Soft Comput (2014) 18:683–694
DOI 10.1007/s00500-013-1176-6

FOCUS

A new FCA algorithm enabling analyzing of complex and dynamic data sets

Petr Gajdoš · Václav Snášel

Published online: 23 November 2013

Abstract Analyzing data with the use of Formal Concept Analysis (FCA) enables complex insights into hidden relationships between objects and features in a studied system. Several improvements in this research area, such as Fuzzy FCA or L-Fuzzy Concepts, bring the possibility to analyze data with a certain rate of indeterminacy. However, the usage of FCA on larger complex data brings several problems relating to the time-complexities of FCA algorithms and the size of generated concept lattices. The fuzzyfication of FCA emphasizes the mentioned problems. This article describes significant improvements of a selected FCA algorithm. The primary focus was given on the system of an effective data storage. The binary data was stored with the use of finite automata that leads to the lower memory consumption. Moreover, the better querying performance was achieved. Next, we focused on the inner process of the computation of all formal concepts. All improvements were integrated into a new FCA algorithm that can be used to analyze more complex data sets.

Communicated by I. Zelinka.

This article has been elaborated in the framework of the IT4Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 funded by Structural Funds of the European Union and the state budget of the Czech Republic. The work is partially supported by Grant of SGS No. SP2013/70, VŠB - Technical University of Ostrava, The Czech Republic. This work was also supported by the Bio-Inspired Methods: research, development and knowledge transfer project, reg. no. CZ.1.07/2.3.00/20.0073 funded by the Operational Programme Education for Competitiveness, co-financed by ESF and the state budget of the Czech Republic.

P. Gajdoš (✉) · V. Snášel
Department of Computer Science, VŠB-Technical University of Ostrava, 17. listopadu 15, Ostrava, The Czech Republic
e-mail: petr.gajdos@vsb.cz

V. Snášel
e-mail: vaclav.snasel@vsb.cz

Keywords Formal concept analysis · Tree data structure · Finite automata · Fuzzy data

1 Introduction

The usage of Formal Concept Analysis (FCA) over large data collections, also large incidence matrices, brings a range of specific problems. The first of them is connected with concept computation. The time complexity grows very fast with the size of input set of objects and attributes. A problem with the storage of computed concepts is associated with it. Other problems came up in the case of lattice visualization. Usually, humans are not able to read complex graph structures that contain tens or hundreds of vertices. There are two possible ways to solve this problem. First, the visualization methods can be improved, e.g., by using colored graphs, zooming, traversing the graph structure, etc. The second way is to reduce the input data at the cost of some acceptable noise. Such an approach is well known in information retrieval; e.g. Boolean Factor Analysis by Attractor Neural Network (Frolov et al. 2007), Formal Concept Analysis Constrained by Attribute-Dependency Formulas (Bělohlávek and Sklenář 2005), or Ordinal Factor Analysis (Ganter and Glodeanu 2012). Nevertheless, both ways require a fast approach in the computation phase to ensure acceptable computation times in the case of analyzing complex data. We developed a new algorithm to be able to use FCA to analyze the data with a certain rate of indeterminacy, dynamic or fuzzy data.

After a brief introduction to FCA, the existing FCA algorithms are introduced and a selected Valtchev's algorithm is described in more detail. Then there are mentioned the proposed improvements. Firstly, the data storage based on finite automata and fast querying into the incidence matrices are described. The effective data storage decreases the

memory consumption as well. Next, we focused on the inner process of the computation of all formal concepts. The new proposed algorithm efficiently utilizes a tree structure and a tree traversing mechanism. Several experiments illustrate the achieved result.

2 Formal concept analysis

Formal concept analysis (FCA) is a data analysis technique that describes the world in terms of objects and the attributes possessed by those objects. The philosophical starting point for FCA was represented by the understanding that a *concept* can be described by its *extension* and *intension*. The mathematical foundations were laid by Birkhoff (1967) who demonstrated the correspondence between partial orders and lattices. Birkhoff showed that a lattice can be constructed for every binary relation between a set of objects and a set of attributes with the resulting lattice providing insight into the structure of the original relation.

Formal concept analysis arose during the early 1980s from Rudolf Wille's (2009) pioneer work. It identifies conceptual structures among data sets based on the primary philosophical understanding of a "concept" as a unit of thought comprising its extension and intension as a way of modelling a domain (Wille 2009; Ganter and Wille 1999). The extension of a concept is formed by all objects to which the concept applies and the intension consists of all attributes possessed by those objects. These generate a conceptual hierarchy of a domain by finding all possible formal concepts which reflect a certain relationship between attributes and objects.

The FCA method has been successfully applied to a wide range of applications in medicine (Cole and Eklund 1996), psychology (Spangenberg et al. 1999), ecology (Brüggemann et al. 1997), software engineering (Snelting 2000) or information science (Eklund et al. 2000). A variety of methods for data analysis and knowledge discovery in databases have also been proposed based on the FCA techniques (Stumme and Wille 1998; Wille 2001). Information Retrieval is also a typical application area of FCA (Godin et al. 1993; Priss 2000; Kaytoue et al. 2011; Galitsky and de la Rosa 2011; Li et al. 2012). Some researches focused on the algebraic approach and tried to improve the theoretical background behind FCA, (e.g. Wang et al. 2010; Medina and Ojeda-Aciego 2010).

2.1 The most important definitions from formal concept analysis

As mentioned above, FCA is a way of describing the world in terms of objects and the attributes possessed by those objects. This section introduces the FCA notation and conventions used throughout this article. All definitions written by Ganter and Wille (1999) are assumed and cited in this section. In other cases, all definitions are labeled according to the reference.

$\mathbb{K}$	a	b	c	d	e	f
1	×	×	×	×		×
2	×	×	×		×	
3	×	×	×			
4		×	×	×	×	
5	×	×				
6						×

Fig. 1 An example of the incidence matrix showing existing relationships between objects {1,2,3,4,5,6} and their attributes {a,b,c,d,e,f}

Definition 1 A **formal context** $\mathbb{K} := (G, M, I)$ consists of two sets G and M and a relation I between G and M. Elements of G are called objects and elements of M are called attributes of the context[1]. In order to express that an object g is in a relation I with an attribute m, we write (gIm) or $(g, m) \in I$ and read "the object g has the attribute m".

The relation I is also called the *incidence relation* of the context. All relations between objects and attributes could be written in a table, also called an *incidence matrix* that is illustrated in Fig. 1.

A set of common attributes and a set of common objects have to be defined before the definition of a formal concept.

Definition 2 For a set $A \subset G$ of objects we define

$$A' = \{m \in M \mid gIm \text{ for all } g \in A\} \tag{1}$$

-the set of attributes common to the objects in A. Similarly, for a set $B \subset M$ of attributes we define

$$B'\{g \in G \mid gIm \text{ for all } m \in B\} \tag{2}$$

-the set of objects which have all attributes in B.

Definition 3 A **Formal concept** of the context $\mathbb{K} := (G, M, I)$ (or $\mathcal{L} := (G, M, I)$) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. We call A the *extent* and B the *intent* of the concept (A, B). The set of all concepts of a context $\mathbb{K}$ is denoted $\mathfrak{B}(G, M, I)$.

A particular concept is represented as a disconnected rectangle in the incidence matrix. Figure 2 shows some concepts.

The whole theory is very extensive, that is why we refer to the work of Ganter and Wille (1999) for a detailed description of the FCA terms. We also refer to Ganter and Wille (1999) for the basic definition of ordered sets. The Definition 4 connects the Theory of Ordered Sets with the algebraic structure.

[1] More precisely, *formal objects* a *formal attributes* of the context.

$\mathbb{K}$	a	b	c	d	e	f
1	×	×	×	×		×
2	×	×	×		×	
3	×	×	×			
4		×	×	×	×	
5	×	×				
6						×

C_0:
extent {1, 2, 3}
intent {a, b, c}

C_1:
extent {1, 6}
intent {f}

Fig. 2 This figure shows two concepts of a context

Definition 4 An ordered set $\mathbf{V} := (V, \leq)$ is a **lattice**, if for any two elements' set {x,y}, where $x \in V$, $y \in V$, the supremum $x \vee y$ and the infimum $x \wedge y$ always exist. V is called a **complete lattice**, if the supremum $\bigvee X$ and the infimum $\bigwedge X$ exist for any subset X of V.

Theorem 1 *The concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice in which infimum and supremum are given by:*

$$\bigwedge_{t\in T}(A_t, B_t) = \left(\bigcap_{t\in T} A_t, \left(\bigcup_{t\in T} B_t\right)''\right)$$

$$\bigvee_{t\in T}(A_t, B_t) = \left(\left(\bigcup_{t\in T} A_t\right)'', \bigcap_{t\in T} B_t\right),$$

where T is an index set and, for every $t \in T$, $A_t \subseteq G$ and $B_t \subseteq M$.

The concept lattice represents the relations between formal concepts and is it used as a basis for the further visualization. Usually, all practical usages of formal concept analysis need some kind of visual representation, because it is more readable than a simple list or table. The concept lattice can be represented by a Hasse diagram as in Example 1(b).

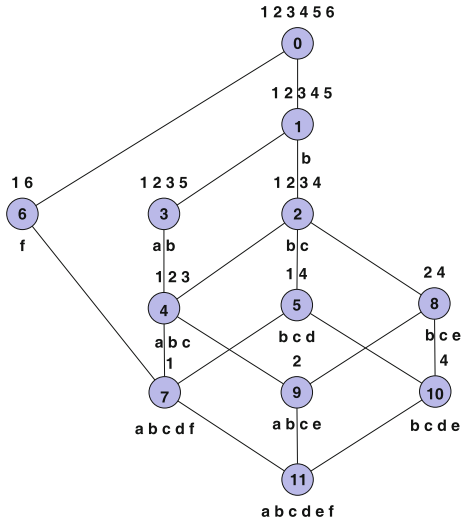
Example 1 Context, concept and lattice. This example refers to Fig. 1. There is the incidence matrix that defines six objects and their features. A computed list of all possible concepts is shown in Fig. 3(a) as well as related concept lattice visualized by the Hasse diagram (b).

3 FCA algorithms—the state of the art

The following sections bring to the reader some information on known approaches of concepts computation. The algorithm classes are described in more detail. The improvement of a selected algorithm and related experiments are presented in a separate section.

index	extent	intent
0	1 2 3 4 5 6	∅
1	1 2 3 4 5	b
2	1 2 3 4	b c
3	1 2 3 5	a b
4	1 2 3	a b c
5	1 4	b c d
6	1 6	f
7	1	a b c d f
8	2 4	b c e
9	2	a b c e
10	4	b c d e
11	∅	a b c d e f

(a) A list of all concepts



(b) Concept lattice

Fig. 3 See the Incidence matrix of an illustrative example (Fig. 1). This figure shows a computed list of all concepts (**a**) and a related concept lattice (**b**). The concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is given by the set of objects $G = \{1, 2, 3, 4, 5, 6\}$, the set of features $M = \{a, b, c, d, e, f\}$ and $I \subset [G, M]$ is the relation illustrated in the Fig. 1

Computing a concept lattice is an important issue and has been widely studied in order to develop more efficient algorithms. As a consequence, a number of batch algorithms (Chain 1969; Ganter 1984; Bordat 1986; Ganter and Reuter 1991; Kuznetsov 1993; Lindig 1999) and incremental algorithms (Norris 1978; Downling 1993; Godin et al. 1995; Carpineto and Romano 1996; Ganter and Kuznetzov 1998; Nourine and Raynaud 1999; Stumme 2000; Valtchev and Missaoui 2001; Krajca et al. 2010; Andrews 2011) are mentioned. Batch algorithms build formal concepts and a concept lattice from the whole context in a bottom–up approach (from the maximal extent or intent to the minimal one) or a top–down approach (from the minimal extent or intent to the maximal one). Incremental algorithms gradually reformulate the concept lattice starting from a single object with its attribute set.

Table 1 The list of most important algorithms

Author (year)	Time complexity
Norris (1978)	$O(\|G\|^2 \|M\| \|L\|)$
Ganter (1984)	$O(\|G\|^2 \|M\| \|L\|)$
Bordat (1986)	$O(\|G\| \|M\|^2 \|L\|)$
Downling (1993)	$O(\|G\|^2 \|M\| \|L\|)$
Kuznetsov (1993)	$O(\|G\|^2 \|M\| \|L\|)$
Godin et al. (1995)	$O(2^{2\mu} \|G\|)$
Carpineto and Romano (1996)	$O(2^{2\mu} \|G\|)$
Pasquier at al. (1998)	N/A
Lindig (1999)	$O(\|G\|^2 \|M\| \|L\|)$
Nourine and Raynaud (1999)	$O((\|G\| + \|M\|) \|G\| \|L\|)$
Stumme at al. (2000)	$O(\|G\|^2 \|M\| \|L\|)$
Valtchev (2001)	$O((\|G\| + \|M\|) \|G\| \|L\|)$
Merwe (2002)	$O(max\{\{g\}' \mid g \in (G, M, I)\} n^2 \|L\|)$
Lévy and Baklouti (2004)	N/A
Krajca et al. (2008)	N/A
Baixeries et al. (2009)	$O(\|G\| \|M\|^2 w(G))$
Andrews (2011)	N/A

Table 1 shows a summary of time complexities of algorithms[2]. $|G|$ denotes the number of objects, $|M|$ the number of attributes and $|L|$ the size of the concept lattice. In Godin's (1995) algorithm, μ designates an upper bound on $|f(x)|$ where the set of objects associated with the attribute x is denoted by $f(x)$. When there is a fixed upper bound μ, the time complexity of this algorithm is $O(|G|)$. The Nourine algorithm Nourine and Raynaud (1999) is half-incremental. It incrementally constructs the concept set, but formulates the lattice graph in a batch. There is a *max* function in Merwe's algorithm. It returns a number of elements in the largest intent of concepts, which extents consist of {g}.

Most of the mentioned algorithms were used in different application areas. We refer to Yevtushenko (2004), Carpineto and Romano (1996), Lindig (1995), Siff and Reps (1997), Snelting (2000), Vogt and Wille (1995), Goethals (2002) for more information on this.

The main issues for computing all the formal concepts of a context are related to the way all the concepts of a context without the repetitive generation of the same concept can be generated. Kuznetsov and Ob'edkov (2001) noted that an empirical comparison of algorithms is not an easy task for a number of reasons. First of all, algorithms described by authors are often unclear, leading to misinterpretations. Secondly, the data structures of the algorithms and their realisations are often not specified. Another issue is related to the setting up of consensus data sets to use as test beds. The context parameters such as an average attribute set associated with an object and vice versa, or the size and density of contexts should be considered. The test environments such as programming languages, implementation techniques and platforms are also crucial factors which influence the performance of algorithms.

[2] Note that not all of the algorithms are indicated in the table.

3.1 Selected algorithm

The Valtchev algorithm extended the Godin et al. (1995) algorithm based on two scenarios. In the first scenario, the algorithm updates the initial lattice by considering new objects one at a time. In the second one, it builds the partial lattice over the new object set first and then merges it with the initial lattice. The first algorithm showed an improvement of the Godin et al. algorithm and was recommended for small sets of objects. On the other hand, the second was recommended as the right choice for medium size sets. Selected incremental algorithm seems to be very effective in computation over small or medium data sets.

In our work we try to extend all the good features of these algorithms in order to be able to use them for the computation of larger incidence matrices. An accurate description of the algorithm Valtchev and Missaoui (2001) brought an idea of using a tree structure in the algorithm to avoid the continuous passing of a concept lattice. This will save a lot of time in the case of large data collections, and also a large and dense incidence matrix. The second improvement relates to effective data storage. We applied the common research of the Amphora Research Group ARG (2012) in the area of finite automata. A sparse binary matrix based on finite automaton was developed and we used it to store the incidence matrix and computed concepts. It allows us to read columns or rows of incidence matrices more quickly in comparison with other types of matrix storage.

3.2 What does "incremental" mean in the context of FCA algorithms

There are several dimensions, along which existing algorithms can be characterised, e.g. computation complexity, mechanism of checking of a new concept, the way of performing a calculation (i.e., batch or incremental data mining), etc. An *incremental algorithm* produces a concept lattice for the first i objects on the i-th execution step. The new set of concepts is obtained as a result of an update of the previous one. When a new object is added to a context, the following steps should be performed in order to obtain the concept lattice of the updated context:

1. Generation of new concepts in the case when the new object is meet-irreducible [3].
2. Update of concepts already existing in the previous lattice. Only the concepts, the intents of which are included into the new object attribute set, are updated. The new object is added to their extents.
3. Update of links between neighbour concepts in the case when the Hasse diagram of the lattice is required.

Listing 1 The simplified scheme of an incremental algorithm for the construction of the concept set

```
1. Procedure IncrementalConceptSetCalculation
      Input: A context (G, M, I)
      Output: A list of all concepts of context

      Concepts := ∅
      Concepts := Concepts ∪ {(empty set, M)}
      for each g in G
            AddObject(g)

2. Procedure AddObject
      Input: g

      for each Concept in Concepts
            if (Concept.Intent ⊆ {g}')
                  Concept.Extent := Concept.Extent ∪ {g}

      if ( ({g},{g}') ∉ Concepts)
            Concept.Add({g},{g}')
            GenerateNewConcepts({g}' || Concept.Intent)
```

The typical scheme of the incremental algorithm is presented in listing 1. The procedure IncrementalConceptSetCalculation demonstrates a general scheme of the work of incremental algorithms, i.e., the generation of the concept lattice on an object per object basis.

The generalization of this algorithm known as Valtchev's algorithm (proposed in Valtchev and Missaoui 2001) became a general pattern for the design of concept lattices. It is generally supposed that the computational complexity of this algorithm is equal to the computational complexity of the Nourine–Raynaud algorithm. However, the description of this algorithm does not include any tree structure to improve searching in a concept list. In our work, a tree structure that stored the concept list in an automaton matrix was used within Valtchev's algorithm. Experimental results show an interesting improvement.

3.3 Valtchev's algorithm

A problem common to all FCA algorithms consists in the size of an incidence matrix and the density of input data. Usually, not the whole concept lattice is computed because it is time-consuming. We decided to use a new alternative approach to data storage and a tree structure to improve Valtchev's algorithm. It enables us to use this algorithm on incidence matrices that are much larger (thousands of objects and attributes). Finally, we are able to use this modified algorithm for example in document retrieval.

The following points summarize the reasons, why we decided to pay attention to Valtchev's algorithm and its possible improvement:

- It is an incremental algorithm and we want to work with dynamic data sets (e.g. the number of documents is increasing).
- It is very difficult to obtain a good specification of each algorithm, because the sources are not usually well defined. Valtchev's algorithm is well described.
- The time complexity of this algorithm is better than the complexity of the other incremental algorithms.
- The tree structure is not implemented. One needs to pass through the whole existing concept lattice to insert a new concept and update lattice structure.
- The algorithm is based on cyclic reusing of the data stored in previous executive steps. Therefore, an effective data structure should be applied.

3.4 The description of Valtchev's algorithm

First, the main idea of Valtchev's approach will be introduced. Listings 2 represents a pivotal procedure called *ADD_OBJECT*. It could be divided into two parts. The first part (lines 1–11) is in charge of any specific treatment required by the presence of some attributes in the description of o_i which are not in $A_i = \bigcup_{j<i}\{o_j\}'$. The second part consists of a top-down traversal of lattice $\mathcal{L}_{i-1}$ with a recognition of the current concept's category (lines 12–28). As for the updates, modified concepts merely receive o_i in their extent, whereas the detection of a generator c leads to the creation of the new concept $\overline{c}$ with its intent being the intersection of the generator's intent with $\{o_i\}'$ and update the generator's extent augmented by o_i (lines 19–25). The new concept is then inserted into the (partially) completed lattice $\mathcal{L}_i$, a task which requires the detection of all its upper covers. Thus the algorithm *FIND_UPPER_COVERS* (see listing 3) looks through a superset of the actual covers and picks up the nodes for which the following holds: the candidate set is made up of all concepts whose intents are strictly smaller in size than $Int(\overline{c})$. A candidate qualifies only if it is a super-concept of $\overline{c}$, but none of its lower covers is a super-concept of $\overline{c}$ on its own. An upper cover is physically linked to $\overline{c}$, and the obsolete links, i.e., those relating the generator c to an upper cover, are removed. The integration of $\overline{c}$ is completed

[3] An element that has only one direct upper neighbour is called *join-irreducible* and an element that has only one direct lower neighbour is called *meet-irreducible*.

by a link to its generator (line 26) (Valtchev and Missaoui 2001).

Listing 2 Valtchev's algorithm Valtchev and Missaoui (2001) - Add_Object

```
1. Procedure Add_Object
     Input:  ℒ          // a lattice
             o          // an object
     Output: ℒ+         // a lattice

2. Local: Classes, Classes_New : array [0..|A ∪ {o}'|]
     of concept set

3. if (ℒ = ∅)
4.    ℒ+ := ({o}, {o}')
5. else
6.    if {o}' ⊈ Intent(⊥(ℒ))
7.       if Extent(⊥(ℒ)) = ∅
8.          Intent(⊥(ℒ)) := Intent(⊥(ℒ)) ∪ {o}'
9.       else
10.         c := (∅,Intent(⊥(ℒ)) ∪ {o}') ; NEW_LINK(c,⊥(ℒ))
11.         ADD(ℒ, c) ; ⊥(ℒ) := c

12.   for i from 0 to |A ∪ {o}'|
13.      Classes := ∅
14.      Classes_New := ∅

15.   for each c̄ in ℒ
16.      ADD(Classes[|Intent(c̄)|], c̄)

17.   for i from 0 to |A ∪ {o}'|
18.      for each c̄ in Classes[i]
19.         if Intent(c̄) ⊆ {o}'
20.            ADD(Extent(c̄), o)
21.         else
22.            Int := Intent(c̄) ∩ {o}'
23.            if not (int', Int) ∈ Classes_New[|Int|]
24.               c := (Extent(c̄) ∪ {o}, Int)
25.               ADD(Classes_New[|Int|], c)
26.               FIND_UPPER_COVERS(c,c̄) ; NEW_LINK(c̄, c)
27.      ADD(Classes_New[i], c̄)

28.   ℒ+ := ⋃_{i=0}^{|A∪{o}'|} Classes_New[i]
```

Listing 3 Valtchev's algorithm Valtchev and Missaoui (2001) - Find_Upper_Covers

```
1. procedure FIND_UPPER_COVERS
     Input:  n          // a new concept
             gen        // a concept generator
2. Int := Intent(new)
3. for j from 0 to |Int|−1
4.    for each c̃ in Classes_New[j]
5.       if Intent(c̃) ⊆ Int
6.          Cover := true
7.          for each ĉ in Cov'(c̃)
8.             if Intent(ĉ) ⊆ Int
9.                Cover := false
10.               break

11.         if Cover = true
12.            if gen ∈ Cov'(c̃) then
13.               DROP_LINK(gen, c̃)
15.            NEW_LINK(new, c̃)
```

4 Proposed improvements

4.1 Data storage and fast querying into an incidence matrix

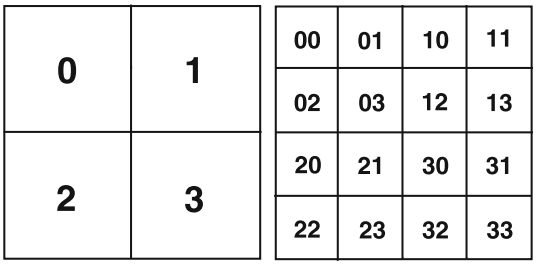
A quick look at Valtchev's algorithm reveals that we should pay attention to the *for-cycles* (lines 12, 15, 17, 18 in listing 2 and lines 3, 4, 7 in listing 3). Especially, line 15 (see listing 2) seems to be very time-consuming in the case of a large concept lattice $\mathcal{L}$. Moreover, Valtchev's algorithm works with an already generated concept lattice and that is why the data should be stored very effectively because of a great number of concepts in the case of a large and dense incidence matrix. The improvement is based on the binary automaton matrix that has been used for data storing and serves as a basis of the implemented tree structure. The binary automaton matrix is a result of common research conducted by the Amphora Research Group (ARG 2012). For details on the description of the effective data storage of binary matrices see Martinovič et al. (2005). This approach was extended among others by the implementation of *meet* and *join* operators.

The concept lattice and tree structure are created in parallel. The tree structure speeds up traversing the concept lattice and concept searching. The tree structure is stored as well as the set of already generated concepts are stored in an automaton matrix. The extents and intents of all concepts are stored separately to save space and memory.

Large sparse binary matrices play an important role in Computer Science (indexing of class hierarchy Dencker et al. 1984), and in many modern information retrieval methods. These methods, such as clustering, web graph computations, web link analysis, binary factor analysis, perform a huge number of computations with matrices. Therefore, all matrices should be carefully designed (e.g. sparse matrix multiplication Yuster and Zwick 2005). Therefore, several storage techniques were developed to fit particular algorithms' requirements, such as Yale Sparse Matrix Format (Bank and Douglas 2012), compressed Row Storage (CRS), Compressed Column Storage (CCS). However, we focused on a storage system based on the Automata Theory in our work. We refer to Eilenberg (1974), Kozen (1997) for more details on this theory.

We consider a square matrix M of order $2^n \times 2^n$ to facilitate the application of finite automata. Each element of such a matrix is represented by an address ε of the length n over the alphabet $\Sigma = \{0, 1, 2, 3\}$. Each element of the matrix corresponds to a subsquare of size 2^{-n} of the unit square.

The main matrix is split recursively into four quadrants (submatrices). The depth of such recursion is n. Each quadrant is signed by one letter of Σ alphabet that represents its current address ω. One letter of the alphabet is added to the quadrants' addresses after the next step of recursion is executed.



M[x,y]	ω	M[x,y]	ω
[1, 1]	00	[3, 1]	20
[1, 2]	01	[3, 2]	21
[1, 3]	10	[3, 3]	30
[1, 4]	11	[3, 4]	31
[2, 1]	02	[4, 1]	22
[2, 2]	03	[4, 2]	23
[2, 3]	12	[4, 3]	32
[2, 4]	13	[4, 4]	33

Fig. 4 Addressing in a 4 × 4 matrix

Example 2 Addressing in a square matrix. Figure 4 shows the level-2 addresses based on the quadrants of a 4 × 4 matrix. The related table shows the addresses (w) of all matrix elements.

In order to specify the values of a matrix of dimension $2^n \times 2^n$, a mapping $\Sigma^n \rightarrow R$ has to be defined, or alternately, it can be specified just by a set of non-zero values, i.e., by a language $L \subseteq \Sigma^n$ and a mapping $f_M : L \rightarrow R$.

This kind of storage system allows direct access into the stored matrix. Each element can be accessed independently of the others and the whole process of getting an element from a matrix has a constant time complexity. Let M be a matrix of order $2^n \times 2^n$. Then the time complexity of the access is bound by $O(\log_2 n)$. For detailed information see Snášel et al. (2002).

This approach to data storage makes it possible to implement the *GET_ROW* or *GET_COLUMN* methods that are equally time-consuming. This aspect is very important in the case of computation within all FCA algorithms.

4.1.1 Binary matrices and finite automata

Let M be a binary matrix of dimension $n_1 \times n_2$, where $n_1, n_2 \in \mathbb{N}$. Let $n = max(n_1, n_2)$ and $size = 2^{\lceil \log_2 n \rceil}$. Finite automaton which is represented by matrix M is defined as:

$$A = (Q; \Sigma; \delta; q_0; F),$$

where:

- Q is the finite set of states;
- $\Sigma = \{0, 1, 2, 3\}$ is the input alphabet;
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function;
- q_0 is the initial state;
- $F = \{q_f\}$ is the set of (one) finite state.

The language L which is accepted by automaton A is defined:

$$L = \left\{ w \in \Sigma^* \mid \exists x, y \in \mathbb{N}, 1 \leq x, y \leq n;\ then\ if \right.$$

$$M_{x,y} = 1, then\ w = l(x, y, size)$$

where $l : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \Sigma^*$:

$$l(x, y, val) = \begin{cases} \varepsilon & \text{for } val = 0 \\ 3l(x - val, y - val, val/2) & \text{for } x \geq val \wedge y \geq val \\ 2l(x - val, y, val/2) & \text{for } x \geq val \wedge y < val \\ 1l(x, y - val, val/2) & \text{for } x < val \wedge y \geq val \\ 0l(x, y, val/2) & \text{for } x < val \wedge y < val \end{cases}$$

4.1.2 Reconstruction of the matrix

For each element of the matrix $M_{x,y}$ holds

$$M_{x,y} = \begin{cases} 1 & \text{automaton } A \text{ accepts } l(x, y, size) \\ 0 & \text{otherwise.} \end{cases}$$

4.1.3 Construction of the automaton

Finite automaton for a given binary matrix can be constructed in several ways, e.g. by the usage of regular expression (Dvorský 2004) or as a composition of elementary automatons (Rozenberg and Salomaa 1997). All these methods of construction have a common disadvantage—the whole matrix, i.e., a complete language has to be known in advance. Better choice consists of an incremental construction of the automaton.

The incremental construction method starts with an empty automaton (with the exception of a global initial and final state) and step by step updating of the automata. As soon as a new element is inserted into the matrix the corresponding word is added to the automaton and the automaton is transformed into a new one. The whole algorithm is well described in Martinovič et al. (2005).

Usually, a high space complexity is the main disadvantage of such algorithms. The number of states of the automaton is increased after a new word $w \in L$ is inserted. Therefore, the so called minimization of the automata is performed after a given number of insertions. Theorems and proofs of one minimization approach are described in Martinovič et al. (2005).

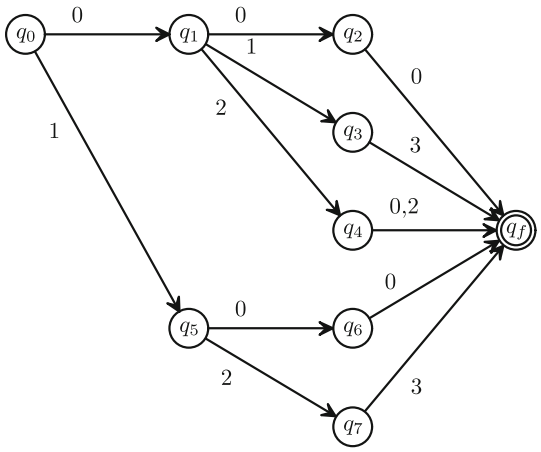


Fig. 5 Non-minimized automaton A_6

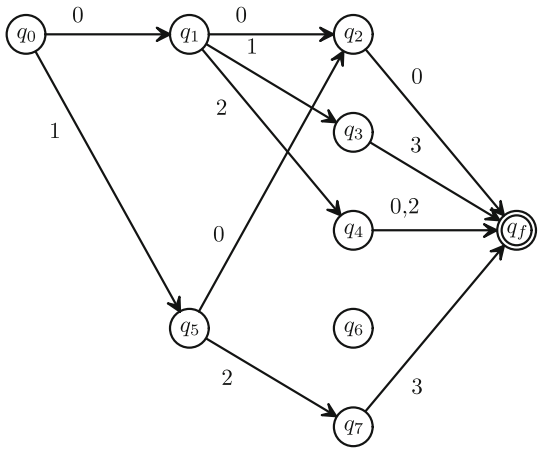


Fig. 6 Automaton A_6, the state q_6 will be deleted

Example 3 Automaton matrix. Let M be a matrix of size 4×6

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix is described by the language $L_M = \{000, 013, 020, 100, 123, 022\}$. Initial automaton A_0 does not accept any language, i.e., $L_0 = \emptyset$. Automaton A_1 is constructed from A_0 and accepts language $L_1 = \{000\}$. Automaton A_2 is an extension of automaton A_1 and accepts language $L_2 = L_1 \cup \{013\} = \{000, 013\}$. In this way automaton A_6 accepts the whole language L_M (see Fig. 5). This is an expanded, non-minimized form of automaton. Now, automaton A_6 can be minimized. The whole minimization process starts in the final state q_f and goes to the initial state of the automaton. The minimizing process takes the two following steps:

1. There are two transitions between q_2 and q_f and q_6 and q_f, respectively. The transition symbol is 0 in both cases. These two transitions can be joined together and state q_6 is deleted (see Fig. 6).
2. There are also two transitions between q_3 and q_7 as initial states and q_f as the final state. Transitions are labelled by symbol 3. As well as in the first step, transitions are unified and state q_7 is deleted (see Fig. 7).

The minimization process plays an important role with respect to the usage of the automaton matrix. The data structure based on a binary matrix automaton is a hidden element of the suggested algorithm, however, it represents an essential part of the improvement of Valtchev's algorithm. The parallel tree structure as well as the incidence matrix or the list of computed concepts are stored in such a data structure, which saves a lot of time and storage space in the case of large data sets.

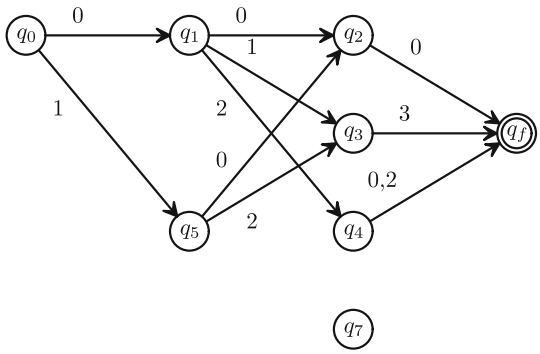


Fig. 7 Minimized automaton A_6, the state q_7 will be deleted

4.2 Algorithm improvements

Listings 4 represents the procedure that creates a new concept list (*CREATE_CONCEPT_LIST*). A list of objects that should be added and a concept lattice (if it exists) are the inputs of this procedure. The procedure can be divided into two parts. The first one is called the *initialization phase* (lines 2–7). There is a decision point that either creates a new concept lattice and a related tree structure, or loads a concept lattice and a tree structure of already existing concepts. The tree structure is represented by a square adjacency automaton matrix $n \times n$, where n represents the number of all existing concepts plus the number of concepts that will be added. The tree structure is dynamic. We refer to Martinovič et al. (2005) to see more details of the process of adding a new element into an automaton matrix. It is done incrementally. That is why the tree structure can dynamically grow with the size of the concept lattice. The method *CREATE_TREE* is responsible for the creation of a new matrix $|\mathcal{L}^+| \times |\mathcal{L}^+|$, whereas the method *LOAD_TREE* loads the existing tree and enlarges its matrix upon $|\mathcal{L}| \times |\mathcal{L}|$. The second part called the *updating phase*, starts with a loop (line 8) over all new objects. The most important method

with respect to lattice updating is *INSERT_CONCEPT* (see listing 5). The method *TRY_ALL_CHILDREN* (see listing 6) is responsible for the updating of the tree structure and will be described in more detail together with the *INSERT_CONCEPT* method. However, there is one more method used within the *CREATE_CONCEPT_LIST* procedure. It is called *GET_NODE_CHILDREN* and its input is the index of a selected node of the tree structure. This method returns a vector that consists of the nodes' indexes of all children of the selected node in the tree structure.

Listing 4 Create_Concept_List - Improved algorithm based on the tree structure and binary automaton matrix

```
1. Procedure CREATE_CONCEPT_LIST
     Input: L%          //a lattice
            o%          //a list of object
            oldTree%    //a tree of existing concepts
     Output: L+%        //a lattice

2.   Local: chIndex%        //child concept index
            pIndex = 0%     //parent concept index
            objectsDone%    //the number of added objects
            oldConceptCounter% //#computed concepts
            tmpChildren : List% //temporary variable

3.   if (L = ∅)
4.     L+ := ({∅}, {∅}')     5.  CREATE_TREE(|o|)
6.   else
7.     LOAD_TREE(oldTree) ; L+ := L
8.   for (chIndex = objectsDone; chIndex < |o|; chIndex++)
9.   if (INSERT_CONCEPT(0, {o[chIndex]}'))
10.      oldConceptCounter = |L+| − 1;
11.      tmpChildren := GET_NODE_CHILDREN(0)
12.      for (pIndex=0; pIndex<|tmpChildren|; pIndex++)
13.        if (tmpChildren[pIndex] != oldConceptCounter)
14.          if (INSERT_CONCEPT(tmpChildren[pIndex],
                 L+[oldConceptCounter].Intent ∩
                 L+[tmpChildren[pIndex]].Intent))
15.            TRY_ALL_CHILDREN(tmpChildren[pIndex],
                                |L+| − 1)
```

This procedure (see listing 5) updates not only the list of concepts but also the tree structure. The inner method *GET_CONCEPT* returns an index of the concept with the specified intent in case it exists in the concept list (line 3). The method returns NULL in case such a concept does not exist; then the *ADD_CONCEPT* method is called. This method computes whole concepts ($'$operator, the list of common objects is returned) and adds them into the concept list. After that, the method *ADD_TREE_NODE* updates the tree structure and a new link between the "parent" concept and the new concept is created. An index of the new concept is $|\mathcal{L}^+|$. If the *GET_CONCEPT* method returns an index, then it means that the concept $\overline{c}$ already exists with a given intent and the list of objects is inserted into the $\overline{c}.Extent$ (line 9). The *INSERT_CONCEPT* procedure returns TRUE if a new concept has been added, otherwise FALSE.

Listing 5 Insert_Concept procedure used in Create_Concept_List

```
1. Procedure INSERT_CONCEPT
     Input: pIndex //parent concept index
            i      //an intent vector
     Output: TRUE if a new concept was added, otherwise FALSE

2.   Local: c̄      //a concept

3.   c̄ := GET_CONCEPT(i, L+)  //it returns a concept
                              //with a given intent,
                              //otherwise NULL
4.   if (c̄ = NULL)
5.     ADD_CONCEPT(L+, ({i}',i))
6.     ADD_TREE_NODE(pIndex, |L+|)
7.     return TRUE
8.   else
9.     ADD_EXTENT(c̄, {i}')
10.    return FALSE
```

This procedure (see listing 6) represents a recursive part of the update phase. If a new object has been inserted into some concept, it is necessary to update all the other concepts linked to this one in the tree structure. So the tree structure is updated. We leave the description of the following procedure in the recursive form, because it is more vivid and it better elucidates the meanings of the updating process. Of course, it is easy to transform it into the non-recursive form which has smaller memory demands.

Listing 6 Try_All_Children procedure used in Create_Concept_List

```
1. Procedure TRY_ALL_CHILDREN
     Input: pIndex          //the node in the tree
            lastAddedConcept //last added node index

2.   Local: tmpChildren : List  // list of children

3.   tmpChildren := GET_NODE_CHILDREN(pIndex)
4.   if (|tmpChildren| = 0)
5.     return
6.   for (i=0; i<|tmpChildren|; i++)
7.     if (tmpChildren[i] != lastAddedConcept)
8.       if (INSERT_CONCEPT(tmpChildren[i],
                 L+[lastAddedConcept].Intent ∩
                 L+[tmpChildren[i]].Intent))
9.         TRY_ALL_CHILDREN(tmpChildren[i], |L+| − 1)
```

5 Comparison and experiments

The new improved FCA algorithm was described in the previous section. The following text is focused on experiments and comparison with the original Valtchev's algorithm. All tested algorithms and their variants were implemented in the same programming language, by the same person, and with an emphasis on maximum performance and code optimization. Here is a short legend for all the tables in this section:

- **i**—the index of computation (test)
- |**G**|—represents the number of objects in the incidence matrix

– |**M**|—represents the number of attributes in the incidence matrix
– **d**—is a density of an incidence matrix, i.e., the percentage of non-zero elements. The density of a context $\mathbb{K} := (G, M, I)$ is given by the following formula:

$$d = \frac{|I|}{|G| \cdot |M|} * 100$$

An incidence matrix is usually said to be dense if $d > 25$, however, the property of being dense is rather vague, and there is no rigorous criterion for it.
– $|\mathcal{L}|$—is a size of concept lattice.
– **Gajdos**—computation time of the improved algorithm. The incidence matrix has been stored by using a binary automaton matrix. The used tree structure is also based on an automaton matrix as well as the output storage space for all computed concepts.
– **Valtchev**—computation time of the original algorithm by Petko Valtchev. The incidence matrix is stored here by the usage of CRS method. The system of storing all concepts depends on the size and density of the incidence matrix.
– **ValtchevA**—computation time of the original algorithm again, however the incidence matrix has been stored using the automaton matrix. The system of storing of all concepts depends on the size and density of the incidence matrix.

All experiments were performed on the following hardware platform: Intel Core2 CPU, 2.4 GHz, 8GB RAM, Windows 7 64-bit.

5.1 An experiment with a fix matrix size and increasing density

In this experiment (see Table 2), three algorithms were used. We want to show that a simple usage of the automaton matrix (see ValtchevA in the table) does not yield any improvement. On the contrary, such an algorithm is even more time consuming, because no tree structure is used and a sparse automaton is created. It is evident, that the usage of the Gajdos algorithm is more effective in the case of dense incidence matrices. Table 2 shows the results up to a density of 50 %. The higher density leads to a dual problem solution where zeros and ones are swapped. Some computation times were too large to have importance for real applications (see >1M values in Table 2).

Table 2 Variable matrix density, $|G| = 100$, $|M| = 100$

i	d	$\lvert\mathcal{L}\rvert$	Gajdos	Valtchev	ValtchevA
1	5	262	0.71	**0.05**	0.53
2	10	1,203	0.78	**0.09**	3.31
3	15	4,054	1.12	**0.48**	25.93
4	20	11,150	**2.43**	3.51	193.44
5	25	29,987	**5.38**	20.23	1,438.36
6	30	84,502	**19.32**	158.62	11,900.32
7	35	249,577	**96.56**	1,316.03	109,791.24
8	40	787,574	**235.13**	14,653.83	>1M
9	45	2,319,430	**934.61**	177,641.38	>1M
10	50	8,434,673	**3,671.33**	>1M	>1M

(time in s)
Bold values indicate the best achieved results (computation time in s)

Table 3 Variable number of attributes, $|G| = 1,000$, $d = 5$ %

i	$\lvert M\rvert$	$\lvert\mathcal{L}\rvert$	Gajdos	Valtchev
1	100	4,251	**1.79**	2.63
2	200	17,053	**9.19**	32.68
3	300	41,680	**29.81**	159.77
4	400	77,853	**67.41**	510.12
5	500	125,393	**133.57**	1,253.79
6	600	182, 066	**234.01**	2,545.75
7	700	246,514	**376.78**	4,770.51
8	800	319,135	**568.68**	8,312.39
9	900	397,605	**816.64**	13,307.96
10	1,000	480,567	**1,176.44**	20,377.15

(time in s)
Bold values indicate the best achieved results (computation time in s)

5.2 An experiment with an increasing number of attributes in the incidence matrix

We did not use the ValtchevA algorithm in this experiment (see Table 3). The previous experiment has shown that such an algorithm is not effective. Particular incidence matrices consist of 1,000 objects and the density has been always set to 5 % of the respective matrix.

5.3 An experiment with an increasing number of objects in the incidence matrix

At the beginning, the incidence matrix consists of 1,000 objects ($|G|$) and 100 attributes ($|M|$). The density d has always been 5%. The number of attributes and the density were fixed. Table 4 shows the total times of computations (in seconds) for the specific size of incidence matrix. The ValtchevA algorithm has not been tested any more due to its obvious ineffectivity.

It is evident, that the improved algorithm is more effective again. The incremental time, that we need to insert 1,000 objects into the existing concept lattice, grows quite slowly in comparison with Valtchev's algorithm (compare two consecutive computation times in the appropriate table column).

Table 4 Variable number of objects, $|M| = 100$, $d = 5\ \%$

i	$\lvert G\rvert$	$\lvert\mathcal{L}\rvert$	Gajdos	Valtchev
1	1,000	4,029	**1.45**	1.65
2	2,000	8,515	**2.39**	10.25
3	3,000	13,159	**4.45**	30.51
4	4,000	17,484	**6.29**	65.38
5	5,000	21,313	**9.11**	114.89
6	6,000	24,486	**11.38**	174.05
7	7, 000	27, 468	**13.61**	246.43
8	8,000	29,904	**15.57**	323.58
9	9,000	32,396	**18.84**	418.11
10	10,000	34,653	**21.22**	532.58

(time in s)
Bold values indicate the best achieved results (computation time in s)

The incidence matrix is not very important for the whole computation because a new object is compared with concepts in the concept lattice $\mathcal{L}$. In this case, the automaton matrix and a related tree structure play an important role within the improved algorithm.

6 Conclusion and future work

The practical approach to all of the complex problems usually leads to particular problems concerning large data sets, needs of the simplification and acceleration of computation phases. In this article, we described several aspects and problems related to Formal Concept Analysis and its usage in the area of information retrieval. The algorithmic aspects were described in more detail and a new algorithm based on Valthev's algorithm was introduced. The reasons for choosing this algorithm were presented as well as several improvements based on the usage of the automaton matrix and tree structures. Finally, a set of experiments demonstrated significant improvements. The proposed algorithm can be applied to larger data sets.

Current research in this area and the near future work will consist of the utilization of modern hardware, e.g. graphics processor units (GPUs) to improve performance. Of course, not every part of the proposed algorithm could be done in a parallel way. On the other hand, designed data structures can easily be implemented so that they can be represented in a suitable form for GPUs, e.g. dynamic textures, etc.

References

Andrews S (2011) In-close2, a high performance formal concept miner. In: ICCS, pp 50–62

ARG: Amphora Research Group (2012) VŠB-Technical University of Ostrava. http://arg.vsb.cz/

Baixeries J, Szathmary L, Valtchev P, Godin R (2009) Yet a faster algorithm for building the Hasse diagram of a concept lattice. In: Ferre's, Rudolph S (eds) Formal Concept Analysis. Lecture notes in computer science, vol 5548. Springer, Heidelberg

Bank RE, Douglas CC (2012) Sparse Matrix Multiplication Package. http://www.mgnet.org/~douglas/Preprints/pub34.pdf

Birkhoff G (1967) Lattice theory, 3rd edn. American Mathematical Society, Providence

Bordat JP (1986) Calcul pratique du Treillis de Galois dune Correspondance. Mathematiques et Sciences Humaines, pp 31–47

Brüggemann R, Voigt K, Steinberg C (1997) Application of formal concept analysis to evaluate environmental databases (1997). Chemosphere 1995:479–486

Bělohlávek, R., Sklenář, V (2005) Formal concept analysis constrained by attribute-dependency formulas. In: International conference on formal concept analysis (ICFCA 2005), vol 3403. Springer, Berlin, pp 176–191

Carpineto C, Romano G (1996) A lattice conceptual clustering system and its application to browsing retrieval. Mach Learn 24:95–122

Chain M (1969) Algorithme de recherche des sous-matrices premieres des sous-matrices. Bulletin Math. Soc. Sci. Math. R.S. Roumanie 13:21–25

Cole RJ, Eklund PW (1996) Text retrieval for medical discharge summaries using snomed and formal concept analysis. In: First Australian Document Computing Symposium (ADCS 1996), pp 50–58 (RJ's first paper)

Dencker P, Dürre K, Heuft J (1984) Optimization of parser tables for portable compilers. ACM Trans Progr Lang Syst 6:546–572

Downling CE (1993) On the irredundant generation of knowledge spaces. Math. Psychol. 37:49–62

Dvorský J (2004) Word-based Compression Methods for Information Retrieval Systems. Ph.D. thesis, Charles University, Prague, Czech Republice

Eilenberg S (1974) Automata, languages, and machines, vol A. Academic Press, London

Eklund P, Groh B, Stumme G, Wille R (2000) A contextual-logic extension of TOSCANA, conceptual structures: logical, linguistic, and computational issues. In: 8th International conference on conceptual structures (ICCS 2000). Springer, Berlin, pp 453–467

Frolov A, Husek D, Muraviev I, Polyakov P (2007) Boolean factor analysis by attractor neural network. IEEE Trans Neural Netw 18(3):698–707

Galitsky B, de la Rosa JL (2011) Concept-based learning of human behavior for customer relationship management. Inf Sci 181(10):2016–2035. doi:10.1016/j.ins.2010.08.027

Ganter B (1984) Two basic algorithms in concept, analysis. FB4-Preprint No. 831 pp 312–340

Ganter B, Glodeanu C (2012) Ordinal factor analysis. Formal Concept Analysis. In: Domenach F, Ignatov D, Poelmans J (eds) Lecture notes in computer science, vol 7278. Springer, Berlin, pp 128–139

Ganter B, Kuznetzov S (1998) Stepwise construction of the Dedekind–McNeille completion, conceptual structures: theory, tools and applications. In: 6th International conference on conceptual structures (ICCS 1998). Springer, Berlin, pp 295–302

Ganter B, Reuter K (1991) Finding all closed sets: a general approach. Order 8:282–290

Ganter B, Wille, R (199) Formal concept analysis. Springer, Berlin (1999)

Godin R, Missaoui R, Alaoui H (1995) Incremental concept formation algorithms based on galois (concept) lattices. Comput Intell 11:246–267

Godin R, Missaoui R, April A (1993) Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. Int J Man Mach Stud, pp 747–767

Goethals B (2002) Efficient Frequent Pattern Mining. Ph.D. thesis, Transnationale Universiteit Limburg, School voor Informatietechnologie

Kaytoue M, Kuznetsov SO, Napoli A, Duplessis S (2011) Mining gene expression data with pattern structures in formal concept analysis. Inf Sci 181(10), 1989–2001. doi:10.1016/j.ins.2010.07.007

Kozen DC (1997) Automata and computability. Springer, Berlin

Krajca P, Outrata J, Vychodil V (2008) V.: Parallel recursive algorithm for FCA. Palacky university, Olomouc, pp 71–82

Krajca P, Outrata J, Vychodil V (2010) Parallel algorithm for computing fixpoints of galois connections. Ann Math Artif Intell 59(2):257–272

Kuznetsov SO (1993) A fast algorithms for computing all intersections of objects in a finite semi-lattic. Autom Documentation Math Linguist 27:11–21

Kuznetsov SO, Ob'edkov SA (2001) Comparing performance of algorithms for generating concept lattices, international workshop on concept lattices-based theory. In: Methods and Tools for Knowledge Discovery in Databases (CLKDD01) in ICCS 2001, pp 35–47, Stanford University

Lévy G, Baklouti F (2004) A distributed version of the Ganter algorithm for general Galois lattices

Li J, Mei C, Lv Y (2012) Knowledge reduction in real decision formal contexts. Inf Sci 189:191–207. doi:10.1016/j.ins.2011.11.041

Lindig C (1995) Concept based component retrieval. In: Working notes of the IJCAJ 1995 workshop: formal approaches to the reuse of plans, proofs, and programs, pp 21–25

Lindig, C.: Algorithmen zur Begriffsanalyse und ihre Anwendung bei Softwarebibliotheken. Master's thesis, Technical University of Braunschweig (1999). http://www.gaertner.de/lindig/papers/diss/

Martinovič J, Dvorský J, Snášel V (2005) Sparse binary matrices. ITAT 2005:103–116

Medina J, Ojeda-Aciego M (2010) Multi-adjoint t-concept lattices. Inf Sci 180(5):712–725. doi:10.1016/j.ins.2009.11.018

Norris EM (1978) An algorithm for computing the maximal rectangles in a binary relation. Revue Roumaine de mathematiques Pures et Alliquees 23:243–250

Nourine L., Raynaud O (1999) A fast algorithm for building lattices. Inf Process Lett 71:199–204

Priss U (2000) Lattice-based information retrieval. Knowl Organ, pp 132–142

Rozenberg GA, Salomaa E (1997) Handbook of formal language, vol I–III

Siff M, Reps T (1997) Identifying modules via concept analysis. In: Proceedings of the international conference on the software maintaince. IEEE Computer Society Press, New York, pp 170–179

Snášel V, Dvorský J, Vondrák V (2002) Random access storage system for sparse matrices. In: Andrejková G, Lencses R (eds) ITAT 2002. Brdo, High Fatra, Slovakia

Snelting G (2000) Software reengineering based on concept lattices. In: 4th European conference on software maintenance and reengineering (CSMR 2000). IEEE Computer Society, New York, pp 3–10

Spangenberg N, Fischer R, Wolff KE (1999) Towards a methodology for the exploration of "tacit structures of knowledge" to gain access to personal knowledge reserve of psychoanalysis: the example of psychoanalysis versus psychotherapy. Psychoanalytic research by means of formal concept, analysis

Stumme G, Taouil RY, Bastide NP, Lakhal L (2000) Fast computation of concept lattices using data mining techniques. In: 7th International workshop on knowledge representation meets databases (KRDB 2000), pp 129–139 (2000)

Stumme G, Wille R (1998) Conceptual knowledge discovery in databases using formal concept analysis methods. Principles of data mining and knowledge discovery. In: 2nd European Symposium on PKDD 1998. Springer, Berlin, pp 450–458

Valtchev P, Missaoui R (2001) Building concept (Galois) lattices from parts: generalizing the incremental methods, conceptual structures: broadening the base. In: 9th international conference on conceptual structures (ICCS 2001). Springer, Berlin, pp 290–303

Vogt F, Wille R (1995) TOSCANA—a graphical tool for analyzing and exploring data. Graph Draw, pp 226–233

Wang L, Liu X, Cao J (2010) A new algebraic structure for formal concept analysis. Inf Sci 180(24):4865–4876. doi:10.1016/j.ins.2010.08.020

Wille R (2001) Why can concept lattice support knowledge discovery in databases? In: International workshop on concept lattices-based theory, methods and tools for knowledge discovery in databases (CLKDD 2001), ICCS 2001, pp 7–20

Wille R (2009) Restructuring lattice theory: an approach based on hierarchies of concepts. In: Proceedings of the 7th international conference on formal concept analysis, ICFCA 2009. Springer, Berlin, pp 314–339. doi:10.1007/978-3-642-01815-2-23

Yevtushenko S (2004) Computing and Visualizing Concept Lattices. Ph.D. thesis, Fachbereich Informatik der Technischen Universität Darmstadt

Yuster R, Zwick U (2005) Fast sparse matrix multiplication. ACM Trans Algorithms 1(1):2–13. doi:10.1145/1077464.1077466