Performances of Galois Sub-hierarchy-building Algorithms

Gabriela Arévalo, Anne Berry, Marianne Huchard, Guillaume Perrot, Alain Sigayret

Performances of Galois Sub-hierarchy-building algorithms

Gabriela Arévalo[1,3], Anne Berry[2], Marianne Huchard[1],
Guillaume Perrot[1], and Alain Sigayret[2]

[1] LIRMM - CNRS UMR 5506 - Université de Montpellier II - Montpellier (France)
{huchard, arevalo, perrot}@lirmm.fr
[2] LIMOS - CNRS UMR 6158 - Univ. Blaise Pascal - Clermont-Ferrand II (France)
{berry, sigayret}@isima.fr
[3] LIFIA - Facultad de Informática (UNLP) - La Plata (Argentina)
garevalo@sol.info.unlp.edu.ar

Abstract. The Galois Sub-hierarchy (GSH) is a polynomial-size representation of a concept lattice which has been applied to several fields, such as software engineering and linguistics.
In this paper, we analyze the performances, in terms of computation time, of three GSH-building algorithms with very different algorithmic strategies: Ares, Ceres and Pluton. We use Java and C++ as implementation languages and Galicia as our development platform.
Our results show that implementations in C++ are significantly faster, and that in most cases Pluton is the best algorithm.

Keywords: Galois Sub-hierarchy, AOC-Poset, Performance Analysis.

1 Introduction

Formal concept analysis (FCA) has been used in a broad spectrum of research fields, such as knowledge representation, data mining, machine learning, software engineering and databases. The main drawback of concept lattices is that the number of concepts may be of much larger size than the relation (or even exponential in the size of the relation). It is therefore feasible, when this problem is encountered, to use a polynomial-size representation of the lattice while preserving the most relevant information.

One of the approaches, which has proved useful in practice, is to restrict the lattice to the concepts which introduce a new object or a new property. This idea is the basis for two very similar structures called the *Galois Sub-hierarchy (GSH)* and the *Attribute Object Concept poset (AOC-poset)*. The Galois Sub-hierarchy has been introduced in the software engineering field by Godin et al. [GM93] for class hierarchy reconstruction and successfully applied in later research work [GMM+98], [AYLCB96], [HDL00], [DHL+02]. The AOC-poset

has been used in applications of FCA to non-monotonic reasoning and domain theory [Hit04] and to produce classifications from linguistic data [OP02], [Pet01].

These structures are interesting not only as a feasible alternative to oversized concept lattices, but also as a conceptual improvement, as human perception of a problem is enhanced by an easy visualization of a restricted number of elements.

As the size of the input may still be large, naturally it is important to have efficient Galois Sub-hierarchy-building algorithms to work with. There are several efficient Galois Sub-hierarchy-building algorithms, with very different algorithmic strategies, and with theoretical worst-time complexity analyses which are difficult to compare. Kuznetsov et al. [KO02] propose a rather extensive implementative comparative analysis of lattice-building algorithms, but to our knowledge the only existing work on comparing algorithms related to GSH-building algorithms is proposed by Godin et al. [GC99], comparing ARES and ISGOOD, which is restrictive, as it builds only the attribute elements of the Galois Sub-hierarchy.

In this paper we address the issue of comparing the execution times of the three main Galois Sub-hierarchy-building algorithms: ARES, CERES and PLUTON, in order to determine which algorithm can be recommended to a user and in which case. This choice is meaningful because these three algorithms are used as tools with a strong user-based interaction, where the response time is a very important factor. The performance factors we tested are the density of the relation and the number of objects and attributes.

The paper is structured as follows: Section 2 introduces the main terminology of Galois Sub-hierarchy. Section 3 explains how the three Galois Sub-hierarchy-building algorithms work. Section 4 details the experimental approach which we used. Section 5 presents our evaluation of the results. We conclude in Section 6.

2 Notations and definitions

In this section, we introduce the main terminology necessary to understanding how the Galois Sub-hierarchy algorithms work. We do not explain in detail the basics of FCA features but focus more on Galois Sub-hierarchy definitions. We refer the reader to Ganter et al. [GW99] for a complete introduction to partial orders and lattices.

In FCA, a formal context is a triple $\mathbb{K} = (G, M, I)$ where G and M are sets (objects and attributes respectively) and I is a binary relation, i.e., $I \subseteq G \times M$. Figure 1(left) shows context $\mathbb{K} = (\{1, 2, 3, 4, 5, 6\}, \{a, b, c, d, e, f, g, h\}, I)$.

For a set $A \subseteq G$ of objects, we define $A' := \{m \in M | gIm \text{ for all } g \in A\}$ (the set of attributes common to the objects in A). Correspondingly, for a set $B \subseteq M$, we define $B' := \{g \in G | gIm \text{ for all } m \in B\}$ (the set of objects which have all attributes in B). Then, a formal concept of the context (G, M, I) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. A is called the *extent* and B the *intent* of the concept (A, B). $\underline{\mathfrak{B}}(G, M, I)$ denotes the set of all concepts of the context (G, M, I). Figure 1 (right) shows the concept lattice corresponding to our example.

	a	b	c	d	e	f	g	h
1		×	×	×	×			
2	×	×	×				×	×
3	×	×				×	×	×
4				×	×			
5			×	×				
6	×							×

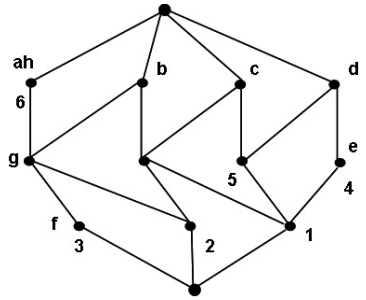

Fig. 1. Left: Binary relation of a context $\mathbb{K}$ - Right: Concept lattice $\underline{\mathfrak{B}}(G, M, I)$

The concepts $C^O = \{\gamma o = (o'', o') | o \in G\}$ are called the *object concepts* of o, and the concepts $C^A = \{\mu a = (a', a'') | a \in A\}$ are called the *attribute concepts*. The object concept which corresponds to object o, γo, is the smallest concept with o in its extent, and dually, the attribute concept which corresponds to attribute a, μa, is the greatest concept with a in its intent. The *Galois Sub-hierarchy* is the sub-order of the lattice made out of the set $C^O \cup C^A$ and the restriction of the lattice order to that set [HDL00]. Figure 1 (right) shows the lattice corresponding to context $\mathbb{K}$. Figure 2 (left) shows the Galois Sub-hierarchy of the context introduced in Figure 1. As we see, in the Hasse diagram of a Galois Sub-hierarchy, empty concepts are omitted. Thus, in the paper we use the notation (4,e) instead of (14,de), and use the terms *simplified intent* (for (4)) and *simplified extent* (for (e)), as well as *simplified concept* (for $(4, e)$) as shown in Figure 2.

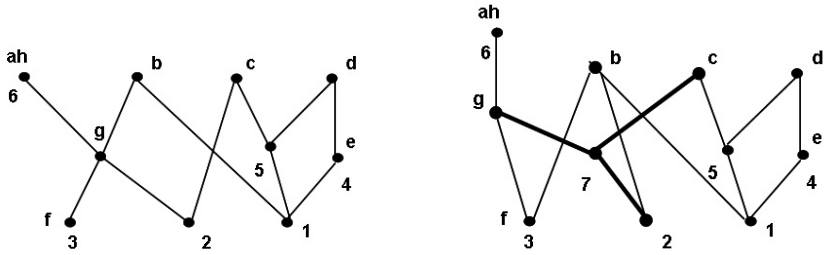

Fig. 2. Left: Galois Sub-hierarchy GSH(I). (b) denotes the concept (123,b) or its simplified form ($\emptyset$,b) - Right: Galois Sub-hierarchy after addition of object 7 (Refer to Section 3 in ARES algorithm)

3 The Algorithms

This section briefly explains the basic features of the Galois Sub-hierarchy-building algorithms which we analyze in this paper. The reader is referred to the cited papers for further details on the corresponding algorithms. We will use the example from Figure 1 to illustrate our explanations.

PLUTON

PLUTON is composed of three algorithms: TOMTHUMB, TOLINEXT, TOGSH. TOMTHUMB produces an ordered list of the simplified extents and intents in Berry et al. [BHM+05]. This ordered list maps to a linear extension of the Galois Sub-hierarchy. Algorithm TOLINEXT then searches the ordered list to merge pairs consisting of a simplified extent and a simplified intent pertaining to the same concept, in order to reconstruct the elements of the Galois Sub-hierarchy. Algorithm TOGSH is then used to compute the edges of the Hasse diagram (transitive reduction) of the Galois Sub-hierarchy.

Algorithm TomThumb uses a sub-algorithm which computes either an ordered partition of objects into simplified extents, or dually a partition of attributes into simplified intents. The order on the simplified closed sets (simplified extents or intents) maps to a linear extension of the Galois Sub-hierarchy.

Algorithm TOMTHUMB uses partition refinement to construct ordered partitions into simplified intents and extents, by using a list of attributes to partition a list of objects (or vice-versa). If for example the list of attributes (a, b, c, d, e, f, g, h) is used to refine the class of objects (123456), the first step using attribute a will split class (123456) into the two classes $(236, 145)$, as $a' = \{2, 3, 6\}$. The process is then repeatedly applied by the next attribute to each of the current classes. Berry et al. [BHM+05] give full details as well as a detailed example.

Algorithm TOMTHUMB proceeds in three steps:

- Computation of an ordered partition $\mathcal{L}_e$ of the simplified extents, using any ordering of the properties as input.
  For example, using the ordering (a, b, c, d, e, f, g, h) of the attributes, the output is $\mathcal{L}_e$ = {(2),(3),(6),(1),(5),(4)}
- Computation of an ordered partition $\mathcal{L}_i$ of the simplified intents, using $\mathcal{L}_e$ as input. In our example, using $\mathcal{L}_e$ = {(2),(3),(6),(1),(5),(4)} as input, the output is $\mathcal{L}_i$ = {(d),(e),(c),(b),(ah),(g),(f)}
- The two partitions are then merged to produce a list of simplified closed sets which can be mapped to a linear extension of the Galois Sub-hierarchy, *e.g.* LIST = {(2),(3),(f),(g),(6),(ah),(1),(b),(5),(c),(4),(e),(d)}, which represents the linear extension {(2),(3,f),(g),(6,ah),(1),(b),(5),(c),(4,e),(d)} of the Galois Sub-hierarchy.

Algorithm ToLinext assembles simplified (non-empty) extent E_r and simplified intent I_r pertaining to a same concept, that is such that for complete extent and intent, $E_r = (I_r)'$. Only pairs formed by an extent directly followed by an intent need be considered. For a simplified extent E_r = LIST$[i]$, we check that: $E = (\text{LIST}[i])' = I = (\text{LIST}[i+1])''$. In our example, the result is:
L = {(2),(3,f),(g),(6,ah),(1),(b),(5),(c),(4,e),(d)}, but to apply the algorithm TOGSH we consider a form of L where simplified empty sets are added:
L = {(2,∅),(3,f),(∅,g),(6,ah),(1,∅),(∅,b),(5,∅),(∅,c),(4,e),(∅,d)}.

Algorithm ToGSH builds the Hasse diagram of the Galois Sub-hierarchy by computing the edges of the graph. The ordering into an linear extension L is used to reduce the number of comparisons, as by definition of a linear extension, an edge can only go from a concept (for example (2,$\emptyset$) to a concept which appears to its right in the list (for example ($\emptyset$,c))). Once an edge is detected, sub-concepts of the origin are marked in order to avoid already visited concepts linked by transitive edges.

Theoretical complexity. In Berry et al. [BHM$^+$05], Tom Thumb's time complexity is analyzed as in $\mathcal{O}(|J|)$. A brute force implementation of ToLinext has a complexity in $\mathcal{O}((|O| + |A|)^3)$. Fura et al. [FLPP05] evaluates the complexity of ToGSH as $\mathcal{O}((|O| + |A|)^2 \times max(|O|, |A|)^2)$. It is worth noting that in the Galicia implementation of Pluton, whole extents and intents are computed in a simple pass of the Galois Sub-hierarchy.

CERES

Ceres mixes the computation of the concepts and that of the Hasse diagram. Concepts are computed respecting an order which maps to a linear extension of the Galois Sub-hierarchy. First, the columns of I are sorted by decreasing number of crosses to generate the concepts of C^A by decreasing extent size. In the example shown in Figure 1, columns could be ordered as follows: a, b, c, d, h, e, g, f. The strategy is then to compute C^A by groups of concepts which have the same extent and adding concepts of $C^O \setminus C^A$ to the GSH under construction, when their intent is covered by the intents of the C^A concepts previously computed. Extents and simplified intents of C^A concepts, as well as closed sets of C^O concepts, are computed using I. Extent inclusion is used to compute edges during a top-down traversal of the current Hasse diagram. Simplified extents and intents of C^O concepts are computed by propagation after edge construction. A simplified execution of the algorithm on our example could be:

- Column size = 3: concepts (6,ah),(b),(c),(d) are generated and included in the Hasse diagram (no edges at this step).
- Concept (5) can be added because attributes c and d have already been found. It is linked to (c) and (d).
- Column size = 2: concepts (4,e) and (g) are generated and linked respectively to (d), and (6,ah), (b).
- Concepts (2) and (1) are added and linked respectively to (g),(c) and (b),(5), (4,e).
- Column size = 1: concept (3,f) is generated and linked to concept (g).

Theoretical complexity The time complexity of Ceres is in $\mathcal{O}(|O| \times (|O| + |A|)^2)$ [Leb00].

ARES

ARES constructs the Galois Sub-hierarchy in an incremental fashion. At each step, it considers the Galois Sub-hierarchy $GSH(I)$ associated with (G, M, I) as well as a new formal object o given with its attribute set $A_o = o'$. The result of the algorithm is the Galois Sub-hierarchy $GSH(I')$ for (G, M', I'), $A' = A \cup \{o\}$, $I' = I \cup \{(o, x) | x \in A_o\}$. The initial GSH is traversed using a linear extension, ensuring that a concept is explored after all its superconcepts. Let us denote by C the current (explored) concept and by RI_o the attribute set which at the end is the simplified intent of the concept introducing o (o is in its simplified extent). Discarding cases such that the intersection between C's intent and o' is empty, four main cases may occur:

- Case 1: C's intent is exactly o'. o is added to C's extent. The Hasse diagram remains unchanged.
- Case 2: C's intent is strictly included in o'. C is or will be a superconcept of o' (the algorithm stores this information). o is added to C's extent. The attributes of C are removed from RI_o.
- Case 3: C's intent includes o'. C is a sub-concept of γo. C either inherits all o's attributes, or some of o's attributes are in C's simplified intent. A new concept C_o with the intent o' is created if needed. C_o is introduced in the Hasse diagram between C and the C's superconcepts which also satisfied Case 2. Intents and extents, as well as their simplified forms, are updated.
- Case 4: C's intent and o' cannot be compared by set inclusion. In this situation, a new concept can be needed to factorize the common attributes not inherited by C. This new concept is introduced in the Hasse diagram between C and the C's superconcepts which also satisfied Case 2. RI_o, intents and extents, as well as their simplified forms, are updated.

If during the exploration, the algorithm did not find an initial concept whose intent is o', it is necessary to create a new concept $C_o = o'$. C_o's extent is o, C_o's simplified intent is RI_o (which is o' minus the attributes found during the GSH exploration). C_o is linked to initial or newly created concepts when their intent is included in o'. In every modification of the Hasse diagram, the algorithm removes transitivity edges as necessary. Meanwhile, when simplified intents are modified, the algorithm checks if for a concept both extent and intent are empty, and if so, the concept is removed.

Let us examine the addition of new object 7 with $A_7 = 7' = \{a, c, g, h\}$. The Galois Sub-hierarchy is traversed by successively analyzing: concept (6,ah) (Case 2), concept (b) (no intersection between intents), (c) (Case 2), (d) (no intersection between intents), (g) (Case 4), a new super-concept of concept (g) and o' is created to factorize the common attribute g, it is attached as a sub-concept of (6,ah) and (c) and as a super-concept of (g) which becomes extent-empty and intent-empty and will be removed at the end), concept (2) (Case 3), concept (3,f) (Case 4, but the common attributes are inherited), etc. The Galois Sub-hierarchy integrating object 7 is shown in Figure 2.

Theoretical complexity. The time complexity of ARES is in $\mathcal{O}(|O| \times |(w \times a + m))$, where w is the width of the Galois Sub-hierarchy (i.e. the maximum number of pairwise non-comparable elements), a is the maximum size for an intent and m the number of edges of the Hasse diagram [DDHL94].

4 Experimental Setup

In this section, we give the parameters we use in our experimentations, and explain our approach.

Parameters used. We have done the experiments using Galicia [Gal]. Galicia is a Java-based platform dedicated to constructing lattices. It offers to FCA researchers advanced tools for performance studies and an open environment to new lattices-related techniques. Galicia is implemented in Java because it ensures a high portability of the entire system. Thus Java was our first choice in the implementation of algorithms ARES, CERES and PLUTON. Because of the first results in performances, we considered C++ as a second choice, as it is known as a language with a good processing speed.

In the rest of the paper, we name as `Ares`, `Ceres` and `Pluton` the algorithms implemented in Java, and `Ares++`, `Ceres++` and `Pluton++` the algorithms implemented in C++. We must remark that all the algorithms were implemented by the same programmer [Per05], so that differences in the implementation style should not be a factor.

Tests: Random Generation of Binary Relations. To perform our experiments we generate a test suite using randomly generated binary relations. Similarly to Kuznetsov et al. [KO02], the binary relations were randomly generated using the following parameters: the number of objects, the numbers of attributes, and the binary relation density defined as follows:

$$\frac{|J|}{|O| \times |A|} \times 100$$

In fact, we use the complexity of the binary relation, defined as follows:

$$\sqrt{\left(\frac{|J|}{|O|}\right)^2 + \left(\frac{|J|}{|A|}\right)^2}$$

which is equal to the density multiplied by $\sqrt{|O|^2 + |A|^2}$ divided by 100.

Test Suite. We generate a test suite considering the variability of the density, the number of objects and the number of attributes, as follows:

- Square binary relations (the same number of objects and attributes) with variable density. In this case, the numbers of objects and attributes are 500 and the density varies from 2 to 82.

– Variable number of objects with fixed number of attributes and density. In this case, the number of attributes is 500, the density is 50 and the number of objects varies from 1000 to 4800 incremented by 200.
– Variable number of attributes with fixed number of objects and density. In this case, the number of objects is 500, the density is 50 and the number of attributes varies from 1000 to 4800 incremented by 200.

Evolutive Approach of Experiments. We developed the experiments in three phases:

– First Phase: We compared the results between `Ares`, `Ceres` and `Pluton` only implemented in Java. In this specific phase, we used `HashSet` and `ArrayList` of the Java language API as our main data structures.
– Second Phase: We compared the results between the implementations in Java (`Ares`, `Ceres` and `Pluton`) and those in C++ (`Ares++`, `Ceres++` and `Pluton++`). In this specific phase, we used `set<>` and `vector<>` of the C++ language API as our main data structures.
– Third Phase: We compared the results between `Ares`, `Ceres`, `Pluton`, `Ares++`, `Ceres++`, `Pluton++` and their dual versions, where we transposed the matrices representing the binary relations, meaning that the rows of the non-dual versions are made into columns in the dual versions, and viceversa.

5 Evaluation and Results [4]

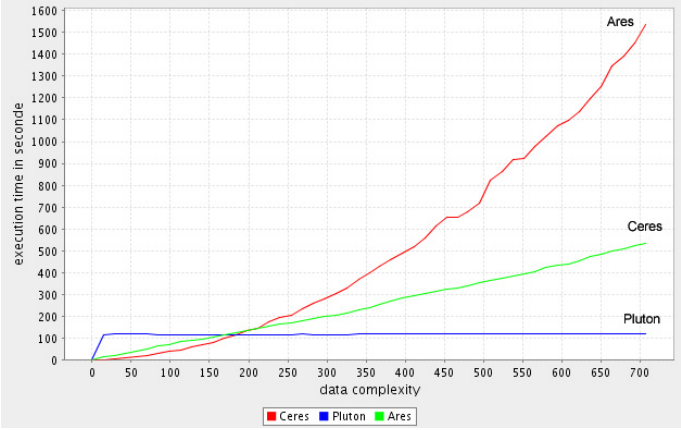

Fig. 3. Test results with a square matrix with variable density

[4] The interested reader can find a colored version of the figures in the original version of the paper in http://www.lirmm.fr/∼huchard/Huchard/pub.frametop.html

5.1 First Phase

In this first phase, we will only consider the Java implementations of the algorithms. Our results show that each algorithm is interesting in its own right for certain input parameters:

- `Pluton` is the best if there is no difference between the number of objects and the number of attributes (shown in Figure 3).
- `Ares` is the best if we vary the number of objects (shown in Figure 4).
- `Ceres` is the best if we vary the number of attributes (shown in Figure 5).

In this first phase, when considering square matrices, we see a common intersection point (ca. (180,120)) where all the algorithms converge, and afterwards we observe major differences in terms of performance. The common point represents a density of around 30% of the binary relation, corresponding to the complexity of 180. This means that up to 30% of the binary relation, `Ceres` and `Ares` (with a small interval of difference) are the best algorithms. But with a larger density, both algorithms increase their time, whereas `Pluton` keeps its monotone shape. From this first phase, with a square matrix, we can conclude that `Pluton` is not influenced by the density, and that with a low density, `Ares` and `Ceres` are the most suitable.

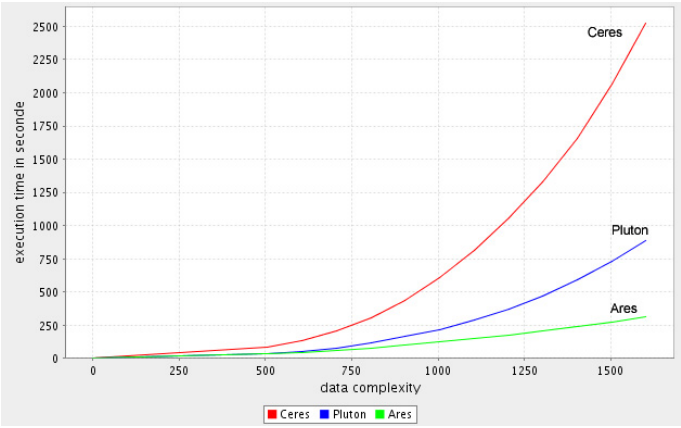

Fig. 4. Test results with a variable number of objects

When considering matrices with a variable number of attributes, we see that the differences in performance between the three algorithms are minimal when we vary the number of attributes between 1000 and 2000 (meaning 500 and 1000 attributes per object) with a density of 50% of the binary relation. After this point, we see that `Ceres` is almost a monotone function, while `Ares` and `Pluton` increase their times. From this we can infer that `Ceres` is the most suitable algorithm.

When considering matrices with a variable number of objects, we see no differences in performance between `Ares` and `Pluton` with a complexity from 0 to 680 with a density of 50% and 50 attributes. However, `Ceres` increases its time as a power function (with an exponent larger than 1) from the minimal complexity.

In the last two cases, we do not see critical points where the algorithms change their performances.

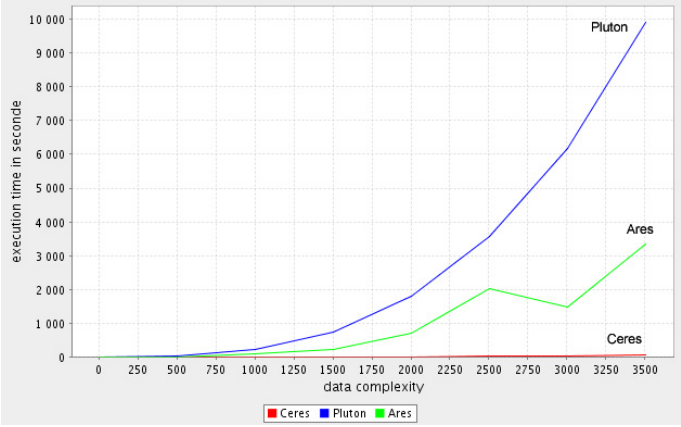

Fig. 5. Test results with a variable number of attributes

5.2 Second Phase

In this second phase, we consider Java and C++ implementations of the algorithms: `Ares`, `Ceres`, `Pluton`, `Ares++`, `Ceres++` and `Pluton++` by completing the results of the first phase with C++ implementations. We see in Figures 6, 7 and 8 that:

- `Pluton++` is the best algorithm when considering square matrices (shown in Figure 6), and when considering a variable number of objects (shown in Figure 7).
- `Pluton++` and `Ceres` are the best ones when considering a variable number of attributes (shown in Figure 8).

Let us discuss the main issues we discover in this phase. Within the context of square matrices (shown in Figure 6), for low densities (up to a complexity of 150, which means 22% of density of binary relation), all the algorithms - except `Pluton` and `Pluton++` - present a monotone increasing function, while `Pluton` and `Pluton++` present almost constant functions. From the complexity of 150, we see three branches of algorithms: `Ares++` and `Ceres`, `Ares` and `Ceres++`, `Pluton` and `Pluton++`. In the first group, the performances of the algorithms increase as power functions (with an exponent larger than 1). In the second group,

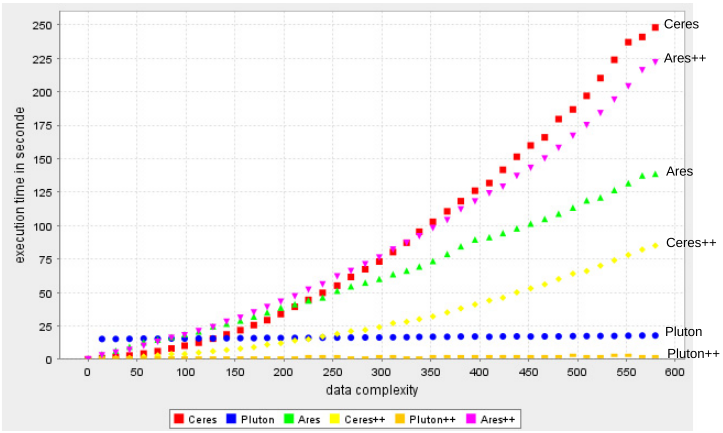

Fig. 6. Test results with a square matrix

the performances are monotone functions; and in the third group, `Pluton` and `Pluton++` remain as almost constant functions. Let us now compare the versions of the algorithms implemented in Java and in C++. If we consider large densities, we see that there is less difference between `Ares` and `Pluton` than between `Ares++` and `Pluton++`, but the situation is reversed in the case of `Ceres` and `Pluton`. There is less difference between `Ceres++` and `Pluton++` than between `Ceres` and `Pluton`.

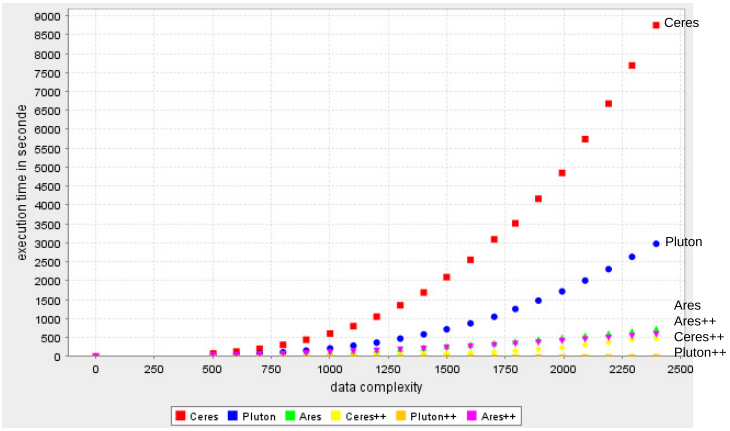

Fig. 7. Test results with a variable number of objects

Generally, we see an improvement of the C++-based algorithms compared to their versions in Java, except in the case of `Ares++`. The major improvement in `Pluton++` illustrates how the API in C++ influences the performances.

When considering a variable number of objects, we see the results in Figure 7. We observe that the implementations in C++ significantly improves the

results, as the slowest algorithm in C++ (`Ares++`) has a better performance than the fastest one in Java (`Ares`). Compared to the results of the square matrix, `Ares++` improves its performance. `Ceres++` and `Ares++` seem to have equivalent performances and `Pluton++` remains - so far - the fastest. As a last issue, we see that there is little difference between the implementations in C++ compared to the implementations in Java, although the difference between the algorithms is very significant.

Regarding a variable number of attributes, Figure 8 shows the results. We observe that there are pairs of algorithms (`Ares++` and `Ares` , `Ceres` and `Ceres++`) that have the same performance with the same variations (with some improvements in the Java version). We also see that the difference between `Pluton` and `Pluton++` is significant. `Pluton` is the slowest while `Pluton++` is the fastest. In addition to this, `Ceres++` seems influenced by the increase in the number of attributes while `Ceres` is not. We should remark that, around the complexity of 2400 (meaning 50 objects, 4800 attributes and a density of 50%), there is an important difference in the performances of `Ares` and `Ares++` regarding complexities smaller than 2400.

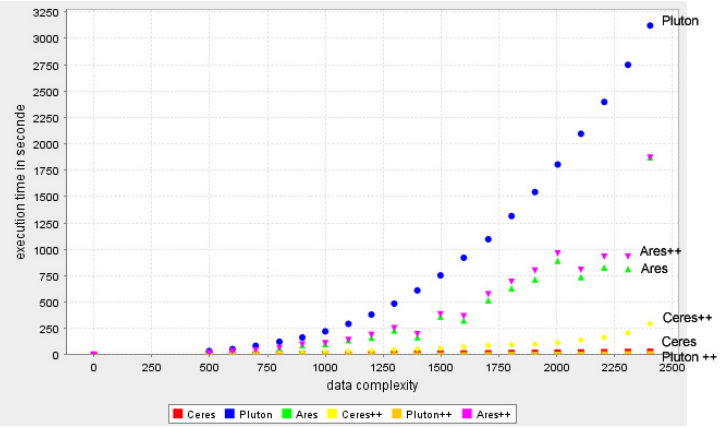

Fig. 8. Test results with a variable number of attributes

As a summary of this phase, we confirm that the API in C++ mostly has a meaningful positive influence on the performances of the algorithms.

5.3 Third Phase

In this phase we decided to test the performances of the algorithms considering the dual versions of the binary relations. When we talk about the dual versions -as said previously-, it does not imply changes in the algorithms, but in the way we deal with the data. In the dual versions, we transpose the matrices representing the binary relations. To perform the experiments, we vary the number of attributes -as we have done in the previous phases. Afterwards, we obtain

the dual versions, meaning that we obtain matrices with a variable number of objects. Then we run the algorithms for non-dual binary relations, and for dual binary relations. Figure 9 shows the corresponding results. In order to detail the local results in the lower part of Figure 9 , we provide a zoomed version in Figure 10. It is worth remarking that in this case, we do not analyze the square matrices, because the matrix transposition (to analyze the dual version) has the same characteristics in our experiments. Despite the fact that Figure 10 is crowded, we see that it represents the superposition of Figure 7 and Figure 8.

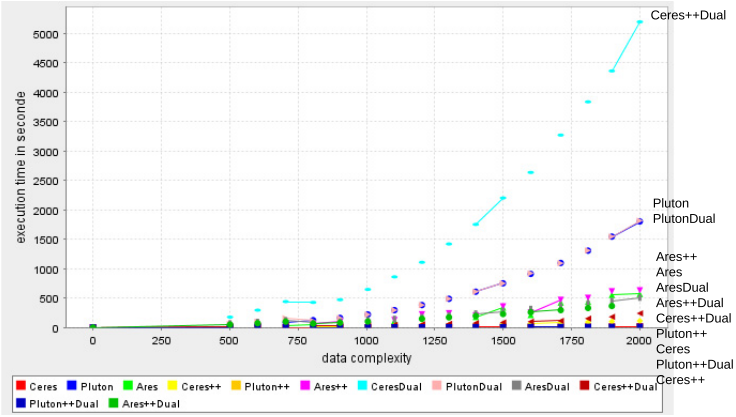

Fig. 9. Test results with a variable number of attributes and their dual versions.

From this phase, we observe that `Pluton` has the same performance if we vary the number of attributes or objects, and the same holds for `Pluton++`. `Pluton++` always has the best performance.

However, `Ceres` changes its performance considerably if we vary the number of objects or the number of attributes. It does not have a good performance when the objects vary but it is faster than the C++ version in the case of attributes. However, `Ceres++` is slower when the number of objects varies than when the number of attributes varies. `Ares` and `Ares++` have the same performance. `Ares++` is better when the number of objects varies.

6 Conclusions and Future Work

This paper compares three different Galois Sub-hierarchy-building algorithms (Ares, Ceres and Pluton) implemented in Java and C++. We see that in most cases, `Pluton++` is the most efficient and stable algorithm. We also see that the API in C++ affects the results of the computation time of these algorithms. It is worth mentioning that for low densities, all algorithms are useful, and the significant differences in performance occur when the binary relations have large densities. Clearly this analysis can guide the user in his/her choice.

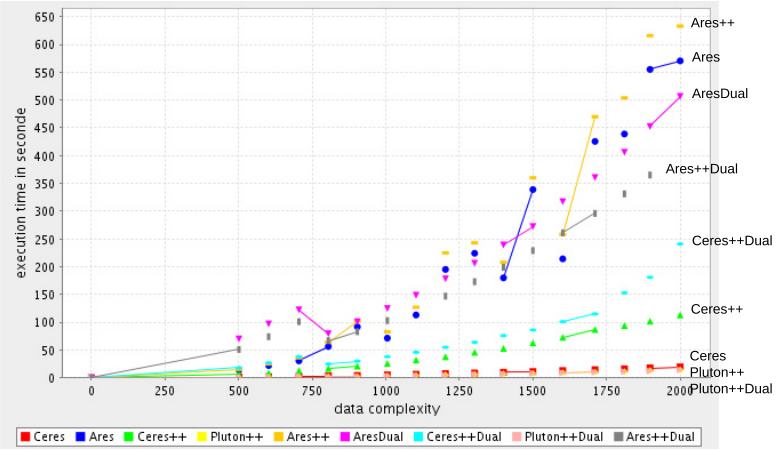

Fig. 10. Zoom of Figure 9

In the future, we plan to extend the analysis to similar algorithms, such as the incremental algorithm ISGOOD [GMM95] and the global algorithm proposed in the work of Mineau et al. [MG95], which build only C^A concepts. Besides this, we propose a profiling of all these algorithms, in order to see which are the critical parts that influence the concepts calculation and performance, taking into account the fact that each algorithm follows a different lattice-building algorithm. As a last issue, we plan to implement the algorithms on another platform (such as Smalltalk/VisualWorks) to see if the C++ implementations are still the best in terms of performance.

Acknowledgements: Gabriela Arévalo gratefully acknowledges the financial support of the Swiss National Foundation for the Project: "Advanced Object-Oriented Reverse Engineering using Formal Concept Analysis" SNF Project No. PBBE2-111194.

References

[AYLCB96] S. Amer-Yahia, L. Lakhal, R. Cicchetti, and J.-P. Bordat. iO2 — An Algorithmic Method for Building Inheritance Graphs in Object Database Design. In *Proc. of ER'96*, volume 1157 of *LNCS*, pages 422–437. Springer-Verlag, 1996.

[BHM+05] A. Berry, M. Huchard, R. M. McConnell, A. Sigayret, and J. P. Spinrad. Efficiently computing a linear extension of the sub-hierarchy of a concept lattice. In *Proc. of ICFCA'05*, volume 3403 of *LNCS*, pages 208–222. Springer-Verlag, 2005.

[DDHL94] H. Dicky, C. Dony, M. Huchard, and T. Libourel. ARES, un algorithme d'ajout avec restructuration dans les hiérarchies de classes. *Proc. of LMO'94, L'Objet*, pages 125–136, 1994.

[DHL+02] M. Dao, M. Huchard, T. Libourel, C. Roume, and H. Leblanc. A New Approach to Factorization: Introducing Metrics. In *Proc. of Metrics '02*, pages 227–236. IEEE Computer Society, 2002.

[FLPP05] L. Fura, G. Laplace, A. Le Provost, and G. Perrot. Algorithme de construction d'une sous-hiérarchie de Galois. Technical report, Université de Montpellier II, 2005.

[Gal] GaLicia: Galois lattice interactive constructor. Université de Montréal. http://www.iro.umontreal.ca/∼galicia.

[GC99] R. Godin and T.-T. Chau. Comparaison d'algorithmes de construction de hiérarchies de classes. *L'Objet*, 5(3/4), 1999.

[GM93] R. Godin and H. Mili. Building and Maintaining Analysis-Level Class Hierarchies using Galois Lattices. In *Proc. of OOPSLA '93*, volume 28, pages 394–410. ACM Press, October 1993.

[GMM95] R. Godin, G. Mineau, and R. Missaoui. Incremental structuring of knowledge bases. In G. Ellis, R. A. Levinson, A. Fall, and V. Dahl, editors, *Proc. of KRUSE'95*, pages 179–193, University of California at Santa Cruz, 1995. Department of Computer Science.

[GMM+98] R. Godin, H. Mili, G. W. Mineau, R. Missaoui, A. Arfi, and T.-T. Chau. Design of Class Hierarchies based on Concept (Galois) Lattices. *Theory and Practice of Object Systems*, 4(2):117–134, 1998.

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

[HDL00] M. Huchard, H. Dicky, and H. Leblanc. Galois Lattice as a Framework to specify Algorithms Building Class Hierarchies. *Theoretical Informatics and Applications*, 34:521–548, 2000.

[Hit04] P. Hitzler. Default reasoning over domains and concept hierarchies. In *Proc. of KI 2004*, volume 3238 of *LNCS*, pages 351–365. Springer Verlag, 2004.

[KO02] Sergei O. Kuznetsov and Sergei A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3):189–216, 2002.

[Leb00] H. Leblanc. *Sous-hiérarchies de Galois: un modèle pour la construction et l'évolution des hiérarchies d'objets*. PhD thesis, Univ. de Montpellier II, 2000.

[MG95] G. W. Mineau and R. Godin. Automatic structuring of knowledge bases by conceptual clustering. *IEEE Trans. Knowl. Data Eng.*, 7(5):824–828, 1995.

[OP02] R. Osswald and W. Petersen. Induction of classifications from linguistic data. In *Proc. of ECAI'02 Workshop*, pages 75–84. Université de Lyon I, July 2002.

[Per05] G. Perrot. Implémentation d'algorithmes de construction de sous-hiérarchies de Galois et étude des performances, 2005.

[Pet01] W. Petersen. A set-theoretical approach for the induction of inheritance hierarchies. In *Electronic Notes in Theoretical in Computer Science*, volume 51. Elsevier, July 2001.