THESIS ON INFORMATICS AND SYSTEM ENGINEERING C131

From Determinacy Analysis to Zero Factor Free Determinacy Analysis and Universal Generator of Hypotheses: Development of Algorithms

GRETE LIND

PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

This dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer Science on October 10, 2017.

Supervisor: Prof. Rein Kuusik
Department of Software Science
Tallinn University of Technology

Opponents: Prof. Sergei O. Kuznetsov, Doctor of Sciences
School of Data Analysis and Artificial Intelligence
Faculty of Computer Science
National Research University Higher School of Economics
Russia

Jilles Vreeken, Ph.D.
Independent Research Group Leader (W2)
Exploratory Data Analysis
Cluster of Excellence MMCI
Saarland University, Saarbrücken, Germany

Defence of the thesis: December 18, 2017, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

/Grete Lind/

European Union
European Social Fund

Investing in your future

ISSN 1406-4731
ISBN 978-9949-83-161-6 (publication)
ISBN 978-9949-83-162-3 (PDF)

INFORMAATIKA JA SÜSTEEMITEHNIKA C131

Determinatsioonanalüüsist nullfaktorivaba determinatsioonanalüüsi ja universaalse hüpoteeside generaatorini: algoritmide arendus

GRETE LIND

TTÜ
KIRJASTUS

Table of Contents

ABSTRACT

Data mining (DM) is often defined as finding hidden information in a database. The two “high-level” primary goals of data mining are prediction and description. A descriptive model serves as a way to explore the properties of the data examined, not to predict new properties (like the predictive model).

The goals of prediction and description are achieved by using different DM tasks, including classification and association rules. Classification maps (classifies) data into predefined groups or classes. This method comes from machine learning, in DM it is used mainly for prediction. An association rule mining uncovers relationships among data, identifying specific types of data associations. It is considered to be a descriptive task.

Both tasks produce IF-THEN rules. The difference is that classification rules are produced for determined class(es) only and class attribute(s) is(are) separated from other attributes; in case of association rules all attributes can be on either side of the rule (not in the same rule) and (generally) the conclusion is not determined in advance.

Usually these tasks are solved separately and for different purposes. There also exist hybrid approaches that combine classification and association rules (either for descriptive or predictive purpose), their results are presented in the form of classification rules. We have created an approach (ZFF DA) that produces both kinds of the rules at the same time (for descriptive purpose). Producing both kinds of rules is different from hybrid approaches as well as classification and association rule discovery because each of them finds one kind of rules.

This work presents developments of descriptive data mining methods called Determinacy Analysis (DA) and Generator of Hypotheses (GH).

Both methods had been created in Soviet time. Being separated from the Western research, they had their own underlying concepts and theories. In this work we have shown the correspondences of the concepts from these methods with the ones that are widely known in data mining area. These correspondences facilitate to share our ideas as well as make findings about those well-known concepts usable for us.

DA is a system of methods for the analysis of rules. It tries to answer the questions: “Who are they (objects of the class)?”, “How can we describe them?”, “What distinguishes them from others?”. DA can be seen as a classification method, in addition it can be related to association rules.

In this thesis DA is developed in order to overcome different drawbacks typical of its different approaches, mainly towards reducing redundancy. Zero factors (ZF) are the elements that cause redundancy and must not to appear in the antecedents of the rules. We have defined two types of them.

Presented developments of DA are:

- Step-by-step method: allowing free length for non-intersecting rules, it helps to reduce redundancy.
- First algorithm for getting intersecting rules produces a possibly small set of (possibly short) rules.
- DSR-approach: instead of one description out of (usually huge) number of all possible ones, it finds all non-redundant rules that can be used for finding a suitable cover.
- Zero-factor-free (ZFF) DA finds all non-redundant rules for all existing classes (not for "target" class only) and enriches these (classification) rules with positive and negative association rules.

Generator of hypotheses is a DM method that uses possibilities offered by algorithm MONSA. We have shown that it finds all closed sets.

MONSA is changed to find all minimal generators with their closed sets, "excluded factors" and class. The elements between a closed set (CS) and its generator form a consequence for an association rule where the generator is an antecedent. Excluded factors are such elements that do not occur in CS, they form a consequence of a negative association rule. Having found that the elements between CS and generator correspond to (one kind of) ZF (in DA) and that a class can be detected the same way as ZF, we have created a MONSA-based algorithm for ZFF DA that produces three kinds of non-redundant rules with a common antecedent: classification rules, positive and negative association rules.

As a CS with all its minimal generators forms an equivalence class (EC), we have created an algorithm for finding all ECs as well.

Finally, we have gathered the tasks solvable by GH and DA under the framework called Universal Generator of Hypotheses.

MONSA and all other algorithms presented in this thesis are based on the theory of monotone systems. We see that ZFF DA algorithm can be used as a basis for different further developments listed in the thesis.

ACKNOWLEDGEMENTS

I am deeply grateful to my supervisor prof. Rein Kuusik for long-time guidance and collaboration. He has always been supportive, patient and very positive. Each time we talked about my thesis, he fulfilled me with enthusiasm and hope.

I would like to thank my good colleagues from the Department of Informatics for making it the best place to work.

I would like to acknowledge organisations that supported me during my PhD work: Tallinn University of Technology, Estonian Information Technology Foundation and ICT Doctoral School.

I thank my family and friends and especially my dear Mette for being with me through all these years.

Thank you all!

Abbreviations

AR	association rule
ARM	association rule mining
CS	closed set
DA	determinacy analysis
DM	data mining
DSR	determinative set of rules
EC	equivalence class
FT	frequency table
GH	generator of hypotheses
ML	machine learning
MONSA	MONotone System Algorithm
MS	monotone systems
UGH	universal generator of hypotheses
ZF	zero factor
ZFF	zero-factor-free

1 INTRODUCTION

The wide area of data mining (DM) can be divided into predicting and describing. Unknown or future values of variables of interest are predicted using known data. Description focuses on finding human-interpretable patterns describing the data. A descriptive model serves as a way to explore the properties of the data examined. In the context of knowledge discovery in databases (KDD), description tends to be more important than prediction. (Fayyad, Piatetsky-Shapiro, & Smyth, 1996)

DM uses methods from many other disciplines: machine learning, pattern recognition, statistics, …. Different methods from machine learning (ML) are used mainly for prediction purposes. The most important ones of them are classification and clustering. The former represents supervised learning and the latter – unsupervised learning. In case of supervised learning there exist data for which the right answer (class, function value, …) is known and the learner is trained to give a right answer to each input in training set. It is assumed that it works on unseen data as well. In case of unsupervised learning there are no correct answers, usually the task is to partition data in some appropriate way.

The classification rules represent found knowledge in the form of IF-THEN rules. The IF-part contains a predicate that can be evaluated as true or false against each tuple in the data. THEN-part shows the target class.

The rules are used mainly for *Classification task*: to find the rules for classification of unknown object(s) on the basis of the learning examples. Additionally the classification rules are used for *Data Analysis and Data Mining task*: to use the found rules for describing the class under analysis, answering the questions: "Who are they (objects of the class)?", "How can we describe them?", "What distinguishes them from others?".

In this work we will use classification rules for description, although primarily they are used for prediction purposes.

Besides the classification rules we will deal with association rules. The difference between these two is that for classification rule the target class is given, whereas in case of association rule each considerable item can be on either side of the rule.

One problem solved in this work is to get rid of redundancies appearing in the rules. For example, if the description of a class contains two characteristics: "Are you living in the countryside?" with a value "Yes", and "Do you have cows?" with a value "Yes", then the first one is redundant because having cows means that one lives in the country (if everyone having cows lives in the country). This way we get a (positive) association rule “has a cow” $\Rightarrow$ “lives in the countryside”.

Besides positive association rules we can easily find the negative ones also (like “has a cow” $\Rightarrow$ NOT “is a frequent traveller”). In some cases negative rules are more valuable than positive ones.

Usually classification rules and association rules are treated separately. In addition to classification and association tasks there exist hybrid approaches that combine the two. The rules found by the hybrid methods have a form of classification rule. We have developed an approach (called zero-factor-free determinacy analysis) that finds for the same antecedent (left side of the rule) three types of conclusions (i.e. right sides of the rule): class, “accompanying” factors and “excluded” factors. The last two correspond to the association rules (positive and negative, accordingly), while the first one is a classification rule. By producing different kinds of rules our approach is different from hybrid approaches as well as classification and association rule discovery because each of these methods finds one kind of rules.

This work is about descriptive data mining methods called Determinacy Analysis (DA) and Generator of Hypotheses (GH). These methods are not intended for big data analysis. Relations between the concepts used in these methods and more widely known concepts of data mining are shown in the thesis. Presented algorithms are based on the theory of monotone systems (MS). Different tasks solvable by GH and DA are gathered under the framework called Universal Generator of Hypothesis (UGH).

1.1 Motivation

Coming from the former USSR, our research themes after achieving an independence are to a great extent related to the topics and approaches that were actual in these times (for example, Determinacy Analysis). Since 1970s in the Institute of Informatics (at Tallinn University of Technology) there has been evolved our own school in data science area, the so-called school of prof. Leo Võhandu. Together with J. Mullat they have developed the theory of monotone systems which was the basis for many methods created here (e.g. Mullat’s kernels, data table reordering methods). As these methods were used mainly by the public opinion and market research companies (carrying out the questionnaire surveys), all the methodology and algorithms created at that time issue from a data table, i.e. the work data were in the form of object-attribute system, not drawn from databases. The approaches and algorithms presented in this work are developed largely based on that assumption, theoretical basis and manner of thought.

The first versions of Determinacy Analysis and Generator of Hypotheses were used in the Computing Centre of Estonian Radio, where a group of sociologists were working. Due to their interestedness the problems of application and further development of the methods were then very actual. Unfortunately the beginning of Estonian independence with its economic difficulties and reforms did not allow to seriously deal with it. Further development of GH and DA rose on the agenda again only in the beginning of 2000s. Also a group of enthusiasts emerged, who were ready to apply these methods (first of all, Saar Poll OÜ, Estonian Academy of Security Sciences). Encouraged by them the development of the method and algorithms (once) again got off the ground, new ideas and approaches arose, which were published in research papers.

The base algorithms and approaches created in 90s were based on the concepts that had been created by the earlier school (prof. Leo Võhandu and others). As the USSR was a closed system, the foreign contacts and access to the foreign publications of the field were missing as well as the foreign scientists could not access our papers. Due to this, after achieving the independence, when we tried to get in the international conferences, we were not understood and our papers were not accepted, because we issued from our concepts. The situation changed only then when we comprehended that and adjusted ourselves. The correspondence between the notions was explained in (Kuusik & Lind, Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques, 2008). By reference to all the foregoing, the methods and algorithms presented in this work are not much influenced by the world's top of the area, our ideas have been evolved rather amid our own school.

I joined the development of DA and HG after defending my master thesis "Monotone systems in data mining" in 1998 and in some years the research in this area developed into one of my main activities.

Looking into these methods and talking to their potential users several usage problems and weak points that needed to develop came out. Also the new ideas how to further develop the methods arose.

1.2 Methods and theories

In this subchapter we will introduce the methods that will be developed in the thesis: determinacy analysis (1.2.1) and generator of hypotheses (1.2.2) as well as theory of monotone systems (1.2.3) on which these developments are based.

1.2.1 Determinacy Analysis

According to Chesnokov (1982, pp. 7-10), Determinacy[1] Analysis (DA) has been created to fill the gap in the analysis of the qualitative sociological and socio-economic data. The tasks (of analysis of sociological and socio-economic data) divide into two groups: studying the structure of relationships between variables

[1] Russian „Детерминационный анализ“ has been translated both as "Determinacy Analysis" and "Determination Analysis". We will use „Determinacy Analysis“ because it is used this way in the papers written in English by the author of the theory Chesnokov, for example: "Determinacy analysis and theoretical orthography" (by S. V. Chesnokov and P. A. Luelsdorff, in: *Theoretical linguistics*, 1991, *17*(1-3), 231-262)

and building secondary aggregated indicators. The second type of tasks is well covered by different coefficients and other measures (e.g. Yule's, Kendall's, Pearson's, Chuprov's, Cramér's coefficients, entropy). While such integral indicators provide a possibility to get the estimations of a general nature, they however, carry away from the characteristics presented by the values of the qualitative attributes in the working documents of empirical studies.

In practice, the summary of direct visual analysis of contingency tables serves as a criterion of truth of the results, obtained by such integral indicators.

Direct examination of percentage breakdown in contingency tables forms the basis for so called content analysis[2] of empirical data. It is feasible when the tables have up to three dimensions. Having more dimensions the considerable difficulties appear: presentation of the material in a visually graspable form becomes complicated, the big number of empty or almost empty cells in tables appears, it becomes complicated to observe the trends and regularities of interest. The techniques for local, fragmentary analysis of contingency tables that could be contribute to overcoming these difficulties, practically have not been discussed in the research literature. DA represents a variant of just such kind of technique.

The task solved by DA is not original. The task is to search for and describe the situations, where by the specific values of some social indicators it was possible to predict sufficiently definitely the values of other indicators. The usual conditional frequency serves as a main characteristic of the ability to predict. If *b* is predicted based on *a* according to the rule "if *a* then *b*" then the accuracy is measured by the value of the conditional frequency P(*b*/*a*). The rule "if *a* then *b*" itself is denoted by the symbol $a \to b$ and is called a determination.

DA enables to get sufficiently accurate and complete determinations, manipulating combinations of elementary characteristics. Differentiation of different characteristics by the degree of essentiality of their contribution to the arguments of determination (measured by the increment of conditional frequencies) is included into this task. Active building of the contexts in which the determinations are examined as well as inclusion of a priori and a posteriori typologies, aggregated indicators, indexes into the analytic process are important elements of DA.

The idea of analysing the conditional frequencies is not new. DA differs from other directions in this area primarily by consistent orientation to a direct manipulation with direct and inverse conditional frequencies and their increments ("contributions").

DA has a similarity with the fuzzy set theory (Zadeh, 1965). In DA we say "there is a determination $a \to b$ with intensity P(*b*/*a*)". Its analogue in the fuzzy sets is

[2] Russian "содержательный анализ"

“there is an element *a*, which belonging to a fuzzy set *b* is characterized by the value of the membership function $\mu_b(a)$=P(*b*/*a*)”.

DA is a systematic exploration method of conditional probabilities or simply percentages contained in usual statistical contingency tables of different attributes. For practical use it has to be supported by software.

DA is an interactive method. It is realised using a dialogue[3]. The user selects the attributes for inclusion into the analysis. In order to combine interesting attributes the user has to have an imagination of the material under examination. Trying every possible combination would take too much time when there are lot of attributes[4]. Thus DA is not the best means for discovering new regularities, but for testing the researcher’s hypotheses. (Kuusik, 1988)

DA is familiar in Russia. It has been used in sociology (Chesnokov, 1980b), linguistics (Luelsdorff & Chesnokov, 1996), medicine (Chesnokov, 1996), and other areas.

1.2.2 Generator of Hypotheses

Generator of Hypotheses is inspired by the JSM[5]-Method for Automatic Generation of Hypotheses (Anshakov, Skvortsov, Finn, & Ivashko, 1987), thence its name comes from. In contrast to DA, both GH and JSM-method enable to automate the process of finding regularities (hypotheses). These methods allow outputting all value combinations that really exist in given data. (Kuusik, 1988) Slow algorithm that had been used to realise the JSM-method (see (Kuusik, 1987)) gave rise to creation of GH.

While DA and JSM-method assume that the given attributes are divided into the causes and effects, GH does not expect any partition of attributes.

The result of GH – value combinations – is presented as a tree, showing the order of finding the combinations. Each embranchment is labelled with the probability P(*b*/*a*). Each branch (or shorter fragment of it) describes a group of data objects and is usable as a work hypothesis for a researcher.

GH uses a Monotone System based algorithm MONSA that was presented in (Kuusik, 1993).

---

[3] Original applications of DA will be described in 3.2.1.

[4] The number of all possible combinations is v^a-1, where a is the number of attributes and v is the number of different values of (each) attribute.

[5] JSM comes from the name of English philosopher John Stuart Mill; the Russian acronym is ДСМ.

1.2.3 Theory of Monotone Systems

Algorithms presented in this thesis are based on the theory of monotone systems that has been created at Tallinn Technical University.

Monotone (or monotonic[6]) systems (MS) were introduced by J. E. Mullat (1971), (1976), (1977). As the name refers these systems wield the monotonicity property which means that a function is either entirely nonincreasing or nondecreasing. "By a monotonic system, we understand a system, for which an action realized on an arbitrary element α involves either only decrease or only increase in the significance levels of all other elements." (Mullat, 1976)

Several methods based on the theory of monotone systems have been developed in Department of Informatics at Tallinn University of Technology. An overview of scale of conformity, data table reordering techniques (minus technique, plus technique, mixed technique), Mullat-Võhandu kernel extracting algorithm, generator of hypotheses and best decision finding is given in (Võhandu, Kuusik, Torim, Aab, & Lind, 2006). The MS approach has been applied in the graph theory to extract all maximal cliques (Kuusik, 1995) and maximum clique (Kuusik, Lind, & Võhandu, 2004).

The current state of MS is kept by J. Mullat at http://www.datalaundering.com/.

1.3 Research aims

When I started the method called GH and a suitable algorithm (MONSA) already existed. For DA the so called step-by-step approach had been created here. GH and DA find different things and use different algorithms (however, both are based on monotone systems). We have dealt with them separately until they got together only in our last development – zero-factor-free DA which uses GH-type algorithm (instead of the type used for other DA algorithms).

Besides further developing these methods it was important to publish already existing algorithms and methods in English because the publications in Russian and Estonian were not available and understandable to the remaining world. Thus, some of the papers (listed in 1.5) contain such "old" material. Also, showing relations between the concepts used here (for GH and DA) and the ones

[6] At the creation time of the theory these systems were called "monotonic" in English, at a later time "monotone". The reason is that this name "Monotone System" was already occupied in "Reliability Theory" unknown to J. Mullat (1976). According to Wolfram MathWorld (2016) "Monotone Increasing (Decreasing)" means strictly increasing (decreasing) i.e. never remaining constant (Weisstein, 2016a), (Weisstein, 2016b) while "Monotonic Function" is entirely nonincreasing or nondecreasing i.e. includes possibility to remain constant (Stover, 2016). To make a difference we can say "**strictly** monotone/monotonic" or "**weakly** monotone/monotonic".

used in closed set mining and association rule mining was an important outcome in order to make our work understandable.

The overall goal of this thesis is to improve and develop a data analysis method called Determinacy Analysis.

The main tasks:

- To investigate the possibilities to create and develop new approaches based on GH and MONSA
- To create a new algorithmics (based on MS) for DA in order to add new functionalities
- To show the correspondences of our concepts of GH/MS and the concepts used in DA with more widely used concepts in DM and ML

The more specific questions (under the main tasks) were not posed all at a time, but arose one after another when we discovered new issues that needed solving. In the next section we will give an overview of what we have done.

1.4 Overview of developments

In the beginning, DA and HG were developed separately.

When I started, for DA the so called step-by-step approach had been created here. Compared to the original application (called DAS) it allows to find rules with different lengths, thus reducing redundancy that appears in the form of inessential (zero) factors (in the antecedents of rules), but not eliminating it. Both approaches find additive systems of rules (i.e. sets of non-intersecting rules[7]).

Step-by-step approach involved a new thing to consider – the order of attributes (factors) in the rules. Attributes are added into the rules in a given order, the completion of a single rule is stopped whenever it occurs to be accurate. In case of different orders the results are different. The number of all possible orders is too big to try them all and then find the most suitable one. Thus we got a new problem to solve – to find the best (or an optimal) order for attributes in the rules.

This order is essential during the work process in order to avoid or lessen the redundancies, in the final rule it is not important any more. We explored the possibility to decide by factor's contribution to accuracy at the moment when it is added into the rule. We reached the conclusion that we cannot say whether the factor remains essential (positive) (regarding the factors that will be added later) or not. This conclusion is valid for the step-by-step approach as well as the approaches that produce non-additive systems of rules.

As an additive system of (non-intersecting) rules generally cannot be free of zero factors, we started to develop algorithms for finding non-additive systems of rules

[7] If the rules do not intersect then there is maximally one rule for each data object.

(where the rules can intersect). The order of attributes can be different in each rule.

First we worked out an algorithm that finds a possibly small set of rules, monitoring and taking into consideration which objects are covered by the found rules already. This MS-based algorithm is not based on MONSA, it uses so called 3D[8] frequency tables. Like a step-by-step approach this one also produces one system of rules that is not always the best one.

Instead of finding one system of rules, it was good to find all non-redundant rules and then combine different covers (rule sets) from them. For that purpose we created an algorithm that produces all non-redundant rules and some redundant rules[9] (that are eliminated by compression). The rules can intersect, regardless of how many times the objects are covered. Redundancy is avoided as much as possible by the nature of the algorithm. Differently from the previous 3D-algorithm this one does not track the coverage of objects, and it uses elimination technique “bringing zeroes down” from MONSA. After elimination of redundant rules (by compression) we get the set of non-redundant rules, called Determinative Set of Rules (DSR) that gives a source for post-processing the rules.

Dealing with redundancy we have found two different types of zero factors: 1) the ones that can be just left out from the left side of the rule (without changing the set of covered objects); 2) the ones that produce a subrule of an existing rule (reducing the set of covered objects).

Meanwhile we have had developed GH and MONSA. My supervisor had a true guess that intersections found by algorithm MONSA are closed sets. After showing this correspondence (Kuusik & Lind, 2008) it became possible to relate MONSA’s results with the widely known concepts of frequent itemset mining and association rule mining (ARM).

This correspondence helped to answer the question “can the leading element be chosen by some other principle (than by the maximal frequency)?” and explain the differences caused by different selection criteria (maximal frequency vs minimal frequency).

After relating the concepts of DA with the ones of ARM we reached a conclusion that (in order to be free of zero factors) the left side of the rule has to be a minimal generator, such that it has no subset that defines a class.

[8] 3D frequency table contains “usual” frequency table (as in MONSA) and additionally a frequency table for such objects that belong to the “target class”.
[9] The rules that are contained in some other rules are considered redundant.

Proceeding from those correspondences we found a solution for one more question – “how to find both (minimal) generators and closed sets?” – and created an algorithm for finding them all.

First we tried to adapt the algorithm that uses 3D frequency tables, but unsuccessfully: we did not get all the closed sets (CSs) and generators characteristic to the target class (the non-accurate descriptions were not found).

Next we tried to use MONSA as a basis of a new algorithm and this time we got an expected outcome. While the original MONSA moves from a CS to a CS, this one moves from a generator to a generator, because it is easier to find a CS for a generator than unknown number of generators for a CS. Actually, we first had a more sophisticated algorithm (MONSA) and then made it easier, thereat retaining a valuable mechanism for detecting the elements between a CS and generator, but treating them differently.

As a CS with all its minimal generators forms an equivalence class (EC), it was straightforward to create an algorithm for finding all ECs (although previously we did not have such goal).

The reason for finding both generators and CSs was that together they give a basis for creating (positive) association rules (ARs): the elements between a CS and its minimal generator (we call them “zero factors” as they correspond to one type of zero factors in DA) can be seen as a conclusion resultant from generator. This way we get a (positive) “zero-factor-free” association rule. Besides positive conclusions the negative ones (called excluded factors) can be of interest as well. MONSA offers an easy way to detect them, thus we get negative ARs.

The way how to detect zero factors gave an answer to the question “how to integrate classes into MONSA?” that had been waited for a solution for a long time. A class can be detected the same way as any other zero factor, the difference is that class attributes cannot occur in the left side of the rules. Having this solution we got ready to create a MONSA-based algorithm for DA.

Putting together the abilities to find all three types of the rules we reached our final development called zero-factor-free (ZFF) DA that produces three kinds of rules: 1) classification rules; 2) (positive) ARs; 3) negative ARs. This approach uses a MONSA-based algorithm (moving from a generator to a generator) and DSR-compression. Compared to the 3D algorithm that (together with a compression) gives a DSR for a target class, ZFF DA gives DSR for all classes (if desired) and enriches the (classification) rules with information about accompanying factors (positive AR) and excluded factors (negative AR).

Finally we have gathered different tasks solvable by GH and DA under the umbrella called Universal Generator of Hypotheses that is one possible framework for that purpose.

1.5 Previously published work

The work of this thesis is based on the following publications:

A Kuusik, R. and Lind, G. (2008). Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques. *WSEAS Transactions on Information Science and Applications, 5*(5), 599-611.

B Lind, G. and Kuusik, R. (2007). Some Ideas for Determinacy Analysis Realisation. *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing. Palma de Mallorca, Spain, August 29-31, 2007* (pp. 185-190). ACTA Press.

C Lind, G. and Kuusik, R. (2008). Some Problems in Determinacy Analysis Approaches Development. *Proceedings of the 2008 International Conference on Data Mining (DMIN 2008), Las Vegas, Nevada, USA, July 14-17, 2008, Volume I,* pp. 102-108. CSREA Press.

D Kuusik, R. and Lind, G. (2010). Some Developments of Determinacy Analysis. *Advanced Data Mining and Applications: The 6th International Conference on Advanced Data Mining and Applications (ADMA2010), Chongqing, China, November 19-21, 2010. LNAI 6440*, pp. 593-602. Berlin Heidelberg: Springer-Verlag.

E Kuusik, R. and Lind, G. (2011). New Developments of Determinacy Analysis. *Advanced Data Mining and Applications - 7th International Conference: ADMA 2011, Beijing, China, December 17-19, 2011. II; LNCS 7121*, p. 223−236. Springer.

F Lind, G. and Kuusik, R. (2016). Algorithm for Finding Zero Factor Free Rules. *Man-Machine Interactions 4: 4th International Conference on Man-Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6-9, 2015. AISC 391*, pp. 421-435. Springer.

These publications are reprinted in the appendices of the thesis.

All these papers are written together with my supervisor prof. Rein Kuusik. Generally, he set the goal and I dealt with the details. In case of all papers I have done most of the writing and preparing examples. Other contributions of mine are listed below.

Paper A: showing the correspondences between concepts used in our algorithm MONSA and the ones used for closed sets; explaining used pruning techniques.

Paper B: showing the correspondence with terminology used in association rule mining; experimenting with DAS (original application of DA), analysing its possibilities and deficiencies; finding how many frequencies is needed to compute the characteristics of the rules.

Paper C: exploring (and translating) the newer theory of DA (Chesnokov, 2002); investigating and explaining the problem with zero factors (normal rules).

Papers D and E: experimenting with the algorithms proposed by my supervisor.

Paper F: defining two types of zero factors, showing their relation with minimal generators; creating the ZFF DA algorithm.

Besides the published material this thesis contains unpublished material as well.

1.6 Contribution of the thesis

The main results:

- We have shown how the concepts used in MONSA correspond to the ones used in CS mining and how the concepts of DA are related to the ones of ARM and ML
- We have elaborated an algorithm for finding all equivalence classes
- We have defined two types of zero factors (in DA)
- We have found a way to detect classes and other zero-zero factors (and that they are detected the same way) for MONSA
- We have proposed ZFF DA that finds three types of non-redundant rules at the same time: classification rules, positive and negative association rules
- We have presented UGH that gathers different tasks solvable by DA and GH

1.7 Organisation of the thesis

Chapter 2 gives an overview of theoretical foundations of this work: data mining, machine learning, theory of monotone systems and determinacy analysis.

Chapter 3 presents our work. Developments of GH/MONSA and developments of DA are given in distinct subchapters, 3.1 and 3.2, accordingly, as they were developed separately until they got together in ZFF DA. ZFF DA is presented at the end of 3.2. Section 3.3 describes UGH and shows how it is covered by our algorithms. In 3.4 further algorithmic developments of ZFF DA are proposed.

In Chapter 4 the work is concluded and the directions for further research are listed.

2 THEORETICAL BACKGROUND

In this chapter we will give an overview of data mining (2.1), machine learning (2.2) and some methods used in these areas (Classification in 2.3, Association rule mining in 2.4 and hybrid approaches in 2.5) as well as the theory of monotone systems (2.6) and determinacy analysis (2.7).

2.1 Data mining

This subchapter is based on (Fayyad, Piatetsky-Shapiro, & Smyth, 1996) and (Dunham, 2002a).

Data mining (DM) is often defined as finding hidden information in a database. According to (Fayyad, Piatetsky-Shapiro, & Smyth, 1996) DM is seen as one step of process called knowledge discovery in databases.

Knowledge discovery in databases (KDD) is the process of finding useful information and patterns in data.

Data is a set of facts.

Pattern is an expression describing facts in subset of all facts, it has to be simpler than the enumeration of all facts in it.

Knowledge is an interesting pattern. What is interesting, is determined by the user.

KDD process is usually a multi-step process, which involves data preparation, search for patterns, knowledge evaluation, and refinement involving iteration after modification. The process is assumed to be non-trivial – that is, to have some degree of search autonomy.

Data mining is the use of algorithms to extract the information and patterns derived by the KDD process.

The two "high-level" primary goals of data mining are *prediction* and *description*. Predictive model is used to predict unknown or future values of variables of interest using known data. Description focuses on finding human-interpretable patterns describing the data. A descriptive model serves as a way to explore the properties of the data examined, not to predict new properties (like the predictive model). In the context of KDD, description tends to be more important than prediction. This is in contrast to pattern recognition and machine learning applications where prediction is often the primary goal.

The goals of prediction and description are achieved by using different DM tasks. The most important of them are:

- Classification
- Clustering
- Association rules

Classification maps (classifies) data into predefined groups or *classes*. It is often referred to as supervised learning because the classes are determined before examining the data.

Clustering is similar to classification except that the groups (clusters) are not predefined, but rather defined by the data alone. It can be thought of as partitioning or segmenting the data into groups that might or might not be disjointed. Clustering is alternatively referred to as unsupervised learning or segmentation.

Determining association rules is the best example of the data mining task of uncovering relationships among data called *link analysis*, or *affinity analysis.* An *association rule* is a model that identifies specific types of data associations.

Classification is used for prediction purposes, while clustering and association rules are considered to be descriptive.

Most data mining methods are based on the concepts from machine learning, pattern recognition and statistics: classification, clustering, graphical models, and so forth.

2.2 Machine Learning

The following is mainly based on (Michalski, Carbonell, & Mitchell, 1984) and (Dunham, 2002a).

Machine learning (ML) is the area of artificial intelligence (AI) that examines how to write programs that can learn. In data mining, machine learning is often used for prediction[10] or classification. The objectives of ML and DM are different. Machine learning looks at things that may be difficult for humans to do or concentrates on how to develop learning techniques that can mimic human behaviour. The objective for data mining is to uncover information that can be used to provide information to humans (not take their place).

Most of the learning strategies involve some amount of inductive inference. *Inductive inference* is a (bottom-up) mode of reasoning that starts with specific facts and concludes general hypotheses or theories. The conclusion is not guaranteed to be true, but usually it is. Reasoning in the opposite direction (top-down – from general to specific), *deductive inference* is the derivation of a logical consequence from a given set of premises, it is a truth-preserving transformation of assertions. In AI deductive reasoning is used, for example, in expert systems, automated theorem proving.

Inductive Learning (i.e. learning by inductive inference) is learning by generalizing facts and observations obtained from a teacher or environment. The

[10] Here *prediction* is one of DM tasks (used for prediction purposes). It is often viewed as forecasting a continuous value, while classification forecasts a discrete value.

major forms of inductive learning are learning from examples and learning from observation and discovery. Learning from examples is inferring a general concept description from examples and (optionally) counter-examples of that concept. Learning from observation is constructing descriptions, hypotheses or theories about given collection of facts or observations whereat there is no a priori classification of observations into sets exemplifying desired concepts. The former represents supervised learning and the latter – unsupervised learning.

In case of learning from examples (supervised learning), the set of training data together with correct answers is given. The computational model is trained to give the correct answer to each entry in the training set. There can be also a test set of data that is used to evaluate the result of learning. It is assumed that such model is good enough to be applied to unseen data. Such assumption is called the inductive learning hypothesis (Mitchell, 1997).

Often only one concept has to be learned. In such case the given training examples can be either positive – the representatives of target concept – or negative – the ones that do not belong to that concept. Usually both are given, in some cases only positive examples are available. In case of multiple-concept learning different classes can be either mutually disjoint or overlapping. Looking for one certain class, the representatives of all other classes are considered as negative examples.

Classification is a typical supervised learning task.

In case of *unsupervised* learning the data exist, but without answers. In this case, usually, the problem is to partition the training set into subsets in some appropriate way. A typical unsupervised task is clustering – identifying a finite set of categories or clusters to describe the data. The clusters can be mutually exclusive and exhaustive, or consist of a richer representation such as hierarchical or overlapping categories (Fayyad, Piatetsky-Shapiro, & Smyth, 1996).

Learning from observation and discovery is a very general form of inductive learning that includes discovery systems, theory-formation tasks, the creation of classification criteria to form taxonomic hierarchies, and similar tasks without benefit of an external teacher.

2.2.1 Definitions of learning from examples

Giving the basic concepts of learning from examples (supervised learning), we mainly originate from the notions of (Gams & Lavrac, 1987).

The purpose of learning from examples is to find a concept description.

A *(concept) description* is a set of (classification) rules:

$$Desc = \{Rule_j\}, j=1,2,\ldots,S .$$

A *rule* is an implication where a condition part is a complex and a conclusion part is a class name:

$$Rule_j = \text{“}Complex_j \Rightarrow Class\text{”}$$

or

$$Rule_j = \text{“if } Complex_j \text{ then } Class\text{”}$$

or

$$Rule_j = (Complex_j\ , Class).$$

A *complex* is a conjunction of selectors:

$$Complex_j = \text{“}(\&\ Selector_k)\text{”},\ k=1,\ldots K,\ 1 \leq K \leq M,$$

where M is the number of attributes.

A *selector* relates an attribute to its disjunctive set of values:

$$Selector_{jk} = \text{“}(Attribute = Values)\text{”}$$

where

$$Values = \text{“}(\vee\ Value_t)\text{”}.$$

Classes are non-intersecting subsets of objects.

A *(learning) example* is a complex with all attributes, each with exactly one value, and its class value.

A *description covers an example* of a *Class* if it contains a rule in the form “if *Complex* then *Class*”, and *Complex* covers the example.

A *complex covers an example* if each *Selector* covers the example.

A *selector* of the form “*Attribute = Values*” *covers an example* with “*Attribute = Value*” if *Value* is in *Values*.

A description is *complete* if it covers all examples of all classes.

A description is *consistent* if it does not cover any pair of examples from different classes.

The learning algorithms have to allow us to find descriptions that are at the same time both consistent and complete.

2.3 Classification

The subchapter is mainly based on (Dunham, 2002a) and (Dunham, 2002b).

Given a database $D=\{t_1,t_2,\ldots,t_n\}$ of tuples (items, records) and a set of classes $C=\{C_1,\ldots,C_m\}$, the *classification problem* is to define a mapping $f : D \rightarrow C$ where each t_i is assigned to one class. A class, Cj, contains precisely those tuples mapped to it.

The classes are predefined, non-overlapping, and partition the entire database.

Usual approach is:

1. Create specific model by evaluating training data (or using domain experts' knowledge).
2. Apply this model to new data.

Classification techniques include: regression, distance, decision trees, rules, neural networks.

Regression is used to map data item to a real valued prediction variable. Regression assumes that the target data fit into some known type of function (e.g., linear) and then determines the best function of this type that models the given data. It can be used either to divide area into regions or to predict a class membership function (input includes desired class).

In case of *distance-based methods* the items are placed in class to which they are "closest". "Alikeness" of different items can be identified by similarity or distance measures (like Euclidean distance, Manhattan distance, Jaccard's coefficient etc.).

(Artificial) neural networks is a graph-based method for learning from examples. Training this network can take a lot of time and the learned target function is not (easily) comprehensible, but it is robust to errors in the training data. Thus it suits well for learning to interpret noisy, complex real-world sensor data (such as inputs from cameras and microphones). (Mitchell, 1997)

The *decision tree* approach to classification is to divide the search space into rectangular regions. A tuple is classified based on the region into which it falls.

Classification rules represent found knowledge in the form of IF-THEN rules. In contrast to decision trees, they can overlap.

Decision trees and rules will be introduced in subsequent subchapters.

2.3.1 Decision trees

The following is based on (Dunham, 2002a).

A decision tree is a predictive modelling technique used in classification, clustering, and prediction tasks.

A decision tree (DT) or a classification tree is a tree where the root and each internal node is labelled with a question. The arcs emanating from each node represent each possible answer to the associated question. Each leaf node represents a prediction of a solution to the problem under consideration.

Decision trees use *divide-and-conquer* technique to split the problem search space into subsets.

The recursive algorithm builds a tree in a top-down fashion by examining the initial training data. Algorithms differ in how they determine the “best attribute” and its “best predicates” to use for splitting. Also, the “stopping criteria” for terminating the creation of tree can be different.

The simplest approach is to stop when all objects/records belong to the same class. In order to prevent overfitting or just creating a larger tree it would be desirable to stop earlier. Sometimes, on the contrary, more levels than needed would be created (when there are data distributions not represented in the training data).

To improve the performance applying the tree for classification, a balanced tree with the fewest levels is desirable. It has been found even that, in order to be meaningful to the user, an internal node (i.e. attribute test) can be the parent of at most one internal node (in so called fully linear trees) (Michie, Spiegelhalter, & Taylor, 1994). Some algorithms create only binary tree. Such trees are easily created, but they tend to be deeper.

DTs are easy to interpret and understand. They scale well for large databases because the tree size is independent of the database size. Trees can be constructed for data with many attributes. Disadvantages are: it is not easy to handle continuous data and missing data, correlations among attributes are ignored, not all classification problems suit for rectangular partitioning.

2.3.2 Classification rules

The following is based on (Dunham, 2002a), (Mitchell, 1997) and (Michie, Spiegelhalter, & Taylor, 1994).

One way to perform classification is to generate IF-THEN rules that cover all cases. The rule consists of two parts: antecedent (IF-part) and consequent (THEN-part). The antecedent contains a predicate that can be evaluated as true or false against each tuple in the data.

There are algorithms that generate rules from decision trees or neural networks as well as algorithms that start from scratch.

A DT can always be used to generate rules, for each leaf node a rule is generated. Rules with the same consequent could be combined together by ORing the antecedents of the simpler rules.

There are following differences between rule and trees:

- The tree has an implied order in which the splitting is performed. Rules have no order.

- A tree is created based on looking at all classes. When generating rules, only one class must be examined at a time.
- A DT is restricted to non-overlapping rules while IF-THEN rules can overlap.

The algorithms that attempt to generate rules that exactly cover a specific class are called *covering* (or *separate-and-conquer*) algorithms. Usually the best attribute-value pair is chosen, as opposed to the best attribute with tree-based algorithms.

One of the most widespread approaches to learning disjunctive sets of rules is *sequential covering* algorithm. It reduces the problem of learning a disjunctive set of rules to a sequence of simpler problems, each requiring that a single conjunctive rule be learned. The strategy it uses is: learn one rule, remove data it covers, then iterate this process on the remaining examples until no examples remain. Learned rules have to have high accuracy, but not necessarily high coverage. High accuracy is needed to make correct predictions. Low coverage means that the rule does not make predictions for every training example. Together the rules cover the full set of positive examples. The final set of rules can be sorted so that more accurate rules will be considered first when a new instance must be classified.

One way to effectively find one rule is to consider all possible branches as in case of DTs, but to follow only the most promising of them. With such a greedy search there is a danger to make a suboptimal choice at any step. To reduce this risk, a *beam search* can be used: instead of single best candidate k best candidates are maintained at each step. On each search step, descendants (specializations) are generated for each of these k candidates, and the resulting set is again reduced to the k most promising members. Beam search keeps track of the most promising alternatives to the current top-rated hypothesis, so that all of their successors can be considered at each search step.

While sequential covering algorithm (such as CN2 (Clark & Niblett, 1987)) learns one rule at a time, DT algorithms (such as ID3 (Quinlan, 1986)) learn the entire set of disjuncts simultaneously and might be called *simultaneous covering* algorithms. At each search step a DT algorithm chooses among alternative attributes by comparing the partitions of the data they generate. In contrast, sequential covering algorithm chooses among alternative attribute-value pairs, by comparing the subsets of data they cover. Thus, sequential covering algorithms make a larger number of independent choices than simultaneous covering algorithms. The first case may be preferred in case of plenty data, the latter – when data is scarce.

Besides above described general-to-specific search, also specific-to-general search is used for finding rules (for example, in AQ (Michalski, 1969)). In such case, maximally specific rule (that specifies a value for every attribute in use) is used as a “seed”. Then specificity is relaxed by dropping attributes one at a time for all the ways of dropping a single attribute, followed by all the ways of dropping two attributes, three attributes etc. Any rule which includes in its cover a “negative example”, is *incorrect* and is discarded during the process. The cycle terminates by saving a set of shortest rules covering only desired class. As a classifier, such a set is guaranteed *correct*, but cannot be guaranteed *complete*.

2.4 Association rule mining

The problem of mining association rules between sets of items in a large database of customer transactions was introduced by Agrawal, Imieliński and Swami (1993). These rules describe association between sets of items. “An example of such an association rule is the statement that 90% of transactions that purchase bread and butter also purchase milk. The antecedent of this rule consists of bread and butter and the consequent consists of milk alone. The number 90% is the confidence factor of the rule.” (*Ibid*., p.207)

A confidence is a measure of the rule's strength. Additionally a rule’s statistical significance is measured by a (transactional) support that shows which part of transactions satisfy the rule. The task is to generate all rules that have at least given support and given confidence.

“Besides statistical significance, another motivation for support constraints comes from the fact that we are usually interested only in rules with support above some minimum threshold for business reasons. If the support is not large enough, it means that the rule is not worth consideration or that it is simply less preferred /…/.” (*Ibid*., p.208)

Additionally syntactic constraints might be specified. “These constraints involve restrictions on items that can appear in a rule”: certain consequent[11], certain items in the antecedent, both, top *k* rules that have certain item in the consequent, etc.

Association rules are not classification rules, there are no pre-specified classes. Each item can appear either in the antecedent or in the consequent.

Agrawal, Imieliński and Swami (1993) decompose the problem of rule mining into two steps:

1. Generate *large* itemsets – all combinations of items that are present in at least *s*% of transactions.
2. Generate from each large itemset the rules that use items from the large itemset.

Further large itemsets are called usually frequent itemsets.

[11] In this work the consequent consists of one item

2.4.1 Frequent itemsets

The following definitions are presented according to Zaki and Hsiao (1999), (2002).

Typically the database is arranged as a set of transactions, where each transaction contains a set of items. Let $I = \{1,2,...,m\}$ be a set of items, and let $T = \{1,2,...,n\}$ be a set of transaction identifiers or *tids*.

A set $X \subseteq I$ is also called an *itemset*, and a set $Y \subseteq T$ is called *tidset*.

$t(X)$ denotes a tidset that corresponds to an itemset X, i.e. the set of all tids that contain X as a subset: $t(X) = \cap_{x \in X} t(x)$.

$i(Y)$ denotes an itemset that corresponds to a tidset Y, i.e. the set of items common to all the tids in Y: $i(Y) = \cap_{y \in Y} i(y)$.

The *support* of an itemset X, denoted $\sigma(X)$, is the number of transactions in which it occurs as a subset, i.e. $\sigma(X) = |t(X)|$.

An itemset is *frequent* if its support is more than or equal to a user-specified *minimum support* (*minsup*) value, i.e. if $\sigma(X) \geq minsup$.

A 'support' can be called a 'frequency' as well. Sometimes 'support' is used for an absolute number of transactions (covered by an itemset) and 'frequency' for a percentage from the number of all transactions in the database (for example, Mielikäinen (2006) defines them this way).

The support/frequency measure is *anti-monotone*: $X_1 \subseteq X_2 \Rightarrow \sigma(X_1) \geq \sigma(X_2)$ and therefore the set of frequent itemsets $\mathcal{F}$ is *downward closed* (Mielikäinen, 2006): if $X_1 \in \mathcal{F}$, then $X_2 \in \mathcal{F}$ for all $X_2 \subseteq X_1$ i.e. all subsets of a frequent itemset are frequent. Also, all supersets of an infrequent itemset are infrequent: if $X_1 \notin \mathcal{F}$, then $X_2 \notin \mathcal{F}$ for all $X_2 \supseteq X_1$. (Pasquier, Bastide, Taouil, & Lakhal, 1999, p. 402). This is known also as an *Apriori* principle.

Zaki and Hsiao (1999) have shown that instead of finding all frequent itemsets it is enough to mine all frequent closed itemsets.

2.4.1.1 Closed sets, generators and equivalence classes

The definitions concerning closed set will be given by (Zaki & Hsiao, 2002) and the definitions for generator by (Zaki, 2004).

A frequent itemset X is called *closed* if there exists no proper superset $Y \supset X$ with $\sigma(X) = \sigma(Y)$.

Closed sets are found using closure operation.

A *closure* of an itemset X, denoted $c(X)$, is defined as the smallest closed set that contains X. An itemset X is closed if and only if $X = c(X)$.

The closure of an itemset X is found as: $c(X) = i(t(X))$.

The support of an itemset X is also equal to the support of its closure, i.e. $\sigma(X) = \sigma(c(X))$.

Itemsets with the same closure (and equal support) form an equivalence class. *Equivalence class* is defined as a set of itemsets that always occur together in the same set of transactions (Bastide, Taouil, Pasquier, Stumme, & Lakhal, 2000).

Closed set is the only maximal set of an equivalence class. An equivalence class may contain more than one minimal itemset. For example: if all transactions contain items A and B, the equivalence class contains itemsets A, B and AB. AB is the maximal (closed) set. Set A and set B are both minimal sets of that equivalence class.

A minimal itemset in an equivalence class is called a (minimal) generator (of closed set) or a free set or a key pattern.

An equivalence class can be uniquely determined and concisely represented by a closed set and a set of minimal generators. Equivalence classes do not overlap. (Li, Liu, & Wong, 2007)

An itemset X' is a *generator* of closed itemset X if and only if (1) $X' \subseteq X$, and (2) $\sigma(X') = \sigma(X)$.

X' is called a *proper* generator if $X' \subset X$ (i.e., $X' \neq X$). If there is no proper generator, X is its own minimal generator.

A generator X' is a *minimal* generator if it has no proper subset $Y \subset X'$ with $\sigma(Y) = \sigma(X')$.

Usually under the 'generator' the 'minimal generator' is meant.

Although the definitions are usually given for frequent (i.e. with support ≥ threshold) closed and free itemsets (=minimal generators), the closedness or freeness of an itemset does not depend on the fact whether it is frequent or infrequent.

The frequency threshold is used to separate a (frequent) part from all closed or free itemsets. The reason is that the number of all closed or free itemsets is usually large.

Besides using support/frequency threshold it is advisable to extract and store only the critical part of all (frequent) itemsets, this part is called a condensed representation or a concise representation. Sometimes a distinction between these two notions is made as follows. A *condensed representation* is a particular subset of the frequent pattern collection, such that we can regenerate from this subset the whole collection (Bykowski & Rigotti, 2003). A *concise representation* has not to be a subset of all itemsets it represents, but preferably it has to be lossless. "By lossless we mean a representation that allows derivation and support determination of all frequent itemsets without accessing the database" (Kryszkiewicz & Gajek, 2002b).

The collection of closed sets (with frequencies) is enough to restore the frequencies of all existing itemsets. The frequency of a non-closed (frequent) itemset is obtained by taking the maximum of the frequencies of its closed supersets. If an itemset has no superset among frequent closed itemsets, then consequently that itemset is infrequent. (Mielikäinen, 2006)

Let us have a small sample data set given in Table 2.1. For example, the frequency of itemset AB is 4, in the following we denote it as AB(4). Let the support threshold be 3. Figure 2.1 shows equivalence classes (surrounded by dashed lines) with **closed sets (shown in bold)** and *(minimal) generators (shown in italic)* in case of frequency threshold equal to 3 for dataset from Table 2.1. There are no non-minimal proper generators.

Table 2.1 Sample data set

ABC
AB
ABD
ABCD
BCD

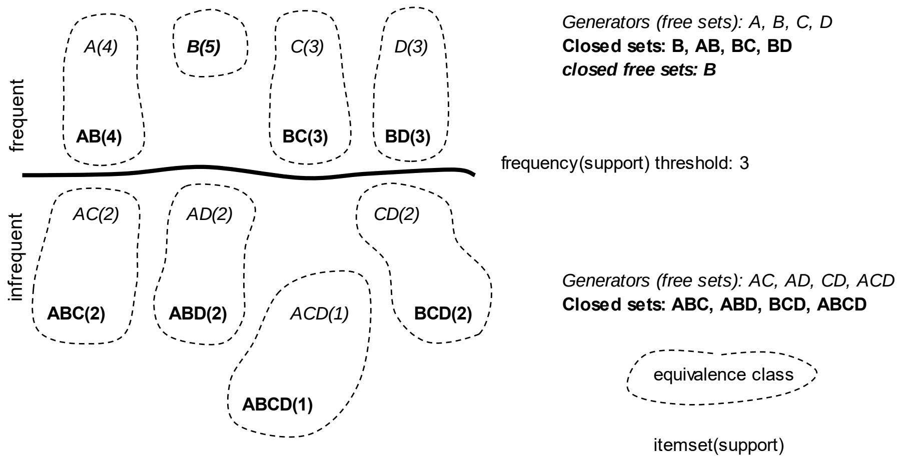
frequent
infrequent
A(4)
AB(4)
B(5)
C(3)
BC(3)
D(3)
BD(3)
AC(2)
ABC(2)
AD(2)
ABD(2)
ACD(1)
ABCD(1)
CD(2)
BCD(2)
Generators (free sets): A, B, C, D
Closed sets: B, AB, BC, BD
closed free sets: B
frequency(support) threshold: 3
Generators (free sets): AC, AD, CD, ACD
Closed sets: ABC, ABD, BCD, ABCD
equivalence class
itemset(support)

Figure 2.1 Equivalence classes, closed sets and generators (for data from Table 2.1)

In this little example the frequent closed itemsets are B(5), AB(4), BC(3) and DB(3) and infrequent closed itemsets are ABC(2), ABD(2), BCD(2) and ABCD(1).

For finding the frequency for itemset A, the maximum should be taken from the frequencies of itemsets AB, AC and AD. Among closed sets only AB is present. Farther closed supersets ABD, ABC and ABCD are also available, but all of them are supersets of AB and therefore are not considered (their frequencies are smaller than σ(AB) anyway). For that reason the frequency of A is the same as for AB (i.e. 4).

Similarly, the frequency of AD is equal to the frequency of ABD (=2). If we had only the frequent part of closed sets, then there was no superset for AD and consequently AD was just infrequent.

2.4.2 Association rules

The following definitions are given as in (Zaki, 2004).

An *association rule (AR)* is an expression $A \overset{q,p}{\longrightarrow} B$, where A and B are itemsets. Usually only rules with disjoint antecedent and consequent ($A \cap B = \varnothing$) are considered.

The *support* of the rule is $q=\sigma(A \cup B)= |t(A \cup B)|$ (i.e., the joint probability of a transaction containing both A and B). Alternatively, support is defined as a percentage of transactions that contain $A \cup B$ (Agrawal & Srikant, 1994).

The *confidence* of the rule is $p = \sigma(A \cup B) / \sigma(A) = |t(A \cup B)|/ |t(A)|$ (i.e., the conditional probability that a transaction contains B, given that it contains A).

A rule is frequent if the itemset $A \cup B$ is frequent (i.e. $q \geq minsup$).

A rule is *confident* if $p \geq minconf$, where *minconf* is a user-specified minimum threshold.

Association rules whose confidence is equal to 100% are called *exact* association rules and rules whose confidence is lower than 100% are called *approximate* association rules (Bastide, Pasquier, Taouil, Stumme, & Lakhal, 2000).

When support is understood, we omit q and write a rule as $A \overset{p}{\rightarrow} B$.

In order to get confident rules, the rules of the form $Y \overset{p}{\rightarrow} X - Y$, are generated for all frequent itemsets X, for all $Y \subset X$, and $Y \neq \varnothing$, and provided $p \geq minconf$. Since X is frequent, the rule is guaranteed to be frequent. For an itemset of size k there are $2^k - 2$ potentially confident rules that can be generated. This follows from the fact that we must consider each subset of the itemset as an antecedent, except for the empty and the full itemset. (Zaki, 2004)

2.4.2.1 Non-redundant association rules

According to (Zaki & Hsiao, 1999, p. 6) “it is sufficient to consider only the rules among closed frequent itemsets”. “A rule between any two itemsets is exactly the same as the rule between their respective closures“ (Zaki, 2004):

The rule $X_1 \xrightarrow{q,p} X_2$ *is equivalent to the rule* $c(X_1) \xrightarrow{q_1,p_1} c(X_2)$, *i.e.,* $q = q_1$ *and* $p = p_1$ (*Ibid*., p. 234).

Moreover, it is sufficient to consider only the rules among *adjacent*[12] closed frequent itemsets, other rules can be inferred by transitivity (*Ibid*., p. 235).

An association rule $X_1 \xrightarrow{p} X_2$ has confidence p=1.0 if and only if $c(X_2) \subseteq c(X_1)$ (or equivalently if and only if $t(X_1) \subseteq t(X_2)$) (*Ibid*., p. 235) i.e. all exact (100% confidence) rules are those that are directed from frequent closed itemsets to their closed subsets. Rules with $c(X_2) = c(X_1)$ are called *self-rules* and rules with $c(X_2) \subset c(X_1)$ *down-rules*.

Approximate association rules (p<100%) are directed from frequent closed itemsets to their closed supersets ($c(X_1) \subset c(X_2)$).

Zaki (2004) defines *non-redundant rules* as the most general ones among rules with equal support and confidence, i.e., “those having minimal antecedents and consequents, in terms of subset relation”. Formally:

Let R_i denote the rule $X_1^i \xrightarrow{q_i,p_i} X_2^i$. We say that a rule R_1 is more general than a rule R_2, denoted $R_1 \preccurlyeq R_2$ provided that R_2 can be generated by adding additional items to either the antecedent or consequent of R_1, i.e., if $X_1^1 \subseteq X_1^2$ and $X_2^1 \subseteq X_2^2$.

Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a set of rules, such that all their supports and confidences are equal, i.e., $q_i = q$ and $p_i = p$ for all $1 \leq i \leq n$. Then we say that a rule R_j is *redundant* if there exists some rule R_i , such that $R_i \preccurlyeq R_j$. Since all the rules in the collection $\mathcal{R}$ have the same support and confidence, the simplest rules in the collection should suffice to represent the whole set.

The set of all non-redundant rules constitutes a *generating set*, i.e., a rule set, from which other confident rules can be inferred (*Ibid*., pp. 237-239).

These rules are found between adjacent closed sets (more precisely: equivalence classes), but presented as rules between minimal generators of those closed sets (equivalence classes). This is because the purpose is to find possibly compact representation form. Actually, such generating set is not the minimal possible generating set, this question is opened as of 2004.

[12] i.e. there is no other closed set between them

Table 2.2 presents sample data set from (Zaki, 2004) and Table 2.3 lists frequent equivalence classes according to *minsup*=3 (**closed sets in bold**, *minimal generators in italic*).

Table 2.2 Sample data set (Zaki, 2004)

ACTW
CDW
ACTW
ACDW
ACDTW
CDT

Table 2.3 Equivalence classes with minsup=3 for data given in Table 2.2

Support	Equivalence classes
6	{***C***}
5	{*W*, **CW**}
4	{*A*, AC, AW, **ACW**}, {*D*, **CD**}, {*T*, **CT**}
3	{*AT*, *TW*, ACT, ATW, CTW, **ACTW**}, {*DW*, **CDW**}

The generating set for exact rules is: $\{TW\rightarrow A,\ A\rightarrow W,\ W\rightarrow C,\ T\rightarrow C,\ D\rightarrow C\}$ (*Ibid*., p. 237). The support of each rule is equal to the support of its antecedent.

While Zaki finds association rules with minimal antecedents and minimal consequents, Bastide et al (Bastide, Pasquier, Taouil, Stumme, & Lakhal, 2000) look for minimal non-redundant association rules with minimal antecedent and maximal consequent. They find that "from the point of view of the user, these rules are the most useful and the most relevant association rules"(*Ibid*., p.985).

An association rule $r : l_1\rightarrow l_2$ is a minimal non-redundant association rule iff there does not exist an association rule $r' : l_1'\rightarrow l_2'$ with $support(r)=support(r')$, $confidence(r)=confidence(r')$, $l_1'\subseteq l_1$ and $l_2\subseteq l_2'$.

In order to get a rule with minimal antecedent and maximal consequent, the antecedent is a minimal generator g and the consequent is its closure (i.e. closed set) diminished by that generator: $g\rightarrow c(g)\backslash g$.

The *exact* association rules, of the form $l_1 \Rightarrow (l_2\backslash\ l_1)$, are rules between two frequent itemsets l_1 and l_2, whose closures are identical: $c(l_1)=c(l_2)$. These rules are valid for all objects (of given set).

Each *approximate* association rule $l_1\rightarrow(l_2\backslash\ l_1)$, is a rule between two frequent itemsets l_1 and l_2 such that the closure of l_1 is a subset of the closure of l_2:

$c(l_1) \subset c(l_2)$. These rules are valid for a proportion of objects equal to their confidence.

For exact association rules a generic basis is defined and for approximate association rules an informative basis is defined. From these bases, their supports and their confidences (needed for approximate rules only), all valid exact and approximate (respectively) association rules, their supports and their confidences can be deduced.

Let *FC* be the set of frequent closed itemsets and for each frequent closed itemset *f*, let denote G_f the set of generators of *f*. The *generic basis* for exact association rules is:

$GB = \{r : g \Rightarrow (f \setminus g) \mid f \in FC \wedge g \in G_f \wedge g \neq f\}$.

Let denote *G* the set of generators of *FC*. The *informative basis* for approximate association rules is:

$IB = \{r : g \rightarrow (f \setminus g) \mid f \in FC \wedge g \in G \wedge c(g) \subset f\}$.

The informative basis can be restricted to its transitive reduction so that generator g ($c(g) \subset f$) is an immediate predecessor of f ($c(g) <\cdot\, f$).

The generic basis for exact rules for data given in Table 2.2 is: $\{W \Rightarrow \boldsymbol{C}$, $T \Rightarrow \boldsymbol{C}$, $D \Rightarrow \boldsymbol{C}$, $A \Rightarrow \mathbf{CW}$, $TW \Rightarrow \mathrm{AC}$, $AT \Rightarrow \mathbf{CW}$, $DW \Rightarrow \boldsymbol{C}\}$.

This set is not the same as the generating set for 100% confidence (i.e. exact) rules by Zaki (see p.36). While Zaki constructs the rules between (adjacent) equivalence classes, Bastide et al construct the rules inside equivalence classes. Only these rules can coincide where the consequent ($f \setminus g$) is a minimal generator of another (adjacent) closed set (in our example three rules with consequent ***C***). Both approaches serve the purpose of creating a non-redundant set of exact association rules (from which all other exact rules can be inferred). In my opinion the second set of rules is better comprehensible for the user.

2.4.2.2 Negative association rules

Association rules in the form $X \Rightarrow Y$ (or $X \rightarrow Y$) show associations between present items and are called *positive association rules*. Besides them the rules considering absent items might be of interest. The absence of an item *X* is shown as a negation of *X*: $\neg X$. Association rules with negated antecedent and/or negated consequent are called *negative association rules*. There are three forms of them: $\neg X \Rightarrow Y$, $X \Rightarrow \neg Y$, $\neg X \Rightarrow \neg Y$.

Support and confidence of negative association rules can be found using supports of positive itemsets by the following formulas (Wang, Zhang, & Chen, 2008):

$\sigma(\neg X) = 1 - \sigma(X)$;

$q(\neg X \Rightarrow Y) = \sigma(Y) - \sigma(X \cup Y)$;

$q(X \Rightarrow \neg Y) = \sigma(X) - \sigma(X \cup Y)$;

$q(\neg X \Rightarrow \neg Y) = 1 - \sigma(X) - \sigma(Y) + \sigma(X \cup Y)$;

$p(C_1 \Rightarrow C_2) = q(C_1 \Rightarrow C_2) / \sigma(C_1)$, where $X \cap Y = \emptyset$, $C_1 \in \{X, \neg X\}$, $C_2 \in \{Y, \neg Y\}$.

Between confidences of different rule forms the following holds (Dong, Sun, Han, & Hou, 2006):

$p(A \Rightarrow B) + p(A \Rightarrow \neg B) = 1$ and

$p(\neg A \Rightarrow B) + p(\neg A \Rightarrow \neg B) = 1$.

Overview of negative association rules can be found in (Antonie, Li, & Zaiane, 2014).

2.5 Combining classification and association rules

ARM (see 2.4) represents descriptive rule learning whereas classification (see 2.3) is a representative of predictive rule learning. Besides different goals there are other differences between the two. In case of ARs all attributes are “equal” in the sense that they can appear both in the antecedent and the consequent (not at the same time). In case of classification the consequent is determined and this class attribute cannot appear in the antecedent. Thus we can say that ARM is a symmetric task regarding attributes and classification is asymmetric.

Further, there are more profound, “semantic” differences between the classification and ARM tasks. As brought out by Freitas (2000) classification has to have an inductive bias (i.e. the basis for favouring one hypothesis over another), it has to fight against overfitting and underfitting[13] and moreover, there is no guarantee that discovered rules have a high predictive accuracy on unseen data, because induction is not truth-preserving. To theoretically overcome the last problem, the inductive learning hypothesis (see p.25) is used. There are no such problems with association rules.

To mine these two types of rules, different techniques are used; ARM algorithms often find many more rules than classification rule mining algorithms.

In addition to ARM and classification there exist approaches that combine these two: associative classification for predictive purpose and subgroup discovery for descriptive purpose.

Associative classification (AC) is a hybrid approach that uses association rule discovery methods to build classifiers (Thabtah & Cowling, 2007). ARM

[13] Overfitting refers to a model that models the training data too well. Underfitting refers to a model that can neither model the training data nor generalize to new data. Both cause poor performance of ML algorithms. (Brownlee, 2017)

algorithms are adapted to discover class-association rules (CARs) – a special subset of association rules whose right-hand-side are restricted to the classification class attribute (Liu, Hsu, & Ma, 1998). Mining CARs is discovery of all classification rules that satisfy the minimum support (*minsup*) and the minimum confidence (*minconf*) thresholds (Nguyen L. , Vo, Hong, & Thanh, 2013). Further suitable rules are selected to form a classifier and this classification model is used to predict class values on new unseen data (Wedyan, 2014). Obviously, AC is a predictive technique.

This approach is reported to give more accurate set of rules than traditional classification approaches. This is due to mining more complete rule set than in case of traditional classification algorithms. At the same time, finding a bigger set of rules takes more time. (Nguyen L. , Vo, Hong, & Thanh, 2013)

Since founding by Liu, Hsu and Ma (1998) many different AC algorithms have been proposed, e.g. CBA (Liu, Hsu, & Ma, 1998), CMAR (Li, Han, & Pei, 2001), CPAR (Yin & Han, 2003), MCAR (Thabtah, Cowling, & Peng, 2005), ACCF (Li, Qin, & Yu, 2008), ACCR (Niu, Xia, & Zhang, 2009), ADA (Wang, Yue, Niu, & Shi, 2011), ACAC (Huang, Zhou, He, & Wang, 2011), CAR-Miner (Nguyen L. , Vo, Hong, & Thanh, 2013). AC approach has been extended to multi-label classification[14] (MMAC (Thabtah, Cowling, & Peng, 2004), RMR (Thabtah & Cowling, 2007), CLAC (Veloso, Meira, Gonçalves, & Zaki, 2007)); to incorporate negative rules (e.g. ARC-PAN (Antonie & Zaïane, 2004), ACN (Kundu, Islam, Munir, & Bari, 2008)) and has been parallelized (e.g. (Nguyen, Vo, & Le, 2014)). For getting an overview of AC we suggest (Wedyan, 2014).

Under descriptive rule discovery, another approach besides ARM, is *subgroup discovery (or subgroup mining)* (Fürnkranz & Kliegr, 2015).

Subgroup discovery (SD) is a descriptive data mining technique using supervised learning (Carmona C. J., González, del Jesus, & Herrera, 2014), it is a descriptive induction technique that extracts interesting relations among different variables with respect to a special property of interest known as target variable (Helal, 2016). Subgroup discovery results in individual rules, where the rule conclusion is a class (the property of interest). The main difference between learning of classification rules and subgroup discovery is that the latter induces single rules (subgroups) of interest, which aim at revealing interesting properties of groups of instances, not necessarily at forming a rule set used for classification. Being descriptive by its purpose, it is at the same time a form of supervised learning due

[14] Multi-label classification consists in learning a model from instances that may be associated with multiple labels (Veloso, Meira, Gonçalves, & Zaki, 2007); while usually (in case of single-label learning) there is only one class label for each instance/example. Overview of multi-label classification can be found in (Tsoumakas & Katakis, 2007) and multi-label learning (including other than rule-based methods as well) in (Madjarov, Kocev, Gjorgjevikj, & Džeroski, 2012), (Zhang & Zhou, 2014).

to a defined property of interest acting as a class. (Fürnkranz, Gamberger, & Lavrač, 2012) Thus this task is considered to be at the intersection of descriptive and predictive induction (Lavrač, Kavšek, Flach, & Todorovski, 2004). SD can be seen as a special case of a more general rule learning task (Novak, Lavrač, & Webb, 2009).

The task of subgroup discovery was defined by Klösgen (1996) and Wrobel (1997) as follows: *Given a population of individuals and a property of those individuals that we are interested in, find population subgroups that are statistically 'most interesting', for example, are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest.* Thus, only the most interesting (the most unusual) rules are searched for. In order to evaluate rule's interestingness different quality measures are used (see (Herrera, Carmona, González, & del Jesus, 2011)).

Both classification rule learners like CN2 (Clark & Niblett, 1989) and ARM algorithms like Apriori (Agrawal & Srikant, 1994) and FP-Growth (Han, Pei, & Yin, 2000) have been adapted to subgroup discovery. Besides them evolutionary algorithms have been used.

Pioneering algorithms EXPLORA (Klösgen, 1996) and MIDOS (Wrobel, 1997) are extensions of classification algorithms using decision trees and performing exhaustive (or heuristic) search. Subsequent algorithms, e.g. SubgroupMiner (Klösgen & May, 2002), SD (Gamberger & Lavrač, 2002) and CN2-SD (Lavrač, Kavšek, Flach, & Todorovski, 2004) use beam search (p.29).

Apiropri-based extensions of association algorithms Apriori-SD (Kavšek & Lavrač, 2006) and its successor SD4TS (Mueller, et al., 2009) use also beam search, whereas algorithms based on FP-Growth – SD-Map (Atzmüller & Puppe, 2006), DpSubgroup (Grosskreutz, Rüping, & Wrobel, 2008) and Merge-SD (Grosskreutz & Rüping, 2009) – perform exhaustive search.

Evolutionary algorithms for extracting subgroups SDIGA (del Jesus, González, Herrera, & Mesonero, 2007), MESDIF (Berlanga, del Jesus, González, Herrera, & Mesonero, 2006) and NMEEF-SD (Carmona C. , González, del Jesus, & Herrera, 2010) use genetic algorithm[15].

For overview of SD we suggest (Herrera, Carmona, González, & del Jesus, 2011), an overview of evolutionary SD can be found in (Carmona C. J., González, del Jesus, & Herrera, 2014).

[15] Genetic algorithm is an evolution-inspired computational model and global optimization technique developed by John Holland in 1975 (book titled *Adaptation in Natural and Artificial Systems*).

Novak, Lavrač and Webb (2009) gather subgroup discovery together with contrast set mining[16] and emerging pattern mining[17] under the unifying framework called *supervised descriptive rule discovery* that deals with finding interesting rules from class labelled data. All three methods are found to be compatible in most of compared aspects[18].

Alternatively, (Bringmann, Nijssen, & Zimmermann, 2009) see these three and some more approaches (correlated patterns[19], discriminative patterns[20], interesting rules[21]) as *pattern-based classification* methods where the "patterns define new features, which can be used in a classification model". Associative classification fits under this view as well.

2.6 Theory of Monotone Systems

According to J. Mullat's works (1976) (1977) the definitions of basic concepts of MS are as follows.

Let a finite discrete *set* X and function π_x on it which maps to each *element* $\alpha \in X$ a certain nonnegative number $\pi_x(\alpha)$, be given.

The function π_x is called a *weight function* if it is defined on any subset $X' \subseteq X$; the number $\pi_x(\alpha)$ is called a *weight* of element α on X'.

A set X with a weight function π_x is called a *system* and is denoted by $S=(X, \pi_x)$.

The system $S'=(X', \pi_{x'})$ where $X' \subseteq X$ is called a *subsystem* of the system $S=(X, \pi_x)$.

The system $S=(X, \pi_x)$ is called *monotone* if in the case of any $\alpha \in X' \setminus \{b\}$, $b \in X$:

$\pi_{x' \setminus \{b\}}(\alpha) \leq \pi_{x'}(\alpha)$ where X' is any subset of X.

---

[16] Contrast set mining (Bay & Pazzani, 2001) searches for discriminating characteristics of groups called contrast sets.

[17] Emerging pattern mining aims at discovering itemsets whose support increases significantly from one data set to another (Dong & Li, 1999).

[18] Although the definitions of the three appear different, the goals of these three mining tasks are very similar, it is primarily the terminology that differs (Novak, Lavrač, & Webb, 2009).

[19] Correlated association rules are ARs that have χ^2 values between the assumption and the conclusion no less than user-specified minimum cutoff value (Morishita & Sese, 2000).

[20] Discriminative patterns w.r.t. class labels are identified using information gain or Fisher score or other measures (Cheng, Yan, Han, & Hsu, 2007).

[21] Interestingness of a rule can be measured according some interestingness metric; (Bayardo Jr. & Agrawal, 1999) use information-theoretic measures like entropy, gini index, χ^2 and others. A survey on interestingness measures can be found in (Geng & Hamilton, 2006).

Let be given data set $X(N,M)$, where N is the number of objects (i=1,…,N) and M is the number of attributes (j=1,…,M). Element α can be X_{ij}, row i, column j or any subtable of X.

To build a monotone system we have to fulfil two conditions:

1. There has to be a weight function $\pi_x(\alpha)$ which will give a measure of influence for every element α of the monotone system on X;
2. Certain activities (adding or removing) can be applied to the elements. There have to be rules f to recompute the weight of the system elements after used activities. Weights can be changed only to one direction (increasing or decreasing).

These conditions give a lot of freedom (to user) to choose the weight functions and rules of weight change in the system. The only constraint we have to keep in mind is that after eliminating all elements α from the system X the final weights of $\alpha \in X$ must be equal to zero.

In our case:

1. A suitable weight function is object's frequency in a (concerned) system. In case of tables of object-attribute type we weigh the attribute's value and the weight is a number of objects having that certain value.
2. Rules for recomputing the weights:
   - Choose the element(s) of interest.
   - Extract the objects having the(se) element(s) from the concerned set. So the set of objects under consideration can only decrease.
   - For the rest of objects calculate new weights using the same weight function. If there are no objects with given elements then the weight is zero.

2.6.1 Algorithm MONSA

MONSA (MONotone System Algorithm) – a hierarchical clustering algorithm based on monotonic systems was introduced by R. Kuusik (1993).

The problem is stated as follows:

Suppose that the finite discrete data matrix $X(N,M)$ (the set of objects X) of object-attribute type is given, where N is the number of objects (examples) and M is the number of attributes. Every attribute j can acquire integer values in the interval $h_j = 0,1,2,...,K_j\text{-}1$.

We should find all existing value combinations (VC) of attributes in set X.

Every VC describes a certain subset X_{VC} of objects and subset X^{VC} of elements in set X. The last set is called a cluster in the theory of classification – that is why we call the process of extracting VCs clustering.

An *element* is an attribute with certain value. We will denote it as **Attribute.value**.

A central concept used here is an intersection. The *intersection* of two (or more) sets is the set of elements, which belong to both (all) sets, simultaneously. Table 2.4 presents an intersection of two objects described by three attributes. The intersection is **A1.1 AND A3.2**.

Table 2.4. Intersection of X(2,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	2	2
O2	1	1	2
Intersection	**1**	*	**2**

For finding really existing VCs the intersections over different sets of objects are found.

Earlier algorithms for that same task had the following drawbacks (Kuusik, 1993):

- Majority of found intersections are either empty or repeating;
- Determining a VC originality (i.e. whether or not it is already generated) is very time-consuming;
- Because of spontaneous intersection, those algorithms are very difficult (if not impossible) to use for optimizing tasks.

The main reason is that the order of finding intersections depends on the order of objects in data table.

MONSA is free of abovementioned deficiencies. It finds only existing VCs without repetitions (in different order of elements).

The algorithm uses frequency tables (FT). A frequency table contains the counts of occurrences of all existing values for each attribute. Each attribute can have a different number of different discrete values.

Table 2.5. Data table X(2,3) and corresponding frequency table

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	2	2
O2	1	1	2

Value \ Attribute	*A1*	*A2*	*A3*
1	**2**	1	0
2	0	1	**2**
Intersection	1	*	2

In Table 2.5 data table X(2,3) from previous example (Table 2.4) is given together its corresponding frequency table. Zero in FT shows that corresponding element does not appear in the data table.

As we see from Table 2.4 and Table 2.5 an intersection is also findable through the frequencies: every value which has frequency equal to the number of objects in the table (shown in bold) belongs to the intersection: A1.1 AND A3.2 is an intersection with frequency 2. This way MONSA finds intersections.

In order to intersect suitable sets of objects, the next node is chosen by element's frequency (from FT) and special "elimination" techniques are used to prevent repetitions. The algorithm works in a depth-first search manner.

From the initial data set MONSA finds a result as a set of intersections (closed sets[22]) and/or a set of trees (forest), they are listed in the order they are found, that order does not depend on the initial order of objects (records). The frequencies of nodes (intersections) decrease strictly along branches of a tree(s). The decrease makes allowable to prune the branches according to the minimal allowed frequency (support). This is similar to the other tree-based algorithms, for example ChARM (Zaki & Hsiao, 2002). The fact that the decrease is strict gives a high potential to the intersection (combination from root to the certain node) to be a closed set. At any level the descendants of a common parent-node are found in a weakly decreasing order of their frequencies, the roots also are found in a weakly descending order. The order of nodes with equal frequency depends on the searching principle (usually by columns or by rows of frequency table).

For every extract of objects its corresponding frequency table is formed. Zero in the initial frequency table means that the corresponding element does not exist. During the work the elements that are exhaustively analysed are set to zero in frequency tables. Such prohibited (eliminated) elements are not included into the intersections any more, only the elements with frequency over zero (or some higher threshold) are considered.

By essence MONSA is a recursive algorithm. Here we present its backtracking version[23] (Kuusik & Lind, 2008).

In this algorithm the following denotations are used:

`t`	the number of the step (or level) of recursion
$\mathtt{FT}_t$	frequency table for a set $\mathtt{X}_t$
$\mathtt{IntSec}_t$	vector of elements over set $\mathtt{X}_t$ (intersection)
`Init`	activity for initial evaluation

[22] The correspondence between intersection and closed set is shown in 3.1.1.

[23] Recursive version of MONSA (containing further developments) is given in 3.1.8

<u>Algorithm MONSA</u>

```
1:  Init
2:  t←0, IntSec_0←{}
3:  Find a table of frequencies FT_0 for all attributes in X_0
4:  DO WHILE there exists FT_s#Ø in {FT_s}, s≤t
5:    FOR EACH element h_f∈FT_t with frequency V=max FT_t(h_f)#0 DO
6:      IF pruning is needed (h_f has to be pruned) THEN GOTO BACK
7:      Separate submatrix X_{t+1}⊂X_t such that X_{t+1}={Xij∈X_t|X.f=h_f}
8:      Find a table of frequencies on X_{t+1} FT_{t+1}
9:      ZeroesDown(t+1)
10:     CheckUniqueness(t+1)
11:     IF new intersection is unique THEN
12:       Add elements j with FT_{t+1}(j)=V into vector IntSec_{t+1}
13:       BackwardComparison(t+1)
14:       Output of IntSec_{t+1}
15:       IF there exist attributes to analyse THEN t←t+1
16:     ENDIF
17:   NEXT
18:   BACK: t←t-1
19:   IntSec_{t+1}←IntSec_t
20: ENDDO
21: All intersections are found
22: END: end of algorithm
```

<u>Elimination (pruning) activities:</u>

```
1)
ZeroesDown(t+1)
    FOR EACH element h_u∈FT_t DO
      IF FT_t(h_u)=0 THEN FT_{t+1}(h_u)←0
    NEXT

2)
BackwardComparison(t+1)
    FOR EACH element h_u∈FT_{t+1} with frequency #0 DO
      IF FT_{t+1}(h_u)=FT_t(h_u) THEN FT_t(h_u)←0
    NEXT

3)
CheckUniqueness(t+1)
    IF there exists on X_{t+1} h_u, 1≤u≤M such, that
    [h_u∈IntSec_{t+1} AND FT_{t+1}(h_u)=0 AND frequency of h_u in X_{t+1}=V]
    THEN
      Intersection is not unique
```

```
ELSE
  Intersection is unique
ENDIF
```

MONSA is a depth-first search algorithm which backtracks when the current branch is exhausted or has to be pruned. Inside the backtracking algorithm the main steps at each level are as follows:

S1: Choose a "leading" element – the first element with maximal frequency (over zero; at least with threshold frequency specified by the user) (line 5), add it into the (potential) intersection and zerofill the corresponding cell in the frequency table

S2: Calculate the next frequency table for objects containing that leading element (line 8)

S3: If there exist element(s) with frequency equal to the leading one check the intersection's uniqueness (line 10) => if it is unique, add these elements (with frequency equal to the leading frequency) into the intersection (line 12); otherwise backtrack

S4: Output intersection (line 14)

S5: "Bring down" zeroes from the frequency table of the previous level i.e. at the current level zerofill all elements that have been zerofilled at the previous level (line 9)

S6: "Backward comparison": elements that have equal frequencies at both levels are zerofilled at the previous level (line 13)

In order to avoid finding "repetitions" i.e. permutations of already found intersections two elimination techniques have been used in algorithm MONSA:

"bringing zeroes down" – activity that prohibits arbitrary output repetition of already separated intersection at the next (deeper) level(s);

"backward comparison" – activity that does not allow the output of the separated intersection at the same (current) level and also at previous (higher) levels (after backtracking).

Impact of these techniques is proved in (Kuusik, 1993)[24].

Appears that these two activities do not prevent repetitious finding (and output) of some subsets of already found intersections. Those repetitions are avoided by "uniqueness check" (that will be explained in 2.6.1.1).

[24] Theorems 5.3 and 5.4 accordingly

Without these elimination techniques the algorithm would find all permutations of all existing value combinations.

Turning back to before listed drawbacks of earlier algorithms for finding all existing value combinations (see p. 43) we have to point out:

- Empty (i.e. non-existing) combinations are avoided by the nature of algorithm, they are not generated (and checked for existence) at all.
- Repetitions are avoided by elimination techniques (introduced above).
- A VC originality is checked using existing frequency tables, there is no need to look through already found intersections.
- The order of finding intersections is driven by data, not by the order in which the objects are presented.

The most important advantages of the algorithm are:

- The frequency is known at the moment the new node is found.
- The ability to find nodes consisting of more than one element, this reduces the number of nodes and the size of the tree.
- The possibility to output the tree immediately during the finding closed sets.
- The possibility to use different pruning criteria that comply with monotonic decrease of the frequency.
- Attributes can have more values than only 0/1.
- The use of new original and very effective elimination techniques.
- In order to prevent repetitions we do not look through the already found result and therefore we do not need additional data structures.

Demonstration how MONSA works is shown in (Kuusik & Lind, 2008) (see Appendix A). In this example the frequency threshold is used for pruning.

Denotations and definitions used for MONSA (Kuusik, 1993) will be given in 3.1.2 where their relations with the concepts used for frequent itemsets (see 2.4.1) will be shown.

2.6.1.1 Explanation of "uniqueness check"

The subset of already found intersection is redundant only if both have equal frequencies (i.e. this (sub)set is non-closed). If subset's frequency is higher (than its superset's) then it covers more objects and is not redundant. Subset's frequency cannot be smaller.

Finding a superset of already found set with the same frequency is impossible, because at any level MONSA finds all co-existing (i.e. contained in the same objects) elements with equal frequencies as one intersection (i.e. maximal EC[25]).

[25] Defined in 3.1.2, p. 65

Table 2.6. Example X(9,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	4	0	8
O2	5	3	4
O3	5	4	3
O4	5	0	7
O5	3	1	8
O6	3	1	8
O7	4	1	7
O8	4	0	8
O9	5	4	3

```
(a) Result as a set of      (b) Result as a set of
trees:                      intersections:

(4) 0.500(2)                I1)  A1.5=4
A1.5=>A2.4&A3.3             I2)  A1.5&A2.4&A3.3=2
    0.250(1)                I3)  A1.5&A2.0&A3.7=1
    =>A2.0&A3.7             I4)  A1.5&A2.3&A3.4=1
    0.250(1)
    =>A2.3&A3.4

(4) 0.500(2)                I5)  A3.8=4
A3.8=>A1.3&A2.1             I6)  A3.8&A1.3&A2.1=2
    0.500(2)                I7)  A3.8&A1.4&A2.0=2
    =>A1.4&A2.0

 (3) 0.667(2)               I8)  A1.4=3
A1.4=>A2.0                  I9)  A1.4&A2.0=2
    0.333(1)                I10) A1.4&A2.1&A3.7=1
    =>A2.1&A3.7

 (3) 0.333(1)               I11) A2.0=3
A2.0=>A3.7                  I12) A2.0&A3.7=1

 (3) 0.333(1)               I13) A2.1=3
A2.1=>A3.7                  I14) A2.1&A3.7=1

 (2)
A3.7                        I15) A3.7=2
```

Figure 2.2 Result found by MONSA (without uniqueness check)

Next we give an example of redundant subsets under consideration. For that we have used MONSA without "uniqueness check" (of a new intersection).

Having the initial data set of nine objects described by three attributes (see Table 2.6) MONSA (without uniqueness check) finds fifteen intersections (with minimal frequency allowed =2). Both representation forms – trees and intersections – are given in Figure 2.2.

Here we have six trees and six roots accordingly. Intersections I1, I5, I8, I11, I13 and I15 correspond to the roots.

Intersection I11 (A2.0=3) is not redundant although A2.0 is contained in the intersections I3, I7 and I9, because these three intersections have lower frequencies and none of them cover all objects covered by I11.

Intersections I9, I12 and I14 are redundant:

- I9 (A1.4&A2.0) is a subset of I7 (A3.8&A1.4&A2.0) with the same frequency =2 (both cover objects O1 and O8);
- I12 (A2.0&A3.7) is a subset of I3 (A1.5&A2.0&A3.7), both frequencies are 1 (they cover object O4) and
- I14 (A2.1&A3.7) is a part of I10 (A1.4&A2.1&A3.7), frequencies equal 1 (they cover O7).

There have to be 12 intersections instead of 15.

Starting from the root of a tree the frequencies of intersections (nodes) always (strictly) decrease along any branch of the tree (due to finding maximal EC at every node). As no branch has two intersections with the same frequency, the redundant subsets do not occur in the same branch, they can appear in different branches of a tree or in different trees.

Among siblings (i.e. direct descendants of a common node) any element can appear in only one intersection. Consequently, redundant subsets (under consideration) do not occur among siblings. This is also true for the root-level (which formally consists of descendants of the initial empty set), therefore these redundant subsets never occur at root-level.

Intersection (closed set) and its redundant sub-intersection (with the same frequency) do not have to appear at the same level of a tree (as in all three cases of our example).

Element(s) that appear in the root node are eliminated from further analysis by zerofilling the corresponding cell(s) in the frequency table. Elements that are fully analysed (exhausted) at deeper levels are prohibited by "backward comparison". All these zeroes are "brought down" to all succeeding levels and therefore prohibited elements never occur in redundant intersections. So (due to the elimination techniques used) no permutation of whole (already found) intersection (closed set) is not found. Elements that are partially analysed (at the non-root levels) are not eliminated. All this is true for any subtree also.

Although prohibited elements are eliminated from the frequency tables they still appear in the subsets of objects extracted by non-prohibited elements. Remind that some set (of elements) and its subset with the same frequency define the same set of objects. If some prohibited (and eliminated) element appears in all objects (of subset) it means that this subset has been analysed already (this element can not be contained in the value combination (potential intersection) on which basis that subset of objects was extracted). All intersections containing a prohibited element are already found and such subsets need no more analysis. This situation indicates a repetitive extraction of certain subset of objects.

To exclude redundant subsets (sub-intersections) we have to detect a situation when some prohibited element occurs in all objects and stop analysing such branch.

In our example (see Table 2.6 and Figure 2.2) intersection I7 (A3.8&A1.4&A2.0) has one more element than redundant set I9 (A1.4&A2.0), namely A3.8. The set of objects extracted by I9 is the same as by I7. Consequently, actual frequency of A3.8 is as much as the number of objects in that set (=2). As A3.8 was prohibited after exhaustive analyzation, the frequency table contains zero in the corresponding cell. Detecting such situation we can say that A1.4&A2.0 (I9) is a redundant set.

Such "uniqueness check" is used by MONSA. It is not necessary to look through the already found intersections to ensure the new one is non-redundant.

Correctness of 'uniqueness check' explained here is proved in (Kuusik, 1995).

It is interesting that those redundant subsets seem to be the same ones for what ChARM (Zaki & Hsiao, 1999) (2002) needs "subsumption checking". In order to ensure that a candidate set is really closed ChARM looks through the (certain) already found closed sets, only those that have 1) the same "tidsum" and 2) the same support (frequency) as the candidate set. (Tidsums of different closed sets with equal frequency tend to be different). For the complete description see (Zaki & Hsiao, 1999) (2002).

As shown already we perform the uniqueness check of a new intersection (potential closed set) otherwise, without looking through the already found (closed) sets.

2.7 Determinacy analysis

Determinacy Analysis (DA) is a system of methods for the analysis of rules that was created at the end of 70s. Its approach combines mathematical statistics and logic. DA's methodology and the underlying mathematics are developed by Russian scientist Sergei Chesnokov (1980a) (1982).

DA-technology provides an alternative way to perform factor analysis of qualitative and quantitative variables. It assists in obtaining regularities, explanations and prognostic rules.

The goal of DA is to describe a given selection of objects (class Y), answering the questions: "Who are they (objects of the class)?", "How can we describe them?", "What distinguishes them from others?".

The following overview of determinacy analysis is based on Chesnokov (1980a) (1982) (2002), (DALSolution, 2007), (Context, 1999b). In 2.7.7 our opinion is presented.

2.7.1 Determination and its characteristics

The idea behind DA is that a rule can be found based on the frequencies of joint occurrence or non-occurrence of events. Such a rule is called determinacy or determination and the mathematical theory of such rules is called determinacy analysis.

If it is observable that an occurrence of X is always followed by an occurrence of Y, it means that there exists a rule "If X then Y", or $X \rightarrow Y$. Such correlation between X and Y is called *determination* (from X to Y). Here X is the *determinative (determining)* and Y is the *determinable*.

Each determination holds in the given context. It might not to be valid in some other context. Context is the characteristic (set of characteristics) on which basis the data set is formed. It is common to all objects of that set. The universal context is characteristic to the initial data set. Often it is not explicitly shown. Extracting a subset of data by some characteristic we determine a narrower, usual (non-universal) context. A determination $X \rightarrow Y$ in the context k is written as $kX \rightarrow kY$ or equivalently $k(X \rightarrow Y)$. Universal context (denoted by ω) is usually omitted.

The determinative (X) consists of one or more factors. A factor is an attribute with its certain value. Each attribute can have many different discrete values and gives as many different factors as many different values it has. Factors coming from the same attribute are not contained in the same X.

Each rule has two characteristics: accuracy and completeness[26].

[26] In the beginning (in (Chesnokov, 1980a), for example) "accuracy" (Russian "*точность*") was called "intensity" and "completeness" ("*полнота*") was called "capacity" ("*емкость*"). More exactly, intensity shows the accuracy (or validity) of determination and capacity shows the completeness (Chesnokov, 1982, p. 24).

Accuracy of determination $X \rightarrow Y$ shows to what extent X determines Y. It is defined as the proportion of occurrences of Y among the occurrences of X:

$A(X \rightarrow Y) = n(X\,Y) / n(X)$, where

$A(X \rightarrow Y)$ is the accuracy of determination,
$n(X)$ is the number of objects having feature X and
$n(X\,Y)$ is the number of objects having both features X and Y.

Completeness of determination $X \rightarrow Y$ shows which part of cases having feature Y can be explained by determination $X \rightarrow Y$. It is the percentage of occurrences of X among the occurrences of Y:

$C(X \rightarrow Y) = n(X\,Y) / n(Y)$, where

$C(X \rightarrow Y)$ is the completeness of determination,
$n(Y)$ is the number of objects having feature Y and
$n(X\,Y)$ is the number of objects having both features X and Y.

Both accuracy and completeness can have values ranging from 0 to 1 (0% .. 100%). A value of 1 shows maximum accuracy or completeness, 0 means that the rule is not accurate or complete at all. A value between 0 and 1 shows quasideterminism.

If all objects having feature X also have feature Y then the determination is (maximally) accurate. In case of accurate determination $A(X \rightarrow Y) = 1$ (100%).

The majority of rules are not accurate. In case of inaccurate rule $A(X \rightarrow Y) < 1$.

In order to make a determination more (or less) accurate, complementary factors are added to the left part of the rule. Adding factor Z into the rule $X \rightarrow Y$, we get the rule $XZ \rightarrow Y$, adding factor W to the rule $XZ \rightarrow Y$, we get the rule $XZW \rightarrow Y$ etc.

The *contribution* of factor *Z to the accuracy* of the rule $XZ \rightarrow Y$ is measured by the increase of accuracy $\Delta A(Z)$ caused by addition of factor Z into the rule $X \rightarrow Y$:

$\Delta A(Z) = A(XZ \rightarrow Y) - A(X \rightarrow Y)$.

The contribution to accuracy can range from -1 to 1.

If $\Delta A(Z)>0$ then Z is a *positive factor*. Adding a positive factor makes the rule more accurate, sometimes the resultant rule is (maximally) accurate. If $\Delta A(Z)<0$ then Z is a *negative factor*. Adding a negative factor decreases the rule's accuracy, sometimes down to zero. If $\Delta A(Z)=0$ then Z is a *zero (or inessential) factor*. Adding a zero factor does not change the rule's accuracy. An *accurate rule* contains no negative factors, all factors are positive or zero factors. A rule consisting of positive factors only, is called a *normal rule*.

If $C(X \to Y)=1$ (100%) then the rule $X \to Y$ is (maximally) complete. It means that Y is always explained by X. In case of an incomplete rule $C(X \to Y)<1$, X does not explain all occurrences of Y.

The *contribution* of factor *Z to the completeness* of the rule $XZ \to Y$ is measured by the increase of completeness $\Delta C(Z)$ by addition of factor Z into the rule $X \to Y$:

$$\Delta C(Z) = C(XZ \to Y) - C(X \to Y)$$

The contribution of whatever factor to completeness is negative or zero.

2.7.2 System of rules

A *system of rules* is a set of rules in the form $S_q = \{x_i \to y \mid i=1,2,...,q\}$, where q is the number of rules.

Every system is characterised by average accuracy, summarised completeness and summarised capacity (the number of objects/cases covered by the rules).

Accuracy of system of rules is:

$$A(S_q) = A((\bigcup_{i=1}^{q} x_i) \to y) = \frac{n(y \bigcup_{i=1}^{q} x_i)}{n(\bigcup_{i=1}^{q} x_i)}.$$

Completeness of system of rules is:

$$C(S_q) = C((\bigcup_{i=1}^{q} x_i) \to y) = \frac{n(y \bigcup_{i=1}^{q} x_i)}{n(y)}.$$

Capacity of system of rules is:

$$n(S_q) = n(\bigcup_{i=1}^{q} x_i).$$

System of rules $S_q = \{x_i \to y \mid i=1,2,...,q\}$ is *additive* when x_i-s pairwise do not overlap (i.e. do not cover the same objects). The capacity of additive system is equal to the sum of capacities of rules it consists of:

$$n(S_q) = n(\bigcup_{i=1}^{q} x_i) = \sum_{i=1}^{q} n(x_i).$$

In case of additive system the previously given formulas can be simplified.

Completeness and capacity of an additive system are just summed up completenesses and capacities of rules:

$$C(S_q) = C((\bigcup_{i=1}^{q} x_i) \to y) = \frac{\sum_{i=1}^{q} n(yx_i)}{n(y)} = \sum_{i=1}^{q} C(x_i \to y);$$

$$n(S_q) = \sum_{i=1}^{q} n(x_i).$$

Accuracy of additive system is not additive (i.e. equal to the sum of rules' accuracies). It is found as a weighted average:

$$A(S_q) = A((\bigcup_{i=1}^{q} x_i) \to y) = \frac{\sum_{i=1}^{q} n(yx_i)}{\sum_{i=1}^{q} n(x_i)} = \sum_{i=1}^{q} \frac{n(x_i)}{\sum_{k=1}^{q} n(x_k)} A(x_i \to y) =$$

$$= \sum_{i=1}^{q} A((\bigcup_{k=1}^{q} x_k) \to x_i) A(x_i \to y).$$

Rank of a rule is a dimension of its left side. Rule in the form $z_1z_2...z_r \to y$ is called a rule of rank r ($r \geq 1$).

System of rules in the form $z_1z_2...z_r \to y$ is called a system of rules of rank r by variables $z_1, z_2, ..., z_r$ relative to feature y.

Every system of rules of rank $r \geq 1$ by fixed set of r variables is additive (Chesnokov, 2002).

System is called *complete* if its completeness is 1. System of rules of rank r by variables $z_1, z_2, ..., z_r$ is complete if it contains all existing rules in the form $z_1z_2...z_r \to y$.

System is called *accurate* if its accuracy is 1. System is accurate when all of its rules are accurate.

An *accurate rule* has no negative factors, all factors are positive or zero factors (Chesnokov, 2002).

A rule in the form $z_1z_2...z_r \to y$ is called *normal rule* (of rank r) if all factors $z_1, z_2, ..., z_r$ are positive. Positive factor (called also binder) makes rule more accurate than it was without that factor.

System S_r consisting of all normal rules in the form $z_1z_2...z_r \to y$ is called *normal system of rank r*. Normal system of rank r is additive i.e. its rules do not intersect pairwise.

Let be given m variables $z_1, z_2, ..., z_m$ and feature y. A *canonical system of order m* is a system of rules that joins all normal systems of rules in the form $S_{jr} = \{z_{j1}z_{j2}...z_{jr} \to y\}$ where $z_{j1}, z_{j2}, \dots, z_{jr}$ – all possible combinations by r variables from set of variables $z_1, z_2, ..., z_m$.

In case of fixed r there are $\binom{m}{r} = \frac{m!}{r!(m-r)!}$ normal systems $S_{jr} = \{z_{j1}z_{j2}...z_{jr} \rightarrow y\}$, j_r gets values from 1 to $\binom{m}{r}$.

Canonical system of order m contains 2^m-1 normal systems. Variables z_1, z_2, ..., z_m are called the basis of canonical system. In general, a canonical system of order m>1 is not additive.

2.7.3 Examples of different systems of rules

Here we present some examples of different accurate and complete systems of rules using data from (Quinlan, 1984). This data table (see Table 2.7) contains 8 objects described by 4 attributes. The last attribute shows object's affiliation to certain class. Attributes Height, Eyes and Class have two possible values each, attribute Hair has three alternative values.

Table 2.7. Data from (Quinlan, 1984)

Height	**Hair**	**Eyes**	**Class**
tall	dark	blue	–
short	dark	blue	–
tall	blond	blue	+
tall	red	blue	+
tall	blond	brown	–
short	blond	blue	+
short	blond	brown	–
tall	dark	brown	–

The purpose is to determine Class '–' by attributes Eyes and Hair.

Next four different systems of rules are given, based on the same data.

1. Additive system with fixed number of factors (r=2) *S1*:
   - Eyes.blue & Hair.dark → Class.– (C = 40%; 2 objects)
   - Eyes.brown & Hair.dark → Class.– (C = 20%; 1 object)
   - Eyes.brown & Hair.blond → Class.– (C = 40%; 2 objects)

The system is (maximally) accurate (overall accuracy is 100%) and (maximally) complete (sum of completenesses is 100%). Maximal completeness shows that all objects (belonging to class) are covered and maximal accuracy says that there is no need to add any more factors into analysis.

2. Additive system consisting of accurate rules of different rank (number of factors) could be *S2*:
   - Hair.dark → Class.– (C = 60%; 3 objects)
   - Hair.blond & Eyes.brown → Class.– (C = 40%; 2 objects)

or *S3*

- Eyes.brown → Class.– (C = 60%; 3 objects)
- Eyes.blue & Hair.dark → Class.– (C = 40%; 2 objects)

In both cases the system is accurate and complete and rules do not cover the same objects.

3. Normal system (of rank 2) would have no rules because each rule (of rank 2) has at least one factor with zero contribution to accuracy (recall that normal rules consist of positive factors only): Hair.dark is accurate alone, without Eyes.blue or Eyes.brown, also Eyes.brown is accurate without Hair.blond or Hair.dark.
4. Canonical system (of order 2) *S4* would have two rules of rank 1:
   - Eyes.brown → Class.– (C = 60%; 3 objects)
   - Hair.dark → Class.– (C = 60%; 3 objects)

These rules cover all objects belonging to Class '–'. One object (having Eyes.brown and Hair.dark) is covered twice, there from the +20% comes. Canonical system is non-additive and it allows to cover objects more than once.

2.7.4 The main task

Systems of rules are used for explanation and forecasting.

Initial data matrix is given and some feature *y*. *The goal is to find maximally accurate and maximally complete system of rules* (to explain or forecast feature *y*).

To solve the main task, the following operations are applied to the rules in the form $z_1z_2...z_r \rightarrow y$, where z_1, z_2, ..., z_r are factors:

1. Removing negative factors
   => Accuracy increases
   => Completeness increases or does not change
2. Removing non-existing factors
   => Accuracy does not change
   => Completeness increases or does not change
3. Substitution of factors
   => Accuracy increases, decreases or does not change
   => Completeness increases, decreases or does not change
4. Addition of new factors
   => Accuracy increases, decreases or does not change
   => Completeness increases, decreases or does not change

Removing zero or negative factors improves or does not make worse the solution. Therefore, the normal rules (that contain only positive factors) are essential. Usually the canonical system of rules is found to solve the main task.

It often holds that the bigger is accuracy the smaller is completeness. Thus, the system of rules has to be as accurate and complete as possible. Accuracy, completeness and statistical significance of a system depend on accuracy, completeness and statistical significance of the rules it consists of. The more accurate, complete and capacious is each rule, the greater is the chance to get an acceptable solution of the main task.

According to (Chesnokov, 1982) the main task is to find in given context all determinations from given variable (attribute) to another given variable (class attribute) that *have at least given minimal allowable accuracy and minimal allowable completeness*.

2.7.5 Basic tasks of DA

The basic tasks of DA represent the possible research questions.

Task 1. Obtaining explanations

Some characteristic is given. We call it explainable i.e. liable to explanation. What (kind of) people and in what conditions possess them? Give a description of these people and conditions i.e. specify the characteristics[27] (called explaining) that explain the originally given characteristic[28].

Task 2. Obtaining specifications

There is an attribute. Can its values make accurate the explanation obtained as a solution of task 1? If yes, then specify the sought values of the attribute and how they make accurate that solution.

Task 3. Obtaining complements

There is an attribute. Can its values complement the explanation obtained as a solution of task 1? If yes, then specify the sought values of the attribute and how they complement that solution.

Task 4. Essentiality of context

Let's say, a solution of task 1 is obtained in some context. Is it essential? Specify the extent of essentiality[29].

Task 5. Essentiality of explaining characteristics

Let's say, there is a solution of task 1. To what extent are essential the explaining characteristics contained in it, that belong to the description of people and conditions, giving itself a solution? Specify their essentiality.

[27] factors

[28] class

[29] The extent of essentiality is measured by the increment of accuracy (caused by context) i.e. contribution to accuracy.

Task 6. Essentiality of explainable characteristics

Let's say, there is a solution of task 1. To what extent are essential the characteristics contained in it, that in combination form an explainable characteristic? Specify their essentiality.

Task 7. Formation of explaining typology

Let there is initially a set of characteristics, each of them separately explains the same characteristic *b*, may be not very completely, but accurately enough. It is required to build a generalizing typological characteristic that would generalize all explaining characteristics from the initial set and would give a sufficiently accurate explanation of characteristic *b*, but herewith the completeness of this explanation being definitely higher than of separate explanations of characteristic *b* by characteristics from the mentioned set.

Task 8. Formation of explainable typology

Let there is initially a set of characteristics, each of them can be explained, let not very accurately, but completely enough, through the same characteristic *a*. It is required to build a generalizing typological characteristic, that would generalize all explainable characteristics from the initial set and herewith, it itself was explained by characteristic *a*, not only completely enough, but definitely more accurately than each of separate characteristics from the mentioned set.

Task 9. Verification of explanatory possibilities of typology

Let be given some typological characteristic, acting as a meaningful typological generalization of number of simpler (less general) characteristics. It is required to determine to what extent it is essential in explaining of some third characteristic.

Task 10. Verification of explanability of typology

Let be given some typological characteristic, acting as a meaningful typological generalization of number of simpler (less general) characteristics. It is required to determine to what extent it can be explained by number of third characteristics.

These tasks are solved by finding required determinations (the main task), specifying contributions of their components and combining variables into typologies.

2.7.6 Three principles

DA emanates from three principles: nominality, concreteness and bounded statisticality.

The principle of nominality states that the qualitative/nominal measurements are fundamental in exploring the social phenomena and processes.

The principle of concreteness provides that relations between sociological variables (indicators) should be measured as relations between separate concrete values of these variables, not between variables in general.

The principle of bounded statisticality stipulates that statisticality in sociological regularities manifests itself only as a violation of determinism limited in its scale.

Among the mathematical methods, used for analysis of sociological data, only DA consistently meets all requirements arising from the principles of nominality, concreteness and bounded statisticality. Any other method possessing these properties should simply coincide with DA. (Chesnokov, 1980a, p. 52)

2.7.7 Place of DA

In our opinion the task of DA is a subtask of machine learning.

The task of supervised inductive learning is to find (minimal) set of classification rules (this set is called description) that cover all learning examples (i.e. objects) without contradictions (Gams & Lavrac, 1987). The description is consistent if each object is covered by the rule(s) of only one class (consistency condition). The description is complete if all objects are covered at least by one rule (completeness condition).

The task of DA is to cover only one class (Y) by non-contradictory rules. Thus it corresponds to single-concept learning (in ML).

DA gives also possibility to loosen the consistency condition and to find rules that are not maximally accurate (i.e. hold with some probability under 100%) and thus allow contradictions[30].

DA can be related with association rule mining (ARM) as well. Next the relations with terminology used in ARM (see 2.4.2) are shown.

The confidence of a rule $X \rightarrow Y$ is a percentage of transactions (records, objects) containing both X and Y among the ones containing X, i.e. it is the same as accuracy of determination (see 2.7.1 for definition).

The support of a rule is a percentage of transactions (objects) containing both X and Y against the number of transactions in the whole database. In DA the completeness of determination is a percentage of objects containing both X and Y against the number of objects (transactions) containing Y. Therefore we can say that completeness of determination (in DA) corresponds to the support of a rule

[30] This is the situation where the chosen set of attributes does not determine object's affiliation uniquely, the same combination of factors (i.e. attributes with certain values) leads to different classes; for example 2/3 of cases belong to class1 and 1/3 belong to class2.

in class *Y*. Rules' support against whole database is not calculated in DA as the task is to find only rules of class *Y*.

In ML the classification rules are found for predictive purposes, while in DA the rules have to describe the class. The predictive power of found rules is not evaluated in DA. ARM serves the descriptive purpose like DA, but usually the consequent of the rule is not determined in advance. Being descriptive and looking for the rules with determined consequent (i.e. class) at the same time, DA could be put under supervised descriptive rule discovery (see 2.5).

Similarly to our finding that DA is a subtask of ML (corresponding to single-concept learning), the methods of supervised descriptive rule discovery can be seen as special cases of a more general rule learning task (Novak, Lavrač, & Webb, 2009). Among them subgroup discovery (SD) is the closest to DA by its purpose – to describe one certain class (whereas two other methods compare two classes).

Comparing DA and SD, the difference is that in SD only the most interesting (the most unusual) rules are searched for. Thus the obtained set of rules might not be complete while DA tries to completely cover the target class. Besides confidence (=accuracy) and support (comparable to completeness) SD algorithms use different quality measures to evaluate rule's interestingness, DA does not use such measures.

Thus, DA is different from the methods belonging to supervised descriptive rule discovery (by Novak, Lavrač and Webb (2009)), but literally it performs supervised descriptive rule discovery.

According to Bringmann, Nijssen and Zimmermann (2009), SD and similar approaches can be used for classification as well, the same is true for DA.

3 DEVELOPMENTS

In this chapter we will present our developments of MONSA and DA, in subchapters 3.1 and 3.2, accordingly. Each of them is presented in a nearly chronological order. Putting both methods into a common order was confusing in my opinion. Developments of both methods will be combined in zero-factor-free (ZFF) DA that is presented at the end of the DA subchapter.

In 3.3 we present a framework gathering descriptive tasks solvable by GH and DA and show how these possibilities are covered by our algorithms.

Finally in 3.4 possible further developments of ZFF DA will be presented.

3.1 Developments of Generator of hypotheses

In this subchapter our developments of Generator of hypotheses (GH) and its underlying algorithm MONSA will be presented.

We will start by introducing Generator of hypotheses in 3.1.1. Actually, this method itself (and its base algorithm MONSA) existed already before I began (see 1.2.2), but it has to be introduced before presenting its developments. Next, in 3.1.2 we will show the correspondence between concepts used in MONSA and the ones of closed set mining. In 3.1.3, we will relate the associations found by GH with association rules (introduced in 2.4.2). 3.1.4 describes the differences of MONSA's work depending on the selection criterion of the next node (either by maximal frequency or by minimal frequency). 3.1.5 deals with items between closed set and its generator. After that an algorithm for finding equivalence classes is presented (in 3.1.6). In 3.1.7 we define "excluded factors" and show how to integrate finding these factors into the algorithm for finding equivalence classes. Finally, in 3.1.8, we will show how to integrate classes into MONSA.

Most of these developments are not ultimate goals themselves, but steps towards ZFF DA that will be presented in 3.2.10.

The contents of 3.1.1 can be found in (Lind & Kuusik, 2012). Material in 3.1.2 is published in (Kuusik & Lind, Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques, 2008) – see Appendix A.

Our algorithm for finding equivalence classes in 3.1.6 is not published, except in master's thesis of Meelis Pruks (2014) where it is compared with DPMiner (Li, Liu, & Wong, 2007), with further purpose to use found equivalence classes for EC-based clustering algorithm ECCC (Liu, Wang, Deng, & Dong, 2011).

Material in 3.1.5 and 3.1.8 is published indirectly as the building blocks of zero factor free DA (that will be introduced in 3.2.10) in (Lind & Kuusik, Algorithm for Finding Zero Factor Free Rules, 2016) – see Appendix F.

The contents of 3.1.3, 3.1.4 and 3.1.7 is not published.

3.1.1 Generator of hypotheses

Generator of hypotheses is a DM method that uses the possibilities offered by algorithm MONSA (described in 2.6.1). It solves the task of hierarchical clustering. The goal is to describe the source data. Used evaluation criteria are deterministic (not probabilistic). The association rules it produces are represented as trees, which are easy to comprehend and interpret.

Hypothesis means here a presumable association found by the generator. The system generates associations that meet the criteria given by user. The final decision about their significance makes the user (researcher).

Source data is given as a table of object-attribute type. The result is represented as a hierarchical grouping tree or as a set of intersections.

Below an example of (a fragment of) a tree formed by GH is given. Used data are shown in Table 2.7 (Quinlan, 1984) (p. 55).

```
(3)         0.667(2)     0.500(1)
Height.tall=>Hair  .dark->Eyes  .blue
                         0.500(1)
                         ->Eyes  .brown
            0.667(2)      0.500(1)
            =>Eyes  .brown->Hair  .blond

(3)         0.667(2)     0.500(1)
Hair  .dark=>Eyes  .blue->Height.short
            0.333(1)
            =>Eyes  .brown


(3)          0.667(2)      0.500(1)
Eyes  .brown=>Hair  .blond->Height.short
```

The trees are represented from left to right. This example consists of three trees, it has three root nodes (on the left). Symbols "=>"[31] separate the root nodes and non-root nodes of a tree.

Usually a node contains one element (attribute.value). A node can consist of more than one attribute-value pairs, then "&" is used to connect them. There are no such nodes in given example.

The numbers above node show node's absolute frequency (in parentheses) and node's relative (to the previous level) frequency (before parentheses).

[31] Here "=>" and "->" have no different meaning (regarding exactness) as they have in case of non-redundant association rules by Bastide et al (see 2.4.2.1, p. 36)

The *absolute frequency* of node *t* shows how many objects have a certain attribute with a certain value (among objects having properties (i.e. certain attributes with certain values) of all previous levels *t*-1,…,1). The *relative frequency* is a ratio *A*/*B*, where *A* is the absolute frequency of node *t* and *B* is the absolute frequency of node *t*-1. For the first level the relative frequency is not calculated.

For example we can translate the first tree (`Height.tall=>`) of the set of trees as "3 persons (objects/ examples) are tall, 67% of them have dark hair, and of those (with `Height.tall` and `Hair.dark`) 50% have blue eyes and 50% have brown eyes. Also, 67% of tall persons have brown eyes and 50% of those have blond hair."

Horizontally the nodes are connected by logical AND and vertically by logical OR. The first tree can be represented as follows:

```
Height.tall AND ((Hair.dark AND (Eyes.blue OR
Eyes.brown)) OR (Eyes.brown AND Hair.blond))
```

All trees in the result can be ORed as well.

The same result (the fragment consisting of 3 trees) in the form of intersections is shown below.

```
Height.tall=3
Height.tall&Hair  .dark=2
Height.tall&Hair  .dark&Eyes  .blue=1
Height.tall&Hair  .dark&Eyes  .brown=1
Height.tall&Eyes  .brown=2
Height.tall&Eyes  .brown&Hair  .blond=1
Hair  .dark=3
Hair  .dark&Eyes  .blue=2
Hair  .dark&Eyes  .blue&Height.short=1
Hair  .dark&Eyes  .brown=1
Eyes  .brown=3
Eyes  .brown&Hair  .blond=2
Eyes  .brown&Hair  .blond&Height.short=1
```

In this representation form after "=" the absolute frequency is listed. The relative frequency is not shown.

GH has the following properties:

- GH enables larger set of discrete values (not only binary);
- GH enables to use several pruning techniques;
- The result is presented in the form of trees;
- GH enables to treat large datasets;
- GH enables sampling.

3.1.2 Correspondence between concepts of MONSA and ARM

We had a true guess that intersections found by algorithm MONSA are closed sets. It was important to show this correspondence in order to be comprehensible for the researchers familiar with closed sets. It made findings about closed sets and related concepts usable for us as well.

Here we present denotations and definitions used for MONSA (described in 2.6.1) as in (Kuusik, 1993) and show how these notions and concepts relate to the ones used for frequent itemsets (see 2.4.1).

(1) X - a set $X = \{Xi\}$, $i = 1,2,...,N$,

where each object Xi is a conjunction of M attribute values: $Xi = \mathop{\&}\limits_{j=1}^{M} h_j$.

X is a set of N objects (records) that are described by M attributes. Set X has not to be a binary dataset, every attribute j can acquire integer values in the interval $h_j = 0,1,2,...,K_j\text{-}1$. X can be a transaction database also.

(2) H - a value combination (VC) of certain attributes $H = \mathop{\&}\limits_{q \in D} h_q$, $D = \{j_e\}$, $e = 1,...,E_H$ (the number of elements h_q in H), $1 \leq E_H \leq M$, $1 \leq j_e \leq M$, $j_f, j_t \in D$, $j_f \# j_t, f \# t$, $H \subseteq Xi$.

A value combination can contain 1 to M attributes (with certain values), each only once (i.e. only with one value); it is a subset of some object or is a whole object.

(3) Each value combination H defines on the set X a subset of objects $X_H = \{Xp\}$, $p = 1,2,...,N_H$, $1 \leq N_H \leq N$,

$\{Xp\}$ are all objects $Xi \in X$ that contain H: $X_H = \{Xi \in X \mid Xi \supseteq H\}$.

The subset of objects defined by the value combination is similar to the tidset that corresponds to some itemset ($t(X)$).

(4) Each value combination H defines on set X a subset of elements $X^H \subseteq X$: $X^H = \{Xij \in X \mid Xij \in H, i = 1,2,...,N, j = 1,2,...,M\}$.

An attribute with certain value is called *element*. 'Element' corresponds to 'item'. The difference is that each attribute produces as many elements as many different values it has. Definition (4) says that a 'value combination' is the same as 'itemset' (with extension that values need not be binary).

(5) *Intersection* over a set $Y = \{Yt\}$, $t = 1,2,...,T$, $Yt = \&h_j$ is a set of such elements h_q which belong simultaneously to all Yt: $\cap Y = \bigcap\limits_{t=1}^{T} Yt = \mathop{\&}\limits_{q} h_q = H$.

In Y for H there exists always a corresponding subset of objects $Y_H = \{Yp\}$, $p = 1,...,N_H$, $N_H \leq N$.

If $N = 1$, then the intersection over Y is an object itself.

If there exist no objects $Yt \in Y$ for which $H \subset Yt$, then $Y_H = \varnothing$.

Definition (5) says that an intersection over a set of objects is a set of common elements, this is the same as itemset that corresponds to some tidset (*i(Y)*).

(6) *Elementary conjunction (EC)* on X_H is such an intersection over the set X_H, where $\cap X_H = A(\supseteq H)$, $X_A = X_H$, $N_H \leq N$.

In the case of $A \supset H$, $X_A = X_H$, A is an EC.

We have a set of elements H and its corresponding set of objects X_H (i.e. *t(H)*) and find an intersection over it: $\cap X_H$ (i.e. *i(t(H))*). The operation *i(t(H))* means finding the closure of H. Therefore the resultant set A (of elements) called elementary conjunction is a closed set. From the viewpoint of the algorithm it is essential to find a technique that guarantees the finding of such subsets X_H only for which $\cap X_H = A(\supseteq H)$ (i.e. finding of closed sets only).

(7) *Maximal EC* on X_H is such an intersection over X_H in case of which for a VC $H = \underset{q}{\&} h_q$ is a valid relation

$$\cap X_H = A(= \underset{e}{\&} h_e) \supset H,\ 1 \leq q < e \leq M,\ X_A = X_H.$$

By definition, VC H is EC if $\cap X_H = H$.

H is a maximal EC if it is EC and contains at least one VC $Ht \subset H$ such, that $|X_{Ht}| = |X_H|$ on X. That means that the set of objects $X_H \subseteq X$ is defined unique.

Definition (7) says that maximal EC is EC that has at least one (non-closed) subset with the same frequency (support) i.e. we can remove at least one element without changing in frequency. Our maximal elementary conjunction here is not the same as maximal (closed) set[32].

Additionally, our '(absolute) frequency' is the same as 'support' and 'relative frequency' corresponds to 'confidence'[33] of a rule.

Thus, algorithm MONSA for finding all intersections finds all closed sets. Support threshold (i.e. minimal allowed frequency) can be applied. The confidence measure (relative frequency) is not anti-monotone and downward closed and therefore cannot be used for pruning similarly to *Apriori* principle (mentioned in 2.4.1).

[32] "A frequent itemset X is called maximal if it is not a subset of any other frequent itemset." (Zaki & Hsiao, 2002)
[33] Defined in 2.4.2

3.1.3 Associations found by GH

Having found a set of intersections (the result of MONSA) we are interested in presenting this result possibly understandably. This is the reason for presenting the result in an alternative form – as a set of trees. The tree is presented exactly in the same order (of nodes) as it is traversed, thus it is possible to output it immediately during the work. This way there is no need for additional data structures for storing the (tree-form) result before outputting.

In the following we will explain which ARs are contained in the trees and whether they can be related to the representation forms presented in 2.4.2.1.

An example of associations found by GH is given in 3.1.1. The last branch of it

```
(3)         0.667(2)      0.500(1)
Eyes  .brown=>Hair  .blond->Height.short
```

contains three nodes: Eyes.brown(3), Eyes.brown&Hair.blond(2), Eyes.brown&Hair.blond&Height.short(1). The value combination in the current node is composed from the elements starting from the root until the current node. Each node in HG tree represents a closed set. Next node in the tree is a closed superset of the previous one. Choosing the next node by the maximal frequency guarantees that the current node and the next node are adjacent closed sets. (This is not guaranteed when an arbitrary node is chosen to be the next one.)

A closed set can have (usually has) more than one adjacent subsets. In case of a tree (where each closed set is presented only once) there is only one path to each closed set, thus not all possible associations between closed sets are represented.

Given fragment presents two approximate association rules:

- Eyes.brown $\xrightarrow{3,2/3}$ Hair.blond;
- Eyes.brown&Hair.blond $\xrightarrow{2,1/2}$ Height.short.

The left side (antecedent) is always a closed set (because every node represents a closed set) and the right side (consequent) is its closed superset diminished by the elements of the current closed itemset. We can say that these rules have maximal antecedent and maximal consequent. This representation form does not coincide with either of the ones presented in 2.4.2.1. Furthermore, it does not pretend to be a minimal generating set for approximate association rules.

The presented rules are directed from subset to superset. If we wanted to get the rules in the opposite direction i.e. exact (confidence=1) rules from superset to subset, we would find rules where antecedent and consequent were not disjoint, for example:

- Eyes.brown&Hair.blond $\overset{2}{\rightarrow}$ Eyes.brown;
- Eyes.brown&Hair.blond&Height.short $\overset{1}{\rightarrow}$ Eyes.brown&Hair.blond.

In these rules the right side is not diminished by the elements of left side. If we did that, the consequent would be empty, because the elements of the right side are contained in the left side. If we tried to shorten the left side it was not maximal (in the same equivalence class) anymore. Moreover, (in the current branch) we do not have information about the support of such left side:

- Hair.blond $\xrightarrow{?,2/?}$ Eyes.brown (because σ(Hair.blond)=?);
- Height.short $\xrightarrow{?,1/?}$ Eyes.brown&Hair.blond (because σ(Height.short)=?).

In short, GH finds approximate association rules with maximal antecedent and maximal consequent between adjacent closed sets (according to given frequency threshold). As the result has the form of tree, it is a subset of all such associations.

3.1.4 Comparison of two criteria for selecting next node

By default MONSA selects the next node by maximal frequency. Is it the only possible criterion for selecting the next node? What happens if we choose by minimal frequency? Will we get the same result? Does it change the amount of work?

For experimenting with minimal frequency (as the selection criterion) only a small change in the program is needed – instead of maximal frequency the minimal frequency is selected from the current frequency table (at line 5 in algorithm MONSA – see p. 45). Bit more changes are needed to make this choice dependent on user's input. In the following we will discuss the differences between working by maximal and by minimal frequency.

In Appendix G both forms of output – intersections and trees – got selecting next node by maximal frequency and by minimal frequency, are listed side-by-side.

As we can see, MONSA finds exactly the same intersections in both cases, just in a different order. The order of elements in an intersection can be different as well. Actually, an arbitrary (suitable) element can be chosen as a next node from FT. The system retains its monotonic nature and finds the same intersections.

When the order of nodes is different, then the tree, got by associating the nodes in the order they are found, is also different.

The overall shape of the trees is different. The first one (by maximal frequency) has many branches with maximal possible depth (12 branches with depth 4 in given case) and some branches consisting of the root only at the end (there are 4 such branches). The second tree (by minimal frequency) is more balanced. In our case it has no branches with maximal depth (=4) and only one branch with depth 1 (i.e. root only). The number of branches is bigger in the latter case (38 vs 35).

It is explained by reaching quickly to the intersections with the lowest frequency that contain all possible attributes (in case of selecting by minimal frequency) while in case of maximal frequency these nodes are reached step-by-step. If we prefer the closed superset with maximal possible frequency then there cannot be any closed set between the current and the chosen one, because there is no superset with frequency smaller than the frequency of the current set and bigger than the frequency of the chosen superset. Therefore always an adjacent closed superset is chosen by maximal frequency. Choosing by minimal frequency we can skip closed sets between the current and the chosen one, thus adjacency is not guaranteed in such case.

In a tree, each node is represented by the element(s) that are not contained in its parent node. Nodes with more than one element contain "&"-signs. The bigger differences between parents and children the more "&"-signs in the tree. Adjacent closed sets can differ by one or more elements, non-adjacent ones differ at least by two elements. Therefore, selecting a next node by minimal frequency, thus preferring non-adjacent supersets, there are more bigger differences and more "&"-signs in the tree (compared to choosing by maximal frequency): 14 "&"-signs in 10 nodes in case of minimal frequency and only 2 nodes and 2 signs in case of maximal frequency. The last two cases are those where adjacent closed sets differ by two elements. Choosing by minimal frequency there are also two such cases (out of 10). If the current and the next node differ by more than one element (independent whether they are adjacent or non-adjacent closed sets), then all those elements are included into the same intersection (node) at a time (line 12 in algorithm MONSA, p. 45).

Our tree got by minimal frequency has one more root node than the one by maximal frequency, namely T2.1&T3.2=4. It happens when some node of initial level (here T2.1 with frequency 4) is fully "subsumed" by another which has a bigger frequency (T3.2=7) i.e. all objects containing that first element (T2.1) contain the other (T3.2) as well (while the set of objects having only the other element is bigger). Preferring bigger frequency such element with smaller frequency is set to zero in the FT before we could select it as a root. Tree by maximal frequency contains T3.2=> T2.1. Both trees contain a root T3.2=7. Described situation is also an occurrence of skipping adjacent superset (of the initial empty set).

Taking into account that in case of depth-first search only current branch is held in the memory, the tree with a smaller depth can be preferable (if the amount of data at each level is big).

For each found intersection an extract of data has been made and the corresponding FT has been found by algorithm. (Extract has not to be "physical" subtable, just indexes of objects/rows can be kept.) Besides these extracts that have given the intersections also some repetitious extracts have been made, for which output was not generated (when the intersection is not unique – at line 10

(MONSA, p. 45)). So, the overall number of extracts made is bigger than the number of found intersections. Comparing algorithm's work by maximal and by minimal frequency, the numbers of "unsuccessfull" extracts are different. In our case 18 by maximal and 10 by minimal frequency. Only 4 of those extracts coincide. (Information about these extracts is taken from log files.) Again, selecting next node by minimal frequency, is less labor-consuming.

This is from the technical aspect. The user's viewpoint might be different as different trees present different subset of (set of all possible) associations. Probably the user expects the associations rather between adjacent closed sets than between non-adjacent sets.

A statement (from the previous subchapter) that GH finds approximate association rules between adjacent closed sets is valid for the case when next node is selected by maximal frequency.

3.1.5 Elements between closed set and its generator

Each intersection (found by MONSA) determines some set of objects and consists of all common elements of those objects. Some elements (of an intersection) are more "considerable" than the others, they determine the set (of objects), while the others just come with them. For example, if the set of people is described as "lives in the country" and "has a cow" then quite probably the same set of people can be determined by "has a cow" only and "living in the country" can be concluded from "having a cow". Such a conclusion is an (exact) association rule (see p. 36): "has a cow" $\Rightarrow$ "lives in the country".

We will associate these different subsets of intersection with notions of FIM and show how (original) MONSA operates with these kinds of elements.

Additionally, we will bring out what changes are needed to find both subsets for each found intersection.

As we have shown in 3.1.2 algorithm MONSA for finding all intersections finds all closed sets (intersection = closed set).

Having found an intersection, a next leading element is chosen. Adding it to the intersection we get a new VC (itemset) with a smaller frequency. For completing the next intersection (closed set) we add all such elements that do not cause a change (decrease) of frequency of the VC. In FIM notion these are the items between closed set and its generator: $c(g)\backslash g$. There is no special name for these elements/items (neither in the original description of MONSA (see 3.1.2) nor in closed set mining). Inspired by DA, we call them "zero factors". Later (in 3.2.9) we will show how these elements are related to the zero factors of DA.

Each (found) intersection contains all relevant zero factors. Each time we add a new leading element (item) into the current intersection (closed set), we get a generator for a next closed set. The lastly added (leading) element is certainly not a zero factor in this new VC/itemset (because it causes a decrease of frequency), but the current closed set can contain zero factors (included at previous levels). Thus, generally this new generator is not a minimal generator, it is just a generator for a new closed set.

If no element of the new extract has a frequency equal to the leading one, then there are no zero factors and this generator itself is a new closed set. If there exist elements with frequency equal to the leading frequency, then these are the zero factors; we add them to the generator and get the corresponding closed set.

An intersection (closed set) can have more than one minimal generators (see 2.4.1.1). MONSA finds each intersection only once, thus it can find only one generator for each intersection, and usually the found generator is not a minimal one. If we want to find them all, then we need to reach each CS as many times as many minimal generators it has. It is easier to find one closed set for each generator than unknown number of generators for each closed set. Moreover, MONSA already contains means for completing a closed set for an arbitrary generator (independent whether it is minimal or not). Thus, in order to find all minimal generators (together with their corresponding closed sets), the program should move from one minimal generator to another instead of original moving from a closet set to a closet set.

MONSA has means to make sure whether the just found intersection (actually, the set of objects it determines) is really new ("unique") or has been extracted already. This information is used to block repetitious intersections. In order to enable repeated finding of already found intersection by another minimal generator we have to remove such blocking, but information about ”uniqueness” ("newness") is still available.

In order to distinguish between zero factors and non-zero factors found at previous levels we should store them separately at each node.

Described approach (generator + zero-factors = closed set) and algorithmic changes will be used in the algorithm for finding equivalence classes (presented in the next subchapter).

3.1.6 Algorithm for finding equivalence classes

The creation of an algorithm for finding equivalence classes was not an end in itself, the idea to create it arose when we thought out how to distinguish between zero factors and generator that together form a closed set (see 3.1.5).

For that purpose the following changes were made to original MONSA: 1) keeping generator separately from closed set; 2) moving from smaller (shorter) generators to bigger (longer) ones (while MONSA moves from smaller closed sets to bigger closed sets); 3) using information about "uniqueness" for organizing ECs (not for blocking repetitious output of already found CSs).

ECs are used to construct association rules with minimal antecedent and maximal consequent: $g \rightarrow c(g) \setminus g$ (see 2.4.2.1), whereas the concept of "equivalence class" itself is not always mentioned. On top of (non-redundant) association rule mining CFD (conditional functional dependency) discovery has been developed. CFDMiner (Fan, Geerts, Li, & Xiong, 2011) uses an EC-based algorithm by Li, Liu and Wong (2007). CFDs are used as rules for data cleaning and data integration.

ECs have found use in building understandable classifiers based on "emerging patterns"[34] that describe significant changes (differences or trends) between two classes of data. Emerging patterns, in turn, are used in many application fields. For example, an algorithm that exploits ECs for finding "delta-discriminative" emerging patterns (Li, Liu, & Wong, 2007), is used for human activity recognition (Gu, Wu, Tao, Pung, & Lu, 2009).

Here we present an algorithm for finding all (frequent) equivalence classes. This algorithm does not construct any kind of rules (based on ECs).

As an equivalence class can be uniquely determined and concisely represented by a closed pattern and a set of generators (Li, Liu, & Wong, 2007), we have to find the (only) closed set and all minimal generators for each existing equivalence class and the frequency (that is equal for all itemsets in the same equivalence class).

Our MS based algorithm for finding all equivalence classes according to given frequency threshold is grown from the algorithm MONSA for finding all intersections (i.e. closed sets) described in 2.6.1. It uses the same technique of making subsequent extracts by the aid of frequency tables. (FT shows for each attribute the frequencies of all its possible values (in the set of objects for which it is found).)

Each generator is found only once. In order to avoid repeatedly finding already found generators, the frequency of the selected element (the "leading" element) is set to zero in the current frequency table. Before selecting the next leading element, those zeroes are "brought down" from the frequency table of the previous level to the current level (except for the initial level).

The next element to be included into the generator is selected by the frequency (from the frequency table). Its frequency has to be bigger than or equal to the given frequency threshold and smaller than the frequency of the current extract.

[34] "Emerging patterns are defined as itemsets whose supports increase significantly from one dataset to another", more specifically, itemsets whose growth rates (support ratio) are larger than a given threshold (Dong & Li, 1999). A survey of emerging patterns for supervised classification can be found in (García-Borroto, Martínez-Trinidad, & Carrasco-Ochoa, 2014).

The latter condition prevents selecting elements that belong to the closure of the current generator.

In order to find minimal generators only (not the ones between a minimal generator and closed set), the minimal one of suitable frequencies is chosen (in contrast with MONSA where the next subset is selected by maximal frequency). If there is more than one element with such frequency, just one of them is selected. The chosen element together with the previously selected elements of the same branch forms a generator and determines a narrower (than the current) set of objects.

At each step it is easy to find a closure of the current generator. All elements in FT that have a frequency equal to the frequency of current generator, belong to the closed set. (That is why those elements are not chosen for forming the next generator.) If there are no such elements, the generator and its closed set coincide.

As there can be more than one generators in EC, we need to know whether the current generator belongs to any already found equivalence class or we have found a new one. This is also easy to make sure (without looking through the already found results): if any of the elements between the generator and its closure (i.e. *c*(*generator*)\generator) has been set to zero in the FT of the previous level, then the set of objects (covered by the current generator) has been extracted already (by another generator of current EC).

The following notation is used in pseudocode of the algorithm:

X_0 – initial data table (objects*attributes);

`t` – number of the step (or level) of the recursion;

X_t – set of objects (extract) at level `t`;

FT_t – frequency table for a set X_t;

gen_t – generator at level `t`;

`CS` – closed set (closure of gen_t);

`CS_is_new` – the truth-value of whether the closed set (and corresponding equivalence class) is new;

`V` – „leading" frequency i.e. frequency of extract;

`minfr` – frequency threshold (minimal allowed number of covered objects);

`Init` – activity for initial evaluation;

Elements are given as $\text{value}_{\text{attribute}}$;

Assignments are indicated by "←" ("=" is for comparison).

Equivalence class (EC) is described by closed set (CS), all its minimal generators and frequency.

The pseudocode of the algorithm is given below.

```
Algorithm for finding all equivalence classes
Given: $X_0$, minfr>0
1: Init
2: t←0 ; $gen_0$←{}
3: find $FT_0$
4: FOR EACH element $h_f \in FT_0$ with frequency V=min $FT_0[h_f] \geq$
   minfr DO
5:     make_extract(t+1; $h_f$; V)
6:     $FT_0[h_f]$←0
    NEXT
7:  output all ECs
End of Algorithm
PROCEDURE make_extract(t; $h_f$; V)
8: $gen_t \leftarrow gen_{t-1} \cup h_f$
9: CS ← $gen_t$
10:CS_is_new ← true
11:IF V=1 THEN
12:    find object obj with $h_f$ from $X_{t-1}$
13:    FOR EACH empty position p in CS DO
14:        value ← $X_{t-1}$[obj; p]
15:        CS ← CS ∪ $value_p$   /* CS[p] ← value
16:        IF $FT_{t-1}[value_p]$=0 THEN
17:            CS_is_new ← false; EXIT FOR-cycle
           ENDIF
       NEXT
18:    IF CS_is_new THEN new_EC(V; CS; $gen_t$) ELSE
       add_gen(V; $gen_t$; $h_f$)
19:ELSE
20:    separate submatrix $X_t \subset X_{t-1}$ such that $X_t$={$Xij \in X_{t-1}$ |
       X.f=$h_f$}
21:    find $FT_t$
22:    FOR EACH empty position p in CS DO
23:        IF exists value(element) $h_p$ such that $FT_t[h_p]$= V
           THEN
24:            CS←CS∪$h_p$
25:            IF $FT_{t-1}[h_p]$=0 THEN
26:                CS_is_new ← false; EXIT FOR-cycle
               ENDIF
           ENDIF
       NEXT
```

```
27:     IF CS_is_new THEN new_EC(V; CS; $gen_t$) ELSE
        add_gen(V; $gen_t$ ; $h_f$)
28:     IF V>minfr THEN
29:         ZeroesDown(t)
30:         $FT_t[h_f] \leftarrow 0$
31:         FOR EACH $h_u \in FT_t$ with frequency V2=min
            $FT_t[h_u] \geq$ minfr and V2<V DO
32:             make_extract(t+1; $h_u$; V2)
33:             $FT_t[h_u] \leftarrow 0$
            NEXT
        ENDIF
    ENDIF
END PROCEDURE
PROCEDURE ZeroesDown(t) /* from $FT_{t-1}$ into $FT_t$
34: FOR EACH element $h_u \in FT_t$ with frequency > 0 DO
35:     IF $FT_{t-1}[h_u]=0$ THEN $FT_t[h_u] \leftarrow 0$
    NEXT
END PROCEDURE
PROCEDURE new_EC(freq; CS; gen)
/* instead of parameters current V, CS, $gen_t$ can be used
/* creates a new equivalence class with frequency freq
containing closed set CS and generator gen
END PROCEDURE
PROCEDURE add_gen(freq; gen; $h_f$)
/* instead of parameters current V, $gen_t$, $h_f$ can be used
/* finds the equivalence class with frequency freq to which
the generator gen belongs
/*knowing the lastly added element $h_f$ might improve the
(element-wise) search/check
/* adds a generator gen into the equivalence class
END PROCEDURE
```

The initial data table X_0 and the frequency threshold `minfr` are given. The main program starts with initializing the structure for holding equivalence classes (step 1) and initial assignments for a level of recursion `t` and the empty generator gen_0 (2). Next the frequency table FT_0 for X_0 is found (3). In step 4 each element with a suitable frequency ($\geq$`minfr`) is chosen as a leading element (for inclusion into generator) in ascending order (by frequencies). An extract by the leading element h_f is made (5) and its frequency in the frequency table FT_0 is set to zero (6). Finally, all found equivalence classes are outputted (7).

While the main program makes extracts from initial data, the recursive procedure `make_extract` handles all deeper levels. It starts by evaluating the current generator gen_t (8) and giving initial values for its closure `CS` and truth-value `CS_is_new` for indicating whether the closed set (and consequently current EC) is new (9-10). (Lines 11-19 in grey colour are for special case when the leading frequency `V` is 1 and will be explained further.) Next the subset of objects X_t is extracted by the leading factor h_f (20) and the corresponding frequency table FT_t is found (21). Step 22 goes through all empty positions (attributes without value) in the current closed set `CS` (as a vector) and step 23 searches for the value (of that attribute) with frequency equal to the leading one `V`. If one exists, it belongs to the current closed set and it is included into `CS` (24). Next we check whether the found element of closed set has been set to zero in the frequency table of previous level (25). If it is so, then this closed set (and whole EC) is not a new one and indicator `CS_is_new` is evaluated accordingly, also the search for elements of current CS is exited (26). After finishing the search for elements of CS, according to the value of `CS_is_new`, either a new equivalence class is created (with current generator gen_t, its closure `CS` and frequency `V`) or an existing equivalence class is complemented with the current generator gen_t (27). Additional parameters of `add_gen V` and h_f are intended for improving the search for the right existing EC. Also, the number of elements in gen_t can be used (the number of elements in corresponding CS has to be bigger than in generator).

After creating or complementing an EC, step 28 checks the reasonability of making a subsequent extract. If the frequency `V` is above the threshold `minfr`, then there is a possibility to find frequency that is $<$`V` and $\geq$`minfr`. If that check (in 28) gives a positive result, then the zeroes are "brought down" from the frequency table of the previous level (29). The procedure `ZeroesDown` goes through the current frequency table and for each element with a frequency over zero (34) its frequency at the previous level is checked (35). If the latter is zero, then the element gets a zero frequency at the current level as well (35).

The frequency of the current leading element h_f is set to zero (30).

Step 31 goes through all elements that are suitable for subsequent extract i.e. with frequency smaller than the leading one (in order to prevent selecting element of current closed set) and greater than or equal to the given frequency threshold `minfr`. Again the order is ascending. A recursive call to procedure `make_extract` is made with a new leading element h_u and its frequency `V2` (32) and the frequency of h_u is set to zero (33).

Lines 11-19 describe a special case when the leading frequency `V` is 1. In such case there is no need to make an extract by h_f. If the frequency is 1 then there is exactly one object that belongs to that (potential) extract, so we can locate it without forming an extract (12). Similarly to the general case, each empty position in `CS` is inspected (13) and filled with respective value from that object (14-15). If this element has been set to zero in the FT of previous level (16) then consequently the current CS (and EC) is not new and the cycle for filling empty positions in `CS` is exited (17). At line 18 either a new EC is created or an existing one is complemented with the current generator $\texttt{gen}_t$.

Practically, this part of code is not important, because the frequency(support) threshold is usually >1 (in case of bigger data) and the code at lines 20-32 is suitable for that case as well.

This algorithm is not published, except in the master's thesis of Meelis Pruks (2014) where it is compared with DPMiner, that is claimed to be the first algorithm mining closed sets and generators simultaneously (Li, Liu, & Wong, 2007), with further purpose to use found equivalence classes for EC-based clustering algorithm ECCC (Liu, Wang, Deng, & Dong, 2011). The result of the comparison is: DPMiner is faster, but less accurate since it finds non-redundant[35] δ-discriminative[36] equivalence classes, because the number of non-redundant δ-discriminative ECs is considerably smaller than of all ECs (with equal support threshold). An important difference between the two algorithms is that DPMiner uses transactional database (i.e. binary data) with class labels as an input, while our MS based algorithm supposes an object-attribute data table where each attribute can have more values and pre-classification is not required.

3.1.7 Finding excluded factors

Besides elements that occur together in some set of objects (i.e. form a closed set) we might be interested in elements that do not occur in any of those objects. For example, it might be true that people who "have a cow" (and "live in the country") are not "frequent travellers". In such case we can form a negative association rule (2.4.2.2): "has a cow" [AND "lives in the country"] $\Rightarrow$ NOT "is a frequent traveller". In some cases negative rules are more valuable than positive ones.

We call these non-presented elements "excluded factors", this is our own name (although "factor" comes from DA). Next we will show how to adapt the previously introduced EC algorithm (3.1.6) to find such information.

Excluded factors are such factors that are not presented in any of the objects of the current extract (covered by the current closed set), but do exist in the initial data table. Excluded factors are not sought for from such attributes that belong to the closed set. As each included attribute can have only one certain value in the

[35] Non-redundancy here is different from non-redundancy of association rules (as defined in 2.4.2.1) where redundant and non-redundant rules are found among rules with equal support (and confidence). Here δ-discriminative closed set EC_2 is said to be redundant with respect to δ-discriminative closed set EC_1 if its closed pattern is a superset of the closed pattern of EC_1 (thus their frequencies are different) i.e. its transaction set is fully subsumed by transaction set of EC_1. Thus only the most general (minimal) equivalence classes are non-redundant. (Li, Liu, & Wong, 2007, p. 3)

[36] δ is a small integer number showing how many covered transactions can belong to other classes than the largest one (thus, the classification of data is supposed).

closed set, we can conclude ourselves that all other values of such attribute cannot be presented in that extract. Thus only the values of attributes that are not included in the CS are considered.

Let us have an extract consisting of two objects given in Table 3.1 together its corresponding frequency table and intersection over this extract. We do not look for excluded factors for attributes A1 and A3 as they belong to the closed set (intersection). We look for zeroes in the remaining columns (attributes) of the frequency table. A2.2 (having zero frequency) is an excluded factor if such value for A2 does exist in the initial data table.

Table 3.1. Data table X(2,3) and corresponding frequency table

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	3	2
O2	1	1	2

Value \ Attribute	*A1*	*A2*	*A3*
1	**2**	1	0
2	0	0	**2**
3	0	1	0

Intersection	1	*	2

Our EC algorithm (in 3.1.6, see p. 73) can be easily supplemented with the part for finding excluded factors. At lines 22-26 there is a FOR-cycle passing through all empty positions of the current not-yet-ready closed set `CS`. IF-clause inside this cycle (lines 23-26) checks whether the current empty position `p` can be filled with a value (that is a zero factor in this `CS`). We add an ELSE-part to this IF-clause for the case when there is no zero factor for the current position.

Below we give a suitable fragment of the code (preserving original lines numbers 22-26) with the extension for finding excluded factors (starting at line 27).

```
22:   FOR EACH empty position p in CS DO
23:       IF exists value(element) $h_p$ such that $FT_t[h_p]$= V
          THEN
24:           CS←CS∪$h_p$
25:           IF $FT_{t-1}[h_p]$=0 THEN
26:               CS_is_new ← false; EXIT FOR-cycle
              ENDIF
27:       ELSE
28:           FOR EACH value v (of attribute p) DO
29:               IF $FT_t[v_p]$= 0 THEN
30:                   IF InitialFT[$v_p$]> 0 THEN
31:                       ExclF←ExclF∪$v_p$
                      ENDIF
```

```
                ENDIF
            NEXT
        ENDIF
    NEXT
```

In the (new) ELSE-part we check each element with zero frequency in column `p` (i.e. values of current attribute) (lines 28-29): if it has a non-zero frequency in the initial FT (line 30) then it is an excluded factor and is inserted into the set of excluded factors `ExclF` (line 31). (Otherwise, such a factor does not exist in the data.) `InitialFT` (at line 30) is a copy of the initial state of FT_0. FT_0 itself is not suitable because the frequencies (of selected leading factors) in it are set to zero during the work (line 6 in EC algorithm).

Although this algorithm determines extracts by minimal generator, the excluded factors are found for their corresponding closed set. Attributes that provide zero factors have a certain value in the extract (selected set of objects), thus there is no need to list their remaining values among excluded factors.

Therefore, excluded factors corresponding to some closed set are looked for and stored only once – when this CS is found for the first time and a new EC is created (procedure `new_EC` gets a new parameter: `ExclF`).

Of course, `ExclF` has to be evaluated with an empty set (`ExclF`←`{}`) each time before processing a next generator – that is in the beginning of procedure `make_extract` (lines 8-10 in the EC algorithm at p. 73).

3.1.8 Integrating classes into MONSA

Heretofore MONSA has found descriptions for non-classified data. If data is partitioned into classes then, obviously, we are interested in describing these classes. For that purpose we will integrate detection of a class into the algorithm.

The question how to incorporate classes into MONSA, has been raised before the questions about zero factors (and excluded factors), but it waited for an answer until the detection of zero factors was solved. Then we saw that the solution is simple – the class can be detected the same way as any zero factor. The difference is that a class attribute is never used for making an extract.

Let us have a data table containing a class attribute (Table 3.2). Both objects belong to the same class (Class.1). An intersection over them contains this class value. As we have seen earlier this is easy to detect from the frequency table.

If the class value is a zero factor here (not the one that has been used for extracting the current set of objects) then we can say that the remaining part of the intersection (A1.1&A3.2) is always accompanied by Class.1

Table 3.2. Data table and corresponding frequency table

Object \ Attribute	*A1*	*A2*	*A3*	*Class*
O1	1	2	2	1
O2	1	1	2	1

Value \ Attribute	*A1*	*A2*	*A3*	*Class*
1	**2**	1	0	**2**
2	0	1	**2**	0

Intersection	1	*	2	1

Next we will present a recursive[37] version of MONSA for finding all closed sets with classes. (This algorithm does not detect zero factors and excluded factors.)

The following notation is used in pseudocode of the algorithm:

X_0 – initial data table (objects*attributes);

`attr` – number of attributes (excluding class);

`t` – number of the step (or level) of the recursion;

X_t – set of objects (extract) at level `t`;

FT_t – frequency table for a set X_t;

CS_t – closed set at level `t`;

cl_t – class at level `t`;

`CS_is_new` – the truth-value of whether the closed set is found for the first time;

`V` – „leading“ frequency i.e. frequency of extract;

`minfr` – frequency threshold (minimal allowed number of covered objects);

`Init` – activity for initial evaluation;

Elements are given as $\text{value}_{\text{attribute}}$;

Assignments are indicated by “←” (“=” is for comparison).

The pseudocode of the algorithm is given below.

```
Algorithm for finding all CSs with classes
Given: $X_0$, minfr >0
1: t←0 ; $CS_0$←{} ; $cl_0$←0
2: find $FT_0$
```

[37] There is no need to use recursion while realising the algorithm; backtracking version (like in 2.6.1) is still suitable.

```
3: FOR EACH element $h_f \in FT_0$ with frequency $V \geq minfr$ DO
4:     $FT_0[h_f] \leftarrow 0$
5:     make_extract(t+1; $h_f$; V)
   NEXT
End of Algorithm
PROCEDURE make_extract(t; $h_f$; V)
6: $CS_t \leftarrow CS_{t-1} \cup h_f$
7: $cl_t \leftarrow cl_{t-1}$
8: CS_is_new ← true
9: separate submatrix $X_t \subset X_{t-1}$ such that $X_t=\{Xij \in X_{t-1} \mid X.f=h_f\}$
10: find $FT_t$
11: FOR EACH empty position p ($p \in 1,\ldots,attr$) in CS DO
12:     IF exists value(element) $h_p$ such that $FT_t[h_p]= V$ THEN
13:         $CS_t \leftarrow CS_t \cup h_p$
14:         IF $FT_{t-1}[h_p]=0$ THEN
15:             CS_is_new ← false; EXIT FOR-cycle
            ENDIF
        ENDIF
    NEXT
16: IF CS_is_new THEN
17:     IF $cl_t=0$ THEN
18:         IF exists value clv such that $FT_t[clv_{attr+1}]=V$ THEN $cl_t \leftarrow clv$
        ENDIF
19:     output $CS_t$, $cl_t$
20:     IF V>minfr THEN
21:         ZeroesDown(t)
        ENDIF
22:     BackwardComparison(t)
23:     IF V>minfr THEN
24:         FOR EACH $h_u \in FT_t$ with frequency $V2 \geq minfr$ and V2<V DO
25:             $FT_t[h_u] \leftarrow 0$
26:             make_extract(t+1; $h_u$; V2)
            NEXT
        ENDIF
    ENDIF
END PROCEDURE
PROCEDURE ZeroesDown(t) /* from $FT_{t-1}$ into $FT_t$
27: FOR EACH element $h_u \in FT_t$ with frequency > 0 DO
28:     IF $FT_{t-1}[h_u]=0$ THEN $FT_t[h_u] \leftarrow 0$
    NEXT
END PROCEDURE
```

```
PROCEDURE BackwardComparison(t)
29: FOR EACH element $h_u \in FT_t$ with frequency > 0 DO
30:     IF $FT_t [h_u]=FT_{t-1}[h_u]$ THEN $FT_{t-1} [h_u] \leftarrow 0$
    NEXT
END PROCEDURE
```

The algorithm works similarly to the original MONSA (see 2.6.1). In the given recursive version the main program treats the initial level and the procedure `make_extract` handles all extracts.

The uniqueness check of a closed set (that was before brought out as a separate procedure `CheckUniqueness`) takes place at lines 11-15. This time it is executed before bringing zeroes down (line 21). This change is not principal, it just makes it easier to perform the check.

One more difference is that the criterion of selecting the next node (by maximal frequency in FT) is left out from the pseudocode (at lines 3 and 24). We still use it, but principally it is not so important, because any other order of nodes would give all the same closed sets (just in a different order).

A condition `V≥minfr` (for considering the support threshold) is revealed here (at lines 3 and 24) while in the previous pseudocode it was hidden into the condition ‘pruning is needed’ (line 6).

The condition `V>minfr` (lines 20 and 23) is a prerequisite for a possibility to find a frequency `V2` that is smaller than `V` and bigger than or equal to `minfr` (`V2≥minfr and V2<V` at line 24). `V2` is the frequency of the next leading element (by which an extract will be made). Its frequency `V2` has to be smaller than the frequency of current leading element (and extract) `V` to prevent making an extract by zero factor (the frequencies of zero factors are equal to the leading frequency). Another possibility to prevent it was to set to zero the frequencies of all zero factors (after they have been detected) – earlier we used this option.

A new property – detection of classes – starts by evaluating a class corresponding to the initial level cl_0 by 0 (or some other non-existing value) with a meaning that objects belong to different classes. At each lower level variable cl_t gets a value from the previous level. If objects belong to the same class at some level, then in every further extract (that is a subset of that object set) they still belong to that same class. Therefore, if the value of cl_t is not (initial) 0 then all objects of the current extract belong to this class and there is no need to check whether they belong to the same class. If there is no common class at previous level (line 17) then the algorithm checks the frequencies of class attribute: if one of them is equal to the leading frequency (and others are zeroes) then all objects of the extract belong to that class and cl_t is evaluated accordingly (line 18). As pointed out earlier this check is analogous to detecting “usual” zero factors (taking place at line 12).

The algorithm assumes that a class is given by one class attribute (in the last column of data table). Instead of using only one attribute it was possible to use more of them. In such case the frequency check (whether it is equal to the leading frequency) should be made for each of these attributes and instead of one value (cl_t) a vector of corresponding values should be used. We can consider each different combination (conjunction) of their values as a separate class or to use more sophisticated logic on them.

The given version of the algorithm outputs all closed sets irrespective do they have a class or not. It does not backtrack after finding a class. In case of insertion of such criterion the order of selecting next node by maximal frequency might be important.

3.2 Developments of Determinacy analysis

The theory of Determinacy Analysis (DA) by Chesnokov is introduced in 2.7. In this subchapter we will present our developments of DA. However, first subchapters introduce things that existed before I began.

We will start by giving an overview of original applications of DA (3.2.1), in order to bring out their limitations (3.2.1). In 3.2.2 we will list these and other deficiencies of DA that have given us a reason to work out better solutions.

First, in order to overcome the problem of equal length of found rules, we have proposed so called step-by-step method[38], where the completion of a single rule is stopped whenever it occurs to be accurate (see 3.2.3). Although this approach gives a better result than the original one, it still finds a set of non-intersecting rules that is an important restriction.

Our further algorithms produce non-additive systems of rules where the rules can intersect. Also, the order of attributes can be different in each rule.

Our first algorithm for finding intersecting rules is presented in 3.2.5. This algorithm produces a possibly small set of rules, monitoring and taking into consideration which objects are covered by the found rules already. Like a step-by-step approach this one also produces one system of rules that is not always the best one.

Besides a possibly small number of rules, we are interested in possibly short rules containing no redundant factors. A factor is redundant when we can remove it without losing accuracy of the rule. These redundant factors are zero factors.

[38] This method itself existed before I started, but then it had not been put into the context of systems (sets) of rules (according to (Chesnokov, 2002)) and it was unpublished as well. It is presented in this thesis because it serves as a starting point for further developments of DA.

Occurs that in the moment when a new factor is inserted into a rule we cannot decide about its redundancy or non-redundancy (in reference to the final rule) by its contribution to the accuracy. This problem is explained in 3.2.4 , it is true for step-by-step method as well as other approaches.

Our next algorithm (in 3.2.7) produces all possible shortest rules i.e. non-redundant rules. Additionally it finds some redundant rules that cannot be avoided. These rules have to be removed by compressing the result. The set of rules we get after such compression is the same in case of some other algorithms as well. We call it Determinative Set of Rules (DSR) – see 3.2.6. DSR gives us the basis for finding different covers – suitable subsets of found rules that cover all objects.

Delving deeper into the problem of zero factors we have found that there are two types of them – see 3.2.8. In 3.2.9 will we show how zero factors and DA rules are related to closed sets and generators. These associations together with developments of MONSA (3.1) have been involved in our final development of DA – Zero Factor Free DA – that will be presented in 3.2.10. The algorithm is described in 3.2.10.1, an example is given in 3.2.10.2, and discussion about detecting zero factors in 3.2.10.3.

The part concerning DA-System (DAS) in 3.2.1 and 3.2.1.1 is published in (Lind & Kuusik, Some Ideas for Determinacy Analysis Realisation, 2007). Step-by-step method is described in the same paper[39]. A discussion in 3.2.4 is based on (Lind & Kuusik, Some Problems in Determinacy Analysis Approaches Development, 2008b). The algorithm in 3.2.5 is presented in (Kuusik & Lind, Some Developments of Determinacy Analysis, 2010). DSR (in 3.2.6) and algorithm giving a suitable result for applying DSR-compression (3.2.7) were proposed in (Kuusik & Lind, New Developments of Determinacy Analysis, 2011). Our latest paper (Lind & Kuusik, Algorithm for Finding Zero Factor Free Rules, 2016) contains material presented in 3.2.8, 3.2.9 and part of material in 3.2.10 and its subchapters. The published version of ZFF DA algorithm contains only two types of rules (out of three). Also, the demonstration how the algorithm works on sample data (in 3.2.10.2) and a discussion (3.2.10.3) were left out of this paper due to size limit. All three types of rules are used in the master's work of Liisa Jõgiste (Prototyping of Zero-factor based DA, 2014). In this work the experiments showing dependency of execution time on the number of objects (rows) or attributes (columns) were carried out. These results are presented also in (Lind & Kuusik, 2016).

The referred papers are reprinted in appendices B, C, D, E and F.

[39] The corresponding algorithm is published in (Lind & Kuusik, 2008a).

3.2.1 About the original applications of DA

We know about two original applications of DA. Although their underlying data structures and algorithms used for finding determinations are different, they share a similar approach from the user's point of view.

The user determines X and Y (and some possible restrictions to the rules) and gets a set (system) of rules corresponding to the input. In this set all the rules have the same number of factors and exactly from the attributes given for X. Thus, the found system of rules is additive consisting of non-intersecting rules i.e. each object can be covered maximally by one rule. If no rules have been excluded (due to given restrictions), then the system is complete – each object is covered (by exactly one rule). In such case all existing value combinations of X are presented as an antecedent of some rule.

The output is given as a table where each row represents a determination. For each determination $X\rightarrow Y$ first its components – factors (i.e. attribute with its certain value) constituting X – are listed (in the order they were given by the user). They are followed by the characteristics of determination:

- Accuracy $A(X\rightarrow Y)$
- Completeness $C(X\rightarrow Y)$
- the Number of rules' Applications $n(X)$[40]
- the Number of rules' Confirmations $n(XY)$[41]
- $n(Y)$ – as it is equal for all determinations with the same Y, it can be shown only once per table (not at each row)

Accuracy and completeness are calculated using $n(X)$, $n(XY)$ and $n(Y)$.

For each factor constituting X its contribution to the accuracy ΔA and its contribution to the completeness ΔC is shown. Each contribution is computed regarding all other factors in X, independent on the order of attributes.

The user can change the input and get a new additive system of rules with equal length.

The realization of DA from 1980ies is described in (Veselov, Deza, & Podrabinovich, 1980).

In this case the user can determine 7 attributes for X and 3 attributes for Y. In this realization two trees are created: one contains both reason-attributes X and consequence-attributes Y (in the given order), the other only consequence-

[40] i.e. the number of objects having determining factors in whole dataset or context

[41] i.e. the number of objects having these factors among objects belonging to the class under investigation

attributes. All objects are described in both trees. Each node (except the root) indicates a certain value of a certain attribute (i.e. factor) and has as many descendants as many different values has the next attribute. The leaves represent the value combinations with length equal to the number of observable attributes. For each node (value combination) its corresponding set of objects is kept in memory using lists. On these two trees the user's queries are executed.

In order to find five characteristics (listed above) for a certain determination the necessary nodes are looked up and the number of corresponding objects is detected. To describe a determination we have to find one leaf (representing *XY*) and one node on the way to that leaf (*X*) in the bigger tree and one leaf (*Y*) in the smaller tree.

Finding needed nodes in such trees is comfortable and the number of comparisons is acceptable[42], but the trees itself take much memory. Therefore the number of observable attributes is limited to seven reason-attributes and three consequence-attributes (Chesnokov, 1980a).

Later, at the end of 1990ies, DA was realised in the software package "DA-system"[43] (shortly DAS) by "Context"[44]. The following overview is about its (Russian) version 4.0 for Windows that was available to us for a limited time, it is described in (Lind & Kuusik, 2007). Materials (DALSolution, 2007), (Context, 1999a) (Context, 1999b) have been used also.

This application allows to determine one certain factor for *Y* at a time. In addition to the five characteristics listed above, it finds totals for the whole set (system) of found rules: total Accuracy (weighted average), total Completeness (sum of rules' completenesses), total Number of Applications and total Number of Confirmations (both are sums of rules' characteristics).

The user can set restrictions to the rules by accuracy, completeness, contribution to accuracy and contribution to completeness. All four apply to the individual rules, not to the separate factors in the rules nor to the whole system. If at least one of those four does not fit, the rule is not included into the result.

By default DAS finds all existing combinations of factor-attributes irrespective of their class affiliation i.e. combinations belonging to other class(es) are also found. Such system is a complete system of rules (where the sum of rules' completenesses is 100%). In order to suppress the rules of other classes the required completeness (or required accuracy) of rules has to be set to >0 by the

[42] i^g comparisons, where i is the average arity (the number of branching) of the nodes and g is the depth of the tree (Veselov, Deza, & Podrabinovich, 1980).
[43] Russian "ДА-система"
[44] Russian "Контекст Медиа"

user. In such case the system remains complete. Using a higher threshold the completeness is not guaranteed.

By default the rules can contain positive, zero and negative factors. It is possible to get rules that consist of positive (and zero) factors only by setting the requirement that the contribution to accuracy has to be >0 (≥ 0). Other rules are excluded, therefore the found system loses completeness.

System of rules can be maximally accurate (average accuracy =100%) if there are no contradictions[45] in data. Accurate system can be required also by the user. In order to find only accurate rules the required accuracy has to be set to 1. System got by applying such restriction may be incomplete.

3.2.1.1 Deficiencies of DA-system

Having been analysed DA-system we have the following observations.

Unfortunately DA-system has no possibility to set restrictions to the individual factors and get rules with different number of factors. This is due to technical reasons. By our opinion such lack makes DAS limited compared to theory of DA by Chesnokov (2002).

DAS (4.0) limits the number of attributes (factors) in rules to 5, due to technical reasons. Always it is not enough to describe a class. The current[46] version of DAS (5.0) is claimed to overcome this limitation. A discussion about the number of frequencies needed to compute the characteristics of the rules can be found in chapter 3 of (Lind & Kuusik, 2007). For example, if we have 5 binary attributes, there can be 32 rules at most and the maximal number of frequencies to find is 224. In case of 7 binary attributes: 128 rules and 1152 frequencies. In case of 5 attributes having 10 different values: 10^5 rules and $3*10^5$ frequencies; in case of 7 attributes already $10*10^6$ rules and $34*10^6$ frequencies. Probably due to such big numbers, DA-system limits the number of attributes (factors) in a rule to 5.

A minor drawback is that DAS finds rules for only one class at a time; while by default it finds value combinations of other classes anyway (without showing their class affiliation). It could be more convenient to find rules for all alternative classes (by the same variable), but this exceeds the bound of task of DA (as defined in 2.7.4).

An important disadvantage is that DAS has no automated search strategies to find better solutions. It only helps manual search (by making calculations), but gives no advice for the subsequent search direction. The system does not say which

[45] This is the situation where the chosen set of attributes does not determine object's affiliation uniquely, the same combination of factors (i.e. attributes with certain values) leads to different classes; for example 2/3 of cases belong to class1 and 1/3 belong to class2.
[46] As of 2007

attributes should be included into or excluded from analysis. Quite probably this comes from the way DA-system solves the task. It seems that DAS does not use a backtracking search algorithm, but implements database queries.

Also, the user manual does not include the methodology how to get needed result with DAS. Probably that knowledge is sold at special training courses.

3.2.2 Problems

The following deficiencies of original applications and methodology of DA were pointed out in the previous section:

- Limited number of attributes/factors in the rules
- Finding rules for one class only at a time
- No automated search strategies

Additionally:

- All rules have equal length
- Only additive systems of rules can be found

In order to explain why the last two properties are significant limitations, we will use the example in 2.7.3 (p.55). All presented systems of rules are complete (i.e. all objects are covered by the rules) and accurate (i.e. all rules are accurate). *S1*, *S2* and *S3* are additive[47] systems and *S4* is a non-additive system.

In this example *S1* is a system of rules where all rules contain exactly the same attributes (with different values) and have equal length (such set can be found with DAS). *S2* and *S3* are both additive systems where the rules can have different lengths. Comparing *S1* either with *S2* or *S3* we can see that *S1* contains more rules (3 vs 2) and longer rules (both *S2* or *S3* have one rule with only one factor (in the left side)). For example, instead of two first rules consisting of two factors in *S1*:

- Eyes.blue & Hair.dark → Class.– (C = 40%; 2 objects)
- Eyes.brown & Hair.dark → Class.– (C = 20%; 1 object)

there is one rule consisting of one factor in *S2*:

- Hair.dark → Class.– (C = 60%; 3 objects).

The value of attribute Eyes is not essential when Hair has value dark. Consequently, there is some redundancy in the rules of *S1*.

Comparing non-additive system of rules *S4* with any of the additive systems *S1*, *S2* or *S3*, we can see that *S4* is more compact, containing no redundant

[47] In case of additive systems the rules do not intersect (i.e. do not cover the same objects), thus there is maximally one rule per object

information. For example, the first rules of *S3* and *S4* are identical, but the second rule of *S3*:

- Eyes.blue & Hair.dark → Class.– (C = 40%; 2 objects)

is longer and with smaller completeness than the second rule of *S4*:

- Hair.dark → Class.– (C = 60%; 3 objects) .

In this case, the non-additive system *S4* has shorter rule(s). Comparing it to *S1* we can see that also the number of rules is smaller (2 vs 3). This shows that allowing intersecting (overlapping) rules, the system (set) of rules can be more compact and contain less redundancy.

In order to overcome the problem of equal length, we have proposed so called step-by-step method, where the order of attributes is fixed, but completion of a single rule is stopped whenever it occurs to be accurate. Although step-by-step approach gives a better result than original approach, it has the following disadvantages:

- The result depends on the order of attributes
- Trying all possible orders of attributes is too laborious

Further we have elaborated different algorithms for getting non-additive systems of rules. These algorithms do not use any given order of attributes, but decide themselves which attribute is added next. In each rule the order of attributes can be different and also the set of used attributes can be different. This leads us to automated search strategies. The latest among these three algorithms – ZFF DA – finds rules for all possible classes intermittently.

All these developments try to solve one more important problem:

- Avoiding zero-factors (in the left part of the rule)

Further the essence of zero factors and all our approaches and algorithms will be explained.

3.2.3 Step-by-step approach

Step-by-step approach is our first development of DA. Compared to DAS it allows to find rules with different lengths, thus reducing redundancy (but not eliminating it). In case of such approach we get a new thing to consider – the order of attributes (factors) in the rules. Attributes are added into the rules in a given order, the completion of a single rule is stopped whenever it occurs to be accurate. In case of different orders the results are different. The number of all possible orders is too big to try them all and then find the most suitable one.

The task. The initial data table is given and some feature *Y* (as a certain class). The goal is to describe *Y* (possibly) completely by non-intersecting (possibly) accurate rules.

In case of non-intersecting (i.e. additive) system each object can be covered by one rule at most while each rule can cover more than one object. Thus, the number of rules is ≤ the number of objects (covered by the rules).

As described before (3.2.1), the original application (DAS) solves this task by finding all existing value combinations that contain all given attributes and meet other criteria given by the user. The result consists of rules with equal length (rank), all of them contain the same attributes – this is a simple way to get an additive system of rules (i.e. non-intersecting rules).

It is reasonable to allow rules with different number of factors in argument (keeping them non-intersecting at the same time), to exclude inessential factors. In such case factors are added into rules in some order (not all at the same time) and their contributions to accuracy and completeness can be calculated regarding those factors only that have been added into rule earlier. Latter factors cannot be considered in those calculations.

Below we will present our "step-by-step" approach where the extracted rules can have different lengths while the system is additive. The order of attributes (factors) is essential, they are added into the rules one by one, in the given order.

If some rule obtains the maximal accuracy (A=1) it is not expanded by adding the next factor. At the same time the completenesses of found accurate rules are summed up. Reaching 100% the coverage is found.

Using the same order of attributes for all rules guarantees that the rules do not intersect. The user decides about the order in which the attributes are included into the rules, from the beginning until the situation when all objects of the class are covered.

The following example will demonstrate the step-by-step approach. It will show also that different orders of attributes can give different results.

In this example Quinlan's data from Table 2.7 (p. 55) will be used again. The purpose is to determine class '+' (to describe the persons belonging to class '+'). This class consists of three objects: $n(Y)$=3.

First, we will add attributes in the order they are given in the table: 1) Height, 2) Hair, 3) Eyes.

The rules containing attribute Height (only) are given in Table 3.3.

Table 3.3. The rules consisting of attribute Hair

Height	n(X)	n(XY)	A	C	ΣC
short	3	1	1/3	1/3	
tall	5	2	2/5	2/3	

Neither of the two (candidate) rules is accurate.

We add the next attribute (Hair) into both rules – see Table 3.4.

Theoretically there can be 6 different value combinations of those two arguments. One of those 6 (Height.short&Hair.red) does not exist in the data (*n*(*X*)=0). Two of them (Height.short&Hair.dark; Height.tall& Hair.dark) do not exist in the given class (*n*(*XY*)=0). These three are excluded from the analysis.

Table 3.4. The rules consisting of attributes Height and Hair

Height	Hair	n(X)	n(XY)	A	C	ΣC
short	dark	1	0	0	0	
short	red	0				
short	blond	2	1	1/2	1/3	
tall	dark	2	0	0	0	
tall	**red**	**1**	**1**	**1**	**1/3**	1/3
tall	blond	2	1	1/2	1/3	

The remaining three rules are proper for the target class. One of them (Height.tall&Hair.red) is accurate (*A*=1) and needs no additional factors. This rule covers 33% of the objects of the class (*C*=1/3). We will sum up completenesses of accurate rules (Σ*C*), with hope to reach to 100% coverage (by accurate rules). Two rules having accuracy between 0 and 1 have to be expanded again. Next we add attribute Eyes into those two rules (see Table 3.5).

Table 3.5. The rules consisting of attributes Height, Hair and Eyes

Height	Hair	Eyes	n(X)	n(XY)	A	C	ΣC
short	**blond**	**blue**	**1**	**1**	**1**	**1/3**	2/3
short	blond	brown	1	0	0	0	
tall	**blond**	**blue**	**1**	**1**	**1**	**1/3**	1
tall	blond	brown	1	0	0	0	

Two rules of four have accuracy 1 and both of them have completeness 1/3. At this point we have found three (accurate) rules for the class '–' with overall completeness 1 (100%). Thus the class is completely described:

- Height.tall&Hair.red → Class.+ (C = 33%)
- Height.short&Hair.blond&Eyes.blue → Class.+ (C=33%)
- Height.tall&Hair.blond&Eyes.blue → Class.+ (C = 33%)

This is one possible description for class '+'.

Now we will describe the same class by the same attributes, adding them in another (freely chosen) order: 1) Hair, 2) Eyes, 3) Height.

The rules consisting of attribute Hair only are given in Table 3.6.

Table 3.6. The rules consisting of attribute Hair

Hair	n(X)	n(XY)	A	C	ΣC
dark	3	0	0		
red	**1**	**1**	**1**	**1/3**	1/3
blond	4	2	2/4		

The rule Hair.red→Class.+ is accurate and is included into the result, it covers 1/3 of the class. The rule with Hair.dark has zero accuracy in the given class and will be not expanded. The rule with Hair.blond has accuracy between 0 and 1, thus we add the next attribute (Eyes) into it (see Table 3.7).

Table 3.7. The rules consisting of attributes Hair and Eyes

Hair	Eyes	n(X)	n(XY)	A	C	ΣC
blond	**blue**	**2**	**2**	**1**	**2/3**	1
blond	brown	2	0	0	0	

The first of found rules has accuracy 1 and will be included into the result. Its completeness is 2/3. Now the cumulative completeness is 100%, thus the class '+' is (fully) covered by the (accurate) rules.

At the same time we see that the other branch (with Eyes.brown) has zero accuracy and there is no reason to expand it.

This time class '+' is covered by two rules:

- Hair.red → Class.+ (C = 33%)
- Hair.blond&Eyes.blue → Class.+ (C = 67%)

We have shown that the same class can be (completely) covered by different sets (systems) of (non-intersecting) rules depending on the order of inclusion of the attributes into the rules.

As we can see, attribute Height is not necessary for distinction of classes. The class is described without using it.

Usually we are interested in the minimal number of rules, but we do not know which order of attributes gives such result. Trying all possible orders is too laborious. For example, if we have 10 dichotomous (2-valued) attributes, then we should look through 2^{10}-1 =1023 combinations of attributes, i.e. by 1, 2, 3, …, 10 attributes. It is not real to extract all these rule systems and to analyse them.

Also, the number of factors in the rules should be possibly small. The additivity constraint (i.e. the rules cannot intersect) does not allow leaving out all redundant (inessential) factors. Compare, for example, non-additive system *S4* with either of additive systems *S2* and *S3* (got by step-by-step approach using different orders of attributes) (in 2.7.3, page 55). Such an additive system of rules corresponds to a decision tree.

3.2.4 Problem with zero factors

In the following we will explore the possibility to extract only such rules that do not contain redundancy (zero factors), deciding by the factor's contribution to accuracy at the moment when it is added into the rule. We will reach the conclusion that we cannot say whether the factor remains positive (regarding factors that will be added later) or not. This conclusion is valid for step-by-step approach as well as approaches that produce non-additive systems of rules.

Chesnokov (2002) suggests a *non-additive* system consisting of *accurate normal* rules of different rank as a solution, but gives no description how to achieve such desired result.

Recall that an accurate rule contains no negative factors, all factors are positive or zero factors, and a normal rule consists of positive factors only. A positive factor makes a rule more accurate than it was without it (a negative makes less accurate) and a zero factor does not change the rule's accuracy. Such influence is measured by the contribution to the accuracy (ΔA). For definitions see 2.7.1.

A normal rule is not always accurate. Taking away whatever factor(s) from a normal rule, its accuracy decreases (because every factor in such rule has a positive contribution to the accuracy).

Also, an accurate rule is not always normal (when it contains zero factor(s)).

Example in 2.7.3 presents four different systems of rules consisting of individual rules that can be part of more than one system. Among them there are two normal accurate rules:

- Eyes.brown $\rightarrow$ Class.– (A=3/3=1)
- Hair.dark $\rightarrow$ Class.– (A=3/3=1)

All longer rules are also accurate, but not normal. For example:

- Eyes.blue & Hair.dark $\rightarrow$ Class.– (A=2/2=1)

where Eyes.blue is a zero factor:

- ΔA(Eyes.blue) =
  = A(Eyes.blue & Hair.dark $\rightarrow$ Class.–) - A(Hair. dark $\rightarrow$ Class.–) =
  = 1 - 1 = 0

and Hair.dark is a positive factor:

- ΔA(Hair.dark) =
  = A(Eyes.blue & Hair.dark $\rightarrow$ Class.–) - A(Eyes.blue $\rightarrow$ Class.–) =
  = 1 - 2/5 = 3/5

Sometimes both (all) factors are zero factors, like in the rule

- Eyes.brown & Hair.dark $\rightarrow$ Class.– (A= 1)

where

- ΔA(Eyes.brown) =
  = A(Eyes.brown & Hair.dark $\rightarrow$ Class.–) - A(Hair.dark $\rightarrow$ Class.–) =
  = 1 - 1 = 0
- ΔA(Hair.dark) =
  = A(Eyes.brown & Hair.dark $\rightarrow$ Class.–) - A(Eyes.brown $\rightarrow$ Class.–) =
  = 1 - 1 = 0

Taking away either of the factors we get an accurate rule.

In case of longer rules, it is possible that some factors are zero factors "together" – so that two or more factors at a time can be removed without changing the accuracy.

Based on the same data (see Table 2.7, p. 55), an accurate rule containing only positive factors i.e. normal rule is:

- Eyes.blue&Hair.blond $\rightarrow$ Class.+ (A = 1)

where:

- ΔA(Eyes.blue) =
  = A(Eyes.blue&Hair.blond $\rightarrow$ Class.+) - A(Hair.blond $\rightarrow$ Class.+) =
  = 1 - 2/4 = 1/2
- ΔA(Hair.blond) =
  = A(Eyes.blue&Hair.blond $\rightarrow$ Class.+) - A(Eyes.blue $\rightarrow$ Class.+) =
  = 1 - 3/5 = 2/5

If we take away factor Eyes.blue, we get a normal non-accurate rule:

- Hair.blond $\rightarrow$ Class.+ (A = 2/4)

where

- ΔA(Hair.blond) = A(Hair.blond $\rightarrow$ Class.+) - A($\varnothing$ $\rightarrow$ Class.+) =
  = 2/4 - 3/8 = 1/8

In case of step-by-step approach (see 3.2.3) it is not easy to say whether the current factor is suitable in a sense it is positive ("essential") regarding factors that will be added later. Adding a factor that makes a rule more accurate it may occur later that the factor is inessential anyway.

For example (using data given in Table 2.7):

- Eyes.blue→ Class.+
  (A=3/5)
- Eyes.blue&Height.tall→ Class.+
  (A=2/3; ΔA(Height.tall)=2/3-3/5=1/15>0)
- Eyes.blue&Height.tall&Hair.blond→ Class.+
  (A=1; ΔA(Hair.blond)=1-2/3=1/3>0)

The contributions (of the new factors) to the accuracy are positive in both steps, but the resultant rule contains a zero factor (Height.tall) – the rule is accurate without it:

- Eyes.blue&Hair.blond→ Class.+ (A=1)

We may consider making a control in the opposite direction:

- Height.tall→ Class.+ (A=2/5)
- Height.tall&Eyes.blue → Class.+
  (A=2/3; ΔA(Eyes.blue)=2/3-2/5=4/15>0)

Occurs that Height.tall and Eyes.blue together form a *normal rule (with accuracy below 1)*. However, such finding does not guarantee that both factors remain positive after addition of the next one(s). Addition of whatever possible factor (i.e. values of attribute Hair) into the last rule does not produce any normal rule, although we get accurate rules:

- Height.tall&Eyes.blue&Hair.blond → Class.+ (A=1)
- Height.tall&Eyes.blue&Hair.red → Class.+ (A=1)

Both rules contain zero factor(s): Height.tall is a zero factor in both rules and Eyes.blue is a zero factor in the last rule.

It means that the fact that some factor has a positive impact to the accuracy at the moment it is added into the rule does not guarantee that the factor retains its positiveness.

In our example we found a branch of a search tree that gives no normal accurate rules. As we have seen also, the factor's contribution to the accuracy (at the moment of addition) is not an adequate criterion to avoid entering such branches.

3.2.5 The first algorithm for finding intersecting rules

Heretofore different approaches to DA have given additive (i.e. non-intersecting) systems of rules. Thus it is principally impossible to eliminate all zero factors from the rules. Therefore our next development was to find systems of rules where the rules can intersect. Our first algorithm to this direction finds a possibly small set of rules, monitoring and taking into consideration which objects are covered by the found rules already. Like a step-by-step approach this one also produces one system of rules that is not always the best one. The presented algorithm is not based on neither the previous one (for step-by-step approach) nor

MONSA, although it is MS based and makes extracts. This algorithm uses so called 3D frequency tables.

It is desired that the rules were relatively short – then it is easier to interpret them. For the same reason, the number of rules has not to be very large. Using the previously introduced step-by-step method we should find all rule systems and according to some criteria find from them a cover i.e. a rule set (with completeness 100%) consisting of the shortest rules or a rule set with the least number of rules. This turns out to be very labour-consuming because all the possible sets and orders (permutations) of attributes should be found.

In order to facilitate the work it is reasonable to find systems of rules where the rules can intersect (overlap) i.e. non-additive systems of rules. It would be good to allow a non-fixed order of including factors as well.

Here we will present our first algorithm for getting intersecting rules for DA. The system of rules it finds is non-additive (i.e. objects may be covered by more than one rule), accurate (i.e. consists of accurate rules only) and complete (i.e. covers all objects of determinable class) if there are no contradictions in the data (the algorithm can find them). The findable set of rules is possibly small – the potential rules are not included into the result if they cover only such objects that are covered already. No given order for including factors into the rules is used.

The pseudocode of the algorithm is given below.

```
Algorithm
Determine tables X and Y
S0. t:=0; $U_t$:=∅
    If all the objects in Y are covered then Goto End
S1. Find frequencies in tables $X_t$ and $Y_t$: $Fx_t$, $Fy_t$
S2. For each factor A such that $Fy_t$(A)=$Fx_t$(A) and all
    objects containing A are not covered by rule(s)
        output rule {$U_i$}&A, i=0,…,t
    If at least one new rule was found Goto S0
S3. Choose a new (free) factor $U_t$
    If there are no factors to add then
        {$U_i$}, i=0,…,t is a contradiction; Goto S0
    t:=t+1; extract subtable of objects containing $U_t$; Goto
        S1
End. System of rules is found
```

As we are searching for accurate rules we need to know the accuracies of factors. Accuracy is findable using two frequencies: 1) the frequency in determinable class Y and 2) the frequency in the whole dataset X (see 2.7.1). At every iteration t we collect those two frequencies for each factor into two frequency tables: Fx_t contains frequencies in the whole dataset and Fy_t in class Y (step S1). Fx_t does not change during the work. Frequencies in Fy_t decrease after finding a new rule. Fx and Fy together are called 3D frequency table. Additionally the frequency table of covered (by the found rules) objects (`Fc` in the examples at pages 97-99) is kept. This is one possible way to detect whether a potential rule covers any uncovered object and detect when the work can be finished. The use of frequencies is explained hereinafter.

Our DA algorithm is recursive and extracts certain subsets X_t (t is a level of recursion) while working, for every X_t its corresponding Fx_t and Fy_t are found.

We can detect accurate rules by the use of frequency tables. In any subset (at any level) holds a rule: if some factor belongs to one class only, we get an accurate rule including it (into the set of factors that are selected already and thus are common to all objects in current X_t). Such factor has equal frequencies in Fx_t (the whole set) and in Fy_t (the observable class) – in case of equal frequencies the accuracy is 1. A situation of this kind is easy to detect because we have both frequencies for each factor in frequency tables. If such new rule covers at least one object (in class `Y`) that is not covered by the rules yet, it is appended to the resultant set of rules. This condition is also easy to detect using the frequencies of covered objects (`Fc`). If factor's frequency in `Fc` is same as in `Fx` and `Fy` then all objects containing it are already covered by the found rules and the potential rule produced by that factor is not added into result, otherwise (when the frequency in `Fc` is lower) the new rule is suitable. This way the last factor of each rule is found. All other factors in a rule are found in step S3.

The frequencies in `Fc` are updated (incremented) every time a new rule is extracted; this table is independent on the recursion level (t). The end of work (the situation when all objects belonging to `Y` are covered by the found rules) is detected using `Fc` as well. If all frequencies in `Fc` are equal to the ones in initial `Fy` then all objects of `Y` are covered and the work can be finished.

Factors before the last one are selected recursively from the current (sub)set X_t. The selection criteria are based on frequencies, the maximal frequency in Fy_t, for example. In case of equal maximal frequencies in `Fy` the one having bigger frequency in `Fx` is preferred.

Every time a new rule(s) is(are) found, the algorithm turns back to the initial level and selects a new first factor. Compared to turning back to the previous level, such strategy enables to find possibly short (i.e. containing possibly few factors) rules.

This algorithm is able to detect a contradiction – the situation where identically described (by used attributes) objects belong to different classes. The algorithm detects it when it does not find any possible factor for extracting a next subset. It means that all attributes are already included into the current set of factors, but the accuracy is below 1. The objects covered by such contradictory set of factors are treated as covered objects when counting frequencies into `Fc`.

In case of contradiction the full coverage (i.e. maximal completeness) by accurate rules is not achievable. Optionally we may consider to output such contradictory non-accurate rules showing their (lower) accuracy.

In the following we will demonstrate the work of the algorithm using data from (Quinlan, 1984) again (given in Table 2.7). This time we use a numerical representation, the coding used for attributes and their values is shown in Table 3.8. The initial data table is given in Table 3.9. Let *X* is X(8,3), X_{ij} = 1,...,3 and *Y*=4.1 {Y_i: 1,2,5,7,8} (i.e. class "–"). The frequencies of attributes' values for *X* and *Y* (Fx and Fy accordingly) are given in Table 3.10.

Table 3.8. The coding used for Quinlan's (1984) data

Attribute Code	**Height** 1	**Hair** 2	**Eyes** 3	**Class** 4
1	short	dark	blue	–
2	tall	red	brown	+
3		blond		

Table 3.9. Coded initial data table

i \ j	1	2	3	4
1	2	1	1	1
2	1	1	1	1
3	2	3	1	2
4	2	2	1	2
5	2	3	2	1
6	1	3	1	2
7	1	3	2	1
8	2	1	2	1

Table 3.10. The frequencies for X and Y

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	5		1	2	3	2
	2	5	1	3		2	3	0	3
	3	0	4	0		3	0	2	0

For value 1 of attribute 2 (shortly 2.1) the frequencies in Fx and Fy are equal which means that all objects having 2.1 belong to Y. Hence we get a rule 2.1=3 (Hair.dark→Class.–). The frequency after "=" shows that the rule covers three objects (namely objects 1, 2 and 8) – this is additional information. Also for 3.2 the frequencies are equal and we get another rule 3.2=3 (Eyes.brown→Class.–) that also covers three objects (5, 7 and 8).

By those two rules

- Hair.dark→Class.– (3 objects)
- Eyes.brown→Class.– (3 objects)

all the objects belonging to Y are covered. The found rules are overlapping, both cover object 8. The result coincides with an example of a non-additive system of rules *S4* in 2.7.3 (p. 55).

In order to demonstrate other steps of the algorithm another example is presented.

This time *Y*=4.2 {Y_i: 3,4,6} (class "+"). The frequency tables Fx and Fy are given in Table 3.11.

Table 3.11. The frequencies for X and Y

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	5		1	1	0	3
	2	5	1	3		2	2	1	0
	3	0	4	0		3	0	2	0

For 2.2 (attribute 2 with value 2) the frequencies in Fx and Fy are equal. The rule 2.2=1 (Hair.red→Class.+) covers object 4.

The "free" frequencies i.e. the frequencies over non-covered objects (in *Y*) after extraction of the first rule are shown in Table 3.12.

Table 3.12. Free frequencies for Y after extraction of the first rule

Fy	Kj \ j	1	2	3
	1	1	0	2
	2	1	0	0
	3	0	2	0

From here the factor starting a new rule is chosen by maximal frequency in Fy. As there are two (equal) maximal frequencies, the choice is made by the bigger frequency in Fx where the factor 3.1 has frequency 5 and 2.3 has frequency 4. Thus 3.1 is selected.

Table 3.13. Extract by 3.1

i \ j	1	2	3	4
1	2	1	1	1
2	1	1	1	1
3	2	3	1	2
4	2	2	1	2
6	1	3	1	2

Table 3.14. The frequencies of extracted data

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	2	2	5		1	1	0	3
	2	3	1	0		2	2	1	0
	3	0	2	0		3	0	2	0

Table 3.15. The frequencies of covered objects after extraction of the first rule

Fc	Kj \ j	1	2	3
	1	0	0	1
	2	1	1	0
	3	0	0	0

Next the objects having 3.1 are extracted (see Table 3.13) and the frequencies for X and Y over the extracted data are found (see Table 3.14). The frequencies of covered objects (Fc) are given in Table 3.15.

In Table 3.14 we can see two potential rules, (3.1) with 2.2 and (3.1) with 2.3. Looking into Fc (Table 3.15) we see that 2.2 has equal frequency here indicating that the object(s) containing factor 2.2 is(are) already covered by the found rules. Therefore 2.2 does not produce a new rule (rule 3.1&2.2=1 would be redundant covering the same object as rule 2.2=1). The factor 2.3 is suitable for completing a rule, its frequency in Fc is smaller than 2. So, the second rule is 3.1&2.3=2 (Eyes.blue&Hair.blond→Class.+), it covers objects 3 and 6.

The two rules we have found cover all the objects belonging to Y. We detect it turning back to the initial level and comparing initial Fy with Fc (after extraction of the last rule – see Table 3.16): all the frequencies in both tables are equal.

Table 3.16. The frequencies of covered objects after extraction of the second rule

Fc	Kj \ j	1	2	3
	1	1	0	3
	2	2	1	0
	3	0	2	0

This time the found rules

- Hair.red → Class.+ (1 object)
- Eyes.blue&Hair.blond→Class.+ (2 objects)

do not intersect. In the given data set it is possible to describe class "+" by non-intersecting rules that do not contain any redundancy.

Generally the presented algorithm finds one non-additive system of accurate rules (for given *X* and *Y*). Compared to the approaches that find only non-intersecting rules (i.e. additive systems) the number of rules is not bigger (usually it is smaller) and the rules are shorter also. The ability to detect a contradiction is also valuable.

However, there is no guarantee that the extracted system of rules is the shortest (with a lowest number of rules). Realizing the algorithm it is possible to apply several different principles for selecting the next factor and thus to get different results, one possibility is to let the user make the decision.

3.2.6 Determinative set of rules

Instead of finding one system of rules, it was good to find all non-redundant rules and then combine different covers (rule sets) from them. For that purpose we need an algorithm that produces (at least) all non-redundant rules as well as a procedure for eliminating redundant rules (if the result of the algorithm contains them).

DSR – the set of non-redundant rules defined in this subchapter – gives a source for post-processing the rules. An algorithm for finding all non-redundant rules will be presented in the next subchapter (3.2.7).

We are interested in possibly short rules. If there are two accurate rules

- Eyes.blue & Hair.dark $\rightarrow$ Class.– (C = 40%; 2 objects)
- Hair.dark $\rightarrow$ Class.– (C = 60%; 3 objects)

we prefer the second one. It covers the two objects covered by the first rule plus one more object. We say that the first rule is contained in the second one or it is a subrule of the second rule. Comparing their left sides, the first one is longer containing all the factors of the shorter rule and some additional factors. Having such rules we consider the longer one to be redundant.

Let a table *X*(*N*,*M*) be given and a set *B* of all possible rules describing (only) the class *Y* and each rule in *B* is presented only once.

The Determinative set of rules (DSR) for class *Y* consists of all rules which are not contained in other rules of *B*.

$B = \{Ri\}$, $i=1, 2,..., K$, where K is a number of all possible rules describing (only) the Class Y. $Ri \neq Rj$, $i \neq j$.

$DSR = \{Ru\}$. $Ru \in DSR$ if there $/\exists$ $Ri \in B$, $Ru \subset Ri$, $i \neq u$. $DSR \subseteq B$

It means that DSR does not contain subrules of its rules. To get DSR from *B* we have to throw out all subrules of rules. We call this process „rule set compression“.

Example. Let *B* contain 4 rules (all possible rules for *Y* = (Class =1)):

- r1: IF T1=1 & T2=1 THEN CLASS=1

- r2: IF T1=1 & T3=2 THEN CLASS=1
- r3: IF T2=1 THEN CLASS=1
- r4: IF T3=2 THEN CLASS=1

As we see, the rule r1 is contained in r3 and r2 is contained in r4. According to the definition DSR_B = {r3, r4}.

The main features of DSR are:

1. there are no redundant attributes (zero factors) in rules,
2. the same object in the class Y can be described by several rules.

Such compression can be applied either during the work of the main algorithm (that finds the potential DSR rules), each time a new rule is found, or as a separate step after the main algorithm. In order to facilitate compression the following aspects could be taken into account:

- The redundant rule is longer (i.e. its left side contains more factors) than the one that forces it out;
- Its left side contains all the factors of the shorter one;
- Its frequency (support) is $\leq$ than this of shorter one;
- Both rules describe the same class;
- The longer (redundant) rule is always found before than the shorter one.

The last property is characteristic to MS algorithms (and likely to other algorithms as well). After finding the rule Hair.dark$\rightarrow$Class.– the possible rule Hair.dark&Eyes.blue$\rightarrow$Class.– is not found, but the rule Eyes.blue&Hair.dark$\rightarrow$ Class.– can be found (only) before finding the rule Hair.dark$\rightarrow$Class.–.

The presence of the non-redundant rules (that belong to DSR) has to be guaranteed by the algorithm used for finding the rules. The compression procedure can be applied to the rule set regardless it contains all needed rules. If it does not, then the result cannot be the expected one.

Our algorithm for finding all the rules needed for getting DSR is given in the next section (3.2.7). It finds possibly few redundant rules.

On the basis of DSR we can form and solve the following tasks – to find:

1. the shortest rules (by rank[48]),
2. the longest rules (by rank),
3. the rules with specific features (for example, all rules of the rank r in DSR),
4. the shortest rule system (i.e. the rule system with the smallest number of rules),
5. the rule system which consists of rules with minimal ranks,

[48] the number of attributes in the rule

6. all the rule systems we can form on the basis of DSR.

Tasks 1-3 are easily solvable, but tasks 4-6 are essentially system covering tasks and they are NP-complex tasks.

These tasks are important for post-analysis of rules giving several new possibilities for experimentation with several rule sets (subsets of DSR) and for describing them. We must not try to minimise the rule set during the work of a rule finding algorithm, we can find the best solution during the post-analysis of DSR.

Using DSR and the post-analysis of rules also gives a possibility to gather statistics about the use of rules for classification in order to analyse the rules' perspective and their power of classification. We can see which rules classify more accurately and which do not on the basis of the information we have about classified (test-set and real) objects. On this basis we can reorder the rules in the rule set. DSR is a good basis for developing this approach.

Finding DSR is laborious, especially in cases of large amounts of data. The user can decide whether it is reasonable. If the purpose is a quick one-time information gathering about a data set under analysis then the use of DSR-based approach may not be the best one. But if the purpose is to describe the data set and through that discover new knowledge and get an opportunity for post-analysis of the rule set then this approach is a good solution.

We have extended this DSR-approach so that the rules for all existing classes are found, solving thus the multiple-concept learning task (Kuusik & Lind, 2012). Having applied the before described compression to the results of our different ML algorithms (Kuusik & Lind, 2012), (Roosmann, Võhandu, Kuusik, Treier, & Lind, 2008) the results are identical – from that observation the idea for DSR comes.

Although DA rules are not association rules, we can compare our DSR to the non-redundant association rules (see 2.4.2.1). Zaki defines non-redundant rules as those that have minimal antecedents and consequents, in terms of subset relation; Bastide et al – as the rules with minimal antecedent and maximal consequent. In case of DA rules the right side of the rules is not a subject to comparison, only the rules with equal right side (i.e. describing the same class) are compared. Thus the left sides of DA rules are compared. Both approaches to non-redundant association rules prefer shorter left sides (minimal antecedents), the same is true for our DSR.

3.2.7 Algorithm for finding all possible shortest rules

As said before, DSR (3.2.6) is a basis for forming different sets (systems) of rules. Here we present a MS based algorithm that finds all rules needed for DSR and some redundant rules[49] (that are eliminated by compression). The rules can intersect, regardless of how many times the objects are covered. Redundancy (zero factors) in the rules is avoided as much as possible by the nature of the algorithm. Differently from the previous 3D-algorithm (in 3.2.5) it uses usual backtracking (while the former turns back to the initial level), does not track the coverage of objects and uses elimination technique "bringing zeroes down" from MONSA (2.6.1).

This is a depth-first-search algorithm that makes subsequent extracts of objects containing certain factors. At each level first of all the rules (of that extract) are detected and then factors for making extracts of the next level are selected one by one.

The algorithm uses frequency tables for X_t (all objects of current extract) and Y_t (objects belonging to observable class of current extract), Fx_t and Fy_t accordingly. If there are equal frequencies in both frequency tables for some factor then this factor completes a rule. The rule includes also the factors chosen on the way to that extract.

The selection criteria for choosing the next factor are based on frequencies, the maximal frequency in Fy_t. In case of equal maximal frequencies in Fy_t the one having lower frequency in Fx_t is preferred. If only one attribute (of the extract) has free (unused) value(s) (indicated by frequencies over zero in Fy) then it is not practical to make a next (further) extract because there would be no free factors to distinguish objects of different classes in that extract. If there are no free factors (i.e. no frequencies over zero) then obviously it is not possible to make a next extract. In both cases the algorithm backtracks to the previous level.

Each factor that is used for making an extract or completing a rule is set to zero in the corresponding Fy. Each Fy (except for the initial level) inherits all zeros of the previous level (we call it "bringing zeroes down"). These zeroing techniques prevent many redundant extracts and rules without losing the rules of DSR.

The pseudocode of the algorithm is given below.

```
Algorithm
Determine tables X and Y
S0. $t:=0;\ U_t:=\varnothing$
S1. Find frequencies in tables $X_t$ and $Y_t$: $Fx_t$, $Fy_t$
    If t>0 then
        For each factor A such that $Fy_{t-1}(A)=0$
            $Fy_t(A):= 0$
S2. For each factor A such that $Fy_t(A)=Fx_t(A)$
        output rule $\{U_i\}\&A,\ i=0,\dots,t;\ Fy_t(A):= 0$
S3. If not enough free factors for making extract then
        If t=0 then Goto End
```

[49] The rules that are contained in some other rules are considered redundant (see 3.2.6).

```
        Else t:=t-1; Goto S3
S4. Choose a new (free) factor Ut
    Fyt(A):= 0
    t:=t+1; extract subtable of objects containing Ut;
    Goto S1
End. System of rules is found
```

Compared to the first algorithm for finding intersecting rules (in section 3.2.5) the differences are following:

- The coverage of the objects (Fc) is not tracked;
- It does not turn back to the initial level after extraction of a rule;
- It uses elimination technique "bringing zeroes down" (similarly to MONSA – see 2.6.1);
- The contradictions are not detected.

Next an example using Quinlan (1984) data is presented. Here we will repeat its numeric representation (Table 3.9), the used coding is in Table 3.8 (p. 97).

The initial data table is given in Table 3.17. Let X is X(8,3), X_{ij} = 1,...,3 and Y=4.2 {Y_i: 3,4,6} (i.e. class "+"). The frequencies of attributes' values for X and Y (Fx and Fy accordingly) are given in Table 3.18.

Table 3.17. The initial data table

i\j	1	2	3	4
1	2	1	1	1
2	1	1	1	1
3	**2**	**3**	**1**	**2**
4	**2**	**2**	**1**	**2**
5	2	3	2	1
6	**1**	**3**	**1**	**2**
7	1	3	2	1
8	2	1	2	1

Table 3.18. The frequencies for X and Y

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	5		1	1	0	3
	2	5	1	3		2	2	1	0
	3	0	4	0		3	0	2	0

For factor 2.2 (Hair.red) the frequencies in Fx and Fy are equal which means that all objects having 2.2 belong to Y. Hence we get a rule R1: 2.2=1 (Hair.red). The frequency after "=" shows that the rule covers one object (namely object 4) – this is an additional information. The frequency of 2.2 is set to zero in the current Fy to avoid using it as a basis for making next extract(s). The current state of the frequency tables is given in Table 3.19.

Table 3.19. The frequencies for X and Y after extraction of the first rule

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	5		1	1	0	**3**
	2	5	*0*	3		2	2	*0*	0
	3	0	4	0		3	0	2	0

Now we have to choose a factor for making a next extract. The factor with the biggest frequency in Fy – 3.1 (Eyes.blue) is selected. An extract (subtable of the table X) by 3.1 is shown in Table 3.20 and the corresponding frequency tables in Table 3.21.

Table 3.20. Extract by 3.1

i \ j	1	2	3	4
1	2	1	1	1
2	1	1	1	1
3	2	3	1	2
4	2	2	1	2
6	1	3	1	2

Table 3.21. The frequencies of extracted data

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	2	2	5		1	1	0	3
	2	3	*0*	0		2	2	*0*	0
	3	0	2	0		3	0	2	0

Every frequency table for *Y* (Fy) inherits all zeroes of the previous level ("bringing zeroes down"), to avoid repetitious extracts and redundant rules. Therefore the actual frequency of 2.2 in Fy (=1) is replaced by 0. If this frequency was not set to zero we would find the rule 3.1&2.2=1 which is a subrule of the already found rule 2.2=1. From the current extract we find the rule R2: 3.1&2.3=2 (Eyes.blue & Hair.blond). The frequency of 2.3 is set to zero after that.

Now all the frequencies over zero (in Fy) – 1.1 and 1.2 – are in the same column (attribute). Making an extract by any of these factors cannot give a rule because there are no attributes to use for distinguishing between different classes at the next level. Therefore the algorithm goes back to the previous level (that is the initial level).

The frequency table Fy of that level (see Table 3.22) has got two zeroes already: 2.2 has been set to zero when the rule with it was extracted and 3.1 has been set to zero due to being the basis for extract. In Fy there are two factors with maximal frequency (=2): 1.2 and 2.3. Their frequencies in Fx are different, we choose the one with smaller frequency in Fx: factor 2.3 (Hair.blond). The extract by 2.3 is in Table 3.23 and its corresponding frequencies in Table 3.24.

Table 3.22. The frequencies of the initial level

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	*0*		1	1	0	*0*
	2	5	*0*	3		2	**2**	*0*	0
	3	0	**4**	0		3	0	**2**	0

Table 3.23. Extract by 2.3

i \ j	1	2	3	4
3	2	3	1	2
5	2	3	2	1
6	1	3	1	2
7	1	3	2	1

Table 3.24. The frequencies of extracted data

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	2	0	*0*		1	1	0	*0*
	2	2	*0*	2		2	1	*0*	0
	3	0	4	0		3	0	2	0

There are no equal frequencies (over zero) in these frequency tables (Table 3.24). Again, all usable (non-zero) factors in Fy (1.1 and 1.2) come from the same attribute and therefore it is not reasonable to make an extract by any of them. The algorithm backtracks to the initial level (see Table 3.25) where again, all non-zero frequencies (in Fy) are in the same column. The algorithm finishes its work.

Table 3.25. The frequencies of the initial level

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	*0*		1	1	0	*0*
	2	5	*0*	3		2	2	*0*	0
	3	0	*0*	0		3	0	*0*	0

During the work two rules have been found:

- R1: 2.2=1 (Hair.red→Class.+)
- R2: 3.1&2.3=2 (Eyes.blue & Hair.blond→Class.+)

From this very small data set we did not find any redundant rule. An example with bit bigger data from which some redundant rules are found also is given in (Kuusik & Lind, 2011) where we presented this algorithm and DSR approach.

The presented algorithm finds all needed non-redundant rules and additionally some redundant rules that can be eliminated afterwards in order to get DSR.

In the extended version of this algorithm that finds the rules for all existing classes (Kuusik & Lind, 2012), for each class its Fy is used and the rules are detected for all classes intermittently.

3.2.8 Types of zero factors

Dealing with redundancy we found two different types of zero factors. Making the difference is important for showing the relation of zero factors (of DA) with the elements between closed set and its generator (see 3.1.5). Here we will introduce different types of ZF, the relations with CSs and generators will be shown in 3.2.9.

We have found that there are two types of zero factors:

1. the ones with zero contribution to the completeness ($\Delta C=0$) that do not change the rule's coverage (set of covered objects) and frequency (the number of objects it covers) and
2. the ones with negative contribution to the completeness ($\Delta C<0$) that decrease the rule's frequency.

We will call them *zero-zero factors* and *zero-negative factors*, accordingly. Recall that zero factor means a factor with zero contribution to the accuracy ($\Delta A=0$), so the accuracy of the rule does not change in either case.

The two different systems of accurate rules found by step-by-step approach (see 3.2.1, p. 90) are:

- A1: Height.tall&Hair.red $\rightarrow$ Class.+ (C = 1/3)
- A2: Height.short&Hair.blond&Eyes.blue $\rightarrow$ Class.+ (C=1/3)
- A3: Height.tall&Hair.blond&Eyes.blue $\rightarrow$ Class.+ (C = 1/3)

and

- B1: Hair.red $\rightarrow$ Class.+ (C = 1/3)
- B2: Hair.blond&Eyes.blue $\rightarrow$ Class.+ (C = 2/3)

An example of zero-zero factor is factor Height.tall in the rule A1 (with completeness 1/3), because the rule without it (B1) has the same completeness (both rules cover exactly the same objects).

Height.short in A2 and Height.tall in A3 are zero-negative factors. If either of them is added into the rule B2 then the completeness of the rule decreases from

2/3 to 1/3. This negative difference (1/3–2/3) is the factor's contribution to the rule's completeness. A2 and A3 are subrules of B2.

3.2.9 Relations of DA rules with closed sets and generators

Relating DA rules with closed sets and generators gives us possibility to define non-redundant rule based on these concepts. We have already shown the correspondences between concepts used in ARM and in MONSA (in 3.1.2), thus now we can relate DA and MONSA in order to use the latter to produce DA rules (3.2.10).

Before we will show how zero factors and DA rules relate to closed sets and generators, recall these concepts (introduced in 2.4.1.1).

In frequent itemset mining an item is a binary attribute that can be either present or not in a transaction (a database record). For example, in market basket databases the items represent purchased goods. Extending the concept to multi-valued attributes, an item is a certain attribute with a certain value from the set of different possible values for that attribute. For example, in case of a market basket database, instead of "bread" there can be either "black bread" or "white bread" (i.e. attribute "bread" with either value). Such an item corresponds to a DA factor.

A *closed (item)set* is the maximal set of items common to a set of objects (Pasquier, Bastide, Taouil, & Lakhal, 1998), it has no superset with the same support (i.e. frequency) (Zaki & Hsiao, 2002). Adding whichever item decreases its coverage and frequency. For example, one of the closed sets in Table 2.7 is Hair.blond&Eyes.blue&Class.+ with frequency 2. If we add Height.short or Height.tall to this itemset, then the frequency changes and the resultant itemset is not the same closed set anymore.

A *closure* is the smallest (minimal) closed itemset containing the given itemset (Bastide, Taouil, Pasquier, Stumme, & Lakhal, 2000) i.e. the itemset's maximal superset with the same frequency. For example, the closure of Hair.red (frequency=1) is Height.tall&Hair.red&Eyes.blue&Class.+ (frequency=1). A closed set is the same as its closure.

A *(minimal) generator* of a closed set is an itemset with the same closure and with no proper subsets with the same closure (Bastide, Pasquier, Taouil, Stumme, & Lakhal, 2000). Taking away whichever item increases its coverage (and frequency). For example, Hair.blond&Eyes.blue with frequency 2 is a generator of the closed set Hair.blond&Eyes.blue&Class.+ with a frequency of 2. Taking away either Hair.blond or Eyes.blue from the generator gives us an itemset with bigger frequency and thus with different closure.

If the number of items in a closed set and its generator differs more than by one then also the sets between the minimal generator and the closed set can be used for generating a closed set and can be called generators (for example, itemsets between Hair.red and Height.tall&Hair.red&Eyes.blue&Class.+). However, mostly “generator” means the minimal generator.

A closed set can have more than one minimal generator. For example, the closed set Hair.blond&Eyes.blue&Class.+ has two (minimal) generators: Hair.blond&Eyes.blue and Hair.blond&Class.+.

A closed set or a generator is said to be *frequent* if its frequency is more than or equal to a given threshold. If the frequency threshold is 2, then Height.tall&Hair.red&Eyes.blue&Class.+ and its generators are infrequent (frequency=1); Hair.blond&Eyes.blue&Class.+ and its generators are frequent (frequency=2).

Now we can turn to the relations.

A closed set is the maximal set of items common to a set of objects and its (minimal) generator is a minimal set of items common to that set of objects. Between the closed set and its generator there are such items, the addition or removal of which does not change the coverage and frequency of the itemset. Those items are similar to zero-zero factors that do not change either the accuracy or the completeness of the DA rule. Just the class-belonging usually is not observed in case of closed sets. Consequently, in order to avoid zero-zero factors the left side of the rule has to be a minimal generator.

Minimal generators do not contain zero-zero factors, but they can contain zero-negative factors. For example, the generator Height.tall&Hair.blond&Eyes.blue determines Class.+ (i.e. rule A3 (in 3.2.8): Height.tall&Hair.blond&Eyes.blue→ Class.+), but Height.tall is a zero-negative factor, because Hair.blond&Eyes.blue is enough to determine Class.+ (rule B2: Hair.blond&Eyes.blue→Class.+). Height.tall decreases the rule’s completeness by 1/3 (from 2/3 to 1/3). Thus, if a generator produces a rule (generator→class) then the rules with super-generators of that generator contain zero-negative factors and are redundant.

Therefore, for class detection we need such (minimal) generators that define a class and have no such subset that defines a class.

In case of non-classified data, elements between CS and its generator correspond to zero-zero factors (in that context we call them just zero factors – as in 3.1.5). Elements/items/factors that would decrease the frequency of the current itemset (when added), could be seen corresponding to zero-negative factors (we do not use such name). Without classification such “zero-negativity” does not mean redundancy as well as being zero-zero factor is not redundancy when the CSs are searched for (CSs are found according to frequency threshold). Actually, “zero” factor (in DA) is a factor that does not change the accuracy (the ratio of two frequencies) of DA rule. CSs do not have accuracy (as they do not have a consequent part); in case of CSs just the frequency changes or not (when some element/factor is included into it).

3.2.10 Zero Factor Free DA

Here we will present a new approach to DA that finds non-redundant rules for all classes (not for one target class only) and additionally positive and negative association rules (at the same time). For the same antecedent three types of consequents can be found. For example, from “has a cow” we can conclude that the person belongs to the class of “rancher” (classification rule), “lives in the country” (positive association rule) and “is not a frequent traveller” (negative association rule): “has a cow” $\Rightarrow$ “is a rancher” AND “lives in the country” AND (NOT “is a frequent traveller”).

Corresponding algorithm is based on MONSA and its developments. In 3.2.9 we reached a conclusion that the left side of DA rule has to be a (minimal) generator (in order to contain no redundancy). We have already a MONSA-based algorithm for finding all generators with their closed sets (3.1.6) and excluded factors (3.1.7) as well as algorithm that integrates classes into MONSA (3.1.8). From the algorithm for finding generators (together CSs and EFs) we leave out gathering ECs and add the possibility to detect a class. DSR-compression (3.2.6) is used to remove some redundant generators (from classification rules).

From minimal generators that define a class, we can build rules IF minimal-generator THEN class (min-gen$\rightarrow$class). In this case we get the rules with zero-factor-free (ZFF) left sides and therefore we call our approach Zero Factor Free DA (ZFF DA).

Zero-zero factors that have to be left out from the left side of the rule can be moved to the right side – the conclusion part: IF minimal-generator THEN zero-zero-factors (min-gen$\rightarrow$zero-factors). Used this way they show which factors are accompanied by which factors. For example, an itemset Height.tall&Hair.red (the left side of the rule A1 in 3.2.8, p.107) contains a zero-zero factor Height.tall. Moving this factor from the left side to the right side we get an accurate rule Hair.red $\rightarrow$ Height.tall:

A(Hair.red $\rightarrow$ Height.tall) = n(Hair.red&Height.tall) / n(Hair.red) = 1/1 =1.

This rule informs us that everyone having red hair is tall (the rule with so low support as here (=1) is not the best example, of course). Such a rule is an exact association rule (for definition see 2.4.2).

Actually, for Hair.red there are more factors suitable for putting to the right side: Hair.red $\rightarrow$ Height.tall & Eyes.blue & Class.+. This example reveals an important observation that a class can be detected the same way as other zero-zero factors. The difference is that the class attribute(s) is(are) never put on the left side of the rule, while all other attributes can be present in either of the sides (but not on both sides in the same rule).

Zero-negative factors cannot be moved from the left side to the right side this way. An itemset Height.short&Hair.blond&Eyes.blue (the left side of an accurate rule A2 in 3.2.8,) contains a zero-negative factor Height.short. Moving that zero-negative factor to the right side, we get a new rule – Hair.blond&Eyes.blue → Height.short – that is not accurate:

A=n(Hair.blond&Eyes.blue&Height.short) / n(Hair.blond&Eyes.blue) = 1/2.

In the following, if we use "zero factor" then zero-zero factor is meant.

Additionally, it is possible to construct rules with a negation on the right side, showing which factors do not occur in the objects covered by the left side of the rule. We call these factors excluded factors. The form of such rule is: IF minimal-generator THEN NOT excluded-factor (min-gen→NOT excl-factor). For example, Eyes.brown→NOT Hair.red. In case of more than one excluded factor each of them is negated separately: (NOT excl-factor1) AND (NOT excl-factor2) [AND …].

Only such attributes are considered that do not provide any other factor in the same rule. For example, if holds Hair.red → Eyes.blue, then true rule(s) Hair.red → NOT Eyes.brown (and thinkable Hair.red → NOT Eyes.green if there were some persons with green eyes) is(are) not produced, because we can deduce ourselves that persons with red hair do not have brown (or green) eyes (because they all have blue eyes).

Among excluded factors (of the same rule) more than one value of the same attribute can be present while for the two previously presented rule types (having non-negated items on the right side), for each used attribute only one possible value can be included. Our little data table does not contain any suitable example. Suppose that there were four possible eye colours: blue, brown, green and grey. Then it was possible that read-haired persons had either blue or grey eyes. In such case a negative rule for other eye colours – Hair.red → NOT Eyes.brown AND NOT Eyes.green – was formed, otherwise not. Attributes with less than three different values cannot appear among excluded factors.

Zero factors devolve from more general rule to more specific rules: if holds A→B then holds A&C→B as well. We will give an example where the zero factor is a class. For example, holds Eyes.brown→Class.– and also more specific rule Eyes.brown&Height.tall→Class.– (that is a redundant rule being a subrule of another classification rule).

Generally such inheritance is true for excluded factors as well, but not in case when an attribute (presented among excluded factors) goes to the constitution of closed set i.e. either generator or zero factor (because we do not consider factors from those attributes as excluded factors). For example, holds a rule Height.short→NOT Hair.red. For Height.short&Eyes.brown we find a zero factor Hair.blond and get the rule Height.short&Eyes.brown→Hair.blond. In such case other hair colours (incl. red) are excluded (by logic) without considering them as excluded factors. When attribute Hair is included into the generator (Height.short&Hair.blond or Height.short&Hair.dark) then again all other hair colours are obviously excluded.

Our ZFF DA offers three types of rules, having a minimal generator on the left side, but with different right sides:

1. Classification rule:
   IF minimal-generator THEN class (min-gen→class)
2. (Positive) Association rule:
   IF minimal-generator THEN zero-zero-factors (min-gen→zero-factors)
3. Negative association rule:
   IF minimal-generator THEN NOT excluded-factor
   (min-gen→NOT excl-factor)

For the same minimal generator all three types can be combined into a single rule:

IF minimal-generator THEN class [AND zero-factor(s)] [AND NOT excluded-factor(s)]
where more than one zero factors are connected by logical conjunction (AND) and excluded factors are negated singly before connecting by conjunction:

(NOT excluded-factor1) AND (NOT excluded-factor2) [AND …] .

Next we present an algorithm for producing zero-factor-free rules – all three types. The algorithm is based on finding generators. For each generator it is possible to detect the difference with its corresponding closed set i.e. zero-zero factors, a possible class among them, and to find excluded factors (from the attributes that are not contained in that closed set) as well. Finding of all needed generators is guaranteed. The majority of their unwanted supergenerators (containing zero factors) can be avoided, the remaining part is excluded by compression of the initial result (after the main algorithm).

Supergenerators of other minimal generators are considered redundant only for the classification rules (the first type). In case of association rules (both positive and negative i.e. types 2 and 3) the support threshold (minimal allowed frequency) limits the depth of such rules. In our algorithm, we also stop going further (with association rules) when we have found a classification rule. For example, if we have found a rule Eyes.brown→Class.–(AND NOT Hair.red) we will not investigate whether there is any zero factor or excluded factor for Eyes.brown&Hair.dark (that actually has zero factors Height.tall and Class.–) and other supergenerators of Eyes.brown (Eyes.brown&Hair.blond, Eyes.brown& Height.tall, Eyes.brown&Height.short).

3.2.10.1 Description of the Algorithm (of Zero Factor Free DA)

This is a depth-first search algorithm that makes subsequent extracts of objects containing certain factors. From the root to the leaves (of search tree), the frequencies of extracts always decrease. Each extract is determined by a generator. Each generator is found only once.

The algorithm uses frequency tables that show for each attribute the frequencies of all its possible values (in the set of objects for which it is found).

The frequencies (in the frequency table) can be equal to or smaller than the current ("leading") frequency (the number of the objects in the current extract). Equal frequency shows that all objects of the extract contain that factor. For each attribute, there can be at most one frequency equal to the leading one, in such case all other frequencies for that attribute are zeroes. Factors with such frequency are zero-zero factors (in the current extract).

Detecting whether the generator determines a class is analogous. If for the class attribute one value has a frequency equal to the leading one (and others are zeroes), then all objects of the extract belong to that class.

In addition to class and zero factors "excluded" factors can be detected. Excluded factor is a factor that is not presented in the current set of objects, but does exist in the initial data set. Only such attributes are considered that do not participate in the generator and zero factors (i.e. closed set).

In order to prevent finding supergenerators (subrules) of the current generator (rule) the algorithm backtracks after detecting a class. Only such supergenerators can be avoided that are not found yet.

If no class was detected (objects of the extract belong to different classes) then we can check for a possibility to find any classification rule from the current branch (of the search tree). This is possible when at least one of the class values has a frequency bigger than or equal to the given threshold. Otherwise the algorithm can backtrack (if desired).

If the check(s) has(have) a positive result then the next factor to be included into the generator (left side of the rule) is selected by the frequency (from the frequency table). Its frequency has to be smaller than the frequency of the current extract and bigger than or equal to the given frequency threshold. The first condition prevents the inclusion of zero-zero factors (of the current extract), the second one is usual in mining frequent sets and rules. In order to find minimal generators only (not the ones between a minimal generator and closed set), the minimal one of suitable frequencies is chosen. If there is more than one factor with such frequency, just one of them is selected. The chosen factor together with the previously selected factors of the same branch forms a generator and determines a narrower (than the current) set of objects.

In order to avoid repeatedly finding already found generators, the frequency of the selected factor (the "leading" factor) is set to zero in the current frequency table. Before selecting the next leading factor, those zeroes are "brought down" from the frequency table of the previous level to the current level (except for the initial level).

The following notation is used in pseudocode of the algorithm:

X_0 – initial data table (objects*attributes);

FirstFT – initial frequency table (values*attribtues);

attr – number of attributes (excluding class);

cl– class attribute;

t – number of the step (or level) of the recursion;

X_t – set of objects (extract) at level t;

FT_t – frequency table for a set X_t;

V – „leading" frequency i.e. frequency of extract;

gen_t – generator at level t;

noclass – the truth-value of whether the class is detected for gen_t;

classpot – the truth-value showing whether there is a possibility to find any classification rule from further extracts;

gclass – class value of gen_t;

zf_t – zero factors (regarding gen_t);

excl – excluded factors (regarding $gen_t \cup zf_t$);

minfr – frequency threshold (minimal allowed number of covered objects);

Factors are given as $value_{attribute}$;

Assignments are indicated by "←" ("=" is for comparison).

The pseudocode of the algorithm is given below.

```
Algorithm for finding minimal generators with zero factors,
excluded factors and class
Given: $X_0$ , minfr >0
A1. t←0 ; $gen_0$←{} ; $zf_0$←{}
A2. find $FT_0$
A3. FirstFT←$FT_0$
A4. FOR EACH factor $h_{f=1,\dots,attr} \in FT_0$ with frequency V=min
    $FT_0[h_f] \geq$ minfr DO
```

```
A5.    $FT_0[h_f] \leftarrow 0$
A6.    make_extract(t+1; $h_f$; V)
   NEXT
End of Algorithm
PROCEDURE make_extract(t; $h_f$; V)
B1. $gen_t \leftarrow gen_{t-1} \cup h_f$
B2. $zf_t \leftarrow zf_{t-1}$ ; excl←{} ; gclass←0 ;
    noclass←true ; classpot←false
B3. separate submatrix $X_t \subset X_{t-1}$ such that $X_t=\{Xij \in X_{t-1} \mid$
    $X.f=h_f\}$
B4. find $FT_t$
B5. IF exists value clv such that $FT_t[clv_{cl}]=V$ THEN
B6.    gclass←clv ; noclass←false
B7. ELSEIF exists value clv such that $FT_t[clv_{cl}] \geq minfr$ THEN
B8.    classpot←true
    ENDIF
B9. FOR EACH empty position p ($p \in 1,\dots,attr$) in $gen_t$ DO
B10.       IF exists value h such that $FT_t[h_p]= V$ THEN
B11.           $zf_t \leftarrow zf_t \cup h_p$
B12.   ELSE
B13.       FOR EACH value v (of attribute p) DO
B14.           IF $FT_t[v_p]= 0$ THEN
B15.               IF FirstFT $[v_p]> 0$ THEN
B16.                   $excl \leftarrow excl \cup v_p$
                   ENDIF
               ENDIF
           NEXT
       ENDIF
   NEXT
B17.output $gen_t$, $zf_t$, gclass, excl, V
B18.IF V>minfr AND noclass AND classpot THEN
B19.   ZeroesDown(t)
B20.   FOR EACH $h_{u=1,\dots,attr} \in FT_t$ with frequency
       $V2=\min FT_t[h_u] \geq minfr$ and V2<V DO
B21.       $FT_t[h_u] \leftarrow 0$
B22.       make_extract(t+1; $h_u$; V2)
       NEXT
   ENDIF
END PROCEDURE
PROCEDURE ZeroesDown(t)
C1. FOR EACH factor $h_{u=1,\dots,attr} \in FT_t$ with frequency >0 DO
C2.    IF $FT_{t-1}[h_u]=0$ THEN $FT_t[h_u] \leftarrow 0$
   NEXT
END PROCEDURE
```

The initial data table X_0 and the frequency threshold minfr are given. The main program starts with initial assignments for a level of recursion t, the empty generator gen_0 and the empty set of zero factors zf_0 (step A1). Next the frequency table FT_0 for X_0 is found (A2) and this initial state is stored in FirstFT (A3). In step A4 each factor with a suitable frequency (≥minfr) is chosen as a leading factor (for inclusion into generator) in ascending order (by frequencies). The frequency of the leading factor h_f is set to zero in the frequency table FT_0 (A5) and an extract by h_f is made (A6).

While the main program makes extracts from initial data, the recursive procedure make_extract handles all deeper levels. It starts with evaluating the current generator gen_t (B1) and giving initial values for the set of zero factors zf_t, the set of excluded factors excl, class value gclass of current generator, truth-value noclass for indicating whether the class is found and truth-value classpot for indicating whether a class can be found from subsequent extracts (B2). Next the subset of objects X_t is extracted by the leading factor h_f (B3) and the corresponding frequency table FT_t is found (B4).

In B5 we check whether there is a value clv of class attribute cl with a frequency equal to the leading one V. If equal frequency is found, then the generator gen_t determines a class and in B6 its class value gclass and indicator noclass are evaluated accordingly. Otherwise we make sure whether there is at least one class value with a frequency ≥minfr (B7). In such case there is a potential to find a classification rule(s) from subsequent extracts and indicator classpot is evaluated accordingly (B8).

Step B9 goes through all empty positions (attributes without value) in current generator gen_t (as a vector) and B10 searches for the value (of that attribute) with frequency a equal to the leading one V. If one exists, it is a zero-zero factor (regarding gen_t) and it is included into the set of zero factors zf_t (B11).

In case when there is no value with leading frequency for position p (B12), we will look for its value with zero frequency (B13-B14). There can be more than one of them. If such element exists (has a non-zero value in FirstFT – step B15) then it is an excluded factor and we add it into the set of excluded factors excl (B16).

In step B17 the generator is outputted together with its frequency, possible zero factors, excluded factors and class. Several conditions can be applied to decide whether to output the current generator or not – this is a possibility to leave out generators without a class and/or without zero factors (or without new zero factors at that level). If zf_t is not empty, the rule IF gen_t THEN zf_t can be produced. If a class was detected then the rule IF gen_t THEN gclass can be produced. If `excl` is not empty, the rule(s) `IF` gen_t `THEN NOT excl` can be produced.

Step B18 checks the suitability of making a subsequent extract. If the frequency `V` is above the threshold `minfr`, then there is a possibility to find frequency that is <`V` and ≥`minfr`. If a class is not found (`noclass=true`), but there is hope to detect it from the subsequent extracts (`classpot=true`) by a longer generator(s), then we can proceed. The last two conditions can be left out if needed.

If that check (in B18) gives a positive result, then the zeroes from the frequency table of the previous level are "brought down" (B19). The procedure `ZeroesDown` goes through the current frequency table and for each factor with a frequency over zero (C1) its frequency at the previous level is checked (C2). If the latter is zero, then the factor gets a zero frequency at the current level as well (C2).

Step B20 goes through all factors that are suitable for subsequent extract i.e. with frequency smaller than the leading one (in order to prevent including zero-zero factors) and greater than or equal to the given frequency threshold `minfr`. Again the order is ascending. The frequency of the selected next factor h_u is set to zero (B21) and a recursive call to procedure `make_extract` is made with a new leading factor h_u and its frequency `V2` (B22).

Note on inheritance of excluded factors. Although excluded factors generally devolve from higher level to deeper levels, there are two reasons not to evaluate the corresponding variable `excl` by its value from the previous level (as we do for `zf` that holds zero factors). First, as pointed out before (p.111), when an attribute presented among excluded factors goes to the constitution of closed set, it is not considered as excluded factor any more. Second, for each attribute there can be more than one values that are excluded and the number of them can grow at each deeper level, thus we need to check that attribute anyway.

After the main algorithm a compression takes place. DSR-compression is suitable for the rules with identical right side. In existing realisation (by Jõgiste) only generators with class are compressed and DSR (see 3.2.6) for each class is got. Trying to compress association rules (both positive and negative) as well, additional problems can occur. We have to take into account the fact that for each generator there can be three different types of conclusions (consequents). The safest option is to compress only those for which all three are identical. But in case of excluded factors we should remind that some attributes can be left out due to belonging to the closed set. Also, comparing either zero factors or excluded factors might be technically more difficult depending on how this information is stored (because there are usually more than one factor in such set).

If we tried to compress generators without class, zero factors and excluded factors (i.e. without the consequent part of rules), then only single-item generators out of them remained.

This algorithm is realized in a master's work of Liisa Jõgiste (2014) in a bit different version: without inheritance of zero factors and without checking whether the current branch can give any classification rule. This last property is presented as a suggestion for improving the algorithm in that work.

For choosing a leading factor the author suggests the following procedure (*Ibid.*, p.70):

1. Store all the values (>0 and ≥ threshold) from data table in a list or array;
2. Sort the list by using custom sorting conditions (1) value 2) column number 3) row number, smaller value is always preferred);
3. Always take the first lead factor from a sorted list and then remove it or mark it used.

In case of non-initial levels "it is also necessary to take into consideration used lead factors from higher levels to avoid using the same factor twice. This can be solved by keeping a list of already used factors and checking possible ones against them."

Such solution complies with the presented (here) algorithm.

3.2.10.2 Example

In the following example we use data from (Quinlan, 1986). The coding of the original text values is given in Table 3.26. We use a numerical representation of these data (see Table 3.27).

The data set consists of 14 objects described by four attributes and class. The frequency threshold is set to 2.

Next the working of the algorithm (3.2.10.1) is demonstrated.

Table 3.26. The coding of the original text values

Attri-bute	Outlook	Tempe-rature	Humi-dity	Windy	Class
Code	Ou	Te	Hu	Wi	Cl
1	sunny	cool	high	true	P
2	overcast	mild	normal	false	N
3	rain	hot			

Table 3.27. Initial data table

obj	Ou	Te	Hu	Wi	Cl
1	*1*	*3*	*1*	*2*	*2*
2	*1*	*3*	*1*	*1*	*2*
3	*2*	*3*	*1*	*2*	*1*
4	*3*	*2*	*1*	*2*	*1*
5	*3*	*1*	*2*	*2*	*1*
6	*3*	*1*	*2*	*1*	*2*
7	*2*	*1*	*2*	*1*	*1*
8	*1*	*2*	*1*	*2*	*2*
9	*1*	*1*	*2*	*2*	*1*
10	*3*	*2*	*2*	*2*	*1*
11	*1*	*2*	*2*	*1*	*1*
12	*2*	*2*	*1*	*1*	*1*
13	*2*	*3*	*2*	*2*	*1*
14	*3*	*2*	*1*	*1*	*2*

Table 3.28. Initial frequency table

FT$_0$	Ou	Te	Hu	Wi	Cl
1	5	4	7	6	9
2	$4 \rightarrow 0$	6	7	8	5
3	5	4	0	0	0

Table 3.29. Extract by Ou.2

G1	2				=**4**
obj	Ou	Te	Hu	Wi	Cl
3	*2*	*3*	*1*	*2*	*1*
7	*2*	*1*	*2*	*1*	*1*
12	*2*	*2*	*1*	*1*	*1*
13	*2*	*3*	*2*	*2*	*1*

Table 3.30. Frequency table for extract by Ou.2

FT$_1$	Ou	Te	Hu	Wi	Cl
1		1	2	2	**4**
2		1	2	2	0
3		2	0	0	0

The initial frequency table FT$_0$ (for initial data) is found (see Table 3.28). From there the first factor with minimal frequency (≥threshold) is chosen: Ou.2=4 (i.e. attribute Ou with value 2 having frequency 4). Its frequency in FT$_0$ is set to zero (indicated by "→0"). Table 3.29 presents the generator Ou.2 and the extract of 4 objects containing it. Generator Ou.2 (indicated by G1) is shown in the first row, "=4" on the right of the row stands for the frequency (of the generator and extract). The extract of objects (with header row) follows. Next the frequency table FT_1 for that extract is found (see Table 3.30).

In G1 (Table 3.29) there are 3 empty positions: for attributes Te, Hu and Wi. None of those columns in FT_1 (Table 3.30) contain a frequency equal to the leading one (=4), thus there are no zero-zero factors for generator G1. There are no excluded factors as well, because the zeroes in that FT are the same as in the initial FT (Hu.3 and Wi.3), thus these factors do not exist (in the data) at all. In the class attribute column there is a frequency 4 for value 1 (and other frequencies are zeroes), so the generator determines a class (IF Ou.2 THEN Cl.1). The algorithm backtracks to the previous level, because the class was found. That prevents finding subrules of the found rule (like IF Ou.2&Te.3 THEN Cl.1).

From the initial frequency table (Table 3.28) the next factor with minimal frequency Te.1=4 is chosen. Extract by Te.1 (G2) and its frequency table are given in Table 3.31. This time the empty positions in generator (G2) are Ou, Hu and Wi. Column Hu contains a frequency equal to the leading one: Hu.2=4. That factor is a zero-zero factor regarding generator G2 and we get the rule (IF Te.1 THEN Hu.2). Other values of the same attribute are not considered as excluded factors, although their frequencies are zeroes. In the class column there is no frequency equal to 4, thus no class is detected by G2.

Table 3.31. Extract by Te.1 and its frequency table

G2		1			=4
obj	Ou	Te	Hu	Wi	Cl
5	*3*	*1*	*2*	*2*	*1*
6	*3*	*1*	*2*	*1*	*2*
7	*2*	*1*	*2*	*1*	*1*
9	*1*	*1*	*2*	*2*	*1*
FT_1					
1	1		0	2	3
2	1 ↓0		**4**	2	1
3	2→0		0	0	0

A class was not detected, the leading frequency (=4) is greater than the frequency threshold and the class column contains a frequency that is bigger than the given threshold (Cl.1=3) – thus there is a possibility to find a classification rule[50] (for

[50] If the number of classes is 2 and minfr=2 then there is no real need for such check. If the leading value is 2 then the program backtracks (line 18) due to this value. In case of higher leading value: if no class was found (noclass=true) then certainly one of the

Cl.1). Therefore, the work continues with “bringing zeroes down”, from the frequency table of the previous level (FT_0) to the current level. At level 0 (see Table 3.28) the factor Ou.2 was set to zero (indicating that it has been used for making extracts), this zero comes into the current frequency table FT_1 (in order to prevent possible further extract by that factor) (indicated by “↓0” in Table 3.31).

This frequency table contains some “suitable” frequencies – smaller than the leading one and greater than or equal to the threshold (i.e. <4 and >=2). The first (minimal) of them Ou.3=2 is chosen for inclusion into the generator and making a subsequent extract. At the current level its frequency is set to zero (“→0” in Table 3.31).

The new generator is Te.1&Ou.3=2 (G3). The corresponding extract with its frequency table is given in Table 3.32. From the frequency table we find no class and no new zero-zero factors (in addition to Hu.2 that is inherited from the previous level). The rule IF Te.1&Ou.3 THEN Hu.2 can be outputted. This is a subrule of association rule found at the previous level (IF Te.1 THEN Hu.2). If such output is not wanted then the current set of zero factors has to be compared to the one of the previous level. In case of no difference the output should be skipped. (That check is not included into the algorithm presented in 3.2.10.1, it can be contained in the output procedure.)

Table 3.32. Extract by Te.1&Ou.3 and its frequency table

G3	3	1			=2
obj	Ou	Te	Hu	Wi	Cl
5	*3*	*1*	*2*	*2*	*1*
6	*3*	*1*	*2*	*1*	*2*
FT_2					
1			0	1	1
2			**2**	1	1
3			0	0	0

As the leading frequency is equal to the threshold (=2) the algorithm backtracks (because there cannot be any frequency <2 and >=2).

From the previous frequency table FT_1 (Table 3.31) the next suitable factor is chosen: Wi.1=2. The corresponding extract and frequency table are shown in Table 3.33. From there we find the rule with inherited zero factor: IF Te.1&Wi.1 THEN Hu.2. Again, the output can be suppressed if an appropriate check is applied. This time we find an excluded factor also (Ou.1=0), that gives a rule IF

frequencies is ≥minfr (if V=3 then frequencies of classes can be 2+1; if V=4 then 3+1 or 2+2). In case of higher minfr this check can have an effect. If minfr=3 and the leading value V=4 then it is possible that all class frequencies are <minfr (2+2).

Te.1&Wi.1 THEN NOT Ou.1. The algorithm backtracks again due to the value of the leading frequency.

Table 3.33. Extract by Te.1&Wi.1 and its frequency table

G4		1		1	=2
obj	Ou	Te	Hu	Wi	Cl
6	*3*	*1*	*2*	*1*	*2*
7	*2*	*1*	*2*	*1*	*1*
FT$_2$					
1	0		0		1
2	1		**2**		1
3	1		0		0

The next choice from FT$_1$ (Table 3.31) is Wi.2=2 (see Table 3.34). This time we find both class and excluded factor in addition to the inherited zero factor: IF Te.1&Wi.2 THEN Hu.2 AND Cl.1 AND NOT Ou.2. The algorithm backtracks because of two reasons: the class is detected and the leading frequency is too low to make further extracts.

Table 3.34. Extract by Te.1&Wi.2 and its frequency table

G5		1		2	=**2**
obj	Ou	Te	Hu	Wi	Cl
5	*3*	*1*	*2*	*2*	*1*
9	*1*	*1*	*2*	*2*	*1*
FT$_2$					
1	1		0		**2**
2	0		**2**		0
3	1		0		0

At the previous level (Table 3.31) all the factors with suitable frequencies (Ou.3=2, Wi.1=2, Wi.2=2) have already been used. The zero factor (Hu.2=4) is not suitable for making an extract.

The algorithm backtracks to the initial level. Now there are two factors with "zeroed" frequencies in the frequency table FT$_0$ (Ou.2 and Te.1 in Table 3.27). No generators containing either of them will be generated during the following work.

The work continues in the same way. Table 3.35 presents all 39 generators found by the algorithm together with their frequency, possible class, zero-zero factors and excluded factors. Generators with inherited zero factors (G3, G4), are not skipped. Similarly, generators with the same excluded factors as their parent generator (G8, G9, G10), are listed here.

Table 3.35. Minimal generators, class, zero factors and excluded factors with frequency >1

	Minimal generator	**Frequ-ency**	**Class**	**Zero factors**	**Excluded factors**
1	Ou.2	4	1		
2	Te.1	4		Hu.2	
3	Te.1&Ou.3	2		Hu.2	
4	Te.1&Wi.1	2		Hu.2	Ou.1
5	Te.1&Wi.2	2	1	Hu.2	Ou.2
6	Te.3	4			Ou.3
7	Te.3&Ou.1	2	2	Hu.1	
8	Te.3&Hu.1	3			Ou.3
9	Te.3&Hu.1&Wi.2	2			Ou.3
10	Te.3&Wi.2	3			Ou.3
11	Ou.1	5			
12	Ou.1&Te.2	2			
13	Ou.1&Hu.2	2	1		Te.3
14	Ou.1&Wi.1	2			Te.1
15	Ou.1&Hu.1	3	2		Te.1
16	Ou.1&Wi.2	3			
17	Ou.3	5			Te.3
18	Ou.3&Hu.1	2		Te.2	
19	Ou.3&Wi.1	2	2		Te.3
20	Ou.3&Te.2	3			
21	Ou.3&Te.2&Wi.2	2	1		
22	Ou.3&Hu.2	3			Te.3
23	Ou.3&Hu.2&Wi.2	2	1		Te.3
24	Ou.3&Wi.2	3	1		Te.3
25	Te.2	6			
26	Te.2&Hu.2	2	1		Ou.2
27	Te.2&Wi.1	3			
28	Te.2&Wi.1&Hu.1	2			Ou.1
29	Te.2&Wi.2	3			Ou.2
30	Te.2&Wi.2&Hu.1	2			Ou.2
31	Te.2&Hu.1	4			
32	Wi.1	6			
33	Wi.1&Hu.1	3			Te.1
34	Wi.1&Hu.2	3			Te.3
35	Hu.1	7			Te.1
36	Hu.1&Wi.2	4			Te.1
37	Hu.2	7			
38	Hu.2&Wi.2	4	1		
39	Wi.2	8			

The result contains 11 generators with class (suitable for producing rules IF generator THEN class), 6 generators with zero factor(s) (suitable for producing rules IF generator THEN zero-factors) and 22 generators with excluded factor(s) (suitable for producing rules IF generator THEN NOT excluded-factor). One generator (G5) has all three types of consequents; some generators have two of them. 10 generators have none of the three. Of course, it is possible not to output them. Also it is possible filter out (or not to output) generators without class or generators without zero factors or without excluded factors.

For Class 1 (Cl.1) 8 generators were found: G1, G5, G13, G21, G23, G24, G26 and G38. Two of them are supergenerators of other generators:

- G21 (Ou.3&Te.2&Wi.2=2) is a super-generator of G24 (Ou.3&Wi.2=3);
- G23 (Ou.3&Hu.2&Wi.2=2) is a super-generator of G24 (Ou.3&Wi.2=3) and G38 (Hu.2&Wi.2=4).

Adding factor Te.2 into G24 causes a decrease in frequency (from 3 to 2), thus Te.2 in G21 is a zero-negative factor. Adding Hu.2 into G24 also decreases the frequency by 1; adding Ou.3 into G38 decreases the frequency by 2 (4-2). Both are zero-negative factors in G23, but not at the same time.

In order to remove such redundant generators (that contain zero factors), a compression is applied to each class in the preliminary result.

In case of removal of such redundant generator we can lose (together with redundant classification rule) possible association rules as well. G21 does not have neither zero factors nor excluded factors. G23 has an excluded factor Te.3. This time two of its parent generators – G22 (without class) and G24 (with class) – have the same excluded factor; thus we will not lose this information.

The redundant generators (in the preliminary result) are always found before their non-redundant subgenerators. It can be useful to take this fact into account while removing them.

In addition, the frequencies can be used to select potential redundant rules. The generators containing zero-negative factors have a smaller frequency than their subgenerators, the ones with zero-zero factors have equal frequency. Thus only the generators with a smaller or equal frequency should be checked.

After the compression 6 generators are left for Class 1. Table 3.36 lists them together with the rules based on them, using the original text values and full attribute names (according to the correspondence shown in Table 3.26).

Table 3.36. Minimal generators of class 1 (Class.P) and corresponding generator-based rules

	Minimal generator	Rules
G1	Outlook.overcast	IF Outlook.overcast THEN Class.P

G5	Temperature.cool &Windy.false	IF Temperature.cool&Windy.false THEN Class.P
		IF Temperature.cool&Windy.false THEN Humidity.normal
		IF Temperature.cool&Windy.false THEN NOT Outlook.overcast

G13	Outlook.sunny& Humidity.normal	IF Outlook.sunny&Humidity.normal THEN Class.P
		IF Outlook.sunny&Humidity.normal THEN Temperature.hot

G24	Outlook.rain &Windy.false	IF Outlook.rain&Windy.false THEN Class.P
		IF Outlook.rain&Windy.false THEN NOT Temperature.hot

G26	Temperature.mild &Humidity.normal	IF Temperature.mild&Humidity.normal THEN Class.P
		IF Temperature.mild&Humidity.normal THEN NOT Outlook.overcast

G38	Humidity.normal &Windy.false	IF Humidity.normal&Windy.false THEN Class.P

For each listed (minimal) generator (in Table 3.36) we get a zero-factor-free classification rule (IF minimal generator THEN class). For example, from G5 we can conclude: IF Temperature.cool&Windy.false THEN Class.P. Such a conclusion holds in the given data set (Table 3.27).

Generator G5 has zero factors also, this gives an association rule (IF minimal generator THEN zero-zero factor(s)): IF Temperature.cool&Windy.false THEN Humidity.normal. It means that Temperature.cool&Windy.false is always accompanied by Humidity.normal.

As we saw already, Hu.normal actually comes with Te.cool (see G2 in Table 3.35) and it descends to all supergenerators (subrules) of Te.cool (G3, G4, G5).

A generator together with its zero-zero factors forms a closed set – the set of all common factors of covered objects. No other object (in given data set) contains all those factors. Two objects covered by minimal generator Temperature.cool&Windy.false, have 3 factors in common: Temperature.cool & Windy.false & Humidity.normal.

G5 has excluded factors also, thus we get a negative association rule (IF minimal generator THEN NOT excluded-factor): IF Temperature.cool&Windy.false THEN NOT Outlook.overcast. This rule says that when it is cool and not windy then the outlook is not overcast (in the given data set).

Thus from minimal generator Temperature.cool&Windy.false (G5) we can conclude Class.P and presence of Humidity.normal and absence of Outlook.overcast.

3.2.10.3 About detecting zero factors

The problem that is not fully solved by the presented algorithm is detecting and avoiding some zero factors in (the left side of) the rules. Being a zero factor regarding the final (class-determining) itemset (left side of the rule) is not detectable at the moment the item is included into (the left side of) the rule.

Only the lastly added factor in the rule (that completed the rule) is certainly a positive (thus non-redundant) factor. Any other factor might turn out to be a zero factor. Therefore, the number of factors to check is one less than the rank (i.e. the number of factors) of the rule.

For making sure whether some factor is inessential regarding other factors in the rule, we need two frequencies of an itemset that consists of all those other factors and does not contain the factor under consideration: 1) how many objects this itemset covers, 2) how many objects from the concluded class it covers. Extracts by such itemsets have not been explored by that moment yet. They will be made later, in different branches of the search tree. The redundant generators are always found before their subgenerators.

The complexity of detecting zero factors grows with the fact that sometimes more than one of them can be excluded at the same time. For example, from the itemset ABCD we might discover zero factors A, B and C in such a way that A is a zero factor "alone", but B and C are zero factors also "together" – thus there are two zero-factor-free itemsets (covering the same objects as ABCD): BCD and AD. Another possibility is that zero factor A can be accompanied by either B or C (ZFF itemsets are CD and BD, accordingly).

In our example (see Table 3.35, p. 123) generator G23 contains two zero factors that are zero-negative "alone" i.e. they cannot be thrown out at the same time. Excluding Hu.2 gives us G24. Excluding Ou.3 gives us G38. If there were a generator Wi.2 (determining Cl.1) instead of G24 and G38, then both zero factors could be excluded from G23 at the same time.

Thus, the detection of zero factors in the rule alone is not enough, we also need to know in which combinations these factors could be excluded from the rule.

Moreover, if we figure out an effective way to detect zero factors then we also have to find a way to prevent finding appropriate ZFF rules (generators) later.

In our approach we do not use this kind of detection. It seems to be more reasonable to apply compression to the found rules. That procedure requires no access to the initial data, only the rules are compared. A shorter rule pushes out a longer one that contains all the factors of the shorter rule. Both rules belong to the same class. A longer (redundant) one is always found before the shorter one and has a smaller or equal frequency. These conditions (same class, finding order, frequency) and also comparing the ranks of the rules reduce the number of possible redundant rules regarding a certain (shorter) rule.

Such a check may be performed each time a new rule is found or after finding all rules (by the main algorithm). We use the second option – compression as a separate compact step. The first option would save storing space of rules during the work of the main algorithm.

3.2.11 Discussion

Compared to original DA we have done several improvements:

- Allowing non-overlapping rules with different length vs the same fixed length for all rules (see 3.2.3)
- Creating 3 algorithms for overlapping rules
    - Finding possibly small set of possibly short rules (3.2.5) – this is similar to usual ML rule set
    - Finding all non-redundant rules (+ DSR-compression) (3.2.7, 3.2.6) – the found set is comparable to the set of CARs[51]
    - ZFF DA (3.2.10): finding all non-redundant classification rules for all classes + positive and negative ARs

ZFF DA has similarities with both ML and ARM. Like the original DA, it can be called descriptive supervised rule discovery (SDRD), but actually does not fit under its definition. We are not aware of any other approach finding classification and association rules at the same time (as does ZFF DA).

Similarly to ML, ZFF DA finds classification rules. The purpose is different: in ML the rules are used to predict future or missing values, ZFF DA uses them for description. Nevertheless, the classification function of ZFF rules[52] (that is not in the scope of this thesis) has been studied by Fjodor Ševtšenko in his master's thesis (Ševtšenko, 2017). He has proposed three different ways to measure the prediction accuracy and has found that ZFF rules have a high predictive power.

For classification usually a possibly small set of rules is preferable. Our approach, on the contrary, is to find all non-redundant rules – that set (called DSR) gives a possibility to find different covers according to user's needs. Such approach is similar to associative classification in case of which from the set of found rules the suitable classifier is formed. Obviously, finding a bigger set is computationally more burdening, but with development of hardware this burden

[51] Class association rules (from associative classification)
[52] Such usage (for predictive purpose) can be seen as associative classification.

is decreasing continually. From a technological side, Ševtšenko (2017) has parallelized the algorithm, obtaining the growth of speed 17 times[53].

Usually the rules are found for one (or more) determined target class(es), not for all existing classes. Determining target class(es) is not a problem for ZFF DA.

Classification rules found by ZFF DA can be called class association rules (CARs)[54] as the algorithm finds them in the manner of finding ARs rather than classification rules.

Compared to "usual" ARM we can bring out two differences. First, our approach finds only exact rules (i.e. with accuracy/confidence 100%) while usually a lower threshold can be used (set by the user). The suitability of so high threshold (i.e. 100%) depends on the data. We admit that in some cases there are no exact rules (with required support/frequency) at all and then we cannot find anything. Second, as ARs are found as additional information to CARs in our case, the search in the current branch is normally discontinued after finding a CAR (while usually backtracking takes place when the support threshold is met).

Compared to the closest approach under SDRD – subgroup discovery (SD) – the difference is that ZFF DA finds all non-redundant CARs according to support threshold while SD looks for the "most interesting" rules (more exactly, the rules that have the most unusual statistical characteristics), thus skipping part of the rules that can be found by ZFF DA.

Concerning negative ARs (2.4.2.2) we have to bring out that ZFF DA finds only one kind of them (out of three possible ones), namely $X \Rightarrow \neg Y$.

All these methods – classification and association rules, subgroup discovery and associative classification – have their roots in the previous century.

Already in 2006 Ceglar and Roddick stated that „the fundamentals of association mining are now well established and there appears little current research on optimizing the performance of classic itemset identification". /…/ "The majority of current research involves the specialization of fundamental association mining algorithms to address specific issues, such as the development of incremental algorithms to facilitate dynamic dataset mining or the inclusion of additional semantics (such as time, space, ontologies, etc.) to discover, for example, temporal or spatial association rules." (Ceglar & Roddick, 2006)

[53] After completing his thesis he obtained the growth of speed 37,5 times already.

[54] This term is from associative classification where ARs with certain class as a consequent are found.

In classification also the basics have been developed many years ago already. In the latest developments the rule-based classification methods seem to be less used than neural networks for example. Especially popular is Deep Learning[55].

The hottest topic in DM is Big Data mining. Big Data[56] means not only a huge amount of data, but also different sources and formats, rapid continuous growth, variety, variability, …. "In response to the problems of analyzing large-scale data, quite a few efficient methods, such as sampling, data condensation, density-based approaches, grid-based approaches, divide and conquer, incremental learning, and distributed computing, have been presented." (Tsai, Lai, Chao, & Vasilakos, 2015).

In 2006 a representative group of awarded researches brought out 10 most influential data mining algorithms (Wu, et al., 2008). Among these we can find 3 rule-based ones: C4.5 (Quinlan, 1993), Apriori (Agrawal & Srikant, 1994) and CART (Breiman, Friedman, Stone, & Olshen, 1984). Apriori represents ARM, while the two others are intended for classification. These good old algorithms are adapted to modern technologies like distributed and parallel processing.

For instance, in frequent pattern mining, three implementations of Apriori[57] in the MapReduce[58] framework are proposed in (Lin, Lee, & Hsueh, 2012) and FP-tree[59] is combined with DH-TRIE in (Yang, Shi, Xu, Liang, & Kirsh, 2011).

[55] Deep learning refers to a class of machine learning techniques, where many layers of information-processing stages in hierarchical architectures are exploited for pattern classification and for feature or representation learning. It is in the intersections among the research areas of neural network, graphical modeling, optimization, pattern recognition, and signal processing. (Deng, 2014)

[56] According to (Borne, 2014) Big Data is defined through 10 Vs (volume, velocity, variety, veracity, validity, value, variability, venue, vocabulary, and vagueness) that "represent ten different challenges associated with the main tasks involving big data (/…/ capture, cleaning, curation, integration, storage, processing, indexing, search, sharing, transfer, mining, analysis, and visualization)".

[57] Apriori (Agrawal & Srikant, 1994) is a basic level-wise algorithm for association rule mining.

[58] MapReduce is a programming model and an associated implementation for processing and generating large data sets enabling automatic parallelization and distribution of large-scale computations, running on a large cluster (Dean & Ghemawat, 2004).

[59] FP-tree (frequent pattern tree) is a compressed structure for storing information about frequent patterns and is used by FP-growth method for finding frequent patterns; both proposed in (Han, Pei, Yin, & Mao, 2004).

For classification (in big data context) other methods than rule-based ones are rather used e.g. SVM[60] and GA[61] for predicting hazardous weather conditions (Lee, Hong, & Lee, 2014), two types of neural networks (SOM[62] and MBP[63]) for biomedical classification problems (Hasan, Shamsuddin, & Lopes, 2015), SVM on quantum computer (Rebentrost, Mohseni, & Lloyd, 2014).

Our work is presented on an algorithmic level, the choice of technological means is left to the developer. Thus, we cannot compare our work to these latest trends in DM.

Literally ZFF DA is supervised descriptive rule discovery, but has no analogue.

3.3 Universal generator of hypotheses

Here we present an idea for Universal Generator of Hypotheses (UGH) – one possible framework for gathering different descriptive tasks solvable by GH and DA. It is needed to get an overview of what is done yet and what still needs a solution.

The block diagram of Universal Generator of Hypotheses is shown in Figure 3.1. In the following “block N” will be often referred to as BN (for example B5 instead of block 5).

First of all, it is possible to define the set of observable objects (narrower than in initial data). It is shown as a logical expression (in block 2). In a sense of DA the narrowing of universal context takes place. Context is the set of qualities that describe the whole group (the ones, on the ground of which the objects are selected). The qualities common to the whole initial data set determine the universal context. In the same data set it is not possible to widen the context, it is the widest there. Thus the context can be changed only by narrowing. For that purpose the qualities on which basis to make the restriction have to be shown. Usually it is needless to observe the attributes that determine the context in the further analysis, since they describe the whole subset under examination. However, these attributes might be of interest if they can have more than one value.

[60] SVM (support vector machine (Vapnik, 1995)) is a supervised machine learning algorithm that classifies vectors in a feature space into one of two sets, given training data from the sets (Rebentrost, Mohseni, & Lloyd, 2014).
[61] GA (genetic algorithm) is an evolution-inspired computational model and global optimization technique developed by John Holland in 1975 (book titled *Adaptation in Natural and Artificial Systems*).
[62] SOM (self-organizing map), introduced in (Kohonen, 1981), is an algorithm for exploratory data analysis which provides mapping from high-dimensional features to low-dimensional features (Hasan, Shamsuddin, & Lopes, 2015).
[63] MBP (multiple back-propagation (Lopes & Ribeiro, 2001)) is a hybrid learning algorithm for multi neural network.

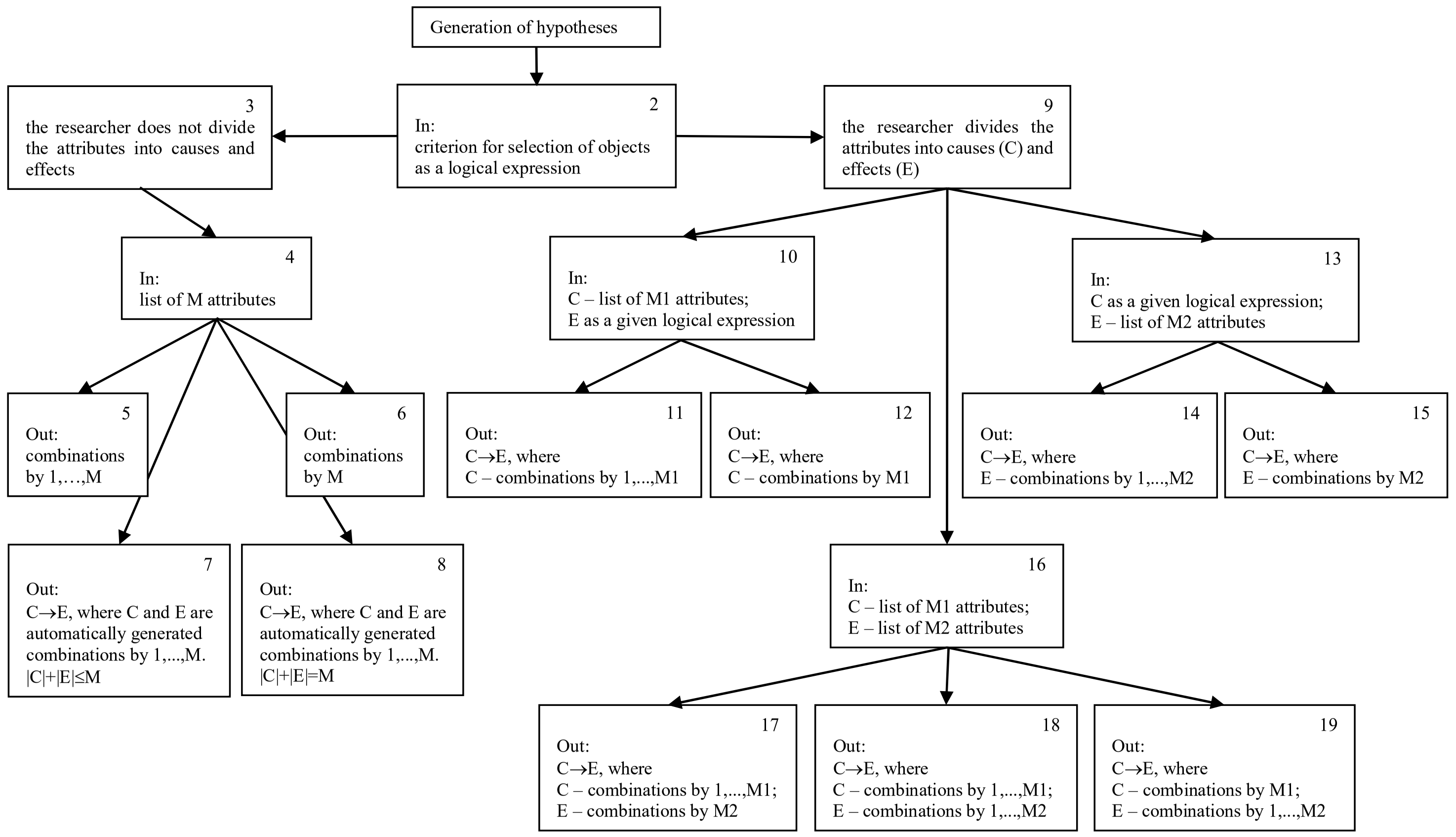
Generation of hypotheses
2 In: criterion for selection of objects as a logical expression
3 the researcher does not divide the attributes into causes and effects
4 In: list of M attributes
5 Out: combinations by 1,…,M
6 Out: combinations by M
7 Out: C→E, where C and E are automatically generated combinations by 1,...,M. |C|+|E|≤M
8 Out: C→E, where C and E are automatically generated combinations by 1,...,M. |C|+|E|=M
9 the researcher divides the attributes into causes (C) and effects (E)
10 In: C – list of M1 attributes; E as a given logical expression
11 Out: C→E, where C – combinations by 1,...,M1
12 Out: C→E, where C – combinations by M1
13 In: C as a given logical expression; E – list of M2 attributes
14 Out: C→E, where E – combinations by 1,...,M2
15 Out: C→E, where E – combinations by M2
16 In: C – list of M1 attributes; E – list of M2 attributes
17 Out: C→E, where C – combinations by 1,...,M1; E – combinations by M2
18 Out: C→E, where C – combinations by 1,...,M1; E – combinations by 1,...,M2
19 Out: C→E, where C – combinations by M1; E – combinations by 1,...,M2

Figure 3.1 Universal Generator of Hypotheses

Further there are two possibilities:

1) The researcher (user) does not partition attributes (objects' characteristics) under consideration – presented by blocks 3..8 on the left side of the scheme;
2) The researcher divides attributes into causes (C) and effects (E) – blocks 9..19 on the right side of the scheme.

In the first case (blocks 3..8) simply the enumeration of analysable attributes is given to the system, i.e. it is not required to observe all the attributes that are used for describing the objects. It is possible to find existing value combinations of those attributes (B5, B6) or associations/rules in the form of C→E where the causes C and effects E are generated automatically, each attribute can be on either side of found association – i.e. association rules are found (B7, B8). In B5 found combinations can consist of different subsets of given attributes ("combinations by 1,..,M" on the diagram). In B6 all existing value combinations of those attributes containing all given attributes ("combinations by M") are found. Similarly, the found rules can contain different subsets of given attributes (B7) or have to contain all given attributes (B8).

If the researcher distinguishes between cause-attributes (C) and effect-attributes (E) (blocks 9..19), then the rules having cause-attributes in their antecedent and effect-attributes in their consequent are found. Both C and E can be given as a list of attributes. Both can be of either fixed-length (containing all given attributes) or free-length (containing a subset of given attributes). If C or E has to contain all given attributes, then obviously zero factors cannot be left out from these combinations.

The general cases are presented in blocks 16..19. Block 18 presents the case where both sides of found rules can contain different subsets of attributes. In B17 the rules are restricted to the ones that contain all given effect-attributes in their right sides (consequents); in B19 the left sides (antecedents) have such restriction.

Either C or E can be given as a logical expression.

In blocks 10..12 the user examines what reasons lead to specified effect. The logical condition over effect-attributes determines the set of observable objects – the target class. This is the usual DA (and ML) task.

Blocks 13..15 represent a case, where the user investigates what are the effects resulting from specified cause(s). The set of observable objects is determined by a logical condition over cause-attributes. For certain C (B13) there can be only one accurate rule or no such rules at all. Therefore lower (than 1) threshold for accuracy (i.e. confidence) could be reasonable.

Determining the set of observable objects is not the same as determining the context (in B2). In case of determining a context the objects that are not contained in it, are totally left out from the analysis. In case of determining certain effects

(B10) or causes (B13) the set of observable objects are positive examples and all others are negative examples (in terms of ML).

Finding “combinations by M” from a given list of M attributes means finding all existing (with at least required frequency) “full-length” combinations i.e. conjunctions that contain all given attributes including zero factors. “Combinations by 1,…,M” can be conjunctions of any subset of given M attributes. These combinations can be either minimal – containing no zero factors (i.e. generators) or maximal – containing all possible zero factors (closed sets). Set of *minimal* combinations “by 1,…,M” usually does not contain all combinations “by M” – the ones containing zero factors are missing. Set of *maximal* combinations “by 1,…,M” contains all combinations “by M”; but it is not guaranteed to contain all combinations “by L” where L<M.

The result of B6 is a subset of the result of B5 in case of equal frequency threshold (and the same list of given attributes and the same context, of course).

Preferring zero factor free (ZFF) rules with minimal antecedent (if C has free length) and maximal consequent (if E has free length) their left sides “by 1,…,M1” (in B11, B17, B18, B7) are minimal combinations and right sides “by 1,…,M2” (in B14, B19, B18, B7) are maximal combinations.

In such case, for the same lists of M1 C-attributes and M2 E-attributes (and other equal conditions: context, frequency threshold, accuracy threshold):

- The result of B17 is a subset of the result of B18; but the result of B19 usually is not a subset of the result of B18.
- The result of B15 is a subset of the result of B14. If B14 does not contain a rule with |E|=M2 (there can be only one such rule) then the result of B15 is empty.
- The result of B12 is not a subset of the result of B11.
- If E (in B10) is a conjunction (or a single item) consisting of M2 attributes then the result of B11 is subset of the result of B17 and the result of B12 is a subset of the result of B19.
- If C (in B13) is a conjunction then the result of B14 is a subset of the result of B19. For the result of B15 there is no such relation.

If the list of attributes in B4 consists of M1 C-attributes and M2 E-attributes from B16 (M=M1+M2) then the result of B7 contains all “minimal combinations by 1,…,M1” (left sides) and thus all rules of B18. If the left side of a rule (in B7) contains less than M1 attributes, then its right side (that is a maximal combination) can contain some E-attributes, but those can be just not considered for B18. For example, if C={C1;C2;C3} and E={E1;E2} and the result of B7 contains a rule C1&C2→C3&E1, then for B18 the rule is C1&C2→E1.

The result of B7 might not contain all the rules that constitute the result of B8, because finding subrules is prevented (in B7). For example, if the set of attributes is {X;Y;Z} and there exists a rule X→Y then its subrule XZ→Y is not found in B7. In B8 XZ→Y is needed instead of X→Y.

Setting thresholds for frequency (i.e. support) and accuracy (i.e. confidence) is not shown on the scheme.

For giving examples we use Table 3.27 (p. 119). If the list of attributes (in B4) is {Ou;Te;Hu} and minimal allowed frequency (support threshold) is 2, then "combinations by 1,…,M" (all closed sets) of those attributes (B5) are:

CS1:	Hu.1=7	CS12:	Hu.2&Ou.1=2
CS2:	Hu.1&Te.2=4	CS13:	Hu.2&Ou.2=2
CS3:	Hu.1&Te.2&Ou.3=2	CS14:	Hu.2&Te.2=2
CS4:	Hu.1&Ou.1=3	CS15:	Te.2=6
CS5:	Hu.1&Ou.1&Te.3=2	CS16:	Te.2&Ou.3=3
CS6:	Hu.1&Te.3=3	CS17:	Te.2&Ou.1=2
CS7:	Hu.1&Ou.2=2	CS18:	Ou.1=5
CS8:	Hu.2=7	CS19:	Ou.3=5
CS9:	Hu.2&Te.1=4	CS20:	Ou.2=4
CS10:	Hu.2&Te.1&Ou.3=2	CS21:	Ou.2&Te.3=2
CS11:	Hu.2&Ou.3=3	CS22:	Te.3=4

There is no combination Te.1=4 because this one is not a closed set, but a part of CS9. (Generally, if A and A&B have equal frequency, then combination A is not presented.) All other possible combinations have too low frequency (for example, Hu.1&Te.3&Ou.2=1) or do not exist (for example: Ou.3&Te.3). Combinations containing all given attributes (B6) are: CS3, CS5, CS10.

Accurate rules of any length (B7) are:

- $R_1$1: Te.1 → Hu.2 (4)
- $R_1$2: Te.3 & Ou.1 → Hu.1 (2)
- $R_1$3: Ou.3 & Hu.1 → Te.2 (2)

All other possible rules have too low frequency (Hu.2&Te.3→Ou.2 (1), for example) and/or accuracy less than 1 (e.g. Te.2→Ou.3 (3) with A=3/6). Rules that contain all given attributes (B8) are: $R_1$2, $R_1$3, and additionally Te.1&Ou.3→Hu.2 (2) that is a subrule of $R_1$1 containing zero-negative factor Ou.3. Other possible subrules of $R_1$1 have frequency 1 (for example: Te.1&Ou.1→Hu.2).

Let C={Ou;Te} and E={Hu;Wi} in B16. The rules with free-length C and free-length E (B18) with frequency threshold 2 are:

- $R_2$1: Te.1 → Hu.2 (4)
- $R_2$2: Te.3 & Ou.1 → Hu.1 (2)
- $R_2$3: Ou.2 & Te.3 → Wi.2 (2)

With frequency threshold 1 we can find additionally the rules having two attributes on their right side:

- $R_2$4: Ou.2 & Te.1 $\rightarrow$ Hu.2&Wi.1 (1)
- $R_2$5: Ou.2 & Te.2 $\rightarrow$ Hu.1&Wi.1 (1)
- $R_2$6: Te.1 & Ou.1 $\rightarrow$ Hu.2&Wi.2 (1)

The result of B19 (where each antecedent has to contain all C-attributes) with frequency threshold 2 would contain $R_2$2, $R_2$3, and additionally Te.1&Ou.3→Hu.2 (2) that is not found in B18, because it is a subrule (specialization) of $R_2$1. At the same time rules $R_2$4 and $R_2$6 containing Te.1 in their antecedents are not subrules of $R_2$1, because they have different consequents.

The result of B17 (where each consequent has to contain all E-attributes) with frequency threshold 2 is empty; with frequency threshold 1 it consists of $R_2$4, $R_2$5, and $R_2$6. Other combinations of Ou and Te either do not exist (Ou.3&Te.3) or do not have an accurate consequent (Ou.1&Te.2 and Ou.3&Te.2) or have only "partial" consequent ($R_2$2, $R_2$3).

If C={Ou;Te;Wi} and E=Hu.2 in B10 then accurate rules having different number of attributes in their antecedent (B11) with frequency threshold 1 are:

- $R_3$1: Te.1 $\rightarrow$ Hu.2 (4)
- $R_3$2: Ou.1 & Te.2 & Wi.1 $\rightarrow$ Hu.2 (1)

The result of B12 (with fixed-length left side) contains rule $R_3$2 and – instead of $R_3$1 – all existing combinations containing Te.1:

- Te.1 & Ou.1 & Wi.2 $\rightarrow$ Hu.2 (1)
- Te.1 & Ou.3 & Wi.2 $\rightarrow$ Hu.2 (1)
- Te.1 & Ou.3 & Wi.1 $\rightarrow$ Hu.2 (1)
- Te.1 & Ou.2 & Wi.1 $\rightarrow$ Hu.2 (1)

If C=Te.1 and E={Hu;Wi} in B13 then the result of B14 is:

- Te.1 $\rightarrow$ Hu.2 (4)

There are no accurate rules containing all given attributes for E (B15). With accuracy threshold 50% the result of B15 would be:

- Te.1 $\rightarrow$ Hu.2 & Wi.1 (2); A=2/4
- Te.1 $\rightarrow$ Hu.2 & Wi.2 (2); A=2/4

3.3.1 Covering UGH with algorithms

The result of B5 can be found by Generator of hypotheses (described in 3.1.1) and the result of B6 is its subset.

In B7 association rules are searched for, the result is achievable using ZFF DA (see 3.2.10, the second type of rules[64]). ZFF DA covers also finding classification rules with one class attribute and free-length left side i.e. the results of B11 and B17 (and B18) with a restriction M2=1.

If we use DSR (consisting of all ZFF rules[65]) as a basis from which to select rules according to the different cases presented in UGH, we can find suitable ZFF rules only, all non-ZFF rules are not found. If the presence of zero factors in the antecedent is expected, then DSR-approach is not suitable. However, our algorithm for finding ZFF rules can be modified in different ways in order to find different rule sets presented in UGH.

ZFF DA can be adapted to allow more (than one) class attributes. For the cases where M2>1 the algorithm has to be extended in two different modes:

1) the rules with only "full" consequent (that contains all M2 attributes) are outputted – for the cases where E has to be of fixed length ("combinations by M2" in B17 (and B15));

2) the rules can have any subset of M2 attributes in their consequent ("combinations by 1, …,M2" in B18 and B19 (and B14)).

After finding a "partial" class (i.e. not all class attributes have a value) in a search tree: 1) output a rule if "partial class" is allowed (and the consequent is not exactly the same as at previous level, otherwise the rule is a specification of that previous one and is not outputted); 2) continue the search. After reaching a "full" class the program has to backtrack (after outputting the rule).

If C has to contain all given M1 attributes (B12, B19) then all frequent combinations of them have to be found and zero-zero factors cannot be left out from the antecedent. In such case both factors constituting a generator and zero-zero factors are considered as a left side and the rules are outputted only when they together include all M1 attributes (and the rule has such right side (consequent) as required). In case of such modification there is nothing to compress (by DSR-compression).

In case of B8 such rules are searched for that their antecedent and consequent together contain all given attributes. In such case shorter rules are not outputted and the program does not backtrack at that point. Only if all attributes are

[64] The given description of ZFF DA does not contain an option to find only association rules (without finding classification rules), but this is a minor change to give such possibility (not to check a class and backtrack after finding a class).
[65] Practically, the quality of DSR depends on the quality of the rule set before compression.

involved it is time to output the rule and backtrack. Again, DSR-compression (that compares rules with identical right sides) is not needed[66].

Reasonable solution for using logical expressions to determine either E (B10) or C (B13) is to create a new attribute such that its positive value corresponds to the given condition. This way more complex expressions (than just conjunction) can be used.

For B10..B12 it means that we have find the rules for positive class (only) and we can use our ZFF DA (without need to extend it to many class attributes) and other methods that consider only one class attribute.

In B13..B15 only one extract is made, by the positive value of the new attribute (corresponding to given C (B13)). As mentioned already, for certain C there can be only one accurate rule or no such rules at all. This number of rules is true for accuracy threshold (i.e. minconf) >50% as well. There is more hope to find a "partial" consequent (for B14) than a "full" consequent (B15, B14).

For cases with fixed-length E ("combinations by M2") in B17 (and B15) all different combinations can be coded and represented by one class attribute – such solution is reasonable for a small number of different combinations. This way the methods considering only one class attribute can be used. In case of fixed-length C (B19, B12) probably the number of all different combinations ("by M1") is not small enough (to be beneficial to code all existing combinations before searching the rules and decode them for presenting the result).

Our other algorithms (besides ZFF DA) are intended for single-concept learning (i.e. finding rules for one class at a time) and enable only one class attribute. To get rules for more classes (represented by one class attribute) they should be used repetitively. The left sides of the rules have free length (i.e. are "combinations by 1,…,M") in all three cases.

An algorithm for finding all possible shortest rules (see 3.2.7) together with DSR-compression gives the same rules as ZFF DA (with compression), but for one class only. The first algorithm for finding intersecting (DA) rules (3.2.5) gives a smaller set of rules trying to cover each object by a possibly small number of rules. Step-by-step approach (3.2.1) gives rules that do not intersect (i.e. additive system) – this is the additional restriction. The last two do not guarantee the minimality of combinations in the antecedents i.e. the found rules can contain zero factors in their left sides.

All three algorithms can give a result for B11 with a restriction M2=1 and when repeated, for B17 with the same restriction, thereat their individualities have to be taken into account.

[66] Compression, based on different principles (comparing rules with different consequents) is probably needed. This topic is not explored yet.

Our algorithms do not find such systems of rules where each rule has to contain all given attributes in its antecedent (i.e. “combinations by M1”). DA-System (described in 3.2.1) can do it for one class, represented by one attribute, at a time. Thus it covers the results of B12 and (in case of appropriate repeating) B19 with a restriction M2=1. As described above, ZFF DA algorithm can be modified to find such rules.

Findable rules could be pruned by frequency (support) of C ∪ E (i.e. $n(XY)$ in DA notation) as it is a monotone property. Accuracy (that corresponds to confidence in rule mining) is not monotone and therefore cannot be used the same way (as a criterion to stop the search in the current branch). Associations with too low accuracy are not considered to be rules. How to adapt our algorithms to allow lower (than 1) accuracy will be shown in 3.4.1.

3.4 Further algorithmic developments of ZFF DA

Universal Generator of Hypotheses solves the main task of DA – to find determinations (rules, associations) – in many variations. However, the formulation of the problem as given in (Chesnokov, 1982) contains giving thresholds for accuracy and completeness of findable rules. The algorithms presented in this thesis find (maximally) accurate rules and do not find completeness, thus do not consider these thresholds. In 3.4.1 we will show how to take them into account.

Solving the basic tasks of DA (listed in 2.7.5) needs – in addition to finding determinations – specifying the contributions of factors in the rules and combining initial variables.

Transforming variables is not a topic of this thesis.

The tasks dealing with essentiality (i.e. contribution to accuracy) are presented as finding the difference of accuracies of two certain rules (see (Chesnokov, 1982, pp. 62-64)). Finding these rules is accomplishable by UGH. To be sure that a needed parent rule (of a given rule) can be found, the accuracy threshold should be low (while performing that task).

Chesnokov (1982) does not expect finding all contributions of all factors in all found rules (determinations), although in “original” applications (described in 3.2.1) they are found both to accuracy and completeness. This is one possible direction for developing ZFF DA: to incorporate finding the contributions both to accuracy and completeness of each factor in every (final) rule. Besides the algorithmic developments there should be the management system supporting to tie the basic tasks.

3.4.1 Involving accuracy and completeness thresholds

According to (Chesnokov, 1982) the main task is to find in a given context all determinations from a given variable (attribute) to another given variable (class attribute) that have at least a given minimal allowable accuracy (i.e. confidence) and minimal allowable completeness (i.e. support in given class – see 2.7.6). These two thresholds can have a value between 0 and 1 (0% .. 100%).

The algorithms presented in this work find rules with maximal accuracy (i.e. A=1), but they can be easily adapted to allow lower accuracy. In our algorithms we do not find completeness, but finding it is not a problem as well as applying a threshold for completeness. In the following we will show how to find accuracy and completeness and use given thresholds for them.

Accuracy is a ratio $n(XY)/n(X)$ (see 2.7.1) where $n(X)$ is a frequency of X in the whole set (context) and $n(XY)$ in the target class. In each extract the „leading“ frequency (i.e. the number of objects in that extract) is $n(X)$ and $n(XY)$ is the frequency of Y in the corresponding frequency table. For accurate rules $n(XY)=n(X)$.

If we are looking for the rules with lower accuracy, with accuracy threshold `minA` (0<`minA`<1), then instead of checking whether $n(XY)=n(X)$ (or equivalently $n(XY)/n(X)=1$) we should check whether $n(XY)/n(X)\geq$`minA` or $n(XY)\geq$`minA`$*n(X)$. Notice that `minA`$*n(X)$ is constant in the current extract.

Adapting our algorithms to lower (than 1) accuracy is feasible. Accuracy check is identical in two presented algorithms using 3D frequency tables – the first algorithm for finding intersecting (DA) rules (p. 95) and the algorithm for finding all possible shortest rules (p. 103), performed in step `S2`:

$\texttt{Fy}_{\texttt{t}}\texttt{(A)=Fx}_{\texttt{t}}\texttt{(A)}$,

where `A` is a factor, $\texttt{Fy}_{\texttt{t}}$ contains frequencies in class Y (at level `t`) and $\texttt{Fx}_{\texttt{t}}$ contains frequencies in the whole dataset (X). In order to allow lower than 1 accuracy that check should be replaced by:

$\texttt{Fy}_{\texttt{t}}\texttt{(A)/Fx}_{\texttt{t}}\texttt{(A)}\geq\texttt{minA}$.

In ZFF DA (algorithm for finding minimal generators with zero factors, excluded factors and class – see p. 115) the accuracy is checked at lines `B5` and `B10`:

$\texttt{FT}_{\texttt{t}}\texttt{[factor]=V}$,

where $\texttt{FT}_{\texttt{t}}$ contains frequencies of all factors (at level `t`), class values are treated as other factors at this point, and `V` is the frequency of the current extract. This check should be replaced by

$\texttt{FT}_{\texttt{t}}\texttt{[factor]} \geq \texttt{V * minA}$.

In addition to changing these checks the actual accuracies should be memorized and outputted with the rules.

Completeness is a ratio $n(XY)/n(Y)$ (see 2.7.1) where $n(Y)$ is a constant for each class Y. Needed $n(Y)$-s can be counted before searching for the rules. In our algorithms we count initial frequencies of all factors into the initial frequency table. Thus $n(Y)$ is available for any Y consisting of one factor. $n(XY)$ is the frequency of Y in the extract by X, these frequencies are counted into current frequency tables. If we find many consequents for the same X (as it is possible in our ZFF DA), then each of the rules $X \rightarrow Y_i$ has its own completeness.

If we have a threshold for completeness `minC` (0<`minC`<1), then each factor has its own minimal allowed frequency to be a suitable (by completeness) consequent (Y) of a rule: $n(Y)$*`minC`. This individual frequency threshold holds in any extract. This is true both for class attributes (that are never put on the left side of the rule) and non-class attributes (that can be on either side of an association rule).

The completeness (of each possible Y) decreases (weakly) together with the frequency of extract along the branches of the search tree. If the frequency of a certain factor is too low to use it as Y, then this is true for any further extract as well. However, we cannot interrupt the search (in the current branch), as long as we have hope to find rules for any other possible class.

In our ZFF DA algorithm (3.2.10.1) we check the possibility to find any classification rule from the current branch: this is possible when at least one of the class values has a frequency bigger than or equal to the given frequency threshold (step B7, p. 115). Having individual frequency thresholds (arising from the completeness threshold) for each class we should compare the frequency of each class value with its own minimal allowed frequency.

This individual threshold (to be suitable consequent by completeness) is not intended to be the criterion by which to reject (or not) the factor to be chosen as a leading factor, because the leading factors (used for making extracts) go into the constitution of X (and are not potential Y-s in that branch any more). A factor that is not frequent enough to be the consequent, might suite to be included into X for some other Y (that has a lower individual frequency threshold). For choosing the leading factors the (usual) general frequency threshold is better.

However, if the general frequency threshold `minfr` is not given, we can use the completeness threshold `minC` to calculate it: `minfr`=$n(Y)$*`minC`, using $n(Y)$ of the class with the least number of objects (i.e. the smallest initial frequency). Applying such frequency threshold no rule with sufficient completeness is not lost due to not making an extract with a suitable frequency. The actual completeness of each possible rule (i.e. the frequency of potential Y compared to its own threshold) still has to be checked.

As said already, the accuracy is not monotone and therefore cannot be used for pruning the search tree.

Thus, each possible consequent *Y* has its own minimal allowed frequency *n*(*Y*)*`minC` (to be suitable by completeness) that holds in any extract of data; and for each extract there is a minimal allowed frequency *n*(*X*)*`minA` (to meet the accuracy requirement) that is common to all potential *Y*-s of that extract. These two thresholds are applicable when producing the rules from the current extract, but (generally) not for pruning the search tree.

4 CONCLUSIONS

The main aim of this work was to develop descriptive data mining methods GH and DA and create corresponding algorithms (based on MS theory).

Both methods had been created in Soviet time. Being separated from the Western research, they had their own underlying concepts and theories. In this work we have shown the correspondences of concepts from these methods with the ones that are widely known in data mining area. These correspondences facilitate to share our ideas as well as make findings about those well-known concepts usable for us.

DA finds classification rules (for descriptive purpose). DA has been developed in order to overcome different drawbacks typical of its different approaches.

The first direction was to find a better set of rules (than by previous approaches). Step-by-step approach reduces redundancy in case of non-intersecting rules. Our first algorithm for finding intersecting rules produces a possibly small set of (possibly short) rules.

The second approach is different: to find all non-redundant rules that form a basis from which to find a suitable cover. We call such rule set Determinative Set of Rules (DSR). This approach comprises an algorithm that produces all non-redundant rules and some of their subrules, and a compression that removes those subrules (that cannot be avoided by the main algorithm).

Our final development of DA, called zero-factor-free DA, finds DSR for all classes and additionally positive and negative association rules (at the same time). Differently from the previous approaches of DA, this one uses an algorithm that has been grown out from MONSA. Here DA and GH meet, the correspondences between different concepts helped to build the bridge.

It is important to make a difference between two types of zero factors (in DA) that cause different kinds of redundancy and can be avoided or removed by different means. We have defined 1) zero-zero factors (ZZF) that can be just left out from the antecedent (without changing the set of covered objects) and 2) zero-negative factors (ZNF) that produce a subrule of an existing rule (reducing the set of covered objects). We have found that, in order to be free of ZZFs, the left side of the rule has to be a minimal generator. In order to be free of ZNF, that minimal generator must not have a subset that defines a class.

We have shown that original GH finds all closed sets. Its base algorithm MONSA has been changed to find all minimal generators with their closed sets, "excluded factors" and class.

Elements between a closed set (CS) and its generator form a consequence for an association rule where the generator is antecedent. Such "accompanying factors" are the characteristics that always occur with the ones in the left side of the rule.

Excluded factors are such elements that do not occur in any of the objects covered by the CS, thus never accompany the elements of the antecedent. Excluded factors serve as a consequence of a negative association rule.

Elements between a CS and its generator correspond to ZZF in DA. Consequently, the factors that are redundant in the antecedent (of a DA rule), can be put into the consequence and form an association rule (this is not valid for ZNF).

Especially important finding is that a class (consequent in a classification rule) can be detected the same way as zero-zero factors. The difference is that the class attribute never occurs in the left side of the rule. This way we can find for the same antecedent both class and (other) zero-zero factors.

Putting all together, we have got a MONSA-based algorithm for ZFF DA that produces 3 kinds of non-redundant rules with the common antecedent – a minimal generator: classification rules, positive and negative association rules. It is not usual to find classification rules and association rules at the same time. As there is no target class determined, the algorithm can find rules for all existing classes intermittently – this is different from all previous approaches of DA. DSR-compression is still needed for classification rules.

By finding classification rules for all classes, ZFF DA corresponds to multiple-concept learning. As these rules are used for descriptive purpose, ZFF DA (like original DA) can be called descriptive supervised rule discovery, but actually does not fit under its definition. The simultaneous finding of ARs makes the placement more complicated.

As a CS with all its minimal generators forms an equivalence class (EC), it was straightforward to create an algorithm for finding all ECs (although previously we did not have such goal).

We have created Universal Generator of Hypotheses (UGH) – one possible framework for gathering different tasks solvable by GH and DA. We have shown to which extent these tasks are covered by ZFF DA and other existing methods.

Our work is presented on an algorithmic level, the choice of technological means is left to the developer. Presented methods are not intended for big data analysis. However, with a continual development of hardware and technological means, its capability increases.

Our approach finds only exact rules, therefore, depending on data and frequency threshold, we can find no rules. The idea for allowing a lower accuracy (as well as setting a threshold for completeness) has been described (without presenting it in the form of algorithm).

Redundancy is defined as being a subrule of another rule, for rules with equal consequent. Redundancy in case of non-identical right sides needs exploration.

We do not use interestingness measures for excluding non-interesting rules. The depth of the rules is determined by the frequency threshold. However, the presented algorithms enable to incorporate for stopping the search such measures that are downward closed.

In ZFF DA the search in the current branch is stopped after finding a classification rule (CAR), thus ARs that are located deeper in such branch are not found. This specialty can be easily removed, if needed.

ZFF DA finds only one type of negative ARs (out of three).

In the future, we plan to discuss about rationality to find all contributions of all factors in found rules and, if it is rational, to design a corresponding algorithm. Another open issue is how to compare the results of different (sub)sets of data.

We have brought out possible directions for further development of ZFF DA based on UGH. I believe that the MONSA-based ZFF DA algorithm has a potential to incorporate different new possibilities (as described in this work).

4.1 Directions for further research

This thesis already contains ideas for further developments of the presented ZFF DA:

1) Extend the algorithm for considering many class attributes in two different ways: a) class is defined by all class attributes; b) class can be defined by a subset of class attributes as well (see 3.3.1)
2) Modify the algorithm for finding the rules with fixed rank (3.3.1)
3) Modify the algorithm for finding the rules where antecedent and consequent together contain all given attributes (3.3.1)
4) Involve thresholds for accuracy and completeness (see 3.4.1)

Ideas that have not been elaborated yet:

5) Finding contributions both to accuracy and completeness for all factors in (final) rules (mentioned in 3.4)
6) Explore the redundancy in case of (association) rules having non-identical consequents (mentioned in 3.3.1)
7) Design the management system for integrating different tasks of UGH (mentioned in 3.4)

As the thesis contains unpublished material, we have a potential to publish:

8) Algorithm for finding equivalence classes (3.1.6)
9) Part concerning excluded factors in ZFF DA (3.1.7, 3.2.10)

Topics to explore:

10) Negative ARs (in connection with excluded factors)

11) Algorithms/approaches for finding ECs (or just generators and closed sets together) (in connection with EC algorithm)
12) Using multiple minimum supports[67] (how it is related to the idea of involving accuracy and completeness thresholds presented in 3.4.1)
13) Discriminative patterns, emerging patterns, contrast sets and other approaches that deal with comparison of classes (to find a way to compare the results of different (sub)sets of data)

[67] Different minimum support threshold is set either for each single item (Liu, Hsu, & Ma, 1999) or for each level of hierarchy (Han & Fu, 1995) or taxonomy (Srikant & Agrawal, 1995).

REFERENCES

Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules. *Proceedings of 20th International Conference on Very Large Data Bases*, (pp. 487-499). Santiago, Chile.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International conference on Management of Data (SIGMOD'93)* (pp. 207-216). ACM Press.

Anshakov, O., Skvortsov, D., Finn, V., & Ivashko, V. (1987). Logical Tools of the JSM-Method of Automatic Generation of Hypotheses: Basic Concepts and System of Rules. *Nauchn. Tekhn. Inform. Ser. 2*(9), 10-18 (in Russian).

Antonie, L., Li, J., & Zaiane, O. (2014). Negative association rules. In C. Aggarwal, & J. Han, *Frequent pattern mining* (pp. 135-145). Springer International Publishing.

Antonie, M.-L., & Zaïane, O. (2004). An associative classifier based on positive and negative rules. *Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (pp. 64 - 69). Paris, France: ACM.

Atzmüller, M., & Puppe, F. (2006). SD-Map—a fast algorithm for exhaustive subgroup discovery. *European Conference on Principles of Data Mining and Knowledge Discovery. LNCS 4213*, pp. 6-17. Springer.

Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., & Lakhal, L. (2000). Mining minimal non-redundant association rules using frequent closed itemsets. *CL'2000 international conference on Computational Logic*, *LNCS 1861*, pp. 972-986.

Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., & Lakhal, L. (2000). Mining Frequent Patterns with Counting Inference. *ACM SIGKDD Explorations, 2*(2), 66–75.

Bay, S., & Pazzani, M. (2001). Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery, 5*(3), 213-246.

Bayardo Jr., R., & Agrawal, R. (1999). Mining the most interesting rules. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 145-154). ACM.

Berlanga, F., del Jesus, M., González, P., Herrera, F., & Mesonero, M. (2006). Multiobjective evolutionary induction of subgroup discovery fuzzy rules: a case study in marketing. In P. Perner (Ed.), *Advances in Data Mining.*

Applications in Medicine, Web Mining, Marketing, Image and Signal Mining. ICDM 2006. LNAI 4065, pp. 337–349. Springer.

Borne, K. (2014). *Top 10 big data challenges a serious look at 10 big data v's.* Retrieved May 2017, from https://mapr.com/blog/top-10-big-data-challenges-serious-look-10-big-data-vs

Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). *Classification and regression trees.*

Bringmann, B., Nijssen, S., & Zimmermann, A. (2009). Pattern-Based Classification: A Unifying Perspective. In A. Knobbe, & J. Fürnkranz (Ed.), *From Local Patterns to Global Models: Proceedings of the ECML/PKDD-09 Workshop (LeGo-09)*, (pp. 36-50). Bled, Slovenia.

Brownlee, J. (2017). *Overfitting and Underfitting With Machine Learning Algorithms*. Retrieved May 2017, from Machine Learning Mastery: http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

Bykowski, A., & Rigotti, C. (2003). DBC: a condensed representation of frequent patterns for efficient mining. *Information Systems, 28*(8), 949-977.

Carmona, C. J., González, P., del Jesus, M. J., & Herrera, F. (2014). Overview on evolutionary subgroup discovery: analysis of the suitability and potential of the search performed by evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 4*(2), 87-103.

Carmona, C., González, P., del Jesus, M., & Herrera, F. (2010). NMEEF-SD: Non-dominated multi-objective evolutionary algorithm for extracting fuzzy rules in subgroup discovery. *IEEE Transactions on Fuzzy Systems, 18*(5), 958-970.

Ceglar, A., & Roddick, J. (2006). Association mining. *ACM Computing Surveys (CSUR), 38*(2), 5.

Cheng, H., Yan, X., Han, J., & Hsu, C.-W. (2007). Discriminative frequent pattern analysis for effective classification. *IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007.* (pp. 716-725). IEEE.

Chesnokov, S. V. (1980a). *Determination-analysis of social-economic data in dialogical regime (Preprint).* Moscow: All-Union Institute for Systems Research (in Russian) .

Chesnokov, S. V. (1980b). Determinacy analysis of social-economic data. *Sociological Studies, #3*, 179-189 (in Russian).

Chesnokov, S. V. (1982). *Determinacy analysis of social-economic data.* Moscow: Nauka (in Russian).

Chesnokov, S. V. (1996). Determinacy Analysis and the search for diagnostic criteria in medicine (the case of comprehensive ultrasonography). *Ultrasonic Diagnostics, #4*, 42-47 (in Russian).

Chesnokov, S. V. (2002). *Determinacy Analysis of Socio-Economic Data. Illustrative Materials to Lectures. Lecture 2: Rules. Lecture 3: Systems of Rules.* Moscow: Lomonosov Moscow State University, Faculty of Economics (unpublished, in Russian).

Clark, P., & Niblett, T. (1987). Induction in noisy domains. *Progress in Machine Learning - Proceedings of EWSL 1987*, (pp. 11-30). Bled, Yugoslavia.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Journal of Machine Learning, 3*(4), 261-283.

Context. (1999a). *DA-system 4.0. User's Guide, ver. 1.0, 1998-1999.* (in Russian).

Context. (1999b). *DA-system 4.0, version 4.0 for Windows 95, Windows 98 and Windows NT. Questions and Answers. DA-system and Technology of Data Analysis.* (in Russian).

DALSolution. (2007, 02 27). *DALSolution software and technology. Questions and Answers*. Retrieved from http://www.dalsolution.com/faq.htm, 27.02.2007

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *OSDI'04 Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation. 6*, pp. 137-149. CA, USA: USENIX Association Berkeley.

del Jesus, M. J., González, P., Herrera, F., & Mesonero, M. (2007). Evolutionary fuzzy rule induction process for subgroup discovery: A case study in marketi. *IEEE Transactions on Fuzzy Systems 15, 15*(4), 578-592.

Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing, 3*.

Dong, G., & Li, J. (1999). Efficient Mining of Emerging Patterns: Discovering Trends and Differences. *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 43-52). ACM.

Dong, X., Sun, F., Han, X., & Hou, R. (2006). Study of Positive and Negative Association Rules Based on Multi-confidence and Chi-Squared Test. In X. Li, O. Zaïane, & Z. Li (Ed.), *ADMA 2006. LNCS (LNAI) 4093*, pp. 100–109. Heidelberg: Springer.

Dunham, M. H. (2002a). *Data Mining: Introductory and Advanced Topics.* Prentice Hall.

Dunham, M. H. (2002b). *DATA MINING: Introductory and Advanced Topics. Part II. Companion slides for the text by Dr. M.H.Dunham, Data Mining,Introductory and Advanced Topics, Prentice Hall, 2002.*

Fan, W., Geerts, F., Li, J., & Xiong, M. (2011). Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering, 23*(5), 683-698.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery: An Overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining* (pp. 1-36). AAAI Press/ The MIT Press.

Freitas, A. (2000). Understanding the crucial differences between classification and discovery of association rules: a position paper. *ACM SIGKDD Explorations Newsletter, 2*(1), 65-69.

Fürnkranz, J., & Kliegr, T. (2015). A brief overview of rule learning. *International Symposium on Rules and Rule Markup Languages for the Semantic Web* (pp. 54-69). Springer International Publishing.

Fürnkranz, J., Gamberger, D., & Lavrač, N. (2012). *Foundations of rule learning.* Springer Science & Business Media.

Gamberger, D., & Lavrač, N. (2002). Expert-guided subgroup discovery: Methodology and application. *Journal of Artificial Intelligence Research, 17*(1), 501-527.

Gams, M., & Lavrac, N. (1987). Review of five empirical learning systems within a proposed schemata. In I. Bratko, & N. Lavrac (Ed.), *Progress in Machine Learning, Proceedings of EWSL 87* (pp. 46-66). Wilmslow: Sigma Press.

García-Borroto, M., Martínez-Trinidad, J. F., & Carrasco-Ochoa, J. A. (2014). A survey of emerging patterns for supervised classification. *Artificial Intelligence Review, 42*(4), 705-721.

Geng, L., & Hamilton, H. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR), 38*(3), 9.

Grosskreutz, H., & Rüping, S. (2009). On subgroup discovery in numerical domains. *Data mining and knowledge discovery, 19*(2), 210-226.

Grosskreutz, H., Rüping, S., & Wrobel, S. (2008). Tight optimistic estimates for fast subgroup discovery. *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 440-456). Springer.

Gu, T., Wu, Z., Tao, X., Pung, H. K., & Lu, J. (2009). epsicar: An emerging patterns based approach to sequential, interleaved and concurrent activity recognition. *IEEE International Conference on Pervasive Computing and Communications, 2009. PerCom 2009.* (pp. 1-9). IEEE.

Han, J., & Fu, Y. (1995). Discovery of multiple-level association rules from large databases. *Proceedings of the 21st VLDB Conference* (pp. 420-431). Morgan Kaufmann Publishers Inc.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM SIGMOD international conference on management of data* (pp. 1-12). ACM.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery, 8*(1), 53-87.

Hasan, S., Shamsuddin, S., & Lopes, N. (2015). Soft computing methods for big data problems. In Y. Cai, & S. See, *GPU Computing and Applications* (pp. 235-247). Singapore: Springer.

Helal, S. (2016). Subgroup Discovery Algorithms: A Survey and Empirical Evaluation. *Journal of Computer Science and Technology, 31*(3), 561-576.

Herrera, F., Carmona, C., González, P., & del Jesus, M. (2011). An overview on subgroup discovery: foundations and applications. *Knowledge and information systems, 29*(3), 495-525.

Huang, Z., Zhou, Z., He, T., & Wang, X. (2011). ACAC: Associative Classification based on All-Confidence. *IEEE International Conference on Granular Computing (GrC)* (pp. 289-293). IEEE.

Jõgiste, L. (2014). *Prototyping of Zero-factor based DA.* Master's Thesis, Tallinn University of Technology, IT Faculty, Tallinn.

Kavšek, B., & Lavrač, N. (2006). APRIORI-SD: Adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence, 20*(7), 543-583.

Klösgen, W. (1996). Explora: A multipattern and multistrategy discovery assistant. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy, *Advances in knowledge discovery and data mining* (pp. 249-271). AAAI Press.

Klösgen, W., & May, M. (2002). Spatial Subgroup Mining Integrated in an Object-Relational Spatial Database. *Principles of Data Mining and Knowledge Discovery. PKDD 2002. LNAI 2431*, pp. 275-286. Springer.

Kohonen, T. (1981). Automatic formation of topological maps of patterns in a self-organizing system. *Proc. 2nd Scandinavian Conf. on lmage Analysis*, (pp. 214-220). Espoo, Finland.

Kryszkiewicz, M., & Gajek, M. (2002b). Why to apply generalized disjunction-free generators representation of frequent patterns? In M.-S. Hacid, Z. W. Ras, D. A. Zighed, & Y. Kodratoff (Ed.), *ISMIS 2002. LNAI 2366*, pp. 383-392. Berlin Heidelberg: Springer-Verlag.

Kundu, G., Islam, M., Munir, S., & Bari, M. (2008). ACN: An Associative Classifier with Negative Rules. *11th IEEE International Conference on Computational Science and Engineering* (pp. 369-375). IEEE.

Kuusik, R. (1987). Generator Hypotheses for Qualitative Data. *Transactions of Tallinn Technical University*(645), 141-148 (in Russian).

Kuusik, R. (1988). On new qualitative datanalysis methods and its applications. *Infotechnology and exact economics. Proceedings of the Respublican Scientific Seminar, II*, 287-290 (in Estonian).

Kuusik, R. (1993). The Super-Fast Algorithm of Hierarchical Clustering and the Theory of Monotone Systems. *Transactions of Tallinn Technical University, 734*, 37-62.

Kuusik, R. (1995). Extracting of all maximal cliques: monotone system approach. *Proceedings of the Estonian Academy of Sciences. Engineering, 1*(2), 113-138.

Kuusik, R., & Lind, G. (2008, May). Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques. *WSEAS Transactions on Information Science and Applications, 5*(5), 599-611.

Kuusik, R., & Lind, G. (2010). Some Developments of Determinacy Analysis. *Advanced Data Mining and Applications: The 6th International Conference on Advanced Data Mining and Applications (ADMA2010), Chongqing, China, November 19-21, 2010. LNAI 6440*, pp. 593-602. Berlin Heidelberg: Springer-Verlag.

Kuusik, R., & Lind, G. (2011). New Developments of Determinacy Analysis. In J. Tang, I. King, L. Chen, & J. Wang (Ed.), *Advanced Data Mining and Applications - 7th International Conference: ADMA 2011, Beijing, China, December 17-19, 2011. II; LNCS 7121*, p. 223−236. Springer.

Kuusik, R., & Lind, G. (2012). An Effective Inductive Learning Algorithm for Extracting Rules. In F. L. Gaol, & Q. V. Nguyen (Ed.), *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science, 2: CACS 2011, Bali, Indonesia, November 15-17, 2011. AISC 145*, pp. 339-344. Berlin Heidelberg: Springer-Verlag.

Kuusik, R., Lind, G., & Võhandu, L. (2004). Frequent pattern mining as a clique extracting task. In N. Callaos, V. Lefebvre, E. Hansen, T. Dickopp, & J. Su (Ed.), *The 8th World Multi-Conference on Systemics, Cybernetics and Informatics, July 18-21, 2004 - Orlando, Florida, USA, SCI 2004 Proceedings. IV*, p. 425−428. Orlando, Florida, USA: International Institute of Informatics and Systemics.

Lavrač, N., Kavšek, B., Flach, P., & Todorovski, L. (2004). Subgroup discovery with CN2-SD. *Journal of Machine Learning Research, 5*(Feb), 153-188.

Lee, J., Hong, S., & Lee, J.-H. (2014). An efficient prediction for heavy rain from big weather data using genetic algorithm. *Proceedings of the International Conference on Ubiquitous Information Management and Communication* (pp. 25:1–25:7). ACM.

Li, J., Liu, G., & Wong, L. (2007). Mining statistically important equivalence classes and delta-discriminative emerging patterns. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 430-439). ACM.

Li, W., Han, J., & Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules. *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM)* (pp. 369-376). IEEE.

Li, X., Qin, D., & Yu, C. (2008). ACCF: Associative Classification Based on Closed Frequent Itemsets. *Proceedings of the Fifth International Conference on Fuzzy Systems and Knowledge Discovery. 2*, pp. 380-384. IEEE.

Lin, M.-Y., Lee, P.-Y., & Hsueh, S.-C. (2012). Apriori-based frequent itemset mining algorithms on MapReduce. *Proceedings of the International Conference on Ubiquitous Information Management and Communication* (pp. 76:1–76:8). ACM.

Lind, G., & Kuusik, R. (2007). Some Ideas for Determinacy Analysis Realisation. *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing. Palma de Mallorca, Spain, August 29-31, 2007* (pp. 185-190). ACTA Press.

Lind, G., & Kuusik, R. (2008a, October). New developments for Determinacy Analysis: diclique-based approach. *WSEAS Transactions on Information Science and Applications, 5*(10), 1458-1469.

Lind, G., & Kuusik, R. (2008b). Some Problems in Determinacy Analysis Approaches Development. *Proceedings of the 2008 International Conference on Data Mining (DMIN 2008), Las Vegas, Nevada, USA, July 14-17, 2008. Volume I*, pp. 102-108. CSREA Press.

Lind, G., & Kuusik, R. (2012). An Idea for Universal Generator of Hypotheses. In L. Maciaszek, A. Cuzzocrea, & J. Cordeiro (Ed.), *Proceedings of ICEIS 2012: the 14th International Conference on Enterprise Information Systems, Wrocław, Poland, 28 June – 1 July. Volume 1*, pp. 169-174. Portugal: SciTePress.

Lind, G., & Kuusik, R. (2016). Algorithm for Finding Zero Factor Free Rules. *Man-Machine Interactions 4: 4th International Conference on Man-Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6-9, 2015. AISC 391*, pp. 421-435. Springer.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating Classification and Association Rule Mining. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (pp. 80-86). AAAI Press.

Liu, B., Hsu, W., & Ma, Y. (1999). Mining association rules with multiple minimum supports. *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Ddiscovery and Data Mining* (pp. 337-341). ACM.

Liu, Q., Wang, W., Deng, S., & Dong, G. (2011). An Equivalence Class Based Clustering Algorithm for Categorical Data. *IMMM 2011 : The First International Conference on Advances in Information Mining and Management*, (pp. 127-130).

Lopes, N., & Ribeiro, B. (2001). Hybrid learning in a multi neural network architecture. *INNS-IEEE International Joint Conference on Neural Networks, IJCNN'01* (pp. 2788–2793). Washington D.C., USA: IEEE.

Luelsdorff, P., & Chesnokov, S. (1996). Determinacy Form as the Essence of Language. *Prague Linguistic Circle, 2*, 205-234.

Madjarov, G., Kocev, D., Gjorgjevikj, D., & Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition, 45*(9), 3084-3104.

MathWorld--A Wolfram Web Resource, created by Eric W. Weisstein. (2016). Retrieved February 2016, from http://mathworld.wolfram.com/

Michalski, R. S. (1969). On the Quasi-Minimal Solution of the Covering Problem. *Proceedings of FCIP 69*, *A3*, pp. 125-128. Bled, Yugoslavia.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (1984). *Machine Learing: An Artificial Intelligence Approach.* Berlin Heidelberg New York Tokio: Springer-Verlag.

Michie, D., Spiegelhalter, D., & Taylor, C. (1994). *Machine Learning, Neural and Statistical Classification.*

Mielikäinen, T. (2006). Transaction databases, frequent itemsets, and their condensed representations. In F. Bonchi, & J.-F. Boulicaut (Ed.), *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Porto, Portugal, October 3, 2005, Revised Selected and Invited Papers. Lecture Notes in Computer Science 3933*, pp. 139-164. Springer.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

Morishita, S., & Sese, J. (2000). Traversing itemset lattice with statistical metric pruning. *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, (pp. 226–236).

Mueller, M., Rosales, R., Steck, H., Krishnan, S., Rao, B., & Kramer, S. (2009). Subgroup discovery for test selection: a novel approach and its application to breast cancer diagnosis. *Advances in Intelligent Data Analysis VIII. IDA 2009. LNCS 5772*, pp. 119-130. Springer.

Mullat, J. E. (1971). On the Maximum Principle for some Set Functions. *Proceedings of the Tallinn Technical University*(313), 37-44. Retrieved 02 2016, from http://www.datalaundering.com/download/modular.pdf

Mullat, J. E. (1976). Extremal Subsystems of Monotonic Systems. I. *Automation and Remote Control, 37*(5), 758-766. Retrieved 02 2016, from www.datalaundering.com/download/extrem01.pdf

Mullat, J. E. (1977). Extremal Subsystems of Monotonic Systems. II. *Automation and Remote Control, 37*(8), 1286-1294. Retrieved 02 2016, from www.datalaundering.com/downlaods/extrem02.pdf

Nguyen, D., Vo, B., & Le, B. (2014). Efficient strategies for parallel mining class association rules. *Expert Systems with Applications, 41*(10), 4716-4729.

Nguyen, L., Vo, B., Hong, T.-P., & Thanh, H. (2013). CAR-Miner: An efficient algorithm for mining class-association rules. *Expert Systems with Applications, 40*(6), 2305-2311.

Niu, Q., Xia, S.-X., & Zhang, L. (2009). Association Classification Based on Compactness of Rules. *Proceedings of the Second International Workshop on Knowledge Discovery and Data Mining - WKDD* (pp. 245-247). IEEE.

Novak, P., Lavrač, N., & Webb, G. (2009). Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research, 10*(Feb), 377-403.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1998). Pruning Closed Itemset Lattices for Association Rules. *Proceedings of the BDA Conference*, (pp. 177-196).

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. *Database Theory - ICDT'99. LNCS 1540*, pp. 398-416. Springer.

Pruks, M. (2014). *Realization of Equivalence Class Based Clustering.* Master's Thesis, Tallinn University of Informatics, IT Faculty, Tallinn.

Quinlan, J. R. (1984). Learning efficient classification procedures and their application to chess and games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning. An Artificial Intelligence Approach* (pp. 463-482). Berlin Heidelberg New York Tokyo: Springer-Verlag.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 11*(1), 81-106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* San Mateo: Morgan Kaufmann Publishers.

Rebentrost, P., Mohseni, M., & Lloyd, S. (2014). Quantum support vector machine for big data classification. *Physical review letters, 113*(13), 130503.

Roosmann, P., Võhandu, L., Kuusik, R., Treier, T., & Lind, G. (2008). Monotone Systems approach in Inductive Learning. *International Journal of Applied Mathematics and Informatics, 2*(2), 47−56.

Ševtšenko, F. (2017). *Zero Factors Free Rules Algorithm: The Study of Classification Function.* Master's Thesis, Tallinn University of Technology, Faculty of Information Technology, Tallinn.

Srikant, R., & Agrawal, R. (1995). Mining generalized association rules. *Proceedings of the 21st VLDB Conference* (pp. 407-419). Morgan Kaufmann Publishers Inc.

Stover, C. (2016). *Monotonic Function*. Retrieved February 2016, from MathWorld--A Wolfram Web Resource, created by Eric W. Weisstein: http://mathworld.wolfram.com/MonotonicFunction.html

Thabtah, F., & Cowling, P. (2007). A greedy classification algorithm based on association rule. *Applied Soft Computing, 7*(3), 1102-1111.

Thabtah, F., Cowling, P., & Peng, Y. (2004). MMAC: A new multiclass, multi-label associative classification approach. *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM '04)* (pp. 217-224). Brighton, UK: IEEE.

Thabtah, F., Cowling, P., & Peng, Y. (2005). MCAR: Multi-class classification based on association rule approach. *Proceedings of the 3rd IEEE*

International Conference on Computer Systems and Applications (pp. 33-I). IEEE.

Tsai, C.-W., Lai, C.-F., Chao, H.-C., & Vasilakos, A. V. (2015). Big data analytics: a survey. *Journal of Big Data, 2*(1), 21.

Tsoumakas, G., & Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehouse and Mining, 3*(3), 1–13.

Vapnik, V. (1995). *The Nature of Statistical Learning.* Springer.

Veloso, A., Meira, W., Gonçalves, M., & Zaki, M. (2007). Multi-label Lazy Associative Classification. *Proceedings of the Principles of Data Mining and Knowledge Discovery - PKDD* (pp. 605-612). Springer.

Veselov, A., Deza, V., & Podrabinovich, A. (1980). Computation of characteristics of determination relationships. In *Methodology of comprehensive analysis of socio-economic systems. Collected papers* (pp. 94-99). Moscow: the Institute for Systems Studies (in Russian).

Võhandu, L., Kuusik, R., Torim, A., Aab, E., & Lind, G. (2006). Some Monotone Systems Algorithms for Data Mining. *WSEAS Transactions on Information Science and applications, 4*(3), 802−809.

Wang, H., Zhang, X., & Chen, G. (2008). Mining a complete set of both positive and negative association rules from large databases. *Advances in Knowledge Discovery and Data Mining* (pp. 777-784). Springer.

Wang, X., Yue, K., Niu, W., & Shi, Z. (2011). An approach for adaptive associative classification. *Expert Systems with Applications, 38*(9), 11873-11883.

Wedyan, S. (2014). Review and comparison of associative classification data mining approaches. *International Journal of Computer, Information, Systems and Control Engineering, 8*(1), 34-45.

Weisstein, E. W. (2016a). *Monotone Increasing*. Retrieved February 2016, from MathWorld--A Wolfram Web Resource: http://mathworld.wolfram.com/MonotoneIncreasing.html

Weisstein, E. W. (2016b). *Monotone Decreasing*. Retrieved February 2016, from MathWorld--A Wolfram Web Resource: http://mathworld.wolfram.com/MonotoneDecreasing.html

Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD)* (pp. 78-87). Springer-Verlag.

Wu, X., Kumar, V., Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., . . . Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and information systems, 14*(1), 1-37.

Yang, L., Shi, Z., Xu, L. D., Liang, F., & Kirsh, I. (2011). DH-TRIE frequent pattern mining on Hadoop using JPA. *2011 IEEE International Conference on Granular Computing*, (pp. 875-878).

Yin, X., & Han, J. (2003). CPAR: Classification based on predictive association rules. *In Proceedings of the 2003 SIAM International Conference on Data Mining* (pp. 331-335). SIAM.

Zadeh, L. (1965). Fuzzy sets. *Information and Control, 8*(3), 338-353.

Zaki, M. J. (2004). Mining Non-Redundant Association Rules. (Fayyad, Mannila, & Ramakrishnan, Eds.) *Data Mining and Knowledge Discovery, 9*, 223-248.

Zaki, M. J., & Hsiao, C.-J. (1999). *CHARM: An efficient algorithm for closed association rule mining.* Technical Report 99-10, Rensselaer Polytechnic Institute, Department of Computer Science.

Zaki, M. J., & Hsiao, C.-J. (2002). CHARM: An Efficient Agorithm for Closed Itemset Mining. *Proceedings of the Second SIAM International Conference on Data Mining*, *2*, pp. 457-473.

Zhang, M.-L., & Zhou, Z.-H. (2014). A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering, 26*(8), 1819-1837.

KOKKUVÕTE

Andmekaevandamist defineeritakse kui peidetud informatsiooni leidmist andmebaasist. Andmekaevandamise kaks kõrgtaseme eesmärki on ennustamine ja kirjeldamine. Kirjeldamine on viis uuritavate andmete omaduste lähemaks uurimiseks (mitte uute omaduste ennustamiseks).

Nimetatud eesmärkide saavutamiseks kasutatakse mitmeid andmekaevandamisülesandeid, sh klassifitseerimine ja assotsiatsioonireeglid. Klassifitseerimisel seatakse andmed vastavusse eelnevalt defineeritud klassidega (gruppidega). See meetod pärineb masinõppest, andmekaeves kasutatakse seda peamiselt ennustamiseks. Assotsiatsioonireeglite kaevandamine paljastab seoseid andmete vahel, identifitseerides kindlat tüüpi kooslusi. Seda peetakse kirjeldavaks ülesandeks.

Mõlemad ülesanded annavad tulemuseks IF-THEN reeglid. Vahe on selles, et klassifikatsioonireeglid leitakse vaid ettemääratud klassi(de)le ja klassitunnus(ed) on eraldatud teistest tunnustest, assotsiatsioonireeglite korral aga võivad kõik atribuudid esineda emmal-kummal reegli poolel ja reegli järelduse osa pole (üldjuhul) ette määratud.

Enamasti lahendatakse neid ülesandeid eraldi ja erinevatel eesmärkidel. Meie loodud lähenemine (nullfaktorivaba determinatsioonanalüüs) leiab korraga mõlemat tüüpi reegleid (kirjeldamise eesmärgil). See erineb teistest meetoditest, milles kombineeritakse klassifitseerimist ja assotsiatsioonireegleid.

Käesolev töö esitab kahe kirjeldava andmekaeve meetodi arendusi. Nendeks meetoditeks on determinatsioonanalüüs (DA) ja hüpoteeside generaator (HG).

Mõlemad meetodid on loodud Nõukogude ajal, eraldatuna Lääne teadusest. Seetõttu põhinevad need omadel mõistetel ja teooriatel. Käesolevas töös näitame nende meetodite poolt kasutatavate mõistete vastavusi üldtuntud andmekaeve mõistetega. Sellised vastavused hõlbustavad meil oma ideid jagada, aga ka ära kasutada teadmisi nende laiemalt tuntud mõistete kohta.

DA on meetod reeglite analüüsimiseks. Püütakse vastata küsimustele: „Kes nad on?“, „Kuidas saame neid kirjeldada?“, „Mis neid teistest eristab?“, et kirjeldada klassi kuuluvaid objekte. Determinatsioonanalüüsi võib lugeda klassifitseerimismeetodiks ning saab seostada ka assotsiatsioonireeglitega.

Käesolevas töös on DAd arendatud selle erinevates lähenemistes esinevate puuduste kõrvaldamiseks, peamiselt liiasuse vähendamise suunas. Nullfaktorid (NF) on elemendid, mis põhjustavad liiasust ja ei tohi esineda reeglite eelduse osas. Defineerisime kaht tüüpi nullfaktorid.

Töös esitatakse järgmised DA arendused:

- Samm-sammuline meetod: aitab liiasust vähendada, leides erineva pikkusega mittelõikuvaid reegleid.
- Esimene lõikuvate reeglite algoritm leiab võimalikult väikese arvu (võimalikult lühikesi) reegleid.
- DSR-lähenemine: selle asemel, et leida üks paljudest eksisteerivatest kirjeldustest, leiab kõik mitteliiased reeglid; seda reeglihulka saab kasutada sobiva katte leidmiseks.
- Nullfaktorivaba DA: leiab kõik mitteliiased reeglid kõigi klasside jaoks (mitte vaid ühe määratud klassi jaoks) ning rikastab neid klassifikatsioonireegleid positiivsete ja negatiivsetete assotsiatsiooni-reeglitega.

Hüpoteeside generaator on andmekaeve meetod, mis kasutab algoritm MONSA pakutavaid võimalusi. Näitame, et HG/MONSA leiab kõik suletud hulgad (*closed sets*).

Muutsime MONSAt nii, et see leiab kõik minimaalsed generaatorid koos neile vastava suletud hulga, „välistatud faktorite“ ja klassiga. Elemendid suletud hulga ja selle generaatori vahel moodustavad järelduse assotsiatsioonireeglile, mille eelduseks on generaator. Välistatud faktoriteks on elemendid, mis ei esine suletud hulgas, need moodustavad järelduse osa negatiivsele assotsiatsioonireeglile. Leidsime, et elemendid suletud hulga ja selle generaatori vahel vastavad ühele (DA) nullfaktori tüübile ja et klassi saab tuvastada samamoodi kui neid nullfaktoreid. Pannes kõik kokku, lõime MONSAst tuletatud algoritmi nullfaktorivaba DA jaoks, mis leiab kolme tüüpi reegleid ühise eelduse osaga: klassifikatsioonireeglid, positiivsed ja negatiivsed assotsiatsioonireeglid.

Et suletud hulk koos kõigi oma minimaalsete generaatoritega moodustab ekvivalentsiklassi, lõime ka algoritmi, mis leiab kõik ekvivalentsiklassid.

Lõpuks kogusime GH ja DA poolt lahendatavad ülesanded ühtsesse raamistikku, mida kutsume universaalseks hüpoteeside generaatoriks.

MONSA ja kõik teised töös esitatud algoritmid põhinevad monotoonsete süsteemide teoorial. Loodud NF-vaba DA algoritm sobib mitmete edasiste arenduste aluseks.

Appendix A

Kuusik, R. and Lind, G. (2008). Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques. *WSEAS Transactions on Information Science and Applications, 5*(5), 599-611.

Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques

REIN KUUSIK, GRETE LIND
Department of Informatics
Tallinn University of Technology
Raja 15, Tallinn 12618
ESTONIA
kuusik@cc.ttu.ee, grete@staff.ttu.ee

Abstract: - In this paper an algorithm named MONSA for closed sets mining is presented. It does not use such kind of techniques as in ChARM by Zaki and Hsiao. MONSA is an exact depth-first search algorithm extracting only frequent closed sets using several new very effective pruning techniques to be free from repetitive and empty patterns. MONSA does not depend on the initial order of objects. In MONSA there is active only one branch which is under construction. The purpose of this paper is to describe the approach used in MONSA and the correspondence of its basics and concepts to the approach by Zaki and Hsiao. A full example of the algorithm's work is presented. By the algorithm the intersections (closed sets) and IF…THEN rules on the subsets of source data set simultaneously are found. MONSA treats not only binary data, but a larger set of discrete values.

Key-Words: - Data mining, Frequent closed sets, Pruning techniques, Depth-first search, Monotone systems

1 Introduction

„The task of mining association rules consists of two main steps. The first involves finding the set of all frequent itemsets. The second involves testing and generating all high confidence rules among itemsets" [1]. Zaki and Hsiao prove in [1] that „It is not necessary to mine all frequent itemsets in the first step, instead it is sufficient to mine the set of closed frequent itemsets, which is much smaller than the set of all frequent itemsets" and „It is also not necessary to mine the set of all possible rules", because „any rule between itemsets is equivalent to some rule between closed itemsets. Thus many redundant rules can be eliminated" [1]. Also it is important to enumerate closed sets directly, without generating them „using Apriori-like bottom-up search methods that examine all subsets of frequent itemsets" or finding them from maximal patterns „since all subsets of the maximal itemsets would again have to be examined" [2].

As we see, for mining association rules we need to find only closed sets. We did it. But in our algorithm MONSA we use other denotations and techniques than Zaki and Hsiao [1] [2].

Algorithm MONSA [3] was created for discovering intersections with special quality (see section 2.2 (6)). Appears that the set of intersections it founds is the same as the set of all (frequent) closed sets. Additionally our algorithm enables to find rules (between closed itemsets) at the same time as closed itemsets itself. Nevertheless this is a subset of all possible rules between closed sets. In order to ensure that these rules hold with at least required confidence it is possible to prune the branches of a search tree according to the threshold given by the user.

The main goal of the paper is to describe our algorithm and the main theoretical conceptions on which it is based.

1.1 The task

MONSA is a depth-first search algorithm used for discovering intersections with special quality. The *intersection* of two (or more) sets is the set of elements, which belong to both (all) sets, simultaneously. Without used pruning techniques MONSA would perform exhaustive search and produce all permutations of all existing value combinations. The *value combination* is a set of elements, an *element* is an attribute with certain value. Taking into account the minimal frequency allowed by the user only the frequent part of the result is found.

There are several problems in intersections' finding process: we have to prevent finding 1) repetitious intersections and 2) empty intersections. Finding empty intersections is avoided by the nature of the algorithm MONSA, it does not generate (theoretical) combinations, but traverses only through the really existing ones. In order to prevent "repetitions" we use pruning techniques called "bringing zeros down" and "backward comparison" (see 3.3). Appears that it is not enough, sometimes we have to perform one more check: 3) is the extracted intersection (i.e. closed set) really non-redundant (i.e. not extracted already) or not. This check is applied only when the current set is not closed (and addition of elements would make it closed). We always know whether the current set is closed or not. In this paper we will show the way we check it, and it differs from the way presented in [1] [2]. We also introduce the approach used in MONSA and the correspondence of its basics and concepts to the approach by Zaki and Hsiao.

1.2 Basic idea of our approach

The fundamental idea of our approach is very simple: frequencies determine a frequent set. We collect the frequencies into a frequency table, which shows how many times different values of attributes appear in data set. For example in Table 1 is given a set of two objects described by three attributes and its corresponding frequency table.

Table 1. Example X(2,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	2	2
O2	1	1	2

Value \ Attribute	*A1*	*A2*	*A3*
1	**2**	1	0
2	0	1	**2**

Frequent set = A1.1 AND A3.2

As we see all elements (i.e. attribute with certain value) that have a frequency equal to the number of objects belong to the frequent set. The frequent set with such properties can be found as an intersection through the table.

Table 2. Intersection through X(2,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	2	2
O2	1	1	2
Intersection	**1**	*	**2**

IntSec$_X$ = A1.1 AND A3.2

As we see from Table 1 and Table 2 an intersection is also findable through the frequencies: every value which has frequency equal to the number of objects in the table belongs to the intersection. But we need special techniques to form tables with this property. Then for every table we have formed we find an intersection over the frequency table as we did in Table 1.

We start the description of our approach with definitions in 2. Terms used in closed set mining are given in 2.1 and definitions used explaining MONSA and the relations between terms of two approaches are shown in 2.2. Section 3 describes MONSA, including the algorithm in 3.1, an example of the working process in 3.2, and the explanation of used pruning techniques in 3.3. Section 4 concludes this paper.

2 Definitions

2.1 Definitions of closed set mining

Definitions used in closed set mining are presented here as in [1] [2].

Typically the database is arranged as a set of transactions, where each transaction contains a set of items. Let $I = \{1,2,...,m\}$ be a set of items, and let $T = \{1,2,...,n\}$ be a set of transaction identifiers or *tids*.

A set $X \subseteq I$ is also called an *itemset*, and a set $Y \subseteq T$ is called *tidset*.

$t(X)$ denotes a tidset that corresponds to an itemset X, i.e. the set of all tids that contain X as a subset: $t(X) = \cap_{x \in X} t(x)$.

$i(Y)$ denotes an itemset that corresponds to a tidset Y, i.e. the set of items common to all the tids in Y: $i(Y) = \cap_{y \in Y} i(y)$.

The *support* of an itemset X, denoted $\sigma(X)$, is the number of transactions in which it occurs as a subset.

An itemset is *frequent* if its support is more than or equal to a user-specified *minimum support* (*min_sup*) value, i.e. if $\sigma(X) \geq min\_sup$.

A frequent itemset X is called *closed* if there exists no proper superset $Y \supset X$ with $\sigma(X) = \sigma(Y)$.

Closed sets are found using closure operation.

A *closure* of an itemset X, denoted $c(X)$, is defined as the smallest closed set that contains X. An itemset X is closed if and only if $X = c(X)$.

The closure of an itemset X is found as: $c(X) = i(t(X))$.

The support of an itemset X is also equal to the support of its closure, i.e. $\sigma(X) = \sigma(c(X))$.

2.2 Definitions used in MONSA

The first version of MONSA was presented in 1993 [3]. Here we present denotations and definitions used for MONSA as in [3]. We also give comments how these notions and concepts relate to the ones used for closed sets (see 2.1).

(1) X - a set $X = \{Xi\}$, $i = 1,2,...,N$, where each object Xi is a conjunction of M attribute values: $Xi = \underset{j=1}{\overset{M}{\&}} h_j$.

X is a set of N objects (records) that are described by M attributes. Set X has not to be a binary dataset, every attribute j can acquire integer values in the interval $h_j = 0,1,2,...,K_j\text{-}1$. X can be a transaction database also.

(2) H - a value combination (VC) of certain attributes $H = \underset{q\in D}{\&} h_q$, $D = \{j_e\}$, $e = 1,...,E_H$ (number of elements h_q in H), $1 \le E_H \le M$, $1 \le j_e \le M$, $j_f, j_t \in D$, $j_f \# j_t$, $f \# t$, $H \subseteq Xi$.

A value combination can contain 1 to M attributes (with certain values), each only once (i.e. only with one value); it is a subset of some object or is a whole object.

(3) Each value combination H defines on the set X a subset of objects $X_H = \{Xp\}$, $p = 1,2,...,N_H$, $1 \le N_H \le N$,
$\{Xp\}$ are all objects $Xi \in X$ that contain H: $X_H = \{Xi \in X \mid Xi \supseteq H\}$.

The subset of objects defined by the value combination is similar to the tidset that corresponds to some itemset (*t(X)*).

(4) Each value combination H defines on set X a subset of elements $X^H \subseteq X$: $X^H = \{Xij \in X \mid Xij \in H$, $i = 1,2,...,N$, $j = 1,2,...,M\}$.

An attribute with certain value is called element. ‘Element’ corresponds to ‘item’. The difference is that each attribute produces as many elements as many different values it has. Definition (4) says that ‘value combination’ is the same as ‘itemset’ (with extension that values need not be binary).

(5) Intersection over a set $Y = \{Yt\}$, $t = 1,2,...,T$, $Yt = \&h_j$ is a set of such elements h_q which belong simultaneously to all Yt: $\cap Y = \bigcap_{t=1}^{T} Yt = \underset{q}{\&} h_q = H$.

In Y for H there exists always a corresponding subset of objects $Y_H = \{Yp\}$, $p = 1,...,N_H$, $N_H \le N$.

If $N = 1$, then intersection over Y is an object itself.

If there exist no objects $Yt \in Y$ for which $H \subset Yt$, then $Y_H = \varnothing$.

Definition (5) says that intersection over a set of objects is a set of common elements, this is the same as itemset that corresponds to some tidset (*i(Y)*).

(6) Elementary conjunction (EC) on X_H is such an intersection over the set X_H, where $\cap X_H = A(\supseteq H)$, $X_A = X_H$, $N_H \le N$.

In the case of $A \supset H$, $X_A = X_H$, A is an EC.

We have a set of elements H and its corresponding set of objects X_H (i.e. *t(H)*) and find an intersection over it: $\cap X_H$ (i.e. *i(t(H))*). The operation *i(t(H))* means finding the closure of H. Therefore the resultant set A (of elements) called elementary conjunction is a closed set. From the viewpoint of the algorithm it is essential to find a technique that guarantees the finding of such subsets X_H only for which $\cap X_H = A(\supseteq H)$ (i.e. finding of closed sets only).

(7) Maximal EC on X_H is such an intersection over X_H in case of which for a VC $H = \underset{q}{\&} h_q$ is a valid relation
$\cap X_H = A\,(= \underset{e}{\&} h_e) \supset H$, $1 \le q < e \le M$, $X_A = X_H$.

By definition, VC H is EC if $\cap X_H = H$.

H is a maximal EC if it is EC and contains at least one VC $Ht \subset H$ such, that $|X_{Ht}| = |X_H|$ on X. That means that the set of objects $X_H \subseteq X$ is defined unique.

Definition (7) says that maximal EC is EC that has at least one (non-closed) subset with the same frequency (support) i.e. we can remove at least one element without changing in frequency. Our maximal elementary conjunction here is not the same as maximal (closed) set[1].

Additionally, our ‘(absolute) frequency’ is the same as ‘support’.

For rules we use also ‘relative frequency’ which corresponds to ‘confidence’.

3 MONSA

MONSA is a depth-first search algorithm. Without pruning techniques it would traverse the complete search tree and find all permutations of all existing value combinations. Empty (i.e. non-existing) combinations are avoided by nature of algorithm, they are not generated (and checked for existence) at all.

[1] “A frequent itemset X is called maximal if it is not a subset of any other frequent itemset.” [2]

From the initial data set MONSA finds a result as a set of intersections (closed sets) and/or a set of trees (forest), they are listed in the order they are found, that order does not depend on the initial order of objects (records). The frequencies of nodes (intersections) decrease strictly along branches of a tree(s). The decrease makes allowable to prune the branches according to the minimal allowed frequency (support). This is similar to the other tree-based algorithms, including ChARM [2]. The fact that the decrease is strict gives a high potential to the intersection (combination from root to the certain node) to be a closed set. At any level the descendants of a common parent-node are found in a weakly decreasing order of their frequencies, the roots also are found in a weakly descending order. The order of nodes with equal frequency depends on the searching principle (usually by columns or by rows of frequency table).

MONSA uses frequency tables (introduced in section 1.2). For every extract of objects its corresponding frequency table is formed. Zero in the initial frequency table means that the corresponding element does not exist. During the work the elements that are exhaustively analyzed are zerofilled in frequency tables. Such prohibited (eliminated) elements are not included into the intersections any more, only the elements with frequency over zero (or some higher threshold) are considered.

3.1 Description of the algorithm

MONSA finds intersections in given set X(N,M), where N is the number of objects (for example transactions), M is the number of attributes and each attribute j has an integer value $h_j=0,1,2,\ldots,K-1$. A pair of attribute and its certain value is called element.

By essence MONSA is a recursive algorithm. Here its backtracking version is presented.

In this algorithm the following notation is used:

t	the number of the step (or level) of the recursion
FT_t	frequency table for a set X_t
$IntSec_t$	vector of elements over set X_t (intersection)
Init	activity for initial evaluation

The algorithm is given in Fig. 1.

It has been proved that if a finite discrete data matrix X(N,M) is given, where $N=K^M$, then the complexity of algorithm MONSA to find all $(K+1)^M$ elementary conjunctions (intersections) as existing value combinations is $O(N^2)$ operations [3].

By our estimation in practice the upper bound of the number of value combinations (with minimal frequency allowed = 1) is

$$L_{UP} \approx N(1+1/K)^M, \qquad (1)$$

but usually it is less.

The precise formula for the number of intersections is as follows:

$$L = \sum_{f=1}^{F} \sum_{p=1}^{(M*K-u)} N_p \ , \qquad (2)$$

where F is the number of formatted frequency tables on set X, u is the number of empty cells in the frequency table FT_f, N_p is the absolute number in a cell of the frequency table (frequency of an attribute value).

The most important advantages of the algorithm are:

- The use of new original and very effective pruning techniques
- The possibility to output the tree immediately during the finding closed sets
- The frequency is known at the moment the new node is found
- The ability to find nodes consisting of more than one element, this reduces the number of nodes and the size of the tree
- In order to prevent repetitions we do not look through the already found result and therefore we do not need additional data structures
- Attributes can have more values than only 0/1

An example in the following chapter explains how MONSA works.

```
Algorithm MONSA

Init
t←0, IntSec_0←{}
Find a table of frequencies FT_0 for all attributes in X_0
DO WHILE there exists FT_s#Ø in {FT_s}, s≤t
  FOR EACH element h_f∈FT_t with frequency V=max FT_t(h_f)#0 DO
    IF pruning is needed (h_f has to be pruned) THEN GOTO BACK
    Separate submatrix X_{t+1}⊂X_t such that X_{t+1}={Xij∈X_t|X.f=h_f}
    Find a table of frequencies on X_{t+1} FT_{t+1}
    ZeroesDown(t+1)
    CheckUniqueness(t+1)
    IF new intersection is unique THEN
      Add elements j with FT_{t+1}(j)=V into vector IntSec_{t+1}
      BackwardComparison(t+1)
      Output of IntSec_{t+1}
      IF there exist attributes to analyse THEN t←t+1
    ENDIF
  NEXT
  BACK: t←t-1
  IntSec_{t+1}←IntSec_t
ENDDO
All intersections are found
END: end of algorithm


Elimination (pruning) activities:

1)
ZeroesDown(t+1)
    FOR EACH element h_u∈FT_t DO
      IF FT_t(h_u)=0 THEN FT_{t+1}(h_u)←0
    NEXT

2)
BackwardComparison(t+1)
    FOR EACH element h_u∈FT_{t+1} with frequency #0 DO
      IF FT_{t+1}(h_u)=FT_t(h_u) THEN FT_t(h_u)←0
    NEXT

3)
CheckUniqueness(t+1)
    IF there exists on X_{t+1} h_u, 1≤u≤M such, that
    [h_u∈IntSec_{t+1} AND FT_{t+1}(h_u)=0 AND frequency of h_u in X_{t+1}=V]
    THEN
      Intersection is not unique
    ELSE
      Intersection is unique
    ENDIF
```

Fig. 1. Algorithm MONSA

3.2 Steps of MONSA

In order to explain the work of MONSA we will demonstrate it using the initial data set given in Table 3.

In order to find intersections MONSA uses frequency tables. A frequency table contains the counts of occurrences of all existing values for each attribute. Each attribute can have a different number of different discrete values.

The frequency table corresponding to the initial data from Table 3 is given in Table 4.

Table 3. Initial data

Object \ Attribute	*A1*	*A2*	*A3*
O1	1	0	3
O2	2	2	1
O3	2	3	0
O4	2	0	2
O5	0	1	3
O6	0	1	3
O7	1	1	2
O8	1	0	3
O9	2	3	0

Table 4. Frequency table (FT)

Value \ Attribute	*A1*	*A2*	*A3*
0	2	3	2
1	3	3	1
2	4	1	2
3	0	2	4

MONSA is a depth-first search algorithm which backtracks when the current branch is exhausted or has to be pruned. Inside the backtracking algorithm the main steps at each level are as follows:

S1: Choose a "leading" element – the first element with maximal frequency (over zero; at least with threshold frequency specified by the user), add it into the (potential) intersection and zerofill the corresponding cell in the frequency table

S2: Calculate the next frequency table for objects containing that leading element

S3: If there exist element(s) with frequency equal to the leading one check the intersection's uniqueness => if it is unique, add these elements (with frequency equal to the leading frequency) into the intersection; otherwise backtrack

S4: Output intersection

S5: "Bring down" zeroes from the frequency table of the previous level i.e. at the current level zerofill all elements that have been zerofilled at the previous level

S6: "Backward comparison": elements that have equal frequencies at both levels are zerofilled at the previous level

Next we will show how the algorithm works. For better understanding we give a frequency table (FT) for each level, the number in subscript shows the level of recursion (starting from the level 0). The row above the table header shows the current potential intersection, where '*' means that certain value for that attribute is not determined yet. The number after "=" is the frequency of shown intersection. We also use a non-positional representation of intersections, for example: A2.0=3 – the intersection consists of one element – attribute A2 with value 0 and its frequency is 3.

During the work we zerofill certain elements in the frequency tables for different reasons. In order to mark those elements the following notation is used:

→0 – a zerofilled leading element (the result of step S1);

! – elements with frequency equal to the leading frequency that are set to zero in case of inclusion into intersection (the result of step S3);

↓0 – zeroes "brought down" from the previous level frequency table (the result of step S5);

↑0 – frequencies equal at current and previous level (set to zero at the previous level, the result of step S6).

Now we will find all intersections with frequency at least 2 for data given in Table 3. This threshold – minimal frequency allowed (=2) – is set by the user.

Having the frequency table for the initial data set (see Table 5a), the first thing (S1) is to find an element with maximal frequency. If the search direction is by columns, then the first element with maximal frequency (which is 4) is attribute 1 (A1) with value 2. We include it into the intersection (2 * * =4; shown above Table 6). In order to prevent the repeating selection of it (A1.2) we also eliminate this element from the frequency table FT_0 by putting zero into the corresponding cell (marked with →0 in Table 5b).

Table 5. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	2	3	2
1	3	3	1
2	4	1	2
3	0	2	4

b)

FT_0	*A1*	*A2*	*A3*
0	2	3	2 ↑0
1	3	3	1 ↑0
2	4→0	1 ↑0	2
3	0	2 ↑0	4

S2: The "leading" element A1.2 is the basis for extracting a subset of objects, i.e. only objects that contain that certain element (A1.2) belong to the subset. As the frequency shows this subset consists of four objects, namely O2, O3, O4 and O9. We calculate a frequency table for the next level by taking into account only those (four) objects. As attribute A1 (with its value 2) was the basis of separating the subset (and it is already included into the intersection), we do not look at this attribute at the current level any more and the new frequency table FT_1 (given in Table 6a) does not contain frequencies for attribute A1.

Table 6. Frequency table for A1.2 (at level 1)

a)

2 * * =4

FT_1	*A1*	*A2*	*A3*
0		1	2
1		0	1
2		1	1
3		2	0

b)

FT_1	*A1*	*A2*	*A3*
0		1	2 ↑0
1		0	1
2		1	1
3		2→0	0

S3, S4: In the new frequency table (see Table 6a) there are no frequencies equal to the leading one (=4), therefore we can output intersection (2 * * =4 or A1.2=4) without "uniqueness check".

S5: There are no zeros to bring down from the initial frequency table (Table 5a) because the only zerofilled element is the leading element.

S6: The next thing to do is to compare the frequencies of the current level (Table 6) with the frequencies of the previous level (Table 5) and find four coincidences (A2.2, A2.3, A3.0 and A3.1). It means that all objects containing (some of) these elements belong to our current subset and will be fully analysed at the current and subsequent levels. Using those elements for extracting subsets at the previous level would cause repetitions. In order to prevent it we zerofill these four cells at the previous level (marked with ↑0 in Table 5b).

Now we will repeat all these activities at the next level.

S1: The (first) element with maximal frequency in FT_1 (Table 6a) is A2.3 with frequency 2, we remember it in the next potential intersection (2 3 * =2; shown above Table 7) and eliminate it (→0 in Table 6b). S2: Next we extract the subset of objects having this element (A2.3) and find the corresponding frequency table (see Table 7).

Table 7. Frequency table for A1.2&A2.3 (at level 2)

0 !

2 3 * =2

FT_2	*A1*	*A2*	*A3*
0			2 !
1			0
2			0
3			0

S3: This time there exists one element that has the frequency equal to the leading one (=2) – this element is A3.0 (! in Table 7). It means that all objects in the subset contain that element and we should include it into the intersection. The intersection would be (2 3 0) instead of (2 3 *). Before we can do it, we have to check whether this intersection (with A3.0) is unique (non-redundant) i.e. is not found already. The check is simple: if the frequency of that element is not zerofilled in the previous frequency table then this intersection is not found yet. The frequency of A3.0 at level 1 (Table 6a) is more than zero, so the new intersection (2 3 0) is unique and is outputted (A1.2&A2.3&A3.0=2). In order to prevent A3.0 to be a basis for extracting a new subset its frequency in the current frequency table (Table 7) will be set to zero after the "backward comparison" (S6).

S5: There are no zeroes to bring down in the column A3 (which is the only attribute under consideration). S6: Comparison of levels 1 and 2 (Table 6a and Table 7) sets (the same) A3.0 to zero at level 1 (↑0 in Table 6b).

Nullifying A3.0 at level 2 (Table 7) makes this frequency table empty. So we will continue at level 1 (Table 6b). Appears that there are no frequencies over the threshold (minimal frequency allowed =2).

So we backtrack to the initial level. Taking into account all zerofilling information (shown in Table 5b) the frequency table looks like in Table 8a. We find that element with maximal frequency is A3.3 (with frequency 4). S1: We include this element into the intersection (* * 3 =4; shown above Table 9) and set to zero its frequency in FT_0 (→0 in Table 8b).

Table 8. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	2	3	0
1	3	3	0
2	0	0	2
3	0	0	4

b)

FT_0	*A1*	*A2*	*A3*
0	2 ↑0	3	0
1	3	3	0
2	0	0	2
3	0	0	4→0

The frequency table for A3.3 is given in Table 9a (S2). It contains no frequencies equal to the leading one (=4), so the intersection A3.3=4 is outputted (S3, S4).

There are no zeros to bring down from FT$_0$ (Table 8a) (S5). Backward comparison (S6) sets to zero element A1.0 at level 0 ($\uparrow$0 in Table 8b).

Table 9. Frequency table for A3.3 (at level 1)

a)

* * 3 =4

FT$_1$	*A1*	*A2*	*A3*
0	2	2	
1	2	2	
2	0	0	
3	0	0	

b)

FT$_1$	*A1*	*A2*	*A3*
0	2$\rightarrow$0	2	
1	2	2 $\uparrow$0	
2	0	0	
3	0	0	

We continue at level 1 (see Table 9a).

S1: The maximal frequency is 2, the first element with it is A1.0. We add it into intersection (0 * 3; shown above Table 10) and set to zero its frequency ($\rightarrow$0 in Table 9b).

S2: We find the frequency table for the next level i.e. level 2 (see Table 10).

Table 10. Frequency table for A3.3&A1.0 (at level 2)

1 !
0 * 3 =2

FT$_2$	*A1*	*A2*	*A3*
0		0	
1		2 !	
2		0	
3		0	

S3: In FT$_2$ (Table 10) A2.1 has the frequency equal to the leading one (=2). Its frequency at previous level (see Table 9a) is not zeroed, so the intersection with A2.1 (0 1 3) is unique and is outputted (A3.3&A1.0&A2.1=2; S4).

S5: There are no zeros to bring down in column A2.

S6: (that same) A2.1 has equal frequencies at both levels (1 and 2; see Table 9a and Table 10), its frequency at level 1 is zerofilled ($\uparrow$0 in Table 9b).

We zerofill A2.1 at level 2 also (due to the inclusion into the intersection). The frequenency table FT$_2$ (Table 10) becomes empty, we backtrack to level 1. After zerofilling (see Table 9b) the frequency table for A3.3 looks like in Table 11a.

Table 11. Frequency table for A3.3 (at level 1)

a)

* * 3 =4

FT$_1$	*A1*	*A2*	*A3*
0	0	2	
1	2	0	
2	0	0	
3	0	0	

b)

FT$_1$	*A1*	*A2*	*A3*
0	0	2 $\uparrow$0	
1	2$\rightarrow$0	0	
2	0	0	
3	0	0	

S1: We find that first element with maximal frequency (equal to 2) is A1.1, include it into the intersection (1 * 3; shown above Table 12) and zerofill the corresponding cell ($\rightarrow$0 in Table 11b). S2: We calculate the frequency table for the next level (see Table 12). S3: Element A2.0 has the frequency equal to 2. As this element is not zerofilled at the level 1 (Table 11a), we include it into the intersection (1 0 3) and output (S4) the intersection (A3.3&A1.1&A2.0=2). S5: There are no zeros to bring down. S6: Element A2.0 has equal frequencies at both levels (see Table 11a and Table 12) and we zerofill it at level 1 (marked with $\uparrow$0 in Table 11b). Due to the inclusion into the intersection the element A2.0 is set to zero at level 2 also. This frequency table (Table 12) is empty now.

Table 12. Frequency table for A3.3&A1.1 (at level 2)

0 !
1 * 3 =2

FT$_2$	*A1*	*A2*	*A3*
0		2 !	
1		0	
2		0	
3		0	

The frequency table at level 1 (Table 11b) is also empty (taking into account the zerofilling information).

The frequency table at level 0 is shown in Table 13a.

Table 13. Frequency table at level 0

a)

FT$_0$	*A1*	*A2*	*A3*
0	0	3	0
1	3	3	0
2	0	0	2
3	0	0	0

b)

FT$_0$	*A1*	*A2*	*A3*
0	0	3	0
1	3$\rightarrow$0	3	0
2	0	0	2
3	0	0	0

S1: The maximal frequency is 3 and the first element with it is A1.1. We add it into the intersection (1 * * =3) and nullify the corresponding cell ($\rightarrow$0 in Table 13b) S2: The frequency table for objects containing A1.1 is found (see Table 14a). S3, S4: It contains no frequencies equal to 3, so the intersection (shown above Table 14) is outputted as it is (A1.1=3).

Table 14. Frequency table for A1.1 (at level 1)

a)

1 * * =3

FT_1	*A1*	*A2*	*A3*
0		2	0
1		1	0
2		0	1
3		0	2

b)

FT_1	*A1*	*A2*	*A3*
0		2	0
1		1	0
2		0	1
3		0	2 ↓0

S5: The cell for A3.3 has been zerofilled at level 0 (Table 13) and this zero is now brought down to level 1 (↓0 in Table 14b). This zero prevents traversing A1.1&A3.3 (which is the subset of already found intersection A3.3&A1.1&A2.0=2 with the same frequency and therefore redundant) and subsequent subsets. S6: Backward comparison finds no equal frequencies.

Table 15. Frequency table for A1.1 (at level 1)

a)

1 * * =3

FT_1	*A1*	*A2*	*A3*
0		2	0
1		1	0
2		0	1
3		0	0

b)

FT_1	*A1*	*A2*	*A3*
0		2→0	0
1		1	0
2		0	1
3		0	0

S1: At current level (see Table 15a) the maximal frequency is 2 and element with it is A2.0 (zerofilling is shown in Table 15b). S2: The frequency table for objects containing A2.0 is given Table 16 and the current intersection (1 2 * =2) is above it.

Table 16. Frequency table for A1.1&A2.0 (at level 2)

1 0 * =2

FT_2	*A1*	*A2*	*A3*
0			0
1			0
2			0
3			2 !

S3: In FT_2 (Table 16) element A3.3 has frequency 2 (equal to the leading one). The "uniqueness check" (i.e. looking into the corresponding cell in the previous frequency table) gives the answer: such intersection (with A3.3) – (1 0 3) has been found already and current subset has been analysed already. In such situation we backtrack to the previous level (without outputting anything).

The frequency table at level 1 (Table 15b) contains no frequencies over the threshold (=2), therefore we backtrack again.

Table 17. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	0	3	0
1	0	3	0
2	0	0	2
3	0	0	0

b)

FT_0	*A1*	*A2*	*A3*
0	0	3→0	0
1	0	3	0
2	0	0	2
3	0	0	0

At level 0 (Table 17a) the maximal frequency is 3 and the first element with such frequency is A2.0 (zerofilling is shown in Table 17b). S1, S2: we add it into the intersection (* 0 *) and find the next frequency table (Table 18a).

Table 18. Frequency table for A2.0 (at level 1)

a)

* 0 * =3

FT_1	*A1*	*A2*	*A3*
0	0		0
1	2		0
2	1		1
3	0		2

b)

FT_1	*A1*	*A2*	*A3*
0	0		0
1	2 ↓0		0
2	1 ↓0		1
3	0		2 ↓0

S3, S4: The current intersection is outputted as it is (A2.0=3), because there are no elements with the same frequency. S5: The comparison of frequency tables of level 1 (Table 18) and level 0 (Table 17) finds 3 zeros to bring down (for A1.1, A1.2, A3.3; shown in Table 18b). S6: Backward comparison finds no coincidences of frequencies.

The current frequency table (Table 18b) contains no frequencies over 2, therefore we go back to the level 0 (see Table 19a).

Table 19. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	3	0
2	0	0	2
3	0	0	0

b)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	3→0	0
2	0	0	2
3	0	0	0

S1: Include element A2.1 with frequency 3 into the intersection (* 1 * =3; shown above Table 20) and zero its frequency (→0 in Table 19b).

S2: Find the frequency table for those 3 objects (Table 20a).

Table 20. Frequency table for A2.1 (at level 1)

a)

	*	1	*	=3
FT_1	*A1*	*A2*	*A3*	
0	2		0	
1	1		0	
2	0		1	
3	0		2	

b)

FT_1	*A1*	*A2*	*A3*
0	2 ↓0		0
1	1 ↓0		0
2	0		1
3	0		2 ↓0

S3: No need for uniqueness check.

S4: Output intersection (A2.1=3).

S5: Bring down zeros from FT_0 (Table 19) to FT_1 (marked with ↓0 in Table 20b).

S6: Backward comparison gives no consequencies.

The frequency table at level 1 (Table 20b) contains no frequencies over the threshold, so we turn back to the level 0 (see Table 21).

Table 21. Frequency table at level 0

a)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	0	0
2	0	0	2
3	0	0	0

b)

FT_0	*A1*	*A2*	*A3*
0	0	0	0
1	0	0	0
2	0	0	2→0
3	0	0	0

S1: Include element A3.2 with frequency 2 into the intersection (* * 2 =2).

S2: Find the frequency table for the next level (see Table 22a).

Table 22. Frequency table for A3.2 (at level 1)

a)

	*	*	2	=2
FT_1	*A1*	*A2*	*A3*	
0	0	1		
1	1	1		
2	1	0		
3	0	0		

b)

FT_1	*A1*	*A2*	*A3*
0	0	1 ↓0	
1	1 ↓0	1 ↓0	
2	1 ↓0	0	
3	0	0	

S3: No uniqueness check.

S4: Output intersection (A3.2=2).

S5: Bring down zeros (for A1.1, A1.2, A2.0, A2.1) – see Table 22b.

S6: Backward comparison finds no elements with equal frequencies at both levels.

The frequency table FT_1 is empty (i.e. contains only zeros). We backtrack to level 0.

The frequency table at level 0 (Table 21b) is also empty.

There are no more levels to go back, thus the algorithm has finished its work. All intersections are found.

As a result we have found 9 intersections (with minimal frequency allowed =2):

2 * *	A1.2=4
2 3 0	A1.2&A2.3&A3.0=2
* * 3	A3.3=4
0 1 3	A3.3&A1.0&A2.1=2
1 0 3	A3.3&A1.1&A2.0=2
1 * *	A1.1=4
* 0 *	A2.0=3
* 1 *	A2.1=3
* * 2	A3.2=2

They all are different (unique), and they are not empty. All three pruning techniques together prevented repetitions, both full repetitions (in different order of elements) and partial repetitions (i.e. subsets of intersections with the same frequency). Uniqueness check was applied only when there was more than one element to include into the intersection at the same step (level) and this check was made without looking through the already found intersections.

Instead of list of intersections the same result can be listed as a set of trees (immediately during the process of finding intersections):

```
(4) 0.500(2)
A1.2=>A2.3&A3.0


(4) 0.500(2)
A3.3=>A1.0&A2.1

    0.500(2)
    =>A1.1&A2.0


(3)
A1.1


(3)
A2.0


(3)
A2.1

(2)
A3.2
```

The trees are represented from left to right. This example consists of six trees, it has six root nodes (on the left). Symbols '=>' separate the nodes of a tree.

Usually a node contains one element – a pair of certain attribute and a certain value of that attribute.

Attribute is shown before period and value is after period. For example, the root node of the first tree contains attribute A1 and its value 2. A node can consist of more than one attribute-value pairs like all non-root nodes in this example, then '&' is used to connect them.

The numbers above node show frequencies.

In the parentheses node's absolute frequency is shown. The absolute frequency of node t shows how many objects have a certain attribute(s) with a certain value(s) among objects having properties (i.e. certain attributes with certain values) of all previous levels t-1,…,1. For example, the first tree shows that among four objects with A1.2 there are two objects having both A2.3 and A3.0.

Before parentheses node's relative (to the previous level) frequency is shown. The relative frequency is a ratio A/B, where A is the absolute frequency of node t and B is the absolute frequency of node t-1. For the first level the relative frequency is not calculated.

3.3 Pruning techniques used in MONSA

In order to avoid finding "repetitions" i.e. permutations of already found intersections two pruning techniques have been used in algorithm MONSA:

"bringing zeroes down" – activity that prohibits arbitrary output repetition of already separated intersection at the next (deeper) level(s);

"backward comparison" – activity that does not allow the output of the separated intersection at the same (current) level and also at previous (higher) levels (after backtracking).

Impact of these techniques is proved in [3][2].

Appears that these two activities do not prevent repetitious finding (and output) of some subsets of already found intersections.

The subset of already found intersection is redundant only if both have equal frequencies (i.e. this (sub)set is non-closed). If subset's frequency is higher (than its superset's) then it covers more objects and is not redundant. Subset's frequency cannot be lesser.

Finding a superset of already found set with the same frequency is impossible, because at any level MONSA finds all co-existing (i.e. contained in the same objects) elements with equal frequencies as one intersection (i.e. maximal EC).

[2] Theorems 5.3 and 5.4 accordingly

Next we give an example of redundant subsets under consideration. For that we have used MONSA without "uniqueness check" (of a new intersection).

Having the initial data set of nine objects described by three attributes (see Table 23) MONSA (without uniqueness check) finds fifteen intersections (with minimal frequenct allowed =2). Both representation forms – trees and intersections – are given in Fig. 2.

Table 23. Example X(9,3)

Object \ Attribute	*A1*	*A2*	*A3*
O1	4	0	8
O2	5	3	4
O3	5	4	3
O4	5	0	7
O5	3	1	8
O6	3	1	8
O7	4	1	7
O8	4	0	8
O9	5	4	3

(a) Result as a set of trees:	(b) Result as a set of intersections:
(4) 0.500(2)	I1) A1.5=4
A1.5=>A2.4&A3.3	I2) A1.5&A2.4&A3.3=2
0.250(1)	I3) A1.5&A2.0&A3.7=1
=>A2.0&A3.7	I4) A1.5&A2.3&A3.4=1
0.250(1)	
=>A2.3&A3.4	
(4) 0.500(2)	I5) A3.8=4
A3.8=>A1.3&A2.1	I6) A3.8&A1.3&A2.1=2
0.500(2)	I7) A3.8&A1.4&A2.0=2
=>A1.4&A2.0	
(3) 0.667(2)	I8) A1.4=3
A1.4=>A2.0	I9) A1.4&A2.0=2
0.333(1)	I10) A1.4&A2.1&A3.7=1
=>A2.1&A3.7	
(3) 0.333(1)	I11) A2.0=3
A2.0=>A3.7	I12) A2.0&A3.7=1
(3) 0.333(1)	I13) A2.1=3
A2.1=>A3.7	I14) A2.1&A3.7=1
(2)	
A3.7	I15) A3.7=2

Fig. 2. Result found by MONSA (without uniqueness check)

Here we have six trees and six roots accordingly. Intersections I1, I5, I8, I11, I13 and I15 correspond to the roots.

Intersection I11 (A2.0=3) is not redundant although A2.0 is contained in the intersections I3, I7 and I9, because these three intersections have lesser frequencies and none of them cover all objects covered by I11.

Intersections I9, I12 and I14 are redundant:

- I9 (A1.4&A2.0) is a subset of I7 (A3.8&A1.4&A2.0) with the same frequency =2 (both cover objects O1 and O8);
- I12 (A2.0&A3.7) is a subset of I3 (A1.5&A2.0&A3.7), both frequencies are 1 (they cover object O4) and
- I14 (A2.1&A3.7) is a part of I10 (A1.4&A2.1&A3.7), frequencies equal 1 (they cover O7).

There have to be 12 intersections instead of 15.

Starting from the root of a tree the frequencies of intersections (nodes) always (strictly) decrease along any branch of the tree (due to finding maximal EC at every node). As no branch has two intersections with the same frequency, the redundant subsets do not occur in the same branch, they can appear in different branches of a tree or in different trees.

Among siblings (i.e. direct descendants of a common node) any element can appear in only one intersection. Consequently, redundant subsets (under consideration) do not occur among siblings. This is also true for the root-level (which formally consists of descendants of the initial empty set), therefore these redundant subsets never occur at root-level.

Intersection (closed set) and its redundant sub-intersection (with the same frequency) do not have to appear at the same level of a tree (as in all three cases of our example).

Element(s) that appear in the root node are eliminated from further analysis by zerofilling the corresponding cell(s) in the frequency table. Elements that are fully analyzed (exhausted) at deeper levels are prohibited by “backward comparison”. All these zeroes are “brought down” to all succeeding levels and therefore prohibited elements never occur in redundant intersections. So (due to the elimination techniques used) no permutation of whole (already found) intersection (*closed set*) is not found. Elements that are partially analyzed (at the non-root levels) are not eliminated. All this is true for any subtree also.

Although prohibited elements are eliminated from the frequency tables they still appear in the subsets of objects extracted by non-prohibited elements. Remind that some set (of elements) and its subset with the same frequency define the same set of objects. If some prohibited (and eliminated) element appears in all objects (of subset) it means that this subset has been analyzed already (this element can not be contained in the value combination (potential intersection) on which basis that subset of objects was extracted). All intersections containing a prohibited element are already found and such subsets need no more analysis. This situation indicates a repetitive extraction of certain subset of objects.

To exclude redundant subsets (sub-intersections) we have to detect a situation when some prohibited element occurs in all objects and stop analyzing such branch.

In our example (see Table 23 and Fig. 2) intersection I7 (A3.8&A1.4&A2.0) has one more element than redundant set I9 (A1.4&A2.0), namely A3.8. The set of objects extracted by I9 is the same as by I7. Consequently, actual frequency of A3.8 is as much as number of objects in that set (=2). As A3.8 was prohibited after exhaustive analyze, the frequency table contains zero in the corresponding cell. Detecting such situation we can say that A1.4&A2.0 (I7) is a redundant set.

Such “uniqueness check” is used by MONSA. It is not necessary to look through the already found intersections to ensure the new one is non-redundant.

Correctness of ‘uniqueness check’ explained here is proved in [4].

It is interesting that those redundant subsets seem to be the same ones for what ChARM [1] [2] needs “subsumption checking”. In order to ensure that a candidate set is really closed ChARM looks through the (certain) already found closed sets, only those that have 1) the same “tidsum” and 2) the same support (frequency) as the candidate set. (Tidsums of different closed sets with equal frequency tend to be different). For the complete description see [1] [2].

As shown already we perform the uniqueness check of a new intersection (potential closed set) otherwise, without looking through the already found (closed) sets.

4 Conclusion

In this paper we have described algorithm MONSA for finding closed sets. The first version of MONSA was created in 1993 and we have developed several versions during last years.

From the theoretical viewpoint the basic concepts used in MONSA are compared to the ones used by Zaki and Hsiao and it is shown that substantially they are the same.

In MONSA we use some new effective pruning techniques: zeros down, backward comparison, uniqueness check. These are simple original techniques to prevent repetitious finding of already found closed sets whitout using additional data structures. Uniqueness (non-redundancy) check of a new potential closed set is made without looking through the already found closed sets.

During the process of finding all closed sets MONSA can find also rules between them. The algorithm works not only on the binary scale.

MONSA is a basic algorithm we have used not only in data mining [5] [6] [7], but also in solving of graph theoretical problems as extracting of all maximal cliques [4] and decision trees constructing [8]. We have some approaches how to use the algorithm presented here in machine learning for the future works.

Acknowledgement

This work was supported in part by the Estonian Information Technology Foundation under Grant 07-03-00-22.

References:

[1] M. J. Zaki and C.-J. Hsiao, CHARM: An efficient algorithm for closed association rule mining, *Technical Report 99-10*, Department of Computer Science, Rensselaer Polytechnic Institute, October 1999.

[2] M. J. Zaki and C.-J. Hsiao, CHARM: An efficient algorithms for closed itemset mining, *Proceedings of the Second SIAM International Conference on Data Mining*, 2002.

[3] R. Kuusik, The Super-Fast Algorithm of Hierarchical Clustering and the Theory of Monotone Systems, *Transactions of Tallinn Technical University*, No 734, 1993, pp. 37-62.

[4] R. Kuusik, Extracting of all maximal cliques: monotone system approach, *Proceedings of the Estonian Academy of Sciences. Engineering*, No 1, 1995, pp. 113-138.

[5] R. Kuusik, G. Lind, L. Võhandu, Data mining: pattern mining as a clique extracting task, *Proceedings of the Sixth International Conference on Enterprise Information Systems*, Vol. 2, Porto, Portugal, 2004, pp. 519-522.

[6] R. Kuusik, G. Lind, New frequency pattern algorithm for data mining, *Proceedings of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, Foça, Izmir, Turkey, 2004, pp. 47-54.

[7] G. Lind, Method for Data Mining – Generator of Hypotheses, *Databases and Information Systems. Proceedings of the 4th International Baltic Workshop*, Vol. 2, Vilnius, 2000, pp. 304-305.

[8] A. Torim, R. Kuusik, Problem and algorithms for finding the best decision, *WSEAS Transactions on Information Science and Applications*, 9 (2), 2005, pp. 1462-1469.

Appendix B

Lind, G. and Kuusik, R. (2007). Some Ideas for Determinacy Analysis Realisation. *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing. Palma de Mallorca, Spain, August 29-31, 2007* (pp. 185-190). ACTA Press.

SOME IDEAS FOR DETERMINACY ANALYSIS REALISATION

Grete Lind, and Rein Kuusik
Tallinn University of Technology, Department of Informatics
Raja 15, 12618 Tallinn
Estonia
grete@staff.ttu.ee, kuusik@cc.ttu.ee

ABSTRACT
This paper gives overview of theory called determinacy analysis and shows its relations with other approaches. The paper introduces original application of DA, brings out its limitations and shows the amount of calculations needed in case of approach used in this application. Different approach to solve the task of DA is presented.

KEY WORDS
Machine learning, determinacy analysis, and rules.

1. Determinacy analysis

Determinacy Analysis (DA) is a system of methods for the analysis of rules that was created at the end of 70s. Its approach combines mathematical statistics and logic. DA's methodology and the underlying mathematics are developed by Russian scientist Sergei Chesnokov [1], [2].
DA-technology provides an alternative way to perform factor analysis of qualitative and quantitative variables. It assists in obtaining regularities, explanations and prognostic rules.
DA has been used in sociology [3], linguistics [4], medicine [5], and other areas (for complete list of references see [6] or [7]).
The realisation of DA has a number of limitations compared to the theory. In the following we treat the method of analysis of rules and its application and propose more effective solution for the realisation.
The overview of determinacy analysis is based on [1], [2], [6], [7], [8].

1.1 Determination and its characteristics

The main idea behind DA is that a rule can be found based on the frequencies of joint occurrence or non-occurrence of events. Such rule is called a determinacy or determination, and the mathematical theory of such rules is called determinacy analysis. [6]
If it is observable that an occurrence of X is always followed by an occurrence of Y, this means that there exists a rule "If X then Y", or $X \rightarrow Y$. Such correlation between X and Y is called determination (from X to Y). Here X is determinative (determining) and Y is determinable.

Each rule has two characteristics: accuracy and completeness.
Accuracy of determination $X \rightarrow Y$ shows to what extent X determines Y. It is defined as a proportion of occurrences of Y among the occurrences of X:
$A(X \rightarrow Y) = N(X\ Y) / N(X)$, where
$A(X \rightarrow Y)$ is accuracy of determination,
$N(X)$ is a number of objects having feature X and
$N(X\ Y)$ is a number of objects having both features X and Y.
Completeness of determination $X \rightarrow Y$ shows which part of cases having Y can be explained by determination $X \rightarrow Y$. It is a percentage of occurrences of X among the occurrences of Y:
$C(X \rightarrow Y) = N(X\ Y) / N(Y)$, where
$C(X \rightarrow Y)$ is completeness of determination,
$N(Y)$ is a number of objects having feature Y and
$N(X\ Y)$ is a number of objects having both features X and Y.
Both accuracy and completeness can have values from 0 to 1. Value 1 shows maximal accuracy or completeness, 0 means that rule is not accurate or complete at all. Value between 0 and 1 shows quasideterminism.

If all objects having feature X have also feature Y then the determination is (maximally) accurate. In case of accurate determination $A(X \rightarrow Y) = 1$ (100%).
Majority of rules are not accurate. In case of inaccurate rule $A(X \rightarrow Y) < 1$.

In order to make determination more (or less) accurate complementary factors are added into the first part of a rule. Adding factor Z into rule $X \rightarrow Y$ we get a rule $XZ \rightarrow Y$.
Contribution of factor Z to accuracy of rule $XZ \rightarrow Y$ is measured by increase of accuracy $\Delta A(Z)$ caused by addition of factor Z into rule $X \rightarrow Y$: $\Delta A(Z) = A(XZ \rightarrow Y) - A(X \rightarrow Y)$.
Contribution to accuracy falls into interval from -1 to 1.
If $\Delta A(Z)>0$ then Z is a positive factor. Addition of positive factor makes rule more accurate, sometimes the resultant rule is (maximally) accurate.
If $\Delta A(Z)<0$ then Z is a negative factor. Addition of negative factor decreases rule's accuracy, sometimes until zero.

If $\Delta A(Z)=0$ then Z is a zero (or inessential) factor. Addition of zero factor does not change rule's accuracy.

If $C(X\rightarrow Y)=1$ (100%) then the rule $X\rightarrow Y$ is (maximally) complete. It means that Y is always explained by X.
In case of incomplete rule $C(X\rightarrow Y)<1$, it means that X does not explain all occurrences of Y.

Contribution of factor Z to completeness of rule $XZ\rightarrow Y$ is measured by increase of completeness $\Delta C(Z)$ by addition of factor Z into rule $X\rightarrow Y$: $\Delta C(Z) = C(XZ\rightarrow Y) - C(X\rightarrow Y)$.
Contribution of whatever factor to completeness is negative or zero [8].

1.2 Analysis of Rules: main task and problems

Initial data matrix is given and some feature Y. The goal is to describe Y (possibly) completely by non-intersecting[1] (possibly) accurate rules. Rules are used for explanation and forecasting.

Completenesses of non-intersecting rules can be summed up. Reaching 100% the coverage is found. The way used to guarantee that rules do not intersect is to include all argument-attributes into all rules i.e. to find all existing (in class Y) value combinations of attributes that form argument. Then every object having Y is covered with exactly one rule. Each rule may cover several objects. Consequently, number_of_rules ≤ number_of_objects.

It may happen that there is a contradiction in the data. This is the situation where chosen set of attributes does not determine object's affiliation uniquely, the same combination of factors (i.e. attributes with certain values) leads to different classes; for example 2/3 of cases belong to class1 and 1/3 belong to class2.
In case of contradiction complete coverage by accurate rules is not achievable. If we accept only accurate rules, then overall completeness remains under 100%. If we want to get 100% coverage then it contains rule(s) (of conflicting combinations) with accuracy less than 100%.

1.3 Place of determinacy analysis

In our opinion the task of DA is a task of machine learning with certain constraint.
The task of inductive learning is to find (minimal) set of classification rules (this set is called description) that cover all learning examples (i.e. objects) without contradictions. [9] Description is consistent if each object is covered by the rule(s) of only one class (consistency condition). Description is complete if all objects are covered at least by one rule (completeness condition).
In our opinion the task of DA is to cover only one class (Y) by non-contradictory rules. It is a sub-task of machine learning task.
In case of approach to exclude intersecting rules we can define different completeness condition for DA: to cover every object by only one rule. It is possible because all value combinations of attributes forming argument are found. Thus every rule represents one or more objects that are identical within limits of attributes under consideration. The task of machine learning does not exclude intersecting rules.
DA gives also possibility to loosen the consistency condition and to find rules that are not maximally accurate (i.e. hold with some probability under 100%) and thus allow contradictions.
Finding rules for all alternative classes (i.e. all value combinations of attributes forming Y) DA solves the machine learning task (with different completeness condition).

Next the relations with terminology used in association rule mining are shown. According to [10]:
An association rule is an implication of the form $X\Rightarrow Y$, where $X\subset\Im$, $Y\subset\Im$ and $X\cap Y=\varnothing$; $\Im$ is a set of items.
The rule $X\Rightarrow Y$ holds in the transaction set D with confidence c if c% of transactions in D that contain X also contain Y.
The rule $X\Rightarrow Y$ has support s in the transaction set D if s% of transactions in D contain $X\cup Y$.

Hence the confidence of a rule is a percentage of rules containing both X and Y among rules containing X, i.e. it is the same as accuracy of determination (see 1.1 for definition).
The support of a rule is a percentage of rules containing both X and Y against number of transactions in the whole database. In DA the completeness of determination is a percentage of rules containing both X and Y against number of objects (transactions) containing Y. Therefore we can say that completeness of determination (in DA) corresponds to the support of a rule in class Y. Rules' support against whole database is not calculated in DA as the task is to find only rules of class Y.

Authors of DA compare DA with classic factor analysis and other classic approaches [6], [7].

2. About realisation of DA method

DA is realised in software package "ДА-система" by "Контекст Медиа". The following overview is about its (Russian) version 4.0 for Windows. Materials [6], [7], [11] have been used also.
For convenience "ДА-система" is called DA-system or DAS here.

[1] Theory of DA [8] covers intersecting rules also, but to our knowledge this wider approach is not realised in original application of DA. For that reason we consider here only the part of theory dealing with non-intersecting rules.

2.1 Possibilities of DA-system

In addition to the analysis of rules DA-system enables to create distribution tables and to find usual statistics (like in Excel). It has original Dictionary of Variables and possibility to create secondary variables.
The following overview is about analysis of rules in DAS.

DA-system enables to find rules and gives information about factors that make them more or less accurate. For the numerical attributes DAS has an "optimisation procedure" to find range(s) of values so that the rule has accuracy at least over some given threshold and the maximal possible completeness.

User of DAS determines the class under consideration and attributes that form the first part of rule(s).
For every rule four characteristics are found: Accuracy, Completeness, Number of rules' Applications (i.e. the number of objects having determining factors in whole dataset or context) and Number of rules' Confirmations (i.e. the number of objects having these factors among objects belonging to the class under investigation).
For the whole system totals of these parameters are found: total Accuracy (weighted average), total Completeness (sum of rules' completenesses), total Number of Applications and total Number of Confirmations (both are sums of rules' characteristics).
For each factor in (every) rule two characteristics are found: its contribution to accuracy and its contribution to completeness (of a rule).
The order of factors does not have an effect on calculation of contributions. Each contribution is computed regarding all other factors, not only the ones preceding that certain factor.
User can set restrictions to the rules by accuracy, completeness, contribution to accuracy and contribution to completeness. All four apply to the individual rules, not to the separate factors in rules nor to the whole system. If at least one of those four does not fit, the rule is not included into the result.

DAS enables to find only additive system of rules (i.e. rules do not intersect) where all rules have equal number of factors (from the same set of attributes). By default DAS finds all existing combinations of factor-attributes irrespective of their class affiliation i.e. combinations belonging to other class(es) are also found. Such system is a complete system of rules (where sum of rules' completenesses is 100%). In order to suppress the rules of other classes the required completeness (or required accuracy) of rules has to be set to >0 by user. In such case system remains complete. Using higher threshold the completeness is not guaranteed.

By default rules can contain positive, zero and negative factors. It is possible to get rules that consist of positive (and zero) factors only by setting requirement that the contribution to accuracy has to be >0 (≥ 0). Other rules are excluded, therefore found system looses completeness.

System of rules can be maximally accurate (average accuracy =100%) if there are no contradictions in data. Accurate system can be required also by user. In order to find only accurate rules the required accuracy has to be set to 1. System got by applying such restriction may be incomplete.

2.2 Deficiencies of DA-system

Have been analysed DA-system we have following observations.
Unfortunately DA-system has no possibility to set restrictions to the individual factors and get rules with different number of factors. This is due to technical reasons. By our opinion such lack makes DAS limited compared to theory of DA by Chesnokov [8].
DAS (4.0) limits the number of attributes (factors) in rules to 5, due to technical reasons. Always it is not enough to describe a class. Next chapter gives an overview of amount of calculations needed (in the worst case). The current version of DAS (5.0) is claimed to overcome this limitation.
A minor drawback is that DAS finds rules for only one class at a time; while by default it finds value combinations of others classes anyway (without showing their class affiliation). It could be more convenient to find rules for all alternative classes (by the same variable), but this exceeds bound of task of DA (as defined in 1.2).
An important disadvantage is that DAS has no automated search strategies to find better solutions. It only helps manual search (by making calculations), but gives no advice for the subsequent search direction. The system does not say which attributes should be included into or excluded from analysis. Quite probably this comes from the way DA-system solves the task. It seems that DAS does not use backtracking search algorithm, but implements database queries.
Also, the user manual does not include the methodology how to get needed result with DAS. Probably that knowledge is sold at special training courses.

3. Discussion about number of frequencies needed to compute characteristics of rules

In order to compute accuracy and completeness for the rule $X \rightarrow Y$ it is necessary to find two frequencies: the number of times X occurs in class Y and the number of times X occurs in a whole set (context). If X consists of more than one factor (attribute with certain value) then it is interesting to find also contribution to accuracy and contribution to completeness of each factor. For that we need (two) frequencies of value combination without that certain factor. The number of sub-combinations to look through is equal to the number of factors in the rule. Thus,

to find characteristics for an individual rule, 2 * (1 + number_of_factors) frequencies are needed. Additionally we need the frequency of class Y which is equal for all rules of that class.
Looking at all rules of class Y (or the whole set) they may share the same sub-combinations. If we remember frequencies of them we do not need to count them again. Then the number of frequencies to count is lesser than number_of_rules * 2 * (1 + number_of_factors).

3.1 Example of calculations

Data from [12] will be used in example (see Table 1). This data table contains 8 objects described by 4 attributes. The last attribute shows object's affiliation to certain class. Attribute Hair has three alternative values, attributes Height, Eyes and Class have two possible values each.

Table 1. Quinlan's data

Height	Hair	Eyes	Class
tall	dark	blue	+
short	dark	blue	+
tall	blond	blue	-
tall	red	blue	-
tall	blond	brown	+
short	blond	blue	-
short	blond	brown	+
tall	dark	brown	+

The purpose is to determine Class '–' by attributes Eyes and Hair.
DA-system finds two rules that contain both attributes:
R1) Hair.red & Eyes.blue → Class.–
R2) Hair.blond & Eyes.blue → Class.–
Only restriction to the solution given by user is that all rules must have accuracy >0 to exclude rules with non-existing (in given class) value combinations of chosen attributes.
Number of objects in Class '–' is three (N(Class.–) = 3).

Look at the first rule (R1).
In order to compute accuracy and completeness we need 2 frequencies
- N(Hair.red & Eyes.blue & Class.–) = 1
- N(Hair.red & Eyes.blue) = 1

and get
- A(Hair.red & Eyes.blue → Class.–) =
  = N(Hair.red & Eyes.blue & Class.–) / N(Hair.red & Eyes.blue) = 1/1 = 1
- C(Hair.red & Eyes.blue → Class.–) =
  = N(Hair.red & Eyes.blue & Class.–) / N(Class.–) =
  = 1/3 = 0,33

In order to find contributions of factor Hair.red we need 2 frequencies
- N(Eyes.blue& Class.–) = 3
- N(Eyes.blue) = 5

and get
- A(Eyes.blue → Class.–) = 3/5 = 0,6
- C(Eyes.blue → Class.–) = 3/3 = 1
- ΔA(Hair.red) = A(Hair.red & Eyes.blue → Class.–) – A(Eyes.blue → Class.–) = 1 – 0,6 = 0,4
- ΔC(Hair.red) = C(Hair.red & Eyes.blue → Class.–) – C(Eyes.blue → Class.–) = 0,33 – 1 = -0,67

In order to find contributions of factor Eyes.blue we need 2 frequencies
- N(Hair.red & Class.–) = 1
- N(Hair.red) = 1

and get
- A(Hair.red → Class.–) = 1/1 = 1
- C(Hair.red → Class.–) = 1/3 = 0,33
- ΔA(Eyes.blue) = A(Hair.red & Eyes.blue → Class.–) – A(Hair.red → Class.–) = 1 – 1 = 0
- ΔC(Eyes.blue) = C(Hair.red & Eyes.blue → Class.–) – C(Hair.red → Class.–) = 0,33 – 0,33 = 0

It is necessary to count 2 frequencies for (primary) characteristics and 2*2 frequencies for contributions, 6 frequencies in total. The number of frequencies needed is 2*(1 + number_of_factors) indeed.

Look at the second rule (R2).
We need 2 frequencies of rule itself and 2 frequencies of rule without factor Eyes.blue.
In order to find the contributions of factor Hair.blond we need again the frequencies already found for calculating contributions of factor Hair.red of the first rule: N(Eyes.blue&Class.–) and N(Eyes.blue). If we remember those frequencies, we need to count 6–2=4 frequencies instead of 6.

3.2 Calculations in case of different number of values per class

Let us have a rule X→Y, where X consists of P attributes: $x_1x_2...x_p \rightarrow Y$. Let v_1, v_2, ..., v_p denote the numbers of different values of attributes.
The maximal possible number of different combinations of attributes' values is $v_1 * v_2 * ... * v_{p-1} * v_p = \prod_{i=1}^{p} v_i$. In order to compute accuracy and completeness it is necessary to find two frequencies of each (existing) combination: the number of occurrences in class Y and the number of occurrences in a whole set. This doubles the before mentioned expression: $2 * \prod_{i=1}^{p} v_i$.
In order to compute contribution to accuracy and contribution to completeness of each factor for all rules it is necessary to find (those two) frequencies of all combinations got by eliminating one factor. If (two or more) rules differ only by one factor they have a common sub-combination without that factor. The number of rules having certain common sub-combination is maximally the number of different values of that omitted attribute (factor).

In the worst case we have to find the frequencies for $(v_2*v_3*...*v_{p-2}*v_{p-1}*v_p + v_1*v_3*...*v_{p-2}*v_{p-1}*v_p + ... + v_1*v_2*v_3*...*v_{p-2}*v_p + v_1*v_2*v_3*...*v_{p-2}*v_{p-1}) = \sum_{j=1}^{p} \frac{\prod_{i=1}^{p} v_i}{v_j}$ combinations, two frequencies for each!

The overall maximal possible number of frequencies to find is $2*\prod_{i=1}^{p} v_i + 2*\sum_{j=1}^{p} \frac{\prod_{i=1}^{p} v_i}{v_j}$.

This is a theoretical maximum for the certain class (Y) and also for the whole analysed set. The real number is likely much smaller. Usually the database does not contain all value combinations of attributes. Even if some (analysed) subset of attributes contains all of them, they probably do not belong to the same class.

3.3 Calculations in case of equal number of values per class

Assume that all attributes (included into analysis) have equal number of different values V. In such case the maximal number of rules is V^P and number of rules (combinations) without one factor is $P*V^{P-1}$. Then it is necessary to find 2 frequencies for each of $(V^P + P*V^{P-1})$ combinations.

If we have binary factors then for 5 factors the maximal number of rules is 32 and the maximal number of frequencies to find is 224. In case of 6 binary factors there can be maximally 64 rules and 512 frequencies and in case of 7 factors 128 rules and 1152 frequencies.

Looking at factors with 10 different values these theoretical maximums are much bigger. In case of 5 factors 10^5 rules and $3*10^5$ frequencies; in case of 6 factors 10^6 rules and $3,2*10^6$ frequencies; in case of 7 factors already $10*10^6$ rules and $34*10^6$ frequencies.

Probably due to such big numbers, DA-system limits the number of attributes (factors) in a rule to 5. In order to include more factors, different approach to the solution is needed.

3.4 Other realisation solution

There exists a realisation by us in which the possible number of factors is in hundreds. We used hash tables for collecting frequencies (N(X) and N(XY)). Huge increase in number of factors was possible because the real number of rules is always ≤ number_of_objects as we show in section 1.2.

4. Possible developments of DA: different approach

It is reasonable to allow rules with different number of factors in argument (keeping them non-intersecting at the same time), to exclude inessential factors. In such case factors are added into rules in some order (not all at the same time) and their contributions to accuracy and completeness are calculated regarding those factors only that have been added into rule earlier. Latter factors cannot be considered in those calculations.

Thus we can conclude that depending on the order the attributes are added into argument different coverages may be found for the same class and the same data. Next example demonstrates two different solutions for the same task, consisting of (non-intersecting) rules having different number of factors.

4.1 Example of rules

In this example Quinlan's data (see Table 1) will be used again.

The purpose is to determine Class '–'. This class consists of three objects: N(Y)=3.

First we will add attributes in the order they are given in the table (i.e. Height, Hair, Eyes).

Rules with attribute Height (only):

Height	N(X)	N(XY)	A	C	ΣC
short	3	1	1/3	1/3	
tall	5	2	2/5	2/3	

Neither of two (candidate) rules is accurate.

Add next attribute (Hair) into both rules:

Height	Hair	N(X)	N(XY)	A	C	ΣC
short	dark	1	0	0	0	
short	red	0				
short	blond	2	1	1/2	1/3	
tall	dark	2	0	0	0	
tall	**red**	**1**	**1**	**1**	**1/3**	1/3
tall	blond	2	1	1/2	1/3	

Theoretically there can be 6 different value combinations of those two arguments. One of those 6 does not exist in the data (Height.short&Hair.red). Two of them do not exist in the given class (Height.short&Hair.dark; Height.tall& Hair.dark). Three others are proper for the class. One of them (Height.tall&Hair.red) has accuracy 1 and needs no additional factors. This rule covers 1/3 of the objects of the class. We will sum up completenesses of accurate rules, with hope to reach to 100% coverage (by accurate rules). Two rules having accuracy between 0 and 1 have to be expanded again. Next add attribute Eyes into those two rules:

Height	Hair	Eyes	N(X)	N(XY)	A	C	ΣC
short	**blond**	**blue**	**1**	**1**	**1**	**1/3**	2/3
short	blond	brown	1	0	0	0	
tall	**blond**	**blue**	**1**	**1**	**1**	**1/3**	1
tall	blond	brown	1	0	0	0	

Two rules of four have accuracy 1 and both have completeness 1/3. We have found three (accurate) rules for the class '–' with overall completeness 1 (100%). Thus the class is completely described:

Height.tall&Hair.red → Class. – (C = 33%)

Height.short&Hair.blond&Eyes.blue → Class. – (C=33%)

Height.tall&Hair.blond&Eyes.blue → Class. – (C = 33%)

Now we will describe the same class by the same attributes, adding them in another order: Hair, Eyes, Height. Rules consisting of attribute Hair:

Hair	N(X)	N(XY)	A	C	ΣC
dark	3	0	0		
red	**1**	**1**	**1**	**1/3**	1/3
blond	4	2	2/4		

One rule (Hair.red) is accurate and is added to the result, it covers 1/3 of the class. Rule with Hair.dark has zero accuracy in given class and will be not expanded. Rule with Hair.blond has accuracy between 0 and 1, thus we add next attribute (Eyes) into it:

Hair	Eyes	N(X)	N(XY)	A	C	ΣC
blond	**blue**	**2**	**2**	**1**	**2/3**	1
blond	brown	2	0	0	0	

First of found rules has accuracy 1 and will be added to the result. It has completeness 2/3 and the summed completeness is now 100%. Class '–' is covered by (accurate) rules. At the same time we see that the other branch has zero accuracy and there is no reason to expand it.

This time class '–' is covered by two rules:

Hair.red → Class. – (C = 33%)

Hair.blond&Eyes.blue → Class. – (C = 67%)

As we can see, attribute Height is not necessary for distinction of classes.

We have shown that the same class can be (completely) covered by different sets (systems) of (non-intersecting) rules. The current realisation does not have possibility to get such solution.

5. Conclusion

This paper introduces the theory called Determinacy Analysis (DA) developed by S. Chesnokov. It is a method of analysis of rules targeted to find (possibly) complete coverage of one class of objects by (possibly) accurate rules. As treated here the task of DA is restricted to find only non-intersecting rules. Thus it is a subtask of machine learning. Also relation of DA with association rules is shown.

Original realisation of DA by "Контекст Медиа" is introduced. Its deficiencies are shown:

- inability of finding rules with different number of factors in rule;
- the number of factors is limited to 5;
- no use of automated search strategies;
- lacking of methodology of using.

In this paper we show how many calculations are needed in case of approach used in this original application. Even in such case the application can be more powerful, due to use of hashing.

From methodological side a different approach to the solution of the DA task was presented. This approach enables to find rules with different numbers of factors in rules and to test different orders of factors in order to get alternative sets (systems) of rules.

References

[1] S. V. Chesnokov, *Determination-analysis of social-economic data in dialogical regime* (Preprint. Moscow, All-Union Institute for Systems Research, 1980, in Russian).

[2] S. V. Chesnokov, *Determinacy analysis of social-economic data* (Moscow, Nauka, 1982, in Russian).

[3] S. V. Chesnokov, Determinacy analysis of social-economic data, *Sociological Studies*, 1980, #3, 179-189 (in Russian).

[4] P. A. Luelsdorff, S. V. Chesnokov, Determinacy Form as the Essence of Language, *Prague Linguistic Circle Papers*, 1996, v. 2, 205-234.

[5] S. V. Chesnokov, Determinacy Analysis and the search for diagnostic criteria in medicine (the case of comprehensive ultrasonography), *Ultrasonic Diagnostics*, 1996, # 4, 42-47 (in Russian).

[6] DALSolution software and technology. Questions and Answers. http://www.dalsolution.com/faq.htm, retrieved 27.02.2007 .

[7] ДА-система 4.0, версия 4.0 для Windows 95, Windows 98 и Windows NT. Вопросы-Ответы. ДА-система и технология анализа данных. © 1999 Фирма "Контекст".

[8] S. V. Chesnokov, Determinacy analysis of socio-economic data. Illustrative materials to lectures. Lecture 2: Rules. Lecture 3: Systems of rules. (Lomonosov Moscow State University, Faculty of Economics, Moscow, 2002, in Russian).

[9] M. Gams M. and N. Lavrac. Review of five empirical learning systems within a proposed schemata. I. Bratko and N. Lavrac (Eds.), *Progress in Machine Learning: Proceedings of EWSL 87*, Bled, Yugoslavia, May 1987, Wilmslow: Sigma Press, 46-66.

[10] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules, *Proc. of the 20th Int'l Conf. on Very Large Databases*, Santiago, Chile, Sep. 1994, 487-499.

[11] ДА-система 4.0 Руководство пользователя Версия 1.0 ©1998, 1999 ООО "Фирма "Контекст".

[12] J. R. Quinlan, Learning efficient classification procedures and their application to chess and games, Carbonell, J. G., Michalski, R. S., Mitchell, T. M.(Eds.), *Machine Learning. An Artificial Intelligence Approach*, Springer-Verlag, 1984, 463-482.

Appendix C

Lind, G. and Kuusik, R. (2008). Some Problems in Determinacy Analysis Approaches Development. *Proceedings of the 2008 International Conference on Data Mining (DMIN 2008), Las Vegas, Nevada, USA, July 14-17, 2008, Volume I,* pp. 102-108. CSREA Press.

Some Problems in Determinacy Analysis Approaches Development

G. Lind, and R. Kuusik
Department of Informatics, Tallinn University of Technology, Tallinn, Estonia

Abstract - *This paper gives an overview of a method called determinacy analysis – a method for finding systems of rules describing a class determined by user. There are two types of rule systems: additive and non-additive. Additive system is a system in which its rules do not intersect pairwise (do not cover the same objects/ transactions). In case of non-additive system they may intersect.*
In this paper, we explain what is a normal rule and for what it differs from rules created on the basis of closed sets. We show that both approaches suggested by the author of determinacy analysis do not guarantee the normality of rules, because it is not easy to say whether some factor is suitable in a sense it is positive regarding factors that will be added later. It means that the order the factors are selected has an impact to the normality of the result. Several examples are added.

Keywords: Determinacy analysis, Rules, Normal rule.

1 Introduction to Determinacy Analysis

Determinacy Analysis (DA) is a system of methods for the analysis of rules, it was created at the end of 70s. Its approach combines mathematical statistics and logic. DA's methodology and the underlying mathematics are developed by Russian scientist Sergei Chesnokov [1], [2].

DA-technology assists in obtaining regularities, explanations and prognostic rules.

DA enables to find rules for arbitrary given class, thereat the class is not determined by only one attribute, but can be described over many attributes using logical and comparison operators (and, or, not, >, <, etc.). Every attribute can have more than two values.

DA uses simple technique to identify a class' rule ($X \rightarrow Y$) – measure A (=n(XY)/n(X), where n(X) is a number of objects containing X and n(XY) is a number objects containing both X and Y). The rule identifies a class unambiguously if its A=1,0 (100%). The class is completely identified by some set of rules if their total C (=n(XY)/n(Y), where n(Y) is a number objects containing Y) =1 (100%). These measures will be introduced in Section 2.

For example, using data given in Table 1, we will describe persons belonging to class '-'. This class consists of three objects. First we will use attributes in the (freely chosen) order: Hair, Eyes, Height.

Table 1. Quinlan's data

Height	Hair	Eyes	Class
tall	dark	blue	+
short	dark	blue	+
tall	blond	blue	-
tall	red	blue	-
tall	blond	brown	+
short	blond	blue	-
short	blond	brown	+
tall	dark	brown	+

Rules containing attribute Hair only are given in Table 2.

Table 2. Rules consisting of attribute Hair

Hair	n(X)	n(XY)	A	C	ΣC
dark	3	0	0		
red	**1**	**1**	**1**	**1/3**	1/3
blond	4	2	2/4		

Rule Hair.red$\rightarrow$ Class.– is accurate (A=1) and needs no additional factors. This rule is included into the result. It covers 1/3 of the objects of the class (C=1/3). The completenesses C of accurate rules will be summed up, with hope to reach to 100% coverage (by accurate rules).

Rule with Hair.dark has zero accuracy (i.e. does not exist) in given class and will be not expanded.

Rule with Hair.blond has accuracy between 0 and 1, thus we expand it adding the next attribute (Eyes) into it (see Table 3).

Table 3. Rules consisting of attributes Hair and Eyes

Hair	Eyes	n(X)	n(XY)	A	C	ΣC
blond	**blue**	**2**	**2**	**1**	**2/3**	1
blond	brown	2	0	0	0	

The first of found rules has accuracy 1 and will be included into the result. Its completeness is 2/3. Now the summed completeness is 100%, thus the class '–' is (fully) covered by (accurate) rules.

As the found rules cover the class completely there is no reason to use the next attribute (Height), so the class is described without using it.

At the same time we can see that the other branch (Hair.blond and Eyes.brown) has zero accuracy and there is no reason to expand it.

Class '–' is covered by two rules:

- Hair.red → Class. – (C = 33%)
- Hair.blond&Eyes.blue → Class. – (C = 67%)

Now we will describe the same class by the same attributes, adding them in another order – the order they are given in the data table (i.e. Height, Hair, Eyes). This time the result is:

- Height.tall&Hair.red → Class. – (C = 33%)
- Height.short&Hair.blond&Eyes.blue → Class. – (C=33%)
- Height.tall&Hair.blond&Eyes.blue → Class. – (C = 33%)

This system of rules consits of three accurate rules with overall completeness 1 (100%). The class is completely described.

This example shows that the same class can be (completely) covered by different sets (systems) of (non-intersecting) rules. For each order of attributes we can get different set of rules.

Looking at the second set (system) of rules we can see that it contains redundant factors. Values of attribute Height are not necessary to identify the objects' class. Taking away the factors representing attribute Height we get the same system as in the first case (two last rules are identical exept for attribute Height and can be taken together with summed coverage). So the rules of the second system are too long (i.e. contain unnecessary, redundant factors) and therefore there are too much of them.

The problem is how to avoid including unnecessary factors into rules. Chesnokov calls rules containing only necessary information (as in the first system) normal rules.

Such manual approach as shown in our example here, was created in 80s by Chesnokov [2]. Later he has proposed a new approach of normal system of rules [3] (see section 2.3), but as we show in this paper, no way of avoiding inessential factors in the rules.

2 Overview of Determinacy Analysis

The overview of determinacy analysis is based on [1]-[5]. DA has been used in sociology [6], linguistics [7], medicine [8], and other areas (for complete list of references see [4] or [5]). It is familiar in Russia.

2.1 Place of Determinacy Analysis

In our opinion the task of DA is a task of machine learning with certain constraint.

The task of inductive learning is to find (minimal) set of classification rules (this set is called description) that cover all learning examples (i.e. objects) without contradictions [9]. Description is consistent if each object is covered by the rule(s) of only one class (consistency condition). Description is complete if all objects are covered at least by one rule (completeness condition).

In our opinion the task of DA is to cover only one class by non-contradictory rules. It is a sub-task of machine learning task.

DA gives also possibility to loosen the consistency condition and to find rules that are not maximally accurate (i.e. hold with some probability under 100%) and thus allow contradictions.

Next the relations with terminology used in association rule mining are shown. According to [10]:

An *association rule* is an implication of the form X⇒Y, where $X \subset \Im$, $Y \subset \Im$ and $X \cap Y = \emptyset$; $\Im$ is a set of items.

The rule X⇒Y holds in the transaction set D with *confidence c* if c% of transactions in D that contain X also contain Y.

The rule $X \Rightarrow Y$ has *support s* in the transaction set D if s% of transactions in D contain $X \cup Y$.

Hence the confidence of a rule is a percentage of rules containing both X and Y among rules containing X, i.e. it is the same as accuracy of determination (see section 2.2 for a definition).

The support of a rule is a percentage of rules containing both X and Y against the number of transactions in the whole database. In DA the completeness of determination is a percentage of rules containing both X and Y against the number of objects (transactions) containing Y (see 2.2). Therefore we can say that completeness of determination (in DA) corresponds to the support of a rule in class Y. Rules' support against the whole database is not calculated in DA as the task is to find only rules of class Y.

2.2 Determination and its characteristics

The main idea behind DA is that a rule can be found based on the frequencies of joint occurrence or non-occurrence of events. Such rule is called a determinacy or determination, and the mathematical theory of such rules is called determinacy analysis [4].

If it is observable that an occurrence of X is always followed by an occurrence of Y, this means that there exists a rule "If X then Y", or $X \rightarrow Y$. Such correlation between X and Y is called *determination* (from X to Y). Here X is *determinative (determining)* and Y is *determinable*.

Each rule has two characteristics: accuracy and completeness.

Accuracy of determination $X \rightarrow Y$ shows to what extent X determines Y. It is defined as a proportion of occurrences of Y among the occurrences of X:

$A(X \rightarrow Y) = n(X\ Y) / n(X)$, where

$A(X \rightarrow Y)$ is accuracy of determination,

$n(X)$ is a number of objects having feature X and

$n(X\ Y)$ is a number of objects having both features X and Y.

Completeness of determination $X \rightarrow Y$ shows which part of cases having Y can be explained by determination $X \rightarrow Y$. It is a percentage of occurrences of X among the occurrences of Y:

$C(X \rightarrow Y) = n(X\ Y) / n(Y)$, where

$C(X \rightarrow Y)$ is completeness of determination,

$n(Y)$ is a number of objects having feature Y and

$n(X\ Y)$ is a number of objects having both features X and Y.

Both accuracy and completeness can have values from 0 to 1. Value 1 shows maximal accuracy or completeness, 0 means that rule is not accurate or complete at all. Value between 0 and 1 shows quasideterminism.

If all objects having feature X have also feature Y then the determination is (maximally) accurate. In case of accurate determination $A(X \rightarrow Y) = 1$ (100%).

Majority of rules are not accurate. In case of inaccurate rule $A(X \rightarrow Y) < 1$.

In order to make a determination more (or less) accurate complementary factors are added into the first part of a rule. Adding factor Z into rule $X \rightarrow Y$ we get a rule $XZ \rightarrow Y$.

Contribution of factor Z *to accuracy* of rule $XZ \rightarrow Y$ is measured by the change of accuracy $\Delta A(Z)$ caused by addition of factor Z into rule $X \rightarrow Y$: $\Delta A(Z) = A(XZ \rightarrow Y) - A(X \rightarrow Y)$.

Contribution to accuracy falls into interval from -1 to 1.

If $\Delta A(Z)>0$ then Z is a positive factor. Addition of positive factor makes a rule more accurate, sometimes the resultant rule is (maximally) accurate.

If $\Delta A(Z)<0$ then Z is a negative factor. Addition of negative factor decreases rule's accuracy, sometimes until zero.

If $\Delta A(Z)=0$ then Z is a zero (or inessential) factor. Addition of zero factor does not change rule's accuracy.

If $C(X \rightarrow Y)=1$ (100%) then the rule $X \rightarrow Y$ is (maximally) complete. It means that Y is always explained by X.

In case of incomplete rule $C(X \rightarrow Y)<1$, it means that X does not explain all occurrences of Y.

Contribution of factor Z *to completeness* of rule $XZ \rightarrow Y$ is measured by the change of completeness $\Delta C(Z)$ caused by addition of factor Z into rule $X \rightarrow Y$: $\Delta C(Z) = C(XZ \rightarrow Y) - C(X \rightarrow Y)$.

Contribution of whatever factor to completeness is negative or zero [3].

2.3 System of rules

System of rules is a set of rules in form $S_q = \{x_i \to y \mid i=1,2,...,q\}$, where q is the number of rules.

Every system is characterised by average accuracy, summarised completeness and summarised capacity (the number of objects/cases covered by rules).

Accuracy of system of rules is:

$$A(S_q) = A((\bigcup_{i=1}^{q} x_i) \to y) = \frac{n(y\bigcup_{i=1}^{q} x_i)}{n(\bigcup_{i=1}^{q} x_i)}. \quad (1)$$

Completeness of system of rules is:

$$C(S_q) = C((\bigcup_{i=1}^{q} x_i) \to y) = \frac{n(y\bigcup_{i=1}^{q} x_i)}{n(y)}. \quad (2)$$

Capacity of system of rules is:

$$n(S_q) = n(\bigcup_{i=1}^{q} x_i). \quad (3)$$

System of rules $S_q = \{x_i \to y \mid i=1,2,...,q\}$ is *additive* when x_i-s pairwise do not intersect (i.e. do not cover the same objects). Capacity of additive system is equal to the sum of capacities of rules it consists of:

$$n(S_q) = n(\bigcup_{i=1}^{q} x_i) = \sum_{i=1}^{q} n(x_i). \quad (4)$$

In case of additive system previously given formulas can be simplified.

Completeness and capacity of additive system are just summed up completenesses and capacities of rules:

$$C(S_q) = C((\bigcup_{i=1}^{q} x_i) \to y) = \frac{\sum_{i=1}^{q} n(yx_i)}{n(y)} = \\ = \sum_{i=1}^{q} C(x_i \to y); \quad (5)$$

$$n(S_q) = \sum_{i=1}^{q} n(x_i). \quad (6)$$

Accuracy of additive system is not additive (i.e. equal to the sum of rules' accuracies). It is found as a weighted average:

$$A(S_q) = A((\bigcup_{i=1}^{q} x_i) \to y) = \frac{\sum_{i=1}^{q} n(yx_i)}{\sum_{i=1}^{q} n(x_i)} = \\ = \sum_{i=1}^{q} \frac{n(x_i)}{\sum_{k=1}^{q} n(x_k)} A(x_i \to y) = \\ = \sum_{i=1}^{q} A((\bigcup_{k=1}^{q} x_k) \to x_i) A(x_i \to y). \quad (7)$$

Rank of a rule is a dimension of its left side. Rule in form $z_1z_2...z_r \to y$ is called a rule of rank r ($r \geq 1$).

System of rules in form $z_1z_2...z_r \to y$ is called a system of rules of rank r by variables $z_1, z_2, ..., z_r$ relative to feature y.

Every system of rules of rank $r \geq 1$ by fixed set of r variables is additive [3].

System is called *complete* if its completeness is 1. System of rules of rank r by variables $z_1, z_2, ..., z_r$ is complete if it contains all existing rules in form $z_1z_2...z_r \to y$.

System is called *accurate* if its accuracy is 1. System is accurate when all of its rules are accurate.

Accurate rule has no negative factors, all factors are positive or zero factors [3].

Rule in form $z_1z_2...z_r \to y$ is called *normal rule* (of rank r) if all factors $z_1, z_2, ..., z_r$ are positive. Positive factor (called also binder) makes rule more accurate than it was without that factor.

System S_r consisting of all normal rules in form $z_1z_2...z_r \to y$ is called *normal system of rank r.* Normal system of rank r is additive i.e. its rules do not intersect pairwise.

Let be given m variables $z_1, z_2, ..., z_m$ and feature y. *Canonical system of order m* is a system of rules that joins all normal systems of rules in form $S_{jr} = \{z_{j1}z_{j2}\ldots z_{jr} \to y\}$ where $z_{j1}, z_{j2}, \ldots, z_{jr}$ – all possible combinations by r variables from set of variables $z_1, z_2, ..., z_m$.

In case of fixed r there are $\binom{m}{r} = \frac{m!}{r!(m-r)!}$ normal systems $S_{jr} = \{z_{j1}z_{j2}\ldots z_{jr} \to y\}$, j_r gets values from 1 to $\binom{m}{r}$.

Canonical system of order m contains 2^m-1 normal systems.

Variables $z_1, z_2, ..., z_m$ are called the basis of canonical system.

In general, canonical system of order m>1 is not additive.

2.4 Examples of different systems of rules

Here we present some examples of different systems of rules using data from [11]. This data table (see Table 1) contains 8 objects described by 4 attributes. The last attribute shows object's affiliation to certain class. Attributes Height, Eyes and Class have two possible values each, attribute Hair has three alternative values.

The purpose is to determine Class '+' by attributes Eyes and Hair.

Next four different systems of rules are given, based on the same data.

1) Additive system with fixed number of factors (r=2):

- Eyes.blue & Hair.dark → Class.+
  C = 40% (2 objects)

- Eyes.brown & Hair.dark → Class.+
  C = 20% (1 object)

- Eyes.brown & Hair.blond → Class.+
  C = 40% (2 objects)

The system is (maximally) accurate (overall accuracy is 100%) and (maximally) complete (sum of completenesses is 100%). Maximal completeness shows that all objects (belonging to class) are covered and maximal accuracy says that there is no need to add any more factors into analysis.

2) Additive system consisting of accurate rules of different rank (number of factors) could be:

- Hair.dark → Class.+
  C = 60% (3 objects)

- Hair.blond & Eyes.brown → Class.+
  C = 40% (2 objects)

or

- Eyes.brown → Class.+
  C = 60% (3 objects)

- Eyes.blue & Hair.dark → Class.+
  C = 40% (2 objects)

In both cases the system is accurate and complete and rules do not cover the same objects.

3) Normal system (of rank 2) would have no rules because each rule (of rank 2) has at least one factor with zero contribution to accuracy (recall that normal rules consist of positive factors only): Hair.dark is accurate alone, without Eyes.blue or Eyes.brown, also Eyes.brown is accurate without Hair.blond or Hair.dark.

4) Canonical system (of order 2) would have two rules of rank 1:

- Eyes.brown → Class.+
  C = 60% (3 objects)

- Hair.dark → Class.+
  C = 60% (3 objects)

These rules cover all objects belonging to Class '+'. One object (having Eyes.brown and Hair.dark) is covered twice, there from the +20% comes. Canonical system is non-additive and it allows to cover objects more than once.

3 The disadvantage of proposed DA approaches

Chesnokov suggests a *non-additive* system consisting of *accurate normal* rules of different rank as a solution [3]. In this chapter we show that proposed algorithm does not guarantee normal rules.

3.1 Problem

In [3] there is no description how to achieve a desired result – a non-additive system consisting of accurate normal rules. We may suppose that there is no heuristic search algorithm behind it. Having seen a realisation for finding additive (non-intersecting) system of rules [5] which finds rules by making database queries and deciding afterwards which ones of them are suitable [12] we may suppose that also in case of non-additive systems the result is achieved by searching through the unnecessarily large set of possible rules got by database queries.

An earlier, stepwise DA approach by Chesnokov [2] is not targeted on finding normal rules (although it contains concepts of positive, (negative) and zero factors).

Normal rule is not always accurate. Taking away whatever factor(s) from a normal rule, its accuracy decreases (because every factor in such rule has a positive contribution to the accuracy).

Also, accurate rule is not always normal (when it contains zero factor(s)).

An accurate rule that contains all possible zero factors is similar to the concept of closed set. The *closed set* is a set of items (i.e. factors) that has no superset with the same support (i.e. frequency) [13]. Addition of whatever factor decreases its coverage (and frequency). An accurate rule without zero factors i.e. normal accurate rule is similar to the concept of generator. *Generator* is the smallest itemset that determines a closed set i.e. the smallest set that covers the same set of objects [13]. A normal (accurate) rule contains only "essential" factors – factors without which the coverage (and frequency) is bigger and less accurate.

Zaki and Hsiao [13] do not search for generators, they propose a way to find all closed sets, not via generators (as done before them [14]), but directly. Closed sets are used to find rules between them. Our task is different: to find rules for determined class. In case of normal accurate rule its left side is a generator, such that determines only objects belonging to that class.

Looking for an accurate rule that contains all possible factors including zero factors it is not important on which steps the zero factors are included. Looking for a normal accurate rule we have to avoid including zero factors.

By our opinion this problem is a main technical problem in proposed DA approaches.

3.2 Some Examples

In case of stepwise approach it is not easy to say whether the current factor is suitable in a sense it is positive ("essential") regarding factors that will be added later. Adding a factor that makes rule more accurate it may occur later that the factor is inessential anyway.

For example (using data given in Table 1):

- Eyes.blue→ Class. –
  (A=3/5)

- Eyes.blue&Height.tall→ Class. –
  (A=2/3; ΔA=2/3-3/5=1/15>0)

- Eyes.blue&Height.tall&Hair.blond→ Class. –
  (A=1; ΔA=1-2/3=1/3>0)

Contributions (of new factors) to the accuracy are positive in both steps, but the resultant rule contains a zero factor (Height.tall) – the rule is accurate without it:

- Eyes.blue&Hair.blond→ Class. –
  (A=1)

We may consider making a control in the opposite direction:

- Height.tall→ Class. –
  (A=2/5)

- Height.tall&Eyes.blue → Class. –
  (A=2/3; ΔA=2/3-2/5=4/15>0)

Occurs that Height.tall and Eyes.blue together form a *normal rule (with accuracy below 1)*. But such finding does not guarantee that both factors remain positive after addition of the next one. Addition of whatever possible factor (i.e. values of attribute Hair) into the last rule does not produce any normal rule, although we get accurate rules:

- Height.tall&Eyes.blue&Hair.blond → Class. – (A=1)

- Height.tall&Eyes.blue&Hair.red → Class. – (A=1)

Both rules contain zero factor(s), Height.tall is a zero factor in both rules and Eyes.blue is a zero factor in the last rule.

It means that the fact that some factor has positive impact to the accuracy at the moment it is added into the rule does not guarantee that the factor retains its positiveness.

In our example we found a branch of a search tree that gives no normal accurate rules. There has to be a way to avoid entering such branches. As we have seen also, the factor's contribution to the accuracy (at the moment of addition) is not an adequate criterion.

4 Conclusion

In this paper we have introduced determinacy analysis (DA) – a method for finding system of rules describing a determined class of objects. Examples of different systems of rules are given. The main disadvantage of previously proposed DA approaches is brought out – the difficulty to get normal rules i.e. to avoid including zero ("inessential") factors into rules. This difficulty is illustrated by an example which shows that at the moment of addition factor's contribution to the rule's accuracy is not the same as in the final rule. The last one determines factor's suitability for normal rule i.e. is it an "essential" component of a rule.

We have an algorithm for finding a complete system of accurate rules, for both additive and non-additive cases.

Our stepwise approach for finding an additive system consisting of accurate rules of different rank, where the order of attributes is arbitrarily fixed and it is the same for all rules, is given in [12]. But our approach does not guarantee the normality of those rules – this is the task ahead.

In future work we will try to create algorithms both for additive and non-additive systems of normal accurate rules. At the moment we are in phase of experiments.

5 References

[1] S. V. Chesnokov. "Determination-analysis of social-economic data in dialogical regime". Preprint. Moscow, All-Union Institute for Systems Research, 1980 (in Russian).

[2] S. V. Chesnokov. "Determinacy analysis of social-economic data". Moscow: Nauka, 1982 (in Russian).

[3] S. V. Chesnokov. "Determinacy analysis of socio-economic data. Illustrative materials to lectures." Lecture 2: Rules. Lecture 3: Systems of rules. Lomonosov Moscow State University, Faculty of Economics, Moscow, 2002, unpublished (in Russian).

[4] DALSolution software and technology. Questions and Answers. Available: http://www.dalsolution.com/faq.htm, retreived: Feb 2007.

[5] DA-system 4.0, version 4.0 for Windows 95, Windows 98 and Windows NT. Questions and Answers. DA-system and technology of data analysis. „Context", 1999 (in Russian).

[6] S. V. Chesnokov. "Determinacy analysis of social-economic data"; Sociological Studies, #3, pp. 179-189, 1980 (in Russian).

[7] P. A. Luelsdorff, S. V. Chesnokov. "Determinacy form as the essence of language"; Prague Linguistic Circle Papers, v. 2, pp. 205-234, 1996.

[8] S. V. Chesnokov. "Determinacy Analysis and the search for diagnostic criteria in medicine (the case of comprehensive ultrasonography)"; Ultrasonic Diagnostics, # 4, pp. 42-47, 1996 (in Russian).

[9] M. Gams, and N. Lavrac. "Review of five empirical learning systems within a proposed schemata"; Progress in Machine Learning: Proceedings of EWSL 87, Eds: I. Bratko and N. Lavrac; Wilmslow: Sigma Press, pp. 46-66, May 1987.

[10] R. Agrawal, and R. Srikant. "Fast algorithms for mining association rules"; Proceedings of the 20th International Confernece on Very Large Databases, pp. 487-499, Sep 1994.

[11] J. R. Quinlan. "Learning efficient classification procedures and their application to chess and games"; Machine Learning. An Artificial Intelligence Approach, Eds: J. G. Carbonell, R. S. Michalski, T. M. Mitchell; Springer-Verlag, 1984.

[12] G. Lind, R. Kuusik. "Some ideas for determinacy analysis realisation"; Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing, ACTA Press, pp. 185-190, 2007.

[13] M. J. Zaki, C.-J. Hsiao. "CHARM: An efficient algorithm for closed itemset mining"; Proceedings of the Second SIAM International Conference on Data Mining, pp. 457-473, 2002.

[14] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. "Discovering frequent closed itemsets for association rules"; Proceedings of the 7th International Confernece on Database Theory, LNCS Vol. 1540, Springer-Verlag, pp. 398-416, Jan 1999.

Appendix D

Kuusik, R. and Lind, G. (2010). Some Developments of Determinacy Analysis. *Advanced Data Mining and Applications: The 6th International Conference on Advanced Data Mining and Applications (ADMA2010), Chongqing, China, November 19-21, 2010. LNAI 6440*, pp. 593-602. Berlin Heidelberg: Springer-Verlag.

Some Developments of Determinacy Analysis

Rein Kuusik and Grete Lind

Department of Informatics, Tallinn University of Technology
Raja 15, Tallinn 12618, Estonia
kuusik@cc.ttu.ee, grete@staff.ttu.ee

Abstract. This paper deals with development of Determincy Analysis (DA), a method for data mining. There are two approaches to DA that give a different result consisting of non-overlapping (exclusive each other) rules. The first method finds exactly one system consisting of rules that have equal length. The second, step by step approach enables to find very many different rule systems where the rules have different length. In the first case the rules contain a lot of redundant attributes, in the second case there are too many different (formally complete) systems of rules what makes the selection hard. A better result can be obtained by finding overlapping rules. This paper presents DA approach and technique that enables to find overlapping rules with different length and algorithm realizing it. Such approach for DA has not been created earlier.

Keywords: Determinacy Analysis, Data Mining, Overlapping rules.

1 Introduction

In this paper we deal with a data mining method called Determinacy Analysis. Determinacy Analysis (DA) is a system of methods for the analysis of rules that was created at the end of 1970s. Its approach combines mathematical statistics and logic. DA's methodology and the underlying mathematics are developed by a Russian scientist Sergei Chesnokov [1], [2].

The primary goal of DA is data mining specifying the objects of class Y to try to answer the question "who/what are they?" i.e. to describe them the best way using specified attributes and to output several sets of rules from which each one covers Y 100%. The user makes the decision which set or sets are the best for describing. Different orders of attributes and different sets of attributes are tried until the most suitable description in the user's opinion is found.

In the middle of 90s a next version of DA was developed by Chesnokov and it is known in Russia and the former republics of USSR as a software package DA-system[1] (Determinacy Analysis System or shortly DAS).

Known methods of DA can extract one or more rule systems where the rules do not overlap. The two main approaches to DA are: 1) DAS, 2) step by step method.

The initial data table and a feature Y (as a certain class) are given. The goal is to describe Y (possibly) completely by the non-overlapping (possibly) accurate rules.

[1] Software package DA-system (Russian version only) by „Context Media“ (http://www.context.ru), is described also in [3]

Different sets of rules may be found and different ways to find them can be used.

DAS extracts the rules with equal length (= the number of factors in the rule), all of them contain the same attributes – this is a simple way to get a set of non-overlapping rules.

In step by step approach described in [3] the extracted rules can have different lengths while they do not overlap. Attributes (factors) are added into the rules one by one. If a rule is accurate it will not be expanded by adding the next factor. At the same time the completenesses of found accurate rules are summed up. Reaching 100% the coverage is found. The user decides about the order in which the attributes are included into the rules, from the beginning until the situation when all objects of the class are covered. The order of factors (attributes) is essential, different orders lead to the different results.

Both approaches have problems with the amount of rules and the lenght of rules. We deal with these problems, analyze them and describe a new approach to DA which can solve these problems. It can extract a rule system with a lower number of rules and with shorter rules.

2 Determinacy Analysis

DA-technology assists in obtaining regularities, explanations and description rules. DA has been used in sociology [4], linguistics [5], medicine [6], and other areas (for the complete list of references see [7] or [8]).

The overview of determinacy analysis is based on [1], [2], [8], [9].

2.1 Determination and Its Characteristics

The main idea behind DA is that a rule can be found based on the frequencies of joint occurrence or non-occurrence of events. Such rule is called a determinacy or determination, and the mathematical theory of such rules is called determinacy analysis [8].

If it is observable that an occurrence of X is always followed by an occurrence of Y, this means that there exists a rule “If X then Y”, or $X \rightarrow Y$. Such correlation between X and Y is called a *determination* (from X to Y). Here X is a *determinative (determining)* and Y is a *determinable*.

The determinative (X) consists of one or more factors. A factor is an attribute with its certain value. Each attribute can give as many different factors as many different values it has. The factors coming from the same attribute are not contained in the same X.

Each rule has two characteristics: accuracy and completeness[2].

The accuracy of the determination $X \rightarrow Y$ shows to what extent X determines Y. It is defined as a proportion of occurrences of Y among the occurrences of X: $A(X \rightarrow Y) = n(X\ Y) / n(X)$, where $A(X \rightarrow Y)$ is the accuracy of determination, n(X) is

[2] In the beginning (in [1], for example) “accuracy” (Russian “*точность*”) was called “intensity” and “completeness” (“*полнота*”) was called “capacity” (“*емкость*”).

the number of objects having feature X and n(X Y) is the number of objects having both features X and Y.

The completeness of the determination $X \to Y$ shows which part of the objects having Y can be explained by the determination $X \to Y$. It is a percentage of occurrences of X among the occurrences of Y: $C(X \to Y) = n(X\ Y) / n(Y)$, where $C(X \to Y)$ is the completeness of determination, n(Y) is the number of objects having feature Y and n(X Y) is the number of objects having both features X and Y.

Both accuracy and completeness can have values from 0 to 1. Value 1 shows maximal accuracy or completeness, 0 means that rule is not accurate or complete at all. Value between 0 and 1 shows quasideterminism.

If all objects having feature X have also feature Y then the determination is (maximally) accurate. In case of an accurate determination $A(X \to Y) = 1$ (100%). Most of the rules are not accurate. In case of an inaccurate rule $A(X \to Y) < 1$.

In order to make a determination more (or less) accurate the complementary factors are added into the left part of the rule. Adding the factor Z into the rule $X \to Y$ we get a rule $XZ \to Y$.

The contribution of factor Z *to accuracy* of rule $XZ \to Y$ is measured by the increase of accuracy $\Delta A(Z)$ caused by addition of factor Z into the rule $X \to Y$: $\Delta A(Z) = A(XZ \to Y) - A(X \to Y)$. The contribution to accuracy falls into interval from -1 to 1.

If $\Delta A(Z)>0$ then Z is a positive factor. Addition of a positive factor makes the rule more accurate, sometimes the resultant rule is (maximally) accurate. If $\Delta A(Z)<0$ then Z is a negative factor. Addition of a negative factor decreases the rule's accuracy, sometimes until zero. If $\Delta A(Z)=0$ then Z is a zero (or inessential) factor. Addition of a zero factor does not change the rule's accuracy. *An accurate rule* contains no negative factors, all factors are positive or zero factors [9].

If $C(X \to Y)=1$ (100%) then the rule $X \to Y$ is (maximally) complete. It means that Y is always explained by X. In case of an incomplete rule $C(X \to Y)<1$, it means that X does not explain all occurrences of Y.

The contribution of factor Z *to completeness* of rule $XZ \to Y$ is measured by the increase of completeness $\Delta C(Z)$ by addition of factor Z into the rule $X \to Y$: $\Delta C(Z) = C(XZ \to Y) - C(X \to Y)$. The contribution of whatever factor to completeness is negative or zero [9].

The system of rules is a set of rules in the form of $S_q = \{x_i \to y \mid i=1,2,...,q\}$, where q is the number of rules. Every system is characterised by average accuracy, summarised completeness and summarised capacity (the number of objects covered by the rules).

The system of rules $S_q = \{x_i \to y \mid i=1,2,...,q\}$ is *additive* when x_i-s pairwise do not overlap. The completeness and capacity of the additive system are just summed up completenesses and capacities of the rules. The accuracy of the additive system is not additive (i.e. equal to the sum of rules' accuracies), it is found as a weighted average.

A system is called *complete* if its completeness is 1. A s*ystem* is called *accurate* if its accuracy is 1. A system is accurate when all of its rules are accurate.

The rank of a rule is a dimension of its left side. A rule in the form of $z_1z_2...z_r \to y$ is called a rule of rank r ($r \geq 1$). A system of rules in the form of $z_1z_2...z_r \to y$ is called a system of rules of rank r by variables $z_1, z_2, ..., z_r$ relative to the feature y. Every system of rules of rank $r \geq 1$ by fixed set of r variables is additive [9].

The theory of DA covers also the non-additive systems of (overlapping) rules, but to our knowledge no algorithms for finding such systems have been published.

2.2 Example

The following example illustrates step by step approach. The data from [10] will be used in example (see Table 1). This data table (object-attribute system) contains 8 objects described by 4 attributes. The last attribute shows the object's belonging to a certain class (feature Y). Attributes Height, Eyes and Class have two possible values each, attribute Hair has three alternative values.

We will describe persons (objects) belonging to class "+". This class consists of five objects (n(Y)=5).

Attributes will be added into the rules in the following (freely chosen) order: Hair, then Eyes and then Height.

Table 1. Quinlan's data

Height	Hair	Eyes	Class
tall	dark	blue	+
short	dark	blue	+
tall	blond	blue	–
tall	red	blue	–
tall	blond	brown	+
short	blond	blue	–
short	blond	brown	+
tall	dark	brown	+

Rules containing attribute Hair only are given in Table 2.

Table 2. The rules consisting of attribute Hair

Hair	n(X)	n(XY)	A	C	ΣC
dark	**3**	**3**	**1**	**3/5**	**3/5**
red	1	0	0	0	
blond	4	2	1/2	2/5	

The rule Hair.dark→Class.+ is accurate (A=1) and needs no additional factors. This rule is included into the result. It covers 60% of the objects of the class (C=3/5). The completenesses C of the accurate rules are summed up (ΣC), with hope to reach the 100% coverage (by accurate rules).

The rule with Hair.red has zero accuracy (i.e. does not exist) in given class and will not be expanded.

The rule with Hair.blond has accuracy between 0 and 1, thus we expand it by adding the next attribute (Eyes) into it (see Table 3).

The second of found rules (with Eyes.brown) has accuracy 1 and will be included into the result. Its completeness is 40% (C=2/5). Now the summed completeness is 100%, thus the class "+" is (fully) covered by the (accurate) rules.

Table 3. The rules consisting of attributes Hair and Eyes

Hair	Eyes	n(X)	n(XY)	A	C	ΣC
blond	blue	2	0	0	0	
blond	**brown**	**2**	**2**	**1**	**2/5**	**1**

At the same time we can see that the other branch (Hair.blond and Eyes.blue) has zero accuracy and there is no reason to expand it.

As the found rules cover the class completely there is no need to use the next attribute (Height), so the class is described without using it.

Class "+" is covered by two rules: 1) Hair.dark $\rightarrow$ Class.+ (C = 60%); 2) Hair.blond&Eyes.brown $\rightarrow$ Class.+ (C = 40%).

This is one possible description for class "+" beginning from the attribute Hair. In the "language" of DA it means that they are people with dark hair or with blond hair and brown eyes. The user has to know what to do with this knowledge. If "+" means "beauties", for example, then if the user is satisfied with such description then (s)he accepts it. If this knowledge is not satisfying then the user can change the order of attributes or add some new attributes.

3 Problems

First we present some examples of different systems of rules using the data from [10] (see Table 1). The purpose is to determine Class "+" by additive rules containing attributes Eyes and Hair.

1. The additive system consisting of accurate rules of different rank (number of factors) got using step by step approach can be (*S1*): 1) Hair.dark $\rightarrow$ Class.+, C = 60% (3 objects); 2) Hair.blond & Eyes.brown $\rightarrow$ Class.+, C = 40% (2 objects) or (*S2*): 1) Eyes.brown $\rightarrow$ Class.+, C = 60% (3 objects); 2) Eyes.blue & Hair.dark $\rightarrow$ Class.+, C = 40% (2 objects).

   S1 is found when attributes are included in the order: first Hair, then Eyes. In case of S2 the order is: first Eyes, then Hair. In both cases the system is accurate and complete and the rules do not cover the same objects. As we can see the different rule systems describe the objects practically differently giving a different knowledge about the subset of objects. The user decides how many rule systems is enough. If the description given by the rule systems is not good enough then some other set of attributes can be chosen or just some attributes can be exchanged.

2. The additive system with a fixed set of factors (rank r=2) got using DAS (*S3*): 1) Eyes.blue & Hair.dark $\rightarrow$ Class.+, C = 40% (2 objects); 2) Eyes.brown & Hair.dark $\rightarrow$ Class.+, C = 20% (1 object); 3) Eyes.brown & Hair.blond $\rightarrow$ Class.+, C = 40% (2 objects).

   The system is (maximally) accurate (the overall accuracy is 100%) and (maximally) complete (the sum of completenesses is 100%). Maximal completeness shows that all objects (belonging to the class) are covered and maximal accuracy says that there is no need to add any more factors into the analysis.

Looking at these examples we can see that an additive system containing rules with different lengths (both S1 and S2) has lesser number of rules than an additive system with a fixed rank and set of attributes (S3). S3 contains the longer rule from S1 and the longer rule from S2 and also a combination of shorter rules from S1 and S2.

It is desired that the rules were relatively short – then it is easier to interpret them. Also the number of rules has not to be very large – for the same reason.

The task to find an additive system of rules can be understood in several ways:

1. Originating from DAS, all the rules contain all given attributes and have equal length.
2. Originating from the step by step approach, it is possible to find a system of rules of different length. Different order of inclusion of attributes can give different systems of rules.

In the first case, on the assumption of the essence of the DAS approach, there is exactly one solution. If each rule has to contain all (given) attributes, then we cannot take away any factors from any rules and consequently we cannot change such system of rules. Determining a different set of attributes (for description) we get a different system which is also the only possible solution for that set of attributes (in case of the same requirement – to contain all attributes). In this case the purpose is to find the most suitable set of attributes i.e. exclude redundant attributes (like Height in our example) and include all necessary attributes to avoid contradictions – a situation where the set of attributes is not sufficient to distinguish between different classes. A suitable set can be found by testing different sets of attributes.

In the second case – non-overlapping rules (i.e. additive system) of different lenght (rank) – the constitution of the system depends on the order in which the factors are included into the rules. It might be hard to find a compact system of such rules. It would be good to allow non-fixed order of including factors.

The recent approach to DA emanates from logic: if one rule system does not satisfy then the next one will be considered etc. Such treatment comes from the fact that the rules of one rule system are derived from the uniform order of attributes and thus they are mutually related. It means that we cannot change the order of attributes in the emergent rule system. This is an important restriction, according to it the best description in a sense can form from single rules of different rule systems. One solution here is to generate all rule systems and according to some criteria find from them a cover i.e. a rule set (with completeness 100%) consisting of the shortest rules or a rule set with the least number of rules. This turns out to be very labor-consuming because all the possible sets and orders (permutations) of attributes should be found. Next we will show that simpler solutions exist. A better result can be obtained by finding overlapping rules i.e. non-additive system of rules.

4 New Approach

Here we describe a new approach to DA and a simple way for finding overlapping rules, the algorithm realizing it and a new very simple technique for rule detection. The system of rules it finds is non-additive (i.e. objects may be covered by more than one rule), accurate (i.e. consists of accurate rules only) and complete (i.e. covers all objects of determinable class) if there are no contradictions in the data (the algorithm can find them). The findable set of rules is possibly small – the potential rules are not included into the result if they cover only such objects that are covered already.

The selection criteria for choosing the next factor are based on frequencies, the maximal frequency in Fy_t, for example. In case of equal maximal frequencies in Fy the one having bigger frequency in Fx is preferred.

```
Algorithm
Determine tables X and Y
S0. t:=0; U_t:=∅
    If all the objects in Y are covered then Goto End
S1. Find frequencies in tables X_t and Y_t: Fx_t, Fy_t
S2. For each factor A such that Fy_t(A)=Fx_t(A) and all
    objects containing A are not covered by rule(s)
        output rule {U_i}&A, i=0,…,t
    If at least one new rule was found Goto S0
S3. Choose a new (free) factor U_t
    If there are no factors to add then
        {U_i}, i=0,…,t is a contradiction; Goto S0
    t:=t+1; extract subtable of objects containing U_t;
        Goto S1
End. System of rules is found
```

4.1 Example 1

In the examples we use the same Quinlan's data [10] as in section 2.2 (see Table 1), but this time a numerical representation of data is used. The coding used for attributes and their values is shown in Table 4. The initial data table is given in Table 5. Let X is X(8,3), X_{ij} = 1,...,3 and Y=4.1 {Y_i: 1,2,5,7,8} (i.e. class "+"). The frequencies of attributes' values for X and Y (Fx and Fy accordingly) are given in Table 6.

Table 4. The coding used for Quinlan data

Attribute	**Height**	**Hair**	**Eyes**	**Class**
Code	1	2	3	4
1	short	dark	blue	–
2	tall	red	brown	+
3		blond		

Table 5. The initial data table

i\j	1	2	3	4
1	2	1	1	1
2	1	1	1	1
3	2	3	1	2
4	2	2	1	2
5	2	3	2	1
6	1	3	1	2

7	1	3	2	1
8	2	1	2	1

Table 6. The frequencies for X and Y

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	5		1	2	3	2
	2	5	1	3		2	3	0	3
	3	0	4	0		3	0	2	0

For the attribute 2 value 1 (shortly 2.1) the frequencies in Fx and Fy are equal which means that all objects having 2.1 belong to Y. Hence we get a rule 2.1=3 (Hair.dark→Class.+). The frequency after "=" shows that the rule covers three objects (namely objects 1, 2 and 8) – this is additional information. Also for 3.2 the frequencies are equal and we get another rule 3.2=3 (Eyes.brown→Class.+) that also covers three objects (5, 7 and 8).

By those two rules all the objects belonging to Y are covered. The found rules are overlapping, both cover object 8.

In order to demonstrate other steps of the algorithm another example is presented.

4.2 Example 2

The same data as in the previous example is used (see Table 5). This time Y=4.2 {Y_i: 3,4,6} (class "–"). The frequency tables Fx and Fy are given in Table 7.

Table 7. The frequencies for X and Y

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	3	3	5		1	1	0	3
	2	5	1	3		2	2	1	0
	3	0	4	0		3	0	2	0

For 2.2 the frequencies in Fx and Fy are equal. The rule 2.2=1 (Hair.red→Class.–) covers object 4.

The "free" frequencies i.e. the frequencies over non-covered objects (in Y) after extracting the first rule are shown in Table 8.

Table 8. Free frequencies for Y after extracting first rule

Fy	Kj \ j	1	2	3
	1	1	0	2
	2	1	0	0
	3	0	2	0

From here the factor starting a new rule is chosen by maximal frequency in Fy. As there are two (equal) maximal frequencies, the choice is made by bigger frequency in Fx where the factor 3.1 has frequency 5 and 2.3 has frequency 4. Thus 3.1 is selected.

Next the objects having 3.1 are extracted (see Table 9) and the frequencies for X and Y over extracted data are found (see Table 10). The frequencies of covered objects (Fc) are given in Table 11.

Table 9. Extract by 3.1

i \ j	1	2	3	4
1	2	1	1	1
2	1	1	1	1
3	2	3	1	2
4	2	2	1	2
6	1	3	1	2

Table 10. The frequencies of extracted data

Fx	Kj \ j	1	2	3	Fy	Kj \ j	1	2	3
	1	2	2	5		1	1	0	3
	2	3	1	0		2	2	1	0
	3	0	2	0		3	0	2	0

Table 11. The frequencies of covered objects after extracting the first rule

Fc	Kj \ j	1	2	3
	1	0	0	1
	2	1	1	0
	3	0	0	0

In Table 10 we can see two potential rules, (3.1) with 2.2 and (3.1) with 2.3. Looking into Fc (Table 11) we see that 2.2 has equal frequency here indicating that the object(s) containing factor 2.2 is(are) already covered by the found rules. Therefore 2.2 does not produce a new rule (rule 3.1&2.2=1 would be redundant covering the same object as rule 2.2=1). The factor 2.3 is suitable for ending a rule, its frequency in Fc is less than 2. So, the second rule is 3.1&2.3=2 (Eyes.blue&Hair.blond→Class.–), it covers objects 3 and 6.

Table 12. The frequencies of covered objects after extracting the second rule

Fc	Kj \ j	1	2	3
	1	1	0	3
	2	2	1	0
	3	0	2	0

The two rules we have found cover all the objects belonging to Y. We detect it turning back to the initial level and comparing initial Fy with Fc (after extracting the last rule – see Table 12): all the frequencies in both tables are equal.

5 Conclusion

This paper gives an overview of Determinacy Analysis, introducing the theory and different approaches. The theory by Chesnokov treats both additive and non-additive systems of rules. Solutions for finding additive systems of rules are known earlier. Non-additive systems allowing overlapping rules could be more compact, but to our knowledge no method or application of DA for non-additive systems is published. In this paper we propose a method and the corresponding algorithm for finding a non-additive system of rules. Our algorithm is based on frequency tables which makes it easy to detect a potential rule and whether it covers any uncovered object. One rule system is found in which the number of rules is not bigger (and usually is smaller) than in case of DAS or step by step method and the rules are shorter also. In addition, the algorithm is able to detect a contradiction – the situation where identically described objects belong to different classes.

Realizing the algoritm it is possible to apply several different principles for selecting the next factor, one possibility is to let the user make the decision. Selection criteria is the direction for the future work.

References

1. Chesnokov, S.V.: Determination-Analysis of Social-Economic Data in Dialogical Regime. Preprint. All-Union Institute for Systems Research, Moscow (1980) (in Russian)
2. Chesnokov, S.V.: Determinacy Analysis of Social-Economic Data. Nauka, Moscow (1982) (in Russian)
3. Lind, G., Kuusik, R.: Some Ideas for Determinacy Analysis Realisation. In: Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing, pp. 185–190. ACTA Press (2007)
4. Chesnokov, S.V.: Determinacy Analysis of Social-Economic Data. Sociological Studies 3, 179–189 (1980) (in Russian)
5. Luelsdorff, P.A., Chesnokov, S.V.: Determinacy Form as the Essence of Language. Prague Linguistic Circle Papers 2, pp. 205–234 (1996)
6. Chesnokov, S.V.: Determinacy Analysis and the Search for Diagnostic Criteria in Medicine (the Case of Comprehensive Ultrasonography). Ultrasonic Diagnostics 4, 42–47 (1996) (in Russian)
7. Main scientific works of S.V. Chesnokov (in Russian), http://www.context.ru/publications
8. DA-system 4.0, version 4.0 for Windows 95, Windows 98 and Windows NT. Questions and Answers. DA-system and Technology of Data Analysis. "Context" (1999) (in Russian)
9. Chesnokov, S.V.: Determinacy Analysis of Socio-Economic Data. Illustrative Materials to Lectures. Lecture 2: Rules. Lecture 3: Systems of Rules. Lomonosov Moscow State University, Faculty of Economics, Moscow (2002) (unpublished, in Russian)
10. Quinlan, J.R.: Learning Efficient Classification Procedures and Their Application to Chess and Games. In: Carbonell, J.G., Michalski, R.S., Mitchell, T.M. (eds.) Machine Learning. An Artificial Intelligence Approach, 463–482. Springer-Verlag (1984)

Appendix E

Kuusik, R. and Lind, G. (2011). New Developments of Determinacy Analysis. *Advanced Data Mining and Applications - 7th International Conference: ADMA 2011, Beijing, China, December 17-19, 2011. II; LNCS 7121*, p. 223−236. Springer.

New Developments of Determinacy Analysis

Rein Kuusik and Grete Lind

Department of Informatics, Tallinn University of Technology
Raja 15, Tallinn 12618, Estonia
kuusik@cc.ttu.ee, grete@staff.ttu.ee

Abstract. This paper deals with development of Determincy Analysis (DA), a method for knowledge discovery. There are three approaches to DA that give different results. The first method finds exactly one system of non-overlapping rules, all with equal length. The second, step by step approach enables to find very many different systems of non-overlapping rules where the rules have different length. The third approach can find one system of overlapping rules. Analysis of these approaches showed that shorter rule systems contain rules which are not contained in other rules. It means that we have to find only all these rules (so called determinative set of rules, DSR) and describe given data table on their basis. This paper presents a new DA approach based on the DSR and algorithm for finding them. Several new tasks are formulated: to generate rule systems with specific features or the shortest or minimal rule system etc.

Keywords: Determinacy Analysis, Data discovery, Rule system, Overlapping rules, Determinative set of rules.

1 Introduction to DA and the problem statement

In this paper we deal with the developments of the knowledge discovery method called Determinacy Analysis (DA), the system of methods for the analysis of rules that was created at the end of 1970s. Its approach combines mathematical statistics and logic [1-2].

DA has been arisen from the analysis of frequency tables. The goal of DA is to describe a given selection of objects (class Y) (Who/what are they? How can we describe them? What distinguishes them from others?), not to classify an unknown object. Therefore it is necessary to know which attributes are more determining than the others, not by attribute's descriptive power (for that purpose we could use, for example, entropy like in ID3 or its developments), but from the viewpoint of typical associations of attributes and attributes that are not necessary for describing the system of objects (Y).

Known methods of DA can extract one or more rule systems where the rules do not overlap. The two such kind approaches to DA are: 1) step by step method, 2) DAS.

The initial data table and a feature Y (as a certain class) are given. The goal is to describe Y (possibly) completely by the non-overlapping (possibly) accurate rules.

In step by step approach described in [3] the extracted rules can have different lengths while they do not overlap. Attributes are added into the rules one by one. If a rule is accurate it is not expanded by adding the next factor (i.e. attribute with certain value). At the same time the completenesses of found accurate rules are summed up. Reaching 100% the coverage is found. The user decides about the order in which the attributes are included into the rules, from the beginning until the situation when all objects of the class are covered. The order of attributes is essential, different orders lead to the different results.

DAS extracts the rules with equal length, all of them contain the same attributes – this is a simple way to get a set of non-overlapping rules.

Both approaches have problems with the amount of rules, zero factors in rules and the length of rules.

The approach of DA which can find one rule system consisting of overlapping rules has been developed [4]. But there is a problem with the best criteria selection in the process of choosing the next attribute for the rule, there is no guarantee that the extracted rule system will be the shortest one (with the lowest number of rules).

We deal with these problems, analyze them and describe a new approach to DA which can solve these problems.

1.1 Determination and Its Characteristics

DA-technology assists in obtaining regularities, explanations and description rules. DA has been used in sociology [5], linguistics [6], medicine [7], and other areas (for the complete list of references see [8] or [9]).

The overview of determinacy analysis is based on [1-2], [9-10].
The main idea behind DA is that a rule can be found based on the frequencies of joint occurrence or non-occurrence of events. Such rule is called a determinacy or determination, and the mathematical theory of such rules is called determinacy analysis [9].

If it is observable that an occurrence of X is always followed by an occurrence of Y, this means that there exists a rule "If X then Y", or $X \rightarrow Y$. Such correlation between X and Y is called a *determination* (from X to Y). Here X is a *determinative (determining)* and Y is a *determinable*.

The determinative (X) consists of one or more factors. A factor is an attribute with its certain value. Each attribute can give as many factors as many different values it has. The factors coming from the same attribute are not contained in the same X.

Each rule has two characteristics: accuracy and completeness[1].

The accuracy of the determination $X \rightarrow Y$ shows to what extent X determines Y. It is defined as a proportion of occurrences of Y among the occurrences of X: $A(X \rightarrow Y) = n(X\ Y) / n(X)$, where $A(X \rightarrow Y)$ is the accuracy of determination, $n(X)$ is the number of objects having feature X and $n(X\ Y)$ is the number of objects having both features X and Y.

[1] In the beginning (in [1], for example) "accuracy" (Russian "*точность*") was called "intensity" and "completeness" ("*полнота*") was called "capacity" ("*емкость*").

The completeness of the determination $X \rightarrow Y$ shows which part of the objects having Y can be explained by the determination $X \rightarrow Y$. It is a percentage of occurrences of X among the occurrences of Y: $C(X \rightarrow Y) = n(X Y) / n(Y)$, where $C(X \rightarrow Y)$ is the completeness of determination, n(Y) is the number of objects having feature Y and n(X Y) is the number of objects having both features X and Y.

Both accuracy and completeness can have values from 0 to 1. Value 1 shows maximal accuracy or completeness, 0 means that rule is not accurate or complete at all. Value between 0 and 1 shows quasideterminism.

If all objects having feature X have also feature Y then the determination is (maximally) accurate. In case of an accurate determination $A(X \rightarrow Y) = 1$ (100%). Most of the rules are not accurate. In case of an inaccurate rule $A(X \rightarrow Y) < 1$. In order to make a determination more (or less) accurate the complementary factors are added into the left part of the rule. Adding the factor Z into the rule $X \rightarrow Y$ we get a rule $XZ \rightarrow Y$.

The contribution of factor Z *to accuracy* of rule $XZ \rightarrow Y$ is measured by the increase of accuracy $\Delta A(Z)$ caused by addition of factor Z into the rule $X \rightarrow Y$: $\Delta A(Z) = A(XZ \rightarrow Y) - A(X \rightarrow Y)$. The contribution to accuracy falls into interval from -1 to 1.

If $\Delta A(Z)>0$ then Z is a positive factor. Addition of a positive factor makes the rule more accurate, sometimes the resultant rule is (maximally) accurate. If $\Delta A(Z)<0$ then Z is a negative factor. Addition of a negative factor decreases the rule's accuracy, sometimes until zero. If $\Delta A(Z)=0$ then Z is a zero (or inessential) factor. Addition of a zero factor does not change the rule's accuracy. *An accurate rule* contains no negative factors, all factors are positive or zero factors [10].

If $C(X \rightarrow Y)=1$ (100%) then the rule $X \rightarrow Y$ is (maximally) complete. It means that Y is always explained by X. In case of an incomplete rule $C(X \rightarrow Y)<1$, it means that X does not explain all occurrences of Y.

The contribution of factor Z *to completeness* of rule $XZ \rightarrow Y$ is measured by the increase of completeness $\Delta C(Z)$ by addition of factor Z into the rule $X \rightarrow Y$: $\Delta C(Z) = C(XZ \rightarrow Y) - C(X \rightarrow Y)$. The contribution of whatever factor to completeness is negative or zero [10].

The system of rules is a set of rules in the form of $S_q = \{x_i \rightarrow y \mid i=1,2,...,q\}$, where q is the number of rules. Every system is characterised by average accuracy, summarised completeness and summarised capacity (the number of objects covered by the rules).

The system of rules $S_q = \{x_i \rightarrow y \mid i=1,2,...,q\}$ is *additive* when x_i-s pairwise do not overlap. The completeness and capacity of the additive system are just summed up completenesses and capacities of the rules. The accuracy of the additive system is not additive (i.e. equal to the sum of rules' accuracies), it is found as a weighted average.

A system is called *complete* if its completeness is 1. A *system* is called *accurate* if its accuracy is 1. A system is accurate when all of its rules are accurate.

The rank of a rule is a dimension of its left side. A rule in the form of $z_1z_2...z_r \rightarrow y$ is called a rule of rank r ($r \geq 1$). A system of rules in the form of $z_1z_2...z_r \rightarrow y$ is called a system of rules of rank r by variables $z_1, z_2, ..., z_r$ relative to the feature y. Every system of rules of rank $r \geq 1$ by fixed set of r variables is additive [10].

The theory of DA covers also the non-additive systems of (overlapping) rules, the first developments, an algorithm for finding such systems is published in [4].

1.2 Example

The following example illustrates the step by step approach. The data from [11] will be used in example (see Table 1). This data table (object-attribute system) contains 14 objects described by 5 attributes – the weather conditions of Saturday morning. The last attribute shows the object's belonging to a certain class (feature Y) - suitability for some unspecified activity (P – positive instances i.e. suitable conditions, N – negative/unsuitable). Attributes Outlook and Temperature have both three alternative values, attributes Humidity, Windy (and Class) have two possible values each.

We will describe objects belonging to class "N". This class consists of five objects (n(Y)=5). Attributes will be added into the rules in the following (freely chosen) order: Outlook, then Humidity, then Windy and last Temperature.

Table 1. Saturday morning data

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
Overcast	cool	normal	true	P
Sunny	mild	high	false	N
Sunny	cool	normal	false	P
Rain	mild	normal	false	P
Sunny	mild	normal	true	P
Overcast	mild	high	true	P
Overcast	hot	normal	false	P
Rain	mild	high	true	N

Rules containing attribute Outlook only are given in Table 2.

Table 2. The rules consisting of attribute Outlook

Outlook	n(X)	n(XY)	A	C	∑C
Sunny	5	3	3/5	3/5	
overcast	4	0	0	0	
Rain	5	2	2/5	2/5	

The rule with Outlook.overcast has zero accuracy (i.e. does not exist) in given class and will not be expanded. Two other rules have accuracy between 0 and 1, thus we expand them by adding the next attribute (Humidity) into rules (see Table 3).

Table 3. The rules consisting of attributes Outlook and Humidity

Outlook	Humidity	n(X)	n(XY)	A	C	∑C

Sunny	**High**	**3**	**3**	**1**	**3/5**	**3/5**
Sunny	Normal	2	0	0	0	
Rain	High	2	1	1/2	1/5	
Rain	Normal	3	1	1/3	1/5	

The rule Outlook.sunny&Humidity.high→Class.N is accurate (A=1) and needs no additional factors, it is included into the result. It covers 60% of the objects of the class (C=3/5). The completenesses C of the accurate rules are summed up (ΣC), with hope to reach the 100% coverage (by accurate rules). The rule with Outlook.sunny and Humidity.normal has zero accuracy in given class and will not be expanded.

Two last rules are expanded by adding attribute Windy (see Table 4).

Table 4. The rules consisting of attributes Outlook, Humidity and Windy

Outlook	Humidity	Windy	n(X)	n(XY)	A	C	ΣC
Rain	**high**	**true**	**1**	**1**	**1**	**1/5**	**4/5**
Rain	high	false	1	0	0	0	
Rain	**normal**	**true**	**1**	**1**	**1**	**1/5**	**1**
Rain	normal	false	2	0	0	0	

From here we find two accurate rules, both with completeness 20%. Now the sum of (accurate) completenesses is 100% which means that the class "N" is (fully) covered by the (accurate) rules. At the same time we can see that two other rules (with Windy.false) have zero accuracies and there is no reason to expand them.

As the found rules cover the class "N" completely there is no need to use the next attribute (Temperature), so the class is described without using it.

Class "N" is covered by three rules (denote this system of rules by *S1*):

1. Outlook.sunny & Humidity.high → Class.N (C = 60%);
2. Outlook.rain & Humidity.high & Windy.true → Class.N (C = 20%);
3. Outlook.rain & Humidity. normal & Windy.true → Class.N (C = 20%).

This is one possible description for class "N" beginning from the attribute Outlook. In the "language" of DA it means that these are saturdays with sunny outlook and high humidity or rainy outlook, high humidity and windy weather or rainy outlook, normal humidity and windy weather. The user has to know what to do with this knowledge. If "N" means "not suitable for certain activity", for example, then if the user is satisfied with such description then (s)he accepts it. If this knowledge is not satisfying then the user can change the order of attributes or add some new attributes.

For example, when the used order of attributes is Temperature, Outlook, Windy and Humidity, then we get four rules (in which attribute Humidity is not used):

1. Temperature.hot & Outlook.sunny → Class.N (C=40%);
2. Temperature.mild & Outlook.sunny & Windy.false → Class.N (C=20%);
3. Temperature.mild & Outlook.rain & Windy.true → Class.N (C=20%);
4. Temperature.cool & Outlook.rain & Windy.true → Class.N (C=20%).

1.3 Problems

In order to describe the problems we first present some results (rule systems) using all approaches of DA on the data from [11] (see Table 1). The purpose is to determine Class "N" by rules containing attributes Outlook, Humidity and Windy. Analysis is given from two aspects: interpretation of results and methods' procedural logic. Analyst is also interested in finding of several shorter rule systems to describe Y.

- The additive system consisting of accurate rules of different rank (number of factors) got using step by step approach can be *S1* consisting of 3 rules (see section 2.2) or (*S2*):

1. Outlook.rain & Windy.true → Class.N (C = 40%);
2. Outlook.sunny & Windy.true & Humidity.high → Class.N (C = 20%);
3. Outlook.sunny & Windy.false & Humidity.high → Class.N (C = 40%).

S1 is found when attributes are included in the order: first Outlook, then Humidity, then Windy. In case of *S2* the order is: first Outlook, then Windy and then Humidity. It is interesting to mention that the first rule of *S1* corresponds to two longer rules of *S2* containing values of attribute Windy as zero factors. Also, the first rule of *S2* is divided into two longer rules of S1 containing values of attribute Humidity as zero factors. In both cases the system is accurate and complete and the rules do not cover the same objects. As we can see the different rule systems describe the objects practically differently giving a different knowledge about the subset of objects. The user decides how many rule systems is enough. If the description given by the rule systems is not good enough then some other set of attributes can be chosen or just some attributes can be exchanged. As we can see, the more the rule system contains the rules, that are not contained in other rules, the shorter the rule system is.

- The additive system with a fixed set of factors (rank r=3) got using DAS (*S3*):

1. Outlook.sunny & Humidity.high & Windy.true → Class.N (C = 20%);
2. Outlook.sunny & Humidity.high & Windy.false → Class.N (C = 40%);
3. Outlook.rain & Humidity.high & Windy.true → Class.N (C = 20%);
4. Outlook.rain & Humidity. normal & Windy.true → Class.N (C = 20%).

Compared to the result of step-by-step approach the first rule of *S1* is divided into two and these two contain values of attribute Windy as zero factors and similarily the first rule fo *S2* is divided into two rules containing zero factors (values of Humidity).

- The non-additive system with overlapping rules got using algorithm described in [4] consists of 2 rules (*S4*):

1. Outlook.sunny & Humidity.high → Class.N (C = 60%);
2. Outlook.rain & Windy.true → Class.N (C=40%).

Although this algorithm can find overlapping rules, in this case the rules do not overlap, each instance of class "N" is covered exactly once; there are no zero factors. As we see it is the shortest rule system.

Looking at these examples we can see that an additive system containing rules with different lengths (both *S1* and *S2*) has lesser number of rules than an additive system with a fixed rank and set of attributes (*S3*). *S3* contains two longer rules from *S1* and two longer rules from *S2*. The set of overlapping rules (*S4*) has the least number of rules and also the shortest rules, it contains the shortest rule from *S1* and the shortest rule from *S2*. Also we can say that shorter rule systems consist of rules which are not contained in other rules.

The task to find an additive system of rules can be understood in several ways:

1. In case of DAS-approach, all the rules contain all given attributes.
2. In case of step by step approach, it is possible to find a system of rules of different length. Different order of inclusion of attributes can give different systems of rules.

In the first case, on the assumption of the essence of the DAS approach, there is exactly one solution. If each rule has to contain all (given) attributes, then we cannot take away any factors from any rules and consequently we cannot change such system of rules. We loose also a great amount of knowledge because of not knowing specific information about attributes' associations (by 2, 3 or more attributes). Determining a different set of attributes (for description) we get a different system which is also the only possible solution for that set of attributes (in case of the same requirement – to contain all attributes). In this case the purpose is to find the most suitable set of attributes i.e. exclude redundant attributes (like Temperature in our example) and include all necessary attributes to avoid contradictions – a situation where the set of attributes is not sufficient to distinguish different classes. A suitable set can be found by testing different sets of attributes.

In the second case – non-overlapping rules (i.e. additive system) of different length (rank) – the constitution of the system depends on the order in which the factors are included into the rules. It might be hard to find a compact system of such rules. In this case there is a huge amount of different orders of attributes in a given set of attributes by 1, 2, 3, ..., M attributes. It is not real to extract all these rule systems and to analyze them. It would be good to allow non-fixed order of including factors.

The recent approach to DA emanates from logic: if one rule system does not satisfy then the next one will be considered etc. Such treatment comes from the fact that the rules of one rule system are derived from the uniform order of attributes and thus they are mutually related. It means that we cannot change the order of attributes in the emergent rule system. This is an important restriction, according to it the best description in a sense can form from single rules of different rule systems. One solution here is to generate all rule systems and according to some criteria find from them the best cover i.e. a rule set (with completeness 100%) consisting of the shortest rules or a rule set with the least number of rules. This turns out to be very labor-consuming because all the possible sets and orders (permutations) of attributes should be found.

DA development for overlapping rules can automatically generate one rule system, but there is no guarantee that the extracted rule system is the shortest (with a lowest number of rules) (or several rule systems when changing criteria for choosing the factor for making extract).

Altogether we can say that 1) shorter rule systems mostly consist of rules which are not contained in other rules and 2) the existing approaches to DA do not include an effective method for finding shorter rule systems.

2 New Approach

In the previous section we mentioned that shorter rule systems consist of rules which are not contained in other rules. It means that for constructing shorter rule systems we need only rules of such type. But how to find them?

Next we present a new approach of DA which gives a solution to this problem. At first we define a new concept "Determinative set of rules", then describe an algorithm that can find it and describe how we can use this rule set for further analysis.

2.1 Basis of the New Approach

Let a table X(N,M) be given and a set B of all possible rules describing (only) the class Y and each rule in B is presented only once.

The Determinative set of rules (DSR) for class Y consists of all rules which are not contained in other rules of B.

B = {Ri}, i=1, 2,..., K, where K is a number of all possible rules describing (only) the Class Y. Ri $\neq$ Rj, i $\neq$ j.
DSR = {Ru}. Ru $\in$ DSR if there /$\exists$ Ri $\in$ B, Ru $\subset$ Ri, i $\neq$ u. DSR $\subseteq$ B

It means that DSR does not contain subrules of its rules. To get DSR from B we have to throw out all subrules of rules. We call this process „rule set compression".

Example. Let B contain 4 rules (all possible rules for Y = (Class =1)):

1. r1: IF T1=1 & T2=1 THEN CLASS=1
2. r2: IF T1=1 & T3=2 THEN CLASS=1
3. r3: IF T2=1 THEN CLASS=1
4. r4: IF T3=2 THEN CLASS=1

As we see, the rule r1 is contained in r3 and r2 is contained in r4. According to the definition DSR_B = {r3, r4}.

The main features of DSR are:

1. there are no redundant attributes (zero factors) in rules,
2. the same object in the class Y can be described by several rules.

On the basis of DSR we can form and solve next tasks, for example, to find

1. the shortest rules (by the number of attributes in the rule),

2. the longest rules (by the number of attributes in the rule in DSR),
3. the rules with specific features (for example, all rules of the rank r in DSR),
4. the shortest rule system (i.e. the rule system with the smallest number of rules),
5. the rule system which consists of rules with minimal ranks,
6. all the rule systems we can form on the basis of DSR.

The tasks 1-3 are easily solvable, but tasks 4-6 are essentially system covering tasks and they are NP-complex tasks.

2.2 Description of the Algorithm

Here we describe the algorithm realizing the new approach to DA. The findable set of rules is DSR together with some redundant rules which are eliminated afterwards.

This is a depth-first-search algorithm that makes subsequent extracts of objects containing certain factors. At each level first of all the rules (of that extract) are detected and then factors for making extracts of the next level are selected one by one.

The algorithm uses frequency tables for X_t (all objects of current extract) and Y_t (objects belonging to observable class of current extract), Fx_t and Fy_t accordingly. If there are equal frequencies in both frequency tables for some factor then this factor completes a rule. The rule includes also the factors chosen on the way to that extract.

The selection criteria for choosing the next factor are based on frequencies, the maximal frequency in Fy_t. In case of equal maximal frequencies in Fy_t the one having lower frequency in Fx_t is preferred. If only one attribute (of the extract) has free (unused) value(s) (indicated by frequencies over zero in Fy) then it is not practical to make a next (further) extract because there would be no free factors to distinguish objects of different classes in that extract. If there are no free factors (i.e. no frequencies over zero) then obviously it is not possible to make a next extract. In both cases the algorithm backtracks to the previous level.

Each factor that is used for making an extract or completing a rule is set to zero in the corresponding Fy. Each Fy (except for the initial level) inherits all zeroes of the previous level (we call it "bringing zeroes down"). These zeroing techniques prevent many redundant extracts and rules.

```
Algorithm
Determine tables X and Y
S0. t:=0; $U_t$:=∅
S1. Find frequencies in tables $X_t$ and $Y_t$: $Fx_t$, $Fy_t$
    If t>0 then
        For each factor A such that $Fy_{t-1}$(A)= 0
            $Fy_t$(A):= 0
S2. For each factor A such that $Fy_t$(A)=$Fx_t$(A)
        output rule {$U_i$}&A, i=0,…,t; $Fy_t$(A):= 0
S3. If not enough free factors for making extract then
        If t=0 then Goto End
        Else t:=t-1; Goto S3
S4. Choose a new (free) factor $U_t$
```

```
    Fy_t(A):= 0;
    t:=t+1; extract subtable of objects containing U_t;
    Goto S1
End. System of rules is found
```

2.3 Example

In the examples we use the same Saturday morning's data [11] as in section 1.2 (see Table 1), but this time in a numerical representation. The coding used for attributes and their values is shown in Table 5. The initial data table is given in Table 6. Let X is X(14,4), X_{ij} = 1,...,3 and Y=5.2 {Y_i: 1,2,6,8,14} (i.e. class "N"). The frequencies of attributes' values for X and Y (Fx and Fy accordingly) are given in Table 7.

Table 5. The coding used for Saturday morning data

Attribute	**Outlook**	**Temperature**	**Humidity**	**Windy**	**Class**
Code	1	2	3	4	5
1	sunny	cool	high	true	P
2	overcast	mild	normal	false	N
3	rain	hot			

Table 6. The initial data table

	1	2	3	4	5
1	1	3	1	2	2
2	1	3	1	1	2
3	2	3	1	2	1
4	3	2	1	2	1
5	3	1	2	2	1
6	3	1	2	1	2
7	2	1	2	1	1
8	1	2	1	2	2
9	1	1	2	2	1
10	3	2	2	2	1
11	1	2	2	1	1
12	2	2	1	1	1
13	2	3	2	2	1
14	3	2	1	1	2

Table 7. The frequencies for X and Y

Fx	1	2	3	4	Fy	1	2	3	4
1	5	4	7	6	1	3	1	4	3
2	4	6	7	8	2	0	2	1	2
3	5	4	0	0	3	2	2	0	0

There are no equal frequencies (over zero) in initial frequency tables. A factor that will be the basis for making an extract is chosen by the biggest value in Fy. This is 3.1 (attribute 3 with value 1) with frequency =4. Extract (subtable of the table X) by 3.1 is shown in Table 8 and the corresponding frequency tables in Table 9.

Table 8. Extract by 3.1 (Humidity.high)

	1	2	3	4	5
1	1	3	1	2	2
2	1	3	1	1	2
3	2	3	1	2	1
4	3	2	1	2	1
8	1	2	1	2	2
12	2	2	1	1	1
14	3	2	1	1	2

Table 9. The frequencies of extracted data

Fx	1	2	3	4	Fy	1	2	3	4
1	3	0	7	3	1	3	0	4	2
2	2	4	0	4	2	0	2	0	2
3	2	3	0	0	3	1	2	0	0

For 1.1 (Outlook.sunny) the frequencies in Fx and Fy are equal which means that all objects (of extract by 3.1) having 1.1 belong to Y. Hence we get a rule R1: 3.1&1.1=3 (Humidity.high&Outlook.sunny). The frequency after "=" shows that the rule covers three objects (namely 1, 2 and 8) – this is additional information. The frequency of 1.1 is set to zero in current Fy to avoid using it as a basis for making next extract(s).

Now we have to choose a factor for making a next extract. In Fy four different factors have the maximal frequency (=2): 2.2; 2.3; 4.1; 4.2. In Fx two of them (2.3 and 4.1) have frequency 3 and two have frequency 4 (2.2; 4.2). Choice is made from those which have lower frequency in Fx. The first one with frequency 3 (in Fx) is 2.3. Extract by (3.1 and) 2.3 is in Table 10 and corresponding frequencies in Table 11.

Table 10. Extract by 3.1 (Humidity.high) and 2.3 (Temperature.hot)

	1	2	3	4	5
1	1	3	1	2	2
2	1	3	1	1	2
3	2	3	1	2	1

Table 11. The frequencies of extracted data

Fx	1	2	3	4	Fy	1	2	3	4
1	2	0	3	1	1	0	0	2	1
2	1	0	0	2	2	0	0	0	1
3	0	3	0	0	3	0	2	0	0

Every frequency table for Y (Fy) inherits all zeroes of the previous level, to avoid repetitious extracts and redundant rules ("bringing zeroes down"). Therefore the actual frequency of 1.1 in Fy (=2) is replaced by 0. If this frequency was not set to zero we would find the rule 3.1&2.3&1.1=2 which is a subrule of the already found rule 3.1&1.1. From the current extract we find the rule R2: 3.1&2.3&4.1=1 (Humidity. High & Temperature.hot&Windy.true). The frequency of 4.1 is set to zero after that. Now there is only one frequency over zero in Fy (4.2=1). Making an extract by only free attribute (factor) cannot give a rule because there are no attributes to use for distinguishing between different classes at the next level. So the algorithm goes back to the previous level (extract by 3.1 – see Table 8).

The frequency table Fy of that level (see Table 9) has got two zeroes meanwhile: 1.1 has been set to zero when the rule with it was extracted and 2.3 has been set to zero due to being the basis for extract. The next (unused) factor with frequency 2 in Fy and frequency 3 in Fx is 4.1. The next extract is made by 4.1. This extract contains objects 2, 12 and 14. The frequencies are shown in Table 12.

Table 12. The frequencies of extract by 3.1 (Humidity.high) and 4.1 (Windy.true)

Fx	1	2	3	4	Fy	1	2	3	4
1	1	0	3	3	1	0	0	2	2
2	1	2	0	0	2	0	1	0	0
3	1	1	0	0	3	1	0	0	0

From that level we get the rule R3: 3.1&4.1&1.3=1 (Humidity.high&Windy.true& Outlook.rain). After zeroing the frequency of 1.3 in Fy there is only one unused frequency over zero in Fy (2.2=1) and algorithm backtracks to the previous level. The frequency table Fy of extract by 3.1 (Table 8) has now one more zero (4.1), its current state is given in Table 13.

Table 13. Frequnecy tables for extract by 3.1

Fx	1	2	3	4	Fy	1	2	3	4
1	3	0	7	3	1	0	0	4	0
2	2	4	0	4	2	0	2	0	2
3	2	3	0	0	3	1	0	0	0

Factors 2.2 and 4.2 have equal frequencies in both tables (2 in Fy and 4 in Fx). First 2.2 is chosen to make the next extract. This extract (containing objects 4, 8, 12 and 14), its sub-extract by 1.3 (objects 4 and 14) and also extract by 4.2 (objects 1, 3, 4, 8) give no rules.

Algorithm is back at the initial level. The only thing that is changed compared to the state shown in Table 7 is that the frequency of 3.1 is set to zero in the initial Fy.

The work continues the same way. Below the whole tree of extracts made during the work is given. Making extract by 3.1 and its sub-extracts have been described already (see tables 7-13).

After "=" are shown frequencies Fy/Fx of extract. Rn after "⇒" indicates a found rule. On the right the objects contained in extract are listed.

```
3.1 (Humidity.high)=4/7 ⇒ R1           1,2,3,4,8,12,14
  & 2.3 (Temperature.hot)=2/3 ⇒ R2     1,2,3
  & 4.1 (Windy.true)=2/3 ⇒ R3          2,12,14
  & 2.2 (Temperature.mild)=2/4         4,8,12,14
    & 1.3 (Outlook.rain)=1/2           4,14
  & 4.2 (Windy.false)=2/4              1,3,4,8
1.1 (Outlook.sunny)=3/5 ⇒ R4           1,2,8,9,11
  & 4.2 (Windy.false)=2/3 ⇒ R5         1,8,9
  & 2.2 (Temperature.mild)=1/2         8,11
4.1 (Windy.true)=3/6 ⇒ R6, R7          2,6,7,11,12,14
  & 2.1 (Temperature.cool)=1/2         6,7
  & 2.2 (Temperature.mild)=1/3         11,12,14
2.3 (Temperature.hot)=2/4              1,2,3,13
1.3 (Outlook.rain)=2/5                 4,5,6,10,14
  & 2.1 (Temperature.cool)=1/2         5,6
  & 2.2 (Temperature.mild)=1/3         4,10,14
2.2 (Temperature.mild)=2/6             4,8,10,11,12,14
4.2 (Windy.false)=2/8                  1,2,3,4,5,8,9,10,13
2.1 (Temperature.cool)=1/4             5,6,7,9
```

As a result 7 rules have been found:

R1: 3.1&1.1=3	(Humidity.high&Outlook.sunny→Class.N)
R2: 3.1&2.3&4.1=1	(Humidity.high&Temperature.hot&Windy.true→Class.N)
R3: 3.1&4.1&1.3=1	(Humidity.high&Windy.true&Outlook.rain→Class.N)
R4: 1.1&2.3=2	(Outlook.sunny&Temperature.hot→Class.N)
R5: 1.1&4.2&2.2=1	(Outlook.sunny &Windy.false&Temperature.mild→Class.N)
R6: 4.1&1.3=2	(Windy.true&Outlook.rain→Class.N)
R7: 4.1&2.3=1	(Windy.true&Temperature.hot→Class.N)

Two of them are redundant: R2 is a subrule of R7 (3.1 (Humidity.high) is a zero factor here), and R3 is a subrule of R6 (due to redundant factor 3.1 (Humidity.high) R3 covers one object less than R6). Subrule of some rule is always found before the rule itself. Subrules that could be found afterwards are prevented by zeroing techniques.

After the rule set compression we got DSR: R1, R4, R5, R6 and R7. On the basis of these five rules (DSR) we can solve several tasks mentioned in section 2.1.

3 Conclusion

This paper gives an overview of Determinacy Analysis, introducing the theory and different approaches, proposes a new method and the corresponding algorithm for finding determinative set of overlapping rules and several ideas how to use it. The algorithm is based on frequency tables which makes it easy to detect a potential zero factor free DSR rule. Finally we can say the following.

1. Somebody might say that the finding of DSR is very laborious, especially in cases of large amounts of data. If so, user can decide, what is the purpose of the work. If the purpose is quick one-time information gathering for a data set under analysis, then the use of DSR-based DA approach maybe not the best one. But if the purpose is to describe the data set and through that discover a new knowledge, then this DA approach is a good solution, because it enables the post analysis of DSR to discover more new knowledge. There is no need to experiment with several attributes and their different orders to get more knowledge.
2. It is mostly desired that the rules are relatively short and/or the number of rules has not to be very large, because of easier interpretation of them. On the other hand, if we want to describe the data, we must know how several attributes act together, what are the typical co-occurrences of attributes, what attributes do not act with others etc. The earlier methods of DA described in the paper can't discover such kind of knowledge, but the new one presented here creates a good basis for that.

The post-analysis of rules will be the topic of the next paper.

References

1. Chesnokov, S.V.: Determination-Analysis of Social-Economic Data in Dialogical Regime. Preprint. All-Union Institute for Systems Research, Moscow (1980) (in Russian)
2. Chesnokov, S.V.: Determinacy Analysis of Social-Economic Data. Nauka, Moscow (1982) (in Russian)
3. Lind, G., Kuusik, R.: Some Ideas for Determinacy Analysis Realisation. In: Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing, pp. 185–190. ACTA Press (2007)
4. Kuusik, R., Lind, G.: Some Developments of Determinacy Analysis. In: Cao, L., Zhong, J., Feng, Y. (eds.) ADMA 2010, Part I, LNCS, vol. 6440, pp. 593–602. Springer-Verlag Berlin Heidelberg (2010)
5. Chesnokov, S.V.: Determinacy Analysis of Social-Economic Data. Sociological Studies 3, pp. 179–189 (1980) (in Russian)
6. Luelsdorff, P.A., Chesnokov, S.V.: Determinacy Form as the Essence of Language. Prague Linguistic Circle Papers 2, pp. 205–234 (1996)
7. Chesnokov, S.V.: Determinacy Analysis and the Search for Diagnostic Criteria in Medicine (the Case of Comprehensive Ultrasonography). Ultrasonic Diagnostics 4, pp. 42–47 (1996) (in Russian)
8. Main scientific works of S.V. Chesnokov (in Russian), http://www.context.ru/publications
9. DA-system 4.0, version 4.0 for Windows 95, Windows 98 and Windows NT. Questions and Answers. DA-system and Technology of Data Analysis. "Context" (1999) (in Russian)
10. Chesnokov, S.V.: Determinacy Analysis of Socio-Economic Data. Illustrative Materials to Lectures. Lecture 2: Rules. Lecture 3: Systems of Rules. Lomonosov Moscow State University, Faculty of Economics, Moscow (2002) (unpublished, in Russian)
11. Quinlan, J.R.: Induction of Decision Trees. Machine Learning, 1, pp. 81–106 (1986)

Appendix F

Lind, G. and Kuusik, R. (2016). Algorithm for Finding Zero Factor Free Rules. *Man-Machine Interactions 4: 4th International Conference on Man-Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6-9, 2015. AISC 391*, pp. 421-435. Springer.

Algorithm for Finding Zero Factor Free Rules

Grete Lind, and Rein Kuusik

Tallinn University of Technology, Department of Informatics, Tallinn, Estonia
{Grete.Lind, Rein.Kuusik1}@ttu.ee

Abstract. Class detection rules are mostly used for classifying new objects. Another possible usage is to describe a set of objects (a class) by the rules. Determinacy Analysis (DA) is a knowledge mining method with such purpose. Sets of rules are used to answer the questions "Who are they (objects of the class)?", "How can we describe them?". Rules found by different DA methods tend to contain some redundant information called zero factors. In this paper we show how zero factors are related to closed sets and minimal generators. We propose a new algorithm that extracts zero-factor-free rules and zero factors themselves, based on finding generators. Knowing zero factors gives to the analyst important additional knowledge for understanding the essence of the described set of objects (a class).

Keywords: Determinacy Analysis · Rule · Zero Factor · Minimal Generator

1 Introduction

The problem that will be solved in this paper has arisen from the method called Determinacy Analysis (DA) which belongs to machine learning field.

There are two directions (subtasks) in machine learning. Direction 1 – the main task is a *Classification task*: to find rules for classification of unknown object(s) on the basis of learning examples. Direction 2 – *Data Analysis and Data Mining task*: to use the found rules for describing the class under analysis answering the questions: "Who are they (objects of the class)?", "How can we describe them?", "What distinguishes them from others?".

The best representative of direction 2 is a method called Determinacy Analysis [1, 2] that presents an original methodology for answering these questions. DA outputs the accurate and complete rule system in the form of rules "IF X THEN Class" so that each object is covered by one rule at most. All extracted rules cover only the set of objects under analysis. Now the analyst can describe the set of objects under analysis and also determine what is specific for the class and what separates different classes. If the description is not good enough he/she can change the order of attributes or add new ones.

Problem. Extracted rules consist of several factors (certain attributes with certain values) and the question is which of them are the most important and which are superfluous, giving no additional knowledge. For example, if a rule contains two factors: (an attribute) "Are you living in the countryside?" with a value "Yes", and (another attribute) "Do you have cows?" with a value "Yes", then the first attribute makes no sense, it is redundant because having cows means that one lives in the country. Consequently, it means that the first factor is an inessential factor (a zero factor) because it does not include any new knowledge for the researcher. The question is how we can avoid such (zero) factors.

DA has an approach how to ascertain zero factors, but this is hindsight. The factors are added into the rules one by one, but we cannot say at the moment of addition whether a certain factor remains essential after adding the next one(s). Therefore the analyst has to measure the influence of every factor regarding all other factors in the final rule. Different orders of factors tend to give different results consisting of different sets of rules (the approach is presented in Section 2.2). It is not realistic to measure the results of all different orders of attributes and it means that this approach is unusable. The task is to elaborate a simple way to avoid zero factors for extracting zero factor free rules.

Next we present a brief description of determinacy analysis in order to understand its essence and then explain the idea for extracting zero factor free rules.

2 Description of Determination Analysis

2.1 Definitions

Definitions are given according to [1, 3].

The idea behind DA is that a rule can be found based on the frequencies of joint occurrence or non-occurrence of events. Such a rule is called determinacy or determination and the mathematical theory of such rules is called determinacy analysis [2].

If it is observable that an occurrence of X is always followed by an occurrence of Y, it means that there exists a rule "If X then Y", or X→Y. Such correlation between X and Y is called *determination* (from X to Y). Here X is the *determinative (determining)* and Y is the *determinable.*

The determinative (X) consists of one or more factors. A factor is an attribute with its certain value. Each attribute can have many different discrete values and gives as many different factors as many different values it has. Factors coming from the same attribute are not contained in the same X.

Each rule has two characteristics: accuracy and completeness.

Accuracy of determination X→Y shows to what extent X determines Y. It is defined as the proportion of occurrences of Y among the occurrences of X:

$$A(X \to Y) = n(XY)/n(X) \tag{1}$$

where

A(X→Y) is the accuracy of determination,

n(X) is the number of objects having feature X and n(X Y) is the number of objects having both features X and Y.

Completeness of determination X→Y shows which part of cases having feature Y can be explained by determination X→Y. It is the percentage of occurrences of X among the occurrences of Y:

$$C(X \to Y) = n(XY)/n(Y) \tag{2}$$

where

C(X→Y) is the completeness of determination,

n(Y) is the number of objects having feature Y and

n(X Y) is the number of objects having both features X and Y.

Both accuracy and completeness can have values ranging from 0 to 1. A value of 1 shows maximum accuracy or completeness, 0 means that the rule is not accurate or complete at all. A value between 0 and 1 shows quasideterminism.

If all objects having feature X also have feature Y then the determination is (maximally) accurate. In case of accurate determination A(X→Y) = 1 (100%).

The majority of rules are not accurate. In case of inaccurate rule A(X→Y) < 1.

In order to make a determination more (or less) accurate, complementary factors are added to the left part of the rule. Adding factor Z into the rule X→Y, we get the rule XZ→Y, adding factor W to the rule XZ→Y, we get the rule XZW→Y etc.

The contribution of factor Z to the accuracy of the rule XZ→Y is measured by the increase of accuracy *Δ*A(Z) caused by addition of factor Z into the rule X→Y:

$$\Delta A(Z) = A(XZ \to Y) - A(X \to Y)\ . \tag{3}$$

The contribution to accuracy can range from -1 to 1.

If *Δ*A(Z)>0 then Z is a *positive factor*. Adding a positive factor makes the rule more accurate, sometimes the resultant rule is (maximally) accurate. If *Δ*A(Z)<0 then Z is a *negative factor*. Adding a negative factor decreases the rules accuracy, some-times down to zero. If *Δ*A(Z)=0 then Z is a *zero (or inessential) factor*. Adding a zero factor does not change the rules accuracy. An *accurate rule* contains no negative factors, all factors are positive or zero factors. A rule consisting of positive factors only, is called a *normal rule*.

If C(X→Y)=1 (100%) then the rule X→Y is (maximally) complete. It means that Y is always explained by X. In case of an incomplete rule C(X→Y)<1, X does not explain all occurrences of Y.

The *contribution* of factor Z *to the completeness* of the rule XZ→Y is measured by the increase of completeness *Δ*C(Z) by addition of factor Z into the rule X→Y:

$$\Delta C(Z) = C(XZ \to Y) - C(X \to Y)\ . \tag{4}$$

The contribution of whatever factor to completeness is negative or zero.

A set of rules is called a *system of rules* and characterized by average accuracy, summarized completeness and summarized capacity (the number of objects/cases covered by rules). A *system* is called *complete* if its completeness is

1. A *system* is called *accurate* if its accuracy is 1. A system is accurate when all of its rules are accurate.

2.2 Example of DA

DA enables to find different sets of rules, depending on the order of inclusion of the attributes into the analysis. Attributes (factors) are added into (the left sides of) the rules (X) one by one in a given order. If a rule is accurate it will not be expanded by adding the next factor. At the same time non-accurate rules will acquire next factors until they become accurate (or there is no more attributes to add). This way a set of non-overlapping rules of different length (called rank) is obtained. If there are no contradictions in the data, then the result covers all objects of the observable class.

Next we give an example of two different sets of rules obtained with different orders of attributes. We use the well-known Quinlan's data set (of eight people characterized by height, hair color and eye color [4] – see Table 1) and describe (the people belonging to) the class "-" by accurate rules.

Table 1. Quinlan's table

Height	Hair	Eyes	Class
tall	dark	blue	+
short	dark	blue	+
tall	blond	blue	-
tall	red	blue	-
tall	blond	brown	+
short	blond	blue	-
short	blond	brown	+
tall	dark	brown	+

If the order is: 1) Hair, 2) Eyes, 3) Height; then we get:

– A1: Hair.red → Class. (C=1/3);
– A2: Hair.blond & Eyes.blue → Class. (C=2/3).

For the order 1) Height, 2) Hair, 3) Eyes; the set of rules is as follows:

– B1: Height.tall&Hair.red → Class. (C=1/3);
– B2: Height.short&Hair.blond&Eyes.blue → Class. (C=1/3);
– B3: Height.tall&Hair.blond&Eyes.blue → Class. (C=1/3).

An algorithm for the presented "step by step" approach is given in [5].

2.3 The Task of DA

As the author of DA Chesnokov states, the main task is to find maximally accurate and complete systems of rules [3]. If we want to get rid of redundant information (like "living in the countryside" in case of "having cows") then the left side of the rules has to contain only such factors that make a rule more accurate than it was without them, i.e. positive factors. As shown in [6] it is not easy to avoid them: "The fact that some factor has a positive impact on the accuracy at the moment it is added into the rule does not guarantee that the factor retains its positiveness" after the addition of the next one(s).

It means that in case of another order a certain factor may be left out of nearly the same rule because actually it is a zero (inessential) factor in the itemset (conjunction of factors) that defines a class.

If we analyze the presented accurate rule systems extracted by DA (see 2.2), we can see that the first system is free of zero factors: A1 consists of one factor only; in A2 both factors are necessary for detecting a class, neither factor alone gives an accurate rule. In the second rule system all 3 rules contain zero factors: Height.tall in B1; Height.short in B2; and Height.tall in B3. As we can see the attribute Height is not needed for determining class "-", but we do not know it in advance.

DA cannot automatically identify which factors in the rule are zero factors and there is no possibility to generate all rule systems based on the entered set of attributes – these are the main weaknesses of the method. We try to solve both problems.

Identification of zero factors would give a possibility to foreshorten rules. Identified zero factors that have to be left out from the left side of the rule can be moved to the right side – the conclusion part.

Instead of generating all possible different systems of rules our solution is to find all zero-factor-free rules that is a suitable basis for forming different (accurate and complete) systems of rules.

Next we will show that it is possible to recognize and avoid zero factors (from the left sides of the rules) and explain how to do it.

3 THEORETICAL FOUNDATIONS

In this section we introduce different types of zero factors, the concepts of generator and closed set and show their relations to DA rules.

3.1 Different Types of Zero Factors

We have found that there are two types of zero factors:

1. the ones with zero contribution to the completeness (ΔC=0) that do not change the rule's coverage (set of covered objects) and frequency (the number of objects it covers) and
2. the ones with negative contribution to the completeness (ΔC<0) that decrease the rule's frequency.

We will call them *zero-zero factors* and *zero-negative factors*, accordingly. Recall that zero factor means a factor with zero contribution to the accuracy (ΔA=0), so the accuracy of the rule does not change in either case.

"Living in the countryside" in case of "having cows" is an example of a zero-zero factor (if everyone who has cows lives in the country).

Also Height.tall in the rule B1 (with completeness 1/3) is a zero-zero factor, because the rule without it (A1) has the same completeness (both rules cover exactly the same objects). Height.short in B2 and Height.tall in B3 are zero-negative factors. If either of them is added into the rule A2 then the completeness of the rule decreases from 2/3 to 1/3. This negative difference (1/3-2/3) is the factor's contribution to the rule's completeness.

3.2 Closed Sets and Generators

In frequent itemset mining an item is a binary attribute that can be either present or not in a transaction (a database record). For example, in market basket databases the items represent purchased goods. Extending the concept to multi-valued attributes, an item is a certain attribute with a certain value from the set of different possible values for that attribute. For example, in case of a market basket database, instead of "bread" there can be either "black bread" or "white bread" (i.e. attribute "bread" with either value). Such an item corresponds to a DA factor.

A *closed (item)set* is the maximal set of items common to a set of objects [7], it has no superset with the same support (i.e. frequency) [8]. Adding whichever item decreases its coverage and frequency. For example, one of the closed sets in Table 1 is Hair.blond&Eyes.blue&Class.- with frequency 2. If we add Height.short or Height.tall to this itemset, then the frequency changes and the resultant itemset is not the same closed set anymore.

A *closure* is the smallest (minimal) closed itemset containing the given itemset [9] i.e. the itemset's maximal superset with the same frequency. For example, the closure of Hair.red (frequency=1) is Height.tall&Hair.red&Eyes.blue&Class.- (frequency=1). A closed set is the same as its closure.

A *(minimal) generator* of a closed set is an itemset with the same closure and with no proper subsets with the same closure [10]. Taking away whichever item increases its coverage (and frequency). For example, Hair.blond&Eyes.blue with frequency 2 is a generator of the closed set Hair.blond&Eyes.blue&Class.- with a frequency of 2. Taking away either Hair.blond or Eyes.blue from the generator gives us an itemset with bigger frequency and thus with different closure.

If the number of items in a closed set and its generator differs more than by one then also the sets between the minimal generator and the closed set can be used for generating a closed set and can be called generators (for example, itemsets between Hair.red and Height.tall&Hair.red&Eyes.blue&Class.-). However, mostly "generator" means the minimal generator.

A closed set can have more than one minimal generator. For example, the closed set Hair.blond&Eyes.blue&Class.- has two (minimal) generators: Hair. blond&Eyes.blue and Hair.blond&Class.-.

A closed set or a generator is said to be frequent if its frequency is more than or equal to a given threshold. If the frequency threshold is 2, then Height.tall&

Hair.red&Eyes.blue&Class.- and its generators are infrequent (frequency=1); Hair.blond&Eyes.blue&Class.- and its generators are frequent (frequency=2).

3.3 Relations

A closed set is the maximal set of items common to a set of objects and its (minimal) generator is a minimal set of items common to that set of objects. Between the closed set and its generator there are such items, the addition or removal of which does not change the coverage and frequency of the itemset. Those items are similar to zero-zero factors that do not change either the accuracy or the completeness of the DA rule. Just the class-belonging usually is not observed in case of closed sets. Consequently, in order to avoid zero-zero factors the left side of the rule has to be a minimal generator.

Minimal generators do not contain zero-zero factors, but they can contain zero-negative factors. For example, the generator Height.tall&Hair.blond&Eyes. blue determines Class.- (i.e. rule B3: Height.tall&Hair.blond&Eyes.blue→Class.-), but Height.tall is a zero-negative factor, because Hair.blond&Eyes.blue is enough to determine Class.- (rule A2: Hair.blond&Eyes.blue→Class.-). Height. tall decreases the rule's completeness by 1/3 (from 2/3 to 1/3). Thus, if a generator produces a rule (generator→class) then the rules with super-generators of that generator contain zero-negative factors and are redundant.

Therefore, for class detection we need such (minimal) generators that define a class and have no such subset that defines a class.

4 Zero Factor Free DA

From minimal generators that define a class, we can build rules IF minimal-generator THEN class (min-gen→class). In this case we get rules with zero-factor-free (ZFF) left sides and therefore we call our approach Zero Factor Free DA (ZFF DA).

Next we present an algorithm for producing zero-factor-free rules – a set of accurate normal rules. Additionally, it can output rules where zero-zero factors are on the right side (min-gen→zero-factors) – association rules. The algorithm is based on finding generators. For each generator it is possible to make sure whether it defines a class and also to detect the difference with its corresponding closed set (i.e. zero-zero factors). Finding of all needed generators is guaranteed. The majority of their unwanted supergenerators (containing zero factors) can be avoided, the remaining part is excluded by compression of the initial result (after the main algorithm).

4.1 Description of the Algorithm

This is a depth-first search algorithm that makes subsequent extracts of objects containing certain factors. From the root to the leaves (of search tree), the frequencies of extracts always decrease. Each extract is determined by a generator. Each generator is found only once.

The algorithm uses frequency tables that show for each attribute the frequencies of all its possible values (in the set of objects for which it is found).

Frequencies (in the frequency table) can be equal to or smaller than the current ("leading") frequency (the number of the objects in the current extract). Equal frequency shows that all objects of the extract contain that factor. For each attribute, there can be at most one frequency equal to the leading one, in such case all other frequencies for that attribute are zeroes. Factors with such frequency are zero-zero factors (in the current extract).

Detecting whether the generator determines a class is analogous. If for the class attribute one value has a frequency equal to the leading one (and others are zeroes), then all objects of the extract belong to that class.

In order to prevent finding supergenerators (subrules) of the current generator (rule) the algorithm backtracks after detecting a class. Only such supergenerators can be avoided that are not found yet.

If no class was detected (objects of the extract belong to different classes) then the next factor to be included into the generator (left side of the rule) is selected by the frequency (from the frequency table). Its frequency has to be smaller than the frequency of the current extract and bigger than or equal to the given frequency threshold. The first condition prevents the inclusion of zero-zero factors (of the current extract), the second one is usual in mining frequent sets and rules. In order to find minimal generators only (not the ones between a minimal generator and closed set), the minimal one of suitable frequencies is chosen. However this condition does not guarantee that the next generator is always minimal, but usually it is. If there is more than one factor with such frequency, just one of them is selected. The chosen factor together with the previously selected factors of the same branch forms a generator and determines a narrower (than the current) set of objects.

In order to avoid repeatedly finding already found generators, the frequency of the selected factor (the "leading" factor) is set to zero in the current frequency table. Before selecting the next leading factor, those zeroes are "brought down" from the frequency table of the previous level to the current level (except for the initial level).

The following notation is used in pseudocode of the algorithm:

`attr` – number of attributes (excluding class);

$\texttt{X}_0$ – initial data table (objects*(`attr`+class));

`t` – number of the step (or level) of the recursion;

$\texttt{X}_t$ – set of objects (extract) at level `t`;

$\texttt{FT}_t$ – frequency table for a set $\texttt{X}_t$;

$\texttt{gen}_t$ – generator at level `t`;

`zf` – zero factors (regarding $\texttt{gen}_t$);

`noclass` – the truth-value of whether the class is detected for $\texttt{gen}_t$;

`gclass` – class value of $\texttt{gen}_t$;

`V` – "leading" frequency i.e. frequency of extract;

`minfr` – frequency threshold (minimal allowed number of covered objects);

Factors are given as $\texttt{value}_{attribute}$;

Assignments are indicated by "$\leftarrow$" ("=" is for comparison).
The pseudocode of the algorithm is given below.

```
Algorithm
Given: $X_0$ , minfr>0
A1. t←0 ; $gen_0$ ←{}
A2. find $FT_0$
A3. FOR EACH factor $h_{f=1,,attr}$ ∈$FT_0$ with frequency V=min $FT_0[h_f]$≥minfr
DO
A4.  $FT_0[h_f]$←0
A5.  make_extract(t+1; $h_f$; V)
    NEXT
End of Algorithm
PROCEDURE make_extract(t; $h_f$; V)
B1. $gen_t$ ←$gen_{t-1}$∪ $h_f$
B2. zf←{} ; gclass←0 ; noclass←true
B3. separate submatrix $X_t$ ⊂$X_{t-1}$ such that $X_t$={Xij∈$X_{t-1}$| X.f=$h_f$}
B4. find $FT_t$
B5. FOR EACH empty position p (p∈1,,attr) in $gen_t$ DO
B6.  IF exists value h such that $FT_t[h_p]$= V THEN
B7.   zf←zf∪$h_p$
     ENDIF
    NEXT
B8. IF exists value clv such that $FT_t[clv_c l]$= V THEN
B9.  gclass←clv ; noclass←false
    ENDIF
B10. output $gen_t$, zf, gclass, V
B11. IF noclass AND V>minfr THEN
B12.  ZeroesDown(t)
B13.  FOR EACH $h_{u=1,,attr}$ ∈$FT_t$ with frequency V2=min $FT_t[h_u]$≥minfr and
V2<V DO
B14.   $FT_t[h_u]$←0
B15.   make_extract(t+1; $h_u$; V2)
      NEXT
     ENDIF
END PROCEDURE
PROCEDURE ZeroesDown(t)
C1. FOR EACH factor $h_{u=1,,attr}$ ∈$FT_t$ with frequency > 0 DO
C2.  IF $FT_{t-1}[h_u]$=0 THEN $FT_t[h_u]$←0
    NEXT
END PROCEDURE
```

The initial data table X_0 and the frequency threshold minfr are given. The main program starts with initial assignments for a level of recursion t and the empty generator gen_0 (step A1). Next the frequency table FT_0 for X_0 is found (A2). In step A3 each factor with a suitable frequency (≥minfr) is chosen as a leading factor (for inclusion into generator) in ascending order (by frequencies).

The frequency of the leading factor $\mathtt{h}_f$ is set to zero in the frequency table $\mathtt{FT}_0$ (A4) and an extract by $\mathtt{h}_f$ is made (A5).

While the main program makes extracts from initial data, the recursive procedure make_extract handles all deeper levels. It starts with evaluating the current generator $\mathtt{gen}_t$ (B1) and giving initial values for the set of zero factors `zf`, class value `gclass` of current generator and truth-value `noclass` for indicating whether the class is found (B2). Next the subset of objects $\mathtt{X}_t$ is extracted by the leading factor $\mathtt{h}_f$ (B3) and the corresponding frequency table $\mathtt{FT}_t$ is found (B4). Step B5 goes through all empty positions (attributes without value) in current generator $\mathtt{gen}_t$ (as a vector) and B6 searches for the value (of that attribute) with frequency equal to the leading one `V`. If one exists, it is a zero-zero factor (regarding $\mathtt{gen}_t$) and it is included into the set of zero factors `zf` (B7). In B8 a similar check is made for class attribute. If equal frequency is found, then the generator $\mathtt{gen}_t$ determines a class and in B9 its class value `gclass` and indicator `noclass` are evaluated accordingly. In step B10 the generator is outputted together with its frequency, possible zero factors and class. Several conditions may be applied to decide whether to output the current generator or not – this is a possibility to leave out generators without a class and/or without zero factors (or without new zero factors at that level). If `zf` is not empty, the rule `IF` $\mathtt{gen}_t$ `THEN zf` can be produced. If class was detected then the rule `IF` $\mathtt{gen}_t$ `THEN gclass` can be produced.

Step B11 checks the suitability of making a subsequent extract. If a class is not found (`noclass=true`), then there is hope to detect it by a longer generator(s). If the frequency `V` is above the threshold `minfr`, then there is a possibility to find frequency that is <`V` and $\geq$ `minfr`. If that check gives a positive result, then the zeroes from the frequency table of the previous level are "brought down" (B12). The procedure `ZeroesDown` goes through the current frequency table and for each factor with a frequency over zero (C1) its frequency at the previous level is checked (C2). If the latter is zero, then the factor gets a zero frequency at the current level as well (C2).

Step B13 goes through all factors that are suitable for subsequent extract i.e. with frequency smaller than the leading one (in order to prevent including zero-zero factors) and greater than or equal to the given frequency threshold `minfr`. Again the order is ascending. The frequency of selected next factor $\mathtt{h}_u$ is set to zero (B14) and recursive call to procedure `make_extract` is made with new leading factor $\mathtt{h}_u$ and its frequency `V2` (B15).

4.2 Example

In the following example we use data from [11], given in Table 2. The data set consists of 14 objects described by four attributes and class.

With the frequency threshold 2 the algorithm finds 39 generators. The result contains 11 generators with class (suitable for producing rules IF generator THEN class) and 6 generators with zero factor(s) (suitable for producing rules IF generator THEN zero-factors); 2 generators have both. 24 generators have

Table 2. Initial data table

obj	Ou(tlook)	Te(mperature)	Hu(midity)	Wi(ndy)	Cl(ass)
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

neither. Of course, it is possible not to output them. Also it is possible filter out (or not to output) generators without class or generators without zero factors.

After the compression 6 generators are left for Class "P", listed in Table 3.

Table 3. Minimal generators of class "P" and corresponding generator-based rules

	Minimal generator	Rule IF minimal generator THEN class Rule IF minimal generator THEN zero-factor(s)
G1	Ou.overcast	IF Outlook.overcast THEN Class.P
G5	Te.cool&Wi.false	IF Temperature.cool&Windy.false THEN Class.P
		IF Temperature.cool&Windy.false THEN Hu.normal
G13	Ou.sunny&Hu.normal	IF Outlook.sunny&Humidity.normal THEN Class.P
G24	Ou.rain&Wi.false	IF Outlook.rain&Windy.false THEN Class.P
G26	Te.mild&Hu.normal	IF Temperature.mild&Humidity.normal THEN Class.P
G38	Hu.normal&Wi.false	IF Humidity.normal&Windy.false THEN Class.P

For each listed (minimal) generator (in Table 3) we get a zero-factor-free class detection rule (IF minimal generator THEN class). For example, from G5 we can conclude: IF Temperature.cool&Windy.false THEN Class.P. Such a conclusion holds in the given data set (Table 2).

Generator G5 has a zero factor also, this gives an association rule (IF minimal generator THEN zero-zero factors): IF Temperature.cool&Windy.false THEN Humidity.normal. It means that Temperature.cool&Windy.false is always accompanied by Humidity.normal. From the viewpoint of the data analysis task it shows that Class is "P" in case of objects which contain factors Temperature.cool&Windy.false&Humidity.normal, their essence is determined by the factors Temperature.cool&Windy.false, but not by Humidity.normal. It is important additional information for the data analyst.

A generator together with its zero-zero factors forms a closed set – the set of all the common factors of the covered objects. No other object in the given data set contains all those factors. The two objects covered by the minimal generator Temperature.cool&Windy.false (G5) have 3 factors in common: Temperature.cool&Windy.false&Humidity.normal.

Thus from the minimal generator Temperature.cool&Windy.false (G5) we can conclude both Class.P and the presence of Humidity.normal. For the analysis task it is important to know which feature (Humidity.normal) is elicited by the other one(s).

4.3 Experiments

The method described in this paper is implemented by L. Jõgiste [12]. Experiments for showing dependency of execution time on number of objects (rows) or attributes (columns) were carried out using Nursery data set from UCI Machine Learning Repository [13]. The dataset contains 12000 rows and 12 columns, 2 of which were added for testing purposes. Attribute value ranges from 1 to 5. Besides execution time the number of extracts made during the work (”steps”) was measured.

Dependency on number of rows was tested with 2000, 4000, 6000, 8000, 10000 and 12000 rows. The number of columns was 12. The result is given in Fig. 1.

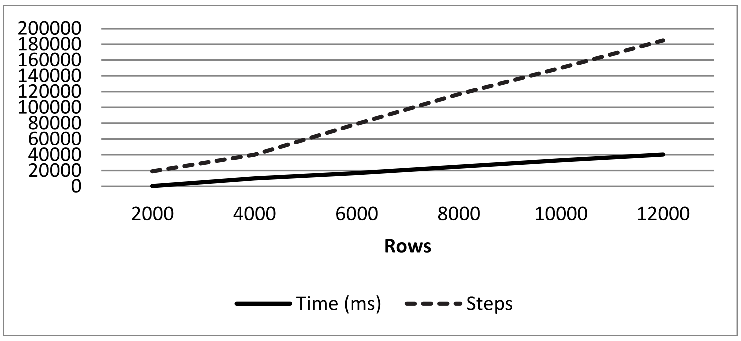

Fig. 1. Execution time dependency on number of rows [12]

Dependency on number of columns (Fig. 2) was tested with 2 to 12 columns with step 2.

As we can see, the number of columns has a strong impact on the execution time and number of extracts, while the number of rows influences them much less.

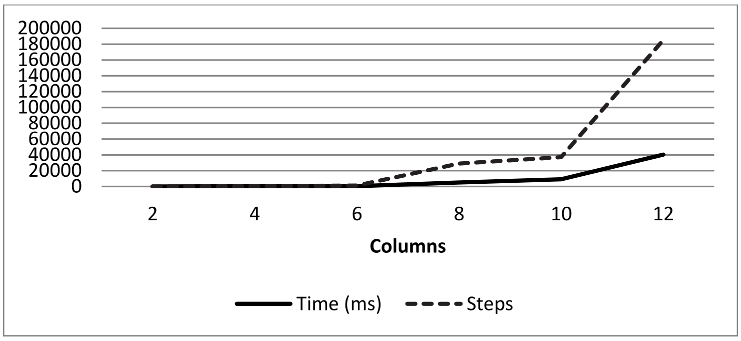

Fig. 2. Execution time dependency on number of columns [12]

Often processing of found rules takes more time than finding the rules. The time for rule processing includes adding rules to specific rule classes, sorting or grouping if necessary and writing them to text file. Rule processing time, number of generators and file size was measured in case of different frequency threshold values, ranging from 1 to 100. Those dependencies are presented in Fig. 3.

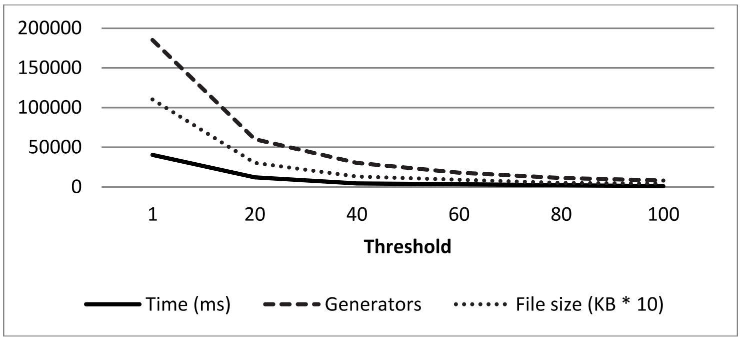

Fig. 3. Rule processing time dependency on frequency threshold [12]

5 Conclusions

In this paper an overview of a rule mining method called determinacy analysis (DA) is given and its main weakness – DA cannot mine rules free from inessential factors – is pointed out. For us the research question arose: how we can recognize such (zero) factors and how DA can mine zero factor free rules. We specified the types of zero factors, showed their connections to the generators and closed sets and found out that desirable DA rules should have a minimal generator in their left side. Based on that, we proposed an algorithm for finding rules where the left side is a minimal generator. The algorithm produces two types of rules: 1) IF generator THEN class; 2) IF generator THEN zero factor(s). The first type is a class detection rule; the second – an association rule. We also show how to interpret these rules for data analysis purposes.

Next we must improve a methodology for exploitation of the method and develop a user friendly solution for querying from the rule base and improve software.

References

1. Chesnokov, S.V.: Determination-analysis of social-economic data in dialogical regime (Preprint). All-Union Institute for Systems Research, Moscow (in Russian) (1980)
2. Chesnokov, S.V.: Determinacy analysis of social-economic data. Nauka, Moscow, Russia (in Russian) (1982)
3. Chesnokov, S.V.: Determinacy Analysis of Socio-Economic Data. Illustrative Materials to Lectures. Lecture 2: Rules. Lecture 3: Systems of Rules. Lomonosov Moscow State University, Faculty of Economics, Moscow, unpublished (in Russian) (2002)
4. Quinlan, J.R.: Learning Efficient Classification Procedures and their Application to Chess End Games. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds.) Machine learning. An Artificial Intelligence Approach, pp. 463–482. Springer-Verlag, Berlin Heidelberg New York Tokyo (1984)
5. Lind, G., Kuusik, R.: New developments for Determinacy Analysis: diclique-based approach. WSEAS Transactions on Information Science and Applications, 10(5), 1458–1469 (2008)
6. Lind, G., Kuusik, R.: Some Problems in Determinacy Analysis Approaches Development. In: Proceedings of the 2008 International Conference on Data Mining (DMIN 2008), pp.102–108 (2008)
7. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Pruning Closed Itemset Lattices for Association Rules. In: Proceedings of the BDA Conference, pp. 177–196 (1998)
8. Zaki, M.J., Hsiao, C.-J.: CHARM: An efficient algorithm for closed itemset mining. In: Proceedings of the Second SIAM International Conference on Data Mining, pp. 457–473 (2002)
9. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining Frequent Patterns with Counting Inference. ACM SIGKDD Explorations, 2(2), 66–75 (2000)
10. Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., Lakhal, L.: Mining Minimal Non-redundant Association Rules Using Frequent Closed Itemsets. In: Lloyd, J. et al. (eds.) Computational Logic – CL 2000. LNAI, vol. 1861, pp. 972–986. Springer (2000)
11. Quinlan, J.R.: Induction of Decision Trees. Machine Learning, 1(1), 81–106 (1986)
12. Jõgiste, L.: Prototyping of Zero-factor based DA. Master's Thesis, Tallinn University of Technology (2014)
13. UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/datasets/Nursery

Appendix G

Output of generator of hypotheses:

a) Intersections
b) Trees

- On the left: nodes have been selected by <u>maximal</u> frequency
- On the right: nodes have been selected by <u>minimal</u> frequency

a) Intersections

```
Alustatud:  2013  11  6  18  49
30.859

Fail: C:\…\SMorn.txt
Objekte  failis:  14,  objektis
tunnuseid: 5
Objektid:
Tunnused: 1-4
Objekte valjavotus: 14
Minimaalne lubatud sagedus = 1
Juhtsageduse valik: max sagedus

T4.2=8
T4.2&T3.1=4
T4.2&T3.1&T1.1=2
T4.2&T3.1&T1.1&T2.2=1
T4.2&T3.1&T1.1&T2.3=1
T4.2&T3.1&T2.2=2
T4.2&T3.1&T2.2&T1.3=1
T4.2&T3.1&T2.3=2
T4.2&T3.1&T2.3&T1.2=1
T4.2&T3.2=4
T4.2&T3.2&T1.3=2
T4.2&T3.2&T1.3&T2.1=1
T4.2&T3.2&T1.3&T2.2=1
T4.2&T3.2&T2.1=2
T4.2&T3.2&T2.1&T1.1=1
T4.2&T3.2&T1.2&T2.3=1
T4.2&T1.1=3
T4.2&T1.3=3
T4.2&T1.3&T2.2=2
T4.2&T2.2=3
T4.2&T2.3=3
T4.2&T2.3&T1.2=2
T3.1=7
T3.1&T2.2=4
T3.1&T2.2&T1.3=2
```

```
Alustatud:  2013  11  6  18  50
12.781

Fail: C:\…\SMorn.txt
Objekte  failis:  14,  objektis
tunnuseid: 5
Objektid:
Tunnused: 1-4
Objekte valjavotus: 14
Minimaalne lubatud sagedus = 1
Juhtsageduse valik: min sagedus

T1.2=4
T1.2&T2.1&T3.2&T4.1=1
T1.2&T2.2&T3.1&T4.1=1
T1.2&T2.3&T4.2=2
T1.2&T2.3&T4.2&T3.1=1
T1.2&T2.3&T4.2&T3.2=1
T1.2&T3.1=2
T1.2&T3.2=2
T1.2&T4.1=2
T2.1&T3.2=4
T2.1&T3.2&T1.1&T4.2=1
T2.1&T3.2&T1.3=2
T2.1&T3.2&T1.3&T4.1=1
T2.1&T3.2&T1.3&T4.2=1
T2.1&T3.2&T4.1=2
T2.1&T3.2&T4.2=2
T2.3=4
T2.3&T1.1&T3.1&T4.1=1
T2.3&T1.1&T3.1=2
T2.3&T1.1&T3.1&T4.2=1
T2.3&T3.1=3
T2.3&T3.1&T4.2=2
T2.3&T4.2=3
T1.1=5
T1.1&T2.2=2
```

```
T3.1&T2.2&T1.3&T4.1=1             T1.1&T2.2&T3.1&T4.2=1
T3.1&T2.2&T4.1=2                  T1.1&T2.2&T3.2&T4.1=1
T3.1&T2.2&T4.1&T1.2=1             T1.1&T3.2=2
T3.1&T1.1=3                       T1.1&T4.1=2
T3.1&T1.1&T2.3=2                  T1.1&T3.1=3
T3.1&T1.1&T2.3&T4.1=1             T1.1&T3.1&T4.2=2
T3.1&T2.3=3                       T1.1&T4.2=3
T3.1&T4.1=3                       T1.3=5
T3.1&T1.2=2                       T1.3&T2.2&T3.1=2
T3.2=7                            T1.3&T2.2&T3.1&T4.1=1
T3.2&T2.1=4                       T1.3&T2.2&T3.1&T4.2=1
T3.2&T2.1&T1.3=2                  T1.3&T4.1=2
T3.2&T2.1&T1.3&T4.1=1             T1.3&T2.2=3
T3.2&T2.1&T4.1=2                  T1.3&T2.2&T3.2&T4.2=1
T3.2&T2.1&T4.1&T1.2=1             T1.3&T2.2&T4.2=2
T3.2&T1.3=3                       T1.3&T3.2=3
T3.2&T4.1=3                       T1.3&T3.2&T4.2=2
T3.2&T4.1&T1.1&T2.2=1             T1.3&T4.2=3
T3.2&T1.1=2                       T2.2=6
T3.2&T1.2=2                       T2.2&T3.2=2
T3.2&T2.2=2                       T2.2&T4.1=3
T2.2=6                            T2.2&T4.1&T3.1=2
T2.2&T1.3=3                       T2.2&T4.2=3
T2.2&T4.1=3                       T2.2&T4.2&T3.1=2
T2.2&T1.1=2                       T2.2&T3.1=4
T4.1=6                            T4.1=6
T4.1&T1.1=2                       T4.1&T3.1=3
T4.1&T1.2=2                       T4.1&T3.2=3
T4.1&T1.3=2                       T3.1=7
T1.1=5                            T3.1&T4.2=4
T1.3=5                            T3.2=7
T1.2=4                            T3.2&T4.2=4
T2.3=4                            T4.2=8


Lopetatud:  2013  11  6  18  49   Lopetatud:  2013  11  6  18  50
31.292                            13.263
```

b) Trees

```
Alustatud:  2013  11  1  17  37   Alustatud:  2013  11  1  17  35
44.122                            48.862


Fail: C:\…\SMorn.txt              Fail: C:\…\SMorn.txt
Objekte  failis:  14,  objektis   Objekte  failis:  14,  objektis
tunnuseid: 5                      tunnuseid: 5
Objektid:                         Objektid:
Tunnused: 1-4                     Tunnused: 1-4
Objekte valjavotus: 14            Objekte valjavotus: 14
Minimaalne lubatud sagedus = 1    Minimaalne lubatud sagedus = 1
Juhtsageduse valik: max sagedus   Juhtsageduse valik: min sagedus
```

```
(8) 0.500(4)0.500(2)0.500(1)
T4.2=>T3.1  ->T1.1  ->T2.2
                    0.500(1)
                    ->T2.3
            0.500(2)0.500(1)
            ->T2.2  ->T1.3
            0.500(2)0.500(1)
            ->T2.3  ->T1.2
    0.500(4)0.500(2)0.500(1)
    =>T3.2  ->T1.3  ->T2.1
                    0.500(1)
                    ->T2.2
            0.500(2)0.500(1)
            ->T2.1  ->T1.1
            0.250(1)
            ->T1.2&T2.3
    0.375(3)
    =>T1.1
    0.375(3)0.667(2)
    =>T1.3  ->T2.2
    0.375(3)
    =>T2.2
    0.375(3)0.667(2)
    =>T2.3  ->T1.2

(7) 0.571(4)0.500(2)0.500(1)
T3.1=>T2.2  ->T1.3  ->T4.1
            0.500(2)0.500(1)
            ->T4.1  ->T1.2
    0.429(3)0.667(2)0.500(1)
    =>T1.1  ->T2.3  ->T4.1
    0.429(3)
    =>T2.3
    0.429(3)
    =>T4.1
    0.286(2)
    =>T1.2

(7) 0.571(4)0.500(2)0.500(1)
T3.2=>T2.1  ->T1.3  ->T4.1
            0.500(2)0.500(1)
            ->T4.1  ->T1.2
    0.429(3)
    =>T1.3
    0.429(3)0.333(1)
    =>T4.1  ->T1.1&T2.2
    0.286(2)
    =>T1.1
    0.286(2)
    =>T1.2
    0.286(2)
    =>T2.2
```

```
(4) 0.250(1)
T1.2=>T2.1&T3.2&T4.1
    0.250(1)
    =>T2.2&T3.1&T4.1
    0.500(2)   0.500(1)
    =>T2.3&T4.2->T3.1
               0.500(1)
               ->T3.2
    0.500(2)
    =>T3.1
    0.500(2)
    =>T3.2
    0.500(2)
    =>T4.1

(4)      0.250(1)
T2.1&T3.2=>T1.1&T4.2
         0.500(2)0.500(1)
         =>T1.3  ->T4.1
                 0.500(1)
                 ->T4.2
         0.500(2)
         =>T4.1
         0.500(2)
         =>T4.2

(4) 0.250(1)
T2.3=>T1.1&T3.1&T4.1
    0.500(2)   0.500(1)
    =>T1.1&T3.1->T4.2
    0.750(3)0.667(2)
    =>T3.1  ->T4.2
    0.750(3)
    =>T4.2

(5) 0.400(2)0.500(1)
T1.1=>T2.2  ->T3.1&T4.2
            0.500(1)
            ->T3.2&T4.1
    0.400(2)
    =>T3.2
    0.400(2)
    =>T4.1
    0.600(3)0.667(2)
    =>T3.1  ->T4.2
    0.600(3)
    =>T4.2

(5) 0.400(2)   0.500(1)
T1.3=>T2.2&T3.1->T4.1
               0.500(1)
               ->T4.2
    0.400(2)
```

```
(6) 0.500(3)
T2.2=>T1.3
    0.500(3)
    =>T4.1
    0.333(2)
    =>T1.1

(6) 0.333(2)
T4.1=>T1.1
    0.333(2)
    =>T1.2
    0.333(2)
    =>T1.3

(5)
T1.1

(5)
T1.3

(4)
T1.2

(4)
T2.3


Lopetatud: 2013 11 1 17 37 44.5
```

```
    =>T4.1
    0.600(3)0.333(1)
    =>T2.2  ->T3.2&T4.2
            0.667(2)
            ->T4.2
    0.600(3)0.667(2)
    =>T3.2  ->T4.2
    0.600(3)
    =>T4.2

(6) 0.333(2)
T2.2=>T3.2
    0.500(3)0.667(2)
    =>T4.1  ->T3.1
    0.500(3)0.667(2)
    =>T4.2  ->T3.1
    0.667(4)
    =>T3.1

(6) 0.500(3)
T4.1=>T3.1
    0.500(3)
    =>T3.2

(7) 0.571(4)
T3.1=>T4.2

(7) 0.571(4)
T3.2=>T4.2

(8)
T4.2


Lopetatud:  2013  11  1  17  35
49.318
```

CURRICULUM VITAE

Personal data

Name:	Grete Lind
Date of birth:	07.05.1971
Place of birth:	Tallinn, Estonia
Citizenship:	Estonian

Contact data

Address:	Akadeemia tee 15a, 12618 Tallinn
Phone:	+372 58847732
E-mail:	Grete.Lind@ttu.ee

Education

1999 – 2004	Tallinn University of Technology, Doctor's Degree Programme in Information Technology
1994 – 1998	Tallinn Technical University, M.Sc. in Information Processing
1989 – 1994	Tallinn Technical University, B.Sc. in Information Processing cum laude
1978 – 1989	Tallinn Secondary School no 37, silver medal

Language competence

Estonian	Mother Tongue
English	Fluent
Russian	Intermediate
French	Basic

Professional employment

1997 – …	Researcher, Tallinn University of Technology, Department of Informatics
Jan 1997 – Aug 1997	Engineer, Tallinn University of Technology, Department of Informatics

Publications

Lind, G. and Kuusik, R. (2016). Algorithm for Finding Zero Factor Free Rules. *Man-Machine Interactions 4: 4th International Conference on Man-Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6-9, 2015. AISC 391*, pp. 421-435. Springer.

Kuusik, R. and Lind, G. (2012). An Effective Inductive Learning Algorithm for Extracting Rules. *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science, 2: CACS 2011, Bali, Indonesia, November 15-17, 2011. AISC 145*, pp. 339-344. Berlin Heidelberg: Springer-Verlag.

Lind, G. and Kuusik, R. (2012). An Idea for Universal Generator of Hypotheses. *Proceedings of ICEIS 2012: the 14th International Conference on Enterprise Information Systems, Wrocław, Poland, 28 June – 1 July. Volume 1*, pp. 169-174. Portugal: SciTePress.

Kuusik, R. and Lind, G. (2011). New Developments of Determinacy Analysis. *Advanced Data Mining and Applications - 7th International Conference: ADMA 2011, Beijing, China, December 17-19, 2011. II; LNCS 7121*, pp. 223−236. Springer.

Kuusik, R. and Lind, G. (2011). New Solution for Extracting Inductive Learning Rules and their Post-Analysis. *The 1st International Conference on Advances in Information Mining and Management (IMMM 2011), Barcelona, Spain, October 23-28, 2011.* XPS (Xpert Publishing Services), pp. 121-126.

Kuusik, R. and Lind, G. (2010). Some Developments of Determinacy Analysis. *Advanced Data Mining and Applications: The 6th International Conference on Advanced Data Mining and Applications (ADMA2010), Chongqing, China, November 19-21, 2010. LNAI 6440*, pp. 593-602. Berlin Heidelberg: Springer-Verlag.

Kuusik, R., Treier, T., Lind, G. and Roosmann, P. (2009). Machine Learning Task as a Diclique Extracting Task. *Sixth International Conference on Fuzzy Systems and Knowledge Discovery: FSKD 2009, Tianjin, China, 14-16 August. Vol. 1*, p. 555−560. IEEE Computer Society.

Kuusik, R. and Lind, G. (2008). Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques. *WSEAS Transactions on Information Science and Applications, 5*(5), 599-611.

Kuusik, R. and Lind, G. (2008). An All Closed Set Finding Algorithm for Data Mining. *Advances on Artificial Intelligence, Knowledge Engineering and Data Bases: the 7th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED '08),*

University of Cambridge, February 20-22. Cambridge, UK: WSEAS Press, pp. 135-141.

Lind, G. and Kuusik, R. (2008). Determinacy Analysis as a diclique extracting task. *Proceedings of the 2nd European Computing Conference (ECC '08): New Aspects on Computing Research, Malta, September 11-13*. WSEAS Press (Recent Advances in Computer Engineering), pp. 119-125.

Lind, G. and Kuusik, R. (2008). New developments for Determinacy Analysis: diclique-based approach. *WSEAS Transactions on Information Science and Applications, 5*(10), 1458-1469.

Lind, G. and Kuusik, R. (2008). Some Problems in Determinacy Analysis Approaches Development. *Proceedings of the 2008 International Conference on Data Mining (DMIN 2008), Las Vegas, Nevada, USA, July 14-17, 2008, Volume I,* pp. 102-108. CSREA Press.

Roosmann, P., Võhandu, L., Kuusik, R., Treier, T. and Lind, G. (2008). A new inductive learning algorithm based on monotone system theory. *Recent Advances in Applied Computer Science - Proceedings of the 8th WSEAS International Conference on Applied Computer Science (ACS '08), Venice, Italy, November 21-23*. WSEAS Press, pp. 310-316.

Roosmann, P., Võhandu, L., Kuusik, R., Treier, T. and Lind, G. (2008). Monotone Systems approach in Inductive Learning. *International Journal of Applied Mathematics and Informatics, 2*(2), 47−56.

Lind, G. and Kuusik, R. (2007). Some Ideas for Determinacy Analysis Realisation. *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing. Palma de Mallorca, Spain, August 29-31, 2007*. ACTA Press, pp. 185-190.

Võhandu, L., Kuusik, R., Torim, A., Aab, E. and Lind, G. (2006). Some algorithms for data table (re)ordering using Monotone Systems. *Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED '06), Madrid, Spain, February 15-17*, pp. 417-422.

Võhandu, L., Kuusik, R., Torim, A., Aab, E. and Lind, G. (2006). Some Monotone Systems Algorithms for Data Mining. *WSEAS Transactions on Information Science and applications, 4*(3), 802−809.

Kuusik, R., Liiv, I. and Lind, G. (2005). An Efficient Method for Post Analysis of Patterns. *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications: February 14-16, 2005. Innsbruck, Austria*. ACTA Press, pp. 101−105.

Kuusik, R. and Lind, G. (2004). Generator of Hypotheses – an Approach of Data Mining Based on Monotone Systems Theory. *International Journal of Computational Intelligence, 1*, 49−53.

Kuusik, R. and Lind, G. (2004). New Frequency Pattern Algorithm for Data Mining. *Proceedings of the International 13th Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN), June 2004, Izmir, Turkey*, pp. 47−54.

Kuusik, R., Lind, G. and Võhandu, L. (2004). Data mining: pattern mining as a clique extracting task. *Proceedings of the Sixth International Conference on Enterprise Information Systems, Porto, Portugal, April 14-17, 2004. Vol. 2*, pp. 519−522. Porto, Portugal: INSTICC.

Kuusik, R., Lind, G. and Võhandu, L. (2004). Frequent pattern mining as a clique extracting task. *The 8th World Multi-Conference on Systemics, Cybernetics and Informatics, July 18-21, 2004 - Orlando, Florida, USA, SCI 2004 Proceedings. IV*, pp. 425−428. International Institute of Informatics and Systemics.

Kuusik, R., Lind, G. and Võhandu, L. (2004). Pattern Mining as a Clique Extracting Task. *Posters. Tenth International Conference IPMU 2004 Information Processing and Management of Uncertainty on Knowledge-Based Systems. Perugia, Italy, July 4-9, 2004*, pp. 19−20.

Kuusik, R. and Lind, G. (2003). An Approach of Data Mining Using Monotone Systems. *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS), April 2003, Angers, France, II*, pp. 482−485.

Lind, G. (2002). Monotone Systems in Data Mining. *Databases and Information Systems. Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, Tallinn, June 3-6, 2002. Vol. 2*, pp. 249−254. Tallinn: Institute of Cybernetics at Tallinn University of Technology.

Lind, G. (2000). Method for Data Mining – Generator of Hypotheses. *Databases and Information Systems. Proceedings of the 4th IEEE International Baltic Workshop, Vilnius, Lithuania, May 2000. Vol. 2*, pp. 304−305. Vilnius: Technika.

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Grete Lind
Sünniaeg:	07.05.1971
Sünnikoht:	Tallinn, Eesti
Kodakondsus:	Eesti

Kontaktandmed

Aadress:	Akadeemia tee 15a, 12618 Tallinn
Telefon:	+372 58847732
E-mail:	Grete.Lind@ttu.ee

Hariduskäik

1999 – 2004	Tallinna Tehnikaülikool, doktoriõpe
1994 – 1998	Tallinna Tehnikaülikool, tehnikateaduste magister
1989 – 1994	Tallinna Tehnikaülikool, infotöötluse eriala, diplom cum laude
1978 – 1989	Tallinna 37. Keskkool, hõbemedal

Keelteoskus

Eesti keel	Emakeel
Inglise keel	Kõrgtase
Vene keel	Kesktase
Prantsuse keel	Algtase

Teenistuskäik

1997 -…	Teadur, Tallinna Tehnikaülikool, Informaatikainstituut
Jaanuar 1997 – august 1997	insener, Tallinna Tehnikaülikool, Informaatikainstituut

Publikatsioonid

Lind, G. and Kuusik, R. (2016). Algorithm for Finding Zero Factor Free Rules. *Man-Machine Interactions 4: 4th International Conference on Man-Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6-9, 2015. AISC 391*, pp. 421-435. Springer.

Kuusik, R. and Lind, G. (2012). An Effective Inductive Learning Algorithm for Extracting Rules. *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science, 2: CACS 2011, Bali, Indonesia, November 15-17, 2011. AISC 145*, pp. 339-344. Berlin Heidelberg: Springer-Verlag.

Lind, G. and Kuusik, R. (2012). An Idea for Universal Generator of Hypotheses. *Proceedings of ICEIS 2012: the 14th International Conference on Enterprise Information Systems, Wrocław, Poland, 28 June – 1 July. Volume 1*, pp. 169-174. Portugal: SciTePress.

Kuusik, R. and Lind, G. (2011). New Developments of Determinacy Analysis. *Advanced Data Mining and Applications - 7th International Conference: ADMA 2011, Beijing, China, December 17-19, 2011. II; LNCS 7121*, pp. 223−236. Springer.

Kuusik, R. and Lind, G. (2011). New Solution for Extracting Inductive Learning Rules and their Post-Analysis. *The 1st International Conference on Advances in Information Mining and Management (IMMM 2011), Barcelona, Spain, October 23-28, 2011*. XPS (Xpert Publishing Services), pp. 121-126.

Kuusik, R. and Lind, G. (2010). Some Developments of Determinacy Analysis. *Advanced Data Mining and Applications: The 6th International Conference on Advanced Data Mining and Applications (ADMA2010), Chongqing, China, November 19-21, 2010. LNAI 6440*, pp. 593-602. Berlin Heidelberg: Springer-Verlag.

Kuusik, R., Treier, T., Lind, G. and Roosmann, P. (2009). Machine Learning Task as a Diclique Extracting Task. *Sixth International Conference on Fuzzy Systems and Knowledge Discovery: FSKD 2009, Tianjin, China, 14-16 August. Vol. 1*, p. 555−560. IEEE Computer Society.

Kuusik, R. and Lind, G. (2008). Algorithm MONSA for All Closed Sets Finding: basic concepts and new pruning techniques. *WSEAS Transactions on Information Science and Applications, 5*(5), 599-611.

Kuusik, R. and Lind, G. (2008). An All Closed Set Finding Algorithm for Data Mining. *Advances on Artificial Intelligence, Knowledge Engineering and Data Bases: the 7th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED '08), University of Cambridge, February 20-22*. Cambridge, UK: WSEAS Press, pp. 135-141.

Lind, G. and Kuusik, R. (2008). Determinacy Analysis as a diclique extracting task. *Proceedings of the 2nd European Computing Conference (ECC '08): New Aspects on Computing Research, Malta, September 11-13*. WSEAS Press (Recent Advances in Computer Engineering), pp. 119-125.

Lind, G. and Kuusik, R. (2008). New developments for Determinacy Analysis: diclique-based approach. *WSEAS Transactions on Information Science and Applications, 5*(10), 1458-1469.

Lind, G. and Kuusik, R. (2008). Some Problems in Determinacy Analysis Approaches Development. *Proceedings of the 2008 International Conference on Data Mining (DMIN 2008), Las Vegas, Nevada, USA, July 14-17, 2008, Volume I,* pp. 102-108. CSREA Press.

Roosmann, P., Võhandu, L., Kuusik, R., Treier, T. and Lind, G. (2008). A new inductive learning algorithm based on monotone system theory. *Recent Advances in Applied Computer Science - Proceedings of the 8th WSEAS International Conference on Applied Computer Science (ACS '08), Venice, Italy, November 21-23*. WSEAS Press, pp. 310-316.

Roosmann, P., Võhandu, L., Kuusik, R., Treier, T. and Lind, G. (2008). Monotone Systems approach in Inductive Learning. *International Journal of Applied Mathematics and Informatics, 2*(2), 47−56.

Lind, G. and Kuusik, R. (2007). Some Ideas for Determinacy Analysis Realisation. *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing. Palma de Mallorca, Spain, August 29-31, 2007*. ACTA Press, pp. 185-190.

Võhandu, L., Kuusik, R., Torim, A., Aab, E. and Lind, G. (2006). Some algorithms for data table (re)ordering using Monotone Systems. *Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED '06), Madrid, Spain, February 15-17*, pp. 417-422.

Võhandu, L., Kuusik, R., Torim, A., Aab, E. and Lind, G. (2006). Some Monotone Systems Algorithms for Data Mining. *WSEAS Transactions on Information Science and applications, 4*(3), 802−809.

Kuusik, R., Liiv, I. and Lind, G. (2005). An Efficient Method for Post Analysis of Patterns. *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications: February 14-16, 2005. Innsbruck, Austria*. ACTA Press, pp. 101−105.

Kuusik, R. and Lind, G. (2004). Generator of Hypotheses – an Approach of Data Mining Based on Monotone Systems Theory. *International Journal of Computational Intelligence, 1*, 49−53.

Kuusik, R. and Lind, G. (2004). New Frequency Pattern Algorithm for Data Mining. *Proceedings of the International 13th Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN), June 2004, Izmir, Turkey*, pp. 47−54.

Kuusik, R., Lind, G. and Võhandu, L. (2004). Data mining: pattern mining as a clique extracting task. *Proceedings of the Sixth International Conference on Enterprise Information Systems, Porto, Portugal, April 14-17, 2004. Vol. 2*, pp. 519−522. Porto, Portugal: INSTICC.

Kuusik, R., Lind, G. and Võhandu, L. (2004). Frequent pattern mining as a clique extracting task. *The 8th World Multi-Conference on Systemics, Cybernetics and Informatics, July 18-21, 2004 - Orlando, Florida, USA, SCI 2004 Proceedings. IV*, pp. 425−428. International Institute of Informatics and Systemics.

Kuusik, R., Lind, G. and Võhandu, L. (2004). Pattern Mining as a Clique Extracting Task. *Posters. Tenth International Conference IPMU 2004 Information Processing and Management of Uncertainty on Knowledge-Based Systems. Perugia, Italy, July 4-9, 2004*, pp. 19−20.

Kuusik, R. and Lind, G. (2003). An Approach of Data Mining Using Monotone Systems. *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS), April 2003, Angers, France, II*, pp. 482−485.

Lind, G. (2002). Monotone Systems in Data Mining. *Databases and Information Systems. Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, Tallinn, June 3-6, 2002. Vol. 2*, pp. 249−254. Tallinn: Institute of Cybernetics at Tallinn University of Technology.

Lind, G. (2000). Method for Data Mining – Generator of Hypotheses. *Databases and Information Systems. Proceedings of the 4th IEEE International Baltic Workshop, Vilnius, Lithuania, May 2000. Vol. 2*, pp. 304−305. Vilnius: Technika.

DISSERTATIONS DEFENDED AT TALLINN UNIVERSITY OF TECHNOLOGY ON *INFORMATICS AND SYSTEM ENGINEERING*

1. **Lea Elmik**. Informational Modelling of a Communication Office. 1992.

2. **Kalle Tammemäe**. Control Intensive Digital System Synthesis. 1997.

3. **Eerik Lossmann**. Complex Signal Classification Algorithms, Based on the Third-Order Statistical Models. 1999.

4. **Kaido Kikkas**. Using the Internet in Rehabilitation of People with Mobility Impairments – Case Studies and Views from Estonia. 1999.

5. **Nazmun Nahar**. Global Electronic Commerce Process: Business-to-Business. 1999.

6. **Jevgeni Riipulk**. Microwave Radiometry for Medical Applications. 2000.

7. **Alar Kuusik**. Compact Smart Home Systems: Design and Verification of Cost Effective Hardware Solutions. 2001.

8. **Jaan Raik**. Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams. 2001.

9. **Andri Riid**. Transparent Fuzzy Systems: Model and Control. 2002.

10. **Marina Brik**. Investigation and Development of Test Generation Methods for Control Part of Digital Systems. 2002.

11. **Raul Land**. Synchronous Approximation and Processing of Sampled Data Signals. 2002.

12. **Ants Ronk**. An Extended Block-Adaptive Fourier Analyser for Analysis and Reproduction of Periodic Components of Band-Limited Discrete-Time Signals. 2002.

13. **Toivo Paavle**. System Level Modeling of the Phase Locked Loops: Behavioral Analysis and Parameterization. 2003.

14. **Irina Astrova**. On Integration of Object-Oriented Applications with Relational Databases. 2003.

15. **Kuldar Taveter**. A Multi-Perspective Methodology for Agent-Oriented Business Modelling and Simulation. 2004.

16. **Taivo Kangilaski**. Eesti Energia käiduhaldussüsteem. 2004.

17. **Artur Jutman**. Selected Issues of Modeling, Verification and Testing of Digital Systems. 2004.

18. **Ander Tenno**. Simulation and Estimation of Electro-Chemical Processes in Maintenance-Free Batteries with Fixed Electrolyte. 2004.

19. **Oleg Korolkov**. Formation of Diffusion Welded Al Contacts to Semiconductor Silicon. 2004.

20. **Risto Vaarandi**. Tools and Techniques for Event Log Analysis. 2005.

21. **Marko Koort**. Transmitter Power Control in Wireless Communication Systems. 2005.

22. **Raul Savimaa**. Modelling Emergent Behaviour of Organizations. Time-Aware, UML and Agent Based Approach. 2005.

23. **Raido Kurel**. Investigation of Electrical Characteristics of SiC Based Complementary JBS Structures. 2005.

24. **Rainer Taniloo**. Ökonoomsete negatiivse diferentsiaaltakistusega astmete ja elementide disainimine ja optimeerimine. 2005.

25. **Pauli Lallo**. Adaptive Secure Data Transmission Method for OSI Level I. 2005.

26. **Deniss Kumlander**. Some Practical Algorithms to Solve the Maximum Clique Problem. 2005.

27. **Tarmo Veskioja**. Stable Marriage Problem and College Admission. 2005.

28. **Elena Fomina**. Low Power Finite State Machine Synthesis. 2005.

29. **Eero Ivask**. Digital Test in WEB-Based Environment 2006.

30. **Виктор Войтович**. Разработка технологий выращивания из жидкой фазы эпитаксиальных структур арсенида галлия с высоковольтным p-n переходом и изготовления диодов на их основе. 2006.

31. **Tanel Alumäe**. Methods for Estonian Large Vocabulary Speech Recognition. 2006.

32. **Erki Eessaar**. Relational and Object-Relational Database Management Systems as Platforms for Managing Softwareengineering Artefacts. 2006.

33. **Rauno Gordon**. Modelling of Cardiac Dynamics and Intracardiac Bio-impedance. 2007.

34. **Madis Listak**. A Task-Oriented Design of a Biologically Inspired Underwater Robot. 2007.

35. **Elmet Orasson**. Hybrid Built-in Self-Test. Methods and Tools for Analysis and Optimization of BIST. 2007.

36. **Eduard Petlenkov**. Neural Networks Based Identification and Control of Nonlinear Systems: ANARX Model Based Approach. 2007.

37. **Toomas Kirt**. Concept Formation in Exploratory Data Analysis: Case Studies of Linguistic and Banking Data. 2007.

38. **Juhan-Peep Ernits**. Two State Space Reduction Techniques for Explicit State Model Checking. 2007.

39. **Innar Liiv**. Pattern Discovery Using Seriation and Matrix Reordering: A Unified View, Extensions and an Application to Inventory Management. 2008.

40. **Andrei Pokatilov**. Development of National Standard for Voltage Unit Based on Solid-State References. 2008.

41. **Karin Lindroos**. Mapping Social Structures by Formal Non-Linear Information Processing Methods: Case Studies of Estonian Islands Environments. 2008.

42. **Maksim Jenihhin**. Simulation-Based Hardware Verification with High-Level Decision Diagrams. 2008.

43. **Ando Saabas**. Logics for Low-Level Code and Proof-Preserving Program Transformations. 2008.

44. **Ilja Tšahhirov**. Security Protocols Analysis in the Computational Model – Dependency Flow Graphs-Based Approach. 2008.

45. **Toomas Ruuben**. Wideband Digital Beamforming in Sonar Systems. 2009.

46. **Sergei Devadze**. Fault Simulation of Digital Systems. 2009.

47. **Andrei Krivošei**. Model Based Method for Adaptive Decomposition of the Thoracic Bio-Impedance Variations into Cardiac and Respiratory Components. 2009.

48. **Vineeth Govind**. DfT-Based External Test and Diagnosis of Mesh-like Networks on Chips. 2009.

49. **Andres Kull**. Model-Based Testing of Reactive Systems. 2009.

50. **Ants Torim**. Formal Concepts in the Theory of Monotone Systems. 2009.

51. **Erika Matsak**. Discovering Logical Constructs from Estonian Children Language. 2009.

52. **Paul Annus**. Multichannel Bioimpedance Spectroscopy: Instrumentation Methods and Design Principles. 2009.

53. **Maris Tõnso**. Computer Algebra Tools for Modelling, Analysis and Synthesis for Nonlinear Control Systems. 2010.

54. **Aivo Jürgenson**. Efficient Semantics of Parallel and Serial Models of Attack Trees. 2010.

55. **Erkki Joasoon**. The Tactile Feedback Device for Multi-Touch User Interfaces. 2010.

56. **Jürgo-Sören Preden**. Enhancing Situation – Awareness Cognition and Reasoning of Ad-Hoc Network Agents. 2010.

57. **Pavel Grigorenko**. Higher-Order Attribute Semantics of Flat Languages. 2010.

58. **Anna Rannaste**. Hierarcical Test Pattern Generation and Untestability Identification Techniques for Synchronous Sequential Circuits. 2010.

59. **Sergei Strik**. Battery Charging and Full-Featured Battery Charger Integrated Circuit for Portable Applications. 2011.

60. **Rain Ottis**. A Systematic Approach to Offensive Volunteer Cyber Militia. 2011.

61. **Natalja Sleptšuk**. Investigation of the Intermediate Layer in the Metal-Silicon Carbide Contact Obtained by Diffusion Welding. 2011.

62. **Martin Jaanus**. The Interactive Learning Environment for Mobile Laboratories. 2011.

63. **Argo Kasemaa**. Analog Front End Components for Bio-Impedance Measurement: Current Source Design and Implementation. 2011.

64. **Kenneth Geers**. Strategic Cyber Security: Evaluating Nation-State Cyber Attack Mitigation Strategies. 2011.

65. **Riina Maigre**. Composition of Web Services on Large Service Models. 2011.

66. **Helena Kruus**. Optimization of Built-in Self-Test in Digital Systems. 2011.

67. **Gunnar Piho**. Archetypes Based Techniques for Development of Domains, Requirements and Sofware. 2011.

68. **Juri Gavšin**. Intrinsic Robot Safety Through Reversibility of Actions. 2011.

69. **Dmitri Mihhailov**. Hardware Implementation of Recursive Sorting Algorithms Using Tree-like Structures and HFSM Models. 2012.

70. **Anton Tšertov**. System Modeling for Processor-Centric Test Automation. 2012.

71. **Sergei Kostin**. Self-Diagnosis in Digital Systems. 2012.

72. **Mihkel Tagel**. System-Level Design of Timing-Sensitive Network-on-Chip Based Dependable Systems. 2012.

73. **Juri Belikov**. Polynomial Methods for Nonlinear Control Systems. 2012.

74. **Kristina Vassiljeva**. Restricted Connectivity Neural Networks based Identification for Control. 2012.

75. **Tarmo Robal**. Towards Adaptive Web – Analysing and Recommending Web Users` Behaviour. 2012.

76. **Anton Karputkin**. Formal Verification and Error Correction on High-Level Decision Diagrams. 2012.

77. **Vadim Kimlaychuk**. Simulations in Multi-Agent Communication System. 2012.

78. **Taavi Viilukas**. Constraints Solving Based Hierarchical Test Generation for Synchronous Sequential Circuits. 2012.

79. **Marko Kääramees**. A Symbolic Approach to Model-based Online Testing. 2012.

80. **Enar Reilent**. Whiteboard Architecture for the Multi-agent Sensor Systems. 2012.

81. **Jaan Ojarand**. Wideband Excitation Signals for Fast Impedance Spectroscopy of Biological Objects. 2012.

82. **Igor Aleksejev**. FPGA-based Embedded Virtual Instrumentation. 2013.

83. **Juri Mihhailov**. Accurate Flexible Current Measurement Method and its Realization in Power and Battery Management Integrated Circuits for Portable Applications. 2013.

84. **Tõnis Saar**. The Piezo-Electric Impedance Spectroscopy: Solutions and Applications. 2013.

85. **Ermo Täks**. An Automated Legal Content Capture and Visualisation Method. 2013.

86. **Uljana Reinsalu**. Fault Simulation and Code Coverage Analysis of RTL Designs Using High-Level Decision Diagrams. 2013.

87. **Anton Tšepurov**. Hardware Modeling for Design Verification and Debug. 2013.

88. **Ivo Müürsepp**. Robust Detectors for Cognitive Radio. 2013.

89. **Jaas Ježov**. Pressure sensitive lateral line for underwater robot. 2013.

90. **Vadim Kaparin**. Transformation of Nonlinear State Equations into Observer Form. 2013.

92. **Reeno Reeder**. Development and Optimisation of Modelling Methods and Algorithms for Terahertz Range Radiation Sources Based on Quantum Well Heterostructures. 2014.

93. **Ants Koel**. GaAs and SiC Semiconductor Materials Based Power Structures: Static and Dynamic Behavior Analysis. 2014.

94. **Jaan Übi**. Methods for Coopetition and Retention Analysis: An Application to University Management. 2014.

115. **Aivo Anier**. Model Based Framework for Distributed Control and Testing of Cyber-Physical Systems. 2016.

116. **Denis Firsov**. Certification of Context-Free Grammar Algorithms. 2016.

117. **Sergei Astapov**. Distributed Signal Processing for Situation Assessment in Cyber-Physical Systems. 2016.

118. **Erkki Moorits**. Embedded Software Solutions for Development of Marine Navigation Light Systems. 2016.

119. **Andres Ojamaa**. Software Technology for Cyber Security Simulations. 2016.

120. **Gert Toming**. Fluid Body Interaction of Biomimetic Underwater Robots. 2016.

121. **Kadri Umbleja**. Competence Based Learning – Framework, Implementation, Analysis and Management of Learning Process. 2017.

122. **Andres Hunt**. Application-Oriented Performance Characterization of the Ionic Polymer Transducers (IPTs). 2017.

123. **Niccolò Veltri**. A Type-Theoretical Study of Nontermination. 2017.

124. **Tauseef Ahmed**. Radio Spectrum and Power Optimization Cognitive Techniques for Wireless Body Area Networks. 2017.

125. **Andre Veski**. Agent-Based Computational Experiments in Two-Sided Matching Markets. 2017

126. **Artjom Rjabov**. Network-Based Hardware Accelerators for Parallel Data Processing. 2017.

127. **Fatih Güllü**. Conformity Analysis of E-Learning Systems at Largest Universities in Estonia and Turkey on the Basis of EES Model. 2017.

128. **Margarita Spitšakova**. Discrete Gravitational Swarm Optimization Algorithm for System Identification. 2017.