Concept Lattice Structure with Attribute Lattice

University of Miskolc, Dept. of Information Technology

László Kovács

Dept. of Information Technology, University of Miskolc

kovacs@iit.uni-miskolc.hu

***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.

1. Introduction

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].

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. 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.

1.2 Standard Concept Lattice

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*:

$$t_{ij} = \begin{array}{ll} 1 & , \text{ if } g_i I a_j \\ 0 & \text{otherwise} \end{array} \quad (1)$$

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

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

$$A' = \{ a \in M \mid g \, I \, a \text{ for } \forall \, g \in A \} \quad (2)$$

and for every $B \subseteq M$

$$B' = \{ g \in G \mid g \, I \, a \text{ for } \forall \, a \in B \} \quad (3)$$

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

$$\begin{array}{l} - A \subseteq G \\ - B \subseteq M \\ - A' = B \\ - B' = A \end{array} \quad (4)$$

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 every $A_i \subseteq G$,

$$(\cup_i A_i)' = \cap_i A'_i \quad (5)$$

and similarly for every $B\text{i} \subseteq M$,

$$(\cup_i B_i)' = \cap_i B'_I \tag{6}$$

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:

$$C_1 \leq C_2 \tag{7}$$

if

$$A_1 \subseteq A_2$$

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:

$$C_1 \wedge C_2 \in \Phi \tag{8}$$

and

$$C_1 \vee C_2 \in \Phi.$$

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:

$$\begin{aligned} \vee_i (A_i, B_i) &= (\cap_i A_i, (\cup_i B_i)'') \\ \wedge_i (A_i, B_i) &= ((\cup_i A_i)'', \cap_i B_i) \end{aligned} \tag{9}$$

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

$$A'' = (A')' \tag{10}$$

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

$$\begin{aligned} &C_1 < C_2 \\ &\neg\exists C_3 \in (\Phi, \leq) : C_1 < C_3 < C_2 \end{aligned} \tag{11}$$

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].

One of the largest problems in 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 rrelevance value. Altough, these algorithms can reduce the number of attributes, providing better efficiency and interpretation, the resulted lattice can not 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.

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 Fig. 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.

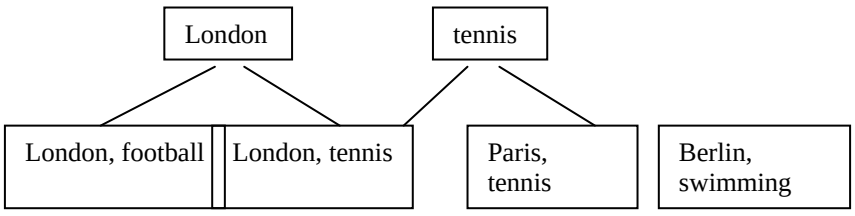

Fig. 1, Document lattice

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 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. 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 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 subset or partial ordering relation is introduced. This new relation is denoted by $\leq^*$ and it is defined in the following way for any $m_1, m_2 \in M$:

$$m_1 \leq^* m_2 \Leftrightarrow m_1 \text{ is an ancestor of } m_2 \text{ in } \Omega\,(M, \leq), \text{ i.e. } m1 \text{ is a generalization of } m2 \quad (m_1 \geq m_2) \text{ based on the } \Omega\,(M, \leq) \text{ lattice.} \tag{12}$$

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.

$$LCA(m_1,m_2) = \{m \in M \mid m \leq^* m_1 \text{ and } m \leq^* m_2 \text{ and not exists } m' : m' \leq^* m_1 \text{ and } m' \leq^* m_2 \text{ and } m \leq^* m'\} \tag{13}$$

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 any $B_1, B_2 \subseteq M$, the $\subseteq^*$ ordering relation is given as follows:

$$B_1 \subseteq^* B_2 \Leftrightarrow \exists f : B_1 \rightarrow B_2 \text{ function so that } x \leq^* f(x) \text{ for every } x \text{ in } B_1. \quad (14)$$

Having four sets of words B_1\{Paris, tennis, cup\}, B_2\{capital, sport\}, B_3\{capital, sport, car\} and B_4\{sport\} the $B_2 \subseteq^* B_1$ relation is true as the f: {capital $\rightarrow$ Paris, sport $\rightarrow$ tennis} function is a good injection. On the other hand, $B_3 \subseteq^* B_1$ relation is false, as the word car can not be mapped to any word in B_1. In the example, the $B_4 \subseteq^* B_1, B_2 \subseteq^* B_3$ relations are also valid.

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

$$B_1 \subseteq B_2 \Rightarrow B_1 \subseteq^* B_2 \quad (15)$$

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:

$$B = B_1 \cap^* B_2 = \cup LCA(m_1,m_2 \mid m_1 \in B_1, m_2 \in B_2) \quad (16)$$

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

$$LCA(m_1,m_2) = m \text{ if } m_1 = m_2 = m \quad (17)$$
$$\varnothing \text{ otherwise.}$$

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. Let's demonstrate this on a simple example mentioned previously. The objects in Figure 1 will be used again extended with an attribute lattice. The attribute lattice has the following structure:

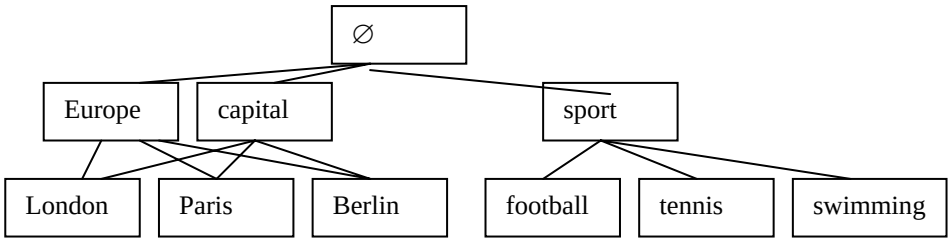

Fig. 2, Attribute lattice

The four objects to be processed are D1{London, football}, D2{London, tennis}, D3{Paris, tennis} and D4{Berlin, swimming}. The new nodes of the concept lattice are generated using the new intersection operator. The objects are processed in the index order. The resulting intent parts are the following:

D1 ∩* D2 => DA{London, sport}
D1 ∩* D3 => DB{Europa, capital, sport}
D2 ∩* D3 => DC{Europa, capital, tennis}
DA ∩* D3 => DB{Europa, capital, sport}
D1 ∩* D4 => DB{Europa, capital, sport}
D2 ∩* D4 => DB{Europa, capital, sport}
D3 ∩* D4 => DB{Europa, capital, sport}
DA ∩* D4 => DB{Europa, capital, sport}
DB ∩* D4 => DB{Europa, capital, sport}
DC ∩* D4 => DB{Europa, capital, sport}

After determining the neighborhood relationship among the concepts, the next step is the building of the document lattice. The resulting lattice will have the following structure:

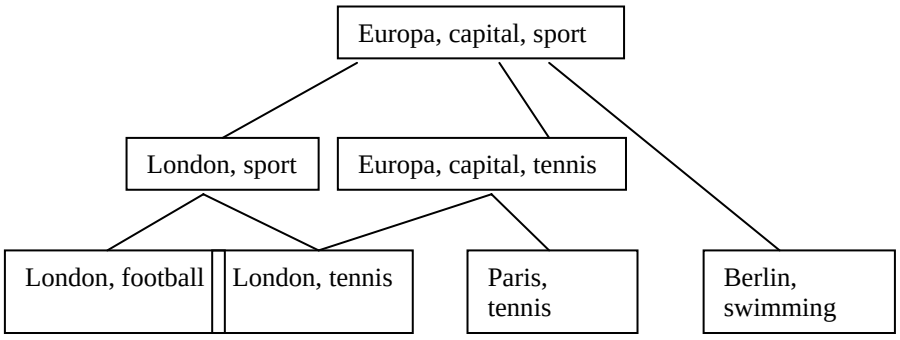

Fig. 3

Comparing this lattice structure with the lattice given in Fig 1, it can be seen that this lattice has a larger descriptive power and it stores more information about the content of the base objects.

Considering the case when the attribute lattice is complex and large, the $\cap^*$ intersection may result in a large intent part containing a lot of attributes. This will cause degradation in efficiency and in readability. To avoid this undesired effect, the size of the intent part should be reduced. One possibility for reduction is to realize that not all *LCA* elements are of equal importance or relevance. Some of them are very far from the base attributes and others may be very close to them. In this sense the attribute set belonging to the intersection set may be filtered using a weight value of the resulting attributes. In this approach the weight value is the composition of two factors:

$$w(a) = w_f(a)*w_d(a). \tag{18}$$

where $w_f(a)$ denotes the frequency component of the a attribute. As an attribute may be contained in several *LCA*'s of different a_i and a_j pairs, the frequency of the attributes in the intersection set may be different. $W_f(a)$ is monotone increasing function of the frequency, i.e. the larger the frequency is the larger the relevance of the attribute is. The second $w_d(a)$ function denotes the distance component of the weight value. The $w_d(a)$ is a monotone decreasing function of the distance value between a and the base attributes (a_i, a_j). Thus the attributes that are close to each other may have larger chance to delegate elements into the intersection set than the attributes that are located far from each other. Using the $w(a)$ value the intersection will contain only those elements which have a greater weight value than a τ threshold value:

$$B = B_1 \cap^* B_2 = \{m \mid m \in \cup LUB(m_1,m_2 \mid m_1 \in B_1, m_2 \in B_2) \text{ and } w(m) > \tau \} \tag{19}$$

Depending on the selected τ value the intersection yields in a set containing larger or smaller number of attributes. As the cost function of the lattice building process is a linear function in the number of attributes in the intent part, the τ value determines the efficiency of the algorithm too.

3. A modified DAG-LCA algorithm

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].

In the case of 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

$$O(n^w) \tag{20}$$

time, where *w* is about 2.376 [15] and $O(n^w)$ is equal to the efficiency value of the fastest matrix multiplication algorithm. The transitive closure of a lattice can be generated within the $O(n^w)$ 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

$$O(n^{w/2-0.5}) \tag{21}$$

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

$$y \in adj(x) \Leftrightarrow (f_k f_{k-1} \circ ... f_1(x,y)) = y \text{ or } (f_k f_{k-1} \circ ... f_1(y,x)) = x \tag{22}$$

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

$$O(\delta(x)\ log^2 n) \tag{23}$$

bit long label and the labels can be computed in

$$O(\delta\, n^2) \tag{24}$$

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 $|f(x)-f(y)| > 1$.

In the proposed version of the 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 node 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:

$$C_1 \leq C_2 \ \Leftrightarrow A_1 \subseteq A_2 \tag{25}$$

Based on this rule it can be seen that

$$A(LCA(m_1,m_2)) \subseteq A(m_1) \cap A(m_2) \tag{26}$$

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 Aj 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

$$a: M \rightarrow B \tag{27}$$

function which meets the following requirements:

$$a(m_1) = a(m_2) \Leftrightarrow m_1 = m_2 \tag{28}$$

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

then

$$a(LCA(m_1,m_2)) \subseteq a(m_1) \cap a(m_2) \tag{29}$$

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

$$(l_1 \cap l_2)_j = l_{1j} \wedge l_{2j} \tag{30}$$

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.

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

   $$a(m) = \cup_{m' \in P(m')} a(m')$$

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 (28) 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

$$a(m_1) = a(m_2) \Leftrightarrow m_1 = m_2 \tag{31}$$

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

$$a(m_1) \subseteq a(m_2) \Leftarrow m_1 \geq m_2 \tag{32}$$

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

$$\neg\ a(m_1) \subseteq a(m_2) \Leftarrow \neg m_1 \geq m_2 \tag{33}$$

According to the (25), (26), (27) formulas, the (28) 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

$$O(A_x f \varepsilon + A_y f \varepsilon + A_{xy}{}^{w)} \tag{34}$$

where

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

ε : 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 (32), if

$$a(m_1) \subseteq a(m_2) \tag{35}$$

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

$$O(A'_x f(\varepsilon+\eta) + \eta A'^{2}_{xy}) \tag{36}$$

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 ε 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

$$O(A_{xy} f(\varepsilon + \eta)) \tag{37}$$

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] B. Ganter and R. Wille: Formal Concept Analysis, *Mathematical Foundations*, Springer Verlag, 1999

[2] R. Godin and R. Missaoui and H. Alaoui: Incremental concept formation algorithms based on Galois lattices, *Computational Intelligence*, 11(2), 1995, 246-267

[3] K. Hu and Y. Lu and C Shi: Incremental Discovering Association Rules: A Concept Lattice Approach, *Proceedings of PAKDD99*, Beijing, 1999, 109-113

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

[5] L. Nourine and O. Raynaud: A Fast Algorithm for Building Lattices, *Information Processing Letters*, 71, 1999, 197-210

[6] S. Radeleczki and T. Tóth, Fogalomhálók alkalmazása a csoporttechnológiában, *Kutatási jelentés, OTKA kutatási jelentés*, Miskolc, Hungary, 2001.

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

[8] 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.

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

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

[11] 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

[12] 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

[13] 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

[14] 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

[15] 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

[16] M. Talamo and P. Vocca: Representing graphs implicitly using almost optimal space, *Discrete Applied Mathematics*, Elsevier Publ., 2001, pp. 193-210

[16] S. Alstup and T. Rauhe: Improved labeling scheme for ancestor queries, *Technical report, University of Copenhagen*, 2001

[17] S. Alstup and C. Gaviolle and H. Kaplan and T. Rauhe: Identifying Nearest Common Ancestors in Distributed Enviroment, *Technical report, University of Copenhagen*, 2001

[18] S. Abiteboul and H. Kaplan and T. Milo: Compact labeling schemes for ancestor queries, *Technical report, INRIA*, 2001

[19] D. Tikk and J. Yang and S. Bang: Text categorization using fuzzy relational thesauri, *Technical report, Chonbuk National University*, Chonju, Korea, 2001