Searching for approximate description of decision classes

L.Polkowski, A. Skowron, P. Synak, J. Wróblewski

Institute of Mathematics
Warsaw University
Banacha 2
02-097 Warsaw
Poland

polk@mimuw.edu.pl
skowron@mimuw.edu.pl
synak@mimuw.edu.pl
jakubw@alfa.mimuw.edu.pl

Abstract

We discuss a searching method for synthesis of approximate description of decision classes in large data tables (decision tables). The method consists of the following stages: (i) searching for basic templates which are next used as elementary building blocks for decision classes description; (ii) performing templates grouping as a pre-processing for generalisation and contraction; (iii) generalisation and contraction operations performed on the results of grouping. The main goal of the method is to synthesize the approximate description of decision classes. The control in the searching method is focused on attempts to reduce uncertainty in approximate description of decision classes. On the other hand uncertainty in temporary synthesised descriptions of decision classes is the main "driving force" for the searching method. In the paper we concentrate on the different methods for template generation and on the discussion of performed computer experiments. We also present a general searching scheme for approximate description of decision classes as well as we point out the relationship of our approach to the rough mereological approach to the synthesis of complex objects. The presented method can be adopted for synthesis of adaptive decision algorithms what is the goal of our ongoing research project.

Key words: adaptive decision systems, data mining systems and tools, templates, generalisation and contraction, genetic algorithms, rough mereological synthesis scheme

1. Introduction

There exist many methods for synthesis of decision rules and/or finding regularities in data tables [5], [8], [11], [12]. However, these methods are usually efficient for relatively small tables. Data mining community is investigating [4], [6], [8], [15], [16], [17] new methods efficient for large data tables. Our approach is based on the decomposition of large data tables into smaller ones in such a way that the approximation of global decision algorithm (related to the whole table) from local ones (related to these smaller tables) can be obtained. The set of these smaller tables can be treated as a set of generators that used together with appropriate operators, like *grouping, generalisation, contraction,* can be applied for achieving the sufficient approximation of the global decision algorithm. We propose to use as the generators the so called "templates". Templates can be described as conjunctions of *attribute=value* expressions. We look for templates with high quality e.g. characterised by the number of objects supporting a template times the number of *attribute=value* pairs describing the template. We discuss here two algorithms for template generation; some algorithms can be based on reduct approximation [12]. The templates are matched against the given decision class and finally the templates included in the decision class to the highest degree are chosen. One can use several measures to estimate the quality of template inclusion into the decision class, e.g. the ratio of objects supporting a given template and included in the decision class to all objects supporting template or/and the ratio of objects supporting the template and included in the decision class to the number of objects in the class. These numbers are treated as parameters which should be tuned up in the process of synthesis of approximate description of the decision class. Strategy in choosing templates can also depend on the estimation of how promising these templates can be for the construction of the decision class approximation by application of different operators like grouping, generalisation, contraction. The grouping procedures are executed after templates are chosen. In this step the following principles should be observed: (i) two templates covering almost the same objects from the class and almost disjoint on objects not belonging to the class should be separated by grouping procedures; (ii) the family of intersections of different templates in one group should not be "close" to the partition of the decision class into one element sets. Groups of templates are received as the results of these procedures. Different approximate coverings of the decision class are constructed by applying generalisation to these groups. Next, the grouping procedures are executed again as a preprocessing for contraction. The process continues until a description of the decision class with a sufficient quality is constructed; otherwise, the process of construction is estimated as unsuccessful and it is redone from some previous construction level by applying another grouping, generalisation or contraction strategies. The simplest cases of generalisation and contraction are union and intersection, respectively. More advanced operations of generalisation can be obtained by introducing tolerance relations. This issue will be discussed elsewhere. In this work, we concentrate on the different methods for template generation and on the discussion of performed computer experiments. We also present a general searching scheme for approximate description of decision classes as well as we point out the relationship of our approach with rough mereological approach to synthesis of complex objects [9], [10], [13].

2. Preliminaries

An *information system* [7] is a pair $\mathbf{A}=(U, A)$, where U is a non-empty set called the *universe* and A is non-empty set of *attributes*. Any $a \in A$ is a mapping, $a: U \to V_a$, where V_a is the *value set of a*. A *decision table* is an information system of the form $\mathbf{A}=(U, A \cup \{d\})$, where $d \notin A$ is a distinguished attribute called the *decision* (with integer values). The decision d determines the partition $\{X_1, ..., X_{r(d)}\}$ of the universe U, where $X_i=\{x \in U$:

$d(x)$=i} for 1≤i≤$r(d)$. The set X_i is called the *i-th decision class of* **A.** For $B \subseteq A$, the *B-indiscernibility relation* [7] is defined by $IND(B)=\{(x,x') \in U \times U$: for any $a \in B$, $a(x)=a(x')\}$. The *set of reducts* [7] is denoted by *RED*(**A**) and contains all minimal subsets (with respect to inclusion) $B \subseteq A$ such that $IND(A)=IND(B)$.

A *template for* $A=\{a_1,...,a_n\}$ is a string $v_1,...,v_n$ where $v_i \in V_{a_i} \cup \{*\}$ for any i and '*' is a special "*don't care*" character. The i-th position of a template corresponds to the attribute a_i. If '*' appears on the i-th position, then it means that the template "ignores" attribute a_i. An object x is *matching a given template* iff $a_i(x)$ is equal to the i-th element of the template if this element is different from *, for any i ([2]).

Example of a decision table and a template (shaded, size 3×3):

Objects:	Attributes:						Decision:
	a_1	a_2	a_3	a_4	.....	a_N	d
x_1	5	0	1	1.16		black	1
x_2	4	0	0	8.33		red	0
x_3	5	0	1	3.13		black	1
x_4	5	0	0	3.22		black	1
... ...							
x_m	1	1	0	3.24		red	0

The template: (5, 0, *, *, …, black) matches objects x_1, x_3, x_4. We are interested in *good* templates, that is in templates containing as many fixed places as possible and matching as many objects as possible.

3. Methods for template generation

3.1. Finding templates using object weights

3.1.1. Overview

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 then the algorithm continues, otherwise it depends on the control variable. The algorithm uses a mechanism of "mutation" i.e. some objects are drawn to be removed once upon a time. It allows to avoid the local extrema.

3.1.2. Weights

Let **A**=(U, A) and $x \in U$. For any $y \in U$, we calculate $g_{x,y} = |\{a \in A: a(x) = a(y)\}|$ 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 $w(x) = \sum_y g_{x,y}^2$.

Our experiments show that these weights allow for very satisfactory clustering of objects into templates while more "naive" values of weights decrease the quality of results.

3.1.3. The algorithm

First we construct a list of all objects and count their weights. It takes the majority of the computation time. In large data tables some initial objects can be randomly chosen and weights can be attached only to them. Next, the idea of "roulette wheel" [1] 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. The corresponding data structures (see below) are updated when a new object is chosen. For the new state we calculate the value of its fitness function.
The calculation of fitness function is very fast for data structures proposed here. 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 is the probability of choosing the worse state. 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.

3.1.4. Data structures

The following data structure can be used for storing the information about chosen objects. We build the table of length n of pointers to the ordered lists of structures containing two pieces of information: *the value of the attribute related to the list* and *the number of the occurrences of this value in the set of chosen objects*. If a new object x is being chosen then for any attribute a we increase the number of occurrences of $a(x)$. If $a(x)$ has not occured yet then we add it to the list. When we remove an object from the set of chosen objects , the corresponding values are correspondingly decreased. If a certain value of the attribute does not exist any longer then we remove the corresponding structure from the list. The lists are being sorted in the process of adding new objects according to the number of occurrences of attribute values. The searching time through the lists is short because for any attribute the most frequent values occur at the beginning of the list. Additionally, an integer table of length n can be created which for any attribute stores the number of its values (for the set of chosen objects). Then the fitness can be computed, e.g. by counting the number of 1 in this table and multiplying the result by the number of chosen objects.

3. 1. 5. Example. Consider the following set of chosen objects:

x1: 1 5 0 0 1
x2: 3 3 0 1 1
x3: 1 2 0 3 1

The following data structure can be created:

2	3	1	3	1

1;2	5;1	0;3	0;1	1;3
3;1	3;1		1;1	
	2;1		3;1	

The value of fitness can be computed as follows:

- the number of 1 in the first table =2
- the number of objects =3
- fitness=2*3=6

3.2. Finding templates using genetic algorithms

3.2.1. Genetic algorithms in optimisation problems [2], [3]

We are given a state space S (finite, but large) and the function: $f: S \to \boldsymbol{R}^+$. Our goal is to find x_O: $f(x_O) = max\, f(x)$, $(x, x_O \in S)$. Elements of set S are "individuals". The main idea of classical genetic algorithms is based on the Darwinian principle of natural selection. We treat the value of the function f as ability ("fitness") to survive in the environment, and we simulate the process of evolution as follows (i) we choose the representation scheme (and change "individuals" to "chromosomes" - usually bit strings). We randomly choose the set of chromosomes as an initial population; (ii) we calculate "fitness" $F(c)$ of each chromosome c as a value of $f(s(c))$, where $s(c)$ is the individual coded by c. Then we create new population, replacing the chromosomes with low fitnesses by those with higher fitnesses; (iii) we randomly choose a new population by mutations (small, random modifications of chromosomes) and crossing - overs (exchange of "genetic material" between some pairs of chromosomes); (iv) we repeat (ii)-(iii) with the new population. The result of evolution is the best individual x_{max} which is usually nearly as good as the global optimum x_O.

3.2.2. Representation

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 = { *, *, … * } and let (b_1 ... b_N)= τ(a_1 ... 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.

3.2.3. Function of fitness

The value of the fitness function, F(T) , depends on two parameters viz. the number O_T of objects in the datatable 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.

3.2.4. Selection method

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

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

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.

3.2.5. 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 [1]): 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.

3.2.6. 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

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

3.3. Comparison of algorithms for template generation

The algorithms described above were tested on several examples of real databases. Computation times for th two methods are presented below:

Size of decision table obj×attr	Genetic method			Weight method	
	Parameters gen×pop	Time min:sec	Result obj×attr	Time min:sec	Result obj×attr
192×61	15×10	0:01	72×2	0:05	25×9
	25×20	0:02	78×3	0:05	23×21
	100×50	0:16	78×3	0:04	25×20
471×33	15×10	0:01	240×3	0:06	36×11
	25×20	0:03	240×3	0:04	30×12
	100×50	0:27	215×4		
225×490	15×10	0:12	86×8	0:13	35×121
	25×20	0:24	156×5	0:30	33×129
	100×50	3:34	122×10	0:12	42×101
15,534×15	15×10	0:25	8,732×2	28:50	275×6
	25×20	1:20	5,825×3		
	100×50	12:11	5,825×3		

Computed on HP Apollo 715

Size of the decision table = number of objects and attributes in decision table;
parameters = number of steps (generations) of the genetic algorithm, the population size;
the result = the size of the best template found.

The results show that the genetic method produces templates with more objects and fewer attributes than the weight method. On the other hand, the time complexity of the first method is $O(N^2 \times m)$, whereas the second method has the time complexity $O(N \times m^2)$, where N is the number of attributes and m is the number of objects. We can therefore decide on the better one for any particular problem.

4. General scheme

We may use templates produced by these algorithms e.g. for searching for regularities in databases. However, our main goal in knowledge discovery is to find rules binding attributes and decisions i.e. a decision algorithm. To this end , we follow the steps described below. Suppose that we are given a decision table **A**. We are interested in the description of its i-th decision class by a set of decision rules i.e. by the decision algorithm for this class.

Step 1. We produce the set of templates covering the decision class, i.e. most objects from the class match one of templates while as few as possible objects from other classes match them. Both algorithms described above can be adapted to this new kind of a template: we can simply change the formula for the template fitness.

Step 2. We combine the templates obtained in the previous step into groups and apply the operations of generalisation and/or contraction. We can use one of these operators in an adaptive way to obtain the decision algorithm of better quality. We repeat Step 2 until the quality of obtained decision algorithm is sufficiently good.

Step 3. If the quality of decision algorithm is not satisfactory **then** we repeat from Step 1 **else** we can use the algorithm (maybe after some post-processing) as the approximate definition of the i-th decision class.

The generalisation operator may be understood in the simplest cases as the union of objects matching one of the templates, alternatively as a minimal template including all the templates. The contraction, in the simplest case, can be defined as the intersection of the templates. For both operators, we may take into account e.g. weights attached to the attributes: the "less important" attribute can be generalised or fixed etc. We may also employ additional techniques concerning with the notion of a tolerance, or uncertainty, of template description.
The quality of decision algorithm obtained by this method depends on how it fits the decision class, and also on its complexity - we tend to produce rules as simple as possible. Our results presented here seem to speak decidedly in favor of this method.

4.1. Finding decision templates using genetic algorithm.

We produce the decision templates using the genetic algorithm similar to the one described in Section 3.2. We have only to change the fitness formula:

$$F(T) = c_1 \cdot \left(\frac{O_+}{O_+ + O_-} \right)^3 + c_2 \cdot \left(\frac{O_+}{O_+ + O_{used}} \right)^3 + \frac{O_+}{O_{class}}$$

where:
O_+ = the number of objects matching the template and belonging to the decision class;
O_- = the number of objects matching the template and not belonging to the decision class;
O_{used} = the number of objects matching the template and belonging to the decision class, but already covered by the previously generated templates.

O_{class} = the number of objects belonging to the decision class;
c_1, c_2 - parameters.
Using this fitness function, we can cover whole decision class with templates. We start with a random object in the class as a base object, then the obtained template is stored in the memory. The next base object is chosen randomly from the class, but we do not take into account objects covered by the already produced templates.
Observe that a high value of c_1 forces the templates chosen not to cover any object outside the class, but the templates are shorter and we have to use more of them. If the value of c_2 is high then obtained templates are separated. We can use the conjunction of obtained templates as our approximate definition of the decision class.

The results of our experiments are shown below:

Size of decision table	Size of decision class	Time min:sec	c_1	Result	Efficiency of decision algorithm
192×61	35	0:13	0.7	3/0 + 3/0 + 2/0 + 3/0 + 2/0 + 3/0 + 2/0 + 2/0 + 2/0 + 2/0 + 1/0 + 1/0 + 2/0 + 2/0 + 1/0 + 2/0 + 2/0	35/0
		0:02	0.3	30/131 + 23/82	35/148
471×33	248	1:25	0.7	7/0 + 3/0 + 8/0 + 2/0 + ... [total: 90 templates]	248/0
		0:52	0.4	4/0 + 5/0 + 189/174 + ... [total: 54 templates]	248/174
225×490	163	4:24	0.7	77/7 + 7/0 + 5/0 + ... [total: 28 templates]	163/7
		0:47	0.4	81/1 + 147/45 + 2/0 + 1/0 + 1/0	163/45
15,534×15	830	114:24	0.5	2/0 + 1/0 + 4/0 + 2/0 + ... [total: 353 templates]	830/0
		0:25	0.3	830/14,704	830/14,704

Each component represents one template. X/Y means that the template matches X objects inside the class and Y outside the class.
Efficiency of decision algorithm = number of objects matching the conjunction of obtained templates: (inside the class)/(outside the class).
The results show, that in real cases we have to be very careful in choosing value of c_1. If the value is too high, then we obtain too many templates; if the value is too low, the obtained decision algorithm is useless: it cannot recognise properly too many objects. On the other hand, there are many templates covering only 1 or 2 objects. These "exceptions" can be omitted if we are interested in short and general rules.

5. Rough mereological approach to synthesis of complex objects

We will put the specific solutions presented in Section 4 into a broader perspective by discussing a rough mereological framework [9], [10], [13]. 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 in Section 4 as an ingredient 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. The approach in Section 4 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 (Φ(*ag*), *st*(*ag*), ε(*ag*)) for any *ag* such that: (i) $st(ag) \in St(ag)$; (ii) *st*(*ag*) satisfies Φ(*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.

In our context of approximating large decision classes, the above scheme can be implemented as follows; any (abstract) agent *ag* is attached to a subtable D(*ag*) of the datatable D, for a family (chosen at random) of subtables. Standards at *ag* are defined as template coverings of test subtables of the same size as D(*ag*) found from experiments. Test experiments are then used to learn values of uncertainty coefficients ε(*ag)* and template coverings in a new case are found which are satisfactorily close to selected standards by using an adequately defined fitness function.

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 datatables 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 datatable D and leaf subtables of D are given but the intermeduate level where findings from subtables are synthesized presents a theoretical and applicational challenge. Our research project is directed towards this goal.

Conclusions

We have presented some methods for template generation and a general scheme for approximate description of decision classes based on the notion of a template. An interesting aspect of our approach is that on the one hand searching methods are oriented towards uncertainty reduction in constructed descriptions of decision classes but on the other hand uncertainty in temporary synthesized descriptions of decision classes is the main "driving force" for the searching methods. The results of computer experiments are showing that the presented methods for template generation are promising even for large tables; however, much more additional work should be done on strategies for the construction of approximate decision class descriptions on the basis of the general mereological scheme. The presented results create the basis for further experiments and research on adaptive decision system synthesis.

References

[1] Goldberg D.E., 1989. *GA in Search, Optimisation, and Machine Learning*. Addison-Wesley.

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

[3] Koza J.R., 1992. *Genetic Programming: On the Programming of Computers by Means of the Natural Selection*. The MIT Press.

[4] 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.

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

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

[7] Pawlak Z., 1991. *Rough sets: Theoretical aspects of reasoning about data*. Kluwer, Dordrecht.

[8] 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.

[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 introduction 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: Familiarisation 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: In: Piatetsky-Shapiro G. and Frawley W.J. (eds.): Knowledge Discovery in Databases, pp. 141 - 157, AAAI/MIT.

[18] Wróblewski J., 1995. *Finding minimal reducts using genetic algorithms.* Proc. of the Second Annual Join Conference on Information Sciences, pp.186-189, September 28-October 1, 1995, Wrightsville Beach, NC.

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