Chapter 2

Rough Set Algorithms in Classification Problem

Jan G. Bazan\textsuperscript{1}, Hung Son Nguyen\textsuperscript{2,3}, Sinh Hoa Nguyen\textsuperscript{2,3}, Piotr Synak\textsuperscript{3}, Jakub Wróblewski\textsuperscript{3,2}

\textsuperscript{1} Institute of Mathematics, Pedagogical University of Rzeszów
Rejtana 16A, 35-310 Rzeszów, Poland
\textsuperscript{2} Institute of Mathematics, Warsaw University
Baracha 2, 02-097 Warsaw, Poland
\textsuperscript{3} Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-088 Warsaw, Poland

Abstract: In this Chapter we present some algorithms, based on rough set theory, that can be used for the problem of new cases classification. Most of the algorithms were implemented and included in Rosetta system [43]. We present several methods for computation of decision rules based on reducts. We discuss the problem of real value attribute discretization for increasing the performance of algorithms and quality of decision rules. Finally we deal with a problem of resolving conflicts between decision rules classifying a new case to different categories (classes).

Keywords: knowledge discovery, rough sets, classification algorithms, reducts, decision rules, real value attribute discretization

1 Introduction

The term "classification" concerns any context in which some decision is taken or a forecast is made on the basis of currently available knowledge or information. A classification algorithm is an algorithm which permits us to repeatedly make a forecast on the basis of accumulated knowledge in new situations. We consider here a classification related to construction of a classification algorithm on the basis of current knowledge. Such algorithm is applied then to classify objects previously unseen. Each new object is assigned to a class belonging to a predefined set of classes on the basis of observed values of suitably chosen attributes (features).

Many approaches have been proposed for constructing classification algorithms, among them we would like to point out classical and modern statistical techniques (see e.g. [33], [20]), neural networks (see e.g. [33], [19]), decision trees (see e.g. [13], [48], [58], [60], [33], [49]), decision rules (see e.g. [14], [31], [59], [10], [32], [61], [22], [55], [34], [43]), inductive logic programming (see e.g. [17]).

In this Chapter we present some methods for extracting laws from data, based on rough set approach (see [44]) and Boolean reasoning (see [11]). Most of them were implemented and included in Rosetta system [43]. Results of performed
experiments on different kinds of data show that they are very promising (see e.g. [3], [4], [5], [7], [8]).

Standard rough set methods (see [44], [45], [52]) are not always sufficient for extracting laws from data. One of the reasons is that these methods are not taking into account the fact that part of the reduct set (see Section 2) is chaotic, i.e. is not stable in randomly chosen samples of a given decision table. We propose a method for selection of feature (attribute) sets relevant for extracting laws from data. These sets of attributes are called dynamic reducts (see [4]). Dynamic reducts are in some sense the most stable reducts of a given decision table, i.e. they are the most frequently appearing reducts in subtables created by random samples of a given decision table.

The most popular method for classification algorithms construction is based on learning rules from examples. The methods based on calculation of all reducts allow to compute, for a given data, the descriptions of concepts by means of decision rules (see [45], [44]). Unfortunately, the decision rules constructed in this way can often be not appropriate to classify unseen cases. We propose a method of the decision rule generation on the basis of dynamic reducts. We suggest that the rules calculated by means of dynamic reducts are better predisposed to classify unseen cases (see [3], [4], [5], [6]).

The Chapter is structured as follows. In Section 2 we present rough set preliminaries. Standard rough set algorithms for synthesis of decision rules from decision tables are described in Section 3.

For purpose of data preprocessing we propose methods for intelligent scaling of real value attributes (see Section 4). Our algorithms are especially useful if the input data table contains continues attributes with many different values. Applying discretization to data decreases further processing time of many methods as well as increases quality of results, e.g. decision rules containing scaled attributes are more general.

Our methods for decision rule generation are based on algorithm for the reduct set computation (see Section 2). Unfortunately, the searching problem for reduct of minimal length (minimal number of attributes) is NP-hard (see [51]). Therefore we often apply approximation algorithms to obtain some knowledge about the reduct set (see [38], [62]). In Section 5 we propose some heuristic for finding local reducts. In Section 6 we present a genetic algorithm for reduct set computation which is very fast and gives its good approximation.

In Sections 7, 8, 9, and 10 we recall the notion of a dynamic reduct and we give some statistical arguments showing that dynamic reducts offer a good tool for extracting laws from decision table.

Some applications of dynamic reducts, e.g. for decision rules generation and dynamic selection of cuts in discretization, are presented in Sections 4, 11 and 12.

In Section 13 we show how the idea of dynamic reducts can be adapted for new methods of dynamic rules computation.

Sometimes the decision rules generated by application of rough set methods can be not acceptable as laws valid for data encoded in a given decision table. This occurs, e.g. when the number of examples supporting the decision rule (see
Section 3) is relatively small. In Section 14 we introduce approximate rules to eliminate this drawback. Different methods (e.g. [1], [35], [46], [67]) are now widely used to generate approximate decision rules.

When a set of decision rules has been computed then it is necessary to decide how to resolve conflicts between rule sets classifying tested objects to different decision classes. In Section 15 we present several measures of the strength of rule set matched by a given tested object and classifying this object to decision determined by the rules from this set.

In Section 16 we present a general scheme of classification algorithms based on methods and techniques described in previous sections.

2 Rough Set Preliminaries

In this section we recall some basic notions related to information systems and rough sets.

An information system is a pair $A = (U, A)$, where $U$ is a non-empty, finite set called the universe and $A = \{a_1, \ldots, a_n\}$ a non-empty, finite set of attributes, i.e. $a : U \rightarrow V_a$ for $a \in A$, where $V_a$ is called the value set of $a$.

Elements of $U$ are called objects and interpreted as, e.g. cases, states, processes, patients, observations. Attributes are interpreted as features, variables, characteristic conditions etc.

We also consider a special case of information systems called decision tables. A decision table is an information system of the form $A = (U, A \cup \{d\})$, where $d \notin A$ is a distinguished attribute called decision. The elements of $A$ are called conditions.

One can interpret the decision attribute as a kind of classifier on the universe of objects given by an expert, a decision-maker, an operator, a physician, etc. In machine learning decision tables are called training sets of examples (see [28]).

The cardinality of the image $d(U) = \{k : d(s) = k \text{ for some } s \in U\}$ is called the rank of $d$ and is denoted by $r(d)$.

We assume that the set $V_d$ of values of the decision $d$ is equal to $\{v^1_d, \ldots, v^{r(d)}_d\}$.

Let us observe that the decision $d$ determines a partition $CLASS_A(d) = \{X^{i}_{\Lambda} \} \cup \{X^{r(d)}_{\Lambda}\}$ of the universe $U$, where $X^{i}_{\Lambda} = \{x \in U : d(x) = v^i_d\}$ for $1 \leq i \leq r(d)$. $CLASS_A(d)$ is called the classification of objects in $A$ determined by the decision $d$. The set $X^{i}_{\Lambda}$ is called the $i$-th decision class of $A$. By $X^{i}_{\Lambda}(u)$ we denote the decision class $\{x \in U : d(x) = v^i_d\}$ for any $u \in U$.

Let $A = (U, A)$ be an information system. For every set of attributes $B \subseteq A$, an equivalence relation, denoted by $IND_A(B)$ and called the $B$-indiscernibility relation, is defined by

$$IND_A(B) = \{(u, u') \in U^2 : \text{ for every } a \in B, a(u) = a(u')\} \quad (1)$$

Objects $u, u'$ satisfying the relation $IND_A(B)$ are indiscernible by attributes from $B$.

An attribute $a \in B \subseteq A$ is dispensable in $B$ if $IND_A(B) = IND_A(B \setminus \{a\})$, otherwise $a$ is indispensable in $B$. A set $B \subseteq A$ is independent in $A$ if every
attribute from $B$ is indispensable in $B$, otherwise the set $B$ is dependent in $A$. A set $B \subseteq A$ is called a reduct in $A$ if $B$ is independent in $A$ and $IND_A(B) = IND_A(A)$. The set of all reducts in $A$ is denoted by $RED(A)$. This is a classical notion of reduct and it is sometimes referred to as global reduct.

Let $A = (U, A)$ be an information system with $n$ objects. By $M(A)$ (see [51]) we denote an $n \times n$ matrix $(c_{ij})$, called the discernibility matrix of $A$ such that

$$c_{ij} = \{ a \in A : a(x_i) \neq a(x_j) \} \text{ for } i, j = 1, \ldots, n .$$

(2)

A discernibility function $f_A$ for an information system $A$ is a boolean function of $m$ boolean variables $\pi_1, \ldots, \pi_m$ corresponding to the attributes $a_1, \ldots, a_m$ respectively, and defined by

$$f_A(\pi_1, \ldots, \pi_m) = \bigwedge \{ \bigvee \pi_j : 1 \leq j \leq m, \pi_j \neq \emptyset \}$$

(3)

where $\pi_j = \{ a : a \in c_{ij} \}$.

It can be shown (see [51]) that the set of all prime implicants of $f_A$ determines the set of all reducts of $A$.

Below we present two deterministic algorithms (see [15]) for computation of the whole reduct set $RED_A(A)$. Both algorithms compute discernibility matrix of $A$.

**Algorithm 1. Reduct set computation**

**Input:**
Information system $A = (U, A)$

**Output:**
Set $RED_A(A)$ of all reducts of $A$

**Method:**
Compute indiscernibility matrix $M(A) = (c_{ij})$
Reduce $M$ using absorption laws
d - number of non-empty fields $C_1, C_2, \ldots, C_d$ of reduced $M$
Build families of sets $R_0, R_1, \ldots, R_d$ in the following way:
begin
$R_0 = \emptyset$
for $i = 1$ to $d$
begin
$R_i = S_i \cup T_i$, where $S_i = \{ R \in R_{i-1} : R \cap C_i \neq \emptyset \}$
and $T_i = (R \cup \{ a \})_{a \in C_i, R \in R_{i-1}, R \cap C_i \neq \emptyset}$
end
end
Remove dispensable attributes from each element of family $R_d$
Remove redundant elements from $R_d$
$RED_A(A) = R_d$

\[ \square \]

It is easy to see that the complexity of this algorithm is exponential. The second algorithm is a modification of Algorithm 1. It allows to stop computation and get a partially computed reduct set.
Algorithm 2. Reduct set computation with stop possibility

Input:
- Information system $A = (U, A)$

Output:
- Set $RED_A(A)$ of all reducts of $A$

Method:
- Compute indiscernibility matrix $M(A) = (C_{ij})$
- Reduce $M$ using absorption laws
- $d$ - number of non-empty fields of reduced $M$
- Build a families of sets $R_0, R_1, ..., R_d$ in the following way:

begin
- Compute $R_0$ (see Algorithm 1)
- $i = 1$
- while $i > 0$ do
  - if stop then return
  - if $R_i = \emptyset$ then
    - begin
      - $i = i - 1$
      - continue
    - end
  - Remove from family $R_i$ the first element
  - Compute $R_{i+1}$ (see Algorithm 1)
    - $i = i + 1$
  - if $i = d$ then
    - begin
      - Remove from $R_d$ redundant elements
      - $RED_A(A) = RED_A(A) \cup R_d$
      - $i = i - 1$
    - end
  - end
end

If $A = (U, A)$ is an information system, $B \subseteq A$ is a set of attributes and $X \subseteq U$ is a set of objects, then the sets: $\{u \in U : [u]_B \subseteq X\}$ and $\{u \in U : [u]_B \cap X \neq \emptyset\}$ are called the $B$-lower and the $B$-upper approximation of $X$ in $A$, and they are denoted by $\overline{B}X$ and $\overline{B}X$, respectively.

The set $BN_B(X) = \overline{B}X - \overline{B}X$ is called the $B$-boundary of $X$. When $B = A$ we also write $BN_A(X)$ instead of $BN_A(X)$.

Sets which are unions of some classes of the indiscernibility relation $IND_A(B)$ are called definable by $B$ (or, $B$-definable, in short). A set $X$ is thus $B$-definable iff $\overline{B}X = \overline{B}X$. Some subsets (categories) of objects in an information system cannot be exactly expressed by employing available attributes but they can be defined roughly.
The set $BX$ is the set of all elements of $U$ which can be classified with certainty as elements of $X$, having the knowledge about them represented by attributes from $B$; the set $BN_B(X)$ is the set of elements which one can classify neither to $X$ nor to $\neg X$ having knowledge about objects represented by $B$.

If $X_1, \ldots, X_{r(d)}$ are decision classes of $A$ then the set $BX_1 \cup \ldots \cup BX_{r(d)}$ is called the $B$-positive region of $A$ and is denoted by $POS_B(d)$.

If $A = (U, A \cup \{d\})$ is a decision table and $B \subseteq A$, then we define a function $\partial_B : U \rightarrow P(V_d)$, called the $B$-generalized decision of $A$, by

$$
\partial_B(x) = \{v \in V_d : \exists x' \in U \ (x' \text{IND}_A(B)x \text{ and } d(x) = v)\}.
$$

The $A$-generalized decision $\partial_A$ of $A$ is called the generalized decision of $A$.

A decision table $A$ is called consistent (deterministic) if $\text{card}(\partial_A(x)) = 1$ for any $x \in U$, otherwise $A$ is inconsistent (non-deterministic). It is easy to see that a decision table $A$ is consistent iff $POS_A(d) = U$. Moreover, if $\partial_B = \partial_B$ then $POS_B(d) = POS_B(d)$ for any pair of non-empty sets $B, B' \subseteq A$.

A subset $B$ of the set $A$ of attributes of a decision table $A = (U, A \cup \{d\})$ is a relative reduce of $A$ iff $B$ is a minimal set with respect to the following property: $\partial_B = \partial_A$. The set of all relative reduces of $A$ is denoted by $RED(A, d)$.

Let $A = (U, A \cup \{d\})$ be a consistent decision table and let $M(A) = (c_{ij})$ be its discernibility matrix. We construct a new matrix $M'(A) = (c'_{ij})$ assuming $c'_{ij} = 0$ if $d(x_i) = d(x_j)$ and $c'_{ij} = c_{ij} - \{d\}$, otherwise. The matrix $M'(A)$ is called the relative discernibility matrix of $A$. Now one can construct the relative discernibility function $f_{M'(A)}$ of $M'(A)$ in the same way as the discernibility function has been constructed from the discernibility matrix.

It can be shown (see [51]) that the set of all prime implicants of $f_{M'(A)}$ determines the set of all relative reducts of $A$.

Another important type of reducts are local reducts. A local reduct $r(x_i) \subseteq A$ (or a local relative to decision and object $x_i \in U$; where $x_i$ is called a base object) is a subset of $A$ such that:

- $\forall x_j \in U, d(x_i) \neq d(x_j) \implies \exists a_k \in r(x_i): a_k(x_i) \neq a_k(x_j)$
- $r(x_i)$ is minimal with respect to inclusion.

If $A = (U, A \cup \{d\})$ is a decision table then any system $B = (U', A \cup \{d\})$ such that $U' \subseteq U$ is called a subtable of $A$.

The problem of new cases classification can be described in the following way. Let $W = (W, A \cup \{d\})$ be a hypothetical universal decision table (including known and unknown objects describing an actual considered aspect of reality) and let $A = (U, A \cup \{d\})$ be a given subtable of the universal decision table. Let $u \in W$ be a so called tested object. Our task consists in assigning the value $d(u)$ of the decision $d$ to the tested objects $u$, knowing only values of condition attributes of $u$, and relying on a given decision table $A$. In another words we would like to classify object $u$ to the proper decision class $X_k \in CLASSES(d)$ (where $k \in 1, \ldots, r(d)$) on the basis of knowledge included in $A$. A solution of this problem is a classification algorithm sufficiently approximating the decision function $d$. 
3 Decision Rules

Let $A = (U, A \cup \{d\})$ be a decision table and let $V = \bigcup \{V_a : a \in A\} \cup V_d$.

Atomic formulas over $B \subseteq A \cup \{d\}$ and $V$ are expressions of the form $a = v$, called descriptors over $B$ and $V$, where $a \in B$ and $v \in V$. The set $F(B, V)$ of formulas over $B$ and $V$ is the least set containing all atomic formulas over $B$ and $V$ and closed with respect to the classical propositional connectives $\lor$ (disjunction), $\land$ (conjunction), and $\neg$ (negation).

Let $\varphi \in F(B, V)$. Then by $|\varphi|_A$ we denote the meaning of $\varphi$ in a decision table $A$, i.e. the set of all objects in $U$ with property $\varphi$, defined inductively by

1. if $\varphi$ is of the form $a = v$ then $|\varphi|_A = \{x \in U : a(x) = v\}$;
2. $|\varphi \land \varphi'|_A = |\varphi|_A \cap |\varphi'|_A$; $|\varphi \lor \varphi'|_A = |\varphi|_A \cup |\varphi'|_A$; $|\neg \varphi|_A = U - |\varphi|_A$

The set $F(A, V)$ is called the set of conditional formulas of $A$ and is denoted by $C(A, V)$.

Any formula of the form $(a_1 = v_1) \land \ldots \land (a_l = v_l)$, where $v_i \in V_{a_i}$ (for $i = 1, \ldots, l$) and $P = \{a_1, \ldots, a_l\} \subseteq A$, is called a $P$-basic formula.

If $\varphi$ is a $P$-basic formula of $A$ and $Q \subseteq P$, then by $\varphi/Q$ we mean the $Q$-basic formula obtained from the formula $\varphi$ by removing from $\varphi$ all its elementary subformulas $(a = v)$ such that $a \in P \setminus Q$.

A decision rule for $A$ is any expression of the form $\varphi \Rightarrow d = v$ where $\varphi \in C(A, V)$, $v \in V_d$ and $|\varphi|_A \neq \emptyset$. Formulas $\varphi$ and $d = v$ are referred to as the predecessor and the successor of the decision rule $\varphi \Rightarrow d = v$ respectively.

If $r$ is a decision rule in $A$, then by $Pred(r)$ we denote the predecessor of $r$ and by $Succ(r)$ we denote the successor of $r$.

An object $u \in U$ is matched by a decision rule $\varphi \Rightarrow d = v_k$ (where $1 \leq k \leq r(d)$) iff $u \in |\varphi|_A$. If $u$ is matched by $\varphi \Rightarrow d = v_k$ then we say that the rule is classifying $u$ to decision class $X_k$.

The number of objects matched by a decision rule $\varphi \Rightarrow d = v$, denoted by $Match_A(\varphi \Rightarrow d = v)$, is equal $\text{card}(|\varphi|_A)$.

The number $\text{Supp}_A(\varphi \Rightarrow d = v) = \text{card}(|\varphi|_A \cap |d = v|_A)$ is called the number of objects supporting a decision rule $\varphi \Rightarrow d = v$.

A decision rule $\varphi \Rightarrow d = v$ for $A$ is true in $A$, symbolically $\varphi \Rightarrow_A d = v$, iff $|\varphi|_A \subseteq |d = v|_A$. If a decision rule $\varphi \Rightarrow d = v$ is true in $A$, we say that the decision rule is consistent in $A$, otherwise the decision rule $\varphi \Rightarrow d = v$ is inconsistent or approximate in $A$.

If $r$ is a decision rule in $A$, then the number $\mu_A(r) = \frac{\text{Supp}_A(r)}{\text{Match}_A(r)}$ is called the coefficient of consistency of the rule $r$. The coefficient $\mu_A(r)$ can be understood as a degree of consistency of the decision rule $r$. It is easy to see that a decision rule $r$ for $A$ is consistent iff $\mu_A(r) = 1$.

The coefficient of consistency of $r$ can be also treated as a degree of inclusion of $|Pred(r)|_A$ in $|Succ(r)|_A$ (see [47]).

If $\varphi \Rightarrow d = v$ is a decision rule for $A$ and $\varphi$ is $P$-basic formula of $A$ (where $P \subseteq A$), then the decision rule $\varphi \Rightarrow d = v$ is called a $P$-basic decision rule for $A$, or basic decision rule in short.
Let $\mathbf{A} = (U, A \cup \{d\})$ be a consistent decision table with $n$ objects $u_1, \ldots, u_n$ and with $m$ condition attributes $a_1, \ldots, a_m$. We present two methods for basic decision rule synthesis for $\mathbf{A}$ (see [52], [53], [43]).

The first method consists of two steps. In the first step we compute the set $\text{RED}(\mathbf{A}, d)$ of all relative reducts of decision table $\mathbf{A}$. In the second step, for each reduct $R = \{b_1, \ldots, b_l\} \in \text{RED}(\mathbf{A}, d)$ (where $l \leq m$) and any object $u \in U$ we generate a decision rule in the following way: as the predecessor of the decision rule we take the conjunction $(b_1 = b_1(u)) \land \ldots \land (b_l = b_l(u))$ and as the successor of the rule we take the decision attribute $d$ with the value $d(u)$. Hence, the constructed decision rule for the reduct $R$ and the object $u$ is of the form

$$(b_1 = b_1(u)) \land \ldots \land (b_l = b_l(u)) \Rightarrow d = d(u).$$

The second method returns basic decision rules with minimal number of descriptors (see [45], [52]).

Let $\varphi \Rightarrow d = v$ be a $P$-basic decision rule of $\mathbf{A}$ (where $P \subseteq A$) and let $a \in P$. We say that the attribute $a$ is dispensable in the rule $\varphi \Rightarrow d = v$ if $\varphi \Rightarrow d = v|_A = U$ implies $|\varphi/(P \setminus \{a\}) \Rightarrow d = v|_A = U$, otherwise attribute $a$ is indispensable in the rule $\varphi \Rightarrow d = v$. If all attributes $a \in P$ are indispensable in the rule $\varphi \Rightarrow d = v$, then $\varphi \Rightarrow d = v$ is called independent in $\mathbf{A}$.

The subset of attributes $R \subseteq P$ is called a reduct of a $P$-basic decision rule $\varphi \Rightarrow d = v$, if $\varphi/R \Rightarrow d = v$ is independent in $\mathbf{A}$ and $\varphi \Rightarrow d = v|_A = U$ implies $\varphi/R \Rightarrow d = v|_A = U$. If $R$ is a reduct of the $P$-basic decision rule $\varphi \Rightarrow d = v$, then $\varphi/R \Rightarrow d = v$ is said to be reduced. If $R$ is a reduct of the $A$-basic decision rule $\varphi \Rightarrow d = v$, then $\varphi/R \Rightarrow d = v$ is said to be an optimal basic decision rule of $\mathbf{A}$ (a basic decision rule with minimal number of descriptors). The set of all optimal basic decision rules of $\mathbf{A}$ is denoted by $\text{RULE}(\mathbf{A})$.

Let $\mathbf{A} = (U, A \cup \{d\})$ be a consistent decision table and $M'(\mathbf{A}) = (c'_{ij})$ be its relative discernibility matrix. We construct a new matrix $M(\mathbf{A}, k) = (c_{ij}^k)$ for any $x_k \in U$ assuming $c_{ij}^k = c'_{ij}$ if $d(x_i) \neq d(x_j)$ if $i = k$ and $c_{ij}^k = \emptyset$, otherwise. The matrix $M(\mathbf{A}, k)$ is called the $k$-relative discernibility matrix of $\mathbf{A}$.

Now one can construct the $k$-relative discernibility function $f_{M(\mathbf{A}, k)}$ of $M(\mathbf{A}, k)$ in the same way as the discernibility function has been constructed from the discernibility matrix (see Section 2).

It can be shown that the set of all prime implicants of functions $f_{M(\mathbf{A}, k)}$ (for $k = 1, \ldots, \text{card}(U)$) determines the set of all basic optimal rules of $\mathbf{A}$.

Let us assume now that considered decision tables are inconsistent. One can transform an arbitrary inconsistent decision table $\mathbf{A} = (U, A \cup \{d\})$ into a consistent decision table $\mathbf{A}_0 = (U, A \cup \{\partial_A\})$ where $\partial_A : U \to P(V_d)$ is the generalized decision of $\mathbf{A}$ defined in Section 2. It is easy to see that $\mathbf{A}_0$ is a consistent decision table and one can apply to $\mathbf{A}_0$ the described methods to construct decision rules. Hence one can compute decision rules for any inconsistent decision table.
4 Discretization

Suppose we have a decision table $A = (U, A \cup \{d\})$ where $\text{card}(V_a)$ is large for some $a \in A$. Then there is a very low chance that a new object is recognized by rules generated directly from this table, because the attribute value vector of a new object will not match any of these rules. Therefore for decision tables with real value attributes some discretization strategies are built to obtain a higher quality of classification rules. This problem is intensively studied in e.g. [39] and we consider discretization methods presented in [39], [37], [38] and [41]. These methods are based on rough set techniques and boolean reasoning.

4.1 Basic notions

Let $A = (U, A \cup \{d\})$ be a decision table where $U = \{x_1, x_2, \ldots, x_n\}$. We assume $V_a = [l_a, r_a] \subset \mathbb{R}$ for any $a \in A$ where $\mathbb{R}$ is the set of real numbers. In the sequel we assume that $A$ is a consistent decision table.

Any pair $(a, c)$, where $a \in A$ and $c \in \mathbb{R}$, defines a partition of $V_a$ into left-hand-side and right-hand-side interval. Formally, any attribute-value pair $(a, c)$ is associated with a new binary attribute $f_{(a,c)} : U \to \{0, 1\}$ such that

$$f_{(a,c)}(u) = \begin{cases} 0 & \text{if } a(u) < c \\ 1 & \text{otherwise} \end{cases}$$

In this sense, the pair $(a, c)$ is called a binary discriminator or cut on $V_a$. Usually, discretization of real value attributes is determined by cuts.

Let us fix an attribute $a \in A$. Any set of cuts

$$D_a = \{(a, c^a_0), (a, c^a_1), \ldots, (a, c^a_{k_a})\}$$

where $k_a \in \mathbb{N}$ and $c^a_0 = l_a < c^a_1 < c^a_2 < \ldots < c^a_{k_a} < r_a = c^a_{k_a+1}$ defines a partition on $V_a$ (for $a \in A$) into sub-intervals i.e.

$$V_a = [c^a_0, c^a_1) \cup [c^a_1, c^a_2) \cup \ldots \cup [c^a_{k_a}, c^a_{k_a+1})$$

The set of cuts $D_a$ on $a$ defines a discretization of $a$, i.e. new discret attribute $a_{D_a} : U \to \{0, \ldots, k_a\}$ such that

$$a_{D_a}(x) = i \iff a(x) \in [c^a_i, c^a_{i+1})$$

for any $x \in U$ and $i \in \{0, \ldots, k_a\}$ (see Figure 1).

Analogously, any global discretization is determined by a set of cuts on all real value attributes $D = \bigcup_{a \in A} D_a$. Any set of cuts

$$D = \bigcup_{a_i \in A} D_{a_i} = \{(a_1, c^1_1), \ldots, (a_1, c^1_{k_1})\} \cup \{(a_2, c^2_1), \ldots, (a_2, c^2_{k_2})\} \cup \ldots$$

transforms the original decision table $A = (U, A \cup \{d\})$ into new (discret) decision table $A|_D = (U, A_D \cup \{d\})$, where $A_D = \{a_{D_a} : a \in A\}$. The table $A|_D$ is called the D-discretized table of $A$. 
Fig. 1. The discretization of real value attribute $a \in A$ defined by the set of cuts $\{(a, c^a_1), (a, c^a_2), \ldots, (a, c^a_{k_a})\}$

It is obvious, that discretization process is associated with a loss of information. Usually, the task of discretization is to determine a minimal set of cuts $\mathbf{D}$ (with respect to inclusion) from a given decision table $\mathbf{A}$ such that, in spite of losing information, the $\mathbf{D}$-discretized table $\mathbf{A}_D$ still keeps some useful properties of $\mathbf{A}$. In the discretization method based on rough set and Boolean reasoning approach, we are trying to keep the discernibility between objects.

Let $\mathbf{A} = (U, A \cup \{d\})$ be a given decision table and $\mathbf{D}$ be a set of cuts. We say that

- $\mathbf{D}$ is consistent with $\mathbf{A}$ (or $\mathbf{A}$-consistent) if $\partial_A = \partial_{A_D}$, where $\partial_A$ and $\partial_{A_D}$ are generalized decisions of $\mathbf{A}$ and $\mathbf{A}_D$. In other words, the set of cuts is $\mathbf{A}$-consistent iff for any two objects $u, v \in U$:

  $$\text{if} \quad \text{(u, v are discerned by A)} \quad \text{then} \quad \text{(u, v are discerned by D)}.$$  

- $\mathbf{D}$ is irreducible in $\mathbf{A}$ if $\mathbf{D}$ is $\mathbf{A}$-consistent and $\mathbf{D}'$ is not $\mathbf{A}$-consistent for any proper subset $\mathbf{D}' \subset \mathbf{D}$.

- $\mathbf{D}$ is optimal in $\mathbf{A}$ if $\text{card}(\mathbf{D}) \leq \text{card}(\mathbf{D}')$ for any $\mathbf{A}$-consistent set of cuts $\mathbf{D}'$.

The problem of searching for optimal set of cuts (Optimal Discretization Problem) has been explored in [39], [37], [38] and [41]. From computational complexity point of view, the Optimal Discretization Problem appears to be hard. We have the following theorem (see [39]):

**Theorem 1.** For a given decision table $\mathbf{A}$ and an integer $k$,

- The decision problem for checking if there exists an irreducible set of cuts $\mathbf{P}$ in $\mathbf{A}$ such that $\text{card}(\mathbf{P}) < k$ ($k$-minimal partition problem) is $\mathbf{NP}$-complete.

- The problem of searching for an optimal set of cuts $\mathbf{P}$ in $\mathbf{A}$ (optimal partition problem) is $\mathbf{NP}$-hard.

Two sets of cuts $\mathbf{D}', \mathbf{D}$ are equivalent (denoted by $\mathbf{D}' \equiv_{\mathbf{A}} \mathbf{D}$), if and only if $\mathbf{A}_D = \mathbf{A}_{D'}$. The equivalence relation $\equiv_{\mathbf{A}}$ has finite number of equivalence classes. In the sequel we do not distinguish between equivalent sets of cuts.
4.2 Maximal Discernibility (MD) Heuristics

We below describe our heuristic for optimal discretization problem.

Let \( A = (U, A \cup \{d\}) \) be a decision table. An arbitrary attribute \( a \in A \) defines a sequence \( v_1^a < v_2^a < \ldots < v_n^a \), where \( \{v_1^a, v_2^a, \ldots, v_n^a\} = \{a(x) : x \in U\} \) and \( n_a \leq n \). Then the set of all possible cuts on \( a \) is denoted by

\[
C_a = \left\{ \left( a, \frac{v_1^a + v_2^a}{2} \right), \left( a, \frac{v_2^a + v_3^a}{2} \right), \ldots, \left( a, \frac{v_{n-1}^a + v_n^a}{2} \right) \right\}
\]

The set of possible cuts on all attributes is denoted by

\[
C_A = \bigcup_{a \in A} C_a
\]

In [39] we have shown that any irreducible set of cuts of \( A \) is a relative reduct of another decision table \( A_1 \) built from \( A \), where \( A_1 = (U_1, A_1 \cup \{d_i\}) \) is defined as follows:

- \( U_1 = \{(x, y) \in U \times U : d(x) \neq d(y)\} \cup \{\text{new}\}, \) where \( \text{new} \notin U \times U \).
- \( d_1 : U_1 \rightarrow \{0, 1\} \) is defined by \( d_1(u) = \begin{cases} 
0 & \text{if } u = \text{new} \\
1 & \text{otherwise} 
\end{cases} \)
- \( A_1 = \{f(a, c) : (a, c) \in C_A\} \) is a set of all discriminators defined by initial set of cuts from \( C_A \) (see Equation (5)).

The algorithm based on Johnson’s strategy described in the previous section is searching for a cut \( c \in A_1 \) which discerns the largest number of pairs of objects. Then we move the cut \( c \) from \( A_1 \) to the resulting set of cuts \( P \) and remove from \( U_1 \) all pairs of objects discerned by \( c \). Our algorithm is continued until \( U_1 = \{\text{new}\} \). Now we present the details of our algorithm.

**Algorithm 3: MD-discretization**

**Input:** The consistent decision table \( A \).
**Output:** The semi-minimal set of cuts \( D \) consistent with \( A \).

**Method:**

\( D = \emptyset; \ C_A = \text{initial set of cuts on } A; \)
\( L = \{(x, y) \in U \times U : d(x) \neq d(y)\}; \)
while \((L \neq \emptyset)\) do

begin

Choose the cut \( c_{\text{max}} \in C_A \) which discerns the largest number of pairs of objects in \( L \). (i.e. the cut corresponding to the column with the largest number of "1")

Insert \( c_{\text{max}} \) into \( D \) and remove it from \( C_A \).

Remove all pairs of objects from \( L \) discerned by \( c_{\text{max}} \)

end

\( \Box \)
The decision table with two real value attributes and its geometric interpretation.

**Example 1.** We consider the decision table with two attributes and seven objects (Figure 2). The set of all cuts consists of four cuts on the attribute $a$ and three cuts on the attribute $b$.

$$C_A = \{(a, 1.1), (a, 1.25), (a, 1.35), (a, 1.5)\} \cup \{(b, 0.75), (b, 1.5), (b, 2.5)\}$$

The new decision table $A_1$ consists of 7 attributes and 13 objects (Figure 3). The MD-discretization algorithm chooses the cut $(b, 1.5)$ first because it discerns 6 pairs of objects, than it chooses the cuts $(a, 1.25)$ and $(a, 1.5)$. The result of the MD-discretization algorithm is the set of cuts $D = \{(a, 1.25), (a, 1.5), (b, 1.5)\}$.

Let $n$ be a number of objects and $k$ be a number of attributes of decision table $A$, then $\text{card}(A_1) \leq (n-1)k$ and $\text{card}(D) \leq \frac{n(n-1)}{2}$. It is easy to note that for any cut $c \in A_1$ we need $O(n^2)$ steps to find the number of all pairs of objects discerned by $c$. Hence the straightforward realization of this algorithm requires $O(kn^2)$ of memory space and $O(kn^3)$ steps to determine one cut, so it is not useful in practice. The algorithm presented below determines the best cut in $O(kn)$ steps using $O(kn)$ space only.
At first we show that the number of pairs of objects discerned by a given cut can be computed faster than $O(n^2)$. For the given cut $(a,c) \in C_A$ on an attribute $a \in A$ and a given subset of objects $X \subseteq U$ we introduce the following notation:

1. $W^X (a,c) =$ number of pairs of objects from $X$ discerned by $(a,c)$.
2. for $j = 1, \ldots, r$:
   \[ l_j^X (a,c) = \text{card} \{ x \in X : [a(x) < c] \land [d(x) = j] \} \] and
   \[ r_j^X (a,c) = \text{card} \{ x \in X : [a(x) > c] \land [d(x) = j] \} \]
   are numbers of objects from $X$ belonging to the $j^{th}$ decision class and being on the left-hand-side and right-hand-side of the cut $(a,c)$, respectively.
3. The number of objects on either side of $(a,c)$ is:
   \[ l^X(a,c) = \sum_{j=1}^r l_j^X (a,c) = \text{card} \{ x \in X : a(x) < c \} ; \]
   \[ r^X(a,c) = \sum_{j=1}^r r_j^X (a,c) = \text{card} \{ x \in X : a(x) > c \} . \]

We obtain the following lemma:

**Lemma 2.** For any cut $(a,c) \in C_A$, and $X \subseteq U$:

\[ W^X (a,c) = l^X (a,c) \cdot r^X (a,c) - \sum_{i=1}^r l_i^X (a,c) \cdot r_i^X (a,c) \]

This Lemma shows, that the number of pairs of objects discerned by a cut $(a,c) \in C_A$ can be computed in $O(n)$ time. The next theorem is showing that if $(a,c_m)$ and $(a,c_{m+1})$ are consecutive cuts on the attribute $a$, then the value \( W^X(a,c_{m+1}) \) can be derived from \( W^X(a,c_m) \) in $O(1)$ time (because the attribute $a$ is fixed, we will use the notations: \( l_j^X (c_m) \), \( r_j^X (c_m) \), \( l^X(c_m) \), \( r^X(c_m) \) and \( W^X(c_m) \) for simplification, instead of \( l_j^X (a,c) \), \( r_j^X (a,c) \), \( l^X(a,c) \), \( r^X(a,c) \) and \( W^X(a,c) \)).

**Theorem 3.** If there is exactly one object $x \in X \subseteq U$ such that \( c_m < a(x) < c_{m+1} \) and let $t = d(x)$ then:

\[ W^X (c_{m+1}) = W^X (c_m) + [r^X (c_m) - l^X (c_m)] - [r_t^X (c_m) - l_t^X (c_m)] \]

**Proof:** We have:

\[ l_j^X (c_{m+1}) = \begin{cases} l_j^X (c_m) & \text{if } j \neq t \\ l_t^X (c_m) + 1 & \text{if } i = t \end{cases} \quad \text{and} \quad r_j^X (c_{m+1}) = \begin{cases} r_j^X (c_m) & \text{if } j \neq t \\ r_t^X (c_m) - 1 & \text{if } j = t. \end{cases} \]

From those equations it follows that
\[ l^X (c_{m+1}) = l^X (c_m) + 1 \text{ and } r^X (c_{m+1}) = r^X (c_m) - 1. \]

From Lemma 2 we have:

\[ W^X (c_m) = l^X (c_m) \cdot r^X (c_m) - \sum_{i=1}^r l_i^X (c_m) \cdot r_i^X (c_m); \]

\[ W^X (c_{m+1}) = l^X (c_{m+1}) \cdot r^X (c_{m+1}) - \sum_{i=1}^r l_i^X (c_{m+1}) \cdot r_i^X (c_{m+1}) \]

\[ W^X (c_{m+1}) = W^X (c_m) + [r^X (c_m) - l^X (c_m)] - [r_t^X (c_m) - l_t^X (c_m)] \]
Thus
\[
W^X (c_{m+1}) - W^X (c_m) = t^X (c_{m+1}) r^X (c_{m+1}) - t^X (c_m) r^X (c_m) +
\]
\[
t_1^X (c_m) r_1^X (c_m) - t_1^X (c_{m+1}) r_1^X (c_{m+1})
\]
\[
= [t^X (c_m) + 1] \cdot [r^X (c_m) - 1] - t^X (c_m) r^X (c_m) +
\]
\[
t_1^X (c_m) r_1^X (c_m) - [t_1^X (c_m) + 1] \cdot [r_1^X (c_m) - 1]
\]
\[
= [r^X (c_m) - t^X (c_m)] - [r_1^X (c_m) - t_1^X (c_m)].
\]

Let us assume that the number \( W^X (a, c_m) \) of pairs of objects from \( X \) discerned by the cut \((a, c_m)\) has been determined. Theorem 3 shows that we can compute the value \( W^X (a, c_{m+1}) \) in time \( O(1) \). In the consequence the best cut on any attribute can be determined in time \( O(n) \).

We propose two strategies of searching for semi-optimal set of cuts. The first, called local strategy, after finding the best cut and dividing the object set into two subsets of objects, repeats this procedure for each object set separately until some stop condition holds. The quality of a cut (i.e. number of objects discerned by cut) in local strategy is computed locally on a subset of objects. In the second strategy, called global strategy, quality of cuts are computed on whole set of objects. Usually, the local strategy is easier for realization than the global one, but the set of cuts obtained by the global strategy is smaller.

4.3 Local strategy

The local strategy can be realized by using decision tree. A typical algorithm for decision tree generation for a given decision table \( A = (U, A \cup \{d\}) \) is described below:

![Decision Tree](https://via.placeholder.com/150)

**Fig. 4.** The decision tree used for local discretization
Algorithm 4: Local discretization

Input: The consistent decision table $A$.
Output: The semi-minimal set of cuts $D$ consistent with $A$.
Method:

1. Initialize the binary tree variable $T$ with the empty tree.
2. Label the root by the set of all objects $U$ and fix the status of the root to be unready.
3. While there is a leaf marked by unready do
   a. For any unready leave $N$ of the tree $T$
   b. If objects labeling $N$ have the same decision value then
      i. Replace the object set at $N$ by its common decision
      ii. Change the status of $N$ to ready.
   c. Else
      i. Compute the value $W^N(a, c)$ for all cuts from $C_A$ and search for cut $(a^*, c^*)$ maximizing the function $W^N(\cdot)$, i.e.
         \[(a^*, c^*) = \arg\max_{(a,c)} W^N(a, c)\]
      ii. Replace the label of $N$ by $(a^*, c^*)$ and mark it as ready.
      iii. Create two new nodes $N_1$ and $N_2$ with status unready as the left and right subtrees of $N$, where:
           \[N_1 = \{ u \in N : a^*(u) < c^* \} \text{ and } N_2 = \{ u \in N : a^*(u) \geq c^* \}\]
   d. End
   e. End
4. End

return $T$

\[\square\]

4.4 Global strategy

Now we describe some properties of the $D$-discretized table $A_D$, where $D \subseteq C_A$ is an arbitrary set of cuts. Assuming $X_1, X_2, \ldots, X_m$ to be equivalence classes of the discernibility relation $IND(A_D)$ of table $A|_{D}$, one can note that the family \[\text{PART}(D) = \{ X_1, X_2, \ldots, X_m \}\] defines a partition of the set of objects $U$ into $m$ disjoint subsets i.e. \[U = X_1 \cup \ldots \cup X_m \text{ and } \forall i \neq j X_i \cap X_j = \emptyset\]
In practice the classes \(X_1, \ldots, X_m\) are stored in memory instead of \(A|D\). Observe that objects from \(X_i\) \((i = 1, \ldots, m)\) are not discerned by any cut from \(D\). Hence the number of pairs of objects discerned by a cut \(c \notin D\) but not discerned by cuts from \(D\) is equal to

\[
W_D(a, c) = W^{X_1}(a, c) + W^{X_2}(a, c) + \ldots + W^{X_m}(a, c).
\]

(6)

Let us consider the situation, when we have the set of cuts \(D\) defining the equivalence classes \(X_1, \ldots, X_m\) and two consecutive cuts \(c_j^a, c_{j+1}^a\) on the attributes \(a\). We can derive the value \(W_D(a, c_{j+1}^a)\) from \(W_D(a, c_j^a)\) in time \(O(1)\) applying the following theorem:

**Theorem 4.** Let \(D\) be a given set of cuts. If there is exactly one object \(x \in U\) such that \(a(x) \in (c_j^a, c_{j+1}^a)\) then:

\[
W_D(a, c_{j+1}^a) - W_D(a, c_j^a) = W^{X_i}(a, c_{j+1}^a) - W^{X_i}(a, c_j^a) =
\]

\[
= (r^{X_i}(c_j^a) - t^{X_i}(c_j^a)) - (r^{X_i}(c_{j+1}^a) - t^{X_i}(c_{j+1}^a))
\]

where \(t = d(x)\) and \(X_i \in \text{PART}(D)\) is the equivalence class containing \(x\).

**Proof:** This fact follows from Theorem 3 and Equation 6. \(\square\)

Now we present the details of our algorithm.

**Algorithm 5: Global discretization**

**Input:** The consistent decision table \(A\).

**Output:** The semi-minimal set of cuts \(P\) consistent with \(A\).

**Data Structure:** \(D\) – the semi-minimal set of cuts; \(L = \text{PART}(D)\) – the partition of \(U\) defined by \(D\); \(C_A\) – the set of all possible cuts on \(A\).

**Method:**
1. \(D = \emptyset; L = \{U\}; A_i = \text{initial set of cuts on } A\)
2. Compute the value \(W_D(a, c)\) for all cuts from \(C_A\) and search for cut \((a^*, c^*)\) maximizing the function \(W_D(\cdot, \cdot)\), i.e.

\[
(a^*, c^*) = \arg \max_{(a, c)} W_D(a, c)
\]

3. Set \(D = D \cup \{(a^*, c^*)\}; C_A = C_A \setminus \{(a^*, c^*)\}\)
4. for \(X \in L\) do
   if \(X\) consists of objects from one decision class then remove \(X\) from \(L\);
   if \(X^*\) divides the set \(X\) into \(X_1, X_2\) then
     – Remove \(X\) from \(L\)
     – Add to \(L\) two sets \(X_1, X_2\)
5. if \(L\) is empty then Stop else Go to 2
**Theorem 5.** Algorithm 5 needs time of order $O(kn|P| + \log n)$ and $O(kn)$ memory space for computing of the semi-minimal set of cuts $P$. 

**Proof:** Step 1 takes $O(kn \log n)$ steps. From Theorem 4 it follows that Loop 2 and Loop 4 require together $O(kn)$ memory space and $O(kn)$ time. 

**Example 2.** We illustrate the local and global strategy on the decision table presented in the Figure 5.

In both cases our algorithms begin with choosing the best cut $(a_3, 4.0)$ discerning 20 pairs of objects from $A$. Theorem 3 assures that this cut can be found in linear time.

In the local strategy, the cut $(a_3, 4.0)$ divides the set of objects into two subsets $U_1 = \{u_1, u_3, u_4, u_5, u_6, u_{10}\}$ and $U_2 = \{u_2, u_6, u_7, u_8\}$. Then the local discretization algorithm chooses best cuts locally on $U_1$ and $U_2$. The cuts $(a_1, 3.5)$, $(a_2, 3.5)$ and $(a_2, 5.5)$ are best cuts for $U_1$ (because they are discerning 6 pairs of objects from $U_1$); the cuts $(a_1, 5.5)$ and $(a_3, 7.0)$ are the best cuts for $U_2$ (4 pairs of objects from $U_2$). Assume that the cuts $(a_2, 3.5)$ and $(a_1, 5.5)$ have been chosen for $U_1$ and $U_2$, respectively. Hence $U_1$ is divided into two subsets $X_1 = \{u_1, u_3, u_{10}\}$ and $X_2 = \{u_3, u_4, u_5\}$, but the set $U_2$ is divided into two sets $X_3 = \{u_2, u_6\}$ and $X_4 = \{u_7, u_8\}$. One can see that the sets $X_1$, $X_3$ and $X_4$ consists of objects from one decision class, then our algorithm continues the searching process for $X_2$. The result of our algorithm is the set of cuts $\{(a_3, 4.0), (a_2, 3.5), (a_1, 5.5), (a_1, 3.5), (a_2, 6.5)\}$ (see Figure 6).

In the global strategy, the best cut $(a_3, 4.0)$ defines a partition $L = \{U_1, U_2\}$ of $U$, where $U_1 = \{u_1, u_3, u_4, u_5, u_9, u_{10}\}$ and $U_2 = \{u_2, u_6, u_7, u_8\}$. Next, the global discretization algorithm chooses the cut $(a_1, 3.5)$ because it discerns 8 pairs of objects (6 pairs in $U_1$ and 2 pairs in $U_2$). The set of two cuts $\{(a_3, 4.0), (a_1, 3.5)\}$ defines a new partition $L = \{\{u_1, u_3, u_4\}, \{u_5, u_9, u_{10}\}, \{u_2\}, \{u_6, u_7, u_8\}\}$. After removing two sets \{u_5, u_9, u_{10}\}, \{u_2\}, which consists of objects from one decision class, the algorithm chooses the cut $(a_3, 3.5)$ (3 pairs). The set of cuts $\{(a_3, 4.0), (a_1, 3.5), (a_2, 3.5)\}$ defines a partition $L = \{\{u_1\}, \{u_3, u_4\}, \{u_6, u_8\}, \{u_7\}\}$. Finally, the algorithm chooses the cut $(a_2, 6.5)$, which discerns all remaining pairs of objects. As the result we have the set of cuts $\{(a_3, 4.0), (a_1, 3.5), (a_2, 3.5), (a_2, 6.5)\}$.

**4.5 Application of dynamic reducts to finding set of cuts**

Now, we propose another method of searching for an irreducible set of cuts of a given decision table $A_1 = (U_1, A_1 \cup \{d_i\})$ (see Section 4.2). This method is based on the dynamic reduct notion (see Section 7, 8). We calculate dynamic reducts (or generalized dynamic reducts) for the table $A_1$ and we choose one with the best stability coefficient. Next, as an irreducible set of cuts we select cuts belonging to the chosen dynamic reduct. Finally, we remove from $U_1$ and respectively $U$ all objects not belonging to the reduct domain (see Section 9) of the chosen dynamic reduct.
Fig. 5. The exemplary decision table with 10 objects, 3 attributes and 3 decision classes and the illustration of cuts for this table.

Fig. 6. The construction of decision tree by applying local discretization algorithm.

5 Local reducts computation

A notion of local reduct (see Section 2 and [65]) seems to be very useful in classification problems. A set of rules generated basing on these reducts is usually less specific and fits more new (unseen) objects than in a classical case.

Let $\mathbf{A} = (U, A \cup \{d\})$ be an information system, where $U$ - set of objects, $A$ - set of attributes, $d$ - decision. A rule generated by a local reduct is concerned with the base object and may not recognize any other object from $U$. To assure that a set of rules will recognize (at least) all objects from the training set, we
have to generate a local reduct for every object. A simple algorithm checking whether a subset is a local superreduct works at a time complexity of $O(mn)$: we have to compare our base object with all other objects and check whether condition a) (see local reduct definition) holds. It takes $O(mn^2)$ time to check this property for all objects (in case we are looking for a local reduct for every object), where $n$ = number of objects, $m$ = number of attributes.

An algorithm presented below realizes the following objective: assuming that the information system $A$ is consistent, find a family of subsets $R_1$, $R_2$,... $R_k$ such that for any object $x_i$ from $U$ at least one $R_j$ is a local reduct (we will say, that $R_j$ covers $x_i$). We look for a possibly small family $R_1$, ... $R_k$, i.e. we prefer these subsets which cover possibly many objects. We assume, that these subsets reflect regularities in data and generate more general rules and that fact means better classification of new samples and less memory required to store rules.

**Algorithm 6 Optimal covering with local reducts.**

**Input:** decision table $A = (U, \{a_1, ..., a_m\} \cup \{d\})$

**Output:** a set $R$ of local reducts covering $U$

**Method:**

$N = \{N_1, \ldots, N_n\} = 0, \ldots, 0$ - numbers of local reducts found for each object

repeat

$R = \{a_1, \ldots, a_m\}$

repeat

$b = \{b_1, \ldots, b_n\} = 0, \ldots, 0$

$c = \{c_1, \ldots, c_m\} = 0, \ldots, 0$ - tables of (locally) covered objects

LocRed($A$, $R$, $c$, $N$) - check which objects are covered by $R$

for $a_i \in R$ do

begin

$R = R - a_i$

$M_i = LocRed(A, R, b, N)$ - number of objects covered by $R$

$R = R \cup a_i$

end

for $i = 0$ to $n$ if $c_i = 1$ and $b_i = 0$ then

begin

$R = R \cup R(u_i)$ - update set of reducts

$N_i = 1$

end

$j = \arg\max(M_i)$ - if there is more than one maximum, select random

$R = R - a_j$

until $R = 0$

until $\forall i \in \{1, \ldots, n\} N_i = 1$

return $R$

\[\square\]

**Lemma.** The algorithm described above generates a covering for all objects in at most $n = |U|$ cycles of the outer loop.
Proof: see [65].

The algorithm is not deterministic, because we use a random selection of maximal $M$. We need to have a method of determining whether a subset is a local superreduct to complete our algorithm. Function $LocRed$ checks for which objects a subset $R$ is a local reduct and returns a number of these objects as well as a list of them (in the form of binary table):


Input:
1. Decision table $A = (U, A \cup \{d\})$
2. Subset $R \subseteq A$
3. Table of already covered objects $N$

Output:
1. Table $b$ of objects for which $R$ is a local superreduct
2. Number of newly covered objects

Method:

Sort $U$ by values of attributes in $R$
Create partition $U = U_1 \cup \ldots \cup U_k$ into indiscernibility classes,
i.e. $u_1, u_2 \in U_i \Rightarrow (u_1, u_2) \in IND(R)$

$newcover = 0$
for $i = 1$ to $k$ do
begin
$uniform = 1$
for $j = 2$ to $|U_i|$ do
begin
if $d(u_{i,j}) \neq d(u_{i,j-1})$ then $uniform = 0$
- where $U_i = \{u_{i,1}, \ldots, u_{i,|U_i|}\}$
end
if $uniform = 1$ then
begin
for $j = 1$ to $|U_i|$ do
begin
$t = a number of object u_{i,j}$ in table $U$
$b_t = 1$
if $N_t = 0$ then $newcover = newcover + 1$
end
end
end
return $newcover$


The partition of $U$ can be done by $n$ operations if $U$ is properly sorted. Since we may use a fast method of sorting, our algorithm has the complexity of $nn \log(n)$, where $n =$ number of objects, $m =$ number of attributes.
6 Computing reducts using Genetic Algorithms

Our methods of decision rules generation from decision tables are based on the reduct set computation. The time cost of the reduct set computation can be too high in case the decision table consists of too many: objects or attributes or different values of attributes. The reason is that in general the size of the reduct set can be exponential with respect to the size of the decision table and the problem of computing a minimal reduct is NP-hard (see [51]). Therefore we are often forced to apply approximation algorithms to obtain some knowledge about the reduct set. One way is to use approximation algorithms that do not give the optimal solutions but require short computing time. Among these algorithms are the following ones: Johnson’s algorithm, algorithms based on simulated annealing and Boltzmann machines, algorithms using neural networks and algorithms based on genetic algorithms, which we would like to present in this section.

6.1 Genetic and hybrid algorithms

The main idea of genetic algorithms is based on the Darwinian principle of “survival of the fittest” (natural selection). In a case of classical genetic algorithms (see [21], [24]) we are given a state space \( S \) (finite, but large) and a function: \( f: S \to R_+ \). Our goal is to find \( x_0 \): \( f(x_0) = \max \{ f(x) : x \in S \} \). Elements of set \( S \) are “individuals”. We treat a value of the function \( f \) as ability to survive in the environment (“fitness”), and we simulate the process of evolution as follows:

1. We choose the representation scheme: a mapping from a space of “individuals” into “chromosomes” - usually bit strings.
2. We randomly choose the set of chromosomes as an initial population.
3. We calculate “fitness” \( F(c) \) of each chromosome \( c \) as a value of \( f(s(c)) \), where \( s(c) \) is the individual encoded by \( c \). Then we create a new population, replacing the chromosomes with low fitness by those with higher fitness.
4. We randomly affect the new population by genetic operators, e.g. mutation (small, random modifications of chromosomes) and crossing-over (exchange of “genetic material” between some pairs of chromosomes).
5. We repeat 3-4 with the new population, until a stopping criterion is satisfied.

The result of evolution is the best individual \( x_{max} \) which is usually nearly as good as the global optimum \( x_0 \).

The scheme presented above is general and domain-independent. On the other hand, in particular problems we often have some approximation algorithms and heuristics producing maybe not optimal, but good results. To exploit advantages of both genetic and heuristic algorithms, one can use a hybridization strategy [64]. The general scheme of hybrid algorithm is as follows:

1. Find a strategy (heuristic algorithm) which gives an approximate result.
2. Modify (parameterize) the strategy using a control sequence, so that the result depends on this sequence (recipe).
3. Encode the control sequence to a chromosome.
4. Use a genetic algorithm to produce control sequences. Proceed with the
   heuristic algorithm controlled by the sequence. Evaluate an object generated
   by the algorithm and use its quality measure as a fitness of the control
   sequence.
5. A result of evolution is the best control sequence, i.e. the sequence producing
   the best object. Send this object to the output of the hybrid algorithm.

Hybrid algorithms proved to be useful and efficient in many areas, including
NP-hard problems of combinatorics. As we will see in the next section, short
reduct finding problem also can be solved efficiently by this class of algorithms.

6.2 Finding reducts using GA

An order-based genetic algorithm is one of the most widely used component of
various hybrid systems. Theoretical foundations and practical construction of
this algorithm are presented in [64]. In this type of genetic algorithm a chro-
mosome is an n-element permutation σ, represented by a sequence of numbers:
σ(1) σ(2) σ(3) . . . σ(n). Mutation of an order-based individual means one ran-
don transposition of its genes (a transposition of random pair of genes). There
are various methods of recombination (crossing-over) considered in literature. In
[21] such methods as PMX (Partially Matched Crossover), CX (Cycle Crossover)
and OX (Order Crossover) are described. Another type of crossing-over oper-
ator is presented in [63]. After crossing-over, fitness function of every individual is
calculated. In the case of a hybrid algorithm a heuristic part is launched under
control of individual; a fitness value depends on the result of heuristic algorithm.
Then, new population is generated using “roulette wheel” algorithm; the fitness
value of every individual is normalized and treated as probability distribution on
population; then we randomly choose M new individuals using this distribution.
Then all these steps are repeated.

In the hybrid algorithm [62] a simple, deterministic method was used for
reduct generation:

Algorithm 8. Finding a reduct basing on permutation.

Input:
1. decision table $A = (U, \{a_1, \ldots, a_n\} \cup \{d\})$
2. permutation $\tau$ generated by genetic algorithm

Output: a reduct $R$ generated basing on permutation $\tau$

Method:
- $R = \{a_1, \ldots, a_n\}$
- $(b_1 \ldots b_n) = \tau (a_1 \ldots a_n)$
- \[ \text{for } i = 1 \text{ to } n \text{ do} \]
  \[ \begin{align*}
    & R = R - b_i \\
    & \text{if not } \text{Reduct}(R, A) \text{ then } R = R \cup b_i
  \end{align*} \]
end
return $R$

A fast algorithm for determining whether $R$ is a superreduct is presented in [40]:


**Input:**
1. Decision table $A = (U, A \cup \{d\})$
2. Subset $R \subseteq A$
3. Binary tree $T_{red}$ of reducts found so far, list $L_{subred}$ of subreducts found so far.

**Output:**
1. “True” if $R$ is a reduct or superreduct of $A$, “False” otherwise.
2. Updated $T_{red}$ and $L_{subred}$ structures.

**Method:**
if $R \in T_{red}$ then return True
for $s \in L_{subred}$ do
    if $R \subseteq s$ return False
end
Sort $U$ by values of attributes in $R$
for $i = 2$ to $m$ do
    begin
        if $d(u_i) \neq d(u_{i-1})$ then
            begin
                all_equal = 1
                for $a_j \in R$ do
                    begin
                        if $a_j(u_i) \neq a_j(u_{i-1})$ then all_equal = 0
                    end
                end
                if all_equal = 1 then
                    begin
                        Add($L_{subred}, R$
                        return False
                    end
            end
        end
    end
    Add($T_{red}, R$
return True

The result of the algorithm will always be a reduct. Every reduct can be found using this algorithm, the result depends on the order of attributes (proof: see [62]). The genetic algorithm is used to generate the proper order. To calculate the function of fitness for a given permutation (order of attributes) we have to
perform one run of the deterministic algorithm and calculate the length of the found reduct. In the selection phase of genetic algorithm we used linear scaling [21] and the following fitness function:

$$F(\tau) = n - L_\tau + 1$$

The hybrid algorithm described above performs much slower than the classical genetic algorithm. On the other hand, the reducts obtained by this algorithm are usually shorter. Moreover, the hybrid algorithm generates from 50 to 500 different reducts in comparison with 5 to 50 reducts generated by the classical GA at the same time.

An algorithm described above generates possibly shortest reducts, i.e. the reducts with as few attributes as possible. On the other hand, our goal is not to calculate reducts, but to construct an efficient system for classification or decision making. Another approach is to select a reduct due to the number of rules it generates rather than to its length. Every reduct generates an indiscernibility relation on the universe and in most cases it identifies some pairs of objects. If a reduct generates less rules, it means, that the rules are more general and they should better recognize new objects.

The number of rules can be easily computed due to the improvement of the reduct generation system described in [40]. The hybrid algorithm described above can be used to find reducts generating the minimal number of rules. The only thing we have to change is the definition of the fitness function:

$$F(\tau) = m - R_\tau + \frac{n - L_\tau + 1}{n}$$

where $R_\tau$ denotes the number of rules generated by the reduct. Now, the primary criterion of optimization is the number of rules, while the secondary is the reduct length. The results of experiments [64] show, that the classification system based on the reducts optimized due to the number of rules performs better (or not worse) than the short reduct based one. Moreover, due to the rule set reduction, it occupies less memory and classifies new objects faster.

7 Dynamic Reducts

The methods based on calculation of reducts allow to compute, for a given decision table $A$, the descriptions of all decision classes of $A$ in the form of decision rules (see previous sections). Unfortunately, decision rules calculated in this way can often be inappropriate to classify unseen cases. We suggest that the rules calculated by means of dynamic reducts are better predisposed to classify unseen cases, because these reducts are the most stable reducts in a process of random sampling of the original decision table.

Let $A = (U, A \cup \{d\})$ be a decision table. By $P(A)$ we denote the set of all subtables of $A$. If $F \subseteq P(A)$ then by $DR(A, F)$ we denote the set

$$RED(A, d) \cap \bigcap_{B \in F} RED(B, d).$$
Any element of $DR(A, F)$ is called an $F$-dynamic reduct of $A$.

From the definition of a dynamic reduct it follows that a relative reduct of $A$ is dynamic if it is also a reduct of all subtables from a given family $F$. This notion can be sometimes too much restrictive so we apply also a generalization of dynamic reducts - $(F, \varepsilon)$-dynamic reducts, where $\varepsilon \in [0, 1]$. The set $DR_\varepsilon(A, F)$, of all $(F, \varepsilon)$-dynamic reducts is defined by

$$DR_\varepsilon(A, F) = \{ C \in RED(A, d) : \frac{\text{card}(\{ B \in F : C \in RED(B, d) \})}{\text{card}(F)} \geq 1 - \varepsilon \}$$

If $C \in RED(A, d)$ then the number:

$$\frac{\text{card}(\{ B \in F : C \in RED(B, d) \})}{\text{card}(F)}$$

is called the stability coefficient of the reduct $C$ (relative to $F$).

**Proposition 6.** Let $A = (U, A \cup \{ d \})$ be a decision table.

1. If $F = \{ A \}$ then $DR(A, F) = RED(A, d)$.
2. If $\varepsilon_1 \leq \varepsilon_2$ then $DR_{\varepsilon_1}(A, F) \subseteq DR_{\varepsilon_2}(A, F)$.
3. $DR(A, F) \subseteq DR_\varepsilon(A, F)$, for any $\varepsilon \in [0, 1]$.
4. $DR_0(A, F) = DR(A, F)$.

### 8 Generalized Dynamic Reducts

From the definition of a dynamic reduct it follows that a relative reduct of any table from a given family $F$ of subtables of $A$ can be dynamic if it is also a reduct of the table $A$. This notion can be sometimes not convenient because we are interested in useful sets of attributes which are not necessarily reducts of the table $A$. Therefore we have to generalize the notion of a dynamic reduct.

Let $A = (U, A \cup \{ d \})$ be a decision table and $F \subseteq P(A)$. By $GDR(A, F)$ we denote the set

$$\bigcap_{B \in F} \text{RED}(B, d).$$

Elements of $GDR(A, F)$ are called $F$-generalized dynamic reducts of $A$.

From the above definition it follows that any subset of $A$ is a generalized dynamic reduct if it is also a reduct of all subtables from a given family $F$. Analogously to dynamic reducts we define a more general notion of generalized dynamic reducts - $(F, \varepsilon)$-generalized dynamic reducts, where $\varepsilon \in [0, 1]$. The set $GDR_\varepsilon(A, F)$ of all $(F, \varepsilon)$-generalized dynamic reducts is defined by

$$GDR_\varepsilon(A, F) = \{ C \subseteq A : \frac{\text{card}(\{ B \in F : C \in RED(B, d) \})}{\text{card}(F)} \geq 1 - \varepsilon \}.$$
If $C \in RED(B, d)$ (for any $B \in F$) then the number:

$$\frac{\text{card}([B \in F : C \in RED(B, d)])}{\text{card}(F)}$$

is called the stability coefficient of the generalized dynamic reduct $C$ (relative to $F$).

**Proposition 7.** Let $A = (U, A \cup \{d\})$ be a decision table.

1. $D_R(A, F) \subseteq GDR(A, F)$.
2. $D_Rc(A, F) \subseteq GDRc(A, F)$ for any $\varepsilon \in [0, 1]$.
3. If $A \in F$ then $D_R(A, F) = GDR(A, F)$.

**9 Reduct Domain**

Let $A = (U, A \cup \{d\})$ be a decision table, $F \subseteq P(A)$, and let $GDRc(A, F)$ (for any $\varepsilon \in [0, 1]$) be the set of $(F, \varepsilon)$-generalized dynamic reducts.

For any $R \in GDRc(A, F)$ we define the reduct domain (denoted by $RD(A, F, R)$) as the set:

$$\bigcup\{U_B : R \in RED(B, d) \text{ and } B = (U_B, A \cup \{d\}) \in F \cap POS_R(d)\}$$

The domain of the reduct $R$ is the set of all objects belonging to decision tables $B \in F$ satisfying $R \in RED(B, d)$ on which the decision can be uniquely determined by attributes from $R$.

The notion of a reduct domain is very important for decision rules generation (see Section 12).

**10 Statistical Inference about Dynamic Reducts**

Statistics is frequently defined as the science of collecting and studying numerical data in which deductions are made on the assumption that the relationships between a sufficient sample of numerical data are characteristic of those between all such data (see [12, 18, 25]). The data in most statistical problems relate to a sample drawn from some parent population or universe (as it is called in rough set theory). When a sample is used to make inferences about the population, we generally assume that the sample is random. This usually means (when the population is finite) that any individual in the population has an equal chance of being included in the sample. It is desirable that sampling should be as nearly random as possible, although this is often difficult to be achieved in practice, because sometimes we do not have equal access to all elements of the universe. This problem can be described in language of rough set theory. If we want to calculate the set of decision rules based on a given decision table, we would like to construct rules which are proper not only for objects from this decision
table but also for still unknown examples of objects. Dynamic reducts (defined in the previous sections) are calculated with respect to a family \( F \) of subtables created by random samples of a given decision table. The family \( F \) is, of course, only a subfamily of all subtables of the hypothetical universal decision table (including known and unknown objects describing a currently considered aspect of reality). The family of all subtables of the universal decision table is denoted by \( G \). We are interesting in reducts which most frequently appear in the family \( G \), because we expect that the decision rules generated from these reducts are better predisposed to designate a value of the decision for objects from the universal decision table \( W \). In other words, we are interested in the probability of the event that a given \( F \)-generalized dynamic reduct \( R \) is a reduct for any subtable from \( G \). This probability is denoted by \( P_G(R) \) and defined as the quotient \( \frac{\text{card}(G_R)}{\text{card}(G)} \), where

\[
G_R = \{ B \in G : R \in RED(B, d) \}. \tag{7}
\]

We would like to select an \( F \)-dynamic reduct \( R \) for which the probability \( P_G(R) \) is not less than the probability \( P_G \) for other dynamic reducts. However we cannot calculate \( P_G \) for any \( F \)-generalized dynamic reduct \( R \) because we do not know the whole family \( G \). In this section we show that the so called stability coefficient of any \((F, \varepsilon)\)-generalized dynamic reduct \( R \) is a proper measure of the probability \( P_G(R) \). Unfortunately, we usually do not have an access to all subtables from the family \( G \), therefore we construct the family \( F \) based only on subtables of a given decision table \( A \). We have to assume that the decision table \( A \) is a representative sample from the universal decision table \( W \).

**Theorem 8.** Stability coefficients as maximum likelihood estimator (see [6]).

Let us assume that

- \( W = (W, A \cup \{ d \}) \) is a universal decision table,
- \( A = (U, A \cup \{ d \}) \) is a given decision table \((U \subseteq W)\),
- \( G = \mathcal{P}(W) \) is a family, called the parent population,
- \( F \subseteq \mathcal{P}(A) \),
- \( R \) is an \((F, \varepsilon)\)-generalized dynamic reduct for some \( \varepsilon \in [0, 1] \).

Then we have: the stability coefficient of the \((F, \varepsilon)\)-generalized dynamic reduct \( R \) of the decision table \( A \) is the maximum likelihood estimator of the probability \( P_G(R) \) (see [25]).

The maximum likelihood estimator of the probability \( P_G(R) \) is denoted by \( \text{MLE}(P_G(R)) \).

**Proof:** Let us first introduce the simple binomial distribution \( X^R_G(B) : G \to \{0, 1\} \) (for the family \( G \) and the reduct \( R \)) defined for any \( B \in G \):

\[
X^R_G(B) = \begin{cases} 1 & \text{for } R \in RED(B, d) \text{(success)} \\ 0 & \text{for } R \notin RED(B, d) \text{(defeat)} \end{cases} \tag{8}
\]
Let $G^1 = \{B \in G : X_G^R(B) = 1\}$ and $G^0 = \{B \in G : X_G^R(B) = 0\}$. Now it is easy to observe that the probability $P$ of the success in our binomial distribution is: $P[X_G^R(B) = 1] = P_G(R)$ and the probability of the defeat in our binomial distribution is: $P[X_G^R(B) = 0] = 1 - P_G(R)$.

From [25] we know that the probability of a success in our distribution $X_G^R$ may be estimated by taking a sample of subtables from the family $G$ (for example $F$) and next using the method of maximum likelihood estimator. The maximum likelihood estimator of success probability in our distribution is the arithmetic mean of values $X_G^R$ for all subtables from the sample (see for instance [25]).

Hence:

$$MLE(P_G(R)) = \frac{\sum_{B \in F} X_G^R(B)}{\text{card}(F)} = \frac{\sum_{B \in F \cap G^1} X_G^R(B) + \sum_{B \in F \cap G^0} X_G^R(B)}{\text{card}(F) \cdot \text{card}(F)} = \frac{\text{card}(F \cap G^1)}{\text{card}(F)}. \quad (9)$$

From our definition of $F$-generalized dynamic reducts (Section 8) we conclude that the number $\frac{\text{card}(F \cap G^1)}{\text{card}(F)}$ is equal to the stability coefficient of the generalized dynamic reduct $R$.

This completes the proof.

\[\square\]

**Remark 1.** Dynamic reducts as the tools for classification

Theorem 8 is showing that dynamic reducts with large stability coefficients are "good" candidates for decision rules generation. They allow to construct rules with better classification quality of unseen objects than reducts with smaller stability coefficients.

**Remark 2.** Minimal size of family $F$

It is easy to observe (see the proof of Theorem 8) that for any dynamic reduct $R$ the problem of calculating the stability coefficient of $R$ is equivalent to the problem of calculating an unknown success probability $P_G(R)$ in binomial distribution $X_G$. Therefore, the maximum likelihood estimator of the probability $P_G(R)$ is equal to $\frac{\text{card}(F \cap G^1)}{\text{card}(F)}$. For calculating the necessary size of family $F$, we need to make some assumption about a confidence coefficient: $1 - \alpha$. The confidence coefficient can be understood as a measure of probability estimation correctness (see for instance [18]). From the Moivre-Laplace theorem (see for instance [18] - Theorem 6.7.1) we know that

$$\frac{\text{MLE}(P_G(R)) - P_G(R)}{\sqrt{\frac{\text{MLE}(P_G(R)) \cdot (1 - \text{MLE}(P_G(R)))}{\text{card}(F)}}}$$

has approximately a standard normal distribution. Hence,

$$P \left[ -t_\alpha < \frac{\text{MLE}(P_G(R)) - P_G(R)}{\sqrt{\frac{\text{MLE}(P_G(R)) \cdot (1 - \text{MLE}(P_G(R)))}{\text{card}(F)}}} < t_\alpha \right] = 1 - \alpha, \quad (11)$$
where the number \( t_a \) is satisfying the equation:

\[
1 - \alpha = \frac{2}{\pi} \int_{-t_a}^{t_a} \exp\left(-\frac{t^2}{2}\right)dt. \tag{12}
\]

From equation (11) we have

\[
P \left[ MLE(P_G(R)) - t_a \cdot \sqrt{\frac{MLE(P_G(R)) \cdot (1 - MLE(P_G(R)))}{\text{card}(F)}} < P_G(R) < 
MLE(P_G(R)) + t_a \cdot \sqrt{\frac{MLE(P_G(R)) \cdot (1 - MLE(P_G(R)))}{\text{card}(F)}} \right] = 1 - \alpha. \tag{13}
\]

Hence, the 100 \( \cdot (1 - \alpha)\% \) confidence interval for \( P_G(R) \) is given by

\[
MLE(P_G(R)) - t_a \cdot \sqrt{\frac{MLE(P_G(R)) \cdot (1 - MLE(P_G(R)))}{\text{card}(F)}} < P_G(R) < 
MLE(P_G(R)) + t_a \cdot \sqrt{\frac{MLE(P_G(R)) \cdot (1 - MLE(P_G(R)))}{\text{card}(F)}}. \tag{14}
\]

If \( \Delta MLE(P_G(R)) \) is a maximal acceptable estimation error of \( MLE(P_G(R)) \), then we conclude from inequality (14) that

\[
t_a \cdot \sqrt{\frac{P_G(R) \cdot (1 - P_G(R))}{\text{card}(F)}} \leq \Delta MLE(P_G(R)). \tag{15}
\]

Hence

\[
\text{card}(F) \geq \frac{t_a^2 \cdot P_G(R) \cdot (1 - P_G(R))}{(\Delta MLE(P_G(R)))^2}. \tag{16}
\]

It is easy to observe that if \( P_G(R) = \frac{1}{4} \) then the product \( P_G(R) \cdot (1 - P_G(R)) \) takes the maximum value of \( \frac{1}{16} \). Therefore it is enough to require that the size of \( \mathbf{F} \) is no less than \( \frac{t_a^2}{4(\Delta MLE(P_G(R)))^2} \).

The value of \( t_a \) one can read from the table of standardized normal distribution function (see e.g. [12], [25]).

For example, if we take \( 1 - \alpha = 0.9 \) and \( \Delta MLE(P_G(R)) = 0.05 \) than the value \( t_a \) is 1.64 and \( \text{card}(F) \geq \frac{1.64^2}{4 \times 0.05^2} = 268.96 \).
11 Techniques for Dynamic Reduct Computation

In this section we present a method for computing generalized dynamic reducts and reduct domains. In our method a random set of subtables \( F \) from a given data table \( A = (U, A \cup \{d\}) \) is taken and reducts for all these tables are calculated. The number of samples \( \text{card}(F) \) can be selected by taking into account the minimal family size (see previous section). The method of the random choice of sub-table consists of the two steps. In the first step we randomly choose some numbers with the probability:

\[
P(k) = \frac{\binom{n}{k}}{\sum_{i=1}^{n} \binom{n}{i}}
\]

where \( k = 1, \ldots, n \) and \( n = \text{card}(U) \). Next, we randomly choose a sub-table of \( A \) consisting of \( k \) objects.

Thus we receive a family \( F \) of decision tables and for each \( A \in F \) we compute reducts. Next step is to compute the reduct domain for every reduct (as the positive region of the table constructed from the sum of all objects from proper subtables). In the following step reducts with the stability coefficients higher than a fixed threshold are extracted. The reducts distinguished in such a way are treated as the true generalized dynamic reducts of the table \( A \) together with their reduct domains.

We would like to mention another method of dynamic reduce computation for decision tables. In this case we assume that the reduct set \( \text{RED}(A) \) is already computed. Instead of computing reduce sets for subtables from \( F \) it is enough to check which reducts from \( \text{RED}(A) \) are also reducts for subtables from \( F \). This can save computing time because time necessary for checking which reducts from \( \text{RED}(A) \) are \( F \)-dynamic reducts of \( A \) is polynomial with respect to the size of \( F \) and \( \text{RED}(A) \). However this method can be used only in case we are able to compute the set of all reducts.

12 Decision Rules Computed from Dynamic Reducts

After a set of dynamic reducts with their reduct domains has been computed it is necessary to decide how to compute the set of decision rules. We discuss methods based on the decision rule set calculation for any chosen dynamic reduct. The rules for each reduct are calculated separately. For example one can calculate decision rules for any chosen dynamic reduct \( R \). Let us consider the two following methods.

In the first method for any object from the reduct domain \( R \) we take the value vector of conditional attributes from \( R \) and the corresponding decision of the object. Unfortunately, decision rules generated by this method have poor performance, because the number of objects supporting such rules is usually very small.
In the second method we generate decision rules with minimal number of descriptors (see Section 3) for the table consisting of the object set equal to the reduct domain of \( R \), the conditional attribute set equal to \( R \), and assuming the decision attribute to be the same as in the original decision table. The final decision rule set is equal to the union of all these sets.

When a new object is to be classified, it is first matched against all decision rules from the constructed decision rule set. Next, the final decision is predicted by applying some strategy constructing the final decision from all "votes" of decision rules (see Section 15).

13 Dynamic Rules

From [3] and [4] we know that the quality of unseen object classification based on dynamic reducts (see Section 12) is usually better than the quality of classification based on the whole set of attributes. Therefore one can adopt an idea of dynamic reducts as a method for decision rules computation. By analogy with dynamic reducts, we propose the following method for dynamic rules computation. At the beginning, from a given data table a random set of subtables is chosen. Next the optimal decision rule sets for all these tables are calculated. In the following step the rule memory is constructed where all rule sets are stored. Intuitively, any dynamic rule is appearing in all (or almost all) of experimental subtables. The decision rules can be computed from the so called \textit{k-relative discernibility matrix} used to generate decision rules with the minimal number of descriptors (see Section 3).

Let \( A = (U, A \cup \{d\}) \) be a decision table and \( F \subseteq P(A) \). A decision rule \( r \in \bigcup_{B \in F} RUL(B) \) is called an \textit{F-dynamic rule of} \( A \) if

\[
\text{Support}(r) \neq 0 \Rightarrow r \in RUL(B), \text{ for any } B \in F.
\]

By \( DRUL(A, F) \) we denote the set of all \( F \)-dynamic rules of \( A \).

From the definition of dynamic rules it follows that any optimal decision rule for \( B \in F \) is an \( F \)-dynamic rule of \( A \) if it is also an optimal rule for all subtables from a given family \( F \) having some objects matched by the considered decision rule. This notion can be sometimes too much restrictive, so we apply also a more general notion of a dynamic rule - (\( F, \varepsilon \))-dynamic rules, where \( \varepsilon \in [0, 1] \).

The set \( DRUL_{\varepsilon}(A, F) \) of all \( (F, \varepsilon) \)-dynamic rules is defined as

\[
DRUL_{\varepsilon}(A, F) = \{ r \in \bigcup_{B \in F} RUL(B) : \frac{\text{card}(\{B \in F : r \in RUL(B)\})}{\text{card}(\{B \in F : \text{Support}(r) \neq 0\})} \geq 1 - \varepsilon \}.
\]

If \( r \in RUL(B) \) (for any \( B \in F \)) then the number:

\[
\frac{\text{card}(\{B \in F : r \in RUL(B)\})}{\text{card}(\{B \in F : \text{Support}(r) \neq 0\})}
\]

is called the \textit{stability coefficient} of the dynamic rule \( r \) (relatively to \( F \)) and it is denoted \( SC_{A}^{F}(r) \).
Proposition 9. Let $A = (U, A \cup \{d\})$ be a decision table.

1. If $F = \{A\}$ then $DRUL(A, F) = RUL(A)$.
2. If $\varepsilon_1 \leq \varepsilon_2$ then $DRUL_{\varepsilon_1}(A, F) \subseteq DRUL_{\varepsilon_2}(A, F)$.
3. $DRUL(A, F) \subseteq DRUL_{\varepsilon}(A, F)$, for any $\varepsilon \geq 0$.
4. $DRUL_0(A, F) = DRUL(A, F)$.

Our methods for dynamic rules generation from decision tables are based on the reduct set computation. A random set of subsets from a given data table is taken (see Section 11) and the optimal rules for all these tables are calculated (see Section 3). The time cost of the reduct set computation can be very high when the decision table has too many objects or attributes or different values of attributes (see Section 3). Therefore we often apply some approximate algorithms to obtain knowledge about optimal rule sets (see Section 12, 15, [38], [62]) and next we use the following proposition to compute stability coefficient of calculated decision rules.

Proposition 10. Let us assume that

- $A = (U, A \cup \{d\})$ is a decision table, where $\text{card}(U) = n$ and $\text{card}(A) = m$,
- $r = ((a_1 = v_1) \land \ldots \land (a_i = v_i) \Rightarrow (d = v_d)) \in DRUL_{\varepsilon}(A, P(A))$ for some $\varepsilon \in [0, 1]$,
- $h_{l-1} = \text{card}(H_{l-1})$, where $H_{l-1} = \{u \in U : \forall i \in \{1, \ldots, l\} : a_i(u) = v_i \land d(u) \neq v_d\}$,
- $h_0 = \text{card}(H_0)$, where $H_0 = \{u \in U : \forall i \in \{1, \ldots, l\} : a_i(u) = v_i \land d(u) = v_d\}$,
- $h_i = \text{card}(H_i)$, where $H_i = \{u \in U : \forall j \in \{1, \ldots, l\} \setminus \{i\} : a_j(u) = v_j \land a_i(u) \neq v_i \land d(u) \neq v_d\}$ for $i = 1, \ldots, l$.

Then we have: the stability coefficient of the $(P(A), \varepsilon)$-dynamic rule $r$ of the decision table $A$ can be computed using the following equation:

$$SC_{A}^{P(A)}(r) = \frac{1}{2^{h_{l-1}}}, \text{ for } l = 1,$$

and by

$$SC_{A}^{P(A)}(r) = \frac{\prod_{i=1}^{l} (2^{h_i} - 1)}{2^{h_{l-1} + \sum_{i=1}^{l} h_i}} \text{ for } l > 1. \quad (19)$$

One can prove the above proposition using some basic facts from combinatorics and rough set theory.

It is easy to construct an algorithm, based on the equation (18) and (19), calculating the stability coefficient $SC_{A}^{P(A)}(r)$ for the rule $r$ in time $O(m \cdot n)$ and space $O(C)$, where $C$ is a constant.
14 Approximate Rules

One can use approximate decision rules instead of optimal decision rules to construct the classification algorithm for a decision table $A$ (see Section 3). We have implemented a method for computing approximate rules. We begin with an algorithm for synthesis of optimal decision rules from a given decision table (see Section 3). Next, we compute approximate rules from already calculated optimal decision rules. Our method is based on the notion of consistency of a decision rule. The original optimal rule is reduced to an approximate rule with the coefficient of consistency exceeding a fixed threshold.

Let $A = (U, A \cup \{d\})$ be a decision table and $r_0 \in RL(A)$. The approximate rule (based on rule $r_0$) is computed using the following algorithm.

**Algorithm 10 Approximate rule synthesis (by descriptor dropping)**

**Input:**
1. decision table $A = (U, A \cup \{d\})$,
2. decision rule $r_0 \in RL(A)$,
3. threshold of consistency $\mu_0$ (e.g., $\mu_0 = 0.9$).

**Output:** the approximate rule $r_{app}$ (based on rule $r_0$).

**Method:**

Calculate the coefficient of consistency $\mu_A(r_0)$

if $\mu_A(r_0) < \mu_0$ then STOP (in this case no approximate rule).

$\mu_{max} = \mu_A(r_0)$ and $r_{app} = r_0$.

while $\mu_{max} > \mu_0$ do

begin

$\mu_{max} = 0$

for $i = 1$ to the number of descriptors from $Pred(r_{app})$ do

begin

$r = r_{app}$.

Remove $i$-th descriptor from $Pred(r)$.

Calculate the coefficient of consistency $\mu_A(r)$ and $\mu = \mu_A(r)$.

if $\mu > \mu_{max}$ then $\mu_{max} = \mu$ and $i_{max} = i$.

end

if $\mu_{max} > \mu_0$ then remove $i_{max}$ -th conditional descriptor from $r_{app}$.

end

return $r_{app}$.


It is easy to see that the time and space complexity of Algorithm 10 are of order $O(l^2 \cdot n)$ and $O(C)$, respectively (where $l$ is the number of conditional descriptors in the original optimal decision rule $r_0$ and $C$ is a constant).

The approximate rules, generated by the above method, can help to extract interesting laws from decision table. By applying approximate rules instead of optimal rules one can slightly decrease the quality of classification of objects from the training set but we expect in return to receive more general rules with the higher quality of classification for new objects (see [9]).
15 Negotiations among Rules

Suppose we have a set of decision rules. In most cases it is necessary to decide how to resolve conflicts between sets of rules classifying tested objects to different decision classes. In this section we present several methods for constructing the measure called the strength of rule set. The strength of rule set is a rational number belonging to $[0,1]$ representing the importance of the sets of decision rules relative to the considered tested object.

Let us assume that:

- $W = (W, A \cup \{d\})$ is a universal decision table,
- $A = (U, A \cup \{d\})$ is a given decision table ($U \subseteq W$),
- $u_t \in W$ is a tested object,
- $Rul(X_j)$ is a set of all calculated basic decision rules for $A$, classifying objects to the decision class $X_j$ (where $v^j_d \in V_d$)
- $MRul(X_j, u_t) \subseteq Rul(X_j)$ is a set of all decision rules from $Rul(X_j)$ matching tested objects $u_t$.

We define several measures for the rule set $MRul(X_j, u_t)$ depending on the number of rules from this set matching tested object, the number of objects supporting decision rules from this set and the stability coefficient of rules.

1. The simple strength of a decision rule set is defined by

$$SimpleStrength(X_j, u_t) = \frac{\text{card}(MRul(X_j, u_t))}{\text{card}(Rul(X_j))}. \quad (20)$$

2. The maximal strength of a decision rule set is defined by

$$MaximalStrength(X_j, u_t) = \max_{r \in MRul(X_j, u_t)} \left\{ \frac{\text{Supp}_A(r)}{\text{card}(d = v^j_d|A)} \right\}. \quad (21)$$

3. The basic strength of a decision rule set is defined by

$$BasicStrength(X_j, u_t) = \frac{\sum_{r \in MRul(X_j, u_t)} \text{Supp}_A(r)}{\sum_{r \in Rul(X_j)} \text{Supp}_A(r)} \quad (22)$$

4. The global strength of a decision rule set is defined by

$$GlobalStrength(X_j, u_t) = \frac{\text{card}\left( \bigcup_{r \in MRul(X_j, u_t)} [\text{Pred}(r]|A \cap d = v^j_d|A] \right)}{\text{card}(d = v^j_d|A)} \quad (23)$$

5. The stability strength of a decision rule set is defined by

$$StabilityStrength(X_j, u_t) = \max_{r \in MRul(X_j, u_t)} \{SC_A^p(r)\}. \quad (24)$$
6. The **maximal stability strength** of a decision rule set is defined by

\[
\text{MaximalStabStrength}(X_j, u_t) = \max_{r \in \text{MRule}(X_j, u_t)} \frac{\text{Supp}_A(r)}{\text{card}(\{d \in \mathcal{D}_d \mid d \in r\})} \cdot SC_{\text{P(A)}}(r).
\]  

(25)

7. The **basic stability strength** of a decision rule set is defined by

\[
\text{BasicStabStrength}(X_j, u_t) = \sum_{r \in \text{MRule}(X_j, u_t)} \frac{\text{Supp}_A(r) \cdot SC_{\text{P(A)}}(r)}{\sum_{r \in \text{Rule}(X_j)} \text{Supp}_A(r) \cdot SC_{\text{P(A)}}(r)}
\]  

(26)

8. The **global stability strength** of a decision rule set (denoted GlobStabS) is defined inductively by

\[
\begin{align*}
\text{GlobStabS}(\{r\}) &= SC_{\text{P(A)}}(r), \text{ for } r \in \text{MRule}(X_j, u_t) \\
\text{GlobStabS}(R \setminus \{r\} \cup \{r\}) &= \\
&= \text{GlobStabS}(\{r\}) + \text{GlobStabS}(R \setminus \{r\}) - \\
&\quad \text{GlobStabS}(\{r\}) \cdot \text{GlobStabS}(R \setminus \{r\})
\end{align*}
\]  

(27)

The maximal strength of a decision rule set is similar to the strength of rule presented in [31] and [57]. The basic strength of a decision rule set is similar to the strength of rule presented in [22] and [23]. The global strength of a decision rule set is similar to the strength of rule presented in [31], [22] and [23]. Measures of strengths of rules defined above can be applied in constructing classification algorithms (see next section).

### 16 General Scheme of Classification Algorithm

In this section we present general scheme of classification algorithms based on methods and techniques described in previous sections. One can choose options presented in the general scheme (see Figure 7) for the construction of a particular classification algorithm.

### 17 Summary

In this Chapter we discussed some methods for extracting laws from data. We presented some techniques based on standard rough set methods (see [44], [45], [52]) like reduct set and rule set computation. We also described dynamic techniques e.g. dynamic reducts and dynamic rules computation (see [3], [4], [5], [6]), that give potentially more general decision rules more capable to new cases classification. For the case of larger data tables we proposed a genetic algorithm based method for computation of a approximate reduct set. We also proposed some discretization algorithms of real value attributes that can be used in the
process of data preprocessing. All presented methods can be successfully used together in order to construct a good classification algorithms.

Acknowledgments: This work was partially supported by the grants of Polish National Committee for Scientific Research (KBN) No. 8T11C01011 and No. 8T11C02417 and also by the ESPRIT project 20288 CRIT-2.

References

42. Nguyen, T., Świąierski, R., Skowron, A., Bazan, J., Thyagarajan, K.: Application of Rough Sets, Neural Networks and Maximum Likelihood for Texture Classification Based on Singular Value Decomposition, Proceedings of the Third Interna-


