DECISION RULES FOR LARGE DATA TABLES

Sinh Hoa Nguyen, Tuan Trung Nguyen, Lech Polkowski, Andrzej Skowron, Piotr Synak, Jakub Wróblewski
Institute of Mathematics, Department of Mathematics, University of Warsaw
Warszawa ul. Banacha 2
email: skowron@mimuw.edu.pl

ABSTRACT

Data mining and knowledge discovery in large databases in recent years have become hot topics not only for researchers, they have as well attraped the attention of many major companies in the information technology domain. In this paper we present an approach to the problem, based on rough set methods. The main aim is to show how rough set and rough mereology theories can be effectively used to extract knowledge from large databases.

1. OVERVIEW

A typical database, which usually is based on the relational model, can be viewed as a single table with columns (fields) and rows (records). In such a base knowledge is extracted (discovered) in the form of relationships between columns, expressed in terms of rough set theory by decision rules. Example: (Age = 40) $\wedge$ (Car = Mustang) $\Rightarrow$ (Nationality = American).

To "mine" data in a database in the view of rough set theory thus means computing decision rules for the decision table formed from columns and rows of the base. While tests clearly shows that simple rule computing algorithms of the rough set theory are fast enough to produce goods results on databases with as much as 10,000 rows, the efficiency of these classic algorithms on larger databases remains under question. One of the possible solutions to the problem is the decomposition technique, which splits the large table into smaller ones so rules may be computed in a reasonable time, and then uses these "local" rules to synthesize global rules for the whole base. Our paper will show how decomposition can be done by means of intelligent splitting algorithms and synthesizing technique based on rough mereology.

2. LARGE TABLE DECOMPOSITION

The target of the decomposition process is to split large data tables into smaller ones in such a way that the approximation of global decision algorithm (related to the whole table) can be obtained from local ones (related to these smaller tables). One of possible strategies to generate such subtables is based on the so called "templates". Templates can be described as conjunctions of *attribute=value* expressions and defined as follows:

Let **A** be an information system (data table) [5], [12]. A *template* T of **A** is any propositional formula $\wedge p_i$, where p_i are propositional variables, called descriptors, of form $a = v$, a is an attribute of **A** and v is a value from the value set of attribute a. Assuming $A=\{a_1 ,...,a_m \}$one can represent any template $T=(a_{i_1}=v_{i_1})\wedge\ldots\wedge(a_{j_k}=v_{j_k})$ by a sequence $x_1,...,x_m$ where on the position p occurs v_{j_i} for $p = j_1,...,j_k$ and * (don't care symbol) otherwise. An object x satisfies the descriptor $a = v$ if it has value v on the attribute a. The object x satisfies (matches) a template if it satisfies all descriptors of the template. The length $l(T)$ of the template T is the number of different descriptors $a = v$ of the template T. For any template T by $fitness_{\mathbf{A}}(T)$ we denote its *fitness* i.e. the number of objects from the information system **A** matching T. If T consists of only one descriptor a=v we also write $n_{\mathbf{A}}(a,v)$ (or $n(a,v)$) instead of $fitness_{\mathbf{A}}(T)$. If s is an integer then by $Template_{\mathbf{A}}(s)$ we denote the set of all templates of **A** with fitness non-less than s.
We are focusing on the templates which can be used to decompose a given decision table into simpler ones.
They should be as long as possible and at the same time should have fitness of a reasonable size. The decomposition should be performed until the tables attached to leaves of decomposition tree will not be of size feasible for the developed so far rough set strategies [5], [12], [18]. We investigate evolutionary programming methods in searching among the decomposition trees of a given decision table for a tree generating the set of decision rules with the quality greater than a given threshold. One can look on decomposition of a given decision table into subtables corresponding to templates with large length as on decomposition of the domain (universe) of the decision table into suitable sub-domains. The existing rough set methods can be applied to them to obtain decision rules for these sub-domains.
We are presenting some heuristics searching for templates. We also report the results of computer experiments on different date tables. The proposed heuristics has been proved by experiments to be efficient for large data tables.

2.1. Methods for templates generation

2.1.1. MAX method. The purpose of our method is to search for templates with large length with fitness not less than certain lower bound s. We propose a heuristic called "*Max method*". The algorithm starts with *null template* e.g. template with length equal to 0. The template is extended by successive adding descriptors of form $a = v_a$ until the fitness of the template is not less than the fixed value s and the template can be extended. If the current template T consists of i-1 variables then the i-*th* descriptor is chosen as follows: we search among attributes not occurring in the template T for an attribute a and a suitable value v_a that the fitness of the new template $T\wedge (a=v_a)$ is maximal. The construction of the template can be realised efficiently as follows:
Let T be the template with i-1 variables and let $\mathbf{A}_{i-1}=(U_{i-1}, A_{i-1})$ where U_{i-1} is the set of objects matching the template T and A_{i-1} consists of all attributes from A not occurring in the template. The algorithm sorts objects from U_{i-1} with respect to the values of any attribute. Among sorted values of all attributes it chooses the attribute a and the value v with

maximal $fitness_{\mathbf{A}_{i-1}}(a=v)$. Below we show the algorithm in the details.

MAX1 Algorithm

Input: A data table **A**=*(U,A)*, *n*=|*U*|, *m*=|*A*| and an integer *s*.

Output: A template *T* from $Template_{\mathbf{A}}(s)$ with semi-maximal ("large") length (i.e. *l(T)* is close to $max\{l(T): T\in Template_{\mathbf{A}}(s)\}$).

```
I.    T = ∅;
II.   while $l(T) < m$ and $fitness_{\mathbf{A}}(T) > s$ do
III.  begin
   A.    for any attribute $a \notin T$ sort objects from $U$ with
         respect to the values of $a$ to determine the value $v_a$,
         occurring in the table most frequently i.e. $n_{\mathbf{A}}(a,v_a) = max\{n_{\mathbf{A}}(a,v): v\in V_a\}$ where $V_a$ is the value set of $a$;
   B.    Choose the descriptor $a = v_a$ with the highest
         fitness among descriptors constructed in the previous
         step i.e. $n_{\mathbf{A}}(a,v_a)=max\{n_{\mathbf{A}}(b,v_b): b\in A-A(T)\}$ where
         $A(T)$ is the set of attributes occurring in $T$;
   C.    $U$ = the set of objects from $U$ matching the
         template $a=v_a$; $A=A-\{a\}$;
   D.    $T = T \cup \{a=v_a\}$;
IV.   end
```

The described algorithm allows to construct large template efficiently but it generates only one template. We present a modification of the MAX1 algorithm to obtain more than one template. Instead of choosing the descriptor with the largest fitness we consider all descriptors constructed in Step 3.1. and choose one from them randomly according to a certain probability. Then the descriptor $a=v_a$ is chosen to add to *T* with a probability:

$$P(a=v_a)=\frac{n_{\mathbf{A}}(a,v_a)}{\sum n_{\mathbf{A}}(a_i,v_{a_i})}.$$

The MAX1 algorithm can be modified as follows:

MAX2 ALGORITHM

```
I.    $T = \emptyset$;
II.   while $l(T) < m$ and $fitness_{\mathbf{A}}(T) < s$ do
III.  begin
   A.    for any attribute $a \notin T$ sort objects from $U$
         with respect to the values of $a$ to determine the value $v_a$,
         that appear in the table most frequently;
   B.    Choose randomly the descriptor $a = v_a$ with
         the probability $P(a=v_a)=\frac{n_{\mathbf{A}}(a,v_a)}{\sum n_{\mathbf{A}}(a_i,v_{a_i})}$
   C.    $U$ = the set of objects from $U$ matching
         template $a=v$; $A=A-\{a\}$;
   D.    $T = T \cup \{a=v\}$;
IV.   end
```

Both algorithms take $O(m^2.n\ logn)$ time in worst case.

2.1.2. Object weights. The idea is based on setting appropriate weights to all objects in the decision table. These weights describe a potential ability of the object to belong to a "good" (in a sense) template. After attaching the weights to objects the process of the selection starts. Objects are being chosen randomly with respect to their weights. Each time a new object is chosen the fitness of new state is being calculated. If the new state is better the algorithm continues, otherwise it depends on the control variable. The algorithm uses a mechanism of so called "mutation" i.e. some objects are drawn to be removed once upon a time. It allows to avoid the local extrema.

First we construct a list of all objects and count their weights. It takes the majority of the computation. Some initial objects can be randomly chosen from data tables and weights can be attached to them only. Next, the idea of "roulette wheel" for the random choice of objects from the list can be applied i.e. objects with greater weights have the better chance to be chosen. For the new state we calculate the value of its fitness function. If the fitness of the new state is better than that of the previous one we continue with it; otherwise, we remain in the new state with some probability, controlled by the variable that decreases together with the progression of the selection process. The further the algorithm has progressed in calculations, the smaller the probability of choosing the worse state is. It is decided with some frequency whether to remove a randomly chosen object from the set of selected objects. The algorithm can be stopped in many different ways e.g. when chosen objects give worse state for several times or when the control variable decreased below some threshold. During the process the template with highest fitness is being stored in the memory.

Weights of objects reflecting potential similarity of objects [8]. Let **A**=(*U*, *A*) and $x\in U$. For any $y\in U$, we calculate $g_{x,y}=|\{i: x(i)=y(i)\}|$ i.e. the number of attributes that have the same value on *x* and *y*. This number denotes the "closeness" of *y* to *x*. Then, for any attribute $a\in A$, we calculate $w_a(x)=\sum_{y:a(x)=a(y)} g_{x,y}$ and finally the weight $w(x)=\sum_{a\in A} w_a(x)$. We have $w(x)=\sum_{y} g^2_{x,y}$.

Weights of objects derived from attribute value frequency. Let **A**=(*U*, *A*) and $x\in U$. Then for any $a\in A$ let $w_a(x)=n_{\mathbf{A}}(a, a(x))$ and $w(x)=\sum_{a\in A} w_a(x)$.

2.1.3. Attribute weights. The idea is very similar to "object weights" method however appropriate weights are being attached to all attributes in the decision table. Within an attribute each attribute value has its own weight too. In the process of searching for templates first the attributes and next the attribute value are being chosen randomly with respect to their weights. Each time new attribute and attribute value is chosen the fitness of obtained template is being calculated. If the new template is better the algorithm continues, otherwise it depends on the control variable. The algorithm uses a mechanism of "mutation" i.e. with some frequency a randomly chosen fixed attribute value in the template is being changed to "don't care" ('*') value. It allows to avoid the local extrema of fitness function.

Weights. Let **A**=(*U*, *A*), *m*=|*U*|, *n*=|*A*|. We can order the attribute values of *a* according to $n_{\mathbf{A}}(a, v)$ for any $a\in A$. Then by v^a_i we denote the *i*-th value of attribute *a* in this order. In this sense the value v_1^a is the most often occurring value of a in **A**. If $n_{\mathbf{A}}(a,v)=n_{\mathbf{A}}(a,u)$ for $u\neq v$ we randomly choose which value is higher in the order. By $w_{\mathbf{A}}(a)$ we denote $\frac{m}{\sum_{i=1}^{|V_a|} i \cdot n_{\mathbf{A}}(a,v_i^a)}$. As one can easily see $w_{\mathbf{A}}(a) \in (0,1]$. For any value u of attribute a we can define the weight of u as $w_{\mathbf{A}}^a(u)=\frac{n_{\mathbf{A}}(a,u)}{m}$. We have $w_{\mathbf{A}}^a(u)\in(0,1]$ and $\sum_{v\in V_a} w_{\mathbf{A}}^a(v)=1$ for any $a\in A$.

Attribute weight algorithm. The following scheme describes the steps of the algorithm.

```
I.    Initialise template
II.   $i = 1$, $k = 1$, fitness = 0
III.  while not STOP do
   A.   randomly choose $r \in [0,1)$;
   B.   if $r < w_{\mathbf{A}}(a_i)$ and template($i$) = '*' do
      1.   choose an integer $l \in \{1,\ldots,|V_{a_i}|\}$ such that
           $\sum_{k=1}^{l-1} w_{\mathbf{A}}^{a_i}(v_k^{a_i}) \le r \le \sum_{k=1}^{l} w_{\mathbf{A}}^{a_i}(v_k^{a_i})$;
      2.   set template($i$) = $v_l^{a_i}$;
      3.   Calculate new_fitness for the template;
      4.   if new_fitness ≤ fitness * fit_coeff then
             template($i$) = '*' else fitness = new_fitness;
             store(template);
   C.   if $k$ = mutation_coeff then change randomly
        chosen value of template and k=0;
   D.   $i = i + 1$; $k = k + 1$;
   E.   if $i = n$ then $i = 1$
```

One can be interested in searching for templates with maybe smaller fitness but with a high number of fixed attribute values. Then the initial template can be set for example by performing operations from Step 3.1. to Step 3.3. In other cases the most important may be the quality of template with no matter to the "length" of templates. Relatively to this the initial template can be set with "don't care" ('*') values. The *fitness_coeff* and *mutation_coeff* have to be set experimentally. They allow to obtain different kinds of templates: with small or high number of fixed attributes.

2.1.4. Genetic algorithm. Note, that every template can be represented by a binary string of length N (indicating which attributes are fixed) and any object matching it (called a *base object*). This representation will be the most natural in problems as follows*: "Find the largest template matching the object x_1"*. There are also other questions, such as: "*Find the globally best template in this information system*", that can be reformulated in terms of finding the best template matching one specific object. In fact, the algorithm described below finds a good template for a given base object.

Assume given a base object x_b ; we are looking for the best template matching x_b. To do this, we use the following greedy algorithm:

1. Let i := 1, $T = \{ *, *, \ldots * \}$ and let $(b_1 \ldots b_N) = \tau(a_1 \ldots a_N)$ be an ordered list of attributes - the order is represented by a permutation τ.
2. Calculate the size of the template *T*.
3. Add the value b_i (x_b) on attribute to *T*, $i := i + 1$.
4. Repeat from 2 until i = N.
5. Choose the best result found in step 2.

Every locally maximal template can be found using this algorithm - the result depends on the order of attributes. The time complexity of this algorithm can be evaluated as $O(N^2\times m)$. The algorithm described above can be optimised in many ways: for example, objects that do not match the base object in any position can be deleted from database (a pre-processing). Our goal is to find the proper order of attributes and we use genetic algorithms to do this job. Our chromosome will be a permutation τ of length N.

There is no simple and fast way to generate globally good templates by this algorithm - generating the best template for any object is rather unacceptable for the long databases. On the other hand, a *globally good* template means that it matches many objects from the database (in examples presented below - up to 60%). Note, that if we choose any of them as our base object, we find global optimum. So, we can simply choose randomly as many objects as possible and calculate the templates - the largest of them will probably be a global optimum.

2.1.4.1. Function of fitness. The value of the fitness function, F(T) , depends on two parameters viz. the number O_T of objects in the data table matching the given template (base object excluded) and the number A_T of fixed positions:

$$F(T) = O_T \times A_T.$$

Calculating F(T) is the most time-consuming operation, but some techniques can be used to make it faster. For example, we can store fitness values in memory and try to recall them instead of calculating them again.

2.1.4.2. Selection method. When the fitness function is calculated for each chromosome, the selection process begins. First, we normalise the fitness values:

$$FN(x) = \frac{F(x)}{\Sigma F(x)}$$

Next, we use these new values FN(x) as a probability distribution, and we choose the new population randomly, using this distribution (applying the "roulette wheel" algorithm).

The results were slightly better in the case when the elitist strategy was used as an additional technique. The best individual was copied to the new population without changes.

2.1.4.3. Crossing-over. Crossing-over affects chromosomes selected to reproduction with the probability of $P_C = 0.7$. We use the method called PMX (Partially Matched Crossover): the parent permutations are cut at two random points and the fragments are combined and treated as a list of transpositions. Then the parent permutations are treated by these transpositions.

2.1.4.4. Mutations. Probability of a mutation on a single position of the permutation was chosen as 0.1. Mutation of the permutation means a single transposition of two elements like in

$$\{ 1\ \mathbf{2}\ \mathbf{3}\ 4\ 5 \} \rightarrow \{ 1\ \mathbf{3}\ \mathbf{2}\ 4\ 5 \}.$$

In the table we also present some results obtained by application of genetic algorithms [6], [13]. This algorithm generates templates almost included in decision classes i.e. generates approximate rules with dominating decision [3]. The results of experiments are showing that the presented heuristics are efficient even for large data tables.

2.2. Results of experiments (HP Apollo 735)

Size of decision table obj×attr	**Genetic method**		**Object weights**		**Attr. Weights**		**MAX1 method**		**MAX2 method**	
	Time min:sec	**Result** obj×attr	**Time** min:sec	**Result** obj×attr	**Time** min:sec	**Result** obj×attr	**Time** min:sec	**Result** obj×attr	**Time** min:sec	**Result** obj×attr
192×61	0:01	72×2	0:01	30×6		130×2	0:01	130×2	0:01	130×2
	0:01	78×3	0:01	28×8		95×3	0:01	95×3	0:01	95×3
	0:04	78×3	0:01	25×9	Σ=0:01	70×4	0:01	70×4	0:01	70×4
			0:04	14×11		25×9			0:01	45×5
			0:03	25×20		23×11			0:01	27×6
			0:04	18×22					0:01	22×7
			0:04	10×26					0:01	17×8
471×33	0:01	240×3	0:01	200×4	0:01	219×4		241×3		257×3
	0:01	240×3	0:01	161×5	0:01	174×5		216×4		219×4
	0:05	215×4	0:01	142×6	0:01	157×6		163×5		174×5
			0:01	52×9	0:01	119×7	Σ=0:01	143×6	Σ=0:01	157×6
			0:01	31×11	0:01	104×8		104×8		119×7
			0:01	30×12	0:06	63×9		63×9		63×9
					0:04	36×11		43×10		47×10
					0:04	28×12		30×11		35×11
225×490	0:03	86×8	0:01	94×27				153×12		101×12
	0:10	156×5	0:01	74×36				145×13		91×13
	1:38	122×10	0:01	61×47	0:05	50×48		120×17		76×17
			0:01	48×58	0:05	57×56	Σ=0:05	104×20	Σ=0:05	62×20
			0:01	38×78	0:07	42×81		44×65		12×69
			0:01	39×95	0:07	40×82		28×99		10×73
			0:01	33×105				20×120		
			0:01	21×146						
			0:01	14×210						
15534×16		13929×2		13930×2	0:03	13930×2		13930×2		13930×2
	Σ=0:05	7868×3		6877×3	0:02	7869×3		7869×3		7869×3
		5825×3		3405×4	0:06	5284×4		5284×4		5284×4
	0:16	13929×2	Σ=0:35	1527×5	0:04	3905×5	Σ=0:07	3901×5	Σ=0:07	3901×5
	2:39	13929×2		530×7	0:07	2341×6		2341×6		2341×6
				202×8	0:04	1227×7		1227×7		1227×7
				103×9	0:07	680×8		680×8		680×8
					0:07	358×9		358×9		358×9

3. GLOBAL DECISION SYNTHESIS.

Once the process of decomposition has been successfully performed, we obtain a collection of smaller decision tables to which existing rough set tools (i.e. reduct computing and rule generation) can be applied in order to extract knowledge in the form of rules.

The global decision then can be computed from local knowledge of the subtables using various strategies. Among the simplest ones is majority voting where the decision value assigned to a object is the value predicted by the largest number of rules from subtables.

However, experiments with practice data have indicated a need for more advanced synthesis strategies, which are being investigated in our ongoing research. They are discussed in the following section.

4. Advanced alternative strategies for global decision synthesis.

4.1. Tolerance between templates. The basic idea of introducing a tolerance between templates, as a generalization operator, is to "propagate" relevant attribute patterns so that knowledge extracted from subtables may be used to approximate the characteristics of a greater collection of objects. The tolerance relation can be defined as follows:

Let $\mathbf{A}=(U,A)$ be a decision table and $u=\{(a_1,v_1),\dots,(a_k,v_k)\}$ and $v=\{(b_1,w_1),\dots,(b_k,w_k)\}$ $a_i, b_i \in A$, $v_i,w_i \in V_A$ be samples on $\mathbf{A}$. We say that u and v are in a tolerant relation τ iff there exist a function $f\colon A \to A$ such that $f(\{a_1,\dots,a_k\}) = \{b_1,\dots,b_k\}$ and $f(a_i)=b_j) \Rightarrow (v_i=w_j)$. By $u_\mathbf{A}$ we denote the set of objects of $\mathbf{A}$ that match u, i.e. $u_\mathbf{A}=\{\boldsymbol{x}\in U\colon u \subseteq \mathrm{Inf}_\mathbf{A}(x)\}$. Then we can define the set of objects that match u in a tolerant sense as $\tau(u)_\mathbf{A} = \{\boldsymbol{x}\in U\colon \exists v\ u\tau v \text{ and } x\in v_\mathbf{A}\}$.

Having found a set of templates we calculate the sets of objects that match them in tolerant sense. Such sets then can be used to cover the decision classes of the original data table and therefore to approximate the global decision. The problem of how to choose tolerance relations that give as accurate an approximation of the global decision as possible is one of the main subjects of our future projects.

4.2. Multi-level decision synthesis. In this approach the global decision is not computed directly from the smaller decision function induced by generated subtables, instead the calculation will be done in a hierarchical, tree-like manner where the decision functions on a upper level a computed from those on lower levels by some sets of operators, the decision subtables being the leaves and the global decision being the root. Once again, there are many methods of how to computer the decision on a upper level from those that are below. We propose a general paradigm to this problem by means of the mereology theory as follows.

The idea of a mereological approximative synthesis of a solution is based on the observation that in many cases a solution to a problem posed under uncertainty is synthesized from "pieces" characterized by the degree in which they are close (or, are "parts" of) to some standard components of the solution, the latter often "ideal" i.e. not known fully to the problem solver. To characterize the degree of being a part, we introduce a function M (called a rough inclusion [9]) with M(x,y) = the degree in which x is a part of y. Usually, the range of M is the unit interval [0,1] and M(x,y)=1 means that x is a full (possibly improper) part of y. In particular, the function $M(T,D) = |[T] \cap [D]| / |[T]|$ generates a rough inclusion (see [10]); it is used to measure fitness of a template T in approximating a decision class D.

The encouraging quality of approximations used with employing this rough inclusion compels us to sketch the idea of further investigations on approximating decision classes via templates along rough mereological lines. Our approach has to do with a sequential exhausting of a decision class D by a sequence of templates $(T_i)_i$. We can regard the successive steps of this approach as a sequence of agents associated to random choices of objects in D and working cooperatively in the way described above.

More general mereological schemes of synthesis are constructed in the following way (cf. [13]). Given a hierarchy (assume a tree) of agents, any agent *ag* is equipped with its universe of objects *U*(*ag*), a rough inclusion *M*(*ag*), a language (say, of unary predicates) *L*(*ag*) interpreted in *U*(*ag*), and a set *St*(*ag*) of standard objects (standards). Given a requirement Φ, the agents negotiate the condition ($\Phi(ag)$, $st(ag)$, $\varepsilon(ag)$) for any *ag* such that: (i) $st(ag) \in St(ag)$; (ii) $st(ag)$ satisfies $\Phi(ag)$; (iii) for any *ag* and its children *ag*(1),..,*ag*(*k*) (if there are any), if $M(ag(i))(x(i), st(ag(i))) \geq \varepsilon(ag(i))$ for $i = 1, 2,.., k$, then the parent *ag* constructs from *x*(1),.., x(*k*) the (unique) object *x* such that $M(ag)(x, st(ag)) \geq \varepsilon(ag)$; (4) for the root agent *R* of the hierarchy, if $M(R)(x(R), st(R)))\geq \varepsilon(R)$ then *x*(*R*) satisfies the requirement Φ in satisfactory degree.

Given a database D, a test family {D(*ag*): *ag*} of subtables (agents indexed) is selected; standards at any agent *ag* are template covering of D(*ag*) found by means of experiments described above. For a new object *x*, the distance $\varepsilon(ag)$ from *x* to standard templates are evaluated and a hierarchical merging system of D(*ag*) is created. At each merging step, e.g. ag_1 and ag_2, new template coverings for the table $D(ag_1) \cup D(ag_2)$ are found based on the fitness function. The object *x* is classified on the basis of distances from *x* to the final standard templates found for $D_0 = \cup\{D(ag):ag\}$.

Let us observe that this interplay between the mereological synthesis theory and the approximation problem is mutual: not only the general mereological synthesis scheme is applicable here in a natural way but also the experiments with real data tables will prove the validity of the theoretical scheme and allow for experimental testing of theoretical conjectures concerning the propagation rules for uncertainty coefficients. Observe that two extreme levels: the data table D and leaf subtables of D are given but the intermediate level where findings from subtables are synthesized presents a theoretical and practical challenge. Our research project is directed towards this goal.

5. CONCLUSION

A general schema for extracting rules from large data tables is discussed. We propose a method to attack the difficult problem of knowledge discovery from large databases. Experiments so far conducted have confirmed that satisfactory results can be obtained in reasonable time. Further theoretical research to extend the presented concepts is being pursued and experiments on very large databases are scheduled.

6. REFERENCES

[1] Fayyad U..M., Uthurusamy R., 1995. *Proc. of the First International Conference on Knowledge Discovery and Data Mining*, Montreal, August 20-21, 1995, AAAI Press.

[2] Holland J.H., 1992. *Adaptation in natural and artificial systems*. The MIT Press, Cambridge.

[3] Mannila H., Toivonen H., Verkamo A.I., 1994. “Efficient Algorithms for Discovering Association Rules”. In: U. Fayyad and R. Uthurusamy (eds.): *AAAI Workshop on Knowledge Discovery in Databases*, pp. 181 - 192, Seattle, WA, July 1994.

[4] Michalski R., Tecuci G., 1994. *Machine Learning. A Multistrategy Approach.* Morgan Kaufmann, San Mateo, CA.

[5] Mollestad T., Skowron A., 1996. "A Rough Set Framework for Data Mining of Propositional Default Rules". To appear in: *Proc. ISMIS-96.*

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

[7] Piatetsky-Shapiro G., 1991. "Discovery, Analysis and Presentation of Strong Rules". In: Piatetsky-Shapiro G. and Frawley W.J. (eds.): *Knowledge Discovery in Databases*, pp. 229 - 247, AAAI/MIT.

[8] Polkowski L., Skowron A., Synak P., Wróblewski J., 1995. *Searching for Approximate Description of Decision Classes*, manuscript.

[9] Polkowski L., Skowron A., 1996. "Rough Mereological Approach to Knowledge-based Distributed AI". In: J.K. Lee, J. Liebowitz, Y. M. Chae (eds.): *Critical Technology. Proc. of The Third World Congress on Expert Systems*, pp. 774-781, Seoul 1996, Cognizant Communication Corporation, New York.

[10] Polkowski L., Skowron A., 1996. "Rough mereology: A New Paradigm for Approximate Reasoning". To appear in: *Journal of Approximate Reasoning*

[11] Quinlan J.R., 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA.

[12] Skowron A., 1995. "Synthesis of Adaptive Decision Systems from Experimental Data". In: J.Komorowski, A. Aamodt (eds.): *SCAI-95. Proc. Fifth Scandinavian Conference on Artificial Intelligence,* pp.220-238, IOS Press, Amsterdam.

[13] Skowron, A., Polkowski, L., 1995. "Rough mereological foundations for analysis, synthesis, design and control in distributive system", Proc. *Second Annual Joint Conference on Information Sciences*, pp. 346-349, Sept. 28 - Oct. 1, 1995, Wrightsville Beach, NC.

[14] Skowron A., Rauszer C., 1992. "The discernibility matrices and functions in information systems". In: R. Słowiński (ed.): *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Sets Theory*, pp.331 - 362, Kluwer, Dordrecht.

[15] Smyth P., Goodman R.M., 1991. "Rule Induction Using Information Theory". In: Piatetsky-Shapiro G. and Frawley W.J. (eds.): *Knowledge Discovery in Databases*, pp. 159 - 176, AAAI/MIT.

[16] Toivonen H., Klemettinen M., Ronkainen P., Hatonen K., Mannila H., 1995. "Pruning an Grouping Discovered Association Rules". In: Mlnet: *Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pp. 47 - 52, Heraklion, Crete, April 1995.

[17] Uthurusamy R., Fayyad U.M., Spangler S. 1991. "Learning Useful Rules from Inconclusive Data". In: Piatetsky-Shapiro G. and Frawley W.J. (eds.): *Knowledge Discovery in Databases*, pp. 141 - 157, AAAI/MIT.

[18] Ziarko W.(ed.), 1994. *Rough Sets, Fuzzy Sets and Knowledge Discovery*, Workshops in Computing, Springer Verlag & British Computer Society.