Concept Lattice algorithm implementation

J. Vásárhelyi*, L. Kovács**
* University of Miskolc, Department of Automation and Communication Technology, Miskolc, Hungary
** University of Miskolc, Department of, Miskolc, Hungary
vajo@mazsola.iit.uni-miskolc.hu, kovacs@iit.uni-miskolc.hu

***Abstract*—The increasing interest on application of concept lattices in the different information systems results in several implementations and algorithm proposals and representation tools. The concept lattice is mainly for representation of the concept generalization structure but it can apply as a classification tool too. A key component of practical applications is the efficient implementation of lattice building. This paper analyses the possibility of algorithm parallelization and implementation in hardware, which allow the speed up of the lattice construction and search for generated concepts.**

Keywords: concept lattice, lattice building, FPGA, parallel algorithms

I. INTRODUCTION

Concept lattices are applied in many application areas in industry to perform knowledge management tasks. The lattice structure represents conceptual hierarchy among the objects in the underlying problem domain. The field of Formal Concept Analysis [1] was born in the 80ies and it is now a powerful method in data analysis, information retrieval and knowledge discovery. In the literature, we can find several applications of concept lattices for data mining, especially for generating association rules [3]. One of the main characteristics of this application area is the large amount of structured data to be analysed. Another important application field is the area of production planning where the concept lattices are used to partition the products into disjoint groups during the optimisation of the production cost [11]. As the cost of building a concept lattice is a super-linear function of the corresponding context size, the efficient computing of concept lattices is a very important issue [12].

The building of a concept lattice consists of two, usually distinct phases. In the first phase, the set of concepts is generated. The lattice is built in the second phase from the generated set. We can find proposals in the literature for a combined optimisation of both phases and there are proposals addressing only one of the two phases.

Based on the analysis of these optimisation methods, the costs for the two phases are about the same order of magnitude and the common asymptotic cost depends in generally on three parameters: the number of objects, the number of attributes and the number of concepts. In the literature, there are two main variants for the concept set building algorithms. The methods of the first group work in batch mode, assuming that every element of the context table is already present before starting the concept lattice building. The main representative of this group is the Ganter's next closure method [1]. The other group of proposals uses an incremental lattice building method. In this case, the concept set is immediately updated when the context is extended with a new object. The method of Godin belongs to this group [2].

II. FORMAL CONCEPT ANALYSIS

The theory of concept lattice is based on the results of *Formal Concept Analysis*. A brief overview will be given in this section, a detailed description can be found among others in [1].

A *K context* is a triple *K (G, M, I)* where *G* and *M* are sets and *I* is a relation between *G* and *M*. The *G* is called the set of *objects* and *M* is the set of *attributes*. The cross table *T* of a context *K (G, M, I)* is the matrix form description of the *relation I*:

$$t_{ij} = \begin{cases} 1 \; if \; g_i I a_j \\ 0 \; otherwise \end{cases} \quad \textbf{(1)}$$

where $g_i \in G,\ a_j \in M$.

For $\forall A \subseteq G$, a *derivation operator* is defined as:

$$A' = \{ a \in M \mid g \; I \; a \text{ for } \forall g \in A \} \quad (1)$$

and for $\forall B \subseteq M$

$$B' = \{ g \in G \mid g \; I \; a \text{ for } \forall a \in B \} \quad (2)$$

The pair *C(A,B)* is a *concept* of the *K* context if

$$\begin{aligned} A &\subseteq G \\ B &\subseteq M \\ A' &= B \\ B' &= A \end{aligned} \quad (3)$$

hold true. In this case, *A* is called the *extent* and *B* is the *intent* of the *C* concept. It can be shown that for $\forall A_i \subseteq G$,

$$(\cup_i A_i)' = \cap_i A'_i \quad (4)$$

and similarly $\forall B_i \subseteq M$,

$$(\cup_i B_i)' = \cap_i B'_I \quad (5)$$

holds true.

Considering the Φ set of all concepts for the *K* context, an *ordering relation* can be introduced for the concept set in the following way:

$$C_1 \leq C_2 \text{ if } A_1 \subseteq A_2 \quad (6)$$

where C_1 and C_2 are arbitrary concepts. It can be proved that for every (C_1, C_2) pair of concepts, the following rules are valid:

$$(C_1 \wedge C_2 \in \Phi) \text{ and } (C_1 \vee C_2 \in \Phi). \quad (7)$$

Based on these features $(\Phi, \leq)$, is a lattice, called concept lattice. According to the Basic Theorem of concept lattices, $(\Phi, \leq)$ is a complete lattice, i.e. the infinum and suprenum exist for every set of concepts. The following rules hold true for every concept:

$$\vee_i (A_i, B_i) = (\cap_i A_i, (\cup_i B_i)'') \quad (8)$$
$$\wedge_i (A_i, B_i) = ((\cup_i A_i)'', \cap_i B_i)$$

where A'' denotes *the closure of set A* and it is defined as the derivation of the derived set:

$$A'' = (A')' \quad (9)$$

The structure of a concept lattice is usually represented with a *Hasse diagram*. The Hasse diagram is a special directed graph. The nodes of the diagram are the concepts and the edges correspond to the neighbourhood relationship among the concepts. If C_1, C_2 are concepts for which

$$C_1 < C_2$$
$$\neg\exists C_3 \in (\Phi, \leq) : C_1 < C_3 < C_2 \quad (10)$$

hold true then there is a directed edge between C_1, C_2 in the Hasse diagram. In this case, the C_1 and C_2 concepts are called *neighbour concepts*. C_1 is a lower neighbour of C_2 and C_2 is an upper neighbour of C_1.

The Hasse diagram of a concept lattice can be used not only to describe the concepts hidden in the underlying data system, but it shows the generalization relation among the objects, and it can be used for clustering purposes, too. A good description on the related chapters of the lattice theory can be found among others in [2].

III. Building Concept Lattice

The process of concept lattice building can be divided into two distinct phases. Initially, the set of concepts is generated from the given context. In the second phase, the lattice is built up from the generated set of concepts. In the literature, there is a large set of algorithms addressing only one of the two phases or covering both steps. There are also methods that combine these two phases into a single unit. Based on the analysis of these methods in the literature, the cost for both steps is about the same order of magnitude and the asymptotic cost depends on mainly three parameters: the number of objects, the number of attributes and the number of concepts. The cost is always larger than the product of these parameters.

Regarding the concept-set generation, there are two main variants of the available algorithms. The methods of the first group work in batch mode, assuming that every element of the context table is already present. The most widely known member of this group is the Ganter's next closure method. The other group of proposals is based on incremental building mode. In this case, the concept set is updated whenever the context is extended with a new object. The Godin's method belongs to this group. Regarding the phase for lattice building, the proposed approaches are based on the considerations that the lattice should be built up in a top-down (or bottom-up) manner because in this case only the elements of the upper (or lower) neighbourhood are to be localised. The second usual optimisation step is to reduce the set of lattice elements tested during the localisation of the nearest upper or lower neighbour elements.

The Godin's method uses an incremental lattice building approach. In this approach, if a new *g* object is added to the original *G* object set, the existing lattice is updated instead of generating the lattice from scratch. By adding a new g object to the context, the concept set and concept lattice are expanded usually by more than one new concept. These new concepts should be generated first and then they are inserted into the lattice. The key point in generation of new concepts is the fact that any new intent part should to be the result of intersecting the attribute part of the new object with some intent part already present in the lattice. Based on this feature, the intent part of any concept is equal to the intersection of the intent parts of the lower neighbour concepts. The intent part of each concept has to be a subset of the M attribute set, i.e. the generalized concepts are described by the same attribute set as the basic objects have.

The basic algorithm for updating the concept lattice can be summarized as follows:

1. cluster the concepts into buckets based on the cardinality of the intent part
2. take each bucket in ascending cardinality order
3. for each H concept in the bucket do
   if the intent part of H is not a subset of the intent part of X then
      new candidate pair is obtained by generating a new intersection from H, X edges of the new intersect result concept are generated
   end if
   end for

In the algorithm, X denotes the attribute part of the new object to be inserted into the context. Applying this kind of lattice generation method to objects having a large number of attributes, some new important problems will arise regarding the efficiency of the lattice building algorithm, and on the other hand the easy usage and interpretation of the resulting lattice. Considering efficiency, it is known that the cost for the Godin's method is [4]:

$$O(C^2NM) \quad (11)$$

where the following denotations are used:

C : number of concepts,

N : number of objects,

M : number of attributes.

According to the cost formula, the total cost value increases linearly with the intent part size of the concepts.

The other drawback of large attribute sets is that the large number of attributes may cause difficulties in understanding and in the interpretation of the resulted lattice. Humans prefer conciseness, i.e. a compact description. Instead of large detailed descriptions, short

compact expressions or concepts are used in the communication.

In the paper of Hu [3], the concept set generation process is coupled with the calculation of the support value in order to discover association rules from the concept lattice. The concept set building part is based on the incremental method of Godin, thus resulting the same asymptotic calculation cost estimation value:

$$O(N\sigma + CNDM). \quad (12)$$

Another proposal is the Titanic algorithm, presented in [14]. This method uses the support values of the different attribute sets to determine the concept intents. It generates the candidate generator sets in increasing order of the size. A set is called a generator set if its closure is a concept intent and it is minimal, i.e. it does not contain any other generators for the same concept intent. The method processes first the one-attribute-long candidates and after then generates the candidate sets for the next level.

The proposal of Lindig given in [13], is aimed at not only the generation of the concept set but on the building of the whole concept lattice. If we consider now only the concept set generation part of the algorithm, this method is related to the Ganter's method in many aspects. It assumes a lexical ordering among the concepts and the concepts are processed according to this ordering. The method also generates for every new concept the set of upper neighbour concepts to use this kind of information during the insertion into the concept lattice.

The neighbours of a concept are generated using the closure operation for the candidate neighbour attribute sets. At every call of the neighbour routine the full context table is scanned. The cost estimation of this algorithm is

$$O(Nc\sigma + CN^2M) \quad (13)$$

Thus the asymptotic complexity is the same as for the Ganter's method.

One of the largest problems in hardware or software implementation of concept lattices is the large number of attributes. Most of the proposals in the literature cope with this problem with elimination of the attributes with low relevance value. Although, these algorithms can reduce the number of attributes, providing better efficiency and interpretation, the resulted lattice cannot be treated as the most optimal one. According to our considerations, this solution may yield in some kind of information lost. This reasoning is based on two elements. First, the information lost is caused by the fact that the parent concepts will contain only some selected attributes of the children and the selected attributes are not always the best to describe the object. Second, during the attribute reduction phase, the meaning of the eliminated attributes will be lost, providing less information in the intersected concept. Let's take an example to demonstrate the described effect.

Example 1. If there are four documents as objects with the following attributes: D1(London, football), D2(London, tennis), D3(Paris, tennis) and D4 (Berlin, swimming) then the possible intersections of the attribute parts will result in only two documents: D5(London) and D6(tennis). The generated lattice is shown in Figure. 1.

In this result lattice, a great part of the information about the document topics was lost, as there were only few common attributes in the original documents. According to the generated lattice, there are no common in D3 and D4. On the other hand, a human could find some common elements in these two documents, for example, both refer to sports or to European capitals.

To improve the quality and usability of the resulting lattice, a modified lattice and concept description form was developed which is described in the next section in details.

IV. Extended Attribute Management

It is assumed that there exists a lattice containing the attributes from the objects. This lattice can be considered as a thesaurus with the generalization relationship among the attributes. Taking the documents as objects and the words as attributes in our example, the attribute lattice shows the specialization and generalization among the different words [9]. In special cases, the lattice may be a single hierarchy. It is also possible to take several disjoint lattices as they can be merged into a new common lattice. Using this attribute lattice, the usual lattice-building operators are redefined to generate a more compact and semantically more powerful concept lattice.

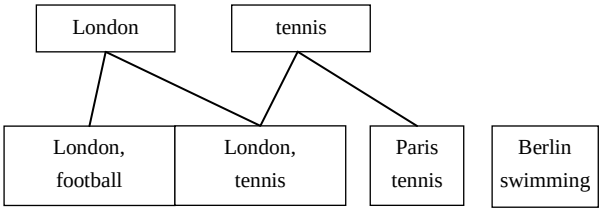

Figure 1. Concept Lattice Example

The proposed lattice construction algorithm is intended for information systems with a relative narrow problem area. In this case, an attribute lattice can be generated within an acceptable time and effort. It is assumed that the attribute lattice contains only those attributes that are relevant for the problem area in question. In this case, the size of the attribute lattice and the intent part of the concepts will be manageable. According to this assumption, the first phase of the document processing is the attribute filtering when the attributes not present in the attribute lattice are eliminated from the intent parts.

The attribute lattice is a subset of the M attribute set. This lattice is denoted by the symbol $\Omega(M, \leq)$. The role of the lattice is to represent the general – special relationship among the attributes. The ordering relation of the attribute lattice is defined in the following way:

For $\forall m_1, m_2 \in M$ and $m_1 > m_2$ if m_1 is a generalization of m_2. Based upon the relationship in $\Omega(M, \leq)$ a redefined subset or partial ordering relation is introduced. This new relation is denoted by $\leq^*$ and it is defined in the following way for $\forall m_1, m_2 \in M$:

$$m_1 \leq^* m_2 \Leftrightarrow m_1 \text{ is an ancestor of } m_2 \text{ in } \Omega(M, \leq),$$
$$\text{i.e. } m1 \text{ is a generalization of } m2\ (m_1 \geq m_2) \quad (14)$$
$$\text{based on the } \Omega(M, \leq) \text{ lattice.}$$

Taking the words as attributes, for example, the word *animal* is a generalization of the word *dog*, so *animal* $\leq^*$ *dog* relation is met.

According to the lattice features, there exists a set of nearest common upper neighbours for any arbitrary pairs of attributes. This set is denoted by $LCA(m_1,m_2)$. For the attribute pair m_1, m_2 we have:

$$LCA(m_1,m_2) = \{m \in M \mid m \leq^* m_1 \text{ and } m \leq^* m_2 \text{ and } \nexists\, m\text{'}: m\text{'} \leq^* m_1 \text{ and } m\text{'} \leq^* m_2 \text{ and } m \leq^* m\text{'}\} \quad (15)$$

The *LCA* denotes the least common ancestor of two nodes in the lattice. The *LCA* set contains exactly the leaf elements of the common ancestor lattice for m_1 and m_2. Based on the partial ordering among the attributes, a similar $\leq^*$ ordering can be defined among the attribute sets. For $\forall\, B_1, B_2 \subseteq M$ the $\subseteq^*$ ordering relation is given as follows:

$$B_1 \subseteq B_2 \;\Leftrightarrow\; \exists f: B_1 \rightarrow B_2 \text{ function} \;\Rightarrow x \leq^* f(x)\ for\ \forall x \in B_1 \quad (16)$$

It is easy to see that the normal subset relation is a special case of the $\subseteq^*$ relation, i.e.:

$$B_1 \subseteq B_2 \Longrightarrow B_1 \subseteq^* B_2 \quad (17)$$

In this case the f: $x \rightarrow x$ mapping can be used to show the correctness of the $\subseteq^*$ relation.

Based on this kind of subset relation, a new intersection operation can be defined. The definition of the new operator is:

$$B = B_1 \cap^* B_2 = \cup\, LCA(m_1, m_2 | m_1 \in B_1, m_2 \in B_2) \quad (18)$$

The intersection operator results in a set containing the nearest common generalizations of the attributes in the operand sets. If the parent node for every normal attribute of the intent sets is the null attribute (which is equivalent to the case when no attribute lattice is defined), the new $\cap^*$ intersection operator will yield in the same result as the standard $\cap$ intersection operator. This is due to the fact that in this case

$$LCA(m_1, m_2) = m\ if\ m_1 = m_2 = m,\ \emptyset\ otherwise \quad (19)$$

Using this kind of subset and intersection operators instead of the usual subset and intersection operators during the concept set and concept lattice building phases, the resulting lattice will be more compact, more readable and manageable than the base concept lattice.

V. Optimization of Lattice Building

On important option for optimization of algorithms is the parallelization of the execution. In the case of concept lattice building, there are some computation phases which are suitable for such parallelization. One of the first parallelization methods is the ParGal [15] method. It works on the fact that the computations of sub-nodes of a given node are independent tasks. The major difficulty of the algorithm is to manage the huge set of concepts already generated. This set is to be tested for elimination of duplicates if a new candidate concept is generated. In the ParGal model, a separate process performs this time consuming task.

In our approach, the focus is set for the first processing phase, when all existing concepts generated already should be intersected with the intent part of the new object (A_o). The intersection

$$A_o \cap A_i \quad (20)$$

can be executed parallel for the different A_i intent sets as they are totally independent from each other's. These sets are the new candidate intension concepts sets. In the next step, the redundancy should be eliminated. This means that the repeated values are removed from the result set. In this phase, the intent part of the generated intersection should be compared with the intent parts of the already generated elements. To perform this step in parallel execution a hash-table approach was implemented. In this case, there is a corresponding hash table with a suitable hash function. The parallel threads can insert the generated elements into the table on an efficient way. The elements with the same intent part are located within the same bucket of the hash table. Thus, the checking of collision can be executed locally, within a bucket. To achieve a better load balancing, the different buckets can be managed by different threads, thus the buckets can be processed parallel. The process of intersection and process for removal of duplicate values can be executed in the same hash table.

VI. Hardware Consideration

The electronically stored information amount increased exponentially since the introduction of the internet. To find the corresponding information, which match the search criteria are a question of time and the speed of the software implemented search engines.

The massive amount of date handled by a document clustering system requires high performance computing. Microprocessors systems are inefficient in handling a large amount of attributes, as they perform these algorithms sequentially. The previously presented concept lattice algorithm has several computation steps that can run in parallel. Parallel algorithms are best suited for implementation in hardware. Field Programmable Gate Arrays (FPGAs) can implement these parallel algorithms. However, there are some limitations in the implementation of these algorithms, which result from the hardware resources. Nevertheless, a concept lattice implemented in hardware as a co-processing engine can speed up document search even if FPGAs are working with lower frequencies than processors. In addition, FPGAs have the advantage against Application Specific Integrated Circuits (ASIC) the possibility of dynamic reconfiguration.

In the literature there is mentioned a high-speed document clustering, which use reconfigurable hardware [10]. In the mentioned paper Covington et all. relate about a full hardware implementation of the K-means clustering algorithm implemented in reconfigurable hardware that clusters 512k documents rapidly. This implementation uses four parallel cosine distance metrics to cluster document vectors.

To demonstrate the operation of concept lattice in hardware, the previously presented example (Figure 1.) was synthesized in FPGA circuit using VHDL-specified modules. These modules were then used to implement

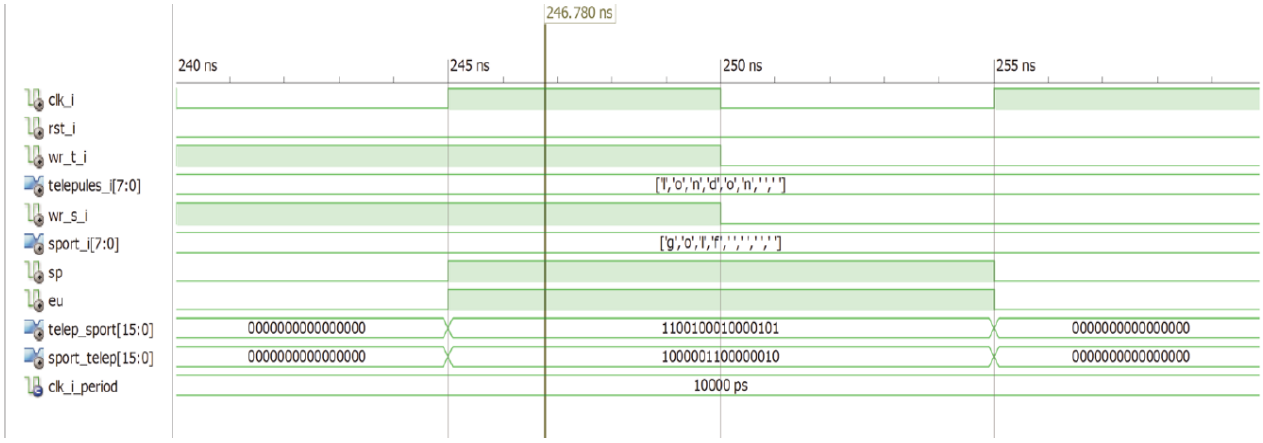

Figure 2. Simulation result of the attribute lattice

logic in a Spartan 3E FPGA. The hardware implemented concept lattice is intended to be a co-processing element of a NLM system.

The simulation result presented in Figure 2. shows the final result of the node (London, sport) in the following interpretation:

The value of output variable “city-sport” (telep_sport) represents the concept node in a 16 bit representation. The meaning of the bits is as follows:

bit 15: city is in Europe
bit 14: city is Capital
bit 13..8 city code
bit 7 there is sport in the city
bit 6..0 sport code

In this interpretation, we have a city in Europe, named London, having sports tennis and football. The simulation also contains another variable “sport-city” (sport_telep) that shows the pairs of sports-city. In this interpretation the lattice attribute sport (in this case golf) it is the characteristic of the node Miskolc (Europe, not capital).

The attribute lattice simulated was implemented as mentioned before in a Xilinx Spartan 3e FPGA. The working frequency of the PCB board is 50MHz (the simulation performed at the frequency 100MHz).

The device utilization summary presented in Table I. shows that the parallel implementation of the algorithm consumed relatively low resources. These resources are mainly utilised for the implementation of the algorithm, and only few resources for the database storage, since the database contain only a few elements. Certainly in a real implementation when the data amount and the number of attributes is high then the hardware needed for implementation is higher.

The presented and implemented example gives us the confirmation that parallel algorithms can be used for implementation of concept lattice.

The next step is to create a real concept with at least ten thousand of data. The concept lattice co-processing element will be implemented in National Instruments RIO card.

The implementation should be characterized as follows:

- has to be extensible if the number of lattice elements increase;
- the construction of concept lattice start from the universe (“11…11”) then each new element which is different than the existing element in the concepts (as result of cut between the lattice vectors and the existing elements of the concept);

TABLE I.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of 4 input LUTs	97	9,312	1%
Number of occupied Slices	53	4,656	1%
Number of Slices containing only related logic	53	53	100%
Number of Slices containing unrelated logic	0	53	0%
Total Number of 4 input LUTs	98	9,312	1%
Number used as logic	97		
Number used as a route-thru	1		

Number of bonded IOBs	166	232	71%
IOB Flip Flops	22		
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	1.73		

- each new element introduced increase dynamically the concept dimension.
- searching in the concept set should be parallel as much as possible. If the amount of data does not permit intensive parallel search, then the data is loaded in sub sets of the concept.

VII. CONCLUSIONS

The parallel implementation of concept lattice resulted in speedup of algorithm execution. Development of a concept lattice co-processor is possible.

The construction should allow some dynamic data change, which is limited by the hardware resources.

ACKNOWLEDGMENT

This Research was carried out as part of the TAMOP-4.2.1.B-10/2/KONV-2010-0001 project with support by the European Union, co-financed by the European Social Fund.

REFERENCES

[1] B. Ganter, "Finger Excersises in Fromal Concept Analysis", Dresden ICCL Summer School, June/July 2006 http://www.math.tu-resden.de/~ganter/psfiles/FingerExercises.pdf

[2] R. Godin, R. Missaoui, H. Alaoui, "*Incremental Concept Formation Algorithms Based On Galois (Concept) Lattices"*, Computational Intelligence, 11 DOI:10.1111/j.1467-8640.1995.tb00031.x, 1995, pp. 246–267.

[3] K. Hu, Y. Lu, C. Shi: "Incremental concept formation algorithms based on Galois lattices", *Proceedings of PAKDD99*, Beijing,1999, pp.109-113.

[4] Sheng-Yong Qiao, Shuo-Pin Wen, Cai-Yun Chen, Zhi-Guo Li, "A Fast Algorithm for Building Concept Lattice" IEEE International Conference on Machine Learning and Cybernetics, ISBN: 0-7803-8131-92003, DOI: 10.1109/ICMLC.2003.1264463, 2003, pp. 163-167.

[5] http://en.wikipedia.org/wiki/Formal_concept_analysis

[6] http://www.upriss.org.uk/fca/

[7] Villerd J., Ranwez S., Crampes M., Carteret D., "Using Concept Lattice for Visual Navigation Assistance in Large Databases: Application to a Patent Database." in CLA 2007, http://www.informatik.uni-trier.de/~ley/db/conf/cla/cla2007.html

[8] Tsopzé N., Nguifo E. M., Tindo G.,Clann, "Concept Lattice-based Artifcial Neural Network for supervised classification", in CLA 2007, http://www.informatik.uni-trier.de/~ley/db/conf/cla/cla2007.html

[9] L. Kovács "Concept Lattice Structure with Attribute Lattice"

[10] G. Adam Covington, Charles L.G. Comstock, Andrew A. Levine, John W. Lockwood, Young H. Cho „High Speed Document Clustering In Reconfigurable Hardware" *Proceedings of IEEE FPL* 2006, 28-30 Aug. 2006, paper 189,2006, pp.1-7.

[11] S. Radeleczki, T. Tóth: *Fogalomhálók alkalmazása a csoporttechnológiában*, OTKA kutatási jelentés, Miskolc, Hungary, 2001

[12] L. Nourine., O. Raynaoud: *A Fast Algorithm for Building Lattices*, Information Processing Letters, 71, 1999, p. 197-210

[13] C. Lindig: *Fast Concept Analysis*, Proceedings of the 8th ICCS, Darmstadt, 2000.

[14] G. Stumme., R. Taouil, Y. Bastide, N. Pasquier., L. Lakhal: *Fast Computation of Concept Lattices Using Data Mining Techniques*, 7th International Workshop on Knowlegde Representation meets Databases (KRDB 2000), Berlin, 2000.

[15] P. Njiwoua, E.M. Nguifo: A Parallel Algorithm to build Concept Lattice, Proc. Groningen Int. Inf. Techn Conf., 1997, pp. 103-107