# The Use of Associative Concepts for Fast Incremental Concept Formation in Sparse Contexts*

Sébastien Ferré

Computer Science Dept., UWA, Penglais, Aberystwyth SY23 3DB, UK
sbf@aber.ac.uk

**Abstract** Formal Concept Analysis (FCA) is interested in the formation of concept lattices from binary relations between objects and attributes, a.k.a. contexts. Many algorithms have been proposed to generate the set of all concepts, and also the edges of the lattice between these concepts. We develop the principle and the code of a new algorithm combining two existing ones, Godin's and Bordat's algorithms. Then, we show by both a theoretical and practical study that it is the most efficient algorithm for sparse contexts, which are often found in real applications. In the case where the number of attributes per object is bounded, the complexity of computing the set of all concepts and edges with our algorithm is $O((|\mathcal{O}| + |\mathcal{A}|).|L|)$, or equivalently $O((|\mathcal{O}| + |\mathcal{A}|).|\mathcal{O}|)$.

## 1 Introduction

Formal Concept Analysis (FCA, [GW99]) is interested in the formation of concept lattices from binary relations between objects and attributes, a.k.a. contexts. These concept lattices have been found useful in many domains, such as software engineering [Sne98,ST00], information retrieval [GMA93,Lin95,CS00,FR01], and knowledge extraction [HSWW00]. Several algorithms have been designed to compute the set of concepts or even the Hasse diagram of the concept lattice. A comparative study, both theoretical and practical, can be found in [KO02].

Among these algorithms, some are *incremental* [GMA95,VM01,NR02], which means they can update the concept lattice when a new object is added to the context, without re-computing the whole lattice. On the contrary, *batch* algorithms have to know the whole context before computing the concept lattice; if the context changes, the concept lattice must be re-computed entirely. Incremental algorithms are especially interesting in information systems where the set of objects is growing all the time.

In previous work [FR02], we designed an *incremental learning* process that helps users in describing/classifying new objects according to the description/classification of existing objects. We also established a connection between

* This work was achieved at Irisa (Rennes, France), and funded by a scholarship from CNRS and Région Bretagne.

this incremental learning and the incremental concept formation, which let us think that a more efficient algorithm could be designed for incremental concept formation.

Section 2 explains the above connection and outlines the principle of our algorithm. Section 3 details this algorithm, along with data structures. Section 4 evaluates the theoretical complexity of our algorithm and of two other well-known efficient algorithms (Godin's and Bordat's) to compare them under similar conditions. This proves our algorithm is the most efficient when the number of attributes per object is bounded (which is often observed in practice). It also discusses complexity for different kinds of contexts. Section 5 performs practical experiments with these three algorithms in order to complete the theoretical comparison. Finally, Section 6 summarises our results and draws a few conclusions. Proofs of theorems are not given in this paper, but can be found in [Fer02]

## 2 Principle for a New Incremental Algorithm

In this section, we compare incremental concept formation [GMA95,VM01,NR02] with incremental learning [FR02]. Both methods are based on the search for specific concepts: old, modified, generator, and new concepts in the first case (see Section 2.1); associative concepts in the second case (see Section 2.2). Surprisingly, we found a close relationship between associative concepts and both modified and new concepts (see Section 2.3).

### 2.1 Incremental Concept Formation

Incremental Concept Formation (ICF) starts with a context $K = (\mathcal{O}, \mathcal{A}, I)$, whose concept lattice $L(K)$ had already been computed. The problem is to compute the lattice $L(K^*)$, where $K^*$ is the result of adding a new object $o^*$ described by the set of attributes $D(o^*)$:

$$K^* = (\mathcal{O} \uplus \{o^*\}, \mathcal{A} \cup D(o^*), I \uplus \{(o^*, a) \mid a \in D(o^*)\}).$$

The principle is that every concept intent of $L(K)$ is still a concept intent of $L(K^*)$; and every new concept intent is the intersection of some existing concept intent and $D(o^*)$. Moreover, the new object $o^*$ may be added to the extent of some existing concepts. Therefore, concepts of $L(K)$ can be classified into 3 categories [GMA95]:

**generator concepts** : the intersection of their intent with $D(o^*)$ results in a new intent, and so, in a **new concept**;
**modified concepts** : their intent is included in $D(o^*)$, and so, $o^*$ must be added to their extent;
**old concepts** : all other concepts, which are kept unchanged.

We now give a formal definition for generator and new, modified, and old concepts (where $(\sigma_K, \tau_K)$ is the usual Galois connection over a context $K$, and $int(L(K))$ denotes the set of all intents).

**Definition 1 (generator and new concepts)** *The* new *concepts are those whose intent is the intersection of the intent of an existing concept (a* generator *concept) and of the description $D(o^*)$ of the new object, and does not already exist:*
$N(o^*) = \{(ext, int \cap D(o^*)) \mid \exists (ext, int) \in L(K) : (int \cap D(o^*)) \notin int(L(K))\}$.

Note that concepts in $N(o^*)$ are nearly concepts in $K^*$, but not exactly: they lack $o^*$ in their extent. This notation will appear useful in Section 2.3.

**Definition 2 (modified concepts)** *The* modified *concepts are those whose intent is included in description $D(o^*)$:*
$M(o^*) = \{(ext, int) \in L(K) \mid D(o^*) \supseteq int\}$.

**Definition 3 (old concepts)** *The* old *concepts are those that are neither generator, nor modified concepts.*

The update of a concept lattice $L(K)$ for inserting a new object $o^*$ consists of inserting all new concepts, and of adding $o^*$ to the extension of all modified and new concepts (see Section 2.4 for more details). Therefore, the main task of incremental concept formation is to find all modified and new concepts. In most efficient existing algorithms, the complexity of building the whole concept lattice is $O(|\mathcal{O}|.(|\mathcal{O}| + |\mathcal{A}|).|L|)$ [VM01,NR02].

## 2.2 Incremental Learning

Incremental Learning (IL) starts with the same data as ICF, but its aim is quite different. The aim of IL is to induce a set of attributes $Ind(o^*)$ that seem fit to the new object, according to the description of existing objects. Then, these induced attributes can be suggested to the user, or automatically added to the description $D(o^*)$ of the new object. The definition of $Ind(o^*)$ is based on hypotheses [GK00] according to the JSM method [Fin83]. There also exists a characterisation based on the notion of *associative* concepts [FR02].

**Definition 4 (sub-context)** *Let $K = (\mathcal{O}, \mathcal{A}, I)$ be a context, and $A \subseteq \mathcal{A}$ be a subset of attributes. We define the* sub-context *of $K$ restricted to the subset of attributes $A$, the context*

$$K(A) = (\mathcal{O}, A, I \cap (\mathcal{O} \times A)).$$

**Definition 5 (associative concept)** *A non-empty concept (i.e., whose extent is not empty) of the sub-context $K(\mathcal{A} \cap D(o^*))$ is called an* associative concept *of $o^*$ in $K$. The set of all such associative concepts is denoted by $AC(o^*)$.*

$AC(o^*)$ organizes the context $K$ in a concept lattice (where the empty concept is missing) that is less finely detailed than $L(K)$. However, this coarser concept lattice is relevant to the attributes of $o^*$. Conversely, the finer details in $L(K)$ cannot be expressed with the attributes of $o^*$.

Then, an *induced feature* can be defined as an attribute that contextually subsumes the intent of some associative concepts.

**Definition 6 (induced property)** *We say an attribute $x$ is an induced property iff there exists an associative concept $c \in AC(o^*)$, such that $ext(c) \subseteq \tau_K(x)$. $Ind(o^*)$ denotes the set of all induced properties of $o^*$ in $K$.*

**Theorem 1** $Ind(o^*) = \bigcup_{(ext,int) \in AC(o^*)} \sigma_K(ext)$.

Intuitively, an associative concept $c$ of a new object $o^*$ is an already existing concept (for previous objects in $K$) that has some similarity with the description of $o^*$. When $x \in Ind(o^*)$ is induced from an associative concept $c$, $ext(c)$ is the *support* of the induction, and $int(c)$ is the *explanation*. A given associative concept can induce several attributes; and a given attribute can be induced by several associative concepts, and so, have several explanations. Induced attributes that do not belong to the description $D(o^*)$ are called *expected features*, which are the features suggested to users.

### 2.3 Connection between Incremental Concept Formation and Incremental Learning

The connection between ICF and IL is that associative concepts exactly match new and modified concepts, with the exception of the empty concept. This connection is stated by the two following theorems.

Theorem 2 states that non-empty modified concepts are the associative concepts such that their intent is equal to the closure of the extent by $\sigma_K$, i.e. exists in $L(K)$.

**Theorem 2** $c \in M(o^*)$ *and* $ext(c) \neq \emptyset$ *iff* $c \in AC(o^*)$ *and* $\sigma_K(ext(c)) = int(c)$.

Theorem 3 states that non-empty new concepts are the associative concepts such that the intent is not equal to the closure of the extent by $\sigma_K$, i.e. does not exist in $L(K)$.

**Theorem 3** $c \in N(o^*)$ *and* $ext(c) \neq \emptyset$ *iff* $c \in AC(o^*)$ *and* $\sigma_K(ext(c)) \neq int(c)$.

To summarize, the non-empty modified and new concepts are exactly the associative concepts. There is an empty modified concept when $D(o^*) \supseteq \mathcal{A}$ and it is $(\emptyset, \mathcal{A})$. There is an empty new concept when $\tau_K(D(o^*)) = \emptyset$ and it is $(\emptyset, D(o^*))$. So, it is sufficient to traverse the concept lattice of $K(D(o^*))$ instead of the whole lattice $L(K)$, because of the definition of associative concepts (Def. 5). Notice we avoid the case where two generator concepts are found but only one new concept is generated. Moreover, the upper covers of some new concept in $L(K^*)$ are all in $AC(o^*)$, because every concept whose intent is included in the intent of an associative concept is also an associative concept (by Definition 5). This implies that upper covers of new concepts can easily be found through the traversal of associative concepts. All this suggests that the incremental concept formation proposed by Godin et al. may be improved by the use of associative concepts.

### 2.4 Informal Description of a New Algorithm

We now informally describe the new algorithm we propose. The idea is to generate all associative concepts $AC(o^*)$, and to test for each of them whether it is modified or new. If it is modified, i.e. if it already exists in $L(K)$, then the new object $o^*$ must be added to its extent. If it is new, i.e. if it does not exist in $L(K)$, the new object must be added to its extent, and it must be inserted in the concept lattice. Then, if no concept has $D(o^*)$ as intent, a concept $(\{o^*\}, D(o^*))$ must also be inserted. Finally, when all objects have been inserted, and if no concept has $\mathcal{A}$ as intent, a concept $(\emptyset, \mathcal{A})$ must also be inserted (bottom concept).

To build the Hasse diagram, the upper and lower covers of each new concept $n$ must be determined: (1) the upper covers of $n$ have their intent included in $int(n)$, and so are directly found during the traversal of $AC(o^*)$; (2) the lower covers of $n$ are known [NR02] to be the greatest generator (computable as the closure of $ext(n)$ in $K$), and possibly smaller new concepts, which can also be found during the traversal of $AC(o^*)$.

Compared to Godin's algorithm, we replace the traversal of the whole concept lattice $L(K)$ by the generation of the coarser concept lattice $L_{K(D(o^*))}$, even for the building of the Hasse diagram. The advantage of our approach is that the set of attributes considered is restricted to $D(o^*)$, the attributes of the new object, which in practice often has a bounded size that does not depend on the number of objects and attributes. The drawback is that we have to generate associative concepts, whereas Godin directly reuses the concept lattice already computed.

So, in order to improve on other algorithms, we should find the most appropriate algorithm to generate associative concepts $AC(o^*)$, that is, the concept lattice of $K(D(o^*))$. Because we noted before that the size of $D(o^*)$ is often bounded in practice, we should prefer the best algorithm when the number of attributes is considered to be constant. From a survey on concept lattice algorithms [KO02], the good choice seems to be Bordat's algorithm [Bor86] because it is only linear in the number of objects, whereas other algorithms are quadratic. This is a batch top-down algorithm. It starts from the top concept, i.e., whose extent contains all objects, and successively builds sub-concepts, while checking if a built concept had already been found.

## 3 A New Incremental Algorithm for Concept Formation

In this section, we detail data structures, procedures and functions of our incremental algorithm for the computation of the concept lattice. The global data structures represent the context and its concept lattice. The main procedure, `add-object`, updates these structures with a new object `o`, described by `D`. For efficiency reasons, the bottom concept is not considered in this procedure, but it can be inserted by procedure `add-bottom`, after all objects have been inserted. The lines beginning with a $\diamond$ concern the building of the Hasse diagram of the concept lattice. They can be dropped if one is only interested in the concept set.

Considering that `obj` is the type of objects, and `attr` is the type of attributes, we respectively represent extents and intents by `obj set` and `attr set`. For building the concept set, we use an array `col` to represent columns of the context (i.e., extents of attributes), a set of attributes `top` to represent the intent of the top concept, and a hashtable `C` of concepts indexed by their intents. For building the Hasse diagram, we also use an array `row` to represent rows of the context (i.e., descriptions of objects), and another hashtable `Sub` that associates to each intent of concept the intents of their sub-concepts.

```
◇ data row : array[1..|$\mathcal{O}$|] of attr set = [∅;..;∅]
data col : array[1..|$\mathcal{A}$|] of obj set = [∅;..;∅]
data top : attr set = ∅
data C : hashtable[attr set] of obj set = ∅
◇ data Sub : hashtable[attr set] of attr set set = ∅
```

`col[a]` denotes the context column of attribute `a`. `C[Int]?` is true if a pair `(Ext,Int)` belongs to the hashtable `C`; and `C[Int]` denotes the extent associated to intent `Int` (when `C[Int]?` is true). In the following procedures, $A \leftarrow B$ denotes the assignment of $B$ to $A$.

We now give the procedure `add-object`, which adds object `o` with description `D`. It first calls procedure `iter-assoc` to perform the generation and traversal of associative concepts, during which modified concepts are updated and new concepts are inserted. If `D` is not yet defined in `C`, the concept `({o},D)` is also inserted (see Section 2.4). Finally, a row is added and columns are updated.

```
procedure add-object(o: obj; D: attr set)
    iter-assoc(o,D);
    if not C[D]? then
        C ← C ∪ {({o},D)};
        ◇ Sub ← Sub ∪ {(D,∅)};
    ◇ row[o] ← D;
    for all a ∈ D do
        col[a] ← col[a] ∪ {o}.
```

Procedure `iter-assoc` initiates the generation of associative concepts of the new object `o`; procedure `iter-assoc-2` is recursive and completes this generation. This generation is inspired from the algorithm of Bordat. In `iter-assoc-2`, `Incr` is a selection of those attributes of $D$ that restrict the current extent `Ext` without making it empty; and $S$ is the set of sub-concepts built with the maximal elements of `Incr` by function `sub-nodes`. Function `insert-sub` updates a set of edges to sub-concept by adding a new edge and removing redundant edges.

```
procedure iter-assoc(o: obj; D: attr set)
    if C[top]? then /* true for every object except the first one*/
        Int ← D ∩ top;
        iter-assoc-2(o, D, true, C[top], Int);
    else /* true for first object only */
        iter-assoc-2(o, D, true, ∅, D).
```

```
procedure iter-assoc-2(o: obj; D: attr set; is-top: bool;
          Ext: obj set; Int: attr set)
   if C[Int]? then /* (Ext,Int) is a modified concept */
      C[Int] ← Ext ∪ {o};
   else /* (Ext,Int) is a new concept */
      C ← C ∪ {(Ext ∪ {o},Int)};
      ◇ if Ext = ∅ then /* true for first object only */
      ◇    Sub ← Sub ∪ {(Int,∅)};
      ◇ else /* the generator $\sigma_K$(Ext) is a sub-concept */
      ◇    Sub ← Sub ∪ {(Int, {$\bigcap_{o' \in Ext}$row[o']})};
      if is-top then top ← Int; /* new top concept */
   Incr ← {(Ext ∩ col[x], x) | x ∈ D, Ext ∩ col[x] ≠ ∅};
   if Incr = ∅ then /* no more associative concepts */
   ◇    if Int ≠ D then Sub[Int] ← insert-sub(D, Sub[Int]);
   else
      S ← sub-nodes(Incr, Int);
      for all (Ext',Int') ∈ S do
         if not C[Int]? or o ∉ C[Int] then
            iter-assoc-2(o, D, false, Ext', Int');
         ◇ Sub[Int] ← insert-sub(Int', Sub[Int]).
◇ function insert-sub(Int: attr set; Sub: attr set set)
   : attr set set
   ◇ return {Int' ∈ Sub | Int' ⊈ Int} ∪ {Int}
function sub-nodes(Incr: (obj set, attr) set; Int: attr set)
   : (obj set, attr set) set
   A ← Incr; S ← ∅;
   for all (Ext,a) ∈ Incr do
      A ← A \ {(Ext,a)}; max ← true; Equal ← {a};
      for all (Ext',a') ∈ A do
         if Ext ⊊ Ext' then max ← false; Stop
         if Ext = Ext' then
            A ← A \ {(Ext',a')}; Equal ← Equal ∪ {a'};
         else /* Ext ⊋ Ext' */
            A ← A \ {(Ext',a')};
      if max = true then S ← S ∪ {(Ext,Int ∪ Equal)};
   return S.
```

The bottom concept has the set of all attributes as an intent. If this intent is not defined in C, procedure add-bottom adds the concept $(\emptyset,\mathcal{A})$, and completes the linking of the Hasse diagram if necessary.

```
procedure add-bottom($\mathcal{A}$ : attr set)
   if not C[$\mathcal{A}$]? then
      C ← C ∪ {(∅,$\mathcal{A}$)};
      ◇ Sub ← Sub ∪ {($\mathcal{A}$,∅)};
      ◇ for all (Int,S) ∈ Sub do
      ◇    if S = ∅ then Sub[Int] ← {$\mathcal{A}$}.
```

This algorithm was implemented in Objective Caml, a programming language that combines functional, imperative and objective features. Attribute sets and object sets are represented by ordered lists, which makes all set operations linear with the size of sets. The hash function on intents is considered as linear in the size of intents, which determines the cost of operations on hashtables.

## 4 Discussion on Theoretical Complexity

In this section, we discuss the theoretical complexity of our algorithm for computing the Hasse diagram of the concept lattice, and compare it with the algorithms of Bordat and Godin. To express theoretical complexities, we retain 3 parameters that determine the size of contexts:

– $n$: the number of objects;
– $m$: the number of attributes;
– $k$: the maximal number of attributes per object.

This last parameter is not usually used in theoretical complexities, but is often put forward to say that the size of the concept lattice is polynomial with the number of objects $n$, when $k$ is bounded. Indeed in this case, the number of concepts, which we denote by $l$, is $O(n2^k) = O(n)$: each object has at most $2^k$ concepts above it (maximal number of subsets of its description). Even if in general the number of concepts may be exponential in the number of objects or the number of attributes, we think it is preferrable to express theoretical complexities with parameter $k$. This makes them more precise.

### 4.1 Bordat's and Godin's algorithms

So, to achieve fair comparisons with other algorithms, we must first reconsider their complexity with regard to parameter $k$. For instance, Bordat's algorithm is known to have complexity $O(nm^2l)$. Should we replace the occurence of $m$ (number of attributes) by an occurence of $k$ (number of attributes per object)? Here, the answer is no because $m$ corresponds to the traversing of all attributes. As for Godin's algorithm, which has complexity $O(n(n+m)l)$, should we replace $m$ by $k$? Here, the answer is yes because this occurence of $m$ corresponds to set operations on intents, whose size is bounded by $k$. So, we now consider that Godin's has complexity $O(n(n+k)l)$.

### 4.2 Our algorithm

In our algorithm, either `iter-assoc` or `iter-assoc-2` is called on all associative concepts generated through the insertion of all objects. By a close examination of these two procedures, we observe that `iter-assoc-2` has a greater complexity than `iter-assoc`. The complexity of the former is determined by the complexity of functions `sub-nodes` and `insert-sub` (called for (at worst) each attribute of the new object):

`sub-nodes` pairwise compares the extent of the attributes of the new object: its (worst-case) complexity is therefore $O(k^2n)$;
`insert-sub` compares a given intent with the intents of a set of sub-concepts. As the number of sub-concepts is bounded by $m$, its complexity is $O(km)$.

Therefore, the complexity of `iter-assoc-2` is $O(k^2(n+m))$. So, the whole theoretical complexity of our algorithm is $O(k^2(n+m)a)$, where $a$ denotes the whole number of associative concepts through the insertion of all objects. As the number of associative concepts $AC(o^*)$ for every new object $o^*$ is bounded by the number of subsets of its description $D(o^*)$, that is $2^k$, we have $a \leq 2^kn$. So, our algorithm has complexity $O(k^2(n+m).2^kn)$.

To compare this complexity with other algorithms, we must relate in some way parameter $a$ (or term $2^kn$) to parameter $l$. Firstly, if we state the reasonable assumption that different objects have different descriptions, the inequality $n \leq l$ is satisfied because every object description is a concept intent. Otherwise, duplicate objects can easily be identified (their description is already an intent), and inserted (add the new object to every extent containing the former object). In conclusion, we can safely replace $a$ by the term $2^kl$. Secondly, $a$ is also equal to the number of object insertions in concept extents (see `iter-assoc-2`), and so, is equal to the sum of the size of all concept extents, which is $O(nl)$. So, we can also safely replace $a$ by the term $nl$. Finally, we can give to our algorithm a complexity $O(k^2(n+m).min(2^k,n)l)$.

## 4.3 Comparisons

To compare Bordat's, Godin's, and our algorithms, we consider 4 cases:

- the general case, where complexity is expressed with $k$, $n$, and $m$;
- the case where $k$ may be as big as $m$ (i.e., $k=m$);
- the case where $k$ is a constant independent of $n$ and $m$;
- the case where $k$ is a constant, and $m$ grows like $n$ (i.e., $m=\alpha n$).

In practice [FR01], we have observed that, while the number of objects is growing, the number of attributes per object remains bounded, which justifies the consideration of $k$ as a constant in the 3rd and 4th cases. Of course, the value of $k$ strongly depends on the considered application. Moreover, we have also observed that often, each object brings with it a few new attributes, such that the number of attributes grows in proportion to the number of objects: this justifies the consideration of $m$ as equal to $n$ in the 4th case. For example, in a bibliographic application, we observed that $k \simeq 60$ and $\frac{m}{n} \simeq 10$, for $n$ going up to 5000.

The 4 cases of complexities for the 3 algorithms are summarized in the following table:

| complexity | Bordat | Godin | Ferré (with $l$) | Ferré (without) |
|---|---|---|---|---|
| general case | $nm^2l$ | $n(n+k)l$ | $k^2(n+m).min(2^k,n)l$ | $k^2(n+m).2^kn$ |
| $k \in O(m)$ | $nm^2l$ | $n(n+m)l$ | $m^2(n+m).nl$ | $m^2(n+m).2^mn$ |
| $k \in O(1)$ | $nm^2l$ | $n^2l$ | $(n+m).l$ | $(n+m).n$ |
| $k \in O(1), m \in O(n)$ | $n^3l$ | $n^2l$ | $n.l$ | $n.n$ |

It appears that, while our algorithm is theoretically worse than other ones in non-sparse contexts (i.e., $k \in O(m)$), it is strictly better in sparse contexts. It is better than Bordat's because we exploit the bounded size of intents, and we carefully avoid to traversing the set of all attributes. It is better than Godin's because we replaced the $n$ traversal of the lattice by the traversal of associative concepts, whose number is bounded by $2^k l$, which is in $O(l)$ when $k$ is a constant. Moreover, the term $(m + n)$ in our complexities can be replaced by $n$ if one is interested in the concept set only. This shows that the computation of the Hasse diagram is not costly in our algorithm, which can be explained by the fact that edges are determined through the traversal of associative concepts.

## 5 Practical Complexity

To complete the theoretical study of the complexity of our algorithm compared to Bordat's and Godin's, we present in this section a practical comparison of these algorithms. We can consider the computation of either the set of concepts or the Hasse diagram of concepts, i.e., the set of concepts plus edges between concepts [KO02]. For Bordat's algorithm, there is no substantial difference since the computation of the set of concepts implies the computation of all sub-concepts of each concept. For Godin's most efficient algorithm, the computation of edges is not sufficiently detailed to be coded, unfortunately. So, we compare in this article the three algorithms for computing the set of concepts. However, we also present the pratical complexity of our algorithm when computing the Hasse diagram. Note that this is to the advantage of Bordat's and Godin's, especially for the latter, whose general complexity is then $O(knl)$. Note also that if a better version of Bordat's algorithm were used, this would also benefit us because our algorithm uses it nearly directly. Computation times were obtained on an UltraSparc II, 440 MHz, 390 Mb with Solaris 7.

### 5.1 General contexts

The aim of this sub-section is to study how pratical complexities depend on each of the three parameters $n$, $m$, and $k$. For each valuation of this triple, a formal context is generated randomly with $n$ objects whose descriptions are composed of $k$ out of $m$ attributes. In each of the three following figures, the computation time is shown as a function of the number of objects $n$ (Figure 1), the number of attributes $m$ (Figure 2), and the number of attributes per object $k$ (Figure 3, logarithmic scale for time).

Figure 1 shows that the computation time increases with the number of objects. According to Section 4, and with the fact that the number of concepts is $O(n)$ when $k$ is fixed, the time complexity is $O(n^2)$.

Surprisingly, Figure 2 shows that the computation time decreases with the number of attributes. The explanation is that the number of objects $n$ has been used as an over-approximation of the size of extents, while their average size is

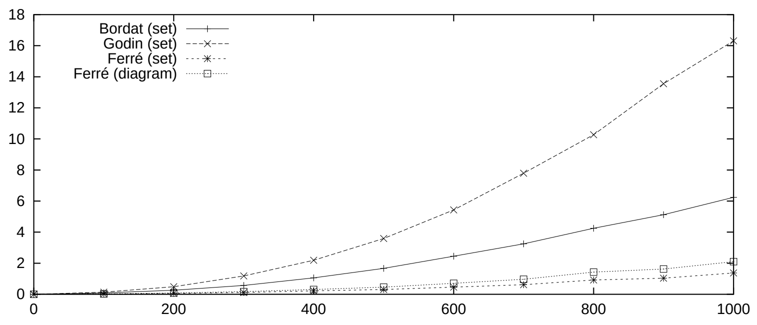

**Figure1.** Time (sec) as a function of $n$: $m = 100$, $k = 6$.

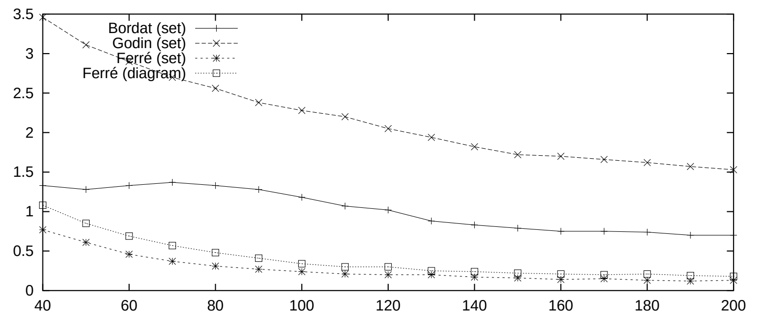

**Figure2.** Time (sec) as a function of $m$: $n = 400$, $k = 6$.

something like $\frac{kn}{m}$. This effect is the most visible for Bordat's and our algorithms because they are intensively based on extent comparisons.

Figure 3 shows that the computation time is exponential with the number of attributes per object. With Figures 1 and 2 showing that this computation time is not exponential with other parameters (i.e., $n$ and $m$), this means that the famous exponential nature of concept lattices and their computation only relies on parameter $k$. This implies that whenever it is fixed (which is often the case in real contexts), the time complexity is polynomial in $n$ with a constant factor depending (exponentially) with $k$.

In conclusion, our algorithm performs better than the others when $k$ is small enough, and this is even true when computing the diagram. However, Godin's algorithm, the worst for small values of $k$, becomes the better above some threshold (here, $k \simeq 17$). Of course, this threshold may depend on other parameters $n$, and $m$. A question that remains is whether the computation of the diagram by Godin's algorithm would be better or worse than ours in this case?

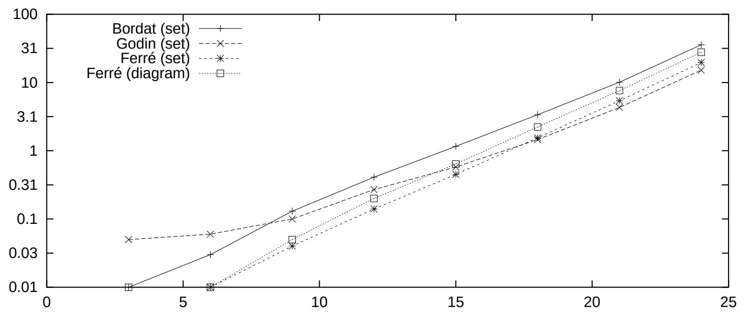

**Figure3.** Time (sec) as a function of $k$: $n = 50$, $m = 50$.

## 5.2 Contranominal contexts

Contranominal contexts are square contexts where only the main diagonal is free. This means that $m = n$ and each object has $n - 1$ attributes. This kind of contexts is not realistic, but it is of theoretical interest because their concept lattices are the power set of objects and attributes, and so have an exponential size. It can be considered to be the worst case for computation.

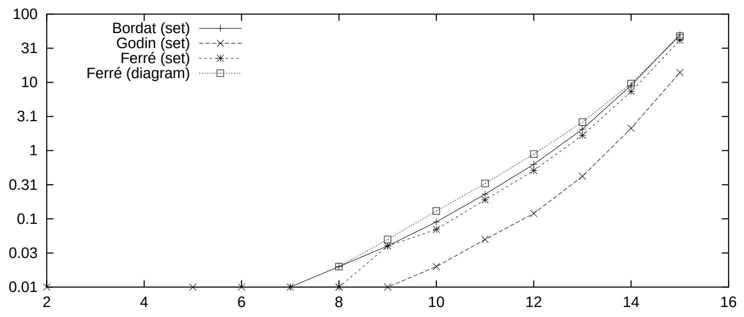

**Figure4.** Time (sec) as a function of $n$: $m = n$, $k = n - 1$ (contranominal context).

Figure 4 shows our algorithm performs worse than Godin's but as well as Bordat's, even when computing the diagram. However, the difference between Godin's and our algorithm is not large and seems to decrease. Once again, the question of how Godin's compares to ours for the diagram remains open.

## 5.3 Logical contexts

In our experiments about logical information systems [FR01], we observed that, while the number of attributes per object is constant in a given application, the number of attributes grows proportionally to the number of objects because each new object brings a few new attributes. This is characteristic of what we call *logical contexts*, where objects are described by a logical formula instead of a set of attributes. There, attributes are *features* automatically extracted from logical descriptions, which leads to the production of new attributes. Another specificity of logical contexts is that some attributes are more general than others: a few are shared by all objects while others are specific to one object. We modeled this phenomena by a Zipf law that associates to each attribute a frequency inversely proportional to its rank in the list of all attributes.

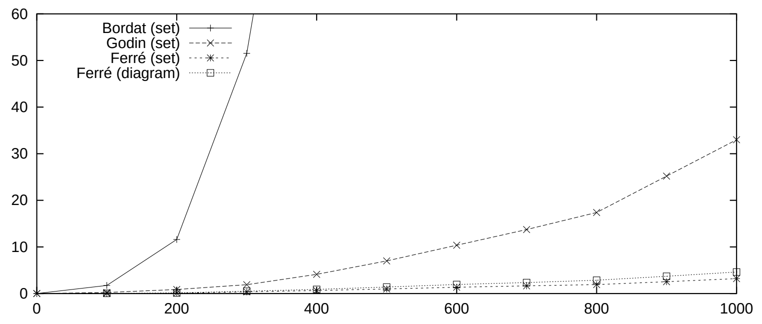

**Figure5.** Time (sec) as a function of $n$: $m = 2 \times n$, $k = 6$ (logical context).

Compared to Figure 1 where $m$ is fixed, Figure 5 shows that our algorithm behaves very well in logical contexts, while Godin's and Bordat's are definitely more costly. These results confirm the theoretical results of Section 4.

## 6 Conclusion and Perspectives

We proposed a new algorithm for computing the concept lattice of a context. It combines existing algorithms from Godin and Bordat. Both theoretical and practical complexity comparisons with these two algorithms prove that our algorithm is the most efficient one in sparse contexts, and the only efficient one in logical contexts. Moreover, it is incremental, which is a necessary condition for some applications.

A future improvement would be to take into account *logical entailments* between attributes, i.e., a taxonomy of attributes, in the generation of sub-concepts (see function `sub-nodes` in Section 3). This would certainly improve the speed of our algorithm, especially in the case of logical contexts.

**Acknowledgements:** I thank Sergei A. Objedkov for insightful discussions that contributed to improve both the algorithm and the complexities study.

## References

[Bor86] J. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Mathématiques, Informatiques et Sciences Humaines*, 24(94):31–47, 1986.

[CS00] R. Cole and G. Stumme. CEM - a conceptual email manager. In G. Mineau and B. Ganter, editors, *Int. Conf. Conceptual Structures*, LNCS 1867, pages 438–452. Springer, 2000.

[Fer02] S. Ferré. Incremental concept formation made more efficient by the use of associative concepts. Research Report RR-4569, Inria, Institut National de Recherche en Informatique et en Automatique, October 2002.

[Fin83] V. K. Finn. On machine-oriented formalization of plausible reasoning in the style of F. Backon–J.S. Mill. *Semiotika Informatika*, 20:35–101, 1983. In Russian.

[FR01] S. Ferré and O. Ridoux. Searching for objects and properties with logical concept analysis. In H. S. Delugach and G. Stumme, editors, *Int. Conf. Conceptual Structures*, LNCS 2120, pages 187–201. Springer, 2001.

[FR02] S. Ferré and O. Ridoux. The use of associative concepts in the incremental building of a logical context. In G. Angelova U. Priss, D. Corbett, editor, *Int. Conf. Conceptual Structures*, LNCS 2393, pages 299–313. Springer, 2002.

[GK00] B. Ganter and S. Kuznetsov. Formalizing hypotheses with concepts. In G. Mineau and B. Ganter, editors, *Int. Conf. Conceptual Structures*, LNCS 1867, pages 342–356. Springer, 2000.

[GMA93] R. Godin, R. Missaoui, and A. April. Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38(5):747–767, 1993.

[GMA95] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer, 1999.

[HSWW00] J. Hereth, G. Stumme, R. Wille, and U. Wille. Conceptual knowledge discovery and data analysis. In G. Mineau and B. Ganter, editors, *Int. Conf. Conceptual Structures*, LNCS 1867, pages 421–437. Springer, 2000.

[KO02] S. O. Kuznetsov and S. A. Objedkov. Comparing performances of algorithms for generating concept lattices. *JETAI: Journal of Experimental & Theoretical Artificial Intelligence*, 14:189–216, 2002.

[Lin95] C. Lindig. Concept-based component retrieval. In *IJCAI95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs*, 1995.

[NR02] L. Nourine and O. Raynaud. A fast incremental algorithm for building lattices. *JETAI: Journal of Experimental & Theoretical Artificial Intelligence*, 14:217–227, 2002.

[Sne98] G. Snelting. Concept analysis — A new framework for program understanding. *ACM SIGPLAN Notices*, 33(7):1–10, July 1998.

[ST00] G. Snelting and F. Tip. Understanding class hierarchies using concept analysis. *ACM Transactions on Programming Languages and Systems*, 22(3):540–582, 2000.

[VM01] P. Valtchev and R. Missaoui. Building concept (Galois) lattices from parts: Generalizing the incremental methods. In H. S. Delugach and G. Stumme, editors, *Int. Conf. Conceptual Structures*, LNCS 2120, pages 290–303. Springer, 2001.