A Fast Algorithm for Building the Hasse Diagram of a Galois Lattice

Petko Valtchev, Rokia Missaoui, Pierre Lebrun
Département d'Informatique, UQAM, C.P. 8888, succ. "Centre Ville",
Montréal, Québec, Canada, H3C 3P8
e-mail : {valtchev, missaoui, lebrun}@info.uqam.ca

Abstract

Formal concept analysis and Galois lattices in general are increasingly used for large contexts that are automatically generated. As the size of the resulting datasets may grow considerably, it becomes essential to keep the algorithmic complexity of the analysis procedures as low as possible. This paper presents an efficient algorithm that computes the Hasse diagram of a Galois lattice from the lattice ground set, i.e., the set of all concepts. The algorithm performs an element-wise completion of the lattice according to a linear extension of the lattice order. This requires only a limited number of comparisons between concepts and therefore makes the global algorithm very efficient. In fact, its asymptotic time complexity is almost linear in the number of concepts. Consequently, the joint use of our algorithm with an efficient procedure for concept generation yields a complete procedure for building the Galois lattice.

Résumé

L'analyse formelle de concepts et, plus généralement, la construction de treillis de Galois sont de plus en plus utilisées dans des circonstances où les données sont générées de manière automatique. La taille des jeux de données pouvant croître considérablement, il est important d'assurer une complexité algorithmique raisonnable pour les procédures d'analyse. Notre papier présente un algorithme efficace qui permet de construire le diagramme de Hasse d'un treillis de Galois en partant de l'ensemble des concepts (nœuds). La stratégie adoptée par l'algorithme est une complétion élément par élément du treillis basée sur une extension linéaire de l'ordre dans le treillis. Elle ne demande qu'un nombre très réduit de comparaisons entre concepts de sorte que l'algorithme entier reste très efficace. Ainsi, la complexité asymptotique de l'algorithme est quasiment linéaire selon le nombre de concepts dans le treillis. En conséquence, notre algorithme peut avantageusement compléter une procédure de génération de concepts au sein d'un module efficace d'analyse formelle.

1 Introduction

Concept analysis has proven to be a valuable tool for gaining insight into complex data [10, 13, 9]. In many applications of concept analysis, experts learn from formal contexts by inspecting their carefully layouted concept lattices. Contexts in these applications tend to have a modest size because otherwise the resulting concept lattices are hard to analyze visually. The algorithmic complexity of concept analysis for these applications is consequently a minor concern. But concept analysis is used increasingly for applications like program analysis inside a compiler [16] or data analysis within a large database [11] where large contexts are constructed automatically. The resulting concept lattice is no longer inspected visually but is considered as a data structure that is part of the program. For these applications, the algorithmic complexity of concept analysis does matter. This paper presents an efficient algorithm for computing the concept lattice from its ground set and provides some analytical complexity results.

We first present the basics of the Galois lattices and formal analysis (Section 2). Then, we provide a row of structural properties of the Galois lattice that help devise an efficient procedure for its construction (Section 3). Next, an algorithm is sketched that implements the suggested strategy in a straightforward manner (Section 4). Finally, some improvements that make the algorithm computational cost decrease are described (Section 5).

2 Formal concept analysis and Galois lattices

Formal concept analysis (FCA) [9] is an approach towards the extraction of highly similar groups of objects from a collection O of objects described by a set of attributes A. The paradigm occurred within the lattice theory [2]: the attributes considered represent binary features, i.e., with only two possible values, *present* or *absent*. In this framework, the discovered groups represent the closed sets of the *Galois connection* [1] induced by I on the couple O and A.

2.1 Basics

The objects in O can be seen as binary vectors and the dataset as a binary table with an incidence relation I (oIa means that the object o has the attribute a). The table (O, A, I), further called *formal context* or simply context, may be processed to extract the natural groupings and relationships between objects and attributes. For that reason, two set-valued functions are defined: f extracts common attributes from a set of objects whereas g computes the dual for attribute sets. Formally, the functions are defined as follows:

- $f : \mathcal{P}(O) \to \mathcal{P}(A)$, $f(X) = \{a \in A | \forall o \in X, oIa\}$
- $g : \mathcal{P}(A) \to \mathcal{P}(O)$, $g(Y) = \{o \in O | \forall a \in Y, oIa\}$

Both functions constitute a Galois connection between $\mathcal{P}(O)$ and $\mathcal{P}(A)$. The closed subsets of both O and A constitute two lattices with respect to the set inclusion. Both lattices are isomorphic, they can be merged into a common structure made up of couples (X, Y) of an object set $X \in \mathcal{P}(O)$ and an attribute set $Y \in \mathcal{P}(A)$. The couples, called *concepts*, are complete couples with respect to I, in the sense that $Y = f(X)$ and $X = g(Y)$. In the FCA framework, X is referred to as the concept *extent* and Y as the concept *intent*. The set $\mathcal{C}_{\mathcal{K}}$ of all concepts over a context $\mathcal{K} = (O, A, I)$ constitute a lattice $\mathcal{L}$ with respect to $\leq_{\mathcal{L}}$, the product of both set inclusions, $\subseteq_O$ and $\supseteq_A$:

- $(X_1, Y_1) \leq (X_2, Y_2) \Leftrightarrow X_1 \subseteq X_2$.
- $(X_1, Y_1) \leq (X_2, Y_2) \Leftrightarrow Y_2 \subseteq Y_1$.

The lattice $\mathcal{L} = \langle \mathcal{C}, \leq_{\mathcal{L}} \rangle$ is called *Galois lattice* [1] or *formal concept lattice* [9]. Lattice operators *join* and *meet* provide the least upper bound (LUB) and the greatest lower bound (GLB) in the concept lattice respectively. They are defined as follows:

	a	b	c	d	e	f	g	h	i
1	x		x			x		x	
2	x		x				x		x
3	x			x			x		x
4		x	x			x		x	
5		x			x		x		

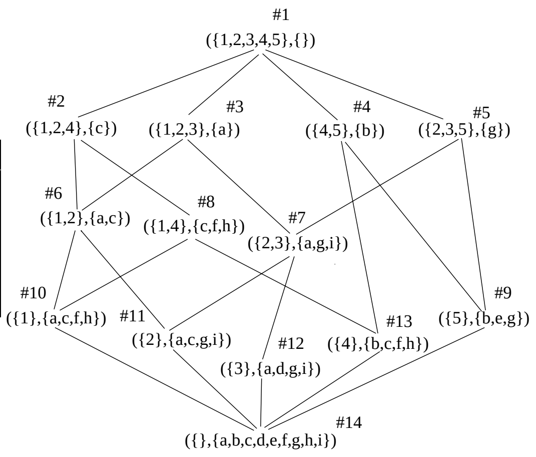

Figure 1: A binary relation and its Galois or concept lattice.

- $(X_1, Y_1) \vee (X_2, Y_2) = (g \circ f(X_1 \cup X_2), Y_1 \cap Y_2)$
- $(X_1, Y_1) \wedge (X_2, Y_2) = (X_1 \cap X_2,\ f \circ g(Y_1 \cup Y_2))$

For conciseness reasons, both basic operations will be denoted by a quote ('), and the closure operators by a double quote ("). Thus, X' and Y' will stand for $f(X)$ and $g(Y)$ respectively, whereas X" and Y" will stand for $g \circ f(X)$ and $f \circ g(Y)$. These notations reflect the symmetric role of both initial operations.

Furthermore, given an object o, there is always a most specific concept whose extent contains o. Dually, for each attribute a, there is a most general concept whose intent includes a. These concepts will be further referred to as *object concept* and *attribute concept* respectively.

2.2 Computing the lattice nodes only

The concepts of a context may be seen as maximal rectangles in the table of the binary relation which represents the context. These are called *complete primary sub-matrices* of the global matrix. From a graph-theoretical viewpoint, the concepts represent complete maximal sub-graphs of the bipartite graph which is described by the context. The wide range of problems that may be reduced to the discovery of the closed sets of a Galois connection explains the variety of the existing algorithms. A detailed description of those algorithms is out of the scope of our paper. Interested readers may refer to [12] for a very comprehensive presentation. We only sketch the basic principles of those algorithms whereas a deeper, performance-oriented comparison may be found in [10].

The first algorithm that may be used to discover the concept set (see Chein [5]) comes from the linear algebra. It generates the concepts in an iterative manner, starting by the most specific ones, i.e., the object concepts. At each step, new concepts are generated as couples where the intent is the intersection of the intents of two already existing concepts. As there are numberous ways of obtaining a concept as the join of two other concepts, the algorithm performs a lot of redundant generations which consists a major drawback.

Another algorithm that only computes the concept set has been proposed by Norris [15]. The underlying principle is an incremental generation of all the concepts over the sub-tables made up of the first i rows, i.e., attributes. The algorithm iterates over i: it starts by the first row and generates, at each step, the possible "enhancements" of the already generated concepts with the entries of the i-th row.

The algorithm suggested by Ganter [7] is undoubtfully the most sophisticated one as it uses a deeper insight in the structure of the concepts to speed-up the computation. Actually, in order to avoid the most expensive part of the concept generation, i.e., the lookup for redundant generations, the algorithm uses a specific order on the concept sets called *lectic*. The concepts are generated according to the lectic order which is total and therefore, each concept is computed only once. The main drawback of the approach, as in the two previous cases, remains the lack of structure over the concept set.

2.3 Computing both the nodes and the Hasse diagram

The Bordat algorithm [3] generates both the concept set and the Hasse diagram of the lattice. The structural properties of the precedence relation between concepts are used to generate the concepts in the appropriate order. Thus, from each concept the algorithm generates its upper covers. The drawback is that a concept is generated a number of times that corresponds to the number of its lower covers (one concept generation per link in the Hasse diagram).

Finally, Godin *et al.* [10] suggested an incremental procedure that maintains lattice structure, i.e., both concept set and Hasse diagram, upon the insertion of a new object into the table. The principle of the algorithm consists of locally modifying the lattice structure (insertion of new concepts, completion of existing ones, deletion of redundant links, etc.) while keeping large parts of the lattice untouched. The procedure may be applied to both incremental and non-incremental concept lattice problems.

2.4 What is missing?

When asymptotic complexity is only considered, the Ganter algorithm shows the best score since it only generates every concept once. However, the algorithm requires an additional effort to build the diagram, a post-processing which may significantly increase the overall complexity. In fact, a naive algorithm that builds the precedence relation of a partially ordered set from its order relation, has a complexity which is cubic in the size of the ground set (the number of concepts here). This exceeds by far the complexity of the main concept building algorithms and, finally, makes all optimization efforts over these algorithms useless. In addition, the expensive diagram building prevents sensible performance-based comparisons between algorithms that compute only the concept set and those which build the diagram. In particular, it is impossible to correctly asset the incremental strategy versus the batch ones.

In a recent paper, Ganter *et al.* made a suggestion [8] about how the Hasse diagram may be efficiently extracted from the concept set. The basic idea is to search for minimal concepts among all the super-concepts of a given concept. The super-concepts are dynamically generated and compared. The idea has been further developed by Lindig [14] who presented an effective algorithm that computes both the concept set and the diagram.

We choose another direction based on a dynamic view of the lattice as a construct which is gradually completed by linking one concept at a time. The completion is done in a top-down manner, starting from the $\top$ node and processing the rest of the nodes according to a total order which is a linear extension of the lattice order. At each step, the current element is connected to each of its immediate successors in the final lattice, further called upper covers. An important feature of such a procedure is that at the time a node is processed, all its upper covers are already completely integrated in the partial structure.

3 Basic facts about precedence relation in $\mathcal{L}$

In the following we use some basic notions of the lattice and partially ordered set theory that we shall not define here. As the vocabulary used is consistent with the excellent work of Davey and Priestley [6], the interested readers may refer to their book.

3.1 Auxiliary definitions

In the following we define the border of a partially constructed lattice. First, the lattice construction is achieved through an element-wise insertion of the lattice nodes. The construction is supported by a structure that, once the algorithm has finished its work, contains the entire lattice. At any time point, the structure represents a partial order (poset) that is a sub-order of the lattice $\mathcal{L}$.

Definition 3.1. *Let P_i denote the partial ordered set represented by the structure after the insertion of the i-th node. Clearly, $P_0 = \langle \emptyset, \emptyset \rangle$, the empty lattice, and $P_l = \mathcal{L}$, where $l = \|\mathcal{C}\|$.*

In the following, the poset P_i will not be distinguished from its ground set. Of course, the poset P_i depends on the particular (linear) order $\leq^e$ in which the nodes are inserted. Our approach relies heavily on some interesting properties of P_i that follow from the choice of $\leq^e$. In fact, several facts concerning P_i may be observed. First, when the order $\leq^e$ is a linear extension of the lattice order $\leq_{\mathcal{L}}$, i.e., both orders match on common couples (noted $\leq_{\mathcal{L}} \Subset \leq^e$), then the set P_i is an *upper set* of $\mathcal{L}$.

Property 3.1. *Whenever $\leq_{\mathcal{L}} \Subset \leq^e$, $\uparrow_{\mathcal{L}} P_i = P_i$ for all i.*

In other words, with every element c already in P_i, the poset includes its upper bounds as well. Of course, we can assume the elements in P_i are correctly linked with respect to $\leq_{\mathcal{L}}$, so that P_i is the sub-order of $\mathcal{L}$ *induced* by its elements (no links are missing). Moreover, the above property may be extended to c_{i+1} ($i \leq l \Leftrightarrow 1$), i.e., the element in $\mathcal{C}$ which is to be integrated into P_i on the next step of the construction.

Property 3.2. *Whenever $\leq_{\mathcal{L}} \Subset \leq^e$, $\forall c \in \mathcal{C}$, $c_i \leq_{\mathcal{L}} c$ implies $c \in P_i$, for all $i \leq l \Leftrightarrow 1$.*

Our attention is drawn by the elements that cover c_i in $\mathcal{L}$ since they have to be physically connected to c_i in order to achieve their integration into P_{i-1}. These elements will be further referred to as *upper covers*, as opposed to the *lower covers* which are the elements covered by c_i.

Definition 3.2. *The set of upper covers of c_i in $\mathcal{L}$ is $c_i^{\nabla} = \{c | c_i \prec_{\mathcal{L}} c\}$.*

Clearly, if a linear extension of $\leq_{\mathcal{L}}$ is used, the entire set is already in P_i. In the following, we assume $\leq_{\mathcal{L}} \Subset \leq^e$ so that our aim will be to devise an algorithm that efficiently detects c_i^{∇} within P_{i-1}.

3.2 Where are the upper covers?

As we noted previously, all the upper covers of a c_i are already in P_{i-1}. With a similar reasoning, one may show that P_{i-1} contains no lower bound of c_i (due to the linear order). This means that c_i is a minimal element of P_i. One may expect, that with some tricky choice of $\leq^e$, all the upper covers of c_i remain minima of P_i, a fact that would greatly simplify their look-up. Unfortunately, a general definition of such an order $\leq^e$ is impossible. However, the idea may be re-tailored in order to fit to the general case of non-minimal upper covers. For that reason, we first define the set of minima in P_i as the "border" of the current lattice construction.

Definition 3.3. *The* border *of P_i is the set of its minima:*

$$\bot(P_i) = \{c \in P_i | \forall c' \in P_i, c' \leq_{P_i} c \ \Rightarrow \ c' = c\}.$$

First, the border is a dynamic structure that changes with P_i, i.e., depends on $\leq^e$. Its evolution between two construction steps i and $i+1$ can be simply formulated: the new border always includes the new element, c_i whereas all elements of the old border that are greater than c_i are dropped out.

Property 3.3.

$$\bot(P_i) = \{c_i\} \cup (P_{i-1} \Leftrightarrow \uparrow_{P_{i-1}} c_i).$$

Next, the set constitutes an *anti-chain* in the entire lattice $\mathcal{L}$. Observe that the set c_i^{∇} is split between the border $\bot(P_{i-1})$ and the inner part of P_{i-1} , i.e., the rest of the set. Moreover, all the elements to drop from the border upon joining c_i are, in fact, upper covers of c_i.

Property 3.4.

$$(P_{i-1} \Leftrightarrow \uparrow_{P_{i-1}} c_i) \subseteq c_i^{\nabla}.$$

As we have noticed earlier, the above set inclusion is strict in the general case. Thus, a simple test of comparability between c_i and the members of $\bot(P_i)$ is not enough to fix c_i^{∇}: some upper covers will lay "higher" in P_{i-1}. The question is how to recognize these nodes that are "shaded" by elements in the border that are inferior to them.

3.3 Which are the shaded covers?

To devise a criterion for upper covers that are not border elements, we need some basic facts. First, for each shaded cover, say c_s, a border element c_b exists which is inferior to the cover, $c_b \leq_{P_{i-1}} c_s$. This follows trivially from the definition of the border. A less trivial fact is that the shaded cover is exactly the join of c_b and the current element c_i. The general proposition may be formulated as follows[1]:

Property 3.5.

$$\forall c_u \in c_i^{\nabla}, \ \exists c_b \in \bot(P_{i-1}), c_b \vee_{\mathcal{L}} c_i = c_u.$$

The above proposition does not yield a recognition criterion alone, but provides its basis. It says all upper covers could be found as joins of the new node and a border element. Thus, a possible search strategy would be to look for all those GLBs, an approach that would require a considerable amount of graph search. In addition, one

[1]Observe that in the degenerated case, c_u and c_b are the same.

has to check the minimalness of each join since not all of them need to be upper covers (the condition is necessary but not sufficient).

Fortunately, we do not need to perform such an exhaustive search. At this point, we may apply a basic result from the Galois lattice theory that helps limit the search. Recall that the intent of a *join* of two concepts is the intersection of both intents.

Property 3.6. *Given two concepts* (X_1, Y_1) *and* (X_2, Y_2) *in* $\mathcal{L}$*, their LUB in* $\mathcal{L}$ *is*

$$(X_1, Y_1) \vee_{\mathcal{L}} (X_2, Y_2) = ((Y_1 \cap Y_2)', Y_1 \cap Y_2).$$

Thus, a possible strategy for the discovery of upper covers could be to generate all the intents of those concepts and to use the intents to localize the nodes themselves within P_{i-1}. The generation is a simple intersection between the intents. Observe that the minimalness constraint holds also on intents, therefore only intents that are "minimal" with respect to the lattice order $\leq_{\mathcal{L}}$ are sought. This means, as $\leq_{\mathcal{L}}$ follows the inverse of the inclusion between intents, $\supseteq$, we are looking for intents that are maximal for $\subseteq$. The total amount of knowledge about the P_{i-1} structure that is necessary for the correct integration of c_i is summarized in the following lemma.

Lemma 3.1. *For an arbitrary* $\leq_e$*, a linear extension of* $\leq_{\mathcal{L}}$*, and* $i \in [1..l]$*, let* $c_i = (X_i, Y_i)$ *be the element of the i-th rank in* $\mathcal{C}$ *and* P_{i-1} *the partial order induced by the first* $i \Leftrightarrow 1$ *elements of* $\mathcal{C}$*. Let also the set* D_r *be the set of all intents of upper covers of* c_i *in* $\mathcal{L}$*,* $D_r = \{Y_j | (X_j, Y_j) \in c_i^{\nabla}\}$*. Finally, let* D_c *be the set of all maxima of the set of intersections between* Y_i *and an intent of a border concept,* $D_c = \{Y_b | Y_b = Y_i \cap Y_j,\ (X_j, Y_j) \in \perp(P_{i-1})\}$*. Then the two sets are identical,* $D_r = D_c$*.*

Once the set of all upper cover intents has been extracted from the set of all generated intents, the task of finding the actual nodes in P_{i-1} could be done by a simple look-up. It is noteworthy that the real gain of using the intents instead of looking for nodes is that the comparability between concepts (nodes) can be established in a far faster way.

4 A straightforward implementation of the diagram building

From the remarks of the previous paragraph, a quite simple algorithm could be drawn.

4.1 The algorithm

The procedure has three steps: the first one generates the intents of all LUBs, the second finds all maximal sets among the intents and the third one first finds all nodes in P_i that correspond to maximal intents and then connects the current node to them. Of course, an additional effort is necessary to maintain the list of border elements which is independent from the upper cover computation.

The above algorithm follows strictly the reasoning we have presented previously. It first sorts the lattice nodes in $\mathcal{C}$ in order to obtain a linear extension of $\leq_{\mathcal{L}}$. Then, the nodes are processed in a decreasing order, each time integrating the current node into the partial structure $\mathcal{L}$ that finally contains the entire lattice. At each step, the intersections between the current concept intent and the intents of all border concepts are computed and stored in the *Intents* set. Then, the maximal elements of *Intents* are

```
procedure HASSE

Input   : $\mathcal{C} = \{c_1, c_2, ..., c_l\}$
Output  : $\mathcal{L} = \langle \mathcal{C}, \leq_{\mathcal{L}} \rangle$

SORT($\mathcal{C}$)
$\mathcal{L} \leftarrow \{c_1\}$
Border $\leftarrow \{c_1\}$
for $i$ from 2 to $l$ do
    Intents $\leftarrow \emptyset$
    for all $\overline{c} \in$ Border do
        Intents $\leftarrow$ Intents $\cup$ (Int($\overline{c}$) $\cap$ Int($c_i$))
    Cover-Intents $\leftarrow$ MAXIMA(Intents)
    for all $Y \in$ Cover-Intents do
        $\hat{c} \leftarrow$ FIND-CONCEPT($Y$)
        MAKE-LINK($c_i$,$\hat{c}$)
    Border $\leftarrow$ (Border - upper-covers($c_i$)) $\cup \{c_i\}$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{c_i\}$
```

Algorithm 1: Building the Hasse diagram of a lattice from its ground set

selected and put in *Cover-Intents* (the MAXIMA primitive). These intents correspond to the most specific concepts that are greater than c_i, i.e., its upper covers. Next, for each element of *Cover-Intents*, the corresponding concept is localized within the partial structure P_i (FIND-CONCEPT) and the respective link is created. Finally, the upper covers of a concept are excluded from the border of the partial structure and the current element is added.

4.2 An example

The following table gives a partial trace of the algorithm by showing some of the iterations on the concepts (previously sorted according to the size of their intent). For example, the processing of concept c_6 generates as many intents as there are nodes in the current border. The corresponding upper covers are therefore $\{a\}$ and $\{c\}$, which give rise to two links: one between c_6 and c_2 (the node having $\{a\}$ as an intent), and another one between c_6 and c_3. The border is then updated by inserting the current node c_6 and deleting the parents of c_6 (i.e., c_2 and c_3).

4.3 Complexity issues

The complexity of this first algorithm may be roughly evaluated as follows. The sorting procedure may be done in $O(l\log(l))$ where $l = ||\mathcal{C}||$. The global loop (lines 9 to 18) on $\mathcal{C}$ takes l steps. The intent generation loop (lines 11-12) is executed once for each border element. As the border is made up of mutually incomparable elements, its size is bound by $\omega(\mathcal{L})$, the size of the maximal anti-chain of $\mathcal{L}$, also called the width of the lattice. Moreover, a straightforward intersection (line 12) algorithm for sets takes time that is linear in the set size. As the intersection is between intents, the cost is bound by $O(m)$ where $m = ||A||$. The complete time complexity of the generation step is therefore $O(\omega(\mathcal{L})m)$ algorithm. The maxima computation, when performed in a direct manner, may require up to $\omega(\mathcal{L})^2$ comparisons of intents, thus reaching a total complexity of

c_i	Intents	Cover-Intents	Links	Border
c_2	$\{\emptyset, \emptyset\}$	$\{\emptyset\}$	(c_1, c_2)	$\{c_2\}$
c_3	$\{\emptyset, \emptyset\}$	$\{\emptyset\}$	(c_1, c_3)	$\{c_2, c_3\}$
c_4	$\{\emptyset, \emptyset\}$	$\{\emptyset\}$	(c_1, c_4)	$\{c_2, c_3, c_4\}$
c_5	$\{\emptyset, \emptyset\}$	$\{\emptyset\}$	(c_1, c_5)	$\{c_2, c_3, c_4, c_5\}$
c_6	$\{\{a\}, \{c\}, \emptyset, \emptyset\}$	$\{\{a\}, \{c\}\}$	$(c_2, c_6); (c_3, c_6)$	$\{c_4, c_5, c_6\}$
c_7	$\{\{a\}, \{g\}, \emptyset\}$	$\{\{a\}, \{g\}\}$	$(c_2, c_7); (c_5, c_7)$	$\{c_4, c_6, c_7\}$
c_8	$\{\{c\}, \emptyset, \emptyset\}$	$\{\{c\}\}$	(c_3, c_8)	$\{c_4, c_6, c_7, c_8\}$
c_9	$\{\{b\}, \{g\}, \emptyset, \emptyset\}$	$\{\{b\}, \{g\}\}$	$(c_4, c_9); (c_5, c_9)$	$\{c_6, c_7, c_8, c_9\}$
c_{12}	$\{\{a\}, \{g\}, \{a, g, i\}\}$	$\{\{a, g, i\}\}$	(c_7, c_{12})	$\{c_9, c_{10}, c_{11}, c_{12}\}$
c_{13}	$\{\{b\}, \{c, f, h\}, \{c\}, \emptyset\}$	$\{\{b\}, \{c, f, h\}\}$	$(c_4, c_{13}); (c_8, c_{13})$	$\{c_9, c_{10}, c_{11}, c_{12}, c_{13}\}$

Table 1: The trace of the execution of Algorithm 1 with the concepts of the context in Figure 1.

the minima computation of $O(\omega(\mathcal{L})^2 m)$. Next, the loop over all maximal intents (lines 14-16) is run once for each upper cover. The number of the covers is bound by another lattice characteristics, $d(\mathcal{L})$, the maximal degree (input or output) of a lattice node. Concept look-up could be supported by a search structure that ensures linear worst-case time, i.e., $O(l)$. Thus, the cover localization takes $O(d(\mathcal{L})lm)$ in time. Finally, the border update may take at worst $O(d(\mathcal{L})\omega(\mathcal{L})m)$ in time since each combination of upper cover and border element may have to be tested.

In sum, the total complexity amounts to $O(l(\log(l) + \omega(\mathcal{L})m(d(\mathcal{L}) + \omega(\mathcal{L}))))$. The most time-consuming task is the maxima computation as the degree $d(\mathcal{L})$ is always bound by the lattice width $\omega(\mathcal{L})$. With the remark that $\log(l)$ is for sure less than $\omega(\mathcal{L})m$, the above formula simplifies to $O(l\omega(\mathcal{L})^2 m)$.

5 An improved algorithm

In the previous algorithm, we made no substantial use of some *ad-hoc* information that could essentially speed-up the computations. For example, one may observe that the first sorting operation, SORT, may be performed in linear time. In a similar way, the operation of finding the maximal intersections, MAXIMA, could take the advantage of a preliminary sorting so that a large number of unnecessary comparisons could be dropped off. Finally, instead of utilizing an additional search-and-insert structure to store the already processed part of the concept set, one may simply use the partial lattice structure. For this purpose, it is enough to define search criteria that efficiently explore the available links between concepts.

In the following, we describe in more detail the above improvements and then provide the new version of the algorithm.

5.1 Sort of concept set

In fact, a simple criteria that yields a linear extension of the lattice order is the size of the intents. In fact, if a concept is considered greater than another one whenever its intent contains more attributes, the obtained order clearly includes the lattice order $\leq \mathcal{L}$. However, the concept intents are of bounded size: the larger one contains all the attributes, so the sizes are at most m. Therefore, the set of possible values to sort upon, or keys, is limited to $\{0, 1, 2, .., m\}$.

In general, the sort of an item set $\mathcal{I}$ with a restricted key domain $D_{\mathcal{K}}$ may be performed efficiently by means of an array which is indexed by the key set (see below). Here, the objective is simply to split the set of concepts into bunches of equal intent sizes and then to concatenate the bunches to obtain the desired order. Actually, the order within a bunch is ignored, therefore, the first task will be to assign to each entry (e.g., 2) all the concepts that score to that value (e.g., $(\{1,2\},\{a,c\})$, $(\{1,4\},\{c,f,h\})$, etc.). To sort the whole item set, it will then be enough to sort the keys and then concatenate the bunches according to the resulting order. However, integer keys are naturally sorted, so no additional effort is to be provided. As splitting into bunches is linear in time, the entire sort operation is linear. The data structure required to implement the efficient sort is a simple array of sets.

```
1:  SORT($\mathcal{C} = \{c_1, c_2, ..., c_l\},$)
2:      Bunches : array [0..m] of sets
3:      Order : list of concepts
4:
5:  for $i$ from 1 to $m$ do
6:      Bunches[$i$] ← $\emptyset$
7:  Order ← $\emptyset$
8:  for $i$ from 1 to $l$ do
9:      $k \leftarrow \|Int(c_i)\|$
10:     Bunches[$k$] ← Bunches[$k$] $\cup \{c_i\}$
11: for $i$ from 1 to $m$ do
12:     for all $c \in$ Bunches[$i$] do
13:         Order ← $c$ & Order
```

Algorithm 2: Linear-time sorting of the concept set

The above algorithm produces a linear extension of the lattice order that is stored in the *Order* list. The algorithm is linear in time but also in space with respect to the number of concepts.

5.2 Order the intersections for comparison

The same linear-time sorting may be used to speed-up the search for maximal intents generated by intersections. The basic idea is that such a sorting will spare some redundant inclusion tests. The algorithm works as follows: first the intents are sorted, then, they are checked for minimalness. The main data structure used for inclusion checking is a list $Maxi$ of the previously discovered minima.

Intuitively, a list of intents that are candidates for minima may be maintained easily. Provided the list is properly initialized, one may expect that a simple comparison of every intent Y with the elements of the list could be enough to validate or invalidate the Y minima status. Concerning the entry/exit discipline for the list, one may consider the following simple strategies. An intent Y enters $Maxi$ whenever the list does not contain another intent Y' which is a super-set of Y. Conversely, each time a new intent Y is found that is a super-set of an intent Y' already in $Maxi$, Y' is dropped off the list.

As a matter of fact, the resulting simple algorithm does not improve the overall complexity of the `Maxima` task, as exposed in the previous section, since a square number of comparisons would be necessary in the worst case. However, when the intents are processed in a decreasing order (with respect to a linear extension that may easily be established upon their cardinalities as described in the previous section), the complexity improves substantially. To see the point, observe that with the decreasing order, no exit operation is to be performed (as for any given concept, all its super-concepts have already been processed). This means that the list can not decrease during the checking of a particular element Y, it may only remain the same (Y is not a minimum) or grow (Y is an effective minimum). Therefore, the maximal size of the list equals the number of effective upper covers of the concept c_i (see the Algorithm 1 above) which is bounded by the number d.

```
MAXIMA($\mathcal{Y} = \{Y_1, Y_2, ..., Y_r\}$)
   Maxi : list of intents

SORT($\mathcal{Y}$)
Maxi ← $\emptyset$
for $i$ from 1 to $r$ do
   is-min ← true
   for all $\hat{Y} \in$ Maxi do
      is-min ← is-min and ($Y_i \not\subseteq \hat{Y}$)
   if is-min then
      Maxi ← Maxi & $Y_i$
return Maxi
```

Algorithm 3: Efficient detection of the minima

Algorithm 3 has a time complexity which is quasi linear in the number of intents in the initial set $\mathcal{Y}$. In fact, the external loop (rows 6-11) will be executed exactly $||\mathcal{Y}||$ number of times. For each intent, the inner loop will perform at most $||Maxi||$ steps, a number which is bounded by $d(\mathcal{L})$. Finally, each inclusion test takes $O(m)$ in time where $m = ||A||$.

5.3 Concept lookup through the partial lattice

Another time consuming task is the lookup of concepts based on their intents, FIND-CONCEPT of Algorithm 1. Usually, operations that require lookup of concepts upon their intents or extents are powered by sophisticated search structures. However, the operations of search and maintenance of such a structure (entry/exit) often represent efforts of different magnitudes. Here, we use the lattice structure itself to find concepts as it contains all the inter-concept links necessary for an efficient search. The basic idea is to start the search of a concept c of intent Y by a lower bound of that concept, say c', and to follow an up-going path until c is found. For that purpose, with each intent Y in the list *Intents* of Algorithm 1 we keep a pointer that (physically) indicates the concept c' which helped in generating Y (by intersection with the intent of c_i). In the Algorithm 4 below, the primitive *generator*() yields the concept that generated an intent. Actually, if the intent is taken as a pure set, then there are several such concepts in the general case. However, as we do not check uniqueness in the intent generation step, all the copies of the same intent are considered different and as such they will point to different concepts. This does not disturb our lookup as any of the pointed concepts lays in an up-going path that leads to the target concept. As an example, at the step 13 of the main loop in Algorithm 1 (see Table 4.2), the *Cover-Intents* set contains the sets

$\{b\}$ and $\{c, f, h\}$. The intent $\{b\}$ has been generated by the intersection of the intents in c_{13} and c_9. However, it is a strict subset of the intent of c_9, $\{b, e, g\}$ whereas the concept whose intent is equal to $\{b\}$, c_4, lays above c_9.

The lookup algorithm is simple, it moves upwards, one sub-concept link at a time. At any step, it checks, in a one-step look-ahead manner the inclusion of the intent Y in the intents of the concepts met. The auxiliary primitive *next* hereafter helps enumerate a collection of items by successfully providing each of its members in an arbitrary order.

```
FIND-CONCEPT($Y$)

$c \leftarrow generator(Y)$
while $Int(c) \neq Y$ do
    $\hat{c} \leftarrow next(upper\text{-}covers(c))$
    while $Y \not\subseteq Int(\hat{c})$ do
        $\hat{c} \leftarrow next(upper\text{-}covers(c))$
    $c \leftarrow \hat{c}$
return $c$
```

Algorithm 4: Efficient lookup by intent.

The algorithm termination with the right concept can be proved trivially based on two facts. First, the intents of the concepts along the path that is walked through by the algorithm represent a strictly decreasing sequence of sets. Next, the intent Y is always a subset of the current intent.

The algorithm has a modest time complexity. Actually, a straightforward evaluation may lead to a result that is linear in the size of the lattice: the up-going path is bounded by the height of the lattice h, i.e., the size of a maximal chain. This value, usually much less than the lattice size, could nevertheless grow up to that size in the worst case (totally ordered lattice). Although, the number of concepts examined could be estimated by the number of elements in the set difference between the intent of $generator(Y)$ and Y. Theoretically, this difference is bounded by m, yielding a total time complexity of $O(m^2 d(\mathcal{L}))$ (as the inner loop executes at most d steps). However, in a realistic context the cardinality of the difference, and therefore the number of steps of the outer loop, would not exceed 4 or 5, since the chances for Y both being a maximum in *Intents* and having much smaller size than the intent of $generator(Y)$ are close to zero.

5.4 Complexity issues

The efficient procedures presented above help improve the overall complexity of the diagram building algorithm. The following table summarizes the complexity of the different stages of the Algorithm 1 both in the straightforward and in the improved version.

Step	Straightforward algorithm	Improved algorithm
SORT	$O(l \log(l))$	$O(l)$
MAXIMA	$O(\omega(\mathcal{L})^2 m)$	$O(d(\mathcal{L})\omega(\mathcal{L})m)$
FIND-CONCEPT	$O(lm)$	$O(m^2 d(\mathcal{L}))$
1st inner **for**	$O(\omega(\mathcal{L})m)$	$O(\omega(\mathcal{L})m)$
2nd inner **for**	$O(d(\mathcal{L})lm)$	$O(md(\mathcal{L})^2)$
outer **for**	$O(l\omega(\mathcal{L})^2 m)$	$O(lm\omega(\mathcal{L})d(\mathcal{L}))$
HASSE	$O(l\omega(\mathcal{L})^2 m)$	$O(lm\omega(\mathcal{L})d(\mathcal{L}))$

As the above table suggests, the overall time complexity of the algorithm is $O(lm\omega(\mathcal{L})d(\mathcal{L}))$. Moreover, the factor m could be further dropped off the formula under some circumstances. In fact, whenever the number of the attributes remains reasonable (at most several hundreds), the comparisons of concepts could be substituted, through an appropriate encoding, by binary vector operations. These operations take constant time, so that the time complexity reduces to $O(l\omega(\mathcal{L})d(\mathcal{L}))$. Concerning space, the algorithm has linear complexity in the number of concepts. In fact, the number of data items relevant to a concept is bounded by the number of the auxiliary structures used, which is constant.

Our complexity results relate to the work reported by Ganter et al. [8] and Lindig [14]. The algorithm suggested by Ganter *et al.* has a time complexity of $O(lg^2m)$ where g is the cardinality of the object set O, $g = \|O\|$. Lindig in turn proposed a complete algorithm for the lattice building task of the same asymptotic time complexity. It is difficult to compare the algorithms with respect to their analytical complexity as the respective functions depend on different lattice parameters. However, one may hypothesize that our algorithm will perform better in the case of a sparse context, i.e., tables with few attributes per object. In fact, with such contexts, the width of the Galois lattice is a linear function of the number of objects/attributes, so the asymptotic complexity will reduce to $O(lm(m+g)d(\mathcal{L}))$ which is essentially linear in lattice size and quadratic in the context size (mg).

6 Conclusions and further research

We presented an algorithm that computes the Hasse diagram of a Galois (concept) lattice given the set of the closed sets (concepts). It relies upon a strategy of gradual insertion of the concepts into the partially built diagram, in a top-down manner. At each step, the portion of the diagram already constructed is maximally reused to speed-up the current insertion.

The algorithm represents a complement of the well-known algorithms of Noris, Chein and Ganter that compute the set of closed sets of a binary relation. In that sense, our algorithm, when it is jointly applied with a concept computing algorithm, is an alternative to the batch procedures of Bordat and Lindig as well as to the incremental procedure of Godin, which constructs both the ground set and the Hasse diagram.

Our algorithm shows asymptotic time complexity that is linear in the number of the concepts, the number of the attributes and the width of the lattice. In case of a sparse context, this result is significantly better than the results of the already existing techniques.

Much of the work on the algorithm is still to come. On the one hand, detailed experiments will be necessary in order to evaluate the practical performances of the method. For the time being, it is implemented in C++ and an initial package of tests are carried out. The testbed, made up of randomly generated data contexts, is aimed at evaluating the practical complexity with respect to various lattice measures (width, height, number of objects and/or attributes, number of couples in the incidence relation, etc.). The initial results suggest very good performances which are to be explained by the efficient detection of the upper-covers by a simple intent intersection (instead of closure computation as suggested by previous work). On the other hand, it will be interesting to find some estimators of the algorithm's key complexity factor, the lattice width, which is not measurable directly from the context. We believe some combinatorial results for digraphs may be successfully adapted to the special case of concept lattices.

Alternatively, the appropriate use of some advanced data structures may help improve the effective complexity score of the algorithm. For example, the use of Ordered Binary-Decision Diagrams [4] for the concept lookup seems to be a promissing research track.

References

[1] M. Barbut and B. Monjardet. *Ordre et Classification: Algèbre et combinatoire*. Hachette, 1970.

[2] B. Birkhoff. *Lattice Theory*, volume 25. American Mathematical Society Colloquium Publ., Providence, revised edition, 1973.

[3] J.-P. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Mathématiques et Sciences Humaines*, 96:31–47, 1986.

[4] R. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[5] M. Chein. Algorithme de recherche des sous-matrices premières d'une matrice. *Bull. Math. de la soc. Sci. de la R.S. de Roumanie*, 13, 1969.

[6] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 1992.

[7] B. Ganter. Two basic algorithms in concept analysis (preprint). Technical Report 831, Technische Hochschule, Darmstadt, 1984.

[8] B. Ganter and S. Kuznetsov. Stepwise Construction of the Dedekind-McNeille Completion. In *Proceedings of the 6th ICCS*, Montpellier, 1998.

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

[10] R. Godin, G. W. Mineau, and R. Missaoui. Incremental structuring of knowledge bases. In *Proceedings of the first International Symposium on Knowledge Retrieval, Use and Storage for Efficiency, Santa Cruz, CA, USA*, pages 179–193, 1995.

[11] R. Godin and R. Missaoui. An Incremental Concept Formation Approach for Learning from Databases. *Theoretical Computer Science*, 133:378–419, 1994.

[12] A. Guénoche. Construction du treillis de galois d'une relation binaire. *Mathématiques et Sciences Humaines*, 109:41–53, 1990.

[13] R. Kent. Rough concept analysis: A synthesis of rough sets and formal concept analysis. *Funadamenta Informaticae*, 27:169–181, 1996.

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

[15] E. M. Norris. An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 23(2):243–250, 1978.

[16] G. Snelting and F. Tip. Reengineering class hierarchies using concept analysis. In *Proceedings of ACM SIGPLAN/SIGSOFT Symposium on Foundations of Software Engineering*, pages 99–110, Orlando, FL, 1998.