts Using Genetic Algorithms. In: Proc. of the 2nd Annual Join Conference on Information Sc., Wrightsville Beach, NC, (1995) 186–189