Binary Factor Analysis with Help of Formal Concepts

Aleš Keprt and Václav Snášel

Department of Computer Science, FEI, VŠB – Technical University Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{ales.keprt, vaclav.snasel}@vsb.cz

Abstract. Binary factor analysis (BFA, also known as Boolean Factor Analysis) is a nonhierarchical analysis of binary data, based on reduction of binary space dimension. It allows us to find hidden relationships in binary data, which can be used for data compression, data mining, or intelligent data comparison for information retrieval. Unfortunately, we can't effectively use classical (i.e. non-binary) factor analysis methods for binary data. In this article we show an approach based on utilizing formal concept analysis to compute nonhierarchical BFA. Computation of a concept lattice is a computationally expensive task too, still it helps us to speed up the BFA computation.

Keywords: Binary factor analysis, boolean algebra, formal concepts

1 Introduction

Binary data are one of the basic stones of computers. In the ancient age of computer science even non-binary information was forcibly transformed into and stored in binary form, usually because of technical limitations. Today we can still see binary data on a physical level, but not on a logical level. Instead, majority of information is usually recorder and stored in its native or nearly native form, which is non-binary in most cases. Sometimes, it goes even further - with the help of fuzzy computing, even vague information can be processed successfully. From one particular point of view, the vagueness can be understood as a counterpart of the strictly binary data.

Data analysis and searching for important, but often hidden information isn't a new theme, it was already subject of statistics (statistology) long time before informatics become an independent science. Although data analysis is in no way new theme, it's still very current one. From computer science point of view, it's important especially because of rapidly growing importance of Internet, or as a consequence of general emphasis on economics and economical achievements.

In this point we realize, that with the current boom of new kinds of data analysis and data mining, binary data is no more in the main focus. Although people perceive the majority of real world quantities as non-binary, some information still has got binary nature. In poetic words: "Binariness can occur."

We can use non-binary data analysis techniques for binary data as well, but these techniques are usually based on linear algebra, approximation or finding of global minima/maxima, and those don't work well in the binary world. They all fail because of the specific nature of binary data, which also requires specific analytical methods.

Our goal is the research in the filed of binary factor analysis (BFA). BFA is a nonlinear analysis of binary data, where neither classical linear algebra, nor mathematical (functional) analysis can be used. In the past, it was repeatedly experimentally proven that classical non-binary methods followed by the alignment of the results into binary or other dichotomous values give ill results. This resulted in creation of some new methods, which use boolean (binary) algebra. These binary methods were published in last 8 years, and are based on neural networks, combinatorial searching, genetic algorithms, and transformation to the problem of concept lattices. This paper focuses to the approach based on formal concepts.

2 Binary factor analysis

2.1 Problem definition

To describe the problem of Binary Factor Analysis (BFA) we can paraphrase BMDP's documentation (Bio-Medical Data Processing, see [1]).

BFA is a factor analysis of dichotomous (binary) data. This kind of analysis differs from the classical factor analysis (see [18]) of binary valued data, even though the goal and the model are symbolically similar. In other words, both classical and binary analysis use symbolically the same notation, but their senses are different.

The goal of BFA is to express p variables $(x_1, x_2, \ldots, x_p)$ by m factors $(f_1, f_2, \ldots, f_m)$, where $m \ll p$ (m is considerably smaller than p). The model can be written as

$$\mathsf{X} = \mathsf{F} \odot \mathsf{A}$$

$$\begin{bmatrix} x_{1,1} & \ldots & x_{1,p} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \ldots & x_{n,p} \end{bmatrix} = \begin{bmatrix} f_{1,1} & \ldots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{n,1} & \ldots & f_{n,m} \end{bmatrix} \odot \begin{bmatrix} a_{1,1} & \ldots & a_{1,p} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \ldots & a_{m,p} \end{bmatrix}$$

where $\odot$ is binary matrix multiplication. For n cases, data matrix X, factor scores matrix F, and factor loadings matrix A. The elements of all matrices are valued 0 or 1 (i.e. binary).

2.2 Difference to classical factor analysis

Binary factor analysis uses boolean algebra, so matrices of factor scores and loadings are both binary. See the following example: The result is 2 in classical algebra

$$[1\ 1\ 0\ 1] \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 = 2$$

but it's 1 when using boolean algebra.

$$[1\ 1\ 0\ 1] \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = 1 \cdot 1 \oplus 1 \cdot 1 \oplus 0 \cdot 0 \oplus 1 \cdot 0 = 1$$

Sign $\oplus$ marks disjunction (logical sum), and sign $\cdot$ mars conjunction (logical conjunction). Note that since we focus to binary values, the logical conjunction is actually identical to the classic product.

In classical factor analysis, the score for each case, for a particular factor, is a linear combination of all variables: variables with large loadings all contribute to the score. In boolean factor analysis, a case has a score of one if it has a positive response for any of the variables dominant in the factor (i.e. those not having zero loadings) and zero otherwise.

2.3 Success and discrepancy

Obviously, not every X can be expressed as $\mathsf{F} \odot \mathsf{A}$. The success of BFA is measured by comparing the observed binary responses (X) with those estimated by multiplying the loadings and the scores ($\hat{\mathsf{X}} = \mathsf{F} \odot \mathsf{A}$). We count both positive and negative discrepancies. Positive discrepancy is when the observed value (in X) is one and the analysis (in $\hat{\mathsf{X}}$) estimates it to be zero, and reversely negative discrepancy is when the observed value is zero and the analysis estimates it to be one. Total count of discrepancies d is a suitable measure of difference between observed values $x_{i,j}$ and calculated values $\hat{x}_{i,j}$.

$$d = \sum_{i=1}^{n} \sum_{j=1}^{p} |\hat{x}_{i,j} - x_{i,j}|$$

2.4 Terminology notes

Let's summarize the terminology we use. Data to be analyzed are in matrix X. Its columns x_j represent **variables**, whereas its rows x_i represent **cases**. The factor analysis comes out from the generic thesis saying that variables, we can observe, are just the effect of the factors, which are the real origin. (You can find more details in [18].) So we focus on factors. We also try to keep number of factors as low as possible, so we can say "reducing variables to factors".

The result is the pair of matrices. Matrix of **factor scores** F expresses the input data by factors instead of variables. Matrix of **factor loadings** A defines the relation between variables and factors, i.e. each row a_i defines one particular factor.

2.5 An example

As a basic example (see [1]) we consider a serological problem[1], where p tests are performed on the blood of each of n subjects (by adding p reagents). The outcome is described as positive (a value of one is assigned for the test in data matrix), or negative (zero is assigned). In medical terms, the scores can be interpreted as antigens (for each subject), and the loading as antibodies (for each test reagent). See [14] for more on these terms.

2.6 Application to text documents

BFA can be also used to analyse a collection of text documents. In that case the data matrix X is built up of a collection of text documents D represented as p-dimensional binary vectors d_i, $i \in 1, 2, \ldots, n$. Columns of X represent particular words. Particular cell $x_{i,j}$ equals to *one* when document i contains word j, and *zero* otherwise. In other words, data matrix X is built in a very intuitive way.

It should be noted that some kind of smart (i.e. semantic) preprocessing could be made in order to let the analysis make more sense. For example we usually want to take *world* and *worlds* as the same word. Although the binary factor analysis has no problems with finding this kind similarities itself, it is computationally very expensive, so any kind of preprocessing which can decrease the size of input data matrix X is very useful. We can also use WordNet, or thesaurus to combine synonyms. For additional details see [5].

3 The goal of exact binary factor analysis

In classic factor analysis, we don't even try to find 100% perfect solution, because it's simply impossible. Fortunately, there are many techniques that give a good suboptimal solution (see [18]). Unfortunately, these classic factor analysis techniques are not directly applicable to our special binary conditions. While classic techniques are based on the system of correlations and approximations, these terms can be hardly used in binary world. Although it is possible to apply classic (i.e. non-boolean non-binary) factor analysis to binary data, if we really focus to BFA with restriction to boolean arithmetic, we must advance another way.

You can find the basic BFA solver in BMDP – Bio-Medical Data Processing software package (see [1]). Unfortunately, BMDP became a commercial product, so the source code of this software package isn't available to the public, and even the BFA solver itself isn't available anymore. Yet worse, there are suspicions saying that BMDP's utility is useless, as it actually just guesses the F and A matrices, and then only explores the similar matrices, so it only finds local minimum of the vector error function.

[1] Serologic test is a blood test to detect the presence of antibodies against microorganism. See serology entry in [14].

One interesting suboptimal BFA method comes from Húsek, Frolov et al. (see [16], [7], [6], [2], [8]). It is based on a Hopfield-like neural network, so it finds a suboptimal solution. The main advantage of this method is that it can analyse very large data sets, which can't be simply processed by exact BFA methods.

Although the mentioned neural network based solver is promising, we actually didn't have any one really exact method, which could be used to proof the other (suboptimal) BFA solvers. So we started to work on it.

4 Blind search based solver

The very basic algorithm blindly searches among all possible combinations of F and A. This is obviously 100% exact, but also extremely computational expensive, which makes this kind of solver in its basic implementation simply unusable.

To be more exact, we can express the limits of blind search solver in units of n, p and m. Since we need to express matrix X as the product of matrices $\mathsf{F} \odot \mathsf{A}$, which are $n \times m$ and $m \times p$ in size, we need to try on all combinations of $m \cdot (n + p)$ bits. And this is very limiting, even when trying to find only 3 factors from 10×10 data set ($m = 3$, $n = 10$, $p = 10$), we end up with computational complexity of $2^{m \cdot (n+p)} = 2^{60}$, which is quite behind the scope of current computers.

Although the blind search algorithm can be optimized (see [8],[11]), it's still quite unusable in real world.

5 Concept lattices

Another method of solving BFA problem is based on concept lattices. This section gives minimum necessary introduction to concept lattices, and especially *formal concepts*, which are the key part of the algorithm.

Definition 1 (Formal context, objects, attributes).
Triple (X, Y, R), where X and Y are sets, and R is a binary relation $R \subseteq X \times Y$, is called **formal context**. *Elements of X are called* **objects**, *and elements of Y are called* **attributes**. *We say "object A has attribute B", just when $A \subseteq X$, $B \subseteq Y$ and $(A, B) \in R$.* □

Definition 2 (Derivation operators).
For subsets $A \subseteq X$ and $B \subseteq Y$, we define

$$A^{\uparrow} = \{b \in B \mid \forall a \in A : (a, b) \in R\}$$
$$B^{\downarrow} = \{a \in A \mid \forall b \in B : (a, b) \in R\}$$

□

In other words, $A^{\uparrow}$ is the set of attributes common to all objects of A, and similarly $B^{\downarrow}$ is the set of all objects, which have all attributes of B.

Note: We just defined two operators $^{\uparrow}$ and $^{\downarrow}$:

$$^{\uparrow} : P(X) \to P(Y)$$
$$^{\downarrow} : P(Y) \to P(X)$$

where $P(X)$ and $P(Y)$ are sets of all subsets of X and Y respectively.

Definition 3 (Formal concept).
Let (X, Y, R) be a formal context. Then pair (A, B), where $A \subseteq X$, $B \subseteq Y$, $A^{\uparrow} = B$ and $B^{\downarrow} = A$, is called **formal concept** *of (X, Y, R).*
Set A is called **extent** *of (A, B), and set B is called* **intent** *of (A, B).* □

Definition 4 (Concept ordering).
Let (A_1, B_1) and (A_2, B_2) be formal concepts. Then (A_1, B_1) is called subconcept of (A_2, B_2), just when $A_1 \subseteq A_2$ (which is equivalent to $B_1 \supseteq B_1$). We write $(A_1, B_1) \leq (A_2, B_2)$. We can also say that (A_2, B_2) is superconcept of (A_1, B_1). □

In this article we just need to know the basics of concepts and their meaning. For more detailed, descriptive, and well understandable introduction to Formal Concept Analysis and Concept Lattices, see [3], [12] or [15].

6 BFA using formal concepts

If we want to speed up the simple blind-search algorithm, we can try to find some factor *candidates*, instead of checking out all possible bit-combinations. The technique which can help us significantly is Formal Concept Analysis (FCA, see [12]). FCA is based on concept lattices, but we actually work with formal concepts only, so the theory we need is quite simple.

6.1 The strategy

We can still use some good parts of the blind-search program (matrix optimizations, optimized bitwise operations using boolean algebra, etc.), but instead of checking out all possible bit combinations, we work with concepts as the factor candidates. In addition, we can adopt some strategy optimizations (as discussed above) to concepts, so the final algorithm is quite fast; its strength actually relies on the concept-building algorithm we use.

So the BFA algorithm is then as follows:

1. Compute all concepts of X. (We use a standalone program based on Lindig's algorithm.)
2. Import the list of concepts, and *optimize* it, so it correspond to our optimized data matrix X. (This is simple. We just merge objects and attributes the same way, as we merged duplicate rows and columns of X respectively.)

3. Remove all concepts with too many one's. (The number of one's per factor is one of our starting constraints.)
4. Use the remaining concepts as the factor candidates, and find the best m-element subset (according to discrepancy formulae).

This way we can find the BFA solution quite fast, compared to the blind search algorithm. Although the algorithm described here looks quite simple[2], there is a couple of things, we must be aware of.

6.2 More details

The most important FCA consequence is that 100% correct BFA solution can always be found among all subsets of concepts. This is very important, because it is the main guarantee of the correctness of the concept based BFA solver. (This is Keprt's theorem, firstly published in [11].).

Other important feature of FCA based concepts is that they never directly generate any negative discrepancy. It is a direct consequence of FCA qualities, and affects the semantic sense of the result. As we discussed above (and see also [1]), negative discrepancy is a case when $\mathsf{F} \odot \mathsf{A}$ gives 1 when it should be 0. From semantic point of view, this (the negative discrepancy) is commonly unwanted phenomenon. In consequence, the fact that there's no negative discrepancy in the concepts, may have negative impact on the result, but the reality is usually right opposite. (Compare this to the quick sort phenomenon.)

The absence of negative discrepancies coming from concepts applies to A matrix only. It doesn't apply to F matrix, we still can use any suitable values for it. In consequence, we always start with concepts not generating negative discrepancy, which are semantically better, and end up with best suitable factor scores F, which give the lowest discrepancy. So it seems to be quite good feature.

6.3 Implementation issues

It's clear that the data matrix X is usually quite large, and makes the finding of the formal concepts the main issue. Currently we use the standalone CL (concept lattice) builder. It is optimized for finding concept lattices, but that's not right what we need. In the future, we should consider adopting some kind of CL building algorithm directly into BFA solver. This will save a lot of time when working with large data sets, because we don't need to know the concept hierarchy.

We don't even need to know all the formal concepts, because the starting constraints limit the maximum number of one's in a factor, which is directly applicable to CL building.

[2] *Everything's simple, when you know it.*

7 Experiments - results and computation times

Here we present three experiments using typical data sets taken from other papers discussing BFA (see [17], [16], [7], [6], [2], [8]). We focus not only to results (i.e. discrepancy of the solutions), but also to the computation times. BFA computation is always a time consuming task, so faster algorithms are generally preferred.

7.1 Data sets p3 and p2

Data set p3 is a representative of the simplest data. It is a sparse matrix, 100×100 bits in size. This data matrix can be expressed with just 5 factors, what leads to searching for two 500-bit matrices. Due to the sparseness of the p3 data set, the computation complexity is quite low, and allows us to use simple blind search.

Data set p2 is the same in size, but this one is not a sparse data set. Its theoretical complexity based on its size is the same as the complexity of p3, but the real complexity makes it impossible to be solved with simple blind search. In other words, our new algorithm based on formal concepts is the only way to compute exact BFA on p2 data set.

Table 1. Computation times

data set	factors	one's	time (m:s)	discrepancy	notes
p3.txt	5	2–4	61:36	0	375 combinations
p3.txt	5	3	0:12	0	120 combinations
p3.txt	5	1–10	0:00	0	8/10 concepts
p2.txt	2	6	11:44	743	54264 combinations
p2.txt	5	1–10	0:07	0	80/111 concepts
p2.txt	5	6–8	0:00	0	30/111 concepts

The results are shown in table 1. Data set p3 is rather simple, its factor loadings (particular rows of A) all contain just 3 one's. The first row in the table shows that it takes over 61 minutes to find these factors, when we search among all combinations with 2, 3 or 4 one's per factor. If we knew that there are just 3 one's per factor, we could specify it as a constraint, and got the result in just 12 seconds (see table 1, row 2). Indeed we usually don't know it in real situations.

Third row shows that when using formal concepts, we can find all factors in just 0 seconds, even when we search all possible combinations with 1 to 10 one's per factor. You can see the concept lattice in figure 1, with factors expressively circled.

Data set p2 is much more complex, because it is created from factors containing 6 one's each. In this case the blind-search algorithm was able to find just 2 factors. It took almost 12 minutes, and discrepancy was 743. In addition, the two found factors are wrong, which is not a surprise according to the fact that there are actually 5 factors, and factors can't be searched individually. It was

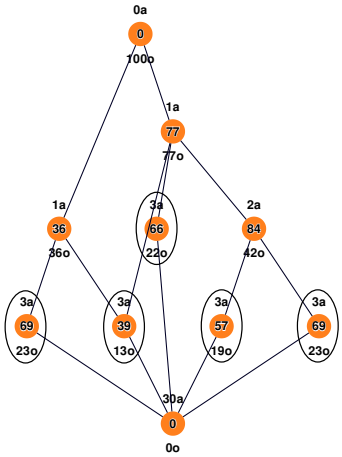

Fig. 1. Concept lattice of `p3` data set.

not possible to find more factors using blind-search algorithm. Estimated times for computing 3 to 5 factors with the same constraints (limiting number of one's per factor to 6) are shown in table 2. It shows that it would take up to 3.5×10^9 years to find all factors.

Table 2. Estimated computation times

data set	factors	one's	estimated time
p2.txt	3	6	440 days
p2.txt	4	6	65700 years
p2.txt	5	6	3.5×10^9 years

As you can see at the bottom of table 1, we can find all 5 factors of `p2` easily in just 7 seconds, searching among candidates containing 1 to 10 one's. The time can be reduced to 0 seconds once again, if we reduce searching to the range of 6 to 8 one's per factor. You can see the concept lattice in figure 2, with factors marked as well. As you can see, the factors are non-overlapping, i.e. they are not connected to each other. Note that in general factors can arbitrarily overlap.

7.2 Berry's data set

Berry's data set is an example of real data. It is a set of text documents describing occurrences of 18 words in 14 documents, i.e. X is 14×18 bits in size. We can't even try to find the factors with simple blind search, because it would take years to find just a few of them. On the other side, we can find 10 or more factors with help of formal concepts.

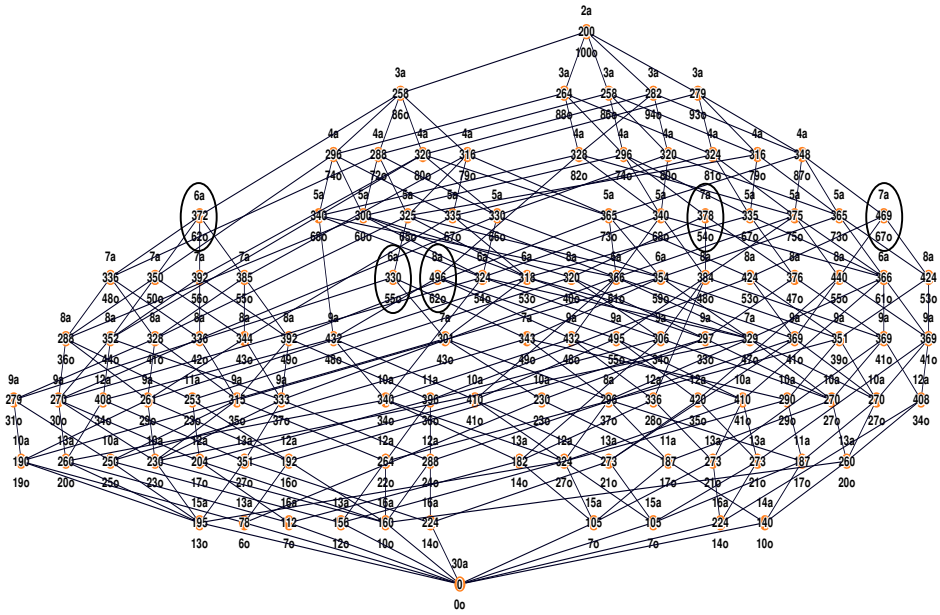

Fig. 2. Concept lattice of `p2` data set.

This data set is interesting in that it needs 14 factors to be successfully expressed as $\mathsf{F} \odot \mathsf{A}$. You can see all factors in figure 3, with all 14 factors circled.

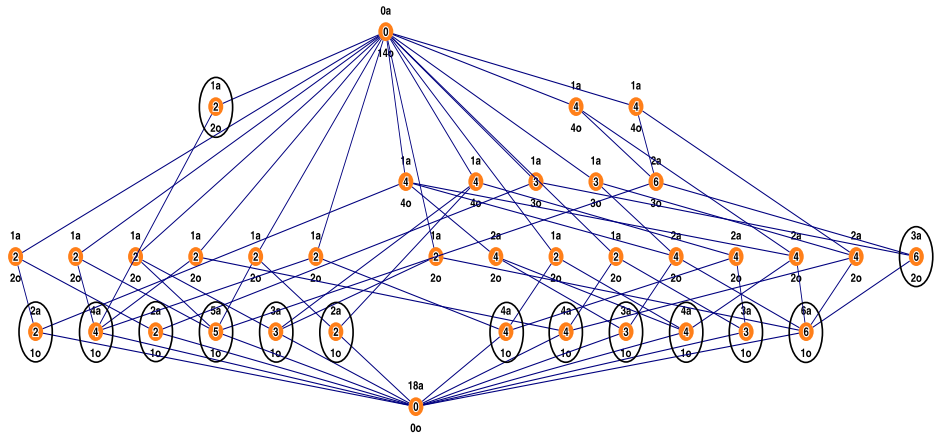

Fig. 3. Concept lattice of Berry's data set.

8 Conclusion

This paper presented an idea of exploiting theory of formal concept analysis to do a nonhierarchical analysis of binary data, namely binary factor analysis. The experiments revealed that this approach is a big step forward from the usual simple blind search.

Still, we don't understand this approach as a final step. It is quite fast on small data sets, but the complexity quickly increases as data set becomes larger. Formal concepts-based algorithm can be also used as a reference algorithm for testing the promising neural network-based solver (see [16], [7], [6], [2], [8]). For sure, the future work will more focus on the possibilities of exploiting formal concepts and concept lattices for BFA.

References

1. *BMDP (Bio-Medical Data Processing).* A statistical software package. SPSS. `http://www.spss.com/`
2. Frolov, A.A., Sirota, A.M., Húsek, D., Muraviev, I.P., Polyakov, P.A.: *Binary factorization in Hopfield-like neural networks: Single-step approximation and computer simulations.* 2003.
3. Ganter, B., Wille, R.: *Formal Concept Analysis: Mathematical Foundations.* Springer–Verlag, Berlin–Heidelberg–New York, 1999.
4. Geist, A., et al.: *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, Massachusetts, USA, 1994.
5. Hotho, A., Stumme, G.: *Conceptual Clustering of Text Clusters.* In Proceedings of FGML Workshop, pp. 37–45. Special Interest Group of German Informatics Society, 2002.
6. Húsek, D., Frolov, A.A., Muraviev, I., Řezanková, H., Snášel, V., Polyakov, P.: *Binary Factorization by Neural Autoassociator.* AIA Artifical Intelligence and Applications - IASTED International Conference, Benalmádena, Málaga, Spain, 2003.
7. Húsek, D., Frolov, A.A., Řezanková, H., Snášel, V.: *Application of Hopfield-like Neural Networks to Nonlinear Factorization.* Proceedings in Computational Statistics Compstat 2002, Humboldt-Universität, Berlin, Germany, 2002.
8. Húsek, D., Frolov, A.A., Řezanková, H., Snášel, V., Keprt, A.: *O jednom neuronovém přístupu k redukci dimenze.* In proceedings of Znalosti 2004, Brno, CZ, 2004. ISBN 80-248-0456-5.
9. Keprt, A.: *Paralelní řešení nelineární booleovské faktorizace.* VŠB Technical University, Ostrava (unpublished paper), 2003.
10. Keprt, A.: *Binary Factor Analysis and Image Compression Using Neural Networks.* In proceedings of Wofex 2003, Ostrava, 2003. ISBN 80-248-0106-X.
11. Keprt, A.: Using Blind Search and Formal Concepts for Binary Factor Analysis. In *Dateso 2004 - proceedings of 4th annual workshop.* Ed. Václav Snášel, Jaroslav Pokorný, Karel Richta, VŠB Technická Univerzita Ostrava, Czech Republic; CEUR WS – Deutsche Bibliothek, Aachen, Germany; **2004**, pp. 120-131, ISBN 80-248-0457-3 (VŠB TUO), ISSN 1613-0073 (CEUR).
12. Lindig, C.: *Introduction to Concept Analysis.* Hardvard University, Cambridge, Massachusetts, USA.

13. Lindig, C.: *Fast Concept Analysis.* Harvard University, Cambridge, Massachusetts, USA.
`http://www.st.cs.uni-sb.de/~lindig/papers/fast-ca/iccs-lindig.pdf`
14. *Medical encyclopedia Medline Plus.* A service of the U.S. National Library of Medicine and the National Institutes of Health.
`http://www.nlm.nih.gov/medlineplus/`
15. Schwarzweller, C.: *Introduction to Concept Lattices.* Journal Of Formalized Mathematics, volume 10, 1998. Inst. of Computer Science, University of Bialystok.
16. Sirota, A.M., Frolov, A.A., Húsek, D.: *Nonlinear Factorization in Sparsely Encoded Hopfield-like Neural Networks.* ESANN European Symposium on Artifical Neural Networks, Bruges, Belgium, 1999.
17. Řezanková, H., Húsek, D., Frolov, A.A.: Using Standard Statistical Procedures for Boolean Factorization. In proceedings of *SIS 2003.* Naples, Italy, 2003.
18. Karl Überla: *Faktorenanalyse* (2^{nd} edition). Springer–Verlag, Berlin–Heidelberg–New York, 1971. ISBN 3-540-04368-3, 0-387-04368-3.
(slovenský překlad: Alfa, Bratislava, 1974)