Production Systems and Information Engineering
Volume 4 (2006), pp. 65–79

CONCEPT LATTICE STRUCTURE WITH ATTRIBUTE LATTICES

LÁSZLÓ KOVÁCS
University of Miskolc, Hungary
Department of Information Technology
kovacs@iit.uni-miskolc.hu

[Received May 2005 and Acceped March 2006]

Abstract. There is an increasing interest on application of concept lattices in the different information systems. The concept lattice may be used for representation of the concept generalisation structure generated from the underlying data set. The paper presents a modified lattice building algorithm where the generated concept nodes may contain not only the attributes of the children nodes but some other generalised attributes, too. The generalisation structure of the attributes is called attribute lattice. Using this kind of lattice building mechanism, the generated lattice and cluster nodes are more natural and readable for humans. The proposed lattice structure can be used in several kinds of information system applications to improve the quality of the query interface.

Keywords: concept lattice, lattice building

1. Standard Concept Lattice

Concept lattices are used in many application areas to represent conceptual hierarchies among the objects in the underlying data. The field of Formal Concept Analysis [1] introduced in the early 80ies has grown to a powerful theory for data analysis, information retrieval and knowledge discovery. There is nowadays an increasing interest in the application 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. A technical oriented application field of Formal Concept Analysis is the area of production planing where the concept lattices are used to partition the products into disjoint groups during the optimisation of the production cost [6]. 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, has been investigated over the last decades [5].

This section gives only a brief overview of the basic notations of the theory for Formal Concept Analysis. For a more detailed description, it is referred to [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:

$$t_{i,j} = \begin{cases} 1 & \text{if } g_i I a_j \\ 0 & \text{otherwise} \end{cases}$$

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

Two Galois connection operators are defined. For every $A \subset G$:

$$f(A) = A' = \{a \in M | \forall g \in A \quad gIa\} \tag{1.1}$$

and for every $B \subset M$

$$g(B) = B' = \{g \in G | \forall a \in B \quad gIa\} \tag{1.2}$$

The Galois closure operator is defined by

$$h = f \circ g$$

and

$$A'' = h(A)$$

is the Galois closure of A. The pair $C(A, B)$ is a closed itemset of the K context if

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

hold. In this case A is called the extent and B is the intent of the C closed itemset. It can be shown that for every $A_i \subseteq G$,

$$(\cup_i A_i)' = \cap_i A_i'$$

and similarly for every $B_i \subseteq M$,

$$(\cup_i B_i)' = \cap_i B_i'$$

holds.

Considering the ϕ set of all concepts for the K context, an ordering relation can be introduced for the set of closed itemsets in the following way:

$$C_1 \leq C_2$$

if

$$A_1 \subseteq A_2$$

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

$$C_1 \wedge C_2 \in \phi$$

and

$$C_1 \vee C_2 \in \phi.$$

Based on these features $(\phi, \wedge)$ is a lattice, called closed itemset lattice. According to the Basic Theorem of closed itemset lattices, $(\phi, \wedge)$ is a complete lattice, i.e. the infimum and supremum exist for every set of closed itemsets. The following rules hold for every family $(A_i, B_i), i \in I$ of concepts:

$$\vee_{i \in I}(A_i, B_i) = (\cap_{i \in I} A_i, (\cup_{i \in I} B_i)'')$$

$$\wedge_{i \in I}(A_i, B_i) = ((\cup_{i \in I} A_i)'', \cap_{i \in I} B_i)$$

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

$$C_1 < C_2$$

$$\neg \exists C_3 \in (\phi, \leq) \quad C_1 < C_3 < C_2$$

holds 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.

There are several approaches in the literature to provide an efficiency lattice management. Each of the proposals provides a mechanism to reduce the number of attributes. These methods are usually based on some kind of statistical calculations. The method presented in [11] uses the principal component analysis to eliminate the redundant attributes from the documents. This method is based on the consideration that the occurrences of some attributes may be correlated. According to the principal component analysis, the original m correlated random variables can be replaced by another set of n un-correlated variables where n is smaller than m. The resulting variables are the linear combination of the original variables. The principal components depend solely on the covariance matrix of the original variables.

Another kind of statistical computation is required if the reduction is based on the relevance values of the attributes. The relevance value of an attribute denotes how important the attribute is in the given object. This relevance value is calculated usually by the 'tf5idf' weighting method. This method defines the relevance value in proportion to the number of occurrences of the attribute in the document f_{ij}, and in inverse proportion to the number of documents in the collection for which the term occurs at least once (n_i):

$$rel_{ij} = f_{ij} \quad log(\frac{N}{n_i})$$

The attributes having smaller relevance value than a threshold are eliminated from the object descriptor set. This kind of reduction method is used for example in [19].

Although the mentioned algorithms can reduce the number of attributes, providing better efficiency and interpretation, the resulted lattice can not be treated as the 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.

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.

2. Concept Lattice with Attribute Lattice

It is assumed that there exists a lattice-like structure containing the attributes from the objects. This lattice-like structure 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. 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 re-defined to generate a more compact and semantically more powerful concept lattice.

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 M' elementset of 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 generalization - specialization relationship among the attributes. The ordering relation of the attribute lattice is defined in the following way. For any m_1, m_2 attributes in M', m_1 is greater than m_2 ($m_1 \geq m_2$) if m_1 is a generalization of m_2. Based upon the relationship in $\Omega(M', \leq)$ a redefined partial ordering relation is introduced to M as an extension of the $\leq$ relation. This new relation is denoted by $\leq *$ and it is defined in the following way for any $m_1, m_2 \in M$:

$$m_1 \leq *m_2 \Leftrightarrow m_1 \text{ is an ancestor, a generalization of } m_2 \text{ in } \Omega(M', \leq)$$

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 neighbors for any arbitrary pairs of attributes. This set is denoted by $LCA(m_1, m_2)$ for the attribute pair m_1, m_2.

$$LCA(m_1, m_2) =$$
$$\{m \in M | m \leq *m_1 \wedge m \leq *m_2 \wedge \neg\exists m' \quad m' \leq *m_2 \wedge m' \leq *m_1 \wedge m \leq *m'\} \tag{2.1}$$

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 ordering can be defined among the attribute sets. For any $B_1, B_2 \subseteq M$, the $\subseteq *$ ordering relation is given as follows:

$$B_1 \subseteq *B_2 \Leftrightarrow \exists f \quad B_1 \rightarrow B_2 \text{ function so that } x \leq *f(x) \text{ for every } x \in B_1.$$

Having four sets of words B1(Paris, tennis, cup), B2(capital, sport), B3(capital, sport, car) and B4(sport) the $B_2 \subseteq *B_1$ relation is true as the $f \quad capital \rightarrow Paris, sport \rightarrow tennis$ function is a good injection. On the other hand, $B_3 \subseteq *B1$ relation is false, as the word 'car' can not be mapped to any word in B_1.

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

$$B_1 \subseteq B_2 \Rightarrow B_1 \subseteq *B_2$$

In this case the $f \quad x \rightarrow x$ 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:

$$B = B1 \cap *B2 = \cup \{LCA(m_1, m_2) | m_1 \in B_1, m_2 \in B_2\} \tag{2.2}$$

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

$$LCA(m_1, m_2) = \begin{cases} m & if m_1 = m_2 = m \\ \oslash & otherwise. \end{cases}$$

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 standard concept lattice. This effect will be achieved by involving attributes into the concept description that would not be present if the standard lattice building method was used.

3. Algorithms for the LCA operation

The key operation in the proposed lattice management is the determination of the LCA set for any arbitrary pair of nodes. This operation is performed several times during the execution of the $\cap*$ intersection operations. As the intersection is a frequent operation the efficiency of the LCA generation is a key element in the efficiency of the whole lattice management.

The computation of the LCA set can be performed basically on two different ways. In the first family of proposals, the common ancestor nodes are located by traversing the paths connecting the two operand-nodes. To reduce the number of candidate paths, the shortest path is determined first. The shortest path is usually calculated by using matrix multiplication. The second group of approaches for determining the LCA elements is based on the labeling concept. In the labeling approach, every vertex is assigned a description string. This label is used not only for identifying the nodes but to represent the ordering relationship among the nodes. In this case, the parents of an arbitrary node can be determined from the labels without the edge descriptions. Beside the problem of LCA generation, the labeling methods are used also to determine the distance between two nodes. This kind of labeling is called a distance labeling [14].

If the lattice is degenerated to a linear structure, the LCA contains only one node and this LCA node can be determined with a linear processing cost. Harel and Tarjan [15] showed first that the tree-LCA can be generated in a

$$O(n)$$

linear preprocessing time. In the last decades, some new proposals were published having the same $O(n)$ execution cost but using a much more simpler algorithm. One of most recent ones among these proposals is the paper of Bender, Colton and Pemmasani[15]. They present an extremely simple, optimal tree-LCA algorithm. It is shown that the tree LCA problem is equivalent to the RMP, the range minimum problem. In the paper, an $O(n)$ RMP algorithm is presented first and then it is converted to a tree-LCA algorithm with $O(n)$ efficiency. An intensive investigated topic in this area is the identifying the nearest common ancestors in dynamic trees. In [16], Alstup presents a pointer machine algorithm which performs n link and m LCA operations in time

$$O(n + m \quad loglogn).$$

The main problem of the algorithms based on path traversing is that the relationship among the nodes of the lattice must be stored explicitly. These relationships are usually described by matrixes or by pointers. In this case, the tree is stored by representing explicitly all vertices and all edges. To save the storage space and to improve the execution efficiency, labeling methods are implemented.

In [17] a simple algorithm is presented that labels the nodes of a rooted tree such that from the labels of two nodes alone one can compute in constant time the label of their nearest common ancestor. The labels assigned to the nodes are of size

$$O(logn)$$

and the labeling algorithm runs in

$$O(n)$$

time. A similar result for this problem can be found among others in [18].

In the case of a lattice or DAG (directed acyclic graph), the LCA problem requires much more computation. In a lattice structure, two nodes may have several LCA nodes. Although the DAG is a widely used structure, the DAG-LCA generation is not so widely investigated as the tree-LCA problem. Based on the work of Bender, Colton and Pemmasani[15], the main results can be summarized as follows. For testing the existence of common ancestors, an ancestor existence matrix is built. Two nodes x and y in lattice G have a common ancestor if and only if (x', y) is in the transitive closure set of the G'' lattice. The G'' lattice is generated by merging the sinks of G' with the sources of G. The G' is the inverse lattice of G, i.e. it contains the same number of nodes and edges but every edge has the inverse direction. The ancestor existence matrix can be computed in

$$O(n \quad w)$$

time, where w is about 2.376 [15] and $O(nw)$ is equal to the efficiency value of the fastest matrix multiplication algorithm. The transitive closure of a lattice can be generated within the $O(nw)$ efficiency class, too. The computation of the LCA set is based on the consideration that the shortest path in the G'' DAG from node x' to node y goes through the LCA of the corresponding nodes. The generation of LCA for a pair of nodes can be calculated in

$$O(\frac{n \quad w}{2} - 0.5)$$

time.

There exist some proposals for finding the LCA nodes in graph by labeling method, too. A k-step labeling method is presented in the paper [16] of Talamo and Vocca. A k-step labeling consists of $f_1, .., f_k$ functions where every f_i is a partial function computable in one step and a composition between f_i and f_{i-1} can be defined. The k-step labeling is a valid labeling if and only if

$$y \in adj(x) \Leftrightarrow (f_k \circ f_{k-1} \circ \quad \circ f_1(x,y)) = y \vee (f_k \circ f_{k-1} \circ \quad \circ f_1(y,x)) = x$$

is met. The paper presents a method for generating the labels where a vertex x has a

$$O(\delta(x) \cdot log^2 n)$$

bit long label and the labels can be computed in

$$O(\delta \cdot n^2)$$

time, where δ denotes the degree of the vertex. The degree of a vertex is equal to the number of adjacent nodes. A modified version of this adjacent labeling can be found among others in [16]. An $L(d,1)$ labeling is a function f that assigns to each vertex a non-negative integer such that if two vertices x and y are adjacent then $|f(x) - f(y)| > d$, and if x and y are not adjacent but there is a two-edge path between them then $|f(x) - f(y)| > 1$.

Considering our requirements, it can be seen, neither of the proposals meets all of the requirements. The main problems in application of the presented methods are the followings:

there is no detailed study on DAG-LCA problem
the path traversing method for DAG [15] retrieves only one element from the LCA
the labeling methods can be used only to determine the adjacent nodes and not all the descendant nodes

Based on these restrictions, it seems reasonable to develop a special DAG-LCA generation algorithm for the lattice structure.

4. A modified DAG-LCA algorithm

The computation of the LCA set can be performed basically on two different ways. In the first family of proposals, the common ancestor nodes are located by traversing the paths connecting the two operand-nodes. To reduce the number of candidate paths, the shortest path is determined first. The shortest path is usually calculated by using matrix multiplication. The second group of approaches for determining the LCA elements is based on the labeling concept. In the labeling approach, every vertex is assigned a description string.

In the case of our search algorithm for finding the LCA nodes, a merging of the path- oriented methods with the labeling methods is implemented. The main idea is to assign a description set to every node in the lattice where this description set has a similar role as the attribute set has in the normal concept lattices. As it is known, there is a strong correlation between the position in the lattice and the content of the intent part. For any pairs of concepts, the concepts are in relation if and only if one of the intent parts is a subset of the other intent part:

$$C_1 \leq C_2 \Leftrightarrow A_2 \subseteq A_1$$

Based on this rule it can be seen that

$$A(LCA(m_1, m_2)) \subseteq A(m_1) \cap A(m_2)$$

also holds where $A(m)$ denotes the intent part of m. In this sense, the search for the LCA nodes may be restricted to the nodes where the intent part is a subset of the intersection of the corresponding A_i and A_j sets. This reduction may increase the efficiency of finding the LCA elements. As a node in the attribute lattice usually does not contain an intent part description, it is not possible to apply this kind of reduction element in the usual lattice building. To include this optimization feature an appropriate intent part should be added to every node of the attribute lattice.

Let B denote the set of binary lists having the same length and containing 0 and 1 elements. If there exists a

$$a \quad M \rightarrow B$$

function which meets the following requirements:

$$a(m_1) = a(m_2) \Leftrightarrow m_1 = m_2$$

$$a(m_1) \subseteq a(m_2) \Leftrightarrow m_1 \geq m_2$$

then

$$a(LCA(m_1, m_2)) \subseteq a(m_1) \cap a(m_2) \tag{4.1}$$

holds also. In these expressions the $\subseteq$ symbol denotes the sub-list operator and the $\cap$ operator is the list intersection. The list-intersection is defined for any l_1, l_2 lists as follows

$$(l_1 \cap l_2)_j = l_{1j} \wedge l_{2j}$$

where the length of the result list is equal to the minimum of the operands lengths. Thus, for example the intersection of 101100 and 111000 is equal to 101000.

This statement can be easily verified as according to (2.1)

$$LCA(m_1, m_2) \geq m_1 \; and \; LCA(m_1, m_2) \geq m_2$$

thus

$$a(LCA(m_1, m_2)) \subseteq a(m_1) \; and \; a(LCA(m_1, m_2)) \subseteq a(m_2)$$

and so

$$a(LCA(m_1, m_2)) \subseteq a(m_1) \cap a(m_2)$$

holds.

To provide an appropriate $a()$ function, the following algorithm is used to calculate the $a(m)$ values. First, the nodes in the lattice are sorted by the depth value. The nodes with low depth value are processed first. Thus before processing of node m, every ancestor of m has been processed already. The root node of the lattice is assigned to an empty list. This root element is the only node with a zero depth value. If all the nodes with depth value less than K are already processed, then the $a()$ values for nodes at depth level $K+1$ are calculated according to the following algorithm.

1. For every m at level $K+1$:
   $a(m) = \cup_{m' \in Parent(m)} a(m')$
2. Nodes having the same $a(m)$ value are extended with tail tags to ensure the uniqueness of the $a(m)$ values.
3. Testing every node at the processed levels. If node m' is not an ancestor of m and $a(m') \subseteq a(m)$ then $a(m')$ is extended with tail tag.
4. The descendants of m' are adjusted to the new m' value.

Lemma. The $a()$ function generated by the given algorithm meets the (4.1) conditions.

Proof. According to the step 2 in the algorithm, every node will have a unique value. In the adjustment phase every processed node will be modified with an unique tag, so the uniqueness of the $a()$ values is ensured in this phase too. According to this considerations, the

$$a(m_1) = a(m_2) \Leftrightarrow m_1 = m_2 \tag{4.2}$$

condition holds.

If the m is a child of m' then $a(m') \subseteq a(m)$. This comes from the fact that the $a(m)$ is generated as the union of all its parents. If $m' \geq m$ then exists a list of parent-child relationship from m to m' Using the transitive property of the relations, it follows that

$$a(m_1) \subseteq a(m_2) \Leftarrow m_1 \geq m_2 \tag{4.3}$$

is met. On the other hand if m' is not greater or equal to m then m' is not an ancestor of m, then the $a()$ value of m' is modified by adding new tags to the list value. After this modification, the $a(m')$ will not be a subset of $a(m)$. Thus

$$\neg a(m_1) \subseteq a(m_2) \Leftarrow \neg m_1 \geq m_2 \tag{4.4}$$

According to the (4.2), (4.3), (4.4) formulas, the (4.1) property is met.

Considering the proposed labeling algorithm, the generated labels are usually not optimal from the viewpoint of the label length. In the tests, the labels were generated for the normal concept nodes having a natural attribute string. Depending on the number of nodes and on the depth of the lattice, the generated labels can be several times longer than the original attribute labels. In the test runs, the proposed labeling algorithm provided always the same lattice relationship among the nodes as the original attribute strings. The length optimality of the generated labels is a topic for further investigations.

After generating the labels, the next step is the identification of the LCA set. In the basic path oriented methods the LCA algorithm consists of the following steps:

1. Generating the A_x ancestor set for x. The ancestors are selected by traversing along the parent-child edges.
2. Generating the A_y ancestor set for y.
3. Calculating the A_{xy} intersection of the two ancestor sets.
4. Selection of vertices in A_{xy} having no descendants in A_{xy}

The cost for the LCA algorithm can be given by

$$O(A_x \quad f \cdot e + A_y \quad f \cdot e + A_{xy} \cdot w)$$

where

f average degree of the vertices, i.e. the average number of parents

e cost for selection of an edge related to a given node. The cost may vary depending on the storage method.

A_x size of the corresponding ancestor set

On the other hand, in the proposed algorithm the generation of the LCA for (x, y) is performed in the following steps.

1. Processing the parents of x in a recursive way
2. If the current element is an ancestor of y, insert the current element into list L and stop the ancestor lookup

3. Selecting elements of L having no descendants in L

In step 2, the ancestor relationship is tested by comparison of the label values. According to (4.3), if

$$a(m_1) \subseteq a(m_2)$$

then m_1 is an ancestor of m_2. If the traversing reaches a y-ancestor, the lookup can be stopped as the ancestors of this node can not be LCA nodes.

The main benefit of this algorithm is the reduced number of nodes to be processed. The cost can be given by

$$O(A'_x \quad f \quad (e+h) + \eta A'^2_{xy})$$

where
A'_x the number of vertices being the ancestor of x
but not being an ancestor of y.
η the cost for comparing two labels
A'_{xy} the number of selected border nodes in A_{xy}.

Comparing the two cost expressions, we can see that the combined method is more efficient than the basic method if

1. A'_x is smaller than A_x and A_y
2. η and e have the same magnitude
3. A'_{xy} is smaller than A_{xy}

Based on these considerations, this bottom up traversing is advantageous if the LCA elements are located near to the x and y nodes. On the other hand, if the LCA elements are near to the root of the lattice, a top-down approach provides a better solution. In this case, the algorithm is the following:

1. Selecting the root of the lattice
2. Testing the children of the current node
3. If the label is a subset of the intersection label
4. Selection of vertices in A_{xy} having no descendants in A_{xy}

This algorithm determines the parents for the intersection of x and y. The label of the intersection node is equal to the intersection of the corresponding labels. The cost value can be given by

$$O(A_{xy} \quad f \quad (e+h))$$

where f denotes the average number of children vertices.

An efficient implementation can involve all of the mentioned algorithms. The LCA generation program should include a decision module that is responsible for selecting an appropriate algorithm. As the number of 1 digits in the label is correlated with the level of the node, an approximation of the LCA levels can be given based on the value of the intersection label. The heuristic rule can be summarized as follows: If the number of 1 digits is low in the intersection label, then a top-down traversing method is used, otherwise a bottom-up traversing is applied.

REFERENCES

[1] S. ABITEBOUL AND H. KAPLAN AND T. MILO: Compact labeling schemes for ancestor queries, *Technical report*, INRIA, 2001

[2] S. ALSTUP AND T. RAUHE: Improved labeling scheme for ancestor queries, *Technical report*, University of Copenhagen, 2001

[3] S. ALSTUP AND C. GAVIOLLE AND H. KAPLAN AND T. RAUHE: Identifying Nearest Common Ancestors in Distributed Enviroment, *Technical report*, University of Copenhagen, 2001

[4] M. BENDER AND M. COLTON AND G. PEMMASANI: Least Common Ancestors in Trees and Directed Acyclic Graphs, *Symposium on Discrete Algorithms*, 2001, pp. 845-854

[5] G. CHANG AND W KE AND D. KUO AND D. LIU AND R. YEH: On L(d,1)-labelings of graphs, *Discrete Mathematics*, Volume 220, Issues 1-3, 6 June 2000, pp. 57-66

[6] B. GANTER AND R. WILLE: *Formal Concept Analysis, Mathematical Foundations*, Springer Verlag, 1999

[7] R. GODIN AND R. MISSAOUI AND H. ALAOUI: Incremental concept formation algorithms based on Galois lattices, *Computational Intelligence*, 11(2), 1995, 246-267

[8] K. HU AND Y LU AND C SHI: Incremental Discovering Association Rules: A Concept Lattice Approach, *Proceedings of PAKDD99, Beijing*, 1999, 109-113

[9] M. KATZ AND N KATZ AND D. PELEG: Distance labeling schemes for well-separated graph classes, *STACS 2000, Lecture Notes In Computer Science*, Springer Verlag, 2000

[10] L. KOVACS: Efficiency Analysis of Building Concept Lattice, *Proceedings of 2nd ISHR on Computational Intelligence*, Budapest, 2001

[11] L. KOVACS: A Fast Algorithm for Building Concept Set, *Proceedings of MicroCAD2002*, Miskolc, Hungary 2002

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

[13] L. NOURINE AND O. RAYNAUD: A Fast Algorithm for Building Lattices, *Information Processing Letters*, 71, 1999, 197-210

[14] S. RADELECZKI AND T. TÓTH: Fogalomhálók alkalmazása a csoporttechnológiában, *OTKA kutatási jelentés*, Miskolc, Hungary, 2001.

[15] J. SILVA AND J. MEXIA AND A. COELHO AND G. LOPEZ: Document Clustering and Cluster Topic Extraction in Multilingual Corpora, *Proc. of the 2001 IEEE Int. Conference on Data Mining*, IEEE Computer Society, pp. 513-520

[16] G. STUMME AND R. TAOUIL AND Y BASTIDE AND N. PASQUIER AND L. LAKHAL: Fast Computation of Concept Lattices Using Data Mining Techniques, *Proc. of 7th International Workshop on Knowlegde Representation meets Databases (KRDB 2000)*, Berlin, 2000

[17] M. TALAMO AND P VOCCA: Representing graphs implicitly using almost optimal space, *Discrete Applied Mathematics*, Elsevier Publ., 2001, pp. 193-210

[18] D. TIKK AND J. YANG AND S. BANG Text categorization using fuzzy relational thesauri, *Technical report*, Chonbuk National University, Chonju, Korea, 2001

[19] M. ZAKI AND M. OGIHARA: Theoretical Foundations of Association Rules, *Proceedings of 3 rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'98)*, Seattle, Washington, USA, June 1998.

[20] U. ZWICK: All Pairs Shortest Path in weighted directed graphs- exact and almost exact algorithms, *IEEE Symposium on Foundation of Computer Science*, 1998, pp. 310-319