Merge-Based Computation of Minimal Generators

Céline Frambourg[1], Petko Valtchev[2], and Robert Godin[1]

[1] Département d'informatique, UQAM, Montréal (Qc), Canada
[2] DIRO, Université de Montréal, Montréal (Qc), Canada

Abstract. Minimal generators (mingens) of concept intents are valuable elements of the Formal Concept Analysis (FCA) landscape, which are widely used in the database field, for data mining but also for database design purposes. The volatility of many real-world datasets has motivated the study of the evolution in the concept set under various modifications of the initial context. We believe this should be extended to the evolution of mingens. In the present paper, we build up on previous work about the incremental maintenance of the mingen family of a context to investigate the case of lattice merge upon context subposition. We first recall the theory underlying the singleton increment and show how it generalizes to lattice merge. Then we present the design of an effective merge procedure for concepts and mingens together with some preliminary experimental results about its performance.

1 Introduction

Formal Concept Analysis (FCA) has been proved to be a suitable tool for representing the knowledge contained in a database. It is also used as a basis for association rule mining (ARM).

ARM from a transaction database is a classical data mining topic, whereby the most challenging problem is the detection of informative patterns in the transaction sets. A major difficulty with association rules is the prohibitive number of itemsets (and hence association rules) that can be generated even from a reasonably large data set. Moreover, this approach generates a large number of redundant rules. Formal concept analysis (FCA) has helped to solve this problem as it introduces closed itemsets (CIs) , which are a promising solution to the problem of reducing the number of reported association rules. A further step in this direction is the construction of association rule bases from CIs: an operation which largely relies on the notion of closed itemset mingens. Yet another difficulty arises with dynamic databases where the transaction set is frequently updated. Although the necessity of processing volatile data in an incremental manner has been repeatedly emphasized in the general data mining literature, few incremental algorithms for association rule generation (and hence frequent itemset detection) have been reported so far.

CIs mingens are a lossless and concise representation of knowledge in databases. In [8], Kryszkiewicz and Gajek explain the construction of the mingens representation of itemsets and they also show that it is sufficient to determine all the itemsets and their supports. The growth of databases may induce another problem in the data mining task, which is the lack of space. To solve this problem, we propose to distribute the

F. Dau, M.-L. Mugnier, G. Stumme (Eds.): ICCS 2005, LNAI 3596, pp. 181–194, 2005.

computation of the lattices prior to a final merging of the results at the end. Some assembly algorithms have been published recently [18, 16], but most of the methods do not take care of the mingens. That is why we study the evolution of the mingen family during the lattice assembly. To answer that question, we made an exhaustive study of the structures used during CIs family assembly. This study is based on the results of the incremental case that can be found in [19].

We first recall the theoretical framework underlying the lattice assembly and the mingens (Section 2). We will then present the algorithmic aspects of the dynamic computation (Section 3) and finally we will present recent algorithmic results of the problems (Section 4).

2 Closures and Mingens

2.1 FCA Basics

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

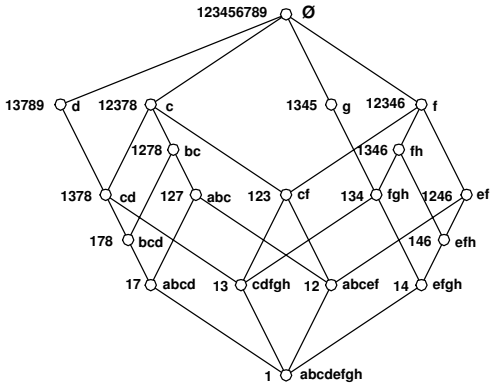

Fig. 1. Left: Context $\mathcal{K}$ (adapted from [4]) with $O = \{1, 2, ..., 9\}$ and $A = \{a, b, ..., h\}$. **Right:** The Hasse diagram of the concept (Galois) lattice derived from $\mathcal{K}$.

FCA studies the way lattices emerge out of data. It considers an incidence relation I over a pair of sets O and A, of objects and attributes, respectively. The relation is given by the matrix which is called a *(formal) context* $\mathcal{K} = (O, A, I)$. Moreover, I gives rise to two $'$ mappings and the composite operators $''$ define *closures* on $\mathcal{P}(O)$ and $\mathcal{P}(A)$, hence each of them induces a family of *closed* subsets, denoted $\mathcal{C}^o_{\mathcal{K}}$ and $\mathcal{C}^a_{\mathcal{K}}$, respectively. Those two families, provided with $\subseteq$ become two *complete lattices* which are dually isomorphic via $'$. These lattices overlap perfectly, thus giving rise to the *concept lattice*[3]. A pair (X, Y) of mutually corresponding subsets, i.e. $X = Y'$ and $Y = X'$, is called a *(formal) concept* whereby X is the *extent* and Y is the *intent*. An itemset Z is called a generator of a closed set X if $Z'' = X$. It is called a minimal generator (mingen) if

[3] Also known as the *Galois lattice*

$T \subset Z$ implies $T'' \subset Z'' = X$. The closure operator on $\mathcal{C}^a_{\mathcal{K}}$ defines an equivalence relation[4] where intents represent the maximum of each equivalence class, and mingens their minimum.

2.2 Mingens in the Literature

In the literature, mingens have been given different names and various properties thereof have been exploited with or without an explicit mention of the concept. For instance, in the database field, *keysets* (see [9]) or *minimal keys* of the sub-relations obtained by decomposing a given relation into 3NF, represent mingens of the attribute sets corresponding to the sub-relations. In early literature on closures and implications, mingens are referred to as *irreducible gaps* thus alluding at their status of minima among all non-closed elements ("gaps") [6] within the same closure class. In a different branch of the same closure field (see [12]), mingens are termed *minimal blockers*, a notion that closely follows that of a minimal transversal of a hypergraph [1].

Remarkable properties of the mingen family include its ideal-shaped structure. In fact, in the Boolean lattice of all the parts of the ground set of a closure family, the mingens represent an order ideal since the family is downwards closed by the subset-of relation. On the cardinality side, it is known that the size of the mingen family can grow up to exponential in the dimensions of the context [10].

Finally, the mingens have been used as target but also as auxiliary structures for lattice algorithms. For instance, the calculation of concepts in Close [11], AClose [11] or Titanic [13] relies on the construction of all mingens. In contrast, in NextClosure [3], the notion of mingen is not explicitly mentioned, but is nevertheless present: a concept intent is canonically generated by its smallest prefix containing a mingen.

3 Summary of the Dynamics of the Closed and the Mingens Computation

So far, three different paradigms of lattice construction have been proposed: batch, incremental and divide-and-conquer. Here we only present the incremental and the merge one.

3.1 Incremental Lattice Construction

The incremental lattice construction paradigm emerged as a response to evolving datasets. Indeed, when a new object is inserted into a dataset, the intent families of the two underlying contexts, i.e., the one "before" ($\mathcal{K}$) and the one "after" ($\mathcal{K}^+$), are tightly related: the later is the closure by intersection of the former augmented by the new object description, o'. The later operation is a computationally less expensive basis for the construction of the lattice of $\mathcal{K}^+$ than the straightforward construction from scratch. Moreover, as shown in [5], instead of computing all possible intersections of subsets from $\mathcal{C}^a \cup \{o'\}$, only pair-wise intersections of o' with an intent from $\mathcal{C}^a$ need to be

[4] $X, Y \subseteq A$ are equivalent iff $X'' = Y''$.

considered. Among those, some are already existing intents while others are specific to $\mathcal{K}^+$. Concepts from the $\mathcal{L}$ are divided into three categories with respect to the intersections produced by their intents. First, some concepts serve as canonical representatives for their intersections: These are the concepts $c = (X, Y)$ whose intents are the closures of the respective intersections, i.e., $Y = (Y \cap o')''$. They are further divided into *modified* (denoted $\mathbf{M}(o)$) and *genitors* (denoted $\mathbf{G}(o)$), meaning that the intersection is itself an intent in $\mathcal{L}$ (i.e., $(Y \cap o')'' = Y \cap o'$) or that this is not the case (i.e., $(Y \cap o')'' \supset Y \cap o'$), respectively. The remaining concepts in $\mathcal{L}$ are called old or unchanged (denoted by $\mathbf{U}(o)$) and have little importance in the approach. As genitor and modified intents are the closures of the corresponding intersections with o', each concept $c = (X, Y)$ from $\mathbf{G}(o) \cup \mathbf{M}(o)$ is the unique maximum in $\mathcal{L}$ among all those generating the intersection $E = Y \cap o'$.

Given $\mathcal{L}$ (concept set and precedence relation) and o, the incremental lattice construction problem amounts to restructuring and completing the data structure representing $\mathcal{L}$ up to reaching a structure that represents $\mathcal{L}^+$. For convenience reasons, the homologous concepts[5] in $\mathcal{L}^+$ of modified and genitors from $\mathcal{L}$ will be denoted by $\mathbf{M}^+(o)$ and $\mathbf{G}^+(o)$, respectively. New concepts in $\mathcal{L}^+$ with respect to $\mathcal{L}$ will be denoted by $\mathbf{N}^+(o)$. The restructuring of $\mathcal{L}$ into $\mathcal{L}^+$ is summarized by the following facts (see [15] for a detailed description):

- $\mathcal{L}$ is isomorphic to a join-sub-semi-lattice of $\mathcal{L}^+$, made of the homologous concepts of those in $\mathcal{L}$,
- the suborder of $\mathcal{L}^+$ induced by the set of new concepts $\mathbf{N}^+(o)$ is isomorphic to the suborder induced by $\mathbf{G}^+(o)$, the homologous concepts of the genitors in $\mathcal{L}$.

3.2 Computation of the Mingens

In a way similar to lattice construction, the incremental mingen computation amounts to transforming the mingen family of an initial context to the one of the augmented context (see [17]). The reasoning underlying the transformation is based on the Boolean lattice 2^A and the equivalence relation induced by the intent family $\mathcal{C}^a$.

First, the equivalence relation of the augmented intent family $\mathcal{C}^{a+}$ is a refinement of the one corresponding to $\mathcal{C}^a$. The classes of the initial relation either remain stable in the new one or are split into two new classes. Second, split classes correspond to genitors, while modified and old have their classes stable. Thus, given a genitor c of intent Y, the resulting two classes for $\mathcal{C}^{a+}$ correspond to the intent of the homologous concept in $\mathcal{L}^+$, i.e., Y, and to the respective new intent $Y \cap o'$, respectively. Recall that $Y \cap o'$ is a former non-closed subset that becomes closed and hence its respective class for $\mathcal{C}^{a+}$ is made of non-closed elements that lay in the class of Y whenever $\mathcal{C}^a$ is considered. For instance, Fig. 5 shows the evolution of the equivalence class of the intent $cdfgh$ in 2^A (diagram on the top) after the insertion of a new closed set d (diagram on the bottom left).

The above fact explains why only generators of genitor concepts need to be examined in the transformation. Indeed, on the one hand, part of those become generators of

[5] A concept from a context is homologous to another one from a different context if both have the same intent.

the respective new intent, while on the other hand, new generators are emerging for the genitor intent. The new generators are minimal sets in the newly formed equivalence class for $\mathcal{C}^{a+}$ that were not minimal in the larger class from $\mathcal{C}^{a}$. In the above example, the initial set of generators of $cdfgh$ is cg, d, ch. After d becomes closed, its equivalence class, here consisting of d itself, is split from the former class of $cdfgh$. The new set of generators of $cdfgh$, i.e., minima in the new equivalence class is much larger: cg, cd, df, dg, dh, ch.

Formally, let us denote by $gen_{\mathcal{K}}()$ the set of mingens of a concept (intent) within the context $\mathcal{K}$ (the subscript will be skipped whenever confusion is excluded). The following property summarizes the evolution in the mingen family between $\mathcal{K}$ and $\mathcal{K}^{+}$:

Proposition 1 *For any $c = (X, Y)$ in $\mathbf{G}(o)$, let c_+ and $\bar{c}$ be its homologous concept in $\mathcal{L}^{+}$ and the generated new concept, respectively ($\bar{c} = (X \cup \{o\}, Y \cap o')$ and $c_+ = (X, Y)$). The sets of generators of $\bar{c}$ and c_+ are as follows:*

- $gen_{\mathcal{K}^{+}}(\bar{c}) = gen_{\mathcal{K}}(c) \cap 2^{Y \cap o'}$,
- $gen_{\mathcal{K}^{+}}(c_+) = \min((gen_{\mathcal{K}}(c) - 2^{Y \cap o'}) \cup (gen_{\mathcal{K}}(c) \cap 2^{Y \cap o'} \times \{Y - o'\}))$.

The only non-trivial aspect of the above proposition is that newly occurring mingens in the class of the homologous concept c_+ are composed of a former mingen that "went" to the new concept $\bar{c}$ to which a single attribute is added which stems from the difference between the genitor intent and the new one ($Y \cap o'$).

```
 1: procedure COMPUTE-MINGENS(In/Out: c, c̄ concepts)
 2:
 3: for all g in c.gens do
 4:    if g ⊆ c̄.Intent then
 5:       c̄.gens ← c̄.gens ∪ {g}
 6: c.gens ← c.gens - c̄.gens
 7: SORT(c̄.gens)
 8: for all ḡ in c̄.gens do
 9:    new-gens ← ∅
10:    for all a in (c.Intent - c̄.Intent) do
11:       gen-cond ← true
12:       for all g in c.gens do
13:          if g ⊆ ḡ ∪ {a} then
14:             gen-cond ← false
15:       if gen-cond then
16:          new-gens ← new-gens ∪ {ḡ ∪ {a}}
17:    c.gens ← c.gens ∪ new-gens
```

Algorithm 1: Computation of the mingens of a new concept and of its genitor.

Algorithm 1 embodies the computation of the mingens for a pair of corresponding concepts, genitor and new. Thus, given a genitor concept c and its corresponding new concept $\bar{c}$, it updates the set of mingens associated with the intent of c and identifies the set of mingens for the intent of $\bar{c}$.

3.3 Generalization of the Incremental Case

The restructuring of the lattice upon insertion of a single new object has been generalized to the case of n such objects. The problem has been reformulated as the one of merging two lattices corresponding to contexts that share their attribute sets. This section describes the basic facts about lattice merge.

Product of Lattices Along Context Subposition Subposition is the horizontal assembly of contexts sharing a same set of attributes [4]. Let $\mathcal{K}_1 = (O_1, A, I_1)$ and $\mathcal{K}_2 = (O_2, A, I_2)$ be two contexts sharing the attribute set A. Their *subposition* is the context $\mathcal{K}_3 = (O_1 \dot{\cup} O_2, A, I_1 \dot{\cup} I_2)$ denoted $\mathcal{K}_3 = \frac{\mathcal{K}_1}{\mathcal{K}_2}$. For example, for the context $\mathcal{K} = (O, A, I)$ given in Fig. 1, let $O_1 = \{1, 2, 3, 4\}$ and $O_2 = \{5, 6, 7, 8, 9\}$. The partial lattices corresponding to $\mathcal{K}_1$ and $\mathcal{K}_2$, say $\mathcal{L}_1$ and $\mathcal{L}_2$, are given in Fig. 2, on the left. In the remainder, to avoid confusion, the derivation operators $'$ for the various contexts will be replaced by the respective indexes i, e.g., ^{2} will stand for $'$ for the context $\mathcal{K}_2$.

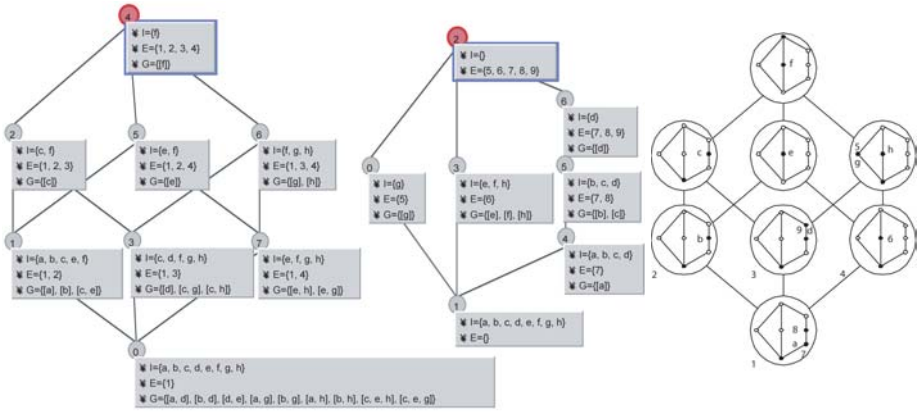

Fig. 2. Left: Factor lattices $\mathcal{L}_1$ and $\mathcal{L}_2$ of the context in Fig. 1. **Right**: The NLD of $\mathcal{L}_3$.

The lattices $\mathcal{L}_1$ and $\mathcal{L}_2$, further called the *factor* lattices, are related to the lattice of the subposed context by two order morphisms. For convenience reasons, the direct product of $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted $\mathcal{L}_{1,2}$ is used in the definition of those morphisms:

Definition 1 *The order morphism $\varphi : \mathcal{L}_3 \rightarrow \mathcal{L}_{1,2}$ maps a concept from the global lattice to a pair of concepts from the partial lattices by splitting its extent over the partial context attribute sets O_1 and O_2:*

$$\varphi((X, Y)) = ((X \cap O_1, (X \cap O_1)'), ((X \cap O_2, (X \cap O_2)'))$$

The morphism $\psi : \mathcal{L}_{1,2} \rightarrow \mathcal{L}_3$ maps a pair of concepts over partial contexts into a global concept by the intersection over their respective intents:

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

In other terms, the composition of factor concepts into a global one is made along the intent dimension shared by $\mathcal{K}_j$ ($j = 1, 2, 3$): The corresponding operation may be seen as the merge of two closure spaces on A. Each node $((X_1, Y_1), (X_2, Y_2))$ from $\mathcal{L}_{1,2}$ is sent to a concept (X, Y) from $\mathcal{L}_3$ such that $Y = Y_1 \cap Y_2$ (e.g., in Fig. 2, $(c_{\#7}, c_{\#3})$ is sent to $(146, efh)$). The underlying mapping ψ is a surjective order morphism that preserves lattice joins (see [18] for details). Conversely, $\mathcal{L}_3$ is mapped onto $\mathcal{L}_j$ ($j = 1, 2$) by simply projecting concept intents on A_j (e.g., $(127, abc)$ is projected to the node $(c_{\#5}, c_{\#6})$). It is noteworthy that $\mathcal{L}_3$ is in general only a meet-sub-semi-lattice of $\mathcal{L}_{1,2}$.

Merge of Factor Lattices Following [18, 16], the factor merge process filters $\mathcal{L}_{1,2}$, and keeps the nodes from the meet-sub-semi-lattice isomorphic to $\mathcal{L}_3$. These are the maximal nodes in the equivalence classes induced by the homomorphism ψ on $\mathcal{L}_{1,2}$ (i.e., equivalence means nodes are sent to the same concept from $\mathcal{L}_3$). More specifically, the maximum node in such a class, say $((X_1, Y_1), (X_2, Y_2))$, is such that if $\psi((X_1, Y_1), (X_2, Y_2)) = (X, Y)$, then $X = X_1 \cup X_2$ and $Y = Y_1 \cap Y_2$. The canonical pairs of concepts can be compared to genitors in the incremental case, hence the concepts from such a pair will be called the *i-genitors* ($i = 1, 2$) of the respective concept from $\mathcal{L}_3$.

The straightforward procedure for lattice merge illustrated by Algorithm 2, presented in the next section, follows the i-genitor definition together with a characterization of the precedence relation in $\mathcal{L}_3$. The procedure, when applied to the lattices on the left of Fig. 2 yields the result presented in Fig. 3.

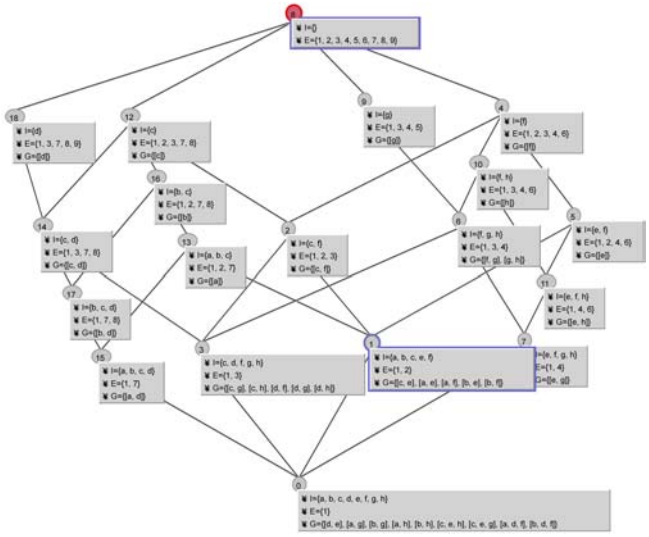

Fig. 3. Result lattice from the merge of $\mathcal{L}_1$ and $\mathcal{L}_2$ in Fig. 2.

4 Generalization of the Incremental Case for the Assembly

4.1 Theoretical Results

Mingen computation can easily be extended to the construction of the subposition-based product of lattices $\mathcal{L}_1$ and $\mathcal{L}_2$. We chose a straightforward approach which consists in applying the incremental paradigm to lattice merging. First, recall that any concept in the product $\mathcal{L}_3$ is created by a pair of factor concepts that play a symmetric role (the genitors) in the generalization of the incremental paradigm. We nevertheless adopt an asymmetric view of the factors and set $\mathcal{L}_1$ to the initial lattice where mingens already exist whereas $\mathcal{L}_2$ is seen as a surrogate for "new" concepts that constitute $\mathcal{L}_3$. Consequently, when such a new concept is detected by the assembly algorithm, its mingens will be computed with respect to its genitor in $\mathcal{L}_1$, i.e., the respective component of the canonical representative in $\mathcal{L}_{1,2}$. Obviously, unlike the specific case of the incremental update, there can be several new concepts per genitor. The challenge will be to determine their mingens without interference between those.

The new concepts created during the merge respect the proposition below.

Proposition 2 *The new concepts corresponding to a 1-genitor* c_1, $\{c_3 \in \mathcal{C}_3 | \Pi_1(\varphi(c_3)) = c_1\}$ *where* Π_1 *is the projection over* $\mathcal{C}_1$ *operator, have intents that lay in the equivalence class* $[Intent(c_1)]_{11}$ *in* 2^A*. Those new concepts will be denoted* $prod_1(c_1)$ *:*

$$\forall c_3 \in prod_1(c_1), Intent(c_3)^{11} = Intent(c_1)$$

Proof. (sketch)

1. $Intent(c_3) \subseteq Intent(c_1)$
2. Let (c_1, c_2) be the genitor pair. $Intent(c_1) \cap Intent(c_2) = Intent(c_3)$ and (c_1, c_2) is maximal. This means that $(Intent(c_1), Intent(c_2))$ is minimal in $(2^A \times 2^A, \subseteq \times \subseteq)$. The canonicity property proves that $Intent(c_1)$ and $Intent(c_2)$ are also minimal. As $Intent(c_1)$ is closed, we have $Intent(c_3)^{11} = Intent(c_1)$.

This equivalence class is partitioned into finer classes according to the ^{33} closure and a unique new concept has the same intent as c_1.

Our aim is to simulate the incremental computation of the mingen family in the merge process. We assume that it is possible but we have to choose a strategy for the concepts insertion. We follow some criteria for this choice, as our main goal was the effectiveness. First, we tried to define a program that will compute the mingens with a minimal number of computation operations, but we also wanted to have no redundancies during the computation.

Three strategies may be suitable. The first one amounts to computing the concepts during a "*top-down*" lattice $\mathcal{L}_3$ exploration (that also means a "*bottom-up*" exploration of $[Intent(c_1)]_{11}$ in the 2^A lattice), the second one is a "*bottom-up*" lattice $\mathcal{L}_3$ exploration ("*top-down*" exploration of $[Intent(c_1)]_{11}$) and the last one is a direct strategy, which means that we make no use of incrementality, but that all the mingens will have to be computed once.

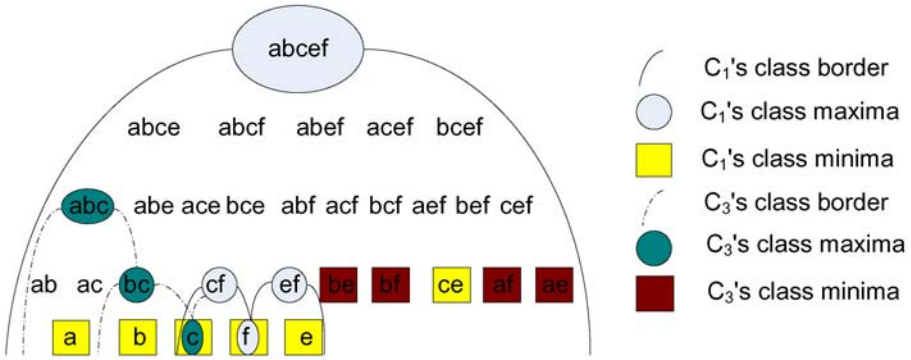

Fig. 4. Example of the partitioning.

The last strategy is the ideal one as it computes the mingens with no redundancies, but we do not have enough information to use the last strategy yet but it could be the object of further study.

The second strategy is that whenever a new concept c_3 is inserted, all its successors of $[Intent(c_1)]_{11}$ in $\mathcal{L}_3$ have to be computed. This means that the $[Intent(c_3)]_{33}$ is only really accessible at the end of all the $prod_1(c_1)$ concept insertions. So, $gen_3(c_3)$ may only be established once all insertions have been made. Therefore, the second strategy is not a suitable one except if the mingens are computed via a batch process.

The first strategy is then the only one which is suitable. We can compute the $gen(c_3)$ set from a temporal mingens list associated to c_1, that reflects the current state of the insertion process in the $[Intent(c_1)]_{11}$ class. We can also say that it is as if each c_3 was inserted separately in an incremental way and in order. The $gen^{inc}(c_1)$ list, representing the mingens of c_1 which evolve during the computation, represents after each insertion of a $c_{3,i}$, the minima of $[Intent(c_1)]_{11} - \bigcup_{j \leqslant i} [Intent\,(c_{3,j})]_{33}$. At the end, the mingens lists are up to date. Once the $gen_3(c_{3,i})$ list is computed, it will never be recomputed again.

The mingens list of a new concept of $\mathcal{L}_3$ may be described as follows:

Proposition 3 *Given an order* $(c_{3,1}, ..., c_{3,n})$ *such that* $\forall i, j | 1 \leqslant i \leqslant j \leqslant n,\ c_{3,j} \not\leqslant_3 c_{3,i}$*:*

$$gen_3\,(c_{3,i}) = \left\{ \begin{array}{l} Y | Y \subseteq Intent\,(c_{3,i}) \\ \quad Y \in \min \left([Intent\,(c_1)]_{11} - \bigcup\limits_{j \leqslant i} [Intent\,(c_{3,j})]_{33} \right) \end{array} \right\}$$

This strategy was chosen for consecutive insertions of the new concepts. It may be explained by the fact that there exist an equivalence between the merge, restricted to the concept c_1, and the insertion of n new concepts in $\mathcal{L}_1$ where c_1 is a genitor. This

approximatively amounts to inserting n new objects having the respective intents of each $c_{3,i}$. By this strategy, we insure that whenever a new $c_{3[c_1]}$ is inserted, all the concepts with a smaller intent are computed and that the mingens can be computed with the mingens of c_1. With this strategy, we are sure that the mingens of every new concept from the class $[Intent(c_1)]_{11}$ can be computed by looking only at the mingens of one concept, c_1. These mingens will have to be updated after each insertion. Given the set of the intents of the new concepts generated by c_1, say $\mathcal{C}_3^a \cap [Intent(c_1)]_{11}$, the previous condition imposes that at any moment the set of already inserted concepts corresponds to an order ideal of that set, provided with inclusion order. A noteworthy fact about the concrete computation method is that it perfectly fits the merge algorithm and Algorithm 1 (with parameters c_1 and c_3). The resulting algorithm is described below (Algorithm 2). Moreover, along all the insertions, the temporary set of mingens that are to be considered for the next insertion is stored at the genitor node within $\mathcal{L}_1$. Indeed, the concept corresponding to c_1 in $\mathcal{L}_3$ (i.e., with the same intent) will be the last one to be created since its intent is the greatest element of the equivalence class.

4.2 Implementation

The COMPUTE-MINGENS method has been associated to the merge process ([17]). The resulting algorithm is given in Algorithm 2.

```
 1: procedure MERGE(In: L1, L2 lattices; Out: L3 a lattice)
 2:
 3: L ← ∅
 4: SORT(C1); SORT(C2) {Decreasing order}
 5: for all (ci, cj) in L1 × L2 do
 6:    I ← Intent(ci) ∩ Intent(cj)
 7:    if CANONICAL((ci, cj), I) then
 8:       c ← NEW-CONCEPT(Extent(ci) ∪ Extent(cj),I)
 9:       L3 ← L3 ∪ {c}
10:       UPDATE-ORDER(c, L3)
11:       COMPUTE-MINGENS(ci, c)
```

Algorithm 2: Assembling the global Concept lattice from a pair of partial ones.

The concept c_i from the lattice $\mathcal{L}_1$ is used as a buffer for the intermediate computation. The COMPUTE-MINGENS method modifies c_i's mingens in a destructive way, i.e. when all the concepts created from c_i have been inserted in $\mathcal{L}_3$, the family of mingens of c_i in $\mathcal{L}_1$ is empty.

For example, the evolution of the equivalence class associated to $cdfgh$ (from $\mathcal{L}_1$) is depicted in Fig. 5. Indeed, the inner loop of Algorithm 2 discovers three new concepts with intents d, cd, and $cdfgh$, respectively, and in this order. These are gradually "inserted" in the class of $cdfgh$: the mingens of the new intent are computed and those of $c_1 = (13, cdfgh)$ are updated in $\mathcal{L}_1$. Thus, the new intent d which corresponds to an

initial mingen of the class forces the creation of four new mingens (cd, df, dg, dh). In contrast, the closed cd merely converts a former mingen into a closure.

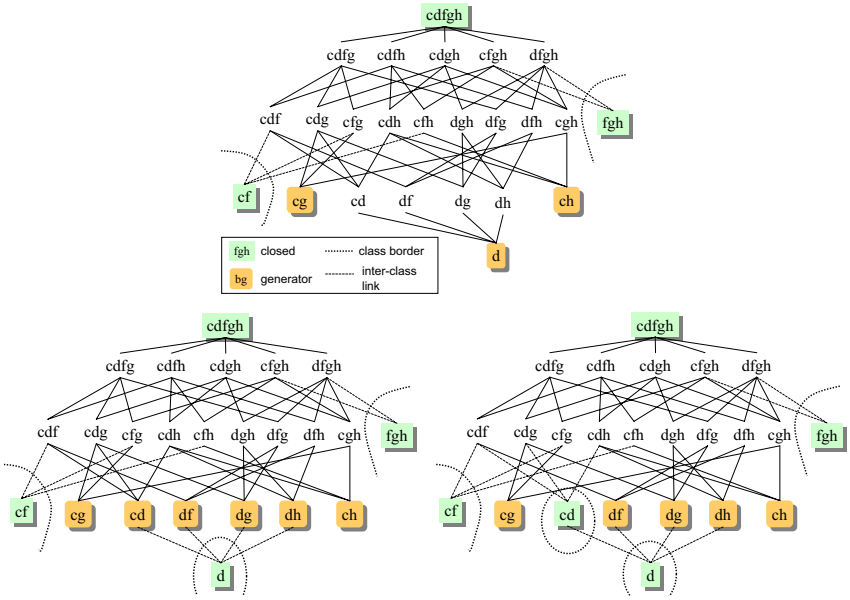

Fig. 5. The evolution of the equivalence class of the closed set $cdfgh$ in $\mathcal{P}(A)$ (initial state up) during the assembly process: after the creation of the new closed d (*left*) and after the creation of cd (*right*).

$Intent(c_1)$	$gen^{inc}_{i-1}(c_1)$	$Intent(c_i)$	$gen(c_i)$	$gen^{inc}_{i}(c_1)$
$cdfgh$	$\{d, cg, ch\}$	d	$\{d\}$	$\{cd, cg, ch, df, dg, dh\}$
$cdfgh$	$\{cd, cg, ch, df, dg, dh\}$	cd	$\{cd\}$	$\{cg, ch, df, dg, dh\}$
$cdfgh$	$\{cg, ch, df, dg, dh\}$	$cdfgh$	$\{cg, ch, df, dg, dh\}$	$\{\}$

Table 1. The evolution of the equivalence class of the closed set $cdfgh$ in $\mathcal{P}(A)$.

4.3 Performance Tests

The MERGE algorithm was implemented in `Java`, within the 2.0 version of the Galicia platform[6]. The method has been tested as a stand-alone application and its performance was compared with two incremental methods, one with the incremental computation of the mingens, and the other with a batch computation of the mingens (computed after

[6] See the website at: `http://www.iro.umontreal.ca/~galicia`.

each object insertion). The experiments were done on a Windows PC station (Pentium Xeon 3.06 GHz with 1.2 GB of RAM) using various subsets of the IBM transaction database T25I10D10K. This dataset is made out of 10 000 transactions over a set of 10 000 items. It is known to be a sparse one, with an average of 28 items per transaction. We did not use a dense dataset, as the lattices to merge contains too many concepts and require too much resources.

In order to improve the results, we combined the COMPUTE-MINGENS algorithm to a batch method called JEN (in [2]). In fact, when a concept is created by the bottom node, its mingens will be computed by JEN. In all the other cases, the COMPUTE-MINGENS algorithm is used. This combination has been motivated by the fact that JEN computes the mingens using the information provided by the successors rather then by the predecessors. This is particularly beneficial for large attribute sets, where the concepts created by the bottom, may imply a lot of computation operation. For example, for the context described previously, that contains 10 000 attributes, suppose the first object contains 28 attributes, the mingens computation of that concept will produce 279 216 operations. The way JEN works has helped solve this problem. This method has also been generalized in the merge case, as the partial order insure that a node is only computed after all its successors have been computed.

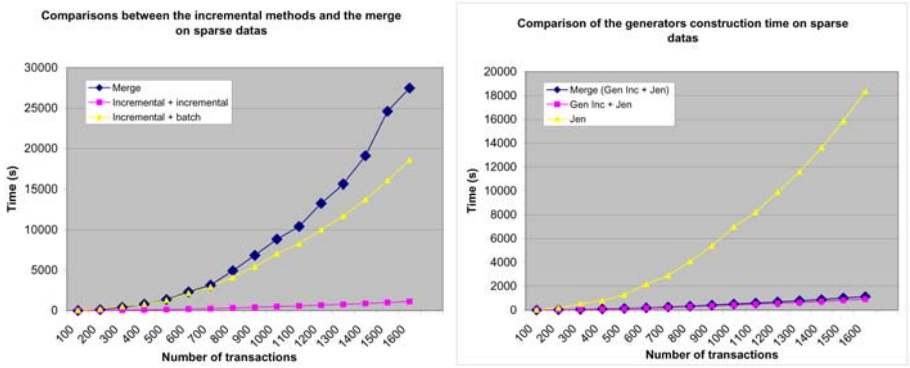

Fig. 6. Cumulative CPU-time for all three algorithms on transaction batches up to 1600 drawn from the T25I10D10K dataset.(*Left*) Lattice construction and computation of the mingens (*Right*) computation of the mingens only.

The graphs drawn in Fig. 6 summarize our findings so far. They clearly show that the incremental method gives currently better results than the merge process. This fact is not surprising given the large number of concepts that the merge algorithm must examine on each merge operation ($l_1 \cdot l_2$, where $l_1 = |\mathcal{L}_1|$ and $l_2 = |\mathcal{L}_2|$) and the even larger number of mingens. However, when we extracted the mingen computation time from the global execution time, we can see that the merge method is about as good as the incremental method and that they are 10 times faster than the batch method used in an incremental way.

The results show that the incremental mingen computation method and its generalization seem promising. It means that provided a good merge algorithm for lattice construction, where the mingens computation can be applied, we would be able to have a better execution time than batch algorithms.

5 Conclusion

The work presented here builds on a previous study of the incremental maintenance of the mingens family of a context. We investigated the case of lattice merge upon context subposition and showed a way to extend the incremental update of the mingen family to this more general case. To that end, first, a precise characterisation of the lattice structures involved in mingen computation/update was provided. The characterization was then embedded into a concrete update method for both concepts and mingens. The performances of the new method were compared to those of two other methods performing concept and mingen construction: an incremental lattice builder (ADD-OBJECT) coupled to a batch procedure for mingen computation (JEN), and a purely incremental lattice and mingen extraction. Although the overall performances of the later method has proven superior to the other two, when only mingen construction cost is considered, merging is as good as one-by-one incremental reconstruction. This fact rises the hopes that with an efficient lattice merge technique, e.g., one that takes advantage of effective decentralization of both data and computation resources, there can be a significant improvement in the speed of the global method computing both concepts and mingens. Such efficient methods have already been designed for lattice merge upon context apposition, i.e., in the dual case of the one considered here, in both sequential [14] and parallel [7] algorithmic settings.

Acknowledgments

This research was supported by the authors' individual NSERC grants as well as by the FQRNT team grant.

References

[1] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*. North Holland, Amsterdam, 1989.
[2] A. Le Floc'h, C.Fisette, R. Missaoui, P. Valtchev, and R. Godin. Jen : un algorithme efficace de construction de générateurs pour l'identification des règles d'association. *numéro spécial de la revue des Nouvelles Technologies de l'Information*, 1(1):135–146, 2003.
[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] Jean François Djoufak Kengue, Petko Valtchev, and Clémentin Tayou Djamegni. A parallel algorithm for lattice construction. In B. Ganter and R. Godin, editors, *roceedings of the 3rd Intl. Conference on Formal Concept Analysis (ICFCA'05), Lens (FR), (14-18 February) 2005*, pages 248–263, 2005.

[8] Marzena Kryszkiewicz and Marcin Gajek. Concise representation of frequent patterns based on generalized disjunction-free generators. In *PAKDD*, pages 159–171, 2002.

[9] D. Maier. *The theory of Relational Databases*. Computer Science Press, 1983.

[10] H. Mannila and K.-J. Räihä. On the complexity of inferring functional dependencies. *Discrete Applied Mathematics*, 40(2):237–243, 1992.

[11] N. Pasquier. *Data Mining : Algorithmes d'extraction et de réduction des règles d'association dans les bases de données*. Ph. d. thesis, Université Blaise Pascal,Clermont-Ferrand II, 2000.

[12] J. Pfaltz and C. Taylor. Scientific discovery through iterative transformations of concept lattices. In *Proceedings of the 1st International Workshop on Discrete Mathematics and Data Mining*, pages 65–74, Washington (DC), USA, April 2002.

[13] 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.

[14] P. Valtchev and V. Duquenne. Towards scalable divide-and-conquer methods for computing concepts and implications. In E. SanJuan, A. Berry, A. Sigayret, and A. Napoli, editors, *Proceedings of the 4th Intl. Conference Journées de l'Informatique Messine (JIM'03): Knowledge Discovery and Discrete Mathematics, Metz (FR), 3-6 September*, pages 3–14. INRIA, 2003.

[15] 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.

[16] 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.

[17] 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.

[18] P. Valtchev, R. Missaoui, and P. Lebrun. A partition-based approach towards building Galois (concept) lattices. *Discrete Mathematics*, 256(3):801–829, 2002.

[19] P. Valtchev, R. Missaoui, M. Rouane-Hacene, and R. Godin. Incremental maintenance of association rule bases. In *Proceedings of the 2nd Workshop on Discrete Mathematics and Data Mining*, San Francisco (CA), USA, May 2003.