

On-line maintenance of iceberg concept lattices

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

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

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

Abstract. The iceberg lattice of a context is a substructure of the complete concept lattice useful for data mining. Its construction has been tackled with batch algorithmic approaches but incremental algorithms are yet to be designed. The paper introduces an approach to iceberg maintenance that is based on a framework for complete lattice on-line maintenance which is here extended with new structural results about iceberg evolution. An algorithm, called MAGALICE, is proposed that performs maintenance in two steps: the first one follows the classical restructuring schema while the second one fills the gaps left in order to obtain a valid structure.

Key Words: Concept lattices, icebergs, lattice constructing algorithms.

1 Introduction

Formal concept analysis (FCA) has been successfully applied to practical problems from a variety of fields, e.g., software engineering, information retrieval, and knowledge discovery from data (KDD). In KDD, FCA was used to formalize the association rule mining (ARM) [PBTL99,ZH99]. ARM consists in detecting frequent patterns, or itemsets, within a database of commercial transactions and further extracting associations between complementary parts of a pattern [AS94]. Itemsets and rules are comparable to attribute sets and (partial) implications from FCA, respectively, whereby the iceberg lattice is the upper part of the concept lattice of the transaction context, i.e., the one reduced to concepts of large enough extents. Batch methods have been proposed for the computation of iceberg lattices (e.g., TITANIC [STB⁺02]). These methods are clearly unfit for dynamic database environments which would rather require an on-line approach to reduce the computational effort due to data evolution (e.g., building up on the incremental lattice construction as in [GMA95]).

We propose such an approach and design a concrete algorithm for iceberg lattice maintenance upon transaction insertion, called MAGALICE. The algorithm follows the generic incremental scheme for complete lattices as in [VHM03] which is refined by some iceberg-related tasks. Its complexity is examined in the paper from both theoretical and practical standpoint.

The paper starts with a summary of relevant results from FCA theory and algorithms (Sec. 2). The results underlying our iceberg maintenance approach

are presented in Sec. 3 while the proposed maintenance algorithm is described in Sec. 4. Sec. 5 discusses preliminary results on the practical performance of the proposed algorithm.

2 Background on concept lattices

2.1 Formal concept analysis basics

Formal Concept Analysis (FCA) focuses on the partially ordered structure, known under the names of *Galois lattice* [BM70] or *concept lattice* [Wil82], which is induced by a binary relation R over a pair of sets O (*objects*) and A (*attributes*). FCA helps identifying groupings of individuals sharing a set of features. A formal context is a triple $\mathcal{K} = (O, A, I)$ where I is a binary (incidence) relation, i.e., $I \subseteq O \times A$ (oIa means that object o has the attribute a). Within a context (see Fig. 1 on the left), objects are denoted by numbers and attribute by small letters. Set notations are separator-free, e.g., 13 stands for $\{1, 3\}$ and bcd for $\{b, c, d\}$. Two functions, f and g , induce a Galois connection [BM70] between $\mathcal{P}(O)$ and $\mathcal{P}(A)$.

$$\begin{aligned} - f : \mathcal{P}(O) &\rightarrow \mathcal{P}(A), f(X) = X' = \{a \in A \mid \forall o \in X, oIa\} \\ - g : \mathcal{P}(A) &\rightarrow \mathcal{P}(O), g(Y) = Y' = \{o \in O \mid \forall a \in Y, oIa\} \end{aligned}$$

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

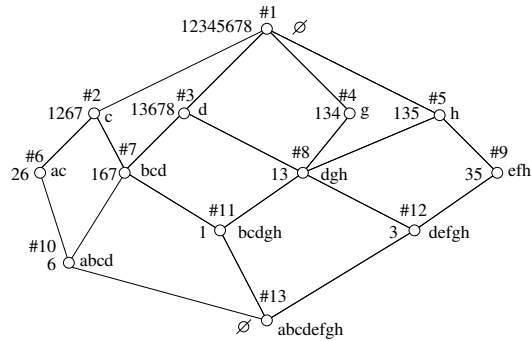


Fig. 1. Left: Binary table $\mathcal{K} = (O = \{1, 2, \dots, 8\}, A = \{a, b, \dots, h\}, I)$ and the object 9. **Right:** The Hasse diagram of the lattice derived from the eight first objects of \mathcal{K} .

Hereafter, both f and g are denoted by $'$. For example, $13' = dgh$ and $bc' = 167$. Furthermore, the compound operators $''$ ($g \circ f(X)$ and $f \circ g(Y)$) are closure operators over $\mathcal{P}(O)$ and $\mathcal{P}(A)$ respectively (e.g., $78'' = 13678$ and $fh'' = efh$). Thus, each of them induces a family of *closed* subsets, called \mathcal{C}^O and \mathcal{C}^A respectively. A couple (X, Y) , of mutually corresponding closed subsets, where $X \in \mathcal{P}(O)$, $Y \in \mathcal{P}(A)$, $X = Y'$ and $Y = X'$, is called a (*formal*) *concept* [Wil82], e.g., $(13, dgh)$ is a concept, but $(35, fh)$ is not (see Figure 1 on the right). Within a concept, X is referred to as the *extent* and Y as the *intent*.

Furthermore, the set $\mathcal{C}_{\mathcal{K}}$ of all concepts of the context $\mathcal{K} = (O, A, I)$ is partially ordered by extent inclusion: $(X_1, Y_1) \leq_{\mathcal{K}} (X_2, Y_2) \Leftrightarrow X_1 \subseteq X_2$. $\mathcal{L} = \langle \mathcal{C}_{\mathcal{K}}, \leq_{\mathcal{K}} \rangle$ is a complete lattice where join (\vee) and meet (\wedge) are given by:

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

For example, the join and the meet of $c_{\#6} = (26, ac)$ and $c_{\#3} = (13678, d)$ are $(12345678, \emptyset)$ and $(6, abcd)$ respectively (see Fig. 1).

The function μ (respectively ν) maps an object o (attribute a) into the *minimal* (respectively *maximal*) concept in the lattice having that object (attribute):

$$\mu : O \rightarrow \mathcal{C}_{\mathcal{K}}, \mu(o) = (o'', o') \text{ and } \nu : A \rightarrow \mathcal{C}_{\mathcal{K}}, \nu(a) = (a', a'')$$

For example, within the lattice in Fig. 1, $\mu(1)=c_{\#11}$ and $\nu(f)=c_{\#9}$.

Iceberg lattices are upper sets of the concept lattice which are generated by a specific sort of maximal anti-chains of the lattice. They are made up of frequent concepts only. The frequency/support γ of a given concept $c = (X, Y)$ is: $\gamma(c) = \|X\| / \|O\|$. Intuitively, an iceberg arises through a complete horizontal cut of the lattice $\mathcal{L} = \langle \mathcal{C}, \leq_{\mathcal{K}} \rangle$ into two parts with respect to a *minimal support threshold*, that is, a real number α in $]0, 1]$. The upper part, denoted $\bar{\mathcal{L}}^{\alpha} = \langle \bar{\mathcal{C}}^{\alpha}, \leq_{\mathcal{K}} \rangle$ where $\bar{\mathcal{C}}^{\alpha} = \{c | c \in \mathcal{C}, \gamma(c) \geq \alpha\}$ is usually referred to as the *iceberg concept lattice* [STB⁺02].

Definition 1. *The α -iceberg, $\alpha \in]0, 1]$, of concept lattice \mathcal{L} is the sup-semi-lattice $\bar{\mathcal{L}}^{\alpha} = \langle \bar{\mathcal{C}}^{\alpha}, \leq_{\mathcal{K}} \rangle$.*

The lower part is denoted $\underline{\mathcal{L}}^{\alpha} = \langle \underline{\mathcal{C}}^{\alpha}, \leq_{\mathcal{K}} \rangle$. In data mining, only icebergs are considered as they include the most general concepts. For example, the iceberg $\bar{\mathcal{L}}^{0.30}$ obtained from the complete lattice on the right-hand side of Fig. 1 using $\alpha=0.30$ is shown in Fig. 2, on the left.

2.2 Incremental construction of the lattice

Incremental methods construct the lattice \mathcal{L} starting from $\mathcal{L}_0 = \langle \{(\emptyset, A)\}, \emptyset \rangle$ and gradually incorporating a new object o_i into the lattice \mathcal{L}_{i-1} which corresponds to a table $\mathcal{K}_{i-1} = (\{o_1, \dots, o_{i-1}\}, A, I)$. Each incorporation involves a set of structural updates [VMGM02].

The basic approach (see [GMA95]) exploits a property of the *Galois connection* saying that both families of closed subsets are themselves closed under intersection [Bir40]. Thus, to turn an arbitrary lattice \mathcal{L} , upon adding an object o , into the lattice \mathcal{L}^+ of the context $\mathcal{K} = (O^+, A, I^+)$, where $O^+ = O \cup \{o\}$ and $I^+ = I \cup \{o\} \times o'$, one only needs to generate some extra concepts and integrate them into the existing structure. The target concepts have intents which are intersections of an existing intent in \mathcal{L} and o' that are not themselves intents in \mathcal{L} and are thus called *new* (denoted $\mathbf{N}^+(o)$). Furthermore, three categories of concepts are distinguished in \mathcal{L} : *genitor* concepts ($\mathbf{G}(o)$) which support the generation of new concepts; *old* concepts ($\mathbf{U}(o)$) which remain unchanged; and *modified* concepts ($\mathbf{M}(o)$) which evolve by integrating o into their extents

while their intents remain stable. A formal definitions of these subsets and a theoretical framework for lattice maintenance is given in [VMGM02].

The main tasks for a reconstruction algorithm include the identification of the three sets within \mathcal{L} and the creation/integration of the new concepts. These jointly lead to the construction of the target lattice \mathcal{L}^+ as explained in [VHM03]. In the rest of this paper, the sets \mathbf{G}^+ , \mathbf{U}^+ and \mathbf{M}^+ refer to the lattice \mathcal{L}^+ . When no confusion is possible, \mathbf{G} , \mathbf{U} and \mathbf{M} are assimilated to \mathbf{G}^+ , \mathbf{U}^+ and \mathbf{M}^+ , respectively.

2.3 Incremental iceberg update problem

The task of iceberg maintenance amounts to the efficient transformation of $\bar{\mathcal{L}}^\alpha$ into $\bar{\mathcal{L}}^{\alpha+}$. To that end, one could apply the above incremental strategy, although this alone will not be enough. Actually, some concepts from $\bar{\mathcal{L}}^\alpha$ will have to be removed as no more frequent in \mathcal{L}^+ . Conversely, unfrequent concepts in \mathcal{L} can become themselves or generate another one which is frequent in \mathcal{L}^+ . The corresponding concepts from $\bar{\mathcal{L}}^{\alpha+}$ cannot be reached by the straightforward lattice maintenance and therefore require additional effort.

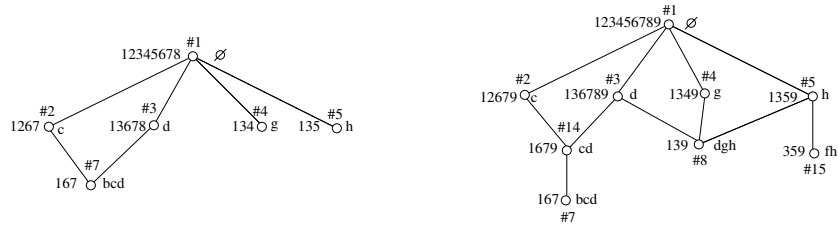


Fig. 2. Left: iceberg $\bar{\mathcal{L}}^{0.30}$ with $O=\{1, \dots, 8\}$. Right: iceberg $\bar{\mathcal{L}}^{0.30+}$ with $O^+=O \cup \{9\}$.

Example 1. Assume $\bar{\mathcal{L}}^\alpha$ is the iceberg induced by the object set 12345678 and the threshold $\alpha = 0.30$ (see Fig. 2, on the left) and consider 9 as the new object to be added. The result of this insertion is the iceberg $\bar{\mathcal{L}}^{\alpha+}$ in Fig. 2 on the right.

In the next section, we characterize the set of problematic concepts and lay the foundations of its efficient detection.

3 Theoretical foundations

Obviously, for each (X, Y) in $\bar{\mathcal{L}}^{\alpha+}$, there is a counterpart in \mathcal{L} with an extent $X - \{o\}$. The latter concept may well be in $\bar{\mathcal{L}}^\alpha$ hence (X, Y) will have to be added to the structure by a specific procedure. To define the set of all such concepts, further referred to as *jumpers* and denoted $\mathbf{H}^+(o)$, we consider a strong

cardinality property on X . In fact, $X - \{o\}$ is unfrequent in \mathcal{L} whereas X is frequent in \mathcal{L}^+ (which can only happen if o is in X , i.e., $(X, Y) \in \uparrow\mu(o)$), hence the definition of $\mathbf{H}^+(o)$:

Definition 2. $\mathbf{H}^+(o) = \{(X, Y) \in \uparrow\mu(o) \mid \|X - \{o\}\| \leq \alpha \|O\|, \|X\| \geq \alpha \|O^+\|\}$.

For example, the jumpers induced by the new object 9 are $\mathbf{H}^+(9) = \{c_{\#8}, c_{\#15}\}$ (see Fig. 2). The next property follows directly:

Property 1. $(X, Y) \in \mathbf{H}^+(o) \iff \alpha \cdot \|O\| + \alpha \leq \|X\| \leq \alpha \cdot \|O\| + 1, \alpha \in]0, 1]$.

Observe that within the range $[\alpha \cdot \|O\| + \alpha, \alpha \cdot \|O\| + 1]$ there is at most one integer whatever the size of the set O :

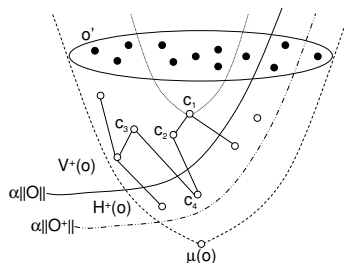
Property 2. $\forall n \in \mathbb{N}, \forall \alpha \in]0, 1], \|[\alpha \cdot n + \alpha, \alpha \cdot n + 1] \cap \mathbb{N}\| \leq 1$.

In other words, all jumpers have the same extent size, hence $\mathbf{H}^+(o)$ is an anti-chain in \mathcal{L}^+ . Moreover, the iceberg update must not only compute the jumpers but also link them to the respective upper covers as in the complete lattice case. Observe that all the target upper covers are in fact frequent concepts having the object o in their extent. Therefore, the set of potential upper covers of the jumpers are members of the frequent part of the order filter $\uparrow\mu(o)$ in \mathcal{L}^+ which further correspond to frequent antecedents in \mathcal{L} . The target concepts are called *visible* and denoted $\mathbf{V}^+(o)$:

Definition 3. $\mathbf{V}^+(o) = \{(X, Y) \in \uparrow\mu(o) \mid \|X\| \geq \alpha * \|O\| + 1\}$.

According to Def. 3, all immediate successors of a jumper concept are indeed in $\mathbf{V}^+(o)$.

Property 3. $\forall c \in \mathbf{H}^+(o), \forall \bar{c} \in \mathcal{L}^+, c \prec^+ \bar{c}$, entails $\bar{c} \in \mathbf{V}^+(o)$.



Property 4. $\forall c_1, c_2 \in \uparrow\mu(o)$, if $c_1 \prec^+ c_2$, then $\exists a \in o' - \text{Int}(c_2) : c_1 = \nu(a) \wedge c_2$.

Therefore, given a concept c in $\mathbf{V}^+(o)$, to find its lower covers in $\mathbf{H}^+(o)$, one has to examine the set of meets of c and the appropriate attribute-concepts. The operator corresponding to an attribute a is denoted \wedge_a : $\wedge_a c = \nu(a) \wedge c$. When the concept $\nu(a) \wedge c$ is to be computed, its extent is $\text{Ext}(\wedge_a c) = \text{Ext}(c) \cap \text{Ext}(\nu(a))$ (or $\text{Ext}(c) \cap a'$), while its intent is more delicate to obtain in general. However, as only $\text{Cov}^l(c)$ are interesting, the following property can be used. It says that if c_1 precedes c_2 , then the attributes in the difference of their intents are exactly those which satisfy $c_1 = \nu(a) \wedge c_2$.

Property 5. $\forall c_1, c_2 \in \mathbf{V}^+(o)$ s.t. $c_1 \prec c_2$, $\text{Int}(c_1) - \text{Int}(c_2) = \{a \in A \mid \wedge_a c_2 = c_1\}$.

Thus, the intent of the concept $\wedge_a c_2$ can be obtained by adding to $\text{Int}(c_2)$ any of the attributes \bar{a} , such that $\text{Ext}(c_2) \cap \bar{a}' = \text{Ext}(c_2) \cap a'$. Clearly, as this holds for all lower covers of a given concept, several subsets of attributes in $o' - \text{Int}(c_1)$ may lead to the generation of valid intents. In contrast, any of the remaining attributes, say a , will produce only a partial intent as the corresponding concept $\wedge_a c_2$ is not a lower cover. The set of attributes which generates jumpers is denoted $\text{gen}_h(c)$. Thus, the main problem is to filter, for any concept c in $\mathbf{V}^+(o)$, the attributes in $\text{gen}_h(c)$.

Seemingly the best way to carry out this filtering is to test if, given a candidate attribute a , the size of the intersection $\text{Ext}(c_2) \cap \bar{a}'$ satisfies Prop. 1). As intersection computation may be expensive, it is important to reduce the number of tests, e.g., by dropping candidates whose $\wedge_a c$ cannot be a jumper. Given a concept c in $\mathbf{V}^+(o)$ and an attribute a in $o' - \text{Int}(c)$, $\wedge_a c$ can only fall in: (i) $\underline{c}^{\alpha+}$, (ii) $\mathbf{H}^+(o)$, or (iii) $\mathbf{V}^+(o)$. Notice that these cases are completely orthogonal to the lower cover status of $\wedge_a c$ with respect to c hence, non immediate predecessors that are still in $\mathbf{H}^+(o)$ must be ignored so that a jumper is generated only by a visible upper cover and not by further successors.

If (i) and (ii) are hard to solve without intersection computing, (iii) can be avoided through an upward intent propagation in $\mathbf{V}^+(o)$, i.e., from concepts in $\downarrow c \cap \mathbf{V}^+(o)$ to c . More precisely, cumulated intents of all predecessors are transmitted to a concept c so that all the included attributes could be ignored during jumper tests. Thus, we define a mapping $T : \mathbf{V}^+(o) \rightarrow \mathcal{P}(A)$ with $T(c) = \bigcup_{\hat{c} \in \downarrow c \cap \mathbf{V}^+(o)} \text{Int}(\hat{c})$.

Even if $T(c)$ can be obtained before the computation of $\mathbf{H}^+(o)$, it is more advantageous to do it during the global traversal of $\mathbf{V}^+(o)$. This way, $T(c)$ can be seconded by the set of attributes which generate jumpers that are not lower covers of c (i.e., are lower covers of a predecessor of c). Thus, we define a mapping T^h that yields all the attributes of frequent predecessors of c except those which generate immediate predecessors in $\mathbf{H}^+(o)$, i.e., $\text{gen}_h(c)$.

Definition 4. $T^h : \mathbf{V}^+(o) \rightarrow \mathcal{P}(A)$, $T^h(c) = \{a \mid a \in \text{Int}(\hat{c}), \hat{c} \in \downarrow c \cap \mathbf{V}^+(o) - \text{Cov}^l(c) \cap \mathbf{H}^+(o)\}$.

$T^h(c)$ represents the maximal set of attributes that can be subtracted from o' , *a priori*, for a given visible concept c . It can be computed from the values

of T^h on immediate visible predecessors of c and the sets of jumper generating attributes for the same concepts.

Property 6. $T^h(c) = \bigcup_{\hat{c} \in Cov^l(c) \cap \mathbf{V}^+(o)} T^h(\hat{c}) \cup h(\hat{c})$.

The set of all attributes which are effectively examined for a concept c is called the *pool* of c and denoted $pool(c)$. Clearly, $pool(c)$ is $o' - T^h(c)$.

4 The proposed incremental method

The results presented in the previous section are transformed into an algorithmic procedure, called MAGALICE, that updates an iceberg upon the insertion of a new object into the context.

4.1 Description of Magalice

The Algorithm 1 (UPDATE-ICEBERG-LATTICE) represents two main parts. The first one is a procedure call (line 3) and is in charge of the insertion of the new object into the iceberg lattice considered as complete lattice (see [VHM03] for algorithm details). The second one (lines 4-7) implements the core tasks of an iceberg maintenance algorithm. It starts by dropping out the unfrequent concepts in $\bar{\mathcal{L}}^{\alpha+}$ (lines 4-6). The second step consists in computing concepts in $\mathbf{H}^+(o)$.

```

1: procedure UPDATE-ICEBERG-LATTICE(In:  $\mathcal{L}$  an iceberg lattice,  $\alpha$  a minimum support,
    $o$  a new object, Context an indexed set of objects)
2:
3: ADD-OBJECT( $\mathcal{L}, o$ )
4: for all  $c$  in  $\mathcal{L}$  do
5:   if  $\gamma(c) < \alpha$  then
6:     DROP( $\mathcal{L}, c$ )
7: FIND-FREQUENT-LOWER-COVERS( $\mathcal{L}, \alpha, \text{Context}, c, o$ )
    
```

Algorithm 1: Iceberg update upon the insertion of new object within Magalice

4.2 Computation of jumpers

The aim of Algorithm 2 (FIND-FREQUENT-LOWER-COVERS) is to generate lower covers of visible concepts ($\mathbf{V}^+(o)$) using the respective pool. The skeleton of the algorithm represents a bottom-up traversal of the iceberg $\bar{\mathcal{L}}^{\alpha+}$. The linear extension of the order is ensured by a preliminary sorting (line 9). Indeed, the first step sorts concepts in $\mathbf{V}^+(o)$ according to the descending order of intent size. During the main stage, each concept c in $\mathbf{V}^+(o)$ is processed in order to generate its jumpers. This processing includes the computation of the frequent

extent intersections X which are stored within the generating-attributes Y in the jumper candidate set. At the end of the processing of the c , the pairs (X, Y) are used to generate the effective jumpers which are in turn added to the global jumper set $\mathbf{H}^+(o)$.

```

1: procedure FIND-FREQUENT-LOWER-COVERS(In/Out:  $\bar{\mathcal{L}}^\alpha$  an iceberg lattice,  $\alpha$  a minimum support, Context an indexed set of objects,  $o$  the new object)
2:
3: Local : Jumpers : set of concepts
4: Local : Th, Pool,  $h$  : set of attributes
5: Local : Extent : set of objects
6: Local : Candidates : set of pairs of tuples  $\langle X, Y \rangle$ 
7:
8: Jumpers  $\leftarrow \emptyset$ 
9: SORT( $\mathbf{V}^+(o)$ ) {in descending order of the Intent}
10: for all  $c \in \mathbf{V}^+(o)$  do
11:   Candidates  $\leftarrow \emptyset$ ; Th  $\leftarrow$  Int( $c$ );  $h \leftarrow \emptyset$ 
12:   for all  $\bar{c} \in \text{Cov}^l(c)$  do
13:     Th  $\leftarrow$  Th  $\cup$   $T^h(\bar{c}) \cup h(\bar{c})$ 
14:     Pool  $\leftarrow$   $o' - \text{Th}$ 
15:     while not Pool =  $\emptyset$  do
16:        $a \leftarrow$  extract-first(Pool); Extent  $\leftarrow$  Ext( $c$ )  $\cap a'$ 
17:       if  $\alpha \cdot \|O\| + \alpha \leq \|Extent\| < \alpha \cdot \|O\| + 1$  then
18:          $\bar{c} \leftarrow$  LOOKUP(Extent, Jumpers)
19:         if  $\bar{c} \neq \text{NULL}$  then
20:            $\text{Cov}^l(c) \leftarrow$   $\text{Cov}^l(c) \cup \{\bar{c}\}$ ;  $h \leftarrow$   $h \cup \text{Int}(\bar{c})$ ; Pool  $\leftarrow$  Pool - Int( $\bar{c}$ )
21:         else
22:            $n \leftarrow$  LOOKUP(Extent, Candidates)
23:           if  $can \neq \text{NULL}$  then
24:              $can.Y \leftarrow$   $can.Y \cup \{a\}$ 
25:           else
26:              $can \leftarrow$  (Extent, (Int( $\bar{c}$ )  $\cup \{a\}$ )); Candidates  $\leftarrow$  Candidates  $\cup \{can\}$ 
27:              $h \leftarrow$   $h \cup \{a\}$ 
28:         for all  $can \in$  Candidates do
29:            $\bar{c} \leftarrow$  NEWCONCEPT( $can.X$ ,  $can.Y$ );  $\text{Cov}^l(c) \leftarrow$   $\text{Cov}^l(c) \cup \{\bar{c}\}$ 
30:           Jumpers  $\leftarrow$  Jumpers  $\cup \{\bar{c}\}$ 
31:            $T^h(c) \leftarrow$  Th;  $gen_h(c) \leftarrow$   $h$ 
32:           UPDATE( $\bar{\mathcal{L}}^\alpha$ , Jumpers)

```

Algorithm 2: Compute Lower covers for a given visible concept.

After an initialization step (line 11), the set Th is established using the propagation mechanism of intents from concepts to their successors (lines 12-13). Then, the pool of c (attributes for potential jumpers) is determined (line 14). The next step is the gradual discovery of jumpers of the current concept (line 15-27). Notice that once the extent of $\wedge_a c$ is computed (line 16) and proved to be frequent (line 17), the algorithm checks whether it is already generated

by another concept (line 18) or not. When the extent of $\wedge_a c$ is found (line 19), first, the corresponding concept \bar{c} is added to the jumpers of the concept c (line 20), then the intent of \bar{c} is used to update both the set of generating-attributes $gen_h(c)$ and the pool (line 20). When the extent of $\wedge_a c$ is not generated by another concept, the algorithm checks the candidates discovered by the current concept (line 22). The result of the check may update the intent of the found candidate (line 24) or create a new candidate (line 26). Next, new jumper concepts are created based on the candidates (line 29), linked to their upper cover (line 29) and added to the list of jumpers (line 30). At the end, attributes of the predecessors in $\mathbf{V}^+(o)$ and discovered jumpers in $\mathbf{H}^+(o)$ are recorded (line 31).

Example 2. To illustrate the jumper computation for a given concept (lines 11-31), let's consider the concept $c_{\#5}$ within the insertion of object 9. Notice that, up to now, the jumpers set $\mathbf{H}^+(9)$ is $c_{\#8}$ (generated by $c_{\#3}$). Since $c_{\#5}$ does not have any predecessor in the iceberg, $pool(c_{\#5}) = cdfg$. The intersection 19 between $Ext(c_{\#5})$ and $\{c\}'$ leads to an unfrequent extent ($||19|| < 3$). The intersection 139 between $Ext(c_{\#5})$ and $\{d\}'$ is frequent. The concept having the resulting extent in $\mathbf{H}^+(9)$ is $c_{\#8}$. Thus, $c_{\#8}$ is used to update $Cov^l(c_{\#5})$, $gen_h(c_{\#5})$ (line 20). As g is in both $pool(c_{\#5})$ and $Intent(c_{\#8})$, no jumper is generated. Therefore, $pool(c_{\#5}) = f$ (line 20). The intersection 359 between $Ext(c_{\#5})$ and $\{f\}'$ is frequent, but is neither in $\mathbf{H}^+(9)$ nor in candidate jumper set (*Candidates*) (line 25). The pair (359, fh) is then added to *Candidates* (line 26). The $pool(c)$ is now empty, a new concept $c_{\#15}$ is created from the candidate (359, fh) (line 29), linked to its upper cover $c_{\#5}$ and added to $\mathbf{H}^+(9)$ (line 30).

4.3 Example

In order to illustrate the way the Algorithm 1 proceeds, assume $\tilde{\mathcal{L}}^\alpha$ the iceberg induced by the object set 12345678 (see Fig. 2 on the left) and consider 9 as the new object whose intent is $\{9\}' = cdfgh$. After the call of ADD-OBJECT (line 3), the set of unchanged concepts is $\mathbf{U}^+(9) = \emptyset$, whereas the set of modified, generators and new concepts are $\mathbf{M}^+(9) = \{c_{\#1}, c_{\#2}, c_{\#3}, c_{\#4}, c_{\#5}\}$, $\mathbf{G}^+(9) = \{c_{\#7}\}$, $\mathbf{N}^+(9) = \{c_{\#14}\}$ respectively. This result is given in the left of Fig. 4. The next step (lines 4-6) drops out all unfrequent concepts which could be old or generator. In this example all concepts in $\tilde{\mathcal{L}}^\alpha +$ remain frequent. The sorted set of visible concepts $\mathbf{V}^+(9)$ used for the generation of jumpers by the procedure FIND-LOWER-COVERS (line 7) is $\{c_{\#14}, c_{\#2}, c_{\#3}, c_{\#4}, c_{\#5}, c_{\#1}\}$.

The Table 4.3 illustrates the running of this procedure on selected concepts within their corresponding pool. For each pool attribute a (table rows), the table should be read as follows: The first column provides $\wedge_a c$, the second is the status of the intersection result, third column presents the set $gen_h(c)$, the fourth one indicates the evolution of the pool variable (*pool*). The Fifth column tracks the candidates set (*Candidates*) while the last column shows the content of the jumpers set ($\mathbf{H}^+(o)$).

Starting with the concept $c_{\#14}$, no jumper is found since concept $c_{\#16}$ remains unfrequent (see Fig. 4). The concept $c_{\#2}$ does not generate any jumper as the

Attribute	Extent	Status	$gen_h(c)$	$pool(c)$	Candidates	$Jumpers(H^+(9))$
Processing of concept $c_{\#3}$						
f	39	Unfreq.	\emptyset	gh	\emptyset	\emptyset
g	139	Freq.	g	h	$(139, dg)$	\emptyset
h	139	Freq.	gh	\emptyset	$(139, dgh)$	$c_{\#8}$
Processing of concept $c_{\#4}$						
c	19	Unfreq.	\emptyset	dfh	\emptyset	$c_{\#8}$
d	139	Freq.	dgh	f	\emptyset	$c_{\#8}$
f	39	Unfreq.	dgh	\emptyset	\emptyset	$c_{\#8}$
g	cancelled by step in line 20 ($Pool \leftarrow Pool - Int(\bar{c})$).					
Processing of concept $c_{\#5}$						
c	19	Unfreq.	\emptyset	dfg	\emptyset	$c_{\#8}$
d	139	Freq.	dgh	f	\emptyset	$c_{\#8}$
f	359	Freq.	$dfgh$	\emptyset	$(359, fh)$	$c_{\#8}, c_{\#15}$
g	cancelled by step in line 20 ($Pool \leftarrow Pool - Int(\bar{c})$).					

Table 1. The trace of Algorithm 2 on concepts $c_{\#3}$, $c_{\#4}$ and $c_{\#5}$ upon the addition of object $(9, cdfgh)$.

computed intersections are unfrequent too. The processing of $c_{\#3}$ leads to the discovery of the jumper $c_{\#8}$. When deal with the concept $c_{\#4}$, the method finds that $c_{\#8}$ is a predecessor of $c_{\#4}$. An order relation is then established between them but there is no jumper to produce. The concept $c_{\#5}$ makes a link with concept $c_{\#8}$ as it is his predecessor and produces the jumper $c_{\#15}$. The last concept $c_{\#1}$ (top of iceberg) does not generate any jumper as its pool is empty.

Finally, the jumpers discovered at the end of the Algorithm-2 are $H^+(9) = \{c_{\#8}, c_{\#15}\}$. The updated iceberg $\bar{\mathcal{L}}^{0.30+}$ which is obtained from $\bar{\mathcal{L}}^{0.30}$ and the object $\#9$ is presented on the right-hand side of Fig. 4.

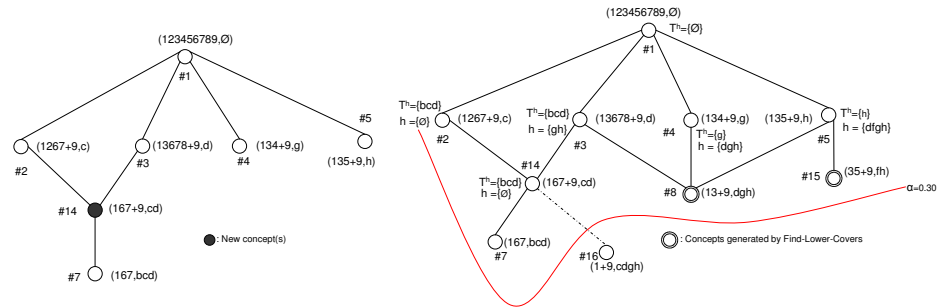


Fig. 4. Jumpers generation ($H^+(9)$).

4.4 Complexity issues

The following table presents basic parameters used in the estimation of the worst case complexity for our incremental algorithm.

Variable	m	l	k	$d(\bar{\mathcal{L}}^{\alpha+})$	$\Delta(l)$
Stands for	$\ A\ $	$\ \mathcal{L}^{\alpha+}\ $	$\ O\ $	$\max_{c \in \bar{\mathcal{L}}^{\alpha+}} (\ Cov^l(c)\)$	$\ \mathcal{L}^{\alpha+}\ - \ \mathcal{L}^{\alpha}\ $

The complexity of Algorithm 2 can be assessed as follows: the sorting of the set $\mathbf{V}^+(o)$ with respect to concept intent sizes (line 9) can be made in a linear time, i.e., $O(l)$, since only the sizes of concepts intents need to be compared. The traversal of this set (line 10) represents the first cost factor and takes $O(l)$ concept examinations as in the worst case all concepts in $\bar{\mathcal{L}}^{\alpha+}$ are also in $\mathbf{V}^+(o)$. The second factor is further split into three additive costs: pool computation cost (lines 11-14), candidates-related cost (lines 15-27) and, the cost of creating and recording new jumpers (lines 28-30). The computation of the pool (lines 11-14) costs $O(m^2)$ since the algorithm performs $O(d(\bar{\mathcal{L}}^{\alpha+}))$ concept examinations (line 12), each examination consists in computing attribute sets unions (line 13) which can be made in $O(m)$. Notice that $O(d(\bar{\mathcal{L}}^{\alpha+}))$ has the upper bound $O(m)$. The candidates-related cost (lines 15-27) is the result of four additive components: the traversal of pool (line 15) which is executed in $O(m)$, extent intersection computation (line 16) which costs $O(k)$, jumper set ($\mathbf{H}^+(o)$) lookup (line 18) that can be made in $O(\Delta(l)k)$ and jumpers candidates lookup (line 22) which costs also $O(mk)$. However, trie-based representation of sets of extents (e.g., in jumpers and candidates) provides a very efficient lookup which cost linear in the number of objects, regardless of the size of the trie. Thus, the cost of the lookup operations is brought back to $O(k)$. This helps assess the candidates-related cost to $O(mk)$. Finally, as creating and properly connecting each jumper to its upper cover (line 29) are executed in constant time, the cost of dealing with new jumpers (line 30) is $O(m)$. The latter cost is due to the fact that the number of new jumpers generated by a given concept is limited by the attribute number. In summary, the total complexity of the Algorithm 2 is bounded by $O(m(m+k)l)$.

The Algorithm 1 performs two traversals to update the iceberg structure. For the first traversal, the Algorithm ADD-OBJECT is used. Its complexity is $O(\Delta(l)k^2 + l(k+m))$ (see [VHM03]). During the second traversal, the Algorithm 1 performs additional steps which guarantee the integrity of the structure. The first step (line 4-6) consists in elimination of unfrequent concepts which can be done in $O(lm)$ as the algorithm checks the frequency of each concept and, if need be, drop it out by visiting its upper covers and updating the order links. During the second step, the Algorithm 2 is used to compute the set $\mathbf{H}^+(o)$ of potential jumpers. As mentioned above, its complexity is in $O(m(m+k)l)$. Consequently, the global restructuring overhead of the iceberg for a single insertion run is in $O(\Delta(l)k^2 + m(k+m)l)$.

5 Experiments and performance evaluation

We conducted a set of experiments in which the practical performances of MAGALICE has been assessed. As the problem of maintaining an iceberg lattice upon evolution in the dataset has not been tackled so far, The performance of MAGALICE have been compared to those of an incremental version of the classical batch algorithm of BORDAT [Bor92] for frequent concept computation. The choice of BORDAT was motivated by the top-down generation of lattice concepts. Both algorithms were implemented in Java, within the 1.1 version of the GALICIA platform³ [VRGR03].

The experiments took place on a Celeron running 1,7 GHz, with 1.5 GB of main memory. The comparisons were performed on the strongly correlated database MUSHROOM (8,124 objects, 119 attributes) and the weakly correlated one T25.I10.D10K (10,000 objects, 1000 attributes) with a varying number of objects.

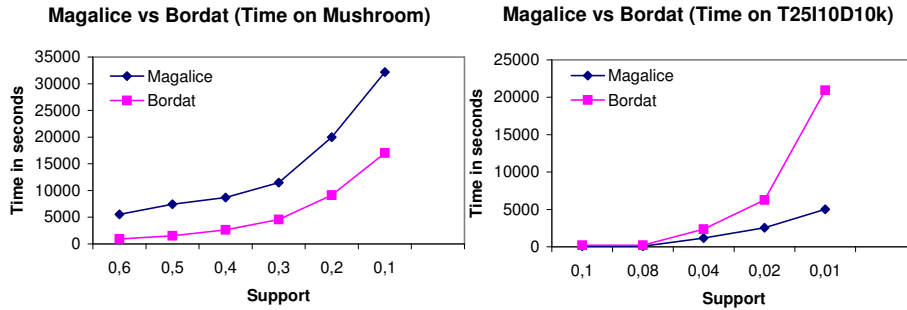


Fig. 5. CPU-time for both MAGALICE and BORDAT for the whole dataset frequent concepts computation using a different value of support.

Two kinds of comparisons have been carried out. The first one (Fig. 5) aimed at comparing the performance of both algorithms as batch procedures. The results show that on T25.I10.D10K database, MAGALICE outperforms BORDAT. Indeed, it runs 2 to 4 times faster. Whereas, on MUSHROOM database, BORDAT is 2 to 5 times faster.

The purpose of the second kind of tests (see Fig. 6) is to show the advantage of MAGALICE running with increment only versus BORDAT reconstruction from scratch of the whole database. On T25.I10.D10K database, the difference between execution time of the both algorithms is important since MAGALICE outperforms BORDAT even in batch mode. On the strongly correlated database, MUSHROOM was split into fixed increments size (2000 objects). MAGALICE is able to perform several hundreds of insertions while BORDAT is running on the entire dataset to reconstruct the whole lattice.

³ See the website at: <http://www.iro.umontreal.ca/~galicia>.

Its noteworthy that GALICIA has been designed as a tool for rapid development of algorithms following the software engineering principles such as the generic code, reusability and extensibility. These are often antinomic with efficiency. Therefore what matters is the relative performance of the compared algorithms and not their absolute time scores.

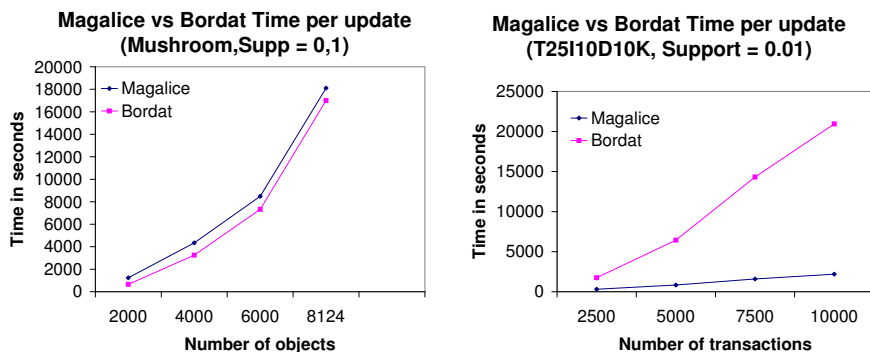


Fig. 6. CPU-time for MAGALICE using a fixed increment of transactions compared to the CPU-time for running BORDAT on the entire transaction set.

6 Discussion

The paper provides a contribution to the incremental maintenance of iceberg concept lattice. A theoretical framework is proposed followed by a novel algorithm which updates the iceberg upon the insertion of a new transaction in the database. Practical performance of the proposed algorithm were compared to those of a batch one.

The next step in our study will focus on the practical analysis of the performance of the various underlying data structures and routines. Moreover, other structures will be studied to speed up the method including an index on unfrequent attributes which enables their reintegration and a list of recent unfrequent concepts. The problems of assembly and split of icebergs will also be studied for the design of appropriate algorithms. A comparison between MAGALICE and major batch algorithms for iceberg concept lattices computation such as TITANIC (including concept's order computation) is to be carried out.

References

- [AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

- [Bir40] G. Birkhoff. *Lattice Theory*, volume XXV of *AMS Colloquium Publications*. AMS, 1940.
- [BM70] M. Barbut and B. Monjardet. *Ordre et Classification: Algèbre et combinatoire*. Hachette, 1970.
- [Bor92] J.-P. Bordat. *Sur l'algorithmique combinatoire d'ordres finis*. Thèse de Doctorat d'État, Université des Sciences et Techniques du Languedoc, Montpellier, 1992.
- [GMA95] R. Godin, R. Missaoui, and H. Alaoui. Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices. *Computational Intelligence*, 11(2):246–267, 1995.
- [PBTL99] 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.
- [STB⁺02] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing Iceberg Concept Lattices with Titanic. *Journal on Knowledge and Data Engineering*, 42(2):189–222, 2002.
- [VHM03] P. Valtchev, M. Rouane Hacene, and R. Missaoui. A generic scheme for the design of efficient on-line algorithms for lattices. In B. Ganter A. de Moor, W. Lex, editor, *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.
- [VMGM02] 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.
- [VRGR03] P. Valtchev, M. H. Rouane, D. Grosser, and C. Roume. An open platform for manipulating lattices. In *Proceedings of the 11th ICCS, Dresden, Germany*, 2003.
- [Wil82] 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.
- [ZH99] M.J. Zaki and C.-J. Hsiao. ChARM: An Efficient Algorithm for Closed Association Rule Mining. RPI Technical Report 99-10, Rensselaer Polytechnic Institute, 1999.