A partition-based approach towards constructing Galois (concept) lattices

P. Valtchev [a,b,*], R. Missaoui [b], P. Lebrun [b]

[a] *INRIA, France*

[b] *Département d'Informatique, UQAM, C.P. 8888, succ. "Centre Ville", Montréal, Québec, Canada, H3C 3P8*

Abstract

Galois lattices and formal concept analysis of binary relations have proved useful in the resolution of many problems of theoretical or practical interest. Recent studies of practical applications in data mining and software engineering have put the emphasis on the need for both efficient and flexible algorithms to construct the lattice. Our paper presents a novel approach for lattice construction based on the apposition of binary relation fragments. We extend the existing theory to a complete characterization of the global Galois (concept) lattice as a substructure of the direct product of the lattices related to fragments. The structural properties underlie a procedure for extracting the global lattice from the direct product, which is the basis for a full-scale lattice construction algorithm implementing a divide-and-conquer strategy. The paper provides a complexity analysis of the algorithm together with some results about its practical performance and describes a class of binary relations for which the algorithm outperforms the most efficient lattice-constructing methods.

Key words: Galois lattice, formal concept analysis, lattice-constructing algorithms, data fragments, context apposition, lattice products

1 Introduction

Formal Concept Analysis (FCA) is a branch of the lattice theory motivated by the need for a clear mathematization of the notions of *concept* and *conceptual hierarchy* [7]. The main concern within FCA is about the lattice structure induced by a binary relation between a pair of sets, called a *Galois* lattice [1] or *concept* lattice [21]. Initially intended as an intuitive foundation for the

* Corresponding author, now at: valtchev@IRO.UMontreal.CA.

whole lattice theory [21], the FCA has inspired a number of studies establishing links between the Galois (concept) lattices and other known classes of partially ordered structures [22,23,17]. Meanwhile, the strong theoretical foundations of FCA have attracted the interest of practitioners from various fields such as data mining [18,25], knowledge acquisition [15], and software engineering [8,19]. Today, there is a constantly growing number of studies within the field about both theoretical and practical issues.

Our own concern is the development of effective algorithms for the construction of Galois/concept lattices from realistic datasets, i.e., large collections of possibly volatile, fragmented and noisy data items. The analysis of such data requires the design of a new generation of lattice-constructing algorithms that combine computational efficiency, robustness and flexibility. In the present paper, we tackle the problem of assembling lattices corresponding to the fragments of a binary table, a problem which arises with various dataset updates. For this purpose, we complete the existing theory of *a posteriori* fragmentation of binary tables due to Wille [24] and provide the foundation of an efficient lattice assembly procedure. The procedure carries out a filtering of the direct product of the partial lattices which retrieves the concepts of the global lattice and their precedence links. The procedure is further extended to a full-range lattice-constructing algorithm implementing a divide-and-conquer strategy which, when applied to a particular class of binary relations, proved more efficient than the most powerful lattice algorithms known to date.

The benefits of our assembly-based strategy lie far beyond mere visualization or CPU-time gain. Aside from a better understanding of the semantics behind context and lattice composition, our structural results constitute a unified framework for devising new lattice algorithms.

The paper is organized as follows. Section 2 gives a background on Galois/concept lattices and nested line diagrams. In Section 3, we recall the definition of the apposition operation and list some important properties of the mappings between concepts in the global lattice and those in the partial ones. Section 4 presents the basic structural results that underlie our algorithmic approach. The approach itself is completely described in Section 5. Finally, we compare our work with some related previous studies in Section 6.

2 Background on formal concept analysis

Formal concept analysis (FCA) [7] is a discipline that studies the hierarchical structures induced by a binary relation between a pair of sets. The structure, made up of the closed subsets (see below) ordered by set-theoretical inclusion, satisfies the properties of a complete lattice and has been first mentioned

in the work of Birkhoff (see [2]). Later on, it has been the subject of an extensive study [1] under the name of *Galois lattice*. The term *concept lattice* and formal concept analysis are due to Wille [21]. In the following, we first recall some basic notions of ordered structure theory [1], which are widely used in the presentation of FCA.

2.1 The basics of ordered structures

$P = \langle G, \leq_P \rangle$ is a *partial order* (poset) over a *ground* set G and a binary relation $\leq_P$ if $\leq_P$ is reflexive, antisymmetric and transitive. For a pair of elements a, b in G, if $b \leq_P a$ we shall say that a *succeeds* (is greater than) b and, inversely, b *precedes* a. If neither $b \leq_P a$ nor $a \leq_P b$, then a and b are said to be *incomparable*. All common successors (predecessors) of a and b are called *upper* (*lower*) *bounds*. The *precedence* relation $\prec_P$ in P is the transitive reduction of $\leq_P$, i.e. $a \prec_P b$ if $a \leq_P b$ and all c such that $a \leq_P c \leq_P b$ satisfy $c = a$ or $c = b$. Given such a pair, a will be referred to as an *immediate predecessor* of b and b as an *immediate successor* of a. Usually, P is represented by its *covering graph* $Cov(P) = (G, \prec_P)$. In this graph, each element a in G is connected to both the set of its immediate predecessors and of its immediate successors, further referred to as *lower covers* (Cov^l) and *upper covers* (Cov^u) respectively. In the following, we shall visualize a partial order by its Hasse diagram, that is the line diagram of the covering graph where each element is located "below" all its successors.

A subset A of G is a chain (anti-chain) in P if all elements in A are mutually comparable (incomparable). A subset B of G is an *order ideal* (*order filter*) if $\forall a \in G, b \in B$, $a \leq_P b \Rightarrow a \in B$ ($b \leq_P a \Rightarrow a \in B$). For a given set $A \subseteq X$, the set $\downarrow_P A = \{c \in X | \exists a \in A, c \leq_P a\}$ is the smallest order ideal containing A. Dually, $\uparrow_P A = \{c \in X | \exists a \in A, a \leq_P c\}$ denotes the smallest order filter containing A. In case of a singleton A, we shall note $\downarrow_P a$ instead of $\downarrow_P \{a\}$ ($\uparrow_P a$ instead of $\uparrow_P \{a\}$). Moreover, the *order interval* $[a, b]$ is the subset of nodes obtained by the intersections of an order filter $\uparrow_P a$ and an order ideal $\downarrow_P b$. A *convex* subset of an order is a subset that includes for any pair of its members the interval they might compose. A mapping ϕ between two posets P and Q such that $\phi : P \to Q$ is said to be *order preserving* if an order relation between two elements of P entails an order relation between their respective images under ϕ in Q:

$$x \leq_P y \Rightarrow \phi(x) \leq_Q \phi(y).$$

Furthermore, ϕ is said to be an *order embedding* of P into Q if the condition is also a sufficient one:

$$x \leq_P y \Leftrightarrow \phi(x) \leq_Q \phi(y).$$

[1] An excellent introduction to the subject may be found in [5].

A lattice $L = \langle G, \leq_L \rangle$ is a partial order where any pair of elements a, b has a unique *greatest lower bound* (GLB) and a unique *least upper bound* (LUB). GLB and LUB define binary operators on G called respectively *join* ($a \vee_L b$) and *meet* ($a \wedge_L b$). In a complete lattice L, all the subsets A of the ground set have a GLB and a LUB. In particular, there are unique maximal (top, $\top$) and minimal (bottom, $\bot$) elements in the lattice. Finally, the elements with a single immediate predecessor (successor) are called *join*-irreducible (*meet*-irreducible), and the set that they constitute is denoted by $\mathcal{J}(L)$ ($\mathcal{M}(L)$).

2.2 Fundamentals of FCA

FCA considers a binary relation I (*incidence*) over a pair of sets O (*objects*) and A (*attributes*). The attributes considered represent binary features, i.e., with only two possible values, *present* or *absent*. In this framework, the meaningful subsets of objects/attributes represent the closed sets of the Galois connection [1] induced by I on the pair O and A.

2.2.1 Basic notions

The binary relation is given by the matrix of its incidence relation I (oIa means that object o has the attribute a). This is called *formal context* or simply *context* (see Figure 1 for an example).

Definition 1 *A formal context is a triple $\mathcal{K} = (O, A, I)$ where O and A are sets (objects and attributes respectively) and I is an incidence relation, i.e., $I \subseteq O \times A$.*

For convenience reasons, we shall denote objects by numbers and attribute by lower-case letters. Furthermore, we simplify the standard set notation by dropping out all the separators (e.g., *127* will stand for the set of objects $\{1, 2, 7\}$, and *abdf* for the set of attributes $\{a, b, d, f\}$.

Two set-valued functions, f and g, summarize the links between objects and attributes established by the context.

Definition 2 *The function f maps a set of objects into the set of common attributes, whereas g is the dual for attribute sets:*

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

For example, w.r.t. the context in Figure 1, $f(678) = acd$ and $g(abgh) = 23$. In what follows, for brevity, both functions will be denoted by $'$. Furthermore,

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

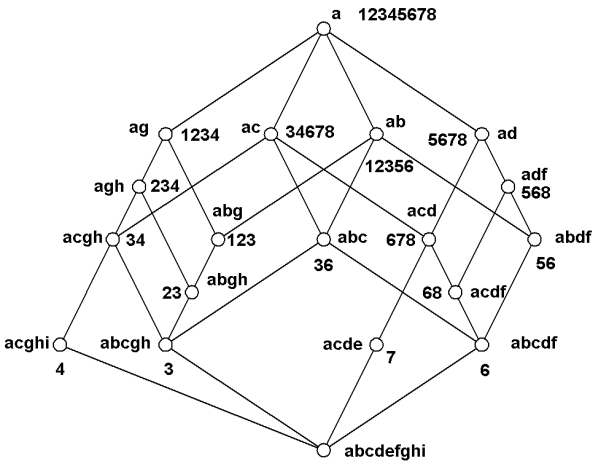

Fig. 1. A sample context (left) and its corresponding Galois/concept lattice (right). Both have been borrowed from [7].

f and g are combined in a pair of composite operators, $g \circ f(X)$ and $f \circ g(Y)$ which map the sets $\mathcal{P}(O)$ and $\mathcal{P}(A)$ respectively into themselves (denoted by $''$). For example, the image of $X = \{5, 6, 7\}$ is $X'' = \{5, 6, 7, 8\}$. The functions f and g induce a *Galois connection* [1] between $\mathcal{P}(O)$ and $\mathcal{P}(A)$ when both sets are taken with the set-inclusion $\subseteq$ as (partial) order relation. It follows that the composite operators are actually *closure operators* and therefore each of them induces a family of *closed* subsets over the respective power-set. The functions f and g constitute bijective mappings between both families. A pair (X, Y), of mutually corresponding subsets is called a *(formal) concept* in [21].

Definition 3 *A formal concept is a pair* (X, Y) *where* $X \in \mathcal{P}(O)$, $Y \in \mathcal{P}(A)$, $X = Y'$ *and* $Y = X'$.

For example (see Figure 1), the pair $c = (678, acd)$ is a concept since the objects 6, 7, and 8 share the properties a, c, and d and, conversely, the attributes a, c, and d are held by the objects 6, 7, and 8. In the FCA framework, X is referred to as the concept *extent* and Y as the concept *intent*. In what follows, concepts will be denoted by the letter c and the auxiliary primitives $Intent()$ and $Extent()$ will refer to their intent and extent respectively.

The set of all concepts of the context $\mathcal{K} = (O, A, I)$, $\mathcal{C}_\mathcal{K}$, is partially ordered by the order induced by intent/extent set theoretic inclusion:

$$(X_1, Y_1) \leq_\mathcal{K} (X_2, Y_2) \Leftrightarrow X_1 \subseteq X_2 (Y_2 \subseteq Y_1).$$

In fact, set inclusion induces a *complete lattice* over each closed family and both lattices are isomorphic to each other with f and g as dual isomorphisms. Both lattices are thus merged into a unique structure called the *Galois lattice* [1] or the *(formal) concept lattice* of the context $\mathcal{K}$ [7].

Proposition 4 *The partial order* $\mathcal{L} = \langle \mathcal{C}_{\mathcal{K}}, \leq_{\mathcal{K}} \rangle$ *is a complete lattice with LUB and GLB as follows:*

- $\bigvee_{i=1}^{k}(X_i, Y_i) = ((\bigcup_{i=1}^{k} X_i)'', \bigcap_{i=1}^{k} Y_i)$,
- $\bigwedge_{i=1}^{k}(X_i, Y_i) = (\bigcap_{i=1}^{k} X_i, (\bigcup_{i=1}^{k} Y_i)'')$.

Figure 1 shows the concept lattice of the context in Figure 1. For example, given the concepts $c_1 = (36, abc)$ and $c_2 = (56, abdf)$, their join and meet are computed as follows:

- $(36, abc) \vee (56, abdf) = (12356, ab)$,
- $(36, abc) \wedge (56, abdf) = (6, abcdf)$.

The interest in FCA and Galois lattices from a theoretical viewpoint is motivated by the following remarkable property [21] basically stating that each complete lattice is isomorphic to the concept lattice of some formal context.

Proposition 5 *Given a lattice* $L = \langle G, \leq_L \rangle$*, let* $\mathcal{J}(L)$ *and* $\mathcal{M}(L)$ *be the sets of its join-irreducible and its meet-irreducible elements respectively. Then* L *is isomorphic to the concept lattice of the context* $\mathcal{K}_L = (\mathcal{J}(L), \mathcal{M}(L), \mathcal{J}(L) \times \mathcal{M}(L) \cap \leq_L)$.

Moreover, well-known constructs from partial order theory such as the *Dedekind-McNeille hull* of a given poset P or the *lattice the maximal anti-chains* of P can be constructed as the concept lattices of suitably chosen contexts.

2.3 Approaches towards the construction of Galois/concept lattices

There is a variety of algorithms that may be used in computing the concept lattice from the binary table. These can be mainly divided into two groups: procedures which extract the set of concepts [4,16,6] only, and algorithms for constructing the entire lattice, i.e., concepts together with the lattice order [3,9,17]. A detailed description of these algorithms being out of the scope of this paper [2], we focus only on the most efficient lattice algorithms known today.

An efficient algorithm has been suggested by Bordat in [3] which generates both the concept set and the Hasse diagram of the lattice. It takes advantage of the structural properties of the precedence relation $\prec_{\mathcal{L}}$ to generate the concepts in an increasing order. Thus, from each concept c the algorithm generates its upper covers - these are the concepts whose intents are maximal

[2] But an interested reader will find a comprehensive study (of partial scope) in [10], while [9] and and [14] could also help

closed subsets of the intent of c. The obvious drawback of the method is that a concept is generated several times, once per each lower cover. A partial remedy of the problem is a concept lookup mechanism which allows each concept extent to be computed only once.

The design of flexible algorithms was pioneered by Godin *et al.* [9] who designed an incremental method for constructing the concept lattices. The lattice $\mathcal{L}$ is thus constructed starting from a single object o_1 and gradually incorporating any new object o_i (on its arrival) into the lattice $\mathcal{L}_i$ (over a context $\mathcal{K} = (\{o_1, ..., o_{i-1}\}, A, I)$), each time carrying out the necessary structural updates. This method avoids starting from scratch each time the context is extended (with a new object/attribute) by making a maximal reuse of the already available structure. Actually, the incremental paradigm, known elsewhere in practical applications of lattice theory [11,20], has been introduced even earlier, with the algorithm of Norris [16] which is basically an incremental procedure even if not stated as such.

Recently, Nourine and Raynaud [17] suggested a general approach towards the computation of closure structures and showed how it could be used to construct Galois/concept lattices. The method proceeds in two steps: first, concepts are generated, and then their precedence links are established. Concept generation is an incremental process similar to the one described in [9]. The second step is a traversal of the concept set whereby for a given concept c the entire order filter $\uparrow_{\mathcal{L}} c$ is computed. Concepts in $\uparrow_{\mathcal{L}} c$ are generated as joins of c and any object o which is not in $Extent(c)$. An upper cover $\bar{c}$ of c is detected within $\uparrow_{\mathcal{L}} c$ by considering the number of the objects in $Extent(\bar{c}) - Extent(c)$. The following proposition summarizes the results from [17]:

Proposition 6 *A concept $\bar{c}$ is an upper cover of another concept c iff it is generated a number of times equal to the size of the set difference between the respective extents:*

$$c \prec_{\mathcal{L}} \bar{c} \Leftrightarrow \|Extent(\bar{c}) - Extent(c)\| = \|\{o \in O | o' \cap Intent(c) = Intent(\bar{c})\|.$$

It is noteworthy that extensive studies of the practical performances of most of the above cited algorithms are provided by [9] (somewhat out-dated) and [14]. Such studies are of high importance for the domain since all the algorithms have exponential worst-case complexity due to the exponential number of concepts, and therefore are just as inefficient as a naive algorithm that examines all possible subsets of attributes/objects. However, in practical cases, only a small number of concepts do occur, so it makes sense to study how each algorithm performs on realistic datasets.

2.4 The evolution of requirements

A very general observation on the current state of the FCA algorithmics, on the one hand, and the intended applications, on the other hand, reveals a big performance gap. In fact, most of the existing algorithms have been designed to work on small binary tables (less than hundred objects/attributes) stored in the memory of a single computer. However, current data in software applications, databases and data warehouses typically give rise to huge contexts which require secondary storage, may contain missing (NULL) or invalid values, and can be distributed over a network. Moreover, databases often constitute highly volatile contexts due to frequent updates to data.

Actually, the one-increment, i.e., adding a single object at a time, is only a partial solution to the problem of volatile data. As a matter of fact, in most databases and data warehouses the updates are not object-wise but rather group-wise, meaning that a whole subset of objects $\delta O = \{o_{i+1}, ..., o_{i+l}\}$ are to be added at a time. Instead of inserting them one by one into $\mathcal{L}_i$, one may think of first extracting the lattice $\delta\mathcal{L}$ corresponding to δO and then construct $\mathcal{L}_{i+l}$ from $\mathcal{L}_i$ and $\delta\mathcal{L}$. The challenging problem is then the *merging* or *assembly* of both lattices, a task which generalizes the conventional, single-object incrementation.

The decomposition of complex problems into a set of smaller problems is a solving strategy used in every area of computer science. In the FCA field, such a decomposition may be carried out in various ways. A natural way of doing it consists of splitting a large context into a set of smaller contexts that share objects and/or attributes. For example, splits of this kind might be used to simulate batch-wise updates of a data warehouse (common attributes/different objects), integration of several viewpoints on a set of individuals (common objects/different attributes) or a distribution of the dataset over a network (different objects/different attributes). Once the sub-contexts are established and the respective lattices constructed, these could be then merged into a unique global structure, corresponding to the entire dataset.

The partition of a context over its object/attribute set has been formalized through the apposition/subposition operations which have been defined to support visualization (see the next Section). However, to the best of our knowledge there have been no detailed studies of the relevant algorithmic problems. In particular, the potential utility of the underlying framework for lattice-constructing purposes has not been explored.

In what follows we present a Galois (concept) lattice assembly procedure for contexts that share the same set of objects and show a way to extend it into a full-scope lattice-constructing method. As our approach is rooted in the

FCA theory related to the apposition/subposition operators, we first recall its basics.

3 Apposition of contexts and partial lattices

Beside classical order product operators for lattices (e.g., *direct*, *subdirect*, *tensorial* products [7]), FCA provides a set of context-oriented operators of specific impact on the corresponding lattices. Here we focus on *apposition/subposition* operators which were initially intended to support visualization of large lattices.

3.1 Apposition and subposition of contexts

Apposition is the horizontal concatenation of contexts sharing the same set of objects [7].

Definition 7 *Let $\mathcal{K}_1 = (O, A_1, I_1)$ and $\mathcal{K}_2 = (O, A_2, I_2)$ be two contexts with the same set of objects O. Then the context $\mathcal{K} = (O, A_1 \dot{\cup} A_2, I_1 \dot{\cup} I_2)$ is called the apposition of $\mathcal{K}_1$ and $\mathcal{K}_2$: $\mathcal{K} = \mathcal{K}_1 | \mathcal{K}_2$.*

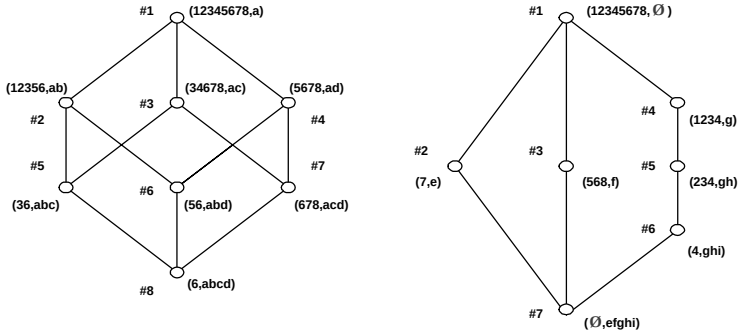

Fig. 2. Partial lattices $\mathcal{L}_1$ and $\mathcal{L}_2$ constructed from a vertical decomposition of the context in Figure 1.

Usually, the intent of $\mathcal{K}$ is set to the *disjoint union* (denoted by $\dot{\cup}$) of the involved context intents, but this constraint is not essential for our own study. For example, with a *global* context $\mathcal{K} = (O, A, I)$ as given in Figure 1, where $O = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $A = \{a, b, c, d, e, f, g, h, i\}$, let $A_1 = \{a, b, c, d\}$ and $A_2 = \{e, f, g, h, i\}$. The two lattices corresponding to $\mathcal{K}_1$ and $\mathcal{K}_2$, say $\mathcal{L}_1$ and $\mathcal{L}_2$, are given in Figure 2. These lattices will further be referred to as *partial* lattices as $\mathcal{K}_1$ and $\mathcal{K}_2$ are partial contexts for $\mathcal{K}$. Some intriguing properties relating $\mathcal{L}_1$ and $\mathcal{L}_2$ to the lattice of the apposition context $\mathcal{L}$ will be discussed in the following paragraphs.

Subposition, or vertical assembly of contexts upon a common attribute set, is dual to apposition. Hence, the results found in this paper are dually valid for subposition.

3.2 Nested line diagrams

Nested line diagrams (NLD) [7] are visualization aids which allow a lattice $\mathcal{L}$ to be drawn as a sub-structure of the *direct* product of a pair[3] of partial lattices $\mathcal{L}_1$ and $\mathcal{L}_2$. Recall that the direct product of a pair of lattices $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{L}_\times = \mathcal{L}_1 \times \mathcal{L}_2$, is itself a lattice $\mathcal{L}_\times = \langle \mathcal{C}_\times, \leq_\times \rangle$ where:

- $\mathcal{C}_\times = \mathcal{C}_{\mathcal{K}_1} \times \mathcal{C}_{\mathcal{K}_2}$,
- $(c_1, c_2) \leq_\times (\bar{c}_1, \bar{c}_2) \;\Leftrightarrow\; c_1 \leq_{\mathcal{L}_1} \bar{c}_1,\; c_2 \leq_{\mathcal{L}_2} \bar{c}_2$.

The nodes of $\mathcal{L}_\times$ are pairs of concepts (c_1, c_2) where c_i appears in $\mathcal{L}_i$ for $i = 1, 2$. For convenience reasons, the concrete concepts from the above example will be identified by their index in the respective partial lattice, a unique number denoted by $\#i$, ranging between 1 and $\|\mathcal{C}_{\mathcal{K}_i}\|$. Within a pair (c_i, c_j) in $\mathcal{L}_1 \times \mathcal{L}_2$, the index of the partial lattice where the concept $c_{\#i}$ is to be taken corresponds to its order. Thus, the pair $(c_{\#7}, c_{\#3})$ denotes the product of the concept #7 of $\mathcal{L}_1$, i.e., the concept $(678, acd)$, with the concept #3 of $\mathcal{L}_2$, $(568, f)$.

A nested line diagram basically represents the product lattice $\mathcal{L}_\times$ by combining the respective line diagrams of the lattices $\mathcal{L}_i$ into a unique complex structure. However, neither the nodes of $\mathcal{L}_\times$ nor their precedence links are directly represented. Instead, the information about them is spread over the various levels of nesting. Figure 3 presents the NLD of the product lattice $\mathcal{L}_\times = \mathcal{L}_1 \times \mathcal{L}_2$. As it can be seen, the line diagram of the lattice $\mathcal{L}_1$ is used as an outer frame in which the diagram of $\mathcal{L}_2$ is embedded. Within the NLD, a node (c_i, c_j) of the product is located by first finding the node for c_i in the outer diagram and then finding the respective node c_j within the local $\mathcal{L}_2$ diagram. For example, the node $(c_{\#3}, c_{\#1})$ (see the numbering in Figure 2) of the product lattice is located within the NLD in Figure 3 at the top of the inner lattice within the outer node labeled by c.

The lattice $\mathcal{L}$ is represented by an isomorphic sub-structure of $\mathcal{L}_\times$. The nodes belonging to the sub-structure are drawn in a distinguishable way on the diagram (here in black as opposed to the remaining "void" nodes drawn in white). The reader may check the isomorphism between the partial order induced by the black nodes (further referred to as *full* nodes) in Figure 3 and the lattice shown in Figure 1.

[3] The definition generalizes easily to an arbitrary number of lattices. However we only consider the two-level nesting.

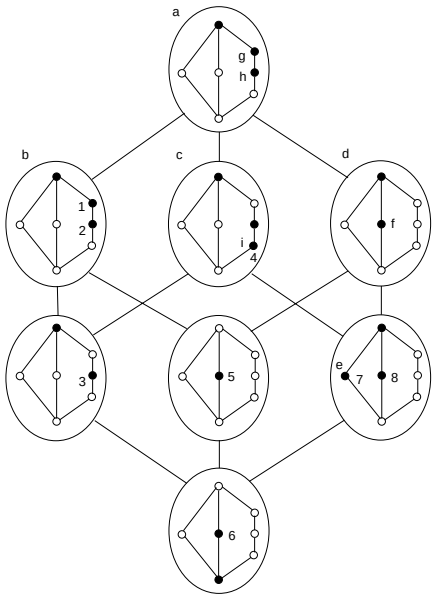

Fig. 3. The nested line diagram of the lattice $\mathcal{L}_1 \times \mathcal{L}_2$.

3.3 Linking the apposition lattice to the partial lattices

Any concept of $\mathcal{L}$ can be "projected" upon the concept lattice, $\mathcal{L}_1$ ($\mathcal{L}_2$) by restricting its intent to the set of "visible" attributes, e.g., those in A_1 (A_2). Combining both functions mapping the global lattice onto the partial ones results in an order homomorphism between $\mathcal{L}$ and the direct product (see [7]).

Definition 8 *The function $\varphi : \mathcal{C} \to \mathcal{C}_\times$, maps a concept from the global lattice into a pair of concepts of the partial lattices by splitting its intent over the partial context attribute sets A_1 and A_2:*

$$\varphi((X,Y)) = (((Y \cap A_1)', Y \cap A_1)\ ,\ ((Y \cap A_2)', Y \cap A_2)).$$

For example, the concept $(568, adf)$ is mapped by φ into the concepts $(5678, ad)$ and $(568, f)$ from $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. The mapping φ is fundamental within the nesting framework, since the set of nodes representing $\mathcal{L}$ within $\mathcal{L}_1 \times \mathcal{L}_2$ are exactly the images of the concepts from $\mathcal{L}$ by φ.

The inverse mapping between $\mathcal{L}_1 \times \mathcal{L}_2$ and $\mathcal{L}$ is also based on intersection, but of concept extents. Actually, the homomorphism ψ sends a pair of partial concepts from the product lattice into a global concept whose extent is the intersection of their own extents.

Definition 9 *The function $\psi : \mathcal{C}_\times \to \mathcal{C}$ maps a pair of concepts over partial contexts into a global concept by the intersection over their respective extents:*

$$\psi(((X_1, Y_1)\ ,\ (X_2, Y_2))) = (X_1 \cap X_2\ ,\ (X_1 \cap X_2)').$$

For example, the image of the pair $(c_{\#7}, c_{\#3})$ (see Figure 2) by ψ is the concept $(68, acdf)$. A set of interesting properties of the mappings φ and ψ have been listed in [12], mostly based on the basic observation that any concept intent in $\mathcal{L}_i$ is the intersection of a concept intent in $\mathcal{L}$ with the attribute set A_i (see Definition 8).

First, φ maps distinctive concepts into distinctive product nodes.

Proposition 10 *The mapping φ is **injective**.*

PROOF. (sketch) As partial concept intents are intersections of global intents with the sets A_i, given distinct concepts (X, Y) and $(\bar{X}, \bar{Y})$ in $\mathcal{L}$, at least one of the following conditions holds:

- $Y \cap A_1 \neq \bar{Y} \cap A_1$,
- $Y \cap A_2 \neq \bar{Y} \cap A_2$.

Hence, the images of (X, Y) and $(\bar{X}, \bar{Y})$ under φ diverge on at least one of the product dimensions.

Next, φ is compatible with the order within $\mathcal{L}_\times$.

Proposition 11 *The mapping φ is an **order embedding** of $\mathcal{L}$ into $\mathcal{L}_\times$:*

$$\varphi : \mathcal{L} \hookrightarrow \mathcal{L}_\times.$$

PROOF. (sketch) φ preserves the order between concepts since the intersection of intents with A_1 (or with A_2) is monotonic with respect to $\mathcal{L}$. The mapping is also injective; so it is an *order embedding* of $\mathcal{L}$ into $\mathcal{L}_\times$.

Moreover, φ permutes with the join operator $\vee$, thus the function also preserves lattice joins.

Proposition 12 *The mapping φ is **join-preserving**:*

$$\forall c_1, c_2 \in \mathcal{C}, \varphi(c_1 \vee c_2) = \varphi(c_1) \vee_\times \varphi(c_2).$$

PROOF. (sketch) Observe that joins in a concept lattice involve intersection of the intents of the argument concepts, just as φ involves intersection of the partial attribute sets. Furthermore, a join over the product is simply the product of the point-wise joins. Finally, the associative property of intersection entails the above equality.

All the above properties have practical implications that jointly enable an easy "recovery" of the apposition lattice structure from the partial lattices. For example, the injection allows the concepts of the apposition lattice to be identified with pairs of concepts from the partial lattices. Similarly, the last two properties state that the global lattice $\mathcal{L}$ is isomorphic to its image under φ on $\mathcal{L}_\times$ and that this image is a proper sub-lattice of $\mathcal{L}_\times$.

Similar properties could be established about the ψ mapping. A first result is that ψ preserves order in the product lattice.

Proposition 13 *The mapping ψ is **order-preserving**:*

$$\forall (c_1, c_2), (c_3, c_4) \in \mathcal{C}_\times, (c_1, c_2) \leq_\times (c_3, c_4) \Rightarrow \psi(c_1, c_2) \leq \psi(c_3, c_4).$$

PROOF. (sketch) Trivially follows from the definition of the lattice order based on the inclusion of extents and the definition of the ψ mapping based on extent intersection.

Moreover, all sets $\psi^{-1}(c)$ of antecedents for a global concept c are convex subsets of $\mathcal{C}_\times$.

Proposition 14 *The classes of the equivalence relation induced by ψ^{-1} are convex subsets of $\mathcal{C}_\times$, i.e., given c in $\mathcal{C}$ and $(c_1, c_2), (c_3, c_4)$ in $\psi^{-1}(c)$:*

$$\forall (c_5, c_6) \in \mathcal{C}_\times,\ (c_1, c_2) \leq_\times (c_5, c_6) \leq_\times (c_3, c_4) \Rightarrow \psi(c_5, c_6) = c.$$

PROOF. (sketch) It is based upon the fact that the extent intersection is monotonic with respect to the order in $\mathcal{L}_\times$ (see the definition of the function $\mathcal{R}$ in the next section).

3.4 Exploring the apposition for lattice computation

Apposition/subposition operations were intended to ease the visualization of concept lattices and do not have straightforward computational interpretation. Thus, the only related algorithmic problem is the detection of the full nodes of the NLD, i.e., the images of the global concepts by φ. To our knowledge, no efficient algorithm has been designed up until now to solve the problem.

Our claim is that, although apposition/subposition have been oriented towards visualization, they may underlie the constructing of the global lattice from the partial ones. For this purpose, nodes and links of the global lattice may

be suitably "recovered" from the product lattice. Actually, an appropriate procedure is necessary for each of the following three tasks:

- identification of global concepts through their respective images by φ,
- computation of the intent and the extent of each global concept,
- computation of the cover relation of $\mathcal{L}$, i.e., the set of lower covers for each concept.

In this new setting, φ and ψ cannot be directly applied because only the product lattice is supposed known. Therefore, new structural results are required to enable the computation of $\mathcal{L}$ by looking only at $\mathcal{L}_\times$, i.e., without a global view of the data.

4 Characterizing the global lattice

In this section, we provide some theoretical results and useful structural properties that help obtain efficient algorithmic solutions for the above listed problems.

4.1 Structural properties

The first group of problems to tackle concerns the constructing of the components of the global lattice $\mathcal{L}$ from the lattices $\mathcal{L}_1$ and $\mathcal{L}_2$, or, equivalently, from the lattice $\mathcal{L}_\times$. For this purpose, we should first identify a subset of the ground (concept) set in $\mathcal{L}_\times$ that reflects the structure in $\mathcal{L}$. Recall that the function φ maps the lattice $\mathcal{L}$ into an isomorphic sub-structure of $\mathcal{L}_\times$. Hence, our first concern will be to provide characteristic properties of the sub-structure. We shall then see how these elements compare within $\mathcal{L}_\times$ in order to establish the order in $\mathcal{L}$.

4.1.1 Localizing $\varphi(c)$

Our first aim is to find a necessary and sufficient condition for an element $\hat{n} = (c_1, c_2)$ in $\mathcal{L}_\times$ to be the image of a concept $c = (X, Y)$ in $\mathcal{L}$ by φ, i.e. $\varphi(c) = \hat{n}$. Suppose this is actually the case. Recall that, by the definition of φ, $c_1 = ((Y \cap A_1)', Y \cap A_1)$ and $c_2 = ((Y \cap A_2)', Y \cap A_2)$. We claim that c is, in turn, the image of $\hat{n}$ by ψ. In fact, when applied to (c_1, c_2) with the above structure, the function gives (see Definition 9 in Section 3): $((Y \cap A_1)' \cap (Y \cap A_2)', ((Y \cap A_1)' \cap (Y \cap A_2)')')$.

For example, with $c = (5678, ad)$ the respective partial concepts are $c_1 =$

$(5678, ad)$ and $c_2 = (12345678, \emptyset)$, (i.e., node $c_{\#4}$ of $\mathcal{L}_1$ and node $c_{\#1}$ of $\mathcal{L}_2$ in Figure 2, respectively). Hence, the product node $\varphi(c)$ is $\hat{n} = (c_{\#4}, c_{\#1})$. Observe now that the extent of the above concept is made up of all the objects which have simultaneously all the attributes in $Y \cap A_1$ and those in $Y \cap A_2$ (see the example in Figure 4). This basically means, since $A_1 \cup A_2 = A$, that the extent is exactly the set of objects having Y, i.e. Y'. Consequently, the concept may be simplified to Y', Y'' which is exactly X, Y. Indeed, the inverse mapping of $(c_{\#4}, c_{\#1})$ retrieves 5678 as the extent of the image concept $\psi(\hat{n})$ which identifies the initial concept c. In summary, for any concept c, its image by φ is among the nodes mapped by ψ into c.

Proposition 15 $\varphi(c) \in \psi^{-1}(c)$.

The next step is to find a characterization of $\psi^{-1}(c)$ within $\mathcal{L}_\times$ and a property of $\varphi(c)$ within that set. Observe that the use of $\psi^{-1}(c)$ as an intermediate structure will help limit the search of a particular element to a small portion of the global lattice $\mathcal{L}_\times$.

4.1.2 Characterization of $\psi^{-1}(c)$

The characterization of $\psi^{-1}(c)$ follows from the trivial observation that a node $\hat{n} = (c_1, c_2)$ in $\mathcal{L}_\times$ is mapped to a concept $c = (X, Y)$ in $\mathcal{L}$ by ψ if and only if the extent of c is exactly the intersection of the extents of c_1 and c_2, say X_1 and X_2: $X = X_1 \cap X_2$. The concept pairs in $\psi^{-1}(c)$ are therefore distinguished by the intersection of the respective extents which defines a set-valued function:

Definition 16 *The function $\mathcal{R} : \mathcal{C}_\times \to \mathcal{P}(O)$ is such that*

$$\mathcal{R}(((X_1, Y_1), (X_2, Y_2))) = X_1 \cap X_2.$$

The values of $\mathcal{R}$ for a subset of product elements are listed in Table 1.

Table 1
The values of $\mathcal{R}$ for a selected subset of $\mathcal{C}_\times$ (see Figures 2 and 3).

Node $\hat{n}$	Value of $\mathcal{R}(\hat{n})$
$(c_{\#6}, c_{\#1})$	$\{5, 6\}$
$(c_{\#7}, c_{\#1})$	$\{6, 7, 8\}$
$(c_{\#4}, c_{\#2})$	$\{7\}$
$(c_{\#4}, c_{\#3})$	$\{5, 6, 8\}$
$(c_{\#4}, c_{\#4})$	$\emptyset$

Node $\hat{n}$	Value of $\mathcal{R}(\hat{n})$
$(c_{\#7}, c_{\#2})$	$\{7\}$
$(c_{\#7}, c_{\#3})$	$\{6, 8\}$
$(c_{\#7}, c_{\#4})$	$\emptyset$
$(c_{\#2}, c_{\#2})$	$\emptyset$
$(c_{\#2}, c_{\#3})$	$\{5, 6\}$

4.1.3 Characterization of $\varphi(c)$

The elements of the lattice $\mathcal{L}_\times$ may be divided into classes sharing the same value of $\mathcal{R}$, $[n]_\mathcal{R}$. These classes correspond to the equivalence classes induced by ψ on $\mathcal{C}_\times$. Hence, $\mathcal{R}$ provides a (locally computable) characterization for $\psi^{-1}(c)$. For example, the class $[(c_{\#7}, c_{\#2})]_\mathcal{R}$ including all the nodes that share the value $\{7\}$ is:

$$[(c_{\#7}, c_{\#2})]_\mathcal{R} = \{(c_{\#1}, c_{\#2}), (c_{\#3}, c_{\#2}), (c_{\#4}, c_{\#2}), (c_{\#7}, c_{\#2})\}.$$

Actually, the investigated nodes of $\mathcal{C}_\times$ are those that are minimal for their class. More formally, a given $\hat{n}$ is the image of a concept c by φ if and only if it is *minimal* in $[\hat{n}]_\mathcal{R}$ with respect to the product lattice order $\leq_\times$. This result is expressed by the following proposition.

Proposition 17 $\forall \hat{n} \in \mathcal{C}_\times \ (\ \exists c \in \mathcal{C} : \hat{n} = \varphi(c)\ ) \Leftrightarrow \min([\hat{n}]_\mathcal{R}) = \{\hat{n}\}$.

PROOF. Let $\hat{n} = (c_1, c_2)$ and let $c_1 = (X_1, Y_1)$ and $c_2 = (X_2, Y_2)$. To prove this property, it is enough to observe that the smallest concept in L_i $(i = 1, 2)$ which incorporates X in its extent and the concept $((Y \cap A_i)', Y \cap A_i) = ((X' \cap A_i)', X' \cap A_i)$ are the same concept.

For example, the node $(c_{\#7}, c_{\#2})$ is minimal, with respect to the product order, in its class, and $(c_{\#6}, c_{\#3})$ is minimal in the class of nodes evaluated to $\{5, 6\}$ by $\mathcal{R}$.

4.1.4 Computing the concept intent and extent

The above proposition identifies the nodes in $\mathcal{L}$ with the minimum of each class $[\hat{n}]_\mathcal{R}$ in $\mathcal{L}_\times$. The extents of a concept $c = \psi(c_1, c_2)$ can be computed as the intersection of the extents of c_1 and c_2. Also the intent of c may be computed locally, i.e., without looking in the data table: it is the union of the intents of c_1 and c_2 where $(c_1, c_2) = \varphi(c)$.

Proposition 18 *For a pair of partial concepts $c_1 = (X_1, Y_1)$ and $c_2 = (X_2, Y_2)$ and a global concept $c = (X, Y)$ such that $\varphi(c) = (c_1, c_2)$,*

$$X = X_1 \cap X_2 \ \textbf{\textit{and}}\ Y = Y_1 \cup Y_2.$$

PROOF. Observe that both c_1 and c_2 are minimal and therefore $X_i = X''$, i.e., the closure of X for $\mathcal{K}_i$ $(i = 1, 2)$. Thus, each Y_i equals X' within $\mathcal{K}_i$ whereas their union $Y_1 \cup Y_2$ is X' in $\mathcal{K}$.

4.1.5 Characterizations of the lower covers of a node

The next step of the lattice computation deals with the order between its nodes. This is usually manipulated in terms of the cover relation $\prec_{\mathcal{L}}$ whose pairs have to be computed. Observe that $\prec_{\mathcal{L}}$ does not strictly correspond to the cover relation in $\mathcal{L}_{\times}$, $\prec_{\times}$ via φ but rather to the broader order $\leq_{\times}$. Thus, a straightforward computation of $\prec_{\mathcal{L}}$ could require the exploration of the entire $\leq_{\times}$, which is a rather expensive task. In the following, a characterization of $\prec_{\mathcal{L}}$ is given that uses only the information about $\prec_{\times}$. It detects the set of *lower covers* of a concept c (denoted by $Cov^l(c)$) by only looking at the lower covers of its image $\varphi(c)$ in $\mathcal{L}_{\times}$.

First, recall the embedding property of φ, i.e., $\underline{c} \leq_{\mathcal{L}} c \Leftrightarrow \varphi(\underline{c}) \leq_{\times} \varphi(c)$. Next, consider a node $\hat{n} = (c_1, c_2)$ with $c_1 = (X_1, Y_1)$ and $c_2 = (X_2, Y_2)$ such that $\{\hat{n}\} = \min([\hat{n}]_{\mathcal{R}})$ and let $c = \psi(\hat{n})$. We are looking for the nodes $\tilde{n}$ with the following properties:

$$\{\tilde{n}\} = \min([\tilde{n}]_{\mathcal{R}}) \text{ and } \underline{c} \prec_{\mathcal{L}} c \text{ where } \underline{c} = \psi(\tilde{n}).$$

Consider a fixed $\tilde{n}$. According to the previous remark, it is less than or equal to $\hat{n}$. We shall now prove that there is a node $\bar{n}$ which is a lower cover of $\hat{n}$ and has the same image by ψ as $\tilde{n}$:

$$\bar{n} \prec_{\times} \hat{n} \text{ and } \psi(\bar{n}) = \psi(\tilde{n}).$$

In fact, among the lower covers of $\hat{n}$, $Cov^l(\hat{n})$, there is no element that has the same value of $\mathcal{R}$ as $\hat{n}$ (by definition). Furthermore, there is at least one $\bar{n}$ which is greater than or equal to our $\tilde{n}$. According to the embedding property, its ψ image is greater than or equal to that of $\tilde{n}$, $\psi(\tilde{n}) \leq_{\mathcal{L}} \psi(\bar{n})$. Similarly, the image is less than the image of $\hat{n}$, $\psi(\bar{n}) \leq_{\mathcal{L}} \psi(\hat{n})$. However, we already know that the latter is preceded by $\psi(\tilde{n})$ in $\mathcal{L}$. We conclude that $\psi(\tilde{n})$ and $\psi(\bar{n})$ are equal. In other words, for all pairs $\underline{c} \prec_{\mathcal{L}} c$ in $\mathcal{L}$, there is a pair $\bar{n} \prec_{\times} \hat{n}$ such that $\psi(\bar{n}) = \underline{c}$ and $\varphi(c) = \hat{n}$. For example, take the node $\hat{n} = (c_{\#2}, c_{\#1})$ whose value of $\mathcal{R}$ is 12356. $\hat{n}$ is the image of $c = (12356, ab)$ which is preceded in $\mathcal{L}$ by $\underline{c} = (56, abdf)$. The latter is mapped into $\hat{n} = (c_{\#6}, c_{\#3})$ by φ. The node $(c_{\#6}, c_{\#3})$ is a predecessor of $(c_{\#2}, c_{\#1})$ in the product lattice but not a lower cover (for example, the node $(c_{\#2}, c_{\#3})$ is between them). However, there is at least one lower cover of $(c_{\#2}, c_{\#1})$ with the same value of $\mathcal{R}$ as $(c_{\#6}, c_{\#3})$. For example, one such node is $(c_{\#6}, c_{\#1})$. Unfortunately, the set of lower covers of a node $\hat{n} = \varphi(c)$ is not necessarily limited to nodes whose images under ψ are predecessors of c. There may be other nodes in $Cov^l(\hat{n})$ whose images, although less than c are not its lower covers. Actually, the nodes we look for are those with values of $\mathcal{R}$ maximal within $Cov^l(\hat{n})$.

Proposition 19 *For each $\hat{n}$ such that $\exists c, \hat{n} = \varphi(c)$, and for each one of its*

lower covers $\bar{n} \in Cov^l(\hat{n})$*, we have:*

$$\psi(\bar{n}) \prec_{\mathcal{L}} c \quad \textbf{if and only if} \quad \mathcal{R}(\bar{n}) \in \max(\{\mathcal{R}(\tilde{n}) | \tilde{n} \in Cov^l(\hat{n})\}).$$

Instead of providing a general proof, we show a detailed example of the computation of the lower covers of the concept $c = (5678, ad)$. The relevant parts of the lattices $\mathcal{L}_\times$ and $\mathcal{L}$ are drawn in Figure 4. The concepts of the global lattice are linked to their images by φ. The concept's image is the node

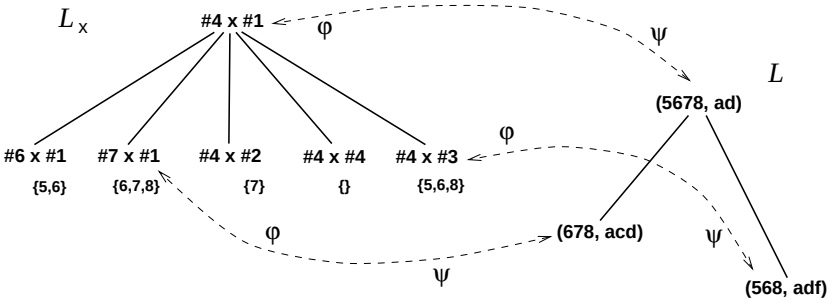

Fig. 4. Part of the product lattice $\mathcal{L}_\times$ (left) representing the node $(c_{\#4}, c_{\#1})$ (product nodes $(c_{\#i}, c_{\#j})$ are denoted by $\#i \times \#j$) and its lower covers together with the corresponding part of the global lattice $\mathcal{L}$ (right). Nodes in $\mathcal{L}$ are linked to their φ-images by dashed lines. Both lower covers of $(5678, ad)$ in $\mathcal{L}$ correspond to lower covers of $(c_{\#4}, c_{\#1})$ in $\mathcal{L}_\times$ whose $\mathcal{R}$ values are maximal: $(c_{\#7}, c_{\#1})$ and $(c_{\#4}, c_{\#3})$.

$(c_{\#4}, c_{\#1})$ whose lower covers are $(c_{\#6}, c_{\#1})$, $(c_{\#7}, c_{\#1})$, $(c_{\#4}, c_{\#2})$, $(c_{\#4}, c_{\#3})$ and $(c_{\#4}, c_{\#4})$. The respective $\mathcal{R}$ values are listed in the left part of Table 1. The maximal values of $\mathcal{R}$ within the set of lower covers of $(c_{\#4}, c_{\#1})$ are thus: $\max(\{\mathcal{R}(\tilde{n}) | \tilde{n} \in Cov^l(\varphi(c))\}) = \{568, 678\}$. These values correspond to the extents of the concepts $(568, adf)$ and $(678, acd)$ which constitute exactly the set of lower covers of the concept c, $Cov^l(c)$, in the global lattice.

Theorem 20 *Given a context* $\mathcal{K}$ *with its concept lattice* $\mathcal{L}$ *and a pair of concept lattices* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *over contexts* $\mathcal{K}_1$ *and* $\mathcal{K}_2$*, such that* $\mathcal{K}_1 | \mathcal{K}_2 = \mathcal{K}$*,* $\forall c = (X, Y) \in \mathcal{L}$*:*

- $\min(\psi^{-1}(c)) = \{\varphi(c)\}$,
- $X = \mathcal{R}(\varphi(c))$,
- $Cov^l(c) = \max(\{\psi(\bar{n}) | \bar{n} \in Cov^l(\varphi(c))\})$

5 Constructing the lattice

The results presented in the previous section are transformed into an algorithmic procedure that physically constructs the global lattice $\mathcal{L}$ from both partial lattices. This is in turn extended to a full-scale lattice constructing algorithm

by integrating it into a "divide-and-conquer" procedure. In what follows, we present these procedures together with some results about their asymptotic complexity.

5.1 Main algorithm

The main procedure CONSTRUCT-LATTICE (see Algorithm 1) constructs the Galois lattice starting from the initial context. It is recursive, with a base case being a single-attribute context (see next paragraph for a description of CONSTRUCT-LATTICE-T). Larger contexts are first split into two parts by selecting a subset of attributes (function SELECT). Then, the main procedure is recursively called on each subcontext, thus retrieving a pair of partial lattices. Finally, the global lattice is obtained by an assembly of the partial ones (see Section 5.3 for the description of ASSEMBLY).

```
 1: procedure CONSTRUCT-LATTICE(In: K = (O, A, I) a context; Out: L = ⟨C, ≤_L⟩
    the lattice of K)
 2:
 3: if ‖A‖ > 1 then
 4:    A_1 ← SELECT(A); A_2 ← A - A_1
 5:    K_1 ← (O, A_1, I ∩ O × A_1) ; K_2 ← (O, A_2, I ∩ O × A_2)
 6:    L_1 ← CONSTRUCT-LATTICE(K_1) ; L_2 ← CONSTRUCT-LATTICE(K_2)
 7:    L ← ASSEMBLY(L_1, L_2)
 8: else
 9:    L ← CONSTRUCT-LATTICE-T(K)
10: return L
```

Algorithm 1: Constructing a Galois lattice with a "divide-and-conquer"-like strategy.

It is important to note that the function SELECT may be implemented in various ways ranging from straightforward equal-size splitting to an advanced calculation of balanced sub-contexts. Such a function may have a strong impact on the practical performance of the algorithm as the sizes of the partial lattices depend on it. We will not dwell on this point as it relates to complex combinatorial problems and is subject to a separate on-going study.

5.2 One-attribute lattice computation

The base case of the lattice computation occurs when a single-column context is reached. With such a small attribute set, say $\{a\}$, the lattice may have no more than two concepts. Indeed, all possible intents are $\{a\}$ and $\emptyset$ corresponding respectively to the lattice *infimum* and *supremum*. In addition, there is one case where the lattice is made up of a single concept: it occurs when a is

shared by all the objects in O, in which case the only concept that remains is (O,a).

```
 1: procedure CONSTRUCT-LATTICE-T(In: 𝒦 = (O, {a}, I) a context; Out: ℒ =
    ⟨𝒞, ≤_ℒ⟩ the lattice of 𝒦)
 2:
 3: ℒ ← ∅
 4: O⁺ ← {o|(o, a) ∈ I}
 5: c ← NEW-CONCEPT(O⁺, {a})
 6: ℒ ← ℒ ∪ {c}
 7: if O⁺ ≠ O then
 8:    c̃ ← NEW-CONCEPT(O, ∅)
 9:    NEW-LINK(c,c̃)
10:    ℒ ← ℒ ∪ {c̃}
11: return ℒ
```

Algorithm 2: Constructing a Galois lattice over a single-attribute context.

In a first step, the procedure CONSTRUCT-LATTICE-T (see Algorithm 2) separates the extent of the lattice infimum, i.e., objects having a and constructs the concept. Then, it creates a distinct supremum if needed (i.e., when a proper subset of O share the attribute a).

5.3 Assembly of partial lattices

The fundamental operation of our lattice constructing algorithm is the assembly of two partial lattices drawn from the complementary fragments of the context.

5.3.1 Principles of the algorithm

The main step of the algorithm is a traversal of the product lattice. The traversal is made in a bottom-up way, following a *linear extension* of the lattice order. The linear extension is the result of a sorting task over each of the partial concept sets.

For each pair of concepts, the intersection of their extents is computed and checked for minimality. Pairs with minimal $\mathcal{R}$ values generate a new concept in $\mathcal{L}$ whose intent and extent are computed according to the results of Section 4. The concept's lower covers in $\mathcal{L}$ are then detected among the global concepts already generated by the bottom-up traversal.

5.3.2 Main data structures and auxiliary primitives

The concept lattices are stored as lists of concepts in which a concept is a record with fields like intent, extent and list of lower covers (accessible by dedicated primitives $Extent$, $Intent$, etc.). An auxiliary field indicates the rank of a concept in the sorted list (primitive **rank**). The mapping ψ is simulated by *Embed*, a two-dimensional array of concepts where indices are concept ranks in the partial lattices. *Embed* simulates the function $\mathcal{R}$ through the extents of the stored concepts.

Except for the above abstract structure-related primitives, the algorithm uses ProductLowerCovers to generate the lower covers in $\mathcal{L}_\times$ of a product node. Another group of primitives carry out maintenance tasks within the physical representation of the lattices (New-Concept, New-Link).

```
 1: procedure Assembly(In: L1 = ⟨C1, ≤L1⟩, L2 = ⟨C2, ≤L2⟩ lattices; Out: L =
    ⟨C, ≤L⟩ a lattice)
 2:
 3: Local   : Embed : array [0..max,0..max] of concepts
 4:
 5: L ← ∅  {Init}
 6: Sort(C1); Sort(C2) {Sort according to a linear extension of ≤Li}
 7: for all ci in C1 do
 8:    for all cj in C2 do
 9:       E ← Extent(ci) ∩ Extent(cj)
10:       PsiImages ← ∅
11:       for all (c̄, c̲) in ProductLowerCovers(ci, cj) do
12:          PsiImages ← PsiImages ∪ {Embed(rank(c̄), rank(c̲))}
13:       c ← Find-Psi(E, PsiImages)
14:       if c = NULL then
15:          c ← New-Concept(E, Intent(ci) ∪ Intent(cj))
16:          Update-Order(c, PsiImages)
17:          L ← L ∪ {c}
18:       Embed[rank(ci), rank(cj)] ← c
19: return L
```

Algorithm 3: Assembling the global Galois lattice from a pair of partial ones.

5.3.3 The algorithm code

For convenience, the major tasks of the overall algorithm have been separated into distinct procedures, each one provided with its own description.

A pre-processing step sorts both sets of concepts $\mathcal{C}_{\mathcal{K}_1}$ and $\mathcal{C}_{\mathcal{K}_2}$ according to a linear extension of the respective lattice orders (see Section 5.5 for the code of Sort). In the main step, a nested **for** loop simulates the traversal of the product lattice. At each product node, the extent intersection of the com-

pound concepts is stored in E for a further test of minimality. The test also requires the set of R values over the lower covers of the current node (retrieved by ProductLowerCovers). These are computed as the extents of the respective images under ψ, retrieved from *Embed*. The images are first stored in *PsiImages* and then passed as a parameter to Find-Psi, the function that finds the image under ψ of the current node if it already exists (non generator node). In this case, no further processing is required. Otherwise, i.e., with a NULL result from Find-Psi meaning that the R value is yet unknown, a new concept is created, then its lower covers are singled out among the candidates in *PsiImages*. In both cases, the array *Embed* is updated.

5.3.4 Example

Table 2 provides an illustration of the way in which the construction of the global lattice proceeds. It contains the trace of the examination of the first ten nodes taken in the order of the nested loop in Algorithm 3. For each node, its $\mathcal{R}$ value is given. For generator nodes (i.e., nodes (c_i, c_j) that generate a concept in $\mathcal{L}$), the generated concept is also indicated whereas for the remaining nodes, a lower cover that shares the same $\mathcal{R}$ (and thus makes the minimality test fail) is indicated. For example, the node $(c_{\#8}, c_{\#3})$ has $\mathcal{R}$ value of $\{6\}$ which is minimal. It therefore generates the global concept $(6, abcdf)$. The node $(c_{\#8}, c_{\#1})$ is also mapped to $\{6\}$, but as its lower cover, $(c_{\#8}, c_{\#3})$, also possesses that value, it generates no global concept.

Table 2
The trace of Algorithm 3 for the first ten nodes of the product lattice in Figure 3.

Node (c_i, c_j)	$\mathcal{R}$	**Generated concept**	Lower cover in $[(c_i, c_j)]_{\mathcal{R}}$
$(c_{\#8}, c_{\#7})$	$\emptyset$	$(\emptyset, abcdefghi)$	–
$(c_{\#8}, c_{\#6})$	$\emptyset$	–	$(c_{\#8}, c_{\#7})$
$(c_{\#8}, c_{\#5})$	$\emptyset$	–	$(c_{\#8}, c_{\#6})$
$(c_{\#8}, c_{\#2})$	$\emptyset$	–	$(c_{\#8}, c_{\#7})$
$(c_{\#8}, c_{\#3})$	$\{6\}$	$(6, abcdf)$	–
$(c_{\#8}, c_{\#4})$	$\emptyset$	–	$(c_{\#8}, c_{\#5})$
$(c_{\#8}, c_{\#1})$	$\{6\}$	–	$(c_{\#8}, c_{\#3})$
$(c_{\#5}, c_{\#7})$	$\emptyset$	–	$(c_{\#8}, c_{\#7})$
$(c_{\#5}, c_{\#6})$	$\emptyset$	–	$(c_{\#5}, c_{\#7})$
$(c_{\#5}, c_{\#5})$	$\{3\}$	$(3, abcgh)$	–

The integration of a newly created concept, Update-Order, as well as the

function FIND-PSI are described in Section 5.4.

5.4 Key operations

In the following, we provide a detailed description of the main operations on concepts, intents, extents and links, used by Algorithm 3. These are essential for the effective implementation of our strategy and for the assessment of its algorithmic complexity.

5.4.1 Updating the order

The integration of a new concept c into the partially constructed lattice structure (the $\mathcal{L}$ list of concepts) involves the computation of the lower covers of c and the physical creation of the corresponding links. It is a two-step process described in Algorithm 4. First, the candidate concepts are sorted (recall those are the images under ψ of all nodes of the product lattice which immediately precede the generator of c, i.e., (c_i, c_j)). The sort is done by increasing order of

```
 1: procedure UPDATE-ORDER(In: c a concept, Candidates a set of concepts)
 2:
 3: Modifies : Cov^l(c)
 4: Local    : KnownIntent : set of attributes
 5:
 6: SORT(Candidates)        {Sort according to a linear extension of the
    inverse of ≤_L}
 7: KnownIntent ← Intent(c)
 8: for all c̃ in Candidates do
 9:    if Intent(c̃) ∩ KnownIntent = Intent(c) then
10:       NEW-LINK(c̃,c)
11:       KnownIntent ← KnownIntent ∪ Intent(c̃)
```

Algorithm 4: Update the order relation of the global lattice.

the intent sizes (the SORT primitive is described later on). Next, the concepts are examined one by one, each time testing whether a maximality condition is met. The condition is inspired by the Bordat algorithm [3] and ensures a minimal number of set-theoretic operations. As the cover relation of the lattice is updated with each lower cover detection, at the end of the algorithm the concept c is completely integrated in the lattice structure.

As an illustration of the way Algorithm 4 works, we provide the trace of its execution on the first six product nodes that generate global concepts. The related values of the $\mathcal{R}$ function for each of the nodes as well as for each of their lower covers in $\mathcal{L}_\times$ are given in Table 3. The table also provides the global

Table 3
A trace of Algorithm 4: the first six global concepts for the example in Figure 3.

Full nodes	$\mathcal{R}$	potential covers with $\mathcal{R}$ values	$\mathcal{R}$ on actual covers	**Cover nodes**
$(c_{\#8}, c_{\#7})$	$\emptyset$	–		
$(c_{\#8}, c_{\#3})$	$\{6\}$	$\mathcal{R}(c_{\#8}, c_{\#7}) = \emptyset$	$\emptyset$	$(\emptyset, abcdefghi)$
$(c_{\#5}, c_{\#5})$	$\{3\}$	$\mathcal{R}(c_{\#5}, c_{\#6}) = \emptyset$ $\mathcal{R}(c_{\#8}, c_{\#5}) = \emptyset$	$\emptyset$	$(\emptyset, abcdefghi)$
$(c_{\#5}, c_{\#1})$	$\{3, 6\}$	$\mathcal{R}(c_{\#5}, c_{\#2}) = \emptyset$ $\mathcal{R}(c_{\#5}, c_{\#3}) = \{6\}$ $\mathcal{R}(c_{\#5}, c_{\#4}) = \{3\}$ $\mathcal{R}(c_{\#8}, c_{\#3}) = \{6\}$	 $\{6\}$ $\{3\}$	 $(6, abcdf)$ $(3, abcgh)$
$(c_{\#6}, c_{\#3})$	$\{5, 6\}$	$\mathcal{R}(c_{\#6}, c_{\#7}) = \emptyset$ $\mathcal{R}(c_{\#8}, c_{\#3}) = \{6\}$	 $\{6\}$	 $(6, abcdf)$
$(c_{\#2}, c_{\#2})$	$\{7\}$	$\mathcal{R}(c_{\#7}, c_{\#7}) = \emptyset$ $\mathcal{R}(c_{\#8}, c_{\#2}) = \emptyset$	$\emptyset$	$(\emptyset, abcdefghi)$
$(c_{\#7}, c_{\#3})$	$\{6, 8\}$	$\mathcal{R}(c_{\#7}, c_{\#7}) = \emptyset$ $\mathcal{R}(c_{\#8}, c_{\#3}) = \{6\}$	 $\{6\}$	 $(6, abcdf)$

concepts which are actual lower covers of the generated concept together with the respective values of $\mathcal{R}$ which helped generate them.

5.4.2 Test for new extents and the computation of ψ

The detection of the minima in the classes induced by $\mathcal{R}$ is efficiently carried out by comparing set cardinalities instead of comparing sets themselves. In fact, as we have shown in Section 4.1, the mapping ψ is monotonic and therefore the function $\mathcal{R}$ is monotonic too. This means, for a given node $\hat{n} = (c_1, c_2)$, its value for $\mathcal{R}$ is a superset of the value on an arbitrary lower cover of $\hat{n}$, say $\bar{n}$: $\mathcal{R}(\bar{n}) \subseteq \mathcal{R}(\hat{n})$. For a node $\hat{n}$ which satisfies the condition of Proposition 17, the superset condition is strict, i.e., $\mathcal{R}(\bar{n}) \subset \mathcal{R}(\hat{n})$ for any lower cover $\bar{n}$. An equivalent condition holds on set cardinalities: the size of $\mathcal{R}(\hat{n})$ is strictly greater than the size of $\mathcal{R}(\bar{n})$.

Consequently, the property of a node $\hat{n}$ being minimal in its class can be checked by only comparing set cardinalities: $\hat{n}$ is minimal if and only if $\|\mathcal{R}(\hat{n})\|$ is different from any cardinality of $\mathcal{R}$ on a lower cover.

```
1: function FIND-PSI(In: E a set of objects, Candidates a set of concepts ; Out: c
   a concept)
2:
3: m ← ||E||
4: for all c in Candidates do
5:    if ||Extent(c)|| = m then
6:       return c
7: return NULL
```

Algorithm 5: Efficient lookup for newly met extents.

The test procedure is described by Algorithm 5. It is noteworthy that the second parameter is a set of global concepts which represent the images of the lower covers of the current node under ψ. The first parameter is thus to be compared to their extents.

5.5 Auxiliary primitives

In what follows, efficient algorithms for some of the auxiliary primitives used by the previous procedures are suggested.

5.5.1 Concept sort

The traversal of a concept set $\mathcal{C}$, not necessarily equal to the entire $\mathcal{C}_\mathcal{K}$, with respect to the lattice order $\leq_\mathcal{K}$, is achieved through a preliminary sorting of $\mathcal{C}$. The sorting procedure typically produces a *linear extension* of $\leq_\mathcal{K}$ on C. Such an extension could be easily, but somewhat inefficiently, computed by checking inclusions between concept intents/extents. In its more efficient version, the sorting, just as the minimality check for product nodes described above, compares intent cardinalities instead of intents. This could be done in a linear

```
 1: procedure SORT(In/Out: C = {c_1, c_2, ..., c_l})
 2:
 3: Local  : Bunches : array [0..||A||] of sets
 4:        : Order : list of concepts
 5:
 6: Order ← ∅
 7: for i from 1 to l do
 8:    Bunches[||Intent(c_i)||] ← Bunches[||Intent(c_i)||] ∪ {c_i}
 9: for i from 1 to ||A|| do
10:    for all c ∈ Bunches[i] do
11:       Order ← ⟨c⟩ & Order {c becomes the head of Order}
12: C ← Order
```

Algorithm 6: Linear-time sorting of the concept set

time as Algorithm 6 shows. Actually, the required operations are the split of

C into slices of equal intent cardinalities and the subsequent enumeration of these groups in descending order. The use of an array indexed by intent cardinality to store the slices makes slice sorting unnecessary. Thus, the whole procedure takes only a time which is linear in the number of concepts.

5.5.2 Lower covers of a node in the product lattice

The function ProductLowerCovers which computes the lower covers of a node $\hat{n} = (c_i, c_j)$ is not explicitly described here. In fact, it adds only a limited computational overhead since it relies exclusively on information stored at the concepts c_i and c_j in the computation. In fact, the lower covers of $\hat{n}$ in the product lattice are exactly the nodes of the form (c_i, c'_j) or (c'_i, c_j) where c'_i is an arbitrary lower cover of the concept c_i in the lattice L_1 and c'_j is an arbitrary lower cover of c_j in L_2.

5.6 Complexity issues

Let the number of concepts in the partial lattices $\mathcal{L}_1$ and $\mathcal{L}_2$ be respectively l_1 and l_2 and the number of concepts in $\mathcal{L}$ be l. Let also m be the total number of attributes and k the number of objects. In our complexity assessment, we shall also use the parameter $d(\mathcal{L})$ which is the branching factor, i.e., the maximal number of lower covers of a node in the lattice $\mathcal{L}$. The above notations are summarized in the following table.

Variable	**Stands for**
l_i	the number of concepts in $\mathcal{L}_i$ $(i = 1, 2)$
l	the number of concepts in $\mathcal{L}$
m	$\|A\|$
k	$\|O\|$
$d(\mathcal{L})$	maximal number of lower covers of a node in $\mathcal{L}$

At this step, we consider only the cost of the assembly operation Assembly. The cost of the recursive calls of the global procedure will be provided in Section 6.

Consider the following basic facts. First, the number of lower covers in a concept lattice $\mathcal{L}$, $d(\mathcal{L})$, is at most equal to the number of attributes, m. Moreover, the number of lower covers in $\mathcal{L}_\times$, $d(\mathcal{L}_\times)$, is bounded above by the sum $d(\mathcal{L}_1) + d(\mathcal{L}_2)$. Finally, with a total order assumed both on O and A object/attribute sets, these collections may be canonically represented as

sorted lists of integers (ranks in the respective order). This allows all set-theoretic operations (e.g., $\cup$, $\cap$, /) to be executed in time linear in the size of the manipulated sets.

Among auxiliary primitives, the SORT procedure is linear in the number of its arguments, $\|\mathcal{L}\|$, whereas the minimality check for intent intersections (FIND-PSI) is linear in the number of lower covers in the product, $O(d(\mathcal{L}_1)+d(\mathcal{L}_2))$ [4]. The update of the precedence relation (UPDATE-ORDER), has time complexity which is a product of the number of the potential lower covers to check and the cost of the check (linear in m), $O((d(\mathcal{L}_1) + d(\mathcal{L}_2))m)$. The complexity of the basic operations is given in the following table.

Primitive	Asymptotic complexity
SORT	$O(l_1 + l_2)$
FIND-PSI	$O(d(\mathcal{L}_1) + d(\mathcal{L}_2))$
UPDATE-ORDER	$O((d(\mathcal{L}_1) + d(\mathcal{L}_2))m)$

The cost of the assembly algorithm heavily relies on the complexity of its dominant part, the nested **for** loop. This is divided into two major parts: the first one is related to the fixed part of the loop and the second one concerns the **if** statement. The first part is executed once for each node of the product, i.e., $l_1 l_2$ times. Its dominant complexity comes either from extent intersection (linear in k), or the computation of PRODUCTLOWERCOVER (linear in the candidate lower cover number, i.e., $d(\mathcal{L}_1) + d(\mathcal{L}_2)$), or minimality test (FIND-PSI) for extent intersection (linear in the actual lower cover number, i.e., $d(\mathcal{L})$). As both $d(\mathcal{L})$ and $d(\mathcal{L}_1) + d(\mathcal{L}_2)$ do not exceed m, the cost of the entire fixed part is $O((k + m)l_1 l_2$.

The **if** part of the loop body executes only on product nodes generating global concepts, i.e., $\|\mathcal{L}\|$ times. Its core complexity comes from the UPDATE-ORDER procedure, so the whole cost is $O((d(\mathcal{L}_1) + d(\mathcal{L}_2))ml)$ which further simplifies to $O(lm^2)$. In summary, the total complexity of the algorithm, beside partial lattice constructing, amounts to $O(l_1 l_2(m + k) + lm^2)$ which is bounded by:

$$O((k + m)(l_1 l_2 + lm)).$$

[4] In fact, the comparisons are carried on cardinalities, i.e., integer numbers, and therefore take constant time. Furthermore, the cardinalities of a concept intent/extent need only to be computed once, upon the concept creation. Therefore, at this point they are assumed given.

6 Discussion

It is generally admitted that the worst-case complexity of the main lattice constructing algorithms is $O((k+m)lmk)$, with the exception of the algorithm in [17] whose complexity is known to be $O((k+m)lm)$. However, comparing lattice constructing algorithms with respect to their asymptotic complexity is a delicate task. On the one hand, there may be exponentially many concepts, a fact which makes any algorithm inefficient. Fortunately, contexts contain, most of the time, only a polynomial number of concepts. On the other hand, there is no way of correctly evaluating the complexity of these algorithms with respect only to their input since the size of the lattice is hard to predict from parameters of the binary table [5] . Indeed, all previous studies of the practical performance of known algorithms have shown that there is no clear "best" algorithm. Instead, the relative performance of the different methods vary according to the nature of the data [6] . A key factor is thus the *density* measuring the relative number of X's in a table:

$$\omega_d = \frac{\|I\|}{\|O\|.\|A\|}.$$

For example, the incremental algorithm of Godin *et al.* is known to perform fast on sparse contexts, $\omega_d < 0.10$, but seems to lag behind competing methods with dense ones, $\omega_d > 0.5$. As an apparent explanation for this fact, the algorithm relies strongly on the number of concepts, which usually grows fast with ω_d. An additional difficulty for practically comparing algorithms is the size of the output (previous studies only considered small-size contexts, i.e., less than a thousand objects).

In summary, situating our own algorithm within the family of existing methods in an absolute way is hard. It means, in particular, transforming the complexity formula into a canonic form, i.e., getting rid of $l_1 l_2$. The question of how $l_1.l_2$ compares to lm amounts to estimating the size of the lattice.However, we claim that the only large contexts which are tractable at present, i.e., produce reasonable-size lattices, show linear growth of the concept set with respect to object number. In such contexts, $l_1.l_2$ is at most $O(lm)$ regardless of the way the attribute set is split. Only for those contexts, the global complexity of our divide-and-conquer strategy may be estimated by:

$$O((k+m)lm\log m).$$

The $\log m$ factor reflects the depth of the binary tree that results from the

[5] The problem of estimating the number of concepts from the context parameters was recently proved to be a hard one [13].

[6] Here the *nature* of a dataset does not refer to a well defined notion, but rather to a set of data parameters, whose influence is not well understood.

recursive splitting of contexts.

A deeper insight into the divide-and-conquer strategy is provided by a set of simulation runs. The tests include two classes of contexts, both of them involving only a linear number of concepts. Both are generated by a random procedure. The first generation uses a uniform probability distribution with a low density factor and simulates contexts from software engineering applications. The second one simulates the contexts derived from tables in relational databases which typically contain numerical and categorical data. A table based on a set of properties P is translated into a binary relation by scaling each property p into a set of Boolean attributes A_p. The resulting context has a particular form: given a row in the database table and a property p, the object o that models the row in the context K can have at most one of the attributes in A_p (exactly one if the respective table entry is not NULL). Thus, for each initial p, the extents of the scaled attributes in A_p are pair-wise disjoint: $\cap_{a_1,a_2 \in A_p}\{a_1, a_2\}' = \emptyset$. The tests have been carried out on an IBM

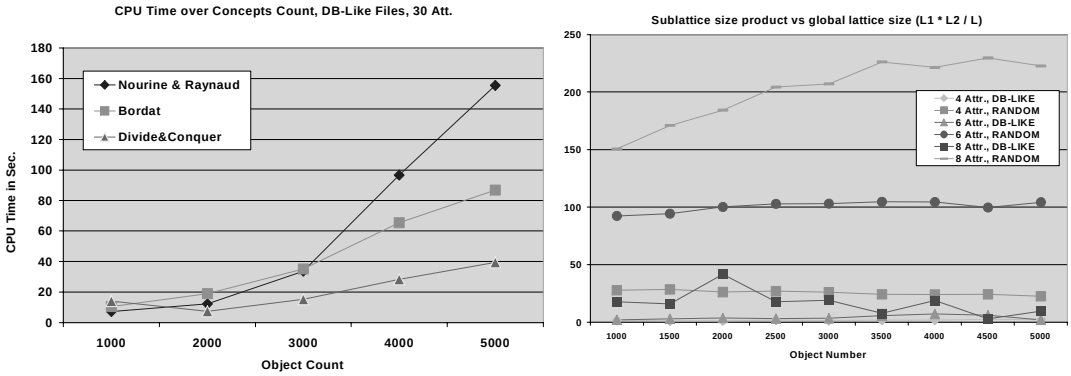

Fig. 5. *Left*: CPU-time diagrams for three algorithms: Bordat, Nourine and Raynaud (R&N), and Divide-and-Conquer (D&C). *Right*: Evolution of $l_1.l_2/l$ over object number.

PC platform (Pentium III, 192MB RAM, Windows 98). Figure 5 on the left shows the time used by three algorithms: Bordat's, Nourine and Raynaud's and our own algorithm over the same dataset. The results have been obtained on database-like contexts, by varying the number of objects between 1000 and 5000 with an increment of 500. Each point on a curve gives the average over 20 different runs.

As it may be seen on the left part of Figure 5, the divide and conquer algorithm outperforms the rest with a time growth close to linear in the number of objects. However, jumping to the conclusion that it is the perfect algorithm for contexts with small number of concepts would be an exaggeration since the results over randomly generated contexts (not included here for the sake of conciseness) do not confirm it. As a possible intuition as to why the divide-and-conquer approach works better on database-like contexts, we include a

second diagram showing the evolution of the ratio between $l_1.l_2$ and l over the same datasets (see Figure 5 on the right). With random sparse data, the ratio grows slowly and tends to stabilize, at relatively high level indeed, whereas database-like contexts show almost no growth.

To conclude, a plausible explanation of the above results seems to lie in the strong inner structure of the database-like contexts. This structure is successfully unveiled by splitting which makes the divide-and-conquer approach perform well whereas other algorithms simply fail to take advantage of it.

7 Conclusion and further research

We presented an approach for lattice assembly together with its theoretical foundations. It underlies a divide-and-conquer approach for constructing Galois/concept lattices whose worst-case complexity and practical performances have been reported.

An important result of our work is the complete characterization of the global Galois/concept lattice as a substructure of the direct product of the lattices related to partial contexts. This helps to establish clear links between partial and global lattices and therefore enables an efficient lattice assembly. Another contribution lies in a novel way of constructing lattices, i.e., from fragments, which may prove useful when data comes from different sources (e.g., in the process of constructing a data warehouse or mining for knowledge from a set of databases). Finally, the possibility of combining several partial results into a single global one allows a set of new lattice constructing algorithms to be devised, in particular algorithms implementing a hybrid batch-incremental strategies.

A large set of intriguing questions are yet to be answered as the work goes on. In the short term, potential improvements of the practical performance of our algorithm are to be examined. Among the set of open issues, we are currently focusing on the effects of "optimal" splits of the contexts, i.e., splits that keep the size of the partial lattices minimal. Another research avenue follows the removal of objects/attributes from a lattice (inverse incrementality) and goes to the challenging issue of using a (global) lattice to discover meaningful viewpoints, i.e., splits in the corresponding context together with the respective (partial) lattices. In a more general way, progress with the split-based product operations is expected to provide the basis for a better mastering of more complex constructs like sub-direct and tensorial products of concept lattices.

Acknowledgments

This work was supported in part by INRIA Post-Doctoral Fellowship and the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant PGPIN-0041899. The authors would like to thank the anonymous referees for their valuable comments and suggestions. Thanks go as well to Robert Godin, Guy Tremblay and Srecko Brlek for the fruitful discussions that helped improve the results of the paper, and to Timothy Walsh for his valuable language assistance.

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] 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.

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

[6] B. Ganter. Two basic algorithms in concept analysis. preprint 831, Technische Hochschule, Darmstadt, 1984.

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

[8] R. Godin, H. Mili, G. W. Mineau, R. Missaoui, A. Arfi, and T. T. Chau. Design of class hierarchies based on concept (galois) lattices. *Theory and Application of Object Systems*, 4(2):117–134, 1998.

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

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

[11] G.-V. Jourdan, J.-X. Rampon, and C. Jard. Computing on-line the lattice of maximal antichains of posets. *Order*, 11(2):197–210, December 1994.

[12] R.E. Kent. Automatic clasification. Intel survey paper, Intel, 1995.

[13] S. Kuznetsov. Some Counting and Decision Problems in Formal Concept Analysis. preprint MATH-AL-14-1999, Technische Universität, Dresden, September 1999.

[14] S. Kuznetsov and S. Objedkov. Algorithms for the Construction of the Set of All Concept and Their Line Diagram. preprint MATH-AL-05-2000, Technische Universität, Dresden, June 2000.

[15] G. W. Mineau and R. Godin. Automatic Structuring of Knowledge Bases by Conceptual Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):824–828, 1995.

[16] 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.

[17] L. Nourine and O. Raynaud. A Fast Algorithm for Building Lattices. *Information Processing Letters*, 71:199–204, 1999.

[18] N. Pasquier, Y. Bastide, T. Taouil, and L. Lakhal. Efficient Mining of Association Rules Using Closed Itemset Lattices. *Information Systems*, 24(1):25–46, 1999.

[19] 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.

[20] P. Valtchev. An algorithm for minimal insertion in a type lattice. *Computational Intelligence*, 15(1):63–78, 1999.

[21] R. Wille. Restructuring the lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered sets*, pages 445–470, Dordrecht-Boston, 1982. Reidel.

[22] R. Wille. Tensoral decomposition of concept lattices. *Order*, 2:81–95, 1985.

[23] R. Wille. Subdirect product construction of concept lattices. *Discrete Mathematics*, 63:305–313, 1987.

[24] R. Wille. Lattices in data analysis: how to draw them with a computer. In I. Rival, editor, *Algorithms and Order*, pages 33–58. Kluwer, Dodrecht-Boston, 1989.

[25] M.J. Zaki and C.-T. Ho. Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):372–390, May/June 2000.