Parallel Algorithms for Concept Lattice

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 [1], [2] and [6]. The concept lattice may be used for representation of the concept generalization structure generated from the underlying data set. This paper analyses the possibility of algorithm parallelisation and implementation in hardware, which allow the speed up of the search and database build.

I. INTRODUCTION

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. One 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 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. 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 Concept lattice theory is presented also in [2] There is presented on the following this description and for a more detailed description see [8].

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

\[
C_1 < C_2 \quad \text{and} \quad \not\exists C_3 \in (\Phi, \leq) : C_1 < C_3 < C_2
\]

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 \( C_1 \).

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.** 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.

Figure 1. Concept Lattice Example

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.
III. 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. Using this attribute lattice, the usual lattice-building operators are redefined to generate a more compact and semantically more powerful concept lattice.

In this case, an attribute lattice can be generated within an acceptable time and effort and easily can be implemented in hardware. 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 $m > m_1$ 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 $\forall m_1, m_2 \in M$:

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

i.e. $m_1$ is a generalization of $m_2$ ($m_1 \leq^* m_2$) (2)

based on the $\Omega (M, \leq)$ lattice.

Taking the words as attributes, for example, the word setlement is a generalization of the word city, so settlement $\leq^*$ city 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 $m_1, m_2$ one have:

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

and $\exists m^* : m^* \leq^* m_1 \text{ and } m^* \leq^* m_2$ (3)

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 $\leq^*$ ordering relation is given as follows:

$$B_1 \subseteq^* B_2 \Leftrightarrow \exists f : B_1 \rightarrow B_2 \text{ function (4)}$$

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

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

$$B = B_1 \cap^* B_2 = \bigcup LCA (m_1, m_2)$$

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

$$LCA (m_1, m_2) = m \text{ if } m_1 = m_2 = m, \emptyset \text{ otherwise (6)}$$

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 attribute lattice has the structure presented in Figure 2, which result from Figure 1. The four objects to be processed are:

- $D_1 \{ \text{London, football} \}$
- $D_2 \{ \text{London, tennis} \}$
- $D_3 \{ \text{Paris, tennis} \}$
- $D_4 \{ \text{Berlin, swimming} \}$

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:

- $D_1 \cap^* D_2 \Rightarrow DA \{ \text{London, sport} \}$
- $D_1 \cap^* D_3 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
- $D_2 \cap^* D_3 \Rightarrow DC \{ \text{Europe, capital, tennis} \}$
- $DA \cap^* D_3 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
- $D_1 \cap^* D_4 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
- $D_2 \cap^* D_4 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
- $D_3 \cap^* D_4 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
- $DA \cap^* D_4 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
- $DB \cap^* D_4 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
- $DC \cap^* D_4 \Rightarrow DB \{ \text{Europe, capital, sport} \}$
After determining the neighbourhood relationship among the concepts, the next step is the building of the document lattice (see Figure 3.)

Comparing this lattice structure with the lattice given in Figure 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 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 3.) was synthesized in FPGA circuit using VHDL-specified modules. These modules were then used to implement 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 4. 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, one 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.
V. CONCLUSIONS

The parallel implementation of concept lattice resulted in speedup of algorithm execution. Development of a concept lattice co-processor unit is possible and future work will concentrate on implementing a co-processing element for more complex problems.

The speedup of the hardware implementation is clear, since the result are given in one clock cycle. Compared to software implementation where several instructions are executed to obtain the same result.

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


[8] L. Kovács “Concept Lattice Structure with Attribute Lattice”