

On Computing the Minimal Generator Family for Concept Lattices and Icebergs

Kamal Nehmé¹, Petko Valtchev¹, Mohamed H. Rouane¹, and Robert Godin²

¹ DIRO, Université de Montréal, Montréal (Qc), Canada

² Département d'informatique, UQAM, Montréal (Qc), Canada

Abstract. Minimal generators (or *mingen*) constitute a remarkable part of the closure space landscape since they are the antipodes of the closures, i.e., minimal sets in the underlying equivalence relation over the powerset of the ground set. As such, they appear in both theoretical and practical problem settings related to closures that stem from fields as diverging as graph theory, database design and data mining. In FCA, though, they have been almost ignored, a fact that has motivated our long-term study of the underlying structures under different perspectives. This paper is a two-fold contribution to the study of *mingen* families associated to a context or, equivalently, a closure space. On the one hand, it sheds light on the evolution of the family upon increases in the context attribute set (e.g., for purposes of interactive data exploration). On the other hand, it proposes a novel method for computing the *mingen* family that, although based on incremental lattice construction, is intended to be run in a batch mode. Theoretical and empirical evidence witnessing the potential of our approach is provided.

1 Introduction

Within the closure operators/systems framework, *minimal generators*, or, as we shall call them for short, *mingen*, are, beside closed and pseudo-closed elements, key elements of the landscape. In some sense they are the antipodes of the closed elements: a *mingen* lays at the bottom of its class in the closure-induced equivalence relation over the ground set, whereas the respective closure is the unique top of the class. This is the reason for *mingen* to appear in almost every context where closures are used, e.g., in fields as diverging as the database design (as *key* sets [7]), graph theory (as *minimal transversals* [2]), data analysis (as *lacunes irréductibles*¹, the name given to them in French in [6]) and data mining (as minimal premises of association rules [8]). In FCA, *mingen* have been used for computational reasons, e.g., in TITANIC [11], where they appear explicitly, as opposed to their implicit use in NextClosure [3] as canonical representations (prefixes) of concept intents.

Despite the important role played by *mingen*, they have been paid little attention so far in the FCA literature. In particular, many computational problems

¹ Irreducible gaps, translation is ours.

related to the mingen family are not well understood, let alone efficiently solved. This observation has motivated an ongoing study focusing on the mingen sets in a formal context that considers them from different standpoints including batch and incremental computation, links to other remarkable members of the closure framework such as pseudo-closed, etc. Recently, we proposed an efficient method for maintaining the mingen family of a context upon increases in the context object set [16]. The extension of the method to lattice merge has been briefly sketched as well. Moreover, the mingen-related part of the lattice maintenance method from [16] was proved to easily fit the iceberg lattice maintenance task as in [10].

In this paper, we study the mingen maintenance problem in dual settings, i.e., upon increases in the attribute set of the context. The study has a two-fold motivation and hence contributes in two different ways to the FCA field. Thus, on the one hand, the evolution of the mingen is given a characterization, in particular, with respect to the sets of stable/vanishing/newly forming mingen. To assess the impact of the provided results, it is noteworthy that although in lattice maintenance the attribute/object cases admit dual resolution, this does not hold for mingen maintenance, hence the necessity to study the attribute case separately. On the other hand, the resulting structure characterizations are embedded into an efficient maintenance method that can, as all other incremental algorithms, be run in a batch mode. The practical performances of the new method as batch iceberg-plus-mingen constructor have been compared to the performances of TITANIC, the algorithm which is reportedly the most efficient one producing the mingen family and the frequent part of the closure family². The results of the comparison proved very encouraging: although our algorithm produces the lattice precedence relation beside concepts and mingen, it outperformed TITANIC when run on a sparse data set. We tend to see this as a clear indication of the potential the incremental paradigm has for mingen computation.

The paper starts with a recall of basic results about lattices, mingen, and incremental lattice update (Section 2). The results of the investigation on the evolution of the mingen family are presented in Section 3 while the proposed maintenance algorithm, INCA-GEN, is described in Section 4. In Section 5, we design a straightforward adaptation of INCA-GEN to iceberg concept lattice maintenance. Section 6 discusses preliminary results of the practical performance study that compared the algorithm to TITANIC.

2 Background on Concept Lattices

In the following, we recall basic results from FCA [18] that will be used in later paragraphs.

² Other algorithms include CLOSE and A-CLOSE [9].

2.1 FCA Basics

Throughout the paper, we use standard FCA notations (see [4]) except for the elements of a formal context for which English-based abbreviations are preferred to German-based ones. Thus, a formal context is a triple $\mathcal{K} = (O, A, I)$ where O and A are sets of objects and attributes, respectively, and I is the binary incidence relation.

We recall that two derivation operators, both denoted by $'$ are defined: for $X \subseteq O$, $X' = \{a \in A \mid \forall o \in X, oIa\}$ and for $Y \subseteq A$, $Y' = \{o \in O \mid \forall a \in Y, oIa\}$. The compound operators $''$ are closure operators over 2^O and 2^A , respectively. Hence each of them induces a family of *closed* subsets, $\mathcal{C}_{\mathcal{K}}^o$ and $\mathcal{C}_{\mathcal{K}}^a$, respectively. A pair (X, Y) of sets, where $X \subseteq O$, $Y \subseteq A$, $X = Y'$ and $Y = X'$, is called a (*formal*) *concept* [18].

Furthermore, the set $\mathcal{C}_{\mathcal{K}}$ of all concepts of the context \mathcal{K} is partially ordered by extent/intent inclusion and the structure $\mathcal{L} = (\mathcal{C}_{\mathcal{K}}, \leq_{\mathcal{K}})$ is a complete lattice. In the remainder, the subscript \mathcal{K} will be avoided whenever confusion is impossible. Fig. 1 shows a sample context where objects correspond to lines and attributes to columns. Its concept lattice is shown next.

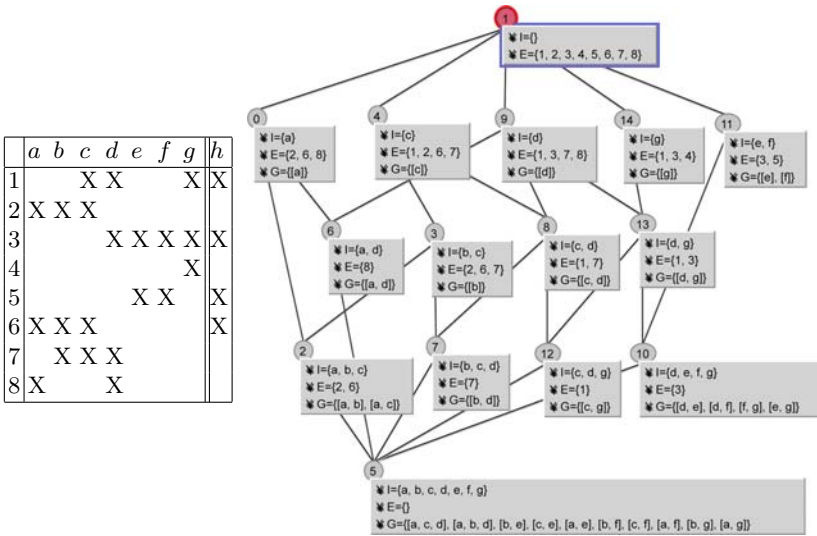


Fig. 1. **Left:** Binary table $\mathcal{K}_1 = (O = \{1, 2, \dots, 8\}, A_1 = \{a, b, \dots, g\}, I_1)$ and the attribute h . **Right:** The Hasse diagram of the lattice \mathcal{L}_1 of \mathcal{K}_1 . Concepts are provided with their respective intent (I), extent (E) and mingen set (G)

Within a context \mathcal{K} , a set $G \subseteq A$ is a *minimal generator* (mingen) of a closed set $Y \subseteq A$ (hence of the concept (Y', Y)) iff G is a minimal subset of Y such that $G'' = Y$. As there may be more than one mingen for a given intent Y , we define the set-valued function *gen*. Formally,

Definition 1. *The function associating to concepts their mingen sets, $gen(c) : \mathcal{C} \rightarrow 2^{2^A}$, is defined as follows:*

$$gen(Y', Y) = \{G \subseteq Y \mid G'' = Y \text{ and } \forall F \subset G, F'' \subset Y\}.$$

In Fig. 1, the concept $c_{\#2} = (26, abc)$ has two mingen: $gen(c_{\#2}) = \{ab, ac\}$. In the remainder, gen will be used both on individual concepts and on concept sets with a straightforward interpretation.

2.2 Incremental Lattice Update, a Recall

Assume that \mathcal{K}_1 and \mathcal{K}_2 are two contexts diverging by only one attribute, i.e., $\mathcal{K}_1 = (O, A_1, I_1)$ and $\mathcal{K}_2 = (O, A_2, I_2)$ with $A_2 = A_1 \cup \{a\}$ and $I_2 = I_1 \cup \{a\} \times a'$. In the following, to avoid confusion, we shall denote the derivation operators in \mathcal{K}_i ($i = 1, 2$), by i . Similarly, mingen functions will be subscripted. Let now \mathcal{L}_1 and \mathcal{L}_2 be the two concept lattices of \mathcal{K}_1 and \mathcal{K}_2 , respectively. If \mathcal{L}_1 is already available, say, as a data structure in the main memory of a computer, then, according to the incremental lattice construction paradigm [5], it can be transformed at a relatively low cost into a structure representing \mathcal{L}_2 . Hence there is no need to construct \mathcal{L}_2 from scratch, i.e., by looking on \mathcal{K}_2 .

In doing the minimal reconstruction that yields \mathcal{L}_2 from \mathcal{L}_1 and (a, a^2) , all the incremental methods rely on basic property of closure systems: \mathcal{C}^o is closed under set intersection [1]. In other words, if $c = (X, Y)$ is a concept of \mathcal{L}_1 then $X \cap a^2$ is closed object set and corresponds to an extent in \mathcal{L}_2 . Hence, the transformation of \mathcal{L}_1 into \mathcal{L}_2 via the attribute a is mainly aimed at computing all the concepts from \mathcal{L}_2 whose extent is not an extent in \mathcal{L}_1 . Those concepts are called the *new* concepts in [5] (here denoted $\mathbf{N}(a)$). As to \mathcal{L}_1 , its concepts are partitioned into three categories. The first one is made of *modified* concepts (labeled $\mathbf{M}(a)$): their extent is included in a^2 , the extent of a , hence they evolve from \mathcal{L}_1 into \mathcal{L}_2 by integrating a into their intents while extents remain stable. The second category is made of *genitor* concepts (denoted $\mathbf{G}(a)$) which help create new concepts but remain themselves stable (changes appear in the sets of neighbor concepts). Finally, *old* concepts (denoted $\mathbf{U}(a)$) remain completely unchanged. As concepts from \mathcal{L}_1 have their counterparts in \mathcal{L}_2 , we shall use subscripts to distinguish both copies of a set. Thus, \mathbf{G}_1 , \mathbf{U}_1 and \mathbf{M}_1 will refer to \mathcal{L}_1 , while \mathbf{G}_2 , \mathbf{U}_2 , \mathbf{M}_2 and \mathbf{N}_2 will refer to \mathcal{L}_2 .

A characterization of each of the above seven concept categories is provided in [17]. It relies on two functions which map concepts to the intersection of their extents with a^2 : $\mathcal{R}_i : \mathcal{C}_i \rightarrow 2^O$ with $\mathcal{R}_i(c) = extent(c) \cap a^2$. Each \mathcal{R}_i induces an equivalence relation on \mathcal{C}_i where $[c]_{\mathcal{R}_i} = \{\bar{c} \in \mathcal{C}_i \mid \mathcal{R}_i(c) = \mathcal{R}_i(\bar{c})\}$. Moreover, following [15], $\mathbf{G}_1(a)$ and $\mathbf{M}_1(a)$ are the minimal concepts in their respective equivalence classes in \mathcal{L}_1 .

Example 1. Assume \mathcal{L}_1 is the lattice induced by the attribute set $abcdefg$ (see Fig. 1 on the right) and consider h the new attribute to add to \mathcal{K}_1 . Fig. 2 shows the resulting lattice \mathcal{L}_2 . The sets of concepts are as follows: $\mathbf{U}_2(h) =$

$\{c_{\#0}, c_{\#3}, c_{\#6}, c_{\#7}, c_{\#8}, c_{\#9}, c_{\#14}\}$; $\mathbf{M}_2(h) = \{c_{\#5}, c_{\#10}, c_{\#11}, c_{\#12}, c_{\#13}\}$; $\mathbf{G}_2(h) = \{c_{\#1}, c_{\#2}, c_{\#4}\}$ and $\mathbf{N}_2(h) = \{c_{\#15}, c_{\#16}, c_{\#17}\}$.

Following [17], three mappings will be used to connect \mathcal{L}_1 into \mathcal{L}_2 . First, σ maps a concept in \mathcal{L}_1 to the concept with the same extent in \mathcal{L}_2 . Second, γ projects a concept from \mathcal{L}_2 on A_1 , i.e., returns the concept having the same attributes but a . Finally, χ^+ returns for a concept c in \mathcal{L}_1 the minimal element of the equivalence class $\llbracket \mathcal{R}_2$ for its counterpart $\sigma(c)$ in \mathcal{L}_2 .

Definition 2. We define the following mappings between \mathcal{L}_1 and \mathcal{L}_2 :

- $\sigma: \mathcal{C}_1 \rightarrow \mathcal{C}_2, \sigma(X, Y) = (X, X^2),$
- $\gamma: \mathcal{C}_2 \rightarrow \mathcal{C}_1, \gamma(X, Y) = (\bar{Y}^1, \bar{Y})$ where $\bar{Y} = Y - \{a\},$
- $\chi^+: \mathcal{C}_1 \rightarrow \mathcal{C}_2, \chi^+(X, Y) = (\bar{X}, \bar{X}^2)$ where $\bar{X} = X \cap a^2.$

3 Structure Characterization

We clarify here the evolution of the mingen family between \mathcal{L}_1 and \mathcal{L}_2 . First, we prove two properties stating, respectively, that no generator vanishes from \mathcal{L}_1 to \mathcal{L}_2 and that only modified and new concepts in \mathcal{L}_2 contribute to the difference $gen_2(\mathcal{C}_2) - gen_1(\mathcal{C}_1)$. Then, we focus on the set of new mingen that forms in each of the two cases and prove that a new mingen is made of an old one augmented by the attribute a .

3.1 Global Properties

Let us first find the relation between the mingen of a concept c in \mathcal{L}_1 and those of its counterpart $\sigma(c)$ in \mathcal{L}_2 . The following property shows that the mingen of c are also mingen for $\sigma(c)$.

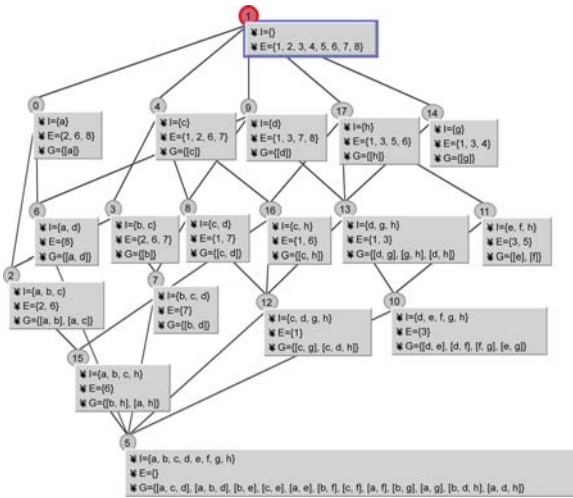


Fig. 2. The Hasse diagram of the new lattice \mathcal{L}_2 derived from context \mathcal{K}_2

Property 1. $\forall c \in \mathcal{L}_1 \text{ gen}_1(c) \subseteq \text{gen}_2(\sigma(c))$.

Proof. If $G \in \text{gen}_1(c)$ then $G^1 = G^2$, hence, $G^{22} = \bar{Y}$. Moreover G is minimal for \bar{Y} , otherwise it could not be a mingen of c .

Consequently, if G is a mingen in \mathcal{L}_1 it can only be a mingen in \mathcal{L}_2 .

Corrolary 1. $\text{gen}(\mathcal{C}_1) \subseteq \text{gen}(\mathcal{C}_2)$

Proof. Given a concept $c = (X, Y)$ in \mathcal{L}_1 , whatever is the category of $\sigma(c)$ in \mathcal{L}_2 (*old, genitor or modified*), we always have $\text{gen}_1(c) \subseteq \text{gen}_2(\sigma(c))$.

For example, consider the concept $c_{\#12} = (1, cdg)$ in \mathcal{L}_1 whose mingen cg is also a mingen of $\sigma(c)_{\#12} = (1, cdgh)$ in \mathcal{L}_2 . However, the concept $c_{\#12}$ in \mathcal{L}_2 has another mingen, cdh , which it does not share with its image in \mathcal{L}_1 . This case can only happen with modified concepts from \mathcal{L}_1 because, as the following property states it, for old and genitor concepts in \mathcal{L}_1 , the mingen of their $\sigma(c)$ -counterpart in \mathcal{L}_2 are exactly the same as their own mingen.

Property 2. $\forall c = (X, Y) \in \mathcal{C}_1$, if $\sigma(c) = c$ then $\text{gen}_1(c) = \text{gen}_2(\sigma(c))$

Proof. Following Property 1, $\text{gen}_1(c) \subseteq \text{gen}_2(\sigma(c))$. Then, $\text{gen}_2(\sigma(c)) \subseteq \text{gen}_1(c)$ comes from the fact that if $G \in \text{gen}_2(\sigma(c))$ and $c = \sigma(c)$ then $G^1 = G^2$ and $G^{11} = G^{22} = Y$. Moreover, G is minimal for c otherwise it would not be a mingen of $\sigma(c)$.

For example, consider the concept $c_{\#2} = (26, abc)$ in \mathcal{L}_1 . It is easily seen that the set of its mingen, $\{ab, ac\}$, is the same as the mingen set of $\sigma(c_{\#2})$ in \mathcal{L}_2 .

3.2 Characterizing the New Mingen

Now that we know that all mingen in \mathcal{L}_1 stay mingen in \mathcal{L}_2 , the next step consists in finding the new mingen in \mathcal{L}_2 . As the modified concepts and the genitor ones are the minimal elements of their equivalence classes in \mathcal{L}_1 , we express the evolution of these classes from \mathcal{L}_1 to \mathcal{L}_2 . In fact, the class of a new concept c in \mathcal{L}_2 is exactly the image of the class of its genitor in \mathcal{L}_1 to which we add c . The class of a modified concept is identical in both \mathcal{L}_1 and \mathcal{L}_2 .

Property 3. *The equivalence classes of new and modified concepts in \mathcal{L}_2 are composed as follows:*

- $\forall c \in \mathbf{N}_2(a), [c]_{\mathcal{R}_2} = [\gamma(c)]_{\mathcal{R}_1} \cup c$
- $\forall c \in \mathbf{M}_2(a), [c]_{\mathcal{R}_2} = [\gamma(c)]_{\mathcal{R}_1}$

In summary, it was established that the equivalence classes $\square_{\mathcal{R}_2}$ differ by at most one element from their counterparts $\square_{\mathcal{R}_1}$ and that new mingen may only appear at the minimal element of each class. The next question to ask is how mingen of concepts in $[\gamma(c)]_{\mathcal{R}_1}$ are related to those of the minimal element of $[c]_{\mathcal{R}_2}$. The first step is to notice that whenever a mingen of a concept c from \mathcal{L}_1

is augmented with the new attribute a , the closure of the resulting set in \mathcal{K}_2 is the intent of the minimal element in the respective class $[\sigma(c)]_{\mathcal{R}_2}$.

Assume $c_{\min} = (X, Y) \in \mathcal{C}_2$ is the minimal element of its class in \mathcal{L}_2 and let $\bar{c} = (\bar{X}, \bar{Y})$ a concept of that class while \bar{G} is a mingen of \bar{c} . According to the definition of $[c]_{\mathcal{R}_i}$, we have: $\bar{X} \cap a^2 = X$ and hence $\bar{G}^2 \cap a^2 = Y^2$. Moreover, it is known that for $A, B \in 2^O$, $(A \cup B)^2 = A^2 \cap B^2$. Thus, $\bar{G}^2 \cap a^2 = Y^2$ can be written as $(\bar{G} \cup a)^2 = Y^2$ and consequently $(\bar{G} \cup a)^{22} = Y^{22}$.

In summary, for every mingen G of a concept in a class $[c]_{\mathcal{R}_1}$, its superset obtained by adding the new attribute a , $\bar{G} \cup a$, has as closure the intent of the minimal element of the corresponding class $[\sigma(c)]_{\mathcal{R}_2}$. The following property states that $(\bar{G} \cup a)$ is a mingen of c iff \bar{G} is minimal among the mingen of the entire equivalence class $[c]_{\mathcal{R}_1}$.

Property 4. $\forall c \in \mathbf{M}_2(a)$ then :

$$gen_2(c) = gen_1(\gamma(c)) \cup \min\left(\bigcup_{\hat{c} \in [c]_{\mathcal{R}_i} \text{ and } \hat{c} \neq c} gen_1(\hat{c})\right) \times \{a\}$$

For example, consider the concept $c_{\#13} = (13, dgh)$ in \mathcal{L}_2 . Here $\mathcal{R}_2(c_{\#13}) = 13$ and $[c_{\#13}]_{\mathcal{L}_2} = \{c_{\#9}, c_{\#14}, c_{\#13}\}$. Clearly, $c_{\#13}$ is minimal in its class, and more precisely, it is a modified concept. Since $gen_1(c_{\#9}) = \{d\}$ and $gen_1(c_{\#9}) = \{g\}$ whereas both mingen d and g are incomparable, the newly formed mingen of $c_{\#13}$ in \mathcal{L}_2 are $\{dh, gh\}$. Consequently, the entire set of mingen for $c_{\#13}$ in \mathcal{L}_2 is $gen_2(c_{\#13}) = \{dg, dh, gh\}$. Indeed, the correctness of that result can be checked upon Fig. 2.

A similar result can be proved for the complementary case for the minima in a class $[\]_{\mathcal{R}_2}$, i.e., for new concepts. The following property states that the mingen of a new concept $c = (X, Y) \in \mathbf{N}_2(a)$ are exactly the sets produced by adding a to each of the mingen that are minimal in the entire class $[\gamma(c)]_{\mathcal{R}_1}$.

Property 5.

$$\forall c \in \mathbf{N}_2(a), gen_2(c) = \min\left(\bigcup_{\hat{c} \in [\gamma(c)]_{\mathcal{R}_1}} gen_1(\hat{c})\right) \times \{a\}$$

For example, consider the concept $c_{\#15} = (6, abch)$ in \mathcal{L}_2 . $\mathcal{R}_2(c_{\#15}) = 6$ and $[c_{\#15}]_{\mathcal{R}_2} = \{c_{\#0}, c_{\#3}, c_{\#2}, c_{\#15}\}$. Clearly, $[\gamma(c_{\#15})]_{\mathcal{R}_1} = \{c_{\#0}, c_{\#3}, c_{\#2}\}$. Moreover, $gen_1(c_{\#0}) = \{a\}$, $gen_1(c_{\#3}) = \{b\}$, $gen_1(c_{\#2}) = \{ab, ac\}$. Thus, $\min(\bigcup_{\hat{c} \in [c_{\#15}]_{\mathcal{R}_1}} gen_1(\hat{c})) = \min\{a, b, ab, ac\} = \{a, b\}$ and hence $gen(c)_{\#15} = \{ah, bh\}$.

Following Properties 4 and 5, we state that every new mingen in \mathcal{L}_2 is obtained by adding a to a mingen from \mathcal{L}_1 .

Corrolary 2. $gen(\mathcal{C}_2) - gen(\mathcal{C}_1) \subseteq gen(\mathcal{C}_1) \times \{a\}$.

In summary, to compute the mingen in \mathcal{L}_2 , one only needs to focus on new and modified elements. In both cases, the essential part of the calculation is the

detection of all the mingen G of concepts from the underlying equivalence classes which are themselves minimal, i.e., there exists no other mingen in the class that is strictly included in G . Obviously, this requires the equivalence classes to be explicitly constructed during the maintenance step.

4 Mingen Maintenance Method

The results presented in the previous section are transformed into an algorithmic procedure, provided in Algorithm 1, that updates both the lattice and the mingen sets of the lattice concepts upon the insertion of a new attribute into the context. The key task of the method hence consists in computing/updating the mingen of the minimal concept in each class $\llbracket \mathcal{R}_2 \rrbracket$ in \mathcal{L}_2 . Such a class will be explicitly represented by a variable, θ , which is a structure with two fields: the minimal concept, *min-concept*, and minimal mingen, *min-gen*. Moreover, the variable for all classes are gathered in a index, called *Classes*, where each variable is indexed by the \mathcal{R}_1 value for the respective class.

```

1: procedure INCA-GEN(In/Out:  $\mathcal{L} = \langle \mathcal{C}, \leq \rangle$  a Lattice, In:  $a$  an attribute)
2: Local : Classes : an indexed structure of classes
3:
4: COMPUTE-CLASSES( $\mathcal{C}, a$ )
5: for all  $\theta$  in Classes do
6:    $c \leftarrow \theta.min\text{-concept}$ 
7:   if  $|\mathcal{R}(c)| = |extent(c)|$  then
8:      $intent(c) \leftarrow intent(c) \cup \{a\}$     { $c$  is modified}
9:   else
10:     $\hat{c} \leftarrow newConcept(\mathcal{R}(c), intent(c) \cup \{a\})$     { $c$  is genitor}
11:     $\mathcal{L} \leftarrow \mathcal{L} \cup \{\hat{c}\}$ 
12:     $updateOrder(c, \hat{c})$ 
13:     $gen(\hat{c}) \leftarrow \emptyset$ 
14:     $\theta.min\text{-concept} \leftarrow \hat{c}$ 
15:     $c \leftarrow \theta.min\text{-concept}$ 
16:     $gen(c) \leftarrow gen(c) \cup \theta.min\text{-gen} \times \{a\}$ 

```

Algorithm 1: Insertion of a new attribute in the context

It is noteworthy that the lattice update part of the work is done in a way which is dual to the object-wise incremental update. Thus, the Algorithm 1 follows the equivalent of the basic steps for object incrementing described in [14]. The work starts with a pre-processing step that extracts the class information from the lattice and stores it in the *Classes* structure (primitive COMPUTE-CLASSES, see Algorithm 2). At a second step, the variables θ corresponding to each class are explored to restructure the lattice and to compute the mingen (lines 6 to 16). First, the kind of restructuring, i.e., modification of an intent versus creation of a new concept, is determined (line 7). Then, the standard

update procedures are carried out for modified (line 8) and genitor concepts (lines 10 to 12). In the second case, the computation specific to the mingen family update is limited to lines 13 and 14. Finally, the mingen set of a minimal concept is updated in a uniform manner that strictly follows the Properties 4 and 5.

The preprocessing step as described in Algorithm 2 basically represents a traversal of the lattice during which the content of the *Classes* structure is gradually collected.

At each concept, the intersection of the extent and the object set of the attribute a is computed (line 5). This provides an entry point for the concept to its equivalence class which is tentatively retrieved from the *Classes* structure using the intersection as a key (line 6). If the class is not yet present in the structure (line 7), which means that the current concept is its first encountered member, the corresponding variable θ is created (line 8), initialized with the information found in c (line 9), and then inserted in *Classes* (line 10). The current concept is also compared to the current minimum of the class (lines 11 and 12) and the current minima of the total mingen set of the class are updated (line 13). At the end, the structure *Classes* comprises the variables of all the equivalence classes in \mathcal{R}_1 with the accurate information about its minimal representative and about the minima of the global mingen set.

```

1: procedure COMPUTE-CLASSES(In/Out:  $\mathcal{C}$  concept set, In:  $a$  an attribute)
2:
3: for all  $c$  in  $\mathcal{C}$  do
4:    $E \leftarrow extent(c) \cap a^2$ 
5:    $\theta \leftarrow lookup(Classes, E)$ 
6:   if ( $\theta = \text{NULL}$ ) then
7:      $\theta \leftarrow newClass()$ 
8:      $\theta.min\text{-concept} \leftarrow c$ 
9:      $put(Classes, \theta, E)$ 
10:  if ( $\theta.min\text{-concept} < c$ ) then
11:     $\theta.min\text{-concept} \leftarrow c$ 
12:   $\theta.min\text{-gen} \leftarrow Min(\theta.min\text{-gen} \cup gen(c))$ 

```

Algorithm 2: Computation of the equivalence classes $\llbracket \mathcal{R} \rrbracket$ in the initial lattice

5 Iceberg Lattice Variant

Let $c = (X, Y)$ a concept in \mathcal{L}_1 . The frequency of c , denoted $freq(c)$, is defined as the ratio of its extent and the size of the object set: $freq(c) = \|X\|/\|O\|$. Given α a minimal threshold of support defined by the user, the concept c is frequent if $freq(c) \geq \alpha$. The *iceberg concept lattice* generated by α , \mathcal{L}_1^α , is made of all frequent concepts. An iceberg is thus a join-semi-lattice, a sub-semi-lattice of the complete concept lattice [11].

For example, the *iceberg* $\mathcal{L}_1^{0.20}$ obtained from the complete lattice of Fig. 1 with $\alpha \geq 0.20$ is shown in Fig. 3.

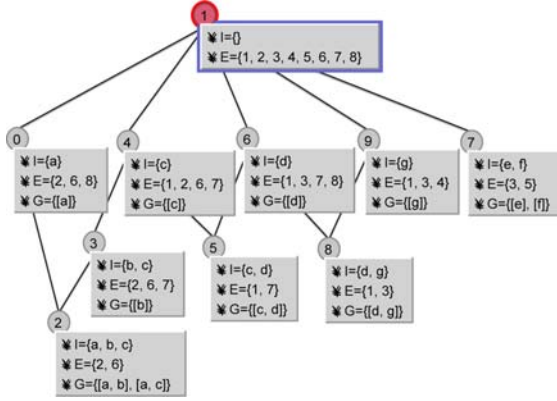


Fig. 3. The iceberg lattice $\mathcal{L}_1^{0.20}$ of the context \mathcal{K}_1 with $A_1 = \{a, \dots, g\}$

Unlike the object-wise incremental iceberg update [10], when a new attribute is added, the maintenance of the iceberg follows strictly that of the complete lattice. This is due to the invariance of concept frequency: Once a concept is generated, its frequent status remains unchanged. Assume c is a concept in \mathcal{L}_1^α . The $\text{freq}(\sigma(c))$ in \mathcal{L}_2^α will be $\frac{|\text{extent}(\sigma(c))|}{|O|}$. As $\text{extent}(\sigma(c)) = \text{extent}(c)$ and the number of objects does not vary along the transition from \mathcal{L}_1^α to \mathcal{L}_2^α it follows that $\text{freq}(c) = \text{freq}(\sigma(c))$.

Moreover, the frequency of the intersection $\mathcal{R}_1(c)$ is monotonously non-decreasing with respect to lattice order since it is the composition of two monotonous non-decreasing functions.

Property 6. $\forall c \in \mathcal{L}_1^\alpha$, if $|\mathcal{R}_1(c)| \leq \alpha|O|$ then $\forall \underline{c}$ s.t. $c \prec \underline{c}$, $|\mathcal{R}_1(\underline{c})| \leq \alpha|O|$.

Exploring the monotonicity of the frequency function and restricting the \mathcal{R}_i functions ($i = 1, 2$) from section 2.2 to icebergs, we obtain the fact that a class may be only partially included in the iceberg. Furthermore, as for any concept $c = (X, Y)$ in \mathcal{L}_1^α , $\text{extent}(\chi^+(c))$ is the extent of a new or a modified concept in \mathcal{L}_2^α , only the frequent intersections could be considered since the concepts corresponding to infrequent ones do not belong to the iceberg.

The above observation could potentially invalidate our mingen calculation mechanism since it relies on the presence of the entire mingen set for a class in \mathcal{R}_1 . However, observe that only the minima of the equivalence classes require some calculation. Moreover, because of the monotonicity, the minima are also the less frequent concepts of a class and thus cannot be present in the iceberg \mathcal{L}_2^α if the class is only partially covered by \mathcal{L}_2^α . Consequently, infrequent intersections could simply be ignored.

In summary, the icebergs could be dealt with in a way similar to that for complete lattices. The only thing to be added is a filter for infrequent intersections. Thus, a concept producing such an intersection should be discarded from the preprocessing step. As a result, only the classes corresponding to frequent

intersections, or, equivalently, having frequent minima, will be sent to the main algorithm for further processing.

```

1: procedure COMPUTE-F-CLASSES(In/Out:  $\mathcal{L}_\alpha$  an iceberg lattice, In:  $a$  an attribute)
2: Local :  $cQ$  : a queue of concepts
3:
4:  $in(cQ, top(\mathcal{L}_\alpha))$ 
5: while  $nonempty(cQ)$  do
6:    $c \leftarrow out(cQ)$ 
7:    $E \leftarrow extent(c) \cap a^2$ 
8:   if  $(|E| \geq \alpha|O|)$  then
9:      $\theta \leftarrow lookup(Classes, E)$ 
10:    if  $(\theta = NULL)$  then
11:       $\theta \leftarrow newClass()$ 
12:       $\theta.min-concept \leftarrow c$ 
13:       $put(Classes, \theta, E)$ 
14:    if  $(\theta.min-concept > c)$  then
15:       $\theta.min-concept \leftarrow c$ 
16:     $\theta.min-gen \leftarrow Min(\theta.min-gen \cup gen(c))$ 
17:    for all  $\hat{c} \in Cov^u(c)$  do
18:       $in(cQ, \hat{c})$ 

```

Algorithm 3: Computation of the frequent equivalence classes $\square_{\mathcal{R}}$ in the initial iceberg lattice

Luckily enough, the difference between the processing of icebergs and that of complete lattices can be confined in the class construction step. Thus, the frequency-aware traversal of the iceberg relies on the monotonicity of the intersection function to prune unnecessary lattice paths. More specifically, it explores the concept set following the lattice order and in a top-down manner whereby the

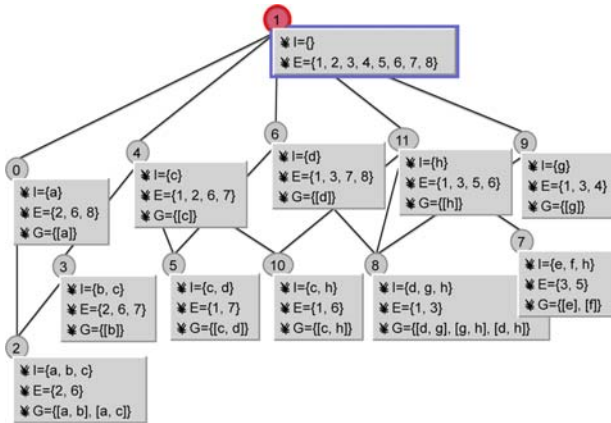


Fig. 4. The iceberg lattice $\mathcal{L}_2^{0.20}$ of the context \mathcal{K}_2 with $A_2 = A_1 \cup \{h\}$

exploration of a new path stops with the first concept producing an infrequent intersection. These differences are reflected in the code of Algorithm 3. The algorithm uses a queue structure to guide the top-down, breadth-first traversal of the iceberg (lines 4 to 6 and 17 to 18). The remaining noteworthy difference with Algorithm 2 is that infrequent concepts are ignored whenever pulled out of the queue (line 8).

Finally, to obtain an iceberg-based version of the lattice algorithm INCA-GEN, it would suffice to replace in Algorithm 1 the call of COMPUTE-CLASSES by COMPUTE-F-CLASSES on line 4. For example, the iceberg lattice $\mathcal{L}_2^{0,20}$ resulting from the addition of the attribute h to the iceberg lattice $\mathcal{L}_1^{0,20}$ in Fig. 3 is depicted in Fig. 4.

6 Experiments and Performance Evaluation

The algorithm INCA-GEN has been implemented in Java and a version thereof is available within the Galicia³ platform [13]. The platform version is designed for portability and genericity and therefore is not optimized for performances.

A stand-alone version of the algorithm, called MAGALICE-A, was devised for use in experimental studies. Its performances have been examined on a comparative basis. In a preliminary series of tests, MAGALICE-A was confronted to the TITANIC algorithm [11]. TITANIC is a batch method which solves a similar problem, known as *frequent closed itemset mining*, and produces comparable results, i.e., the set of frequent concept intents and the corresponding mingen sets. The choice of TITANIC was further motivated by the reported efficiency of the method and its status of reference algorithm in the FCA community. We used our own Java implementation of TITANIC for the experiments since at the time of the study, no code was publicly available⁴.

The experiments were carried out on a Pentium IV 2 GHz workstation running Windows XP, with 1 GB of RAM. The comparisons were performed on two types of datasets:

- subsets of MUSHROOM, a real-world dataset which is also a dense one (8, 124 objects, 119 attributes, average of 23 attributes per object), and
- subsets of T25.I10.D10K, a sparse synthetic dataset popular with the data mining community (10,000 objects, 1000 attributes, average of 25 attributes per object).

The choice of both datasets was motivated by the following observation. It is now widely admitted that incremental lattice algorithms perform well on sparse datasets but lag behind batch methods when applied to dense ones. Our goal was to test whether this trend persists when the mingen families are fed into the computation process. Indeed, the experimental results seem to confirm the

³ <http://galicia.sourceforge.net>

⁴ The authors would like to thank G. Stumme for the valuable information he provided about TITANIC.

hypothesis that incremental methods may be used as efficient batch procedure whenever dealing with low-density data tables.

More concretely, two types of statistics have been gathered. On the one hand, the efficiency of both algorithms has been directly related to the CPU time that was required to solve the task. On the other hand, we have recorded the memory consumption as an important secondary indicator of how suitable the algorithm is for large dataset analysis.

Fig. 5 depicts the CPU time of the analysis of both datasets: the dense one, on the left, and the sparse one, on the right. The first diagram indicates that TITANIC outperforms MAGALICE-A by far: it runs 2 to 7 times faster. However, the reader should bear in mind the fact that beside the concept set and the mingen, MAGALICE-A also maintains the order in the iceberg lattice while TITANIC does not.

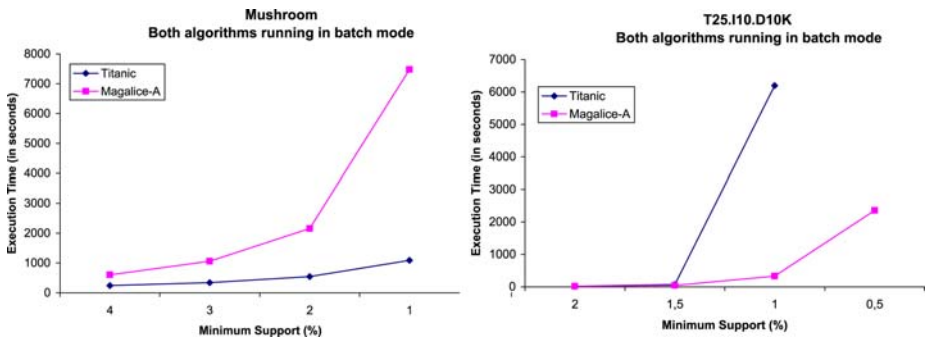


Fig. 5. CPU-time for MAGALICE-A and TITANIC. The tests involved the entire dataset from which only the frequent concepts had to be computed using a range of support threshold values

The second diagram reverses the situation: MAGALICE-A beats TITANIC by a factor going up to 18. A more careful analysis would be necessary to explain such a dramatic shift in performances. However, the main reason seems to lay in the fact that the performance of the incremental algorithm is strongly impacted by the actual number of concepts in the iceberg. Thus, with a higher number of concepts as with the MUSHROOM dataset, the algorithm suffers a significant slow-down whereas with T25.I10.D10K where the number of concepts is low, it performs well. The closure computation used by TITANIC to find concept intents depends to a much lesser degree on the number of concepts in the iceberg and therefore the performances of the algorithm vary in a narrow interval.

Fig. 6 visualizes the results about the memory usage for both algorithms. The same trends as with CPU-time seem to appear here. Thus, while dense datasets increase substantially the memory consumption of our algorithm, TITANIC remains at a reasonable usage rate. With sparse datasets however, the figures are mirrored: MAGALICE-A leads by far with a much smaller memory demand. Once again, the number of frequent concepts might be the key to the interpretation

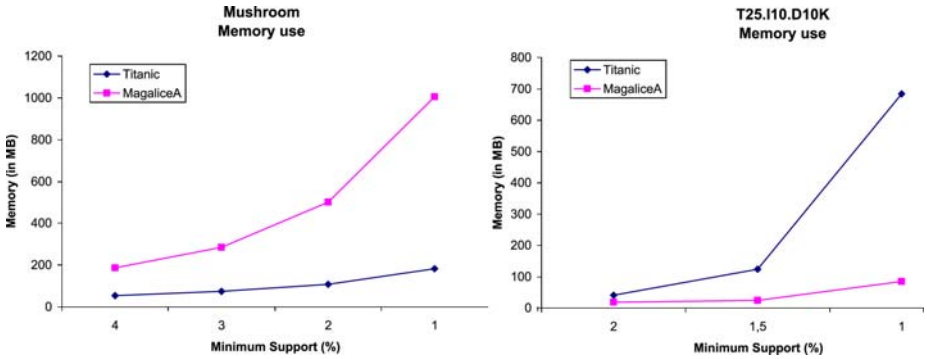


Fig. 6. Memory consumption for MAGALICE-A and TITANIC. The tests involved the entire dataset from which only the frequent concepts had to be computed using a range of support threshold values

of the results: with larger number of concepts, the overhead due to additional computations in MAGALICE-A, i.e., equivalence class constitution, extent and order computation/storage, etc., takes over the core tasks of computing intents and mingen. Conversely, with a small number of frequent intents, the number of mingen is (proportionally) larger. MAGALICE-A does well since only small amount of computing is performed on a mingen, whereas TITANIC wastes time in computing a large number of closures.

It is noteworthy that for support values of 1% and less, both algorithms exhaust the main memory capacity and relied on swapping to continue their work. For TITANIC, this happens when working on the sparse dataset while for MAGALICE-A it is the case with the dense one.

7 Conclusion

Minimal generators of concept intents are intriguing members of the FCA landscape with strong links to practical and theoretical problems from neighbor areas. Because of their important role, it is worth studying their behavior under different circumstances, in particular their evolution upon small changes in the input context. In this paper we studied the evolution of the mingen family of a context upon increases of the attribute set. The operation has a certain practical value since in many FCA tools, dynamic changes in the set of “visible” attributes are admitted. However, in this study we looked at the incrementing of the attribute set as a pure computational technique and examined its relative merits compared to those of an existing batch method.

The results up to date suggest that the incremental paradigm has its place in this particular branch of FCA algorithmic practices. They also motivate the research on a second generation algorithms that would improve on the design of the initial, rather straightforward procedures, INCA-GEN and MAGALICE-A.

The presented research is a first stage in a broader study on the dynamic behavior of FCA-related subset families: closures, pseudo-closed, mingen, etc. An even more intriguing subject is the cross-fertilization between methods for computing separate families in the way it is done in TITANIC or in the MERGE algorithm described in [12].

References

1. M. Barbut and B. Monjardet. *Ordre et Classification: Algèbre et combinatoire*. Hachette, 1970.
2. C. Berge. *Hypergraphs: Combinatorics of Finite Sets*. North Holland, Amsterdam, 1989.
3. B. Ganter. Two basic algorithms in concept analysis. preprint 831, Technische Hochschule, Darmstadt, 1984.
4. B. Ganter and R. Wille. *Formal Concept Analysis, Mathematical Foundations*. Springer-Verlag, 1999.
5. R. Godin, R. Missaoui, and H. Alaoui. Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices. *Computational Intelligence*, 11(2):246–267, 1995.
6. J.L. Guigues and V. Duquenne. Familles minimales d’implications informatives résultant d’un tableau de données binaires. *Mathématiques et Sciences Humaines*, 95:5–18, 1986.
7. D. Maier. *The theory of Relational Databases*. Computer Science Press, 1983.
8. N. Pasquier. Extraction de bases pour les règles d’association à partir des itemsets fermés fréquents. In *Proceedings of the 18th INFORSID’2000*, pages 56–77, Lyon, France, 2000.
9. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings, ICDT-99*, pages 398–416, Jerusalem, Israel, 1999.
10. M. H. Rouane, K. Nehme, P. Valtchev, and R. Godin. On-line maintenance of iceberg concept lattices. In *Contributions to the 12th ICCS*, page 14 p., Huntsville (AL), 2004. Shaker Verlag.
11. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing Iceberg Concept Lattices with Titanic. *Data and Knowledge Engineering*, 42(2):189–222, 2002.
12. P. Valtchev and V. Duquenne. Implication-based methods for the merge of factor concept lattices (32 p.). *submitted to Discrete Applied Mathematics*.
13. P. Valtchev, D. Grosser, C. Roume, and M. Rouane Hacene. GALICIA: an open platform for lattices. In B. Ganter and A. de Moor, editors, *Using Conceptual Structures: Contributions to 11th Intl. Conference on Conceptual Structures (ICCS’03)*, pages 241–254, Aachen (DE), 2003. Shaker Verlag.
14. P. Valtchev, M. Rouane Hacene, and R. Missaoui. A generic scheme for the design of efficient on-line algorithms for lattices. In A. de Moor, W. Lex, and B. Ganter, editors, *Proceedings of the 11th Intl. Conference on Conceptual Structures (ICCS’03)*, volume 2746 of *Lecture Notes in Computer Science*, pages 282–295, Berlin (DE), 2003. Springer-Verlag.
15. P. Valtchev and R. Missaoui. Building concept (Galois) lattices from parts: generalizing the incremental methods. In H. Delugach and G. Stumme, editors, *Proceedings of the ICCS’01*, volume 2120 of *Lecture Notes in Computer Science*, pages 290–303, 2001.

16. P. Valtchev, R. Missaoui, and R. Godin. Formal Concept Analysis for Knowledge Discovery and Data Mining: The New Challenges. In P. Eklund, editor, *Concept Lattices: Proceedings of the 2nd Int. Conf. on Formal Concept Analysis (FCA'04)*, volume 2961 of *Lecture Notes in Computer Science*, pages 352–371. Springer-Verlag, 2004.
17. P. Valtchev, R. Missaoui, R. Godin, and M. Meridji. Generating Frequent Itemsets Incrementally: Two Novel Approaches Based On Galois Lattice Theory. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3):115–142, 2002.
18. R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered sets*, pages 445–470, Dordrecht-Boston, 1982. Reidel.