Generation algorithm of a concept lattice with limited object access

Ch. Demko♦★, K. Bertet♦

♦ L3I - Université de La Rochelle - av Michel Crépeau - 17042 La Rochelle
cdemko,kbertet@univ-lr.fr
★ Joomla! Production Leadership Team
christophe.demko@joomla.org

Abstract. Classical algorithms for generating the concept lattice $(C, \leq)$ of a binary table (O, I, R) have a complexity in $O(|C| * |I|^2 * |O|)$. Although the number of concepts is exponential in the size of the table in the worst case, the generation of a concept is output polynomial. In practice, the number of concepts is often polynomial in the size of the table. However, the cost of generating a concept remains high when the table is composed of a large number of objects.
We propose in this paper an algorithm for generating the lattice with limited object access, which can improve the computation time. Experiments were conducted with Joomla!, a content management system based on relational algebra, and located on a MySQL database.

keywords: concept lattice ; databases ; algorithm

1 Introduction

Galois lattices (or *concept lattices*) were first introduced in a formal way in the graph and ordered structures theory [1–3]. Later, they were developed in the field of Formal Concept Analysis (FCA) [4] for data analysis and classification. The concept lattice structure, based on the notion of *concept*, enables data description while preserving its diversity. It is used to analyse data when organised by a binary relation between objects and attributes.

Galois lattice is a graph providing a representation of all the possible correspondences between a set of *objects* (or examples) O and a set of *attributes* (or features) I. The technological improvements of the last decades enable use of these structures for data mining problems though they are exponential in space/time (worst case). It has to be noted that in practice, in most cases, the size of the lattice remains reasonable.

In addition, some applications offer to only generate some concepts from the huge amount of available data. Bordat's algorithm [5] is the more appropriate since it generates the cover relation between concepts, and thus allows an on-demand generation of concepts. Moreover, huge amount of data are often described by a huge amount of objects. It is the case in databases where sophisticated key-indexation techniques are used to improve object access.

ISBN 978–2–905267–78–8, INRIA Nancy – Grand Est and LORIA, 2011.

In this paper, we propose the Limited Object Access algorithm (LOA algorithm), an extension of Bordat's immediate successors generation with a limited access to objects. This algorithm, compounded with an on-demand strategy, and with sophisticated key-indexation techniques to improve objects's access, aims to improve time computation for a large amount of objects. However, worst case theoretical complexity remains the same as Bordat's algorithm. Experiments were conducted with Joomla!, a content management system based on relational algebra, and located on a MySQL database.

This paper is organized as follows. In section 2, we describe the Galois lattice structure and the Bordat's generation algorithm. In section 3, we describe our limited object access algorithm, illustrated by an example and some experiments.

2 Description and generation of a concept lattice

2.1 Description of a concept lattice

The *concept lattice* is a particular graph defined and generated from a relation R between objects O and attributes I. This graph is composed of a set of *concepts* ordered by a relation verifying the properties of a *lattice*, i.e. an order relation $\leq$ (transitive, reflexive and antisymmetric relation) such that, for each pair of concepts in the graph, there exists both a lower bound and an upper bound. Therefore, a lattice contains a minimum (resp. maximum) element according to the relation $\leq$ called the *bottom* (resp. *top*) of the lattice. The *Hasse diagram* of a graph [1] is the cover relation of $\leq$ denoted as $\prec$, i.e. the suppression on the graph of both transitivity and reflexivity edges.

We associate to a set of objects $A \subseteq O$ the set $f(A)$ of attributes in relation R with the objects of A:

$$f(A) = \{y \in I \mid xRy \ \forall \ x \in A\}$$

Dually, to a set of attributes $B \subseteq I$, we define the set $g(B)$ of objects in relation with the attributes of B:

$$g(B) = \{x \in O \mid xRy \ \forall \ y \in B\}$$

These two functions f and g defined between objects and attributes form a *Galois correspondence*. The relation between the set of objects and the set of attributes is described by a *formal context*. A formal context C is a triplet $C = (O, I, R)$ (or $C = (O, I, (f, g))$) represented by a table.

A *formal concept* represents maximal objects-attributes correspondences (following relation R) by a pair (A, B) with $A \subseteq O$ and $B \subseteq I$, which verifies $f(A) = B$ and $g(B) = A$. The whole set of formal concepts thus corresponds to all the possible maximal correspondences between a set of objects O and a set of attributes I.

Two formal concepts (A_1, B_1) and (A_2, B_2) are in relation in the lattice when they verify the following inclusion property:

$$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow \left\| \begin{array}{l} A_2 \subseteq A_1 \\ \text{(equivalent to } B_1 \subseteq B_2\text{)} \end{array} \right.$$

The whole set of formal concepts fitted out by the order relation $\leq$ is called *concept lattice* or *Galois lattice* because it verifies the lattice properties: the relation $\leq$ is clearly an order relation, and for each pair of concepts (A_1, B_1) and (A_2, B_2), there exists the greatest lower bound (resp. the least upper bound) called *meet* (resp. *join*) denoted $(A_1, B_1) \wedge (A_2, B_2)$ (resp. $(A_1, B_1) \vee (A_2, B_2)$) defined by:

$$(A_1, B_1) \wedge (A_2, B_2) = (g(B_1 \cap B_2), (B_1 \cap B_2)) \quad (1)$$

$$(A_1, B_1) \vee (A_2, B_2) = ((A_1 \cap A_2), f(A_1 \cap A_2)) \quad (2)$$

The concepts $\perp = (O, f(O))$ and $\top = (g(I), I)$ respectively correspond to the bottom and the top of the concept lattice.

In *formal concept analysis* (FCA) concept lattices are used to analyse data when organised by a binary relation between objects and attributes. See the book of Ganter and Wille [4] for a more complete description of formal concept analysis.

In the following, we abuse notation and use $X + x$ (respectively, $X \setminus x$) for $X \cup \{x\}$ (respectively, $X \backslash \{x\}$).

2.2 Generation algorithms of a concept lattice

Numerous generation algorithms for concept lattices have been proposed in literature [6,7,5,8]. Although all these algorithms generate the same lattice, they propose different strategies. Some of these algorithms are incremental [6,9]. Ganter's NextClosure [7] is the reference algorithm that determines the concepts in lectical order (next, the concepts may be ordered by $\leq$ to form the concept lattice) while Bordat's algorithm [5] is the first algorithm that computes directly the Hasse diagram of the lattice. Recent work [10] proposed a generic algorithm unifying the existing algorithms in a unique framework, which makes easier the comparison of these algorithms. A formal and experimental comparative study of the different algorithms has been published [11].

All of these proposed algorithms have a polynomial complexity with respect to the number of concepts (at best quadratic in [8]). The complexity is therefore determined by the size of the lattice, this size being bounded by $2^{|O+I|}$ in the worst case and by $|O + I|$ in the best case. Studies on average complexity are difficult to carry out because the size of the concept lattice depends both on the dimensionality of the data to classify and on their organization and diversity. However, in practice the size of the Galois lattice generally remains reasonable.

Some applications offer to only generate some concepts from the huge amount of available data. Bordat's algorithm [5] is the more appropriate since it generates the cover relation between concepts, and thus allows an on-demand generation of concepts usually used in concrete applications. Bordat's algorithm is issued from a corollary of Bordat's theorem:

Theorem 1 (Bordat [5]). *Let (A, B) and (A', B') be two concepts of a context (O, I, R). Then $(A, B) \prec (A', B')$ if and only if A' is inclusion-maximal in the*

following set system $\mathcal{F}_A$ *defined on* O[1] *:*

$$\mathcal{F}_A = \{g(x + B) \ : \ x \in I \backslash B\} \tag{3}$$

Corollary 2 (Bordat [5]). *Let* (A, B) *be a concept. There is a one-to-one mapping between the immediate successors of* (A, B) *in the Hasse diagram of the lattice and the inclusion-maximal subsets of* $\mathcal{F}_A$*.*

Bordat's algorithm recursively computes all the concepts of $\mathbb{C}$ by computing immediate successors for each concept (A, B), starting from the bottom concept $\perp = (f(G), G)$, until all concepts are generated. Immediate successors are generated using Corollary 2 in $O(|I|^2 * |O|)$: the set system has first to be generated in a linear time ; then inclusion-maximal subsets of $\mathcal{F}_B$, can easily be computed in $O(|I|^2 * |O|)$.

3 Limited Object Access Algorithm (LOA)

3.1 Description of the LOA Algorithm

Large data are often described by a huge amount of objects, as in databases for example where the number of recordings (i.e. objects) can be huge, indexed using sophisticated key-indexation techniques. In this section, we describe our Limited Object Access algorithm (LOA algorithm), an extension of Bordat's immediate successors generation with a limited object access. This algorithm, compounded with an on-demand strategy aims to improve time computation for large amount of objects.

Our algorithm considers the restriction of a concept lattice to the attributes. A nice result establishes that any concept lattice $(\mathbb{C}, \leq_C)$ is isomorphic to the lattice $(\mathbb{C}_I, \subseteq)$ defined on the set I of attributes, with $\mathbb{C}_I$ the restriction of $\mathbb{C}$ to the attributes in each concept. The lattice $(\mathbb{C}_I, \subseteq)$ is also known as the closed sets lattice on the attributes I of a context (O, I, R), where the set system $\mathbb{C}_I$ is composed of all closed set - i.e. fixed points - for the closure operator $\varphi = g \circ f$. See the survey of Caspard and Monjardet [12] for more details about closed set lattices.

Using the closed sets lattice $(\mathbb{C}_I, \subseteq)$ instead of the whole concept lattice $(\mathbb{C}, \leq_C)$ gives raise to a storage improvement, for example in case of large amount of objects.

A closed sets lattice can be generated using an algorithm similar to Bordat's algorithm, and therefore enabling an on-demand generation in order to reduce the whole amount of closed sets. This algorithm (see Alg. 1) recursively computes immediate successors (see Alg. 2) of a closed set B, starting from the bottom closed set $\perp = \varphi(\emptyset)$, until I is generated.

The `Immediates_Successors_LOA` algorithm we propose (see Alg. 3) reinforces the object access limitation by considering the cardinality of the subset $g(X + B)$ instead of the subset itself to compute the inclusion-maximal subsets of $\mathcal{F}_A$ using the following property:

[1] In [5], the equivalent formulation $g(x) \cap A$ is used instead of $g(x + B)$

Proposition 3. *Consider a concept* (A, B)*, and two subsets* X *and* Y *of attributes in* $B \backslash I$*. Then*

$$g(X + B) \subseteq g(Y + B) \iff |g(X + B)| = |g(X + Y + B)| \tag{4}$$

This proposition is a direct consequence of the two following remarks:

1. The equivalence between inclusion and intersection set operations ($C \subseteq D \iff C = C \cap B$) allows to deduce the equivalence between $g(X + B) \subseteq g(Y + B)$ and $g(X + B) = g(X + B) \cap g(Y + B)$:
2. Then, by definition of g, we have $g(X + B) \cap g(Y + B) = g(X + Y + B)$.

More precisely, the `Immediates_Successors_LOA` algorithm (see Alg. 3) first initialize the set *Succ* of immediate successors of a closed set B with the emptyset. The set *Succ* is then updated by considering each attribute x of $I \backslash B$ and another already inserted potential successor $X \subseteq I \backslash B$ by considering the following four cases, where $c_B(Y)$ denotes the cardinality of $g(B + Y)$ for a Y of attributes:

Merge x with X: When $g(x + B) = g(X + B)$, then x and X belongs to the same closed set, and thus have to be merged in a same potential successor of B. By Proposition 3, this case is tested by $c_B(X + x) = c_B(X)$ and $c_B(X) = c_B(x)$.

Eliminate X: When $g(X + B) \subset g(x + B)$, then the closed set containing X isn't inclusion-maximal in $\mathcal{F}_A$, and thus hasn't to be considered as a potential successor of B. By Proposition 3, this case is tested by $c_B(X + x) = c_B(X)$ and $c_B(X) < c_B(x)$.

Eliminate x: When $g(x + B) \subset g(X + B)$, then the closed set containing x isn't inclusion-maximal if $\mathcal{F}_A$, and thus hasn't to be considered as a potential successor of B. By Proposition 3, this case is tested by $c_B(X + x) = c_B(X)$ and $c_B(x) < c_B(X)$.

Insert x: When x is neither eliminated or merged with X, then it is added as a potential successor of B ; another attribute is then considered.

3.2 Example

To illustrate our algorithm, we use the following context where numbers from 1 to 9 are described by some properties: the number is a prime number, an odd or even number, a square, a composite number or a factorial number.

	(p)rime	o(dd)	(e)ven	(s)quare	(c)omposite	(f)actorial
nb 1		×		×		×
nb 2	×		×			×
nb 3	×	×				
nb 4			×	×	×	
nb 5	×	×				
nb 6			×		×	×
nb 7	×	×				
nb 8			×		×	
nb 9		×		×	×	

Name: `Closed_Set_Lattice`
Data: A context $K = (O, I, R)$
Result: The Hasse diagram $(\mathbb{C}_I, \prec)$ of the lattice $(\mathbb{C}_I, \subseteq)$
begin
  $\mathbb{C}_I = \{f(O)\}$;
  foreach $B \in \mathbb{C}_I$ *not marked* **do**
    $Succ_B$=`Immediates_successors` (K, B);
    foreach $X \in Succ_B$ **do**
      $B' = B + X$;
      if $B' \notin \mathbb{C}_I$ **then** add B' to $\mathbb{C}_I$;
      add a cover relation $B \prec B'$
    end
    mark B
  end
  return $(\mathbb{C}_I, \prec)$
end

Algorithm 1: Generation of the Hasse diagram of the closed set lattice $(\mathbb{C}_I, \subseteq)$

Name: `Immediates_Successors`
Data: A context K ; A closed set B of the closed set lattice $(\mathbb{C}_I, \subseteq)$ of K
Result: The immediate successors of B in the lattice
begin
  initialize the set system $\mathcal{F}_A$ with $\emptyset$;
  foreach $x \in I \backslash B$ **do**
    add $g(x + B)$ to $\mathcal{F}_A$
  end
  $Succ$=maximal inclusion subsets of $\mathcal{F}_A$;
  return $Succ$
end

Algorithm 2: Generation of the immediate successors of a closed set in the Hasse diagram of the lattice $(\mathbb{C}_I, \subseteq)$

```
Name: Immediates_Successors_LOA
Data: A context K ; A closed set B of the closed set lattice (C_I, ⊆) of K
Result: The immediate successors of B in the lattice
begin
    initialize the Succ_B family to an empty set;
    foreach x ∈ I \ B do
        add = true;
        foreach X ∈ Succ_B do
            \\ Merge x and X in the same potential successor
            if c_B(x) = c_B(X) then
                if c_B(X + x) = c_B(x) then
                    replace X by X + x in Succ_B;
                    add=false; break;
                end
            end
            \\ Eliminate x as potential successor
            if c_B(x) < c_B(X) then
                if c_B(X + x) = c_B(x) then
                    add=false; break;
                end
            end
            \\ Eliminate X as potential successor
            if c_B(x) > c_B(X) then
                if c_B(X + x) = c_B(X) then
                    delete X from Succ_B
                end
            end
        end
        \\ Insert x as a new potential successor ;
        if add then  add {x} to Succ_B
    end
    return Succ_B;
end
```

Algorithm 3: Generation of the immediate successors of a closed set in the Hasse diagram of the lattice $(\mathbb{C}_I, \subseteq)$

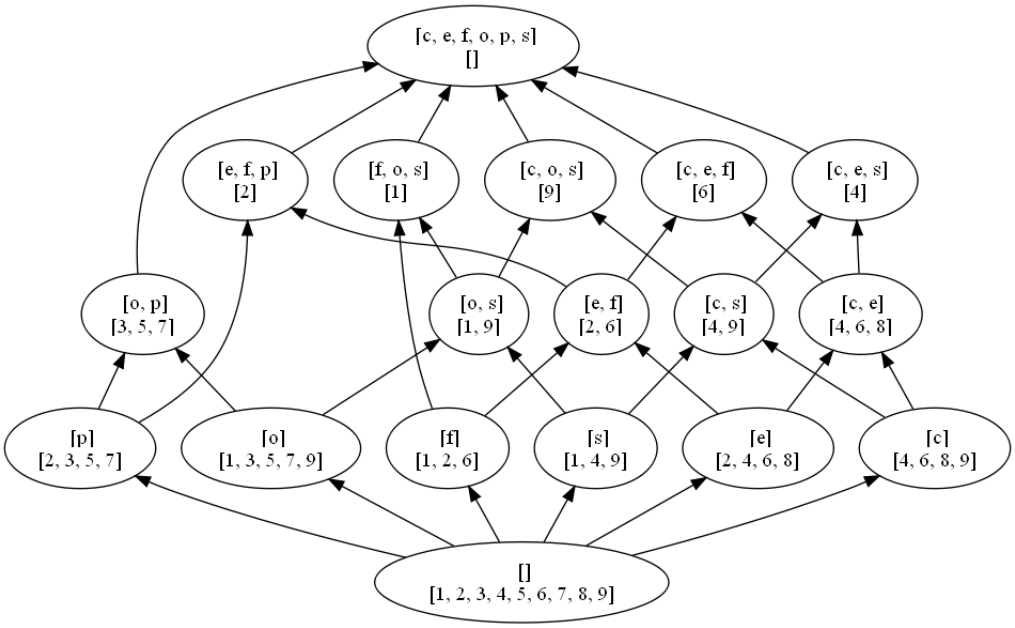


Fig. 1. Concept lattice

Figure 1 gives the concept lattice of this context. When the algorithm computes the successors of the closed sets e (resp. p), it proceeds as described in TABLE 1 (resp. TABLE 2). The different steps of these two examples show the different actions taken by the algorithm.

$Succ_F$	x	$c_B(x)$	X	$c_B(X)$	$c_B(x+X)$	Case	Action
$\emptyset$	p	1					Insert $[p]$
$\{[p]\}$	o	0	$[p]$	1	0	$c_B(x+X) = c_B(x) < c_B(X)$	Eliminate $[o]$
$\{[p]\}$	s	1	$[p]$	1	0	$c_B(x+X) < c_B(X) = c_B(x)$	
$\{[p]\}$	c	3	$[p]$	1	0	$c_B(x+X) < c_B(X) < c_B(x)$	Insert $[c]$
$\{[p],[c]\}$	f	2	$[p]$	1	1	$c_B(x+X) = c_B(X) < c_B(x)$	Eliminate $[p]$
$\{[c]\}$	f	2	$[c]$	3	1	$c_B(x+X) < c_B(x) < c_B(X)$	Insert $[f]$
$\{[c],[f]\}$							

Table 1. Immediate successors of $[e]$

3.3 Complexity

The complexity of computing the immediate successors of a closed set B using the `Immediates_Successors_LOA` algorithm is:

$$\frac{(|I| - |B|)(|I| - |B|)}{2} * O(c_B(X))$$

$Succ_F$	x	$c_B(x)$	X	$c_B(X)$	$c_B(x+X)$	Case	Action
$\emptyset$	o	3					Insert $[o]$
$\{[o]\}$	e	1	$[o]$	3	0	$c_B(x+X) < c_B(x) < c_B(X)$	Insert $[e]$
$\{[o],[e]\}$	s	0	$[o]$	3	0	$c_B(x+X) = c_B(x) < c_B(X)$	Eliminate $[s]$
$\{[o],[e]\}$	c	0	$[o]$	3	0	$c_B(x+X) = c_B(x) < c_B(X)$	Eliminate $[c]$
$\{[o],[e]\}$	f	1	$[o]$	3	0	$c_B(x+X) < c_B(x) < c_B(X)$	
$\{[o],[e]\}$	f	1	$[e]$	1	1	$c_B(x+X) = c_B(x) = c_B(X)$	Merge $[e]$, $[f]$
$\{[o],[ef]\}$							

Table 2. Immediate successors of $[p]$

which leads to

$$O((|I| - |B|)^2 * O(c_B(X)))$$

using the big O notation.

This has to be compared with $O(|I|^2 * |O|)$ of the `Immediates_Successors` algorithm. In addition the cost $O(C_B(x))$ of computing the cardinality of objects satisfying the required properties can be based on multiple keys and robust algorithms used in databases that do not need to load all data for computing a cardinality.

3.4 Experimentations

In the experiment, we use a dataset composed of:

54 attributes: the 6 attributes of the example, and attributes corresponding to the property to be multiple of $\{3 - 50\}$.
100000 objects: the integers between 1 and 100000

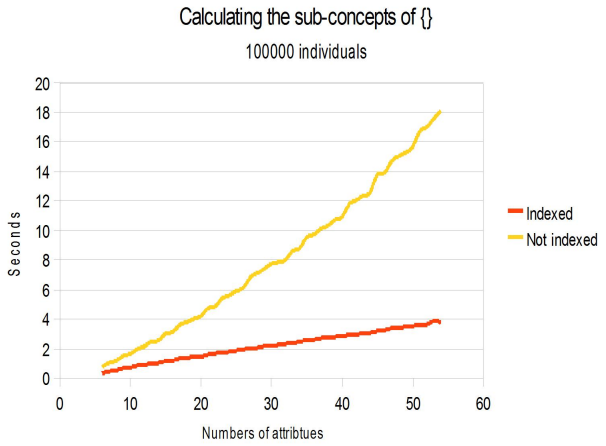
The dataset is stored in a database MySQL 5.5.14. We have implemented our `Immediates_Successors_LOA` algorithm using PhP 5.3.6. The counting of objects satisfying a set of properties is realised by the SQL request comparing indexes with a constant:

```
select count (*) from att1=1 and att2=1
```

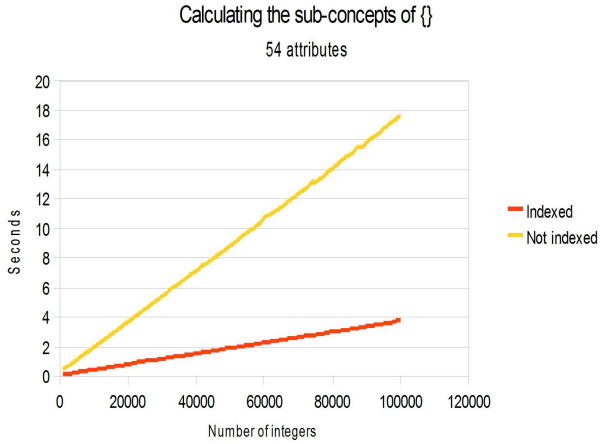
We compare the processing time of our `Immediates_Successors_LOA` algorithm in the two following cases:

Indexed: Each attribute is defined to be an index. Objects are indexed by their attributes, and MySQL can quickly retrieve them in the dataset using a B-tree indexation with a logarithmic complexity [13]: $O(c_B(X)) = O(log|I|)$.
Not indexed: Objects are not indexed and a scan of all the lines is necessary to retrieve objects. The complexity is then similar to those of the `Immediates_Successors` algorithm: $O(c_B(X)) = O(|I| - |B|)$.

We compare the processing time of computing the immediate successors of the botom element $\emptyset$ in this two cases (indexed and not indexed):



(a) 100000 first integers as objects ; a number of attributes between 6 to 54.



(b) 54 attributes ; integers between 1000 and 100000

Fig. 2. Calculating the immediate successors of $\emptyset$

Fig.2(a): for the 100000 first integers as objects, and a number of attributes between 6 to 54.

Fig.2(b): for the 54 attributes, and integer between 1000 and 100000.

In the results, the time processing is really improved with an indexed dataset, and seems to be near to linear in $O(|I| + |O|)$. Immediate successors of the $\emptyset$ for 100000 objects and 54 attributes are computed in 3 seconds with the indexed algorithm, and in 18 seconds with the not indexed one.

Moreover, the `explain` of the count operation shows that an `index-merge` operation is realized on indexes corresponding to an `intersect` computation:

```
mysql> explain select count(*) from numbers where p=1 and o=1;
+----+-------------+------+---------+------+----------------------+
| id | select_type | key  | key_len | rows | Extra                |
+----+-------------+------+---------+------+----------------------+
|  1 | index_merge | p,o  | 1,1     |    2 | Using intersect(p,o); |
|    |             |      |         |      | Using where;         |
|    |             |      |         |      | Using index          |
+----+-------------+------+---------+------+----------------------+
1 row in set (0.00 sec)
```

Therefore, optimizing the intersection operation, with an adaptated sort on the lines for example, would be a possible optimization of our algorithm.

4 Conclusion

In this paper, we described a new algorithm for computing the immediate successors of a concept using the counting of objects satisfying a set of properties. By separating the counting from the rest of the algorithm, new systems for exploring concept lattices can now rely on optimization algorithms used in relational databases. If the tests we will realize on PostgreSQL and MySQL databases are successfull in terms of manipulating a huge amounts of data, we plan to propose a library for extending content management system such as Joomla!.

References

1. Birkhoff, G.: Lattice theory. 3d edn. American Mathematical Society (1967)
2. Barbut, M., Monjardet, B.: Ordres et classifications : Algèbre et combinatoire. Hachette, Paris (1970) 2 tomes.
3. Davey, B., Priestley, H.: Introduction to lattices and orders. 2nd edn. Cambridge University Press (1991)
4. Ganter, B., Wille, R.: Formal Concept Analysis, Mathematical foundations. Springer Verlag, Berlin (1999)
5. Bordat, J.: Calcul pratique du treillis de Galois d'une correspondance. Math. Sci. Hum. **96** (1986) 31–47
6. Norris, E.: An algorithm for computing the maximal rectangles in a binary relation. Revue Roumaine de Mathématiques Pures et Appliquées **23** (1978)
7. Ganter, B.: Two basic algorithms in concept analysis. Technische Hochschule Darmstadt (Preprint 831) (1984)
8. Nourine, L., Raynaud, O.: A fast algorithm for building lattices. Information Processing Letters **71** (1999) 199–204
9. Gödin, R., Missaoui, R., Alaoui, H.: Incremental concept formation algorithms based on Galois (concept) lattices. Computational Intelligence **11** (1995) 246–267

10. Gely, A.: A generic algorithm for generating closed sets of binary relation. Third International Conference on Formal Concept Analysis (ICFCA 2005) (2005) 223–234
11. Kuznetsov, S., Obiedkov, S.: Comparing performance of algorithms for generating concept lattices. Journal of Experimental and Theorical Artificial Intelligence **14** (2002) 189–216
12. Caspard, N., Monjardet, B.: The lattice of closure systems, closure operators and implicational systems on a finite set: a survey. Discrete Applied Mathematics **127** (2003) 241–269
13. Bayer, R. et McCreight, E.M.: Organization and maintenance of large ordered indexes. Acta Informatica **1** (1972) 173–189