

# Incremental concept formation algorithms based on Galois (concept) lattices

Appeared in *Computational Intelligence* (1995), 11(2), 246-267

ROBERT GODIN, ROKIA MISSAOUI, HASSAN ALAOUI

*Département d'Informatique*

*Université du Québec à Montréal*

*C.P. 8888, Succursale "Centre Ville"*

*Montréal (Québec), Canada, H3C 3P8*

*(514)987-3088*

E-mail: godin.robert@uqam.ca

**Abstract.** The Galois (or concept) lattice produced from a binary relation has been proved useful for many applications. Building the Galois lattice can be considered as a conceptual clustering method since it results in a concept hierarchy. This article presents incremental algorithms for updating the Galois lattice and corresponding graph, resulting in an incremental concept formation method. Different strategies are considered based on a characterization of the modifications implied by such an update. Results of empirical tests are given in order to compare the performance of the incremental algorithms to three other batch algorithms. Surprisingly, when the total time for incremental generation is used, the simplest and less efficient variant of the incremental algorithms outperforms the batch algorithms in most cases. When only the incremental update time is used, the incremental algorithm outperforms all the batch algorithms. Empirical evidence shows that, on the average, the incremental update is done in time proportional to the number of instances previously treated. Although the worst case is exponential, when there is a fixed upper bound on the number of features related to an instance, which is usually the case in practical applications, the worst case analysis of the algorithm also shows linear growth with respect to the number of instances.

**Keywords.** Concept lattice, Galois lattice, conceptual clustering, incremental algorithm, concept formation.

**Subject Categories.** Learning, Knowledge Representation.

## 1. Introduction

Building the Galois lattice of a binary relation has many important applications. Wille (1982) proposed to consider each element in the lattice as a concept and the corresponding graph (Hasse diagram) as the generalization/specialization relationship between concepts. From this perspective, the lattice represents a concept hierarchy. Each concept is a pair composed of an extension representing a subset of instances and an intension representing the common features for this set of instances. The task of inducing a concept hierarchy in an incremental manner is known as *incremental concept formation* or simply concept formation (Fisher, Pazzani & Langley, 1991; Gennari, Langley & Fisher, 1990) and is a fundamental process of human learning. In this article we present and analyze incremental algorithms for building Galois lattices. The main distinguishing characteristics of this method with respect to other well-known concept formation methods such as UNIMEM (Lebowitz, 1987), COBWEB (Fisher, 1987) and CLASSIT (Gennari et al., 1990) are that the concept hierarchy is a lattice, not restricted to a tree hierarchy, and although the algorithm is incremental, the resulting hierarchy does not depend on any tunable parameters, algorithm specifics, or the order in which the instances are acquired. A good overview of work on concept formation is found in (Fisher et al., 1991; Gennari et al., 1990).

The practical application of using Galois lattices has resulted in many developments concerning the visualization of the lattice using computer generated diagrams (Wille, 1984), its simplification through decompositions or pruning heuristics (Ganter & Wille, 1989; Godin & Mili, 1993; Mephu Nguifo, 1993), its use in an interactive knowledge acquisition process and the generation of rules from the lattice (Guigues & Duquenne, 1986; Wille, 1992) which can be used for knowledge discovery in databases (Godin & Missaoui, 1994). In (Missaoui & Godin, 1994) we show how the concept lattice can usefully be (i) exploited (by avoiding combinatorial computations) as a framework for computing the basic measures usually calculated in the rough sets studies (Pawlak, 1991), (ii) used to define additional notions related to the intensional dimension of a classification. Carpinieto and Romano (1993) show how the lattice can be used successfully for class discovery and class prediction using several machine learning data sets. The Galois lattice can also be useful as a browsing space for information retrieval (Godin, Missaoui & April, 1993; Godin, Saunders & Gecsei, 1986). A structure very similar to the Galois lattice has been proposed for retrieval of classes in a large library based on class features (Oosthuizen, Bekker & Avenant, 1992). Recently, several other software engineering applications have been considered (Godin & Mili, 1993; Godin, Mineau, Missaoui, St-Germain & Faraj, 1995; Krone & Snelting, 1994). Extensions and refinements of the basic theory have been proposed for dealing with richer representations such as attribute-value pairs (Godin et al., 1986; Wille, 1992), conceptual graphs and use of feature taxonomies (Godin & Mili, 1993; Mineau & Godin, 1994).

Many algorithms have been proposed for generating the Galois lattice from a binary relation (Bordat, 1986; Carpinieto & Romano, 1993; Chein, 1969; Fay, 1975; Ganter, 1984; Godin, Missaoui & Alaoui, 1991; Malgrange, 1962; Norris, 1978). Only two of these (Carpinieto & Romano, 1993; Godin et al., 1991) incrementally update the lattice and the corresponding Hasse diagram. This feature is necessary in many applications. As argued in (Frawley, Piatetsky-Shapiro & Matheus, 1991), two important features for the practical feasibility of automated knowledge discovery methods are that they should be incremental and efficient. From the currently known algorithms, the Bordat algorithm (Bordat, 1986) builds the Hasse diagram but is not incremental, and the Norris algorithm is incremental but it does not build the Hasse diagram (Guénoche, 1990). Furthermore, very little is known about the complexity of these algorithms. The algorithms we present in this article generate both the lattice and Hasse diagram incrementally. Furthermore, a detailed comparison is made to show the relative complexity of these algorithms.

The following section recalls basic definitions related to the concept of Galois lattice. Section 3 presents incremental algorithms after giving a characterization of the modifications induced by adding a new instance.

Section 4 presents detailed comparisons with other well-known algorithms, some of which were adapted to generate the Hasse diagram.

## 2. Basic definitions

This section recalls basic definitions related to the concept of Galois lattice for a binary relation. We limit our description to the aspects relevant to this paper. The following references give a more complete and formal coverage of the subject (Barbut & Monjardet, 1970; Davey & Priestley, 1992; Wille, 1982; Wille, 1992). Given a set of instances  $E$  and a set of features  $E'$ , and a binary relation  $R$  between these two sets,  $R \subseteq E \times E'$  (Table 1), there is a unique Galois lattice corresponding to this binary relation (Figure 1). Each element of the lattice is a pair, noted  $(X, X')$ , composed of a set  $X \subseteq P(E)$  and a set  $X' \subseteq P(E')$ .  $P(A)$  represents the powerset of  $A$ . Each pair must be a complete pair as defined in the following.

A pair  $(X, X')$  from  $P(E) \times P(E')$  is *complete* with respect to  $R$  if and only if the two following properties are satisfied:

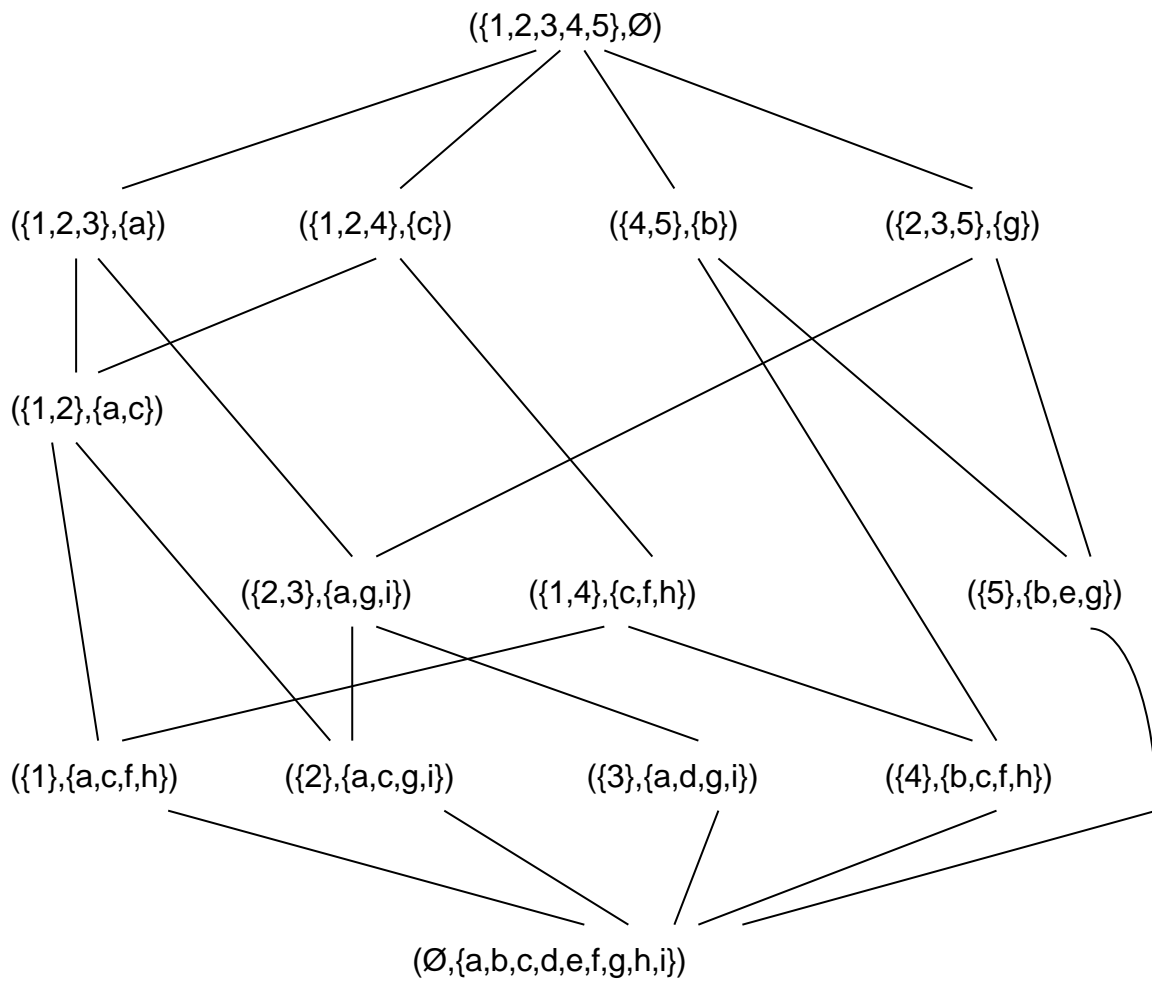
- 1)  $X' = f(X)$  where  $f(X) = \{x' \in E' \mid \forall x \in X, xRx'\}$ .
- 2)  $X = f'(X')$  where  $f'(X') = \{x \in E \mid \forall x' \in X', xRx'\}$

Notice that  $f(\emptyset) = E'$  and  $f'(\emptyset) = E$ . Only maximally extended pairs are kept in the hierarchy. If a subset  $X$  of instances is not maximally extended in the sense that there are other instances described by the features common to  $X$ , then the subset has to be extended to contain this instance. Symmetrically, if  $X'$  does not contain a feature that is common to every instance in  $X$ , it has to be extended to include this feature. The idea of maximally extending the sets is formalized by the mathematical notion of a *closure* in ordered sets. A closure in an ordered set  $(E, =)$  is an application,  $h: E \rightarrow E$ , having the following properties:

- i)  $\forall x \forall y, x = y \Rightarrow h(x) = h(y)$
- ii)  $\forall x, h(x) = x$
- iii)  $\forall x, h(h(x)) = h(x)$

		E'									
		a	b	c	d	e	f	g	h	i	
E	R	1	0	1	0	0	1	0	1	0	
	2	1	0	1	0	0	0	1	0	1	
	3	1	0	0	1	0	0	1	0	1	
	4	0	1	1	0	0	1	0	1	0	
	5	0	1	0	0	1	0	1	0	0	

**Figure 1.** Matrix representation of a binary relation  $R$ .



**Figure 1.** Galois lattice.

The image  $h(x)$  of  $x$  in  $E$  is called  $h$ -closure of  $x$ ; if  $x = h(x)$ ,  $x$  is  $h$ -closed or a closed element for  $h$ . The applications  $h = f \circ f'$  and  $h' = f' \circ f$  are respectively closures in  $E$  and  $E'$ . Therefore the  $h$ -closed elements of  $E$  correspond to the maximally extended instance subsets and they form a lattice  $H$  isomorphic to the Galois lattice. The isomorphism is simply to put into correspondence the closed elements with the  $X$  sets of  $G$ . Symmetrically, the  $h'$ -closed elements of  $E'$  are the maximally extended feature subsets and they correspond to the  $X'$  sets of the  $G$ .

The couple of functions  $(f, f')$  is a *Galois connection* between  $P(E)$  and  $P(E')$  and *the Galois lattice*  $G$  for the binary relation is the set of all complete pairs (Barbut & Monjardet, 1970) with the following partial order:

$$\text{given } C_1=(X_1,X'_1) \text{ and } C_2=(X_2,X'_2), C_1 < C_2 \iff X'_1 \subset X'_2 .$$

There is a dual relationship between the  $X$  and  $X'$  sets in the lattice, i.e.,

$$X'_1 \subset X'_2 \iff X_2 \subset X_1$$

and therefore,

$$C_1 < C_2 \iff X_2 \subset X_1 .$$

The partial order is used to generate the lattice graph in the following way: there is an edge  $(C_1,C_2)$  if  $C_1 < C_2$  and there is no other element  $C_3$  in the lattice such that  $C_1 < C_3 < C_2$ .  $C_1$  is called parent of  $C_2$  and  $C_2$  child of  $C_1$ . The graph is usually called a Hasse diagram. When drawing a Hasse diagram, the edge direction is either downwards or upwards. The Hasse diagram therefore reveals the generalization/specialization

relationship between the concepts corresponding to the subset relationship between the extension or intension of the concepts. Given,  $C$ , a set of elements from  $G$ ,  $\inf(C)$  and  $\sup(C)$  will denote respectively the greatest lower bound and least upper bound of the elements in  $C$ . A fundamental property of  $G$  is that it is a complete lattice since for any subset of  $G$  (complete pairs) there exists a unique greatest lower bound and a unique least upper bound (Davey & Priestley, 1992).

### 3. Incremental Update Algorithms

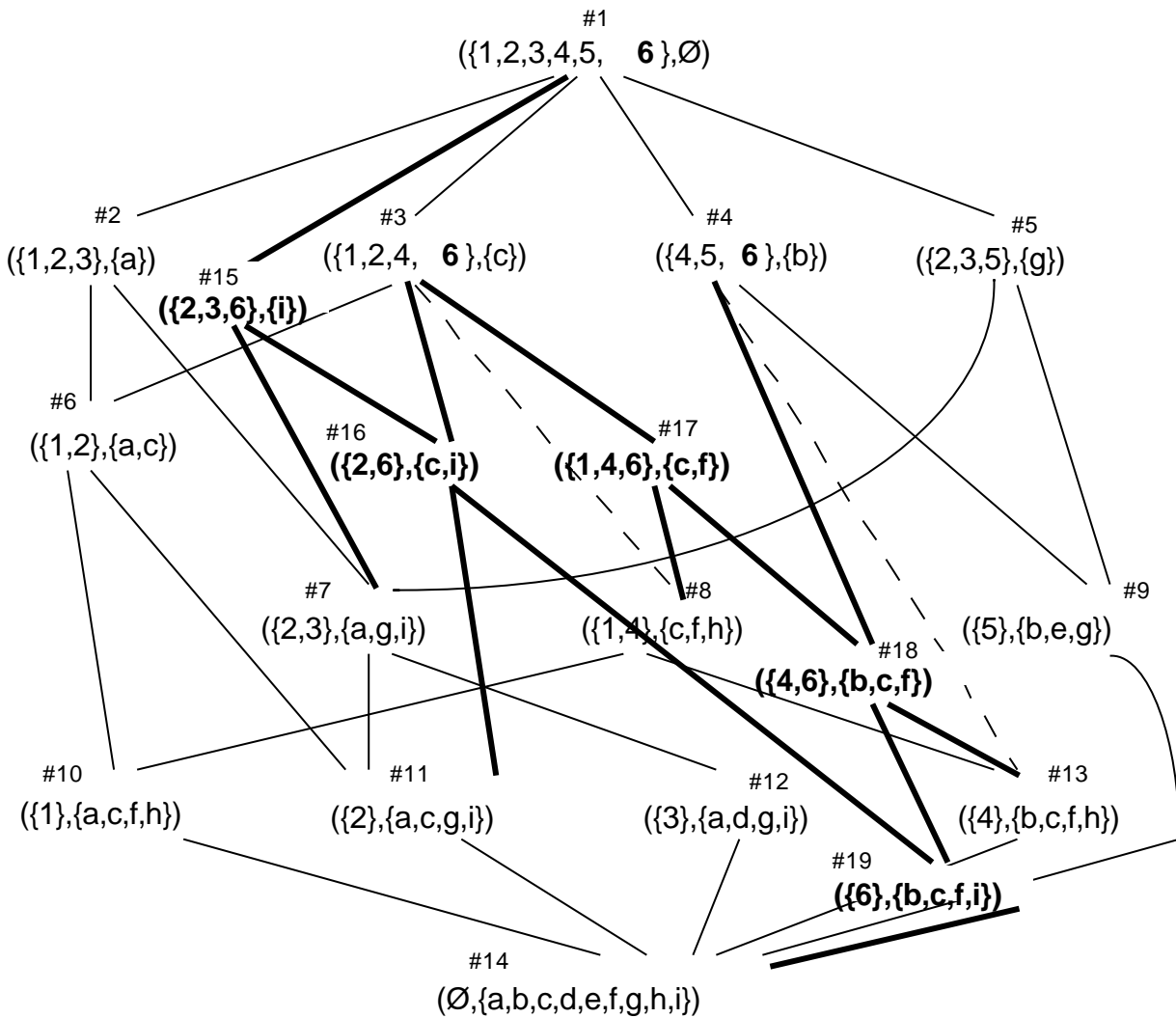
Before presenting the incremental update algorithms, a description of the update problem is first given in Section 3.1. The presentation of this paper extends on the material in (Godin et al., 1991) by giving a more complete and detailed presentation of the algorithms. Some of this material is also used in (Godin & Missaoui, 1994) to describe an algorithm for the incremental update of rules generated from a database using the lattice.

#### 3.1 Characterization of the update problem

For most applications, it is necessary to generate not only the complete pairs of the lattice  $G$  but also the Hasse diagram of the lattice. It is also important in many applications to be able to incrementally add a new instance,  $x^*$ , by modifying the lattice without having to regenerate it from scratch. For example, adding the new element,  $x^* = 6$ , related to the set,  $f^*({x^*}) = \{b, c, f, i\}$ , would result in the modifications shown in Figure 2 representing the new lattice noted  $G^*$ .  $G^*$  can be obtained from  $G$  by taking all the pairs in  $G$  and modifying the  $X$  part of the nodes for which  $X' \subseteq f^*({x^*})$  by adding  $x^*$ . The pairs that remain intact are called *old pairs* and the others, *modified pairs* in  $G^*$ . In addition, *new pairs* are created. These new pairs are always in the form  $(Y \cup \{x^*\}, Y' \cap f^*({x^*}))$  for some pair  $(Y, Y') \in G$ . They correspond to new  $X'$  sets with respect to  $G$ . The related  $(Y, Y')$  node in  $G$  is called the *generator* for this new pair. If the generators can be characterized in some manner, the new nodes can be generated from them. Proposition 1 is one possible characterization and is the basis for Algorithm 1.

#### Proposition 1.

If  $(X, X') = \inf\{(Y, Y') \in G \mid X' = Y' \cap f^*({x^*})\}$  for some set  $X'$  and there is no node of the form  $(Z, X') \in G$  then  $(Y, Y')$  is the generator of a new pair  $(X = Y \cup \{x^*\}, X' = Y' \cap f^*({x^*})) \in G^*$ .



**Figure 2.** Modifications resulting from adding element 6 related to the set {b, c, f, i}.

Any new  $X'$  set in  $G^*$  will have to be the result of intersecting  $f^*({x^*})$  with some  $Y'$  set already present in the lattice  $G$ . There may be many pairs in  $G$  that give a particular new intersection in this manner. For example, in Figure 2, the new  $X'$  set,  $\{i\}$ , in the new pair #15, can be formed by intersecting  $f^*({x^*}) = \{b, c, f, i\}$  with the  $Y'$  set of pair #7,  $\{a, g, i\}$ , or the  $Y'$  set of pair #12,  $\{a, d, g, i\}$ . However, there is only one of these that is the generator of the new pair, #7 in this example. This pair is the smallest (inf) old pair that produces the intersection. Unicity of the greatest lower bound is guaranteed by the lattice property of  $G$ . In Figure 2, the generator pairs for the new pairs #15, #16, #17, #18, #19 are respectively #7, #11, #8, #13, #14.

One minor problem is for the case of the generator being  $\text{sup}(G) = (\emptyset, E')$ . If the new instance  $x^*$  contains new features not contained in  $E'$ , there will be no generators for the pair  $(\{x^*\}, f(\{x^*\}))$ . In practical applications, we may want  $E'$  to grow as new features are encountered. This is easily taken into account by simply adding the new features to  $E'$  as a first step in the algorithm and therefore the characterization remains valid.

In addition, the edges of the Hasse diagram also have to be updated. First, the generator of a new pair will always be a child to the new pair in the Hasse diagram. The children of old pairs do not change. The parents of generator pairs, however, have to be changed. The generator is the only old pair that becomes a child of the new pair. There may be another child but it will be a new pair. For example in Figure 2, there is an edge

from the new pair # 17 to its generator #8 and there is another child #18 that is a new pair. The parents of old pairs that are not generators remain unchanged.

Also, the parents of modified pairs never change. However, the children of some modified pairs may be modified. There may be new children that are new pairs. This implies that some old children may have to be removed if the new children fall in between the old child and the modified pair. This is the case when the new pair falls between a generator and one of its parents. The result is that the edge from that parent to the generator (e.g., edge(#3,#8) in Figure 2) is replaced by two edges, one from the parent to the new pair (e.g., edge(#3,#17)) and one from the parent to the generator (e.g., edge(#17,#8)).

Table 2 is a summary of the modifications resulting from the update process with respect to our classification of pairs. The updating is decomposed into four aspects: the X set, the X' set, the parents and children. The first three categories represent the pairs that are in G and remain in G\* with possibly some modifications. As opposed to category 3, categories 1 and 2 are pairs that have to be modified. The fourth category is for the new pairs.

**TABLE 2.** Summary of the modifications in the update process.

Type of node	X set	X' set	parents	children
1. Modified node (Y, Y') ∈ G	Add x*	No change	No change	Add new nodes in some cases Remove a generator when a new node in between
2. Old node generator of N	No change	No change	Add new node N and remove parent when N is in between this parent and the generator	No change
3. Old node non generator	No change	No change	No change	No change
4. New node having generator (Y, Y')	$Y \cup \{x^*\}$	$Y' \cap f(\{x^*\})$	Old nodes and new nodes	Generator and possibly new node

### 3.2 Basic algorithms

Algorithm 1 is a basic algorithm and a refinement which could be considered for large data sets is given in Algorithm 2. For a node H, X(H) and X'(H) will denote respectively the extent (first component) and intent (second component) of the complete pair. Sup(G) and inf(G) will denote respectively the lowest upper bound and greatest lower bound of the lattice.

The lattice is initialized with one element:  $(\emptyset, \emptyset)$ . This means that  $E = E' = \emptyset$ . The algorithm updates E and E' as new elements are added. If we suppose that E and E' contain in advance every element with an empty R, the lattice would be initialized with the two elements:  $(E, \emptyset)$  and  $(\emptyset, E')$ . This would slightly simplify the algorithm because adjusting E' by adding new elements from  $f(\{x^*\})$  would not be necessary.



**Algorithm 1** Add ( $x^*$ : new object;  $f(\{x^*\})$ : elements related to  $x^*$  by R);

**BEGIN**

```

1  {Adjust (sup(G)) for new elements in E'}
2  IF sup(G) = ( $\emptyset$ ,  $\emptyset$ ) THEN
3    Replace sup(G) by: ( $\{x^*\}, f(\{x^*\})$ )
4  ELSE
5    IF NOT ( $f^*(\{x^*\}) \subseteq X'(\text{sup}(G))$ ) THEN
6      IF  $X(\text{sup}(G)) = \emptyset$  THEN  $X'(\text{sup}(G)) := X(\text{sup}(G)) \cup f(\{x^*\})$ 
7      ELSE
8        Add new pair H {becomes sup(G*)}: ( $\emptyset, X'(\text{sup}(G)) \cup f(\{x^*\})$ );
9        Add new edge sup(G)->H
10     END IF
11   END IF;
12    $C[i] \leftarrow \{H: |X'(H)|=i\}$ ; {Class pairs in buckets with same cardinality of the X' sets}
13    $C'[i] \leftarrow \emptyset$ ; {Initialize the C' sets}
14   {Treat each bucket in ascending cardinality order}
15. FOR  $i : 0$  TO maximum cardinality DO
16   FOR each pair H in  $C[i]$ 
17     IF  $X'(H) \subseteq f(\{x^*\})$  THEN {modified pair}
18       Add  $x^*$  to  $X(H)$ ;
19       Add H to  $C'[i]$ ;
20     IF  $X'(H) = f(\{x^*\})$  THEN exit algorithm
21     ELSE {old pair}
22        $\text{int} \leftarrow X'(H) \cap f(\{x^*\})$ ;
23     IF  $\neg \exists H_1 \in C'[|\text{int}|]$  such that  $X'(H_1) = \text{int}$  THEN {H is a generator}
24       Create new pair  $H_n = (X(H) \cup \{x^*\}, \text{int})$  and add to  $C'[|\text{int}|]$ ;
25       Add edge  $H_n \rightarrow H$ ;
26       {Modify edges}
27       FOR  $j : 0$  TO  $|\text{int}|-1$ 
28         FOR each  $H_a \in C'[j]$ 
29           IF  $X'(H_a) \subset \text{int}$  { $H_a$  is a potential parent of  $H_n$ }
30              $\text{parent} \leftarrow \text{true}$ ;
31             FOR each  $H_d$  child of  $H_a$ 
32               IF  $X'(H_d) \subset \text{int}$   $\text{parent} \leftarrow \text{false}$ ; exit FOR END IF
33             END FOR;
34             IF  $\text{parent}$ 
35               IF  $H_a$  is a parent of H
36                 eliminate edge  $H_a \rightarrow H$  END IF;
37               Add edge  $H_a \rightarrow H_n$ 
38             END IF
39           END IF
40         END FOR
41       END FOR;
42     IF  $\text{int} = f^*(\{x^*\})$  THEN exit algorithm END IF
43   END IF
44 END IF
45 END FOR
46 END FOR
47 END IF
END {Add}

```

As introduced earlier, the basic idea is to generate  $G^*$  from  $G$ , changing the  $X$  sets, and links of the pairs already in  $G$  and adding some new pairs for the new  $X'$  sets and linking them. The new pairs are generated by finding the generator pairs using the characterization of Proposition 1. The function of lines 1 to 11 is to take into account the case when new features appear by adjusting  $E'$  in  $\text{sup}(G)$ . Line 12 classifies the pairs into

buckets with the same  $\|X'\|$ . The main loop (lines 15-46) iterates on every pair in ascending  $\|X'\|$ . New pairs are obtained by systematically trying to generate a new intersection from each pair  $(Y, Y')$  already in the lattice by intersecting  $Y'$  with  $f(\{x^*\})$  (line 22). Verifying that this intersection is not already present is done by looking in the sets already encountered which are subsets of  $f(\{x^*\})$  (line 23). These sets are kept in  $C'$  (line 19, line 24). This is valid because the pairs are treated in ascending cardinality of the  $X'$  sets. Furthermore, the first pair encountered which gives a new intersection is the generator of the new pair because it is necessarily the infimum. Thus, we compute the  $X$  set of the New pair by adding  $x^*$  to the generator's  $X$  set (line 24). Also, there is automatically an edge from the new pair to the generator (line 25). When a new pair is added, some edges have to be added from modified or other new pairs to the new pair. The candidates are necessarily in the  $C'$  sets since their  $X'$  set must be a subset of  $f(\{x^*\})$ . These parents of the new pair are determined by examining the pairs in  $C'$  (lines 27-41), testing if the  $X'$  sets are subsets of the  $X'$  set of the new pair (line 29) and verifying that no child of the potential parent has this property (lines 30-34). It is necessary to eliminate an edge between the new parent and the generator when there is such an edge (lines 34-35). The exhaustive iteration on the  $C'$  set is a fairly simple minded approach to modifying the edges and could be optimized somewhat. Lines 17-20 process the modified pairs and the rest of the treatment is skipped for these pairs because they cannot be generators.

Although in the worst case, the lattice may grow exponentially, the growth is linearly bounded with respect to  $\|E\|$ , when there is a fixed upper bound on  $\|f(\{x\})\|$ . This is the case in practical applications such as information retrieval where for each document  $x$  in  $E$  there is a reasonable fixed upper bound on the number of terms from  $E'$  related to  $x$  (Godin et al., 1986). Clearly any new  $X'$  other than the updating of  $E'$  is a subset of  $f(\{x^*\})$ . This implies that there is at most  $2^{\|f(\{x^*\})\|}$  additional pairs generated by adding  $x^*$ . If there is an upper bound  $K$ , independent of the  $\|E\|$ , on  $\|f(\{x\})\|$ , the lattice grows linearly with respect to  $\|E\|$ :

$$\|G\| = 2^K \|E\|.$$

Experimental applications and theoretical results based on a uniform distribution hypothesis show that the average growth factor obtained is far less than the  $2^K$  bound. In every application, we observe that:

$$\|G\| = k \|E\|,$$

where  $k$  is the mean value for  $\|f(\{x\})\|$ . Furthermore, theoretical estimations based on different values of  $k$  suggest that the lattice may grow linearly with respect to the number of terms per document. Using a formula derived in (Godin et al., 1986) for the mean value of  $\|G^*\|$  with random assignment of index terms within documents, we have computed the theoretical values for different values of  $k$ , keeping other parameters fixed. Again, we obtain  $\|G^*\|/n < k$  and the observed growth is linear in  $k$  (Godin et al., 1993). More details on the complexity of the lattice are found in (Godin, 1989; Godin et al., 1986).

The time complexity of iterating on the pairs for creating the intersections and verifying the existence of the intersection in  $C'$  is the major factor in analyzing the complexity of the algorithm. Although the linking process is complicated, the pairs affected are limited and this part is only done when a generator pair is encountered. This is why we give a fairly straightforward algorithm for this process. Since  $\|G\| = 2^K \|E\|$  and the  $\|C'\|$  is bounded by  $2^K$ , the whole time complexity is:

$$\mathbf{O}(\|G\| 2^K) = \mathbf{O}(2^{2K} \|E\|),$$

which is  $\mathbf{O}(\|E\|)$  for a fixed  $K$ . In applications where  $k$  is large, a more refined algorithm for the edge updating is to consider.

Algorithm 1 iterates on almost every pair of  $G$ . It is possible to do the work by looking at a limited subset of  $G$ : the pairs that correspond to generators and modified pairs. This can be done by keeping for each  $x'$  in  $E'$  a pointer  $P_{X'}$  on the smallest pair containing  $x'$  and using these pointers as entry points for a top-down depth-first search starting with every  $x'$  in  $f(\{x^*\})$ . This guarantees that any pair encountered will have at least  $x'$  in common with  $f(\{x^*\})$ . Algorithm 2 performs the search in this manner. Some minor modifications have to be introduced for maintaining these pointers but the details are left out.



**Algorithm 2** Add(  $x^*$  : new object;  $f(\{x^*\})$  : elements related to  $x$  by  $R$ );

**Procedure** SelectAndClassifyPairs

**Procedure** Search( $H$ : pair)

**BEGIN**

Mark  $H$  as visited and add  $H$  to  $C[|X'(H)|]$ ;

**FOR** each  $H_d$  child of  $H$

**IF**  $H_d$  is not marked as visited

Search( $H_d$ )

**END IF**

**END FOR**

**END** {search}

**BEGIN**

Mark  $\text{inf}(G)$  as visited and add  $\text{inf}(G)$  to  $C[|X'(\text{inf}(G))|]$

**FOR** each  $x'$  in  $f(\{x^*\})$ .

Search( $P_{x'}$ )

**END FOR**

**END** {SelectAndClassifyPairs}

**BEGIN**

....

12. SelectAndClassifyPairs;

....

**END.** {Add}

Compared to Algorithm 1, Algorithm 2 can offer significant gains. For example, in an information retrieval application involving more than 3000 documents (Godin, 1986), the proportion of nodes treated by Algorithm 2 varied between 10% to 15% of the whole lattice. However, even if there is an important gain with this strategy compared to the basic Algorithm 1, the asymptotical complexity is unchanged with respect to Algorithm 1 because generally the subset of pairs is of the same order of complexity as the whole lattice.

### 3.3 Other variants

In this section we describe some variants of the previous algorithms which could prove to be useful in some contexts. However, we have not considered these variants in our experiments. Although the algorithm by Norris (Norris, 1978). generates the pairs using the same basic method as in Algorithm 1 by scanning each node in  $G$ , testing for modified pairs and testing for generators in order to add new pairs, it differs from ours in the way the generator is characterized. In the Norris algorithm, the complete pair property of the potential new pair,  $(Y \cup \{x^*\}, Y' \cap f^*(\{x^*\}))$ , is explicitly computed by testing if  $f(Y' \cap f^*(\{x^*\})) = Y \cup \{x^*\}$ . Therefore, the information necessary for this computation must be incrementally maintained and the computation time grows with  $|E|$  as opposed to our approach for testing generators. Furthermore, the Norris algorithm does not address the graph update part. Their method of determining generators could nevertheless be incorporated in Algorithms 1 and 2 giving other variants that might be advantageous in some cases. In particular, when the average for  $|f(\{x\})|$  is large, the number of pairs in  $C'$  can grow so much that computing the  $f$  function would be less expensive.

Another possible variation to be considered is to use the generator characterization of Proposition 2. It is a more local characterization since it only depends on conditions expressed in terms of the candidate generator and its parents.

**Proposition 2.**

$(Y, Y') \in G$  is the generator of the new pair  $(X, X') \in G^*$  if and only if

$$Y' \not\subseteq f^*({x^*}) \text{ and for every parent } (Z, Z') \text{ of } (Y, Y') \text{ in } G, Y' \cap f^*({x^*}) \not\subseteq Z'$$

This would mean replacing the search of the  $C'$  set to determine if the intersection is new by looking at the set of parents of the pair to be tested. Empirical evidence from a large information retrieval application (Godin, 1986) suggests that the mean number of parents (which is equal to the mean number of children) grows proportionally to  $\ln(|E|)$ , and that this number is often smaller than the number of pairs in  $C'$ . This suggests some potential saving in CPU time if the characterization of Proposition 2 is used instead of searching  $C'$ . The algorithm proposed in (Carpineto & Romano, 1993) uses this approach to generate the new nodes. One major disadvantage in their algorithm with respect to Algorithm 2 is that it iterates over all the nodes in the lattice as our Algorithm 1. Since this strategy systematically considers the set of parents of a node, a natural way proposed in (Godin, 1986) is to use a bottom-up strategy starting with  $\text{sup}(G)$  and searching the lattice with a depth-first search and using the characterization of Proposition 2 for the generator. This strategy however looks at more nodes than Algorithm 2 because some nodes for which  $X'(H) \cap f^*({x^*}) = \emptyset$  have to be treated. In particular, for  $\text{sup}(G)$ , a large number of non relevant parent nodes have to be considered.

To deal with this problem, another algorithm has been tested (Godin, 1986) which searches the lattice as in Algorithm 2 using a depth first search based on the characterization of Proposition 2 but in a different manner than suggested above. Instead of verifying the characterization of Proposition 2 directly by looking at all the parents at once, nodes are created temporarily until a parent that contradicts the property is found. Therefore, some nodes that have been created by mistake will be deleted. The algorithm is more complex than Algorithm 2 and did not give any improvement for our experimental applications. There may be some advantage in this strategy when central memory is limited because only local information is used to process a node.

A major advantage of using the characterization of Proposition 2 for generating new nodes is when considering parallel algorithms. Given the localized characterization of the generators, these can be tested independently in parallel. A parallel algorithm for building the lattice in such a manner is proposed in (Ouaggag, Godin, Missaoui & Mili, 1993).

**3.4 Removing instances**

Removing an instance from the lattice might also be considered for some applications. This problem is much easier than adding an instance. One simple method is to start from the lowest pair containing the instance to remove and to search upwards recursively, verifying if the parents are still necessary. When deleting a parent, care must be taken for relinking nodes, if necessary. The number of pairs to examine is limited to pairs  $(X, X')$  where  $X' \subseteq f({x})$ ,  $x$  being the instance. This process will therefore be much more efficient than adding an instance. More details are found in (Godin, 1986).

## 4 Experimental evaluation of the algorithms

Experiments have been conducted to evaluate empirically the behavior of algorithms. Incremental algorithms are useful only if the cost of incremental update is less than regenerating from scratch using some other batch algorithm. Therefore, we have compared the performance of our incremental algorithms with other batch algorithms found in the literature. The following subsection describes these algorithms.

### 4.1 Batch algorithms for generating the lattice and Hasse diagram

Many non-incremental batch algorithms have been proposed to generate the Galois lattice of a binary relation. From the currently known algorithms, only the Bordat algorithm (Bordat, 1986) builds the Hasse diagram (Guénoche, 1990), and was therefore used for the experiments. The basic idea of this algorithm is given in Algorithm 3. The lattice  $G$  is constructed by starting with the infimum of the lattice,  $(E, f(E))$ , and generating all the children which are then added to  $G$  and linked to their parent. This child generation process is iteratively repeated for every new pair.

**Algorithm 3** Build  $G$  and Hasse diagram (Bordat's algorithm)

```

BEGIN
  G:= {(E,f(E))};
  L:= {(E,f(E))};
  WHILE L ≠ ∅
    L' := ∅
    FOR each pair H in L
      Generate each child pair Hc for H
      IF pair Hc was not previously generated
        Add child to G and to L'
      END IF;
      Add edge H->Hc
    END FOR;
    L:=L'
  END WHILE
END

```

One problem with this approach is that each pair is generated as many times as it has parents in the final lattice. To avoid duplication, it is necessary to verify for each pair if it was generated previously. The children are determined by finding the minimal extensions of  $X'$  for the current node that preserve the complete pair property using in the submatrix of  $R$  where the  $X'$  columns of the current node are removed. Details are found in Bordat's paper (Bordat, 1986).

Ganter's algorithm uses the characteristic vector representation of the  $X'$  sets to enumerate the  $X'$  sets of the lattice. The  $X$  parts are not treated in the following but this can be integrated easily. Given a vector  $X' = (x'_1, x'_2, \dots, x'_m)$  for a complete pair, it finds the next  $X'$  vector (with respect to the lexicographic order of the vectors) which is part of a complete pair. The ordered list of vectors produced in this manner is topologically sorted along the inclusion of  $X'$  sets.

**Algorithm 4** Enumerate  $X'$  sets of  $G$  (Ganter's algorithm)

```

1  A := (0, 0, ..., 0)
2  WHILE A ≠ (1, 1, ..., 1)

```

```

3     i := m
4     IF ai = 1
5         i := i - 1; goto 4
6     ELSE
7         ai = 1
8         FOR j := i + 1 to m DO
9             aj := 0
10        END FOR
11        X' := f(f(A))
12        FOR j = 1 à i - 1 DO
13            IF aj < xj Alors
14                i := i - 1; goto 4
15            END IF
16        END FOR;
17        A := X'
18    END IF
19 END WHILE

```

The algorithm will produce the X' sets in the following order on the example of Table 1:

$f(f(0, 0, 0, 0, 0, 0, 1, 0, 0)) = (0, 0, 0, 0, 0, 0, 1, 0, 0)$  giving {g}  
 $f(f(0, 0, 1, 0, 0, 0, 0, 0, 0)) = (0, 0, 1, 0, 0, 0, 0, 0, 0)$  giving {c}  
 $f(f(0, 0, 1, 0, 0, 1, 0, 0, 0)) = (0, 0, 1, 0, 0, 1, 0, 1, 0)$  giving {c, f, h}  
 $f(f(0, 1, 0, 0, 0, 0, 0, 0, 0)) = (0, 1, 0, 0, 0, 0, 0, 0, 0)$  giving {b}  
 $f(f(0, 1, 0, 0, 1, 0, 0, 0, 0)) = (0, 1, 0, 0, 1, 0, 1, 0, 0)$  giving {b, e, g}  
 $f(f(0, 1, 1, 0, 0, 0, 0, 0, 0)) = (0, 1, 1, 0, 0, 1, 0, 1, 0)$  giving {b, c, f, h}  
 $f(f(1, 0, 0, 0, 0, 0, 1, 0, 0)) = (1, 0, 0, 0, 0, 0, 0, 0, 0)$  giving {a}  
 $f(f(1, 0, 0, 0, 0, 0, 1, 0, 0)) = (1, 0, 0, 0, 0, 0, 1, 0, 1)$  giving {a, g, i}  
 $f(f(1, 0, 0, 1, 0, 0, 0, 0, 0)) = (1, 0, 0, 1, 0, 0, 1, 0, 1)$  giving {a, d, g, i}  
 $f(f(1, 0, 1, 0, 0, 0, 0, 0, 0)) = (1, 0, 1, 0, 0, 0, 0, 0, 0)$  giving {a, c}  
 $f(f(1, 0, 1, 0, 0, 0, 1, 0, 0)) = (1, 0, 1, 0, 0, 0, 1, 0, 1)$  giving {a, c, g, i}  
 $f(f(1, 0, 1, 0, 0, 1, 0, 0, 0)) = (1, 0, 1, 0, 0, 1, 0, 1, 0)$  giving {a, c, f, h}  
 $f(f(1, 1, 0, 0, 0, 0, 0, 0, 0)) = (1, 1, 1, 1, 1, 1, 1, 1, 1)$  giving E'.

A special case is made for the infimum with an empty feature set. This algorithm does not produce the Hasse diagram. However, we have extended it to produce the graph. We have not found any easy way to extend Ganter's algorithm to produce the graph because there is no direct relationship between the order of production of the lattice elements and the edges in the graph. The only useful aspect to this end is the ordering that is compatible with the partial order represented in the graph. Given a set X' in the list, we know the children must follow. The following algorithm proposed by Alaoui (1992) can be used after Algorithm 4 to produce the Hasse diagram. The combination named Ganter-Alaoui was used in the experiments.

**Algorithm 5** Build Hasse diagram using output of Ganter's algorithm

Input: the list X'<sub>i</sub>, i=1,...,||G|| of vectors produced from Algorithm 4.

```

1     FOR i:=1 to ||G|| DO
2         FOR j:= i+1 to ||G|| DO
3             IF X'i ⊂ X'j

```

```

4         IF there is no children  $Y'$  of  $X'_i$  such that  $Y' \subset X'_j$ 
5              $X'_j$  is a new children of  $X'_i$ 
6         END IF
7     END IF
8 END FOR
9 END FOR

```

The last algorithm we considered in the experiments is the Chain algorithm (Chain, 1969) which is a refinement of the Malgrange algorithm (Malgrange, 1962). This is more of a bottom-up algorithm where the lattice is built from the bottom going up level by level. The algorithm starts with level 1,  $L_1$ , containing the set of pairs  $(\{x\}, f(\{x\}))$  for all  $x$  in  $E$ . Iteratively, it tries to build new pairs in  $L_{k+1}$  by systematically combining two pairs of  $L_k$ . In this process there might be  $X'$  sets that jump from level  $L_k$  to level  $L_{k+1}$  and are eliminated from the previous level  $L_k$ .

**Algorithm 6** Build  $G$  (Chain algorithm)

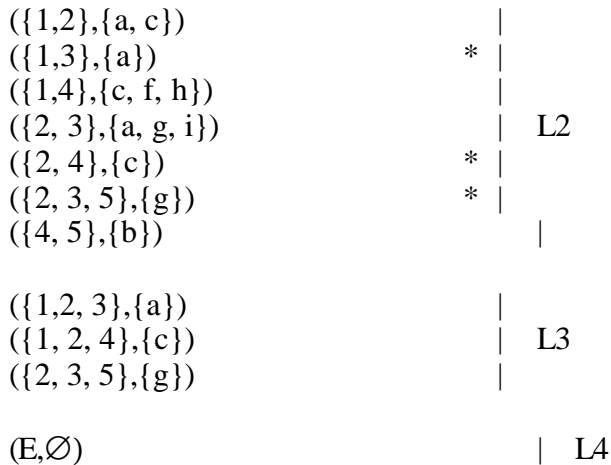
```

1   $L_1 := \{(\{x\}, f(\{x\}))\}$  for all  $x$  in  $E$ 
2   $k := 1$ 
3  WHILE  $\|L_k\| > 1$  DO
4       $L_{k+1} := \emptyset$ 
5      FOR ALL combination of pairs  $(X, X')$  and  $(Y, Y')$  in  $L_k$  DO
6           $Z := X' \cap Y'$ 
7          IF  $Z \neq \emptyset$  THEN
8              IF there is a pair  $(Z, Z') \in L_{k+1}$  THEN
9                   $Z := Z \cup Y'$ 
10             ELSE
11                  $L_{k+1} := L_{k+1} \cup \{X \cup Y, Z\}$ 
12             END IF
13             IF  $Z = X'$  THEN
14                 mark  $(X, X')$  in  $L_k$ 
15             END IF
16             IF  $Z = Y'$  THEN
17                 mark  $(Y, Y')$  in  $L_k$ 
18             END IF
19         END IF
20     END FOR
21      $k := k + 1$ 
22 END WHILE

```

The algorithm will produce the following pairs.  $G$  is the set of unmarked pairs (a "\*" in the following trace represents a marked pair).

$(\{1\}, \{a, c, f, h\})$		
$(\{2\}, \{a, c, g, i\})$		
$(\{3\}, \{a, d, g, i\})$		L1
$(\{4\}, \{b, c, f, h\})$		
$(\{5\}, \{b, e, g\})$		



Again the graph generation is not addressed but the manner in which the lattice is produced level by level is useful for the linking process. The following algorithm proposed by Alaoui (Alaoui, 1992) completes Algorithm 6 by adding the links between the pairs using the level structure inherent in Algorithm 7. The combination of Algorithm 6 and 7 is called Chein-Alaoui in the following.

**Algorithm 7** Build Hasse diagram using output of Chein's algorithm

```

1  FOR each level from 1 to last level DO
2    FOR each non marked pair (X,X') of Li DO
3      FOR each non marked pair (Y,Y') of Li+1 DO
4        IF Y' ⊂ X' THEN
5          (X,X') is a child of (Y,Y')
6        END IF
7      END FOR
8    END FOR
9  END FOR

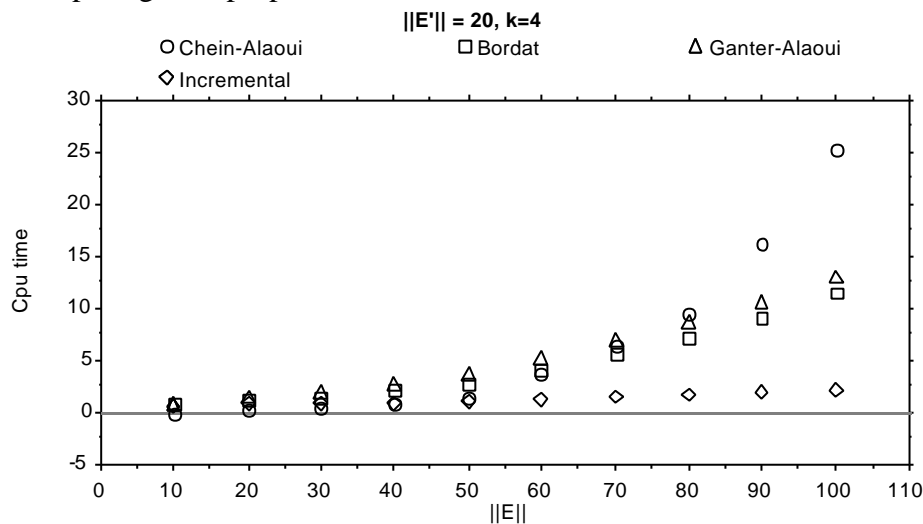
```

#### 4.2 Empirical comparison of the algorithms

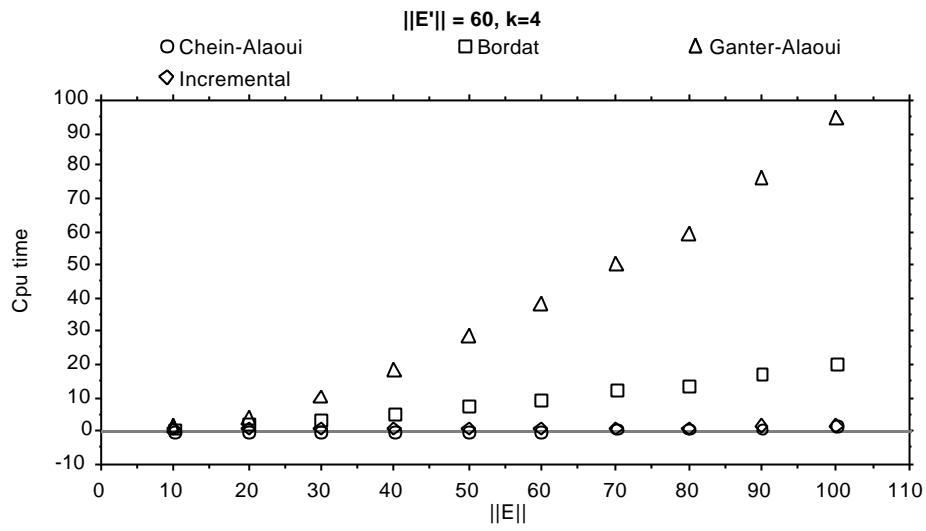
To get some understanding about the relative behavior of the incremental algorithms proposed in Section 3.1 and the batch algorithms of Section 4.1, implementations were done in C on a SUN SPARC1+™ workstation. For the incremental algorithms, the simplest and less efficient variant, Algorithm 1, was implemented. For the batch algorithms, we used Algorithm 3 from Bordat and the adaptations of Ganter's (Algorithms 4 and 5) and Chein's (Algorithms 6 and 7) algorithms that incorporate the Hasse diagram updating. These are called Ganter-Alaoui and Chein-Alaoui in the following. The same basic data structure was used for each algorithm in order minimize its effect on the performance.

Two types of comparison were performed. The first is based on simulating a binary relation using a uniform distribution and the second using real application data. Although, the uniform distribution is not realistic for most applications, it is used to gain some insight into the behavior of the algorithms with respect to different application parameters such as  $\|E\|$ ,  $\|E'\|$  and  $k$ , the mean value for  $\|f(\{x\})\|$ . In order to compare the incremental algorithm with the others, the CPU time measured for the incremental algorithm is the *total time* necessary to build the lattice by adding the instances one by one using the incremental process. Usually, it is expected that the batch algorithms will perform better under these circumstances. The main questions are: (i) how more efficient is the batch algorithm than the incremental one?, and (ii) under which circumstances is it better to use the incremental algorithm (to update the existing lattice) or to use a batch algorithm (to rebuild the whole lattice from scratch)?

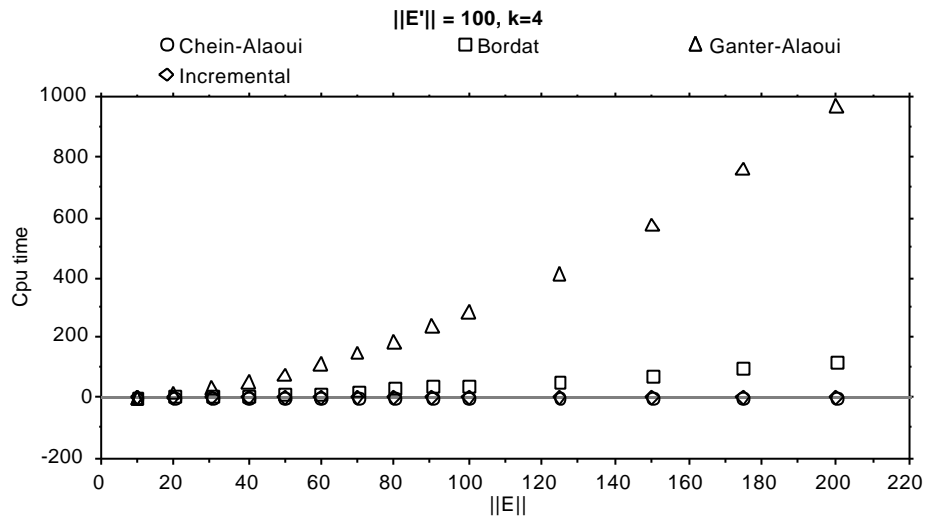
Figures 3 through 5 show the CPU time (in seconds) obtained for the simulations done with  $\|E'\| = 20, 60, 100$  and values of  $k = 4$  using the four algorithms. The number of instances was limited to 100 for  $\|E'\| = 20$  and 60, and 200 for  $\|E'\| = 100$ . The Chein-Alaoui algorithm shows the most varying behavior passing from the best under some circumstances to worst in other cases. For example in Figure 3, when  $\|E'\| < 40$ , it is the best algorithm. When  $\|E'\| > 80$ , it becomes the worst algorithm. These figures reveal the poor performance for the Ganter-Alaoui and Bordat algorithms. There is no circumstance where one of these is the best algorithm. In every case, the best algorithm is either Chein-Alaoui or the incremental algorithm. Many other combinations were tested and the same general behavior was observed. In order to see the relative behavior of the two algorithms, tests were made using larger values of  $\|E'\|$ . Figure 7 illustrates the typical relative behavior of the two algorithms. For small values of  $\|E'\|$ , Chein-Alaoui outperforms the incremental algorithm although the difference is relatively small because we are dealing with small values of  $\|E'\|$ . There is always a point, however, where the incremental algorithm becomes the best and, beyond that point, Chein-Alaoui becomes much worse. This crossover point depends on the density of the relation. When  $k/\|E'\|$  gets bigger, the crossover point becomes smaller. Looking at the pattern in Figures 3, 6 and 7, we can observe that for a fixed  $k = 4$ , the crossover point grows as  $\|E'\|$  becomes larger. Overall, the incremental algorithm has a much more stable behavior than Chein-Alaoui, which is an important consideration for applications. These results are quite surprising and counterintuitive since non-incremental algorithms usually outperform the incremental ones because they can exploit global properties of the data.



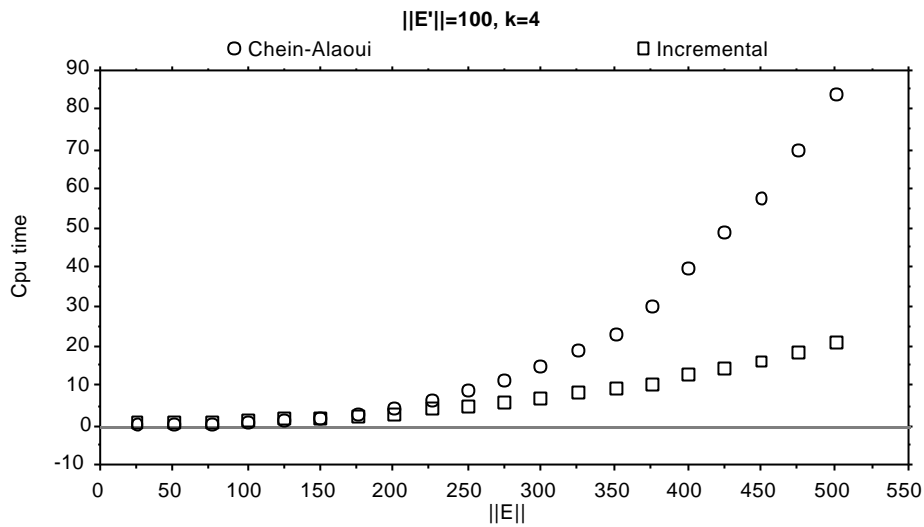
**Figure 3.** CPU time for simulation with uniform distribution.



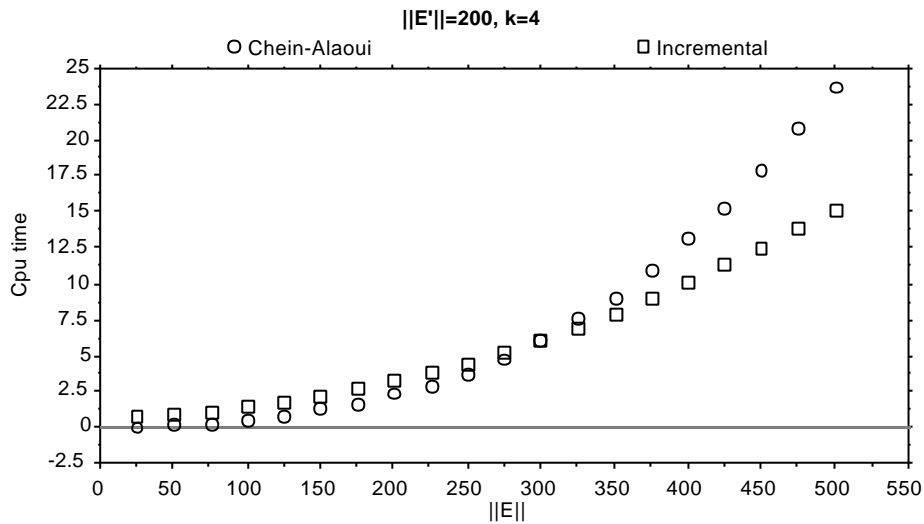
**Figure 4.** CPU time for simulation with uniform distribution.



**Figure 5.** CPU time for simulation with uniform distribution.



**Figure 6.** CPU time for simulation with uniform distribution.



**Figure 7.** CPU time for simulation with uniform distribution.

These general patterns were confirmed in a second series of tests done with real application data. To analyze the behavior of the algorithms, we used one data set of 500 instances from a software engineering reuse research project (Godin et al., 1995). Each instance represents a data element from a data repository and the features are controlled index terms used to identify the elements. Figures 8 a-c give the results. In Figure 8-a, the four algorithms are compared. The value of  $\|E\|$  was limited because of the large CPU time for the Ganter-Alaoui algorithm. The bad performance for Ganter-Alaoui and Bordat is again apparent. Figure 8-b compares Chein-Alaoui and the incremental algorithm. The relative behavior observed for the simulations is confirmed. Chein-Alaoui is better at the beginning but is dramatically inefficient for larger values of  $\|E\|$ . Figure 8-c shows the results for larger values of  $\|E\|$  confirming the tendency. In addition, Figure 8-c gives the average CPU time for each batch of instances without accumulating the CPU time for previous ones. Therefore this shows the cost of incrementally adding one new instance versus rebuilding the whole lattice. A noticeable fact is that even for small number of instances Chein-Alaoui is more expensive than the incremental update.

To see the empirical complexity of the incremental algorithm, Figure 9 gives a quadratic regression for CPU time for the incremental algorithm. This is consistent with the analysis of Algorithm 1 where the incremental

update process was shown to be  $O(\|E\|)$  and therefore the total time should grow quadratically in the worst case when the number of features has a constant upper bound. In this application, the average number of features per instance is 5.34 and the maximum is 12.

These results show without any doubt the advantages of the incremental algorithm when incrementality is an application requirement and it is even the most attractive algorithm for most applications when incrementality is not needed given its overall good performance and stable behavior. Notice also that we have used the simplest and less efficient variant, Algorithm 1, in the implementation. We should therefore expect that the advantages would be even stronger for Algorithm 2. To get an idea of the perspective gain, both Algorithms 1 and 2 have been implemented using Smalltalk-80<sup>TM</sup> on a Macintosh<sup>TM</sup> IIsi, and results of a test comparing these two algorithms on the data element example are shown in Figure 10. The advantage of Algorithm 2 is important and the difference grows with  $\|E\|$ . Although Algorithm 1 might be sufficient for small applications, Algorithm 2 should be preferred for larger ones.

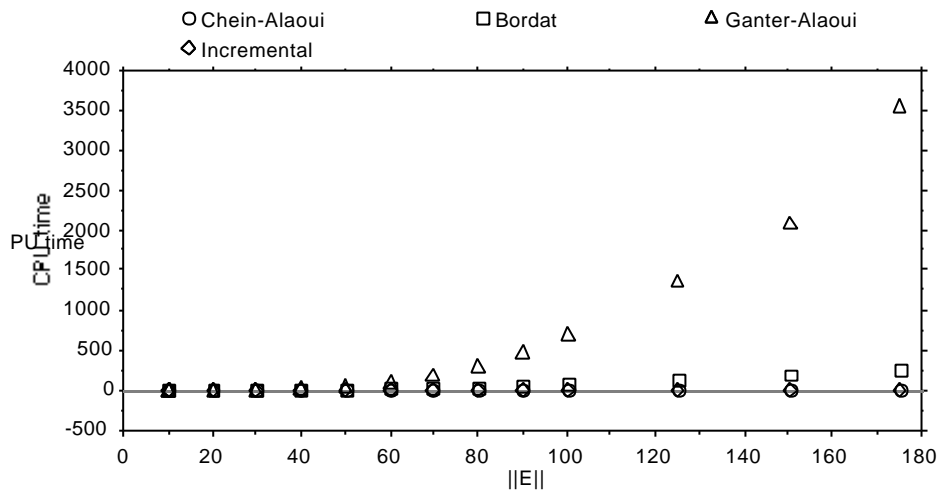


Figure 8-a. CPU time for data elements.

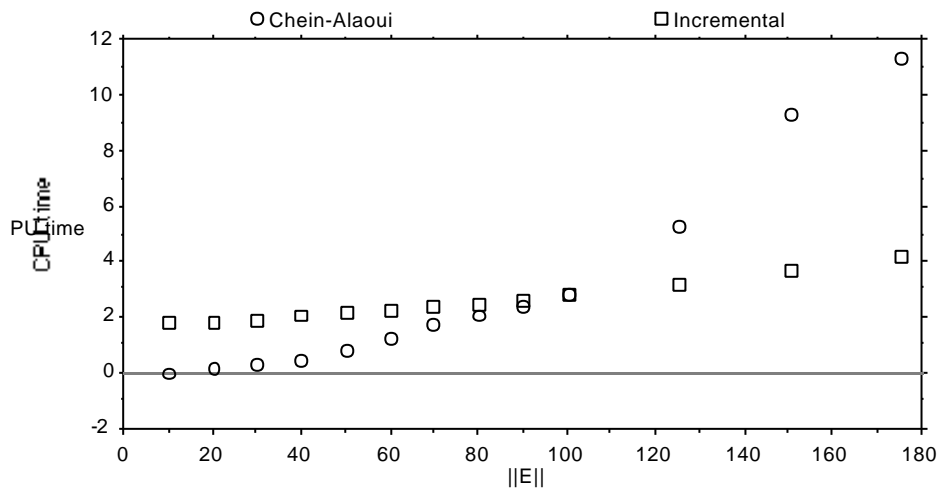


Figure 8-b. CPU time for data elements.

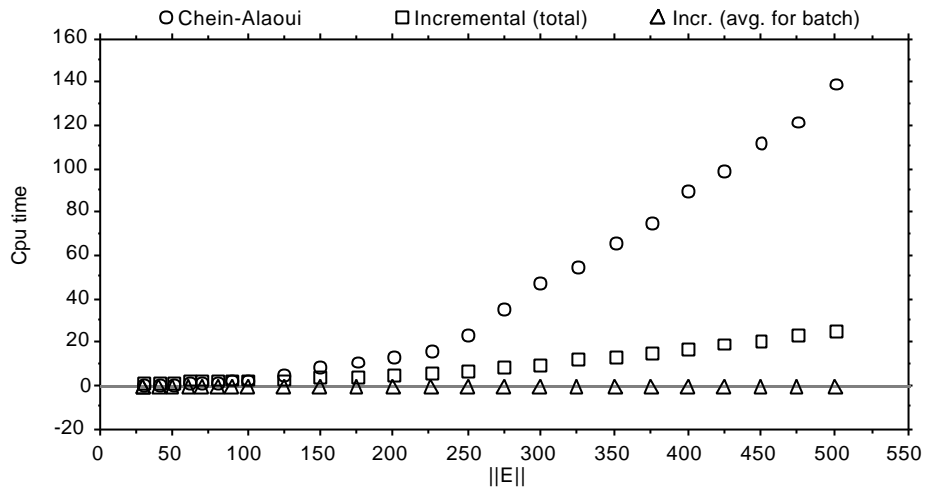


Figure 8-c. CPU time for data elements.

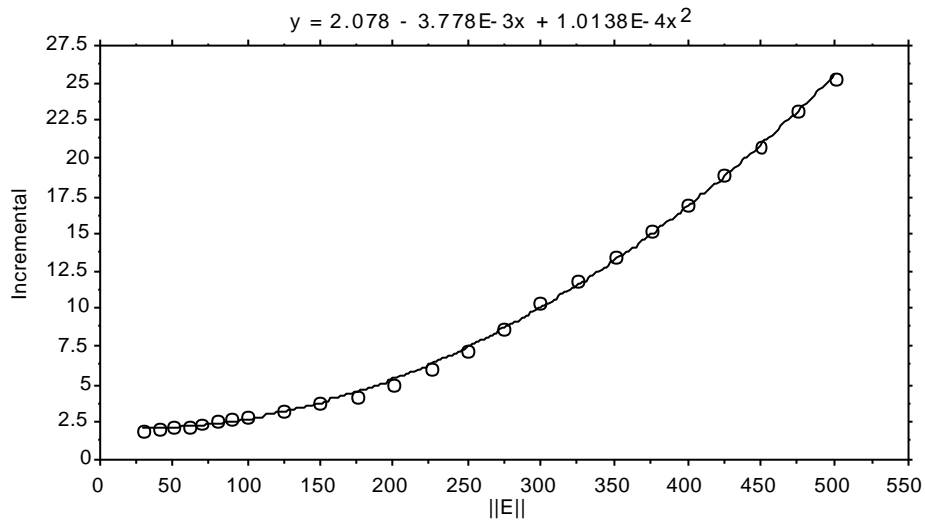


Figure 9. Regression for CPU time.

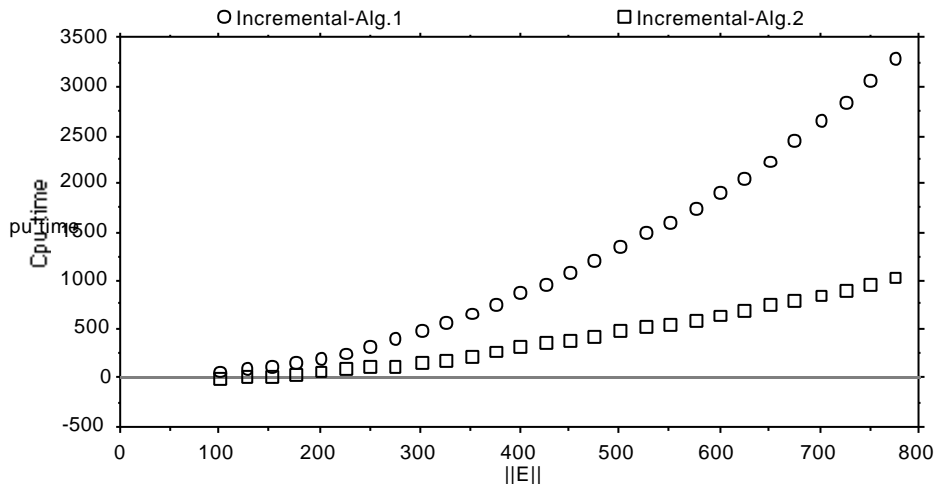


Figure 10. CPU time for data elements (Smalltalk-80 on Macintosh IIsi).

## CONCLUSION

Incremental algorithms for updating the Galois lattice and Hasse diagram for a binary relation have been proposed and analyzed. The resulting algorithms can be used for incremental concept formation. Different strategies for the incremental update are considered based on a characterization of the modifications implied by such an update. Empirical comparisons were done with three other batch algorithms to see if the incremental algorithm is worth considering. Surprisingly, when the total time for incremental generation is used, the simplest and less efficient variant of the incremental algorithms (algorithm 1) outperforms the batch algorithms in most cases. Only the Chein-Alaoui variant of Chein's algorithm is better in some cases. However, when the number of instances grows, there is always a point beyond which the incremental algorithm becomes better and the gap between the performance of the two algorithms gets larger as the number of instances grows. This point depends on the density of the relation. When the ratio between the average number of features per instance and the total number of instances gets bigger, the crossover point becomes smaller. When only the incremental update time is used, the incremental algorithm outperforms all the batch algorithms. Empirical evidence shows that, on the average, the incremental update is done in time proportional to the number of instances previously treated. Although the theoretical worst case is exponential, when we suppose there is a fixed upper bound  $K$  on the number of features related to an instance, which is usually the case in practical applications, the worst case analysis of the algorithm shows linear growth with respect to the number of instances.

This work shows that the incremental algorithms perform very well for the generation of Galois lattices compared to currently known batch algorithms but some questions still remain to be addressed in the future. First, there might be other better batch algorithms than the ones we used in our tests. In particular, there might be better ways to find the Hasse diagram edges for the Chein and Ganter algorithms. Second, we have only compared a basic incremental algorithm (Algorithm 1) with the batch algorithms. There are many variants and optimizations which are also described in the paper. One test shows that the optimization made in Algorithm 2 can lead to substantial savings. However, there are many other variations exposed in Section 3.3 that may show some improvements under certain circumstances. More elaborate tests could be performed for all these alternatives.

As argued in (Frawley et al., 1991), two important features for the practical feasibility of using automated learning methods are that they should be incremental and efficient. This work about incremental concept formation using Galois lattices is a contribution in that direction.

## ACKNOWLEDGMENTS

We would like to thank Martin Vanasse for his contribution in the experiments. This research has been partly supported by NSERC (the Natural Sciences and Engineering Research Council of Canada) under grants No OGP0041899, OGP0009184, FCAR Funds (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) and the software engineering IT Macroscopic Project, managed by the DMR Group Inc., and involving the Centre de Recherche en Informatique de Montréal.

## REFERENCES

- Alaoui, H. (1992). *Algorithmes de Manipulation du Treillis de Galois d'une Relation Binaire et Applications*. Masters Thesis, Université du Québec à Montréal.
- Barbut, M. & Monjardet, B. (1970). *Ordre et Classification. Algèbre et Combinatoire, Tome II*. Hachette.
- Bordat, J. P. (1986). Calcul Pratique du Treillis de Galois d'une Correspondance. *Mathématiques et Sciences Humaines*, **96**, 31-47.
- Carpineto, C. & Romano, G. (1993). GALOIS: An Order-Theoretic Approach to Conceptual Clustering. In *Proceedings of the Machine Learning Conference*, pp. 33-40.
- Chein, M. (1969). Algorithme de Recherche des Sous-Matrices Premières d'une Matrice. *Bull. Math. Soc. Sci. Math. R.S. Roumanie*, **13**, 21-25.
- Davey, B. A. & Priestley, H. A. (1992). *Introduction to Lattices and Order*. Cambridge: Cambridge University Press.
- Fay, G. (1975). An Algorithm for Finite Galois Connexions. *Journal of Computational Linguistic and Languages*, **10**, 99-123.
- Fisher, D. (1987). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, **2**, 139-172.
- Fisher, D. H., Pazzani, M. J. & Langley, P. (Ed.). (1991). *Concept Formation: Knowledge and Experience in Unsupervised Learning*. San Mateo, CA: Morgan Kaufmann.
- Frawley, W. J., Piatetsky-Shapiro, G. & Matheus, C. J. (1991). Knowledge Discovery in Databases: An Overview. In G. Piatetsky-Shapiro & W. J. Frawley (Eds.), *Knowledge Discovery in Databases*, (pp. 1-27). Menlo Park, CA: AAAI Press/ The MIT Press.
- Ganter, B. (1984). *Two Basic Algorithms in Concept Analysis*. Preprint 831, Technische Hochschule Darmstadt.
- Ganter, B. & Wille, R. (1989). Conceptual Scaling. In F. Roberts (Eds.), *Applications of Combinatorics and Graph Theory to the Biological and Social Sciences*, (pp. 139-167). New York: Springer-Verlag.
- Gennari, J. H., Langley, P. & Fisher, D. (1990). Models of Incremental Concept Formation. In J. Carbonell (Eds.), *Machine Learning: Paradigms and Methods*, (pp. 11-62). Amsterdam, The Netherlands: MIT Press.
- Godin, R. (1986). *L'utilisation de Treillis pour l'Accès aux Systèmes d'Information*. Ph.D. Thesis, Université de Montréal.
- Godin, R. (1989). Complexité de Structures de Treillis. *Annales des Sciences Mathématiques du Québec*, **13**(1), 19-38.
- Godin, R. & Mili, H. (1993). Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93)*, A. Paepcke (Ed.), Washington, DC: ACM Press, pp. 394-410.
- Godin, R., Mineau, G., Missaoui, R., St-Germain, M. & Faraj, N. (1995). Applying Concept Formation Methods to Software Reuse. *International Journal of Knowledge Engineering and Software Engineering*, To appear.
- Godin, R. & Missaoui, R. (1994). An Incremental Concept Formation Approach for Learning from Databases. *Theoretical Computer Science, Special Issue on Formal Methods in Databases and Software Engineering*, **133**, 387-419.
- Godin, R., Missaoui, R. & Alaoui, H. (1991). Learning Algorithms Using a Galois Lattice Structure. In *Proceedings of the Third International Conference on Tools for Artificial Intelligence*, S. Lee, B. Wah, N. G. Bourbakis, & W. T. Tsai (Ed.), San Jose, Calif.: IEEE Computer Society Press, pp. 22-29.
- Godin, R., Missaoui, R. & April, A. (1993). Experimental Comparison of Navigation in a Galois Lattice with Conventional Information Retrieval Methods. *International Journal of Man-Machine Studies*, **38**, 747-767.
- Godin, R., Saunders, E. & Gecsei, J. (1986). Lattice Model of Browsible Data Spaces. *Information Sciences*, **40**, 89-116.

- Guénoche, A. (1990). Construction du Treillis de Galois d'une Relation Binaire. *Mathématiques et Sciences Humaines*, **109**, 41-53.
- Guigues, J. L. & Duquenne, V. (1986). Familles Minimales d'Implications Informatives Résultant d'un Tableau de Données Binaires. *Mathématiques et Sciences Humaines*, **95**, 5-18.
- Krone, M. & Snelting, G. (1994). On The Inference of Configuration Structures from Source Code. In *Proceedings of the 16th International Conference on Software Engineering*, IEEE Computer Society Press,
- Lebowitz, M. (1987). Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning*, **2**, 103-138.
- Malgrange, Y. (1962). Recherche des Sous-Matrices Premières d'une Matrice à Coefficients Binaires; Applications à Certains Problème de Graphes. In *Proceedings of the Deuxième Congrès de l'AFCALTI*, Gauthier-Villars, pp. 231-242.
- Mephu Nguifo, E. (1993). *Concevoir une Abstraction à Partir de Ressemblances*. Doctorat Thesis, Montpellier II - Sciences et Techniques du Languedoc.
- Mineau, G. W. & Godin, R. (1994). Automatic Structuring of Knowledge Bases by Conceptual Clustering. *IEEE Transactions on Knowledge and Data Engineering*, To appear.
- Missaoui, R. & Godin, R. (1994). Search for Concepts and Dependencies in Databases. In *Proceedings of the International Workshop on Rough Sets and Knowledge Discovery*, Banff, Alberta: to appear in Workshop in Computing Series, Springer-Verlag,
- Norris, E. M. (1978). An Algorithm for Computing the Maximal Rectangles in a Binary Relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, **23**(2), 243-250.
- Oosthuizen, G. D., Bekker, C. & Avenant, C. (1992). Managing Classes in Very Large Class Repositories. In *Proceedings of the Tools*, Paris: pp. 625-633.
- Ouaggag, H., Godin, R., Missaoui, R. & Mili, H. (1993). *Traitement Parallèle dans le Processus de Classification pour le Repérage de l'Information*. Projet ALEX: Centre ATO-CI, Université du Québec à Montréal.
- Pawlak, Z. (1991). *Rough Sets - Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers.
- Wille, R. (1982). Restructuring Lattice Theory: an Approach Based on Hierarchies of Concepts. In I. Rival (Eds.), *Ordered Sets*, (pp. 445-470). Dordrecht-Boston: Reidel.
- Wille, R. (1984). Line Diagrams of Hierarchical Concept Systems. *Int. Classif.*, **11**(2), 77-86.
- Wille, R. (1992). Concept Lattices and Conceptual Knowledge Systems. *Computers Mth. Applic.*, **23**(6-9), 493-515.