Learning Algorithms using a Galois Lattice Structure

Robert Godin, Rokia Missaoui, Hassan Alaoui

Département de Mathématiques et d'Informatique
Université du Québec à Montréal
Montréal (Québec), Canada, H3C 3P8

Abstract

The Galois lattice of a binary relation between a set of objects and a set of properties may be used to discover concepts and rules related to the objects and their properties. An incremental algorithm for updating the Galois lattice is proposed where new objects may be dynamically added by modifying the existing lattice. A large experimental application reveals that adding a new object may be done in time proportional to the number of objects on the average. When there is a fixed upper bound on the number of properties related to an object, which is the case in practical applications, the worst case analysis of the algorithm confirms the experimental observations of linear growth with respect to the number of objects. Algorithms for generating rules from the lattice are also given.

1 Introduction

The Galois lattice of a binary relation between a set of objects and a set of properties may be used to discover concepts and rules related to the objects and their properties. The Galois lattice is a form of conceptual classification of the objects or concept hierarchy where each node represents a subset of objects with their common properties. These nodes can be considered as concepts underlying the basic data and the Hasse diagram of the lattice as a generalization/specialization relationship between the concepts [20]. Therefore, building the lattice and Hasse diagram corresponding to a set of objects, each described by some properties, can be used as an effective tool for symbolic data analysis and knowledge acquisition [4, 13, 20]. Since the lattice and Hasse diagram form a classification, they can also be used as a browsing space for information retrieval [7, 9, 10]. In this context, the Galois lattice is generated from the usual binary relationship between documents and indexing terms. The Hasse diagram is used as the basic structure supporting a browsing interface which permits gradual enlargement or refinement of the user's query using document and term subsets present in the lattice.

Generating the lattice and Hasse diagram is a difficult problem. Many algorithms have been proposed for generating the elements of the lattice but only Bordat's algorithm generates the Hasse diagram [11] which is necessary for the considered applications. Furthermore, none of these algorithms is incremental and all the objects and their properties have to be known in advance to build the lattice. If new objects are added, the lattice will have to be rebuilt from scratch. We therefore propose an incremental algorithm for generating the lattice and Hasse diagram where new objects can be gradually integrated into the lattice as they appear, one at a time. Details on its theoretical and empirical complexity will be given. The task of inducing a concept hierarchy in an incremental manner is labeled incremental concept formation or simply concept formation in [5] and is a fundamental process of human learning. Concept formation is similar to conceptual clustering which also builds concept hierarchies [16]. However, the former approach is partially distinguished from the latter by the fact that the learning is incremental. Concept formation falls into the category of unsupervised learning also called learning from observation [2] since the concepts to learn are not predetermined by a teacher and the instances are not pre-classified with respect to these concepts.

The lattice may also be used to discover implication rules that apply to the data [12, 13]. Algorithms to generate these rules automatically from the lattice will be given. As opposed to explanation-based learning (EBL) methods [3], this approach falls into the class of empirical learning [15] since no background knowledge is used to derive the concepts and rules.

Section two recalls basic definitions related to the concept of Galois lattice. Section three presents an incremental algorithm. Details from experimental applications and performance statistics are also presented. Section four gives algorithms for discovering rules from the lattice and reasoning about them.

2 Basic Definitions

This section recalls basic definitions related to the concept of Galois lattice for a binary relation. More details are found in [1]. Given two finite sets E and E', and a binary relation R (Figure 1) between these two sets,
there is a unique Galois lattice corresponding to this binary relation (Figure 2).

\[
\begin{array}{cccccccccccc}
R & a & b & c & d & e & f & g & h & i \\
\hline
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
4 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
5 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]

Figure 1. Matrix representation of a binary relation R.

Each element of the lattice is a couple, noted \((X,X')\), composed of a set \(X \in \mathcal{P}(E)\) and a set \(X' \in \mathcal{P}(E')\). Each couple must be a complete couple as defined in the following. A couple \((X,X')\) from \(\mathcal{P}(E) \times \mathcal{P}(E')\) is complete with respect to \(R\) 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 \(f(X) = \{x \in E \mid \forall x' \in X', xRx'\}\).

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 couples \([l]\) with the following partial order:

\[C_1 = (X_1,X'_1) \text{ and } C_2 = (X_2,X'_2) \text{, } C_1 \leq C_2 \iff X'_1 \subseteq X'_2.\]

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

\[X'_1 \subseteq X'_2 \iff X_2 \subseteq X_1\]

and therefore,

\[C_1 \leq C_2 \iff X_2 \subseteq X_1.\]

The partial order is used to generate the 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. By convention, when drawing a Hasse diagram, the edge direction is downwards. Given \(C\), a set of elements from \(G\), \(\inf(C)\) and \(\sup(C)\) will denote respectively the greatest lower bound and lowest upper bound of the elements in \(C\).

The set \(E\) may be identified to some set of objects and the set \(E'\) to properties of these objects. The properties in the above definitions are atomic elements. However, the lattice may be generalized to incorporate attribute-value pairs as properties of the objects [10] as is the case for most concept formation methods [5].

### 3 Incremental Update Algorithm

In many applications, it is necessary to maintain not only the elements of \(G\) but also the Hasse diagram of the lattice. It is also important to have the possibility of dynamically adding a new element, say \(x^*\), to \(E\), which is related to a set of elements, from \(E'\) and possibly new elements not in \(E'\), by modifying the existing lattice and Hasse diagram without having to regenerate it from scratch. An incremental algorithm for performing this task will be presented. In the following, the notation \(E^*, E'^*, G^*, R^*, f^*, f^*\) will be used to represent the corresponding items, \(E, E', G, R, f, f\), after adding the new element \(x^*\). The set of elements related to \(x^*\) is therefore denoted by \(f^*(\{x^*\})\).

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 3 which represents the new lattice \(G^*\).

The items in bold correspond to new couples and edges.

In the following, the term node will be used to refer to the representation of a couple in the Hasse diagram. As will be explained, the new lattice \(G^*\) may be obtained by taking the lattice \(G\), modifying the \(X\) sets of some nodes,
adding new nodes and edges and deleting some edges. Deleted edges are marked with an "X". The following subsection gives details on the lattice update process. The concepts presented are important to understand the proposed algorithm.

3.1 Informal characterization of the update process.

By definition of a complete couple \((X,X')\), an important property of the lattice is that any set \(X\) is equal to \(f(X)\), the intersection of the sets \(f'(x')\) for \(x'\) in \(X'\), and symmetrically any set \(X'\) is equal to \(f(X)\), the intersection of the sets \(f'(x)\) for \(x\) in \(X\). The converse is also true: any such intersection of sets \(f'(x')\) or \(f'(x)\) must be in the lattice for one couple. Consider the \(X'\) sets in \(G^*\). Every \(X'\) set present in \(G\), except possibly \(E'\), will also be in \(G^*\) because the sets \(f'(x)\) do not change for \(x\) in \(E\) (nodes #1 to #14 in Figure 3). \(E'\) may not appear in \(G^*\) because there might be elements related to \(x^*\) which are not in \(E\). In addition there may be new sets \(X'=f(X)\) for the \(X'\) sets containing the new element \(x^*\) (nodes #15 to #19 in Figure 3). The basic strategy used for updating the lattice will consist into keeping the nodes in \(G\) by updating some of the \(X'\) sets and creating new nodes for new \(X'\) sets. In that perspective, we propose the following classification of nodes in \(G^*\). Let \((X,X')\) be a node of \(G^*\) and \((Y,Y')\) a node of \(G\).

**New nodes** (New): nodes for which \((X,X')\) does not exist in \(G\) (\#15-\#19).

**Old nodes** (Old): nodes for which \((X,X')\) are the same as in \(G\) (\#2,\#5, \#6 to \#14).

**Modified nodes** (Mod): nodes for which \((X,X')\) is in \(G\) but the \(X\) set is changed in \(G^*\) (\#1, \#3, \#4).

The \(X\) set of Mod nodes is always equal to \(Y \cup \{x^*\}\) where \((Y,Y')\) is in \(G\) and \((X,X')\) is in \(G\). The corresponding old node \((Y,Y')\) has the property: \(Y' \subseteq f^*(x^*)\). Furthermore, every node \((Y,Y')\) in \(G\) with that property will produce a new node \((X,X')\) in \(G^*\). The other nodes in \(G\) will produce the Old nodes of \(G^*\).

Producing the new nodes is more complicated. Any new \(X'\) set in \(G^*\) (except possibly for \(E^*\) and \(f^*(x^*)\)) when \(E' \neq E^*\) will have to be the result of intersecting \(f^*(x^*)\) with some \(Y'\) set already present in the lattice \(G\). There may be many nodes in \(G\) which give a particular new intersection in this manner. For example, in Figure 3, the new \(X'\) set, \(\{1\}\), in the new node #15, can be formed by intersecting \(f^*(x^*)\) with \(\{b,c,f,i\}\) with the \(Y'\) set of node \#7, \(\{a,g,i\}\), or the \(Y'\) set of node \#12, \(\{a,d,g,i\}\). However, there is only one of these, called generator node which will be a child of the New node, \#7, in this example. This node is the smallest Old node which produces the intersection and we will define the generator as such. In Figure 3, the generator nodes for the new nodes #15, #16, #17, #18, #19 are respectively #7, #11, #8, #13, #14. We may therefore partition the Old nodes into two subclasses: generator Old nodes (OldGen) and non generator Old nodes (OldNGen).

**Generator node** (OldGen): the node \((Y,Y')\) is the generator of the New node \((X,X')\) in \(G^*\) if and only if it is equal to \(\inf((Z,Z') \cap f^*(x^*))\). This one-to-one correspondence between New nodes and the OldGen nodes from \(G\) suggest that we may generate the New nodes by finding the OldGen nodes using this characterization of the generators. Formal proofs for all the preceding facts may be found in [8].

The process of linking nodes is a bit complicated but the complexity of this task is not the major factor because it is usually limited to a small subset of the nodes. More details are found in [8].

3.2 Incremental update algorithm

This section gives a basic algorithm and proposes some refinements for incrementally updating the Galois lattice based on the preceding characterization. Details are given in [8]. For a node \(H\), \(X(H)\) and \(X'(H)\) will denote respectively the first and second component of the complete couple. \(Sup(G)\) and \(inf(G)\) will denote respectively the supremum and infimum of the lattice.

**Algorithm 1** Add

\((x^*: new\ object; \ f^*(x^*): elements\ related\ to\ x^*\ by\ R)\);

**BEGIN**

1. Build \(Sup(G^*)\) by adding new properties to \(Sup(G)\) if \(X(Sup(G)) = \emptyset\) or else by creating a new node;

2. (Class nodes in buckets with same cardinality of the \(X'\) sets) \(C[i]\) <- \(\{H: \|X'(H)\| = i\}\);

3. \(C[i]\) <- \emptyset; (Initialize the \(C\) sets)

4. (Treat each bucket in ascending cardinality order)

5. FOR \(i: 0\ TO\ maximum\ cardinality\ DO\)

6. FOR each node \(H\) in \(C[i]\)

7. IF \(X'(H) \subseteq f^*(x^*)\) THEN \{Mod node\}

8. Add \(x^*\) to \(X(H)\);

9. Add \(H\) to \(C[i]\);

10. IF \(X'(H) = f^*(x^*)\) THEN exit algorithm

11. ELSE \{Old node\}

12. Int <- \(X'(H) \cap f^*(x^*)\);

13. IF Int does not appear in \(C[\|Int\|]\) THEN \(H\) is an OldGen node

14. Create New node \(H_n = (X(H) \cup \{x^*\}, int)\) and add \(H_n\) to \(C[\|Int\|]\);

15. Add edge \(H_n \rightarrow H\);

16. Modify edges with respect to \(H_n\) by going through the nodes in \(C[\|Int\|]\) with smaller cardinality than \(H_n\);

17. IF Int does not appear in \(C'_[\|Int\|]\) THEN exit algorithm

18. END IF

19. END IF

20. END FOR

21. END FOR

END \{Add\}

The lattice is initialized with one element: \((\emptyset, \emptyset)\). This means that \(E=E'=\emptyset\). The algorithm supposes that \(E\) and \(E'\) grow 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 process because adjusting the supremum of the lattice by adding new elements from $P^*(\{x^*\})$ would not be necessary. This initial step (line 1) simplifies the rest of the algorithm by simulating the case $E' = E^*$. The basic idea is to generate $G^*$ from $G$, changing the $X$ sets and links of the nodes already in $G$ and adding some New nodes for the new $X'$ sets and linking them. The New nodes are generated by finding the generator nodes using the characterization given in the preceding section. The function of line 1 is to take into account the case when $E' \neq E^*$ by adjusting sup($G$). Line 2 classifies the nodes into buckets with the same $II X' II$. The main loop (lines 4-21) iterates on every node in ascending $II X' II$. New nodes are obtained by systematically trying to generate a new intersection from each couple $(Y, Y')$ already in the lattice. The new intersection is formed by computing $Y' \cap P^*(\{x^*\})$ (line 12). Verifying that this intersection is not already present is done by looking in the sets already encountered which are subsets of $P^*(\{x^*\})$ (line 13). These sets are kept in $C$ (line 9, line 14). This is valid because the nodes are treated in ascending cardinality of the $X'$ sets. Furthermore, the first node encountered which gives a new intersection is the generator of the New node because it is necessarily the infimum. Thus, the $X$ set of the New node is computed by adding $x^*$ to the generator's $X$ set (line 14). Also, there is automatically an edge from the New node to the generator (line 15). When a New node is added, some edges have to be added from Mod or other New nodes to the New node. The candidates are necessarily in the $C$ sets since their $X'$ set must be a subset of $P^*(\{x^*\})$. These parents of the new node are determined by examining the nodes in $C$ (line 16) and testing if the $X'$ sets are subsets of the $X'$ set of the New node and verifying that no child of the potential parent has this property. It is necessary to eliminate an edge between the new parent and the generator when there is such an edge. Lines 7-10 process the Mod nodes and the rest of the treatment is skipped for these nodes because they cannot be generators.

Although in the worst case, the lattice may grow exponentially, the growth is linearly bounded with respect to $II E II$, when there is a fixed upper bound $K$ on $llf(\{x\})ll$, which is the case in practical applications [10]: $II G II \leq 2^K II E II$. Experimental applications and theoretical results based on uniform distribution hypothesis show that the growth factor obtained is far less than the $2^K$ bound. In each experimental application (see Table 1), we observe that: $II G II \leq k II E II$, where $k$ is the mean value for $II f(\{x\})ll$. More details are found in [6, 10]. The time complexity of iterating on the couples 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 a bit tedious, the nodes affected are limited and this part is only done when a generator node is encountered. Since $II G II \leq 2^K II E II$ and the $II C' II$ is bounded by $2^K$, the whole time complexity is: $O(II G II 2^K) = O(22^K II E II)$, which is $O(II E II)$ for a fixed $K$.

Algorithm 1 iterates on almost every node of $G$ and it is possible to do the work by looking at a limited subset of $G$: the nodes that correspond to generators and Mod nodes. Except maybe for $inf(G)$, all these nodes have at least one element in their $X'$ set which is in $P^*(\{x^*\})$. Limiting the nodes to be processed may be accomplished by using a depth-first search of the graph and cutting paths when there is nothing in common between the $X'$ set and $P^*(\{x^*\})$. A further refinement is to keep for each $x'$ in $E'$ a pointer $P_{x'}$ on the smallest node containing $x'$ and using these pointers as entry points for a top-down depth-first search starting with every $x'$ in $P^*(\{x^*\})$. This guarantees that any node encountered will have at least $x'$ in common with $P^*(\{x^*\})$. Some minor modifications have to be introduced for maintaining these pointers. Even though there is an important gain with these refinements, the overall complexity remains $O(II E II)$ for a fixed $K$. More details are found in [8].

### 3.3 Experiments

Experiments have been conducted to evaluate empirically the behavior of the lattice and algorithm with respect to space and processing time requirements. Algorithm 1 has been implemented in Pascal on a Vax11/750 under VMS along with the refinements proposed. The lattice is kept on disk for persistency and each update is materialized on disk with no caching of the nodes in main memory from one update to another. Several applications were tested in the domain of information retrieval where the set $E$ is a collection of documents, the set $E'$ is a vocabulary of index terms and $R$ is the indexing relationship between documents and terms. The first three applications are described in more details in [10]. Table 1 gives, for each application, the values of $II G II$, $II E II$, $II G II / II E II$ and the mean number, $k$, of elements from $E'$ (here index terms) assigned to each element from $E$ (here documents).

<table>
<thead>
<tr>
<th>Experiment</th>
<th>$II G II$</th>
<th>$II E II$</th>
<th>$II G II / II E II$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical reports</td>
<td>23,128</td>
<td>3006</td>
<td>7.7</td>
<td>11.1</td>
</tr>
<tr>
<td>Videotex database</td>
<td>450</td>
<td>317</td>
<td>1.4</td>
<td>4.7</td>
</tr>
<tr>
<td>Course catalog</td>
<td>237</td>
<td>106</td>
<td>2.2</td>
<td>4.8</td>
</tr>
<tr>
<td>ONF animation films</td>
<td>325</td>
<td>113</td>
<td>2.9</td>
<td>6.5</td>
</tr>
</tbody>
</table>

The largest and most significant experiment was performed on a database of 3006 technical reports (set $E$). Each report was indexed by a set of terms (from $E'$) with
a total of 9270 different index terms. The binary relation \( R \) was the indexing relationship between documents and index terms. The mean number of terms per document was 11.1. The resulting lattice was composed of 23,128 nodes. The mean number of parents (or children) per node was 3.25. Statistics were measured for batches of approximately 100 documents each. Figure 4 shows the evolution of the number of disk accesses for each batch. We observe an approximately linear growth in the number of accesses with respect to \( \|E\| \) with a correlation coefficient equal to 0.87. The major factor to be considered as expected in the analysis of the previous section is the number of nodes to be read in order to make the update. We know that this number is bounded by the number of nodes in the whole lattice which is \( O(\|E\|) \) as previously shown and each node usually requires one disk access. The evolution of \( \|GI\|/\|EI\| \) is given in Figure 6 which reveals that the growth factor of \( \|GI\| \) with respect to \( \|EI\| \) is fairly stable and far less than the \( 2^{1.1} \) upper bound. Figure 5 gives the CPU time measured for each batch of updates. As for disk accesses, the growth is approximately linear.

![Figure 4. Number of disk accesses for each batch of documents.](image)

![Figure 5. Cpu time (minutes) for each batch of documents.](image)

4 Learning Rules from the Galois Lattice

In addition to be a classification tool for defining concepts, the Galois lattice may be exploited to discover implication rules. As explained earlier, each concept \((X, Y)\) is represented by a node of the lattice and defined by its intensity \( X' \) (i.e. properties) and its extension \( X \) (i.e. the objects that share the same intension). The general format of an implication rule is: \( P \rightarrow Q \) where \( P \) and \( Q \) are comparable in that they both represent either a set of objects or a set of properties. Two kinds of implication rules can be considered: IRDs in which \( P \) and \( Q \) are sets of properties and IROs in which \( P \) and \( Q \) concern sets of objects. In this section, we show how the Galois lattice structure may be exploited to discover implication rules.

Then, we adapt the work done on reasoning about functional dependencies [14, 19] to implication rules.

For the purposes of illustration, the lattice shown in figure 3 will be used. It corresponds to the binary relation between the set \( E \) of six objects and the set \( E' \) of nine properties.

The meaning of the properties is as follows:
- \( a = \text{IsAGraduateStudent} \)
- \( b = \text{IsEligibleToPositionUniversity} \)
- \( c = \text{HasAtLeast}^{3} \text{B'Score} \)
- \( d = \text{HasAResearchSubject} \)
- \( e = \text{IsTeaching} \)
- \( f = \text{Excellent} \)
- \( g = \text{OptionManagement} \)
- \( h = \text{HasAScholarship} \)
- \( i = \text{HasAccessToLaboratory}^{X'} \).

4.1 Implication rule determination

In [12], the authors deal with the problem of finding a minimal set of implications rules from the Galois lattice structure. However, they do not propose any complete algorithm to extract these rules from this structure. In the following, we define formally the implication rules and show how to compute them.

The implication rule IRD between two subsets \( X' \) and \( Y' \) of \( P(E') \), denoted by \( X' \rightarrow Y' \), means that if a non-empty set of objects is defined by the properties found in \( X' \), then it is necessarily defined by the properties contained in \( Y' \). In other words, the properties found in \( X' \) subsume the properties found in \( Y' \).
Example: IsExcellent --> HasAtLeastBScore. The excellence of a student subsumes a score at least equal to B. X' --> Y' is said to be elementary if \|X'\| = 1 and Y' \subseteq X'. X' and \neg \exists Z' \in X' such that Z' --> Y'. X' --> Y' is said to be trivial if Y' \subseteq X'.

Proposition 1

X' --> Y' \iff \{(W, W') = \inf \{ (Z, Z') \in G \mid X' \subseteq Z' \text{ and } Z \neq \emptyset \} \rightarrow Y' \subseteq W' \}

In other words, X' --> Y' if and only if the first encountered node (during a breadth-first search of the lattice) containing X' as a part of its intension is also described by Y'.

The implication rule IRO between X and Y, denoted by X --> Y, means that if a non-empty set of properties is associated with the objects in X, then it is also associated with the objects in Y.

Example:

\{(dog) \rightarrow \{bulldog, poodle\}\} means that bulldogs and poodles have at least the properties attached to dogs, i.e. that these animals are dogs with possibly additional properties.

The concepts of elementary and trivial implication rule are similar to those for IRDs. Proposition 2 is the dual of proposition 1 for IROs rules.

Proposition 2

X --> Y \iff \{(W, W') = \sup \{ (Z, Z') \in G \mid \neg \exists Z' \in Z \text{ and } Z \neq \emptyset \} \rightarrow Y \subseteq W \}

In other words, X --> Y if and only if the first encountered node (during a bottom-up search of the lattice) containing X as a part of its extension is also described by Y.

4.2. Reasoning about Implication Rules

For ease of exposition, we define the concepts of this section only for IRDs. However, the definitions and algorithms can be adapted without difficulty to IROs. In the following we use P, Q, R, ..Z to denote sets of properties, while we use lower-case letters to name atomic properties. The notation abc is a simplification of the notation \{a, b, c\}.

The equivalence class of P for IROs is the set of alternative properties P1, P2, Pn which subsume Q. This concept may be useful for abductive reasoning.

Definitions

The closure P+ of a set P of properties according to a particular lattice structure (or a set of IRDs) is a set of properties containing P defined by:

P+ = P \cup \{(Q \mid Z \subseteq P^+ \text{ and } Z \rightarrow Q \in \Sigma)\} where \Sigma is a set of IRDs (see algorithm 4).

A minimal cover Zm for a set Z of IRDs is a subset of Z such that for no P \rightarrow R in Zm is Zm - \{P \rightarrow R\} equivalent to Zm. Two sets \Sigma1 and \Sigma2 are said to be equivalent if they have a same closure [19]. As in the case of functional dependencies, the minimal cover for IRDs may not be unique.

The determinate set P' for Q is the set of alternative properties P1, P2, Pn which subsume Q. This concept may be useful for abductive reasoning.

Algorithm 4 uses the Galois lattice as input to compute the closure of P. Since it is well known that P --> R if and only if P \subseteq P+ holds, the closure may be useful to check if P --> R holds.

Algorithm 4: Determination of the closure of P using G

\textbf{Input:} P (a subset of E) and the lattice G

\textbf{Output:} the closure P+

\begin{algorithm}
\textbf{Procedure} Closure (P, G)
\begin{algorithmic}
\STATE Find (Z, \inf ((X, X') E G \mid P \subseteq Z' \text{ and } X \neq \emptyset) \rightarrow P^+ : = Z')
\end{algorithmic}
\end{algorithm}

The closure of P can also be used to detect equivalent sets of properties and determine the equivalence class for a particular P.

The equivalence class of P \in P(E) with respect to the relation "closure", denoted by [P]+, is defined by:

\[ [P]^+ = \{Q \in P(E) \mid Q^+ = P \} \]

P and Q are then equivalent if and only if [P] = [Q].

Example: the properties (e) and (bg) present in node # are equivalent since [e] = [bg] = [ebg]4.
Another way to compute the closure of P consists into using a set $Z$ of IRDs which can itself be computed by means of algorithm 5.

Algorithm 5: Determination of the set $\Sigma$ of IRDs

Input: the lattice $G$
Output: a set $\Sigma$ of IRDs: $P_i \rightarrow P_j$

Begin
/* The search is done in a breadth-first way (i.e. level by level). The nodes at the level $i$ are \{(X, X') | II X' II = i\} */
$\Sigma := \emptyset$;
For each node $N=(X, X') \in G$ do
Begin
$A := \{\text{IRDs generated from the current node}\}$
If $X \neq \emptyset$ and $II X' II > 1$ then
For each $P \in \{P'(X') \cdot \emptyset \cdot X\}$ do
If $\neg \exists a$ parent $M = (Y, Y')$ of $N$ such that $P \subset Y'$ then
If $\neg \exists$ $P' \rightarrow Q$ in $\Delta$ such that $P' \subset P$ then
$\Delta := \Delta \cup \{P \rightarrow X' \cdot P\}$
EndIf
EndIf
EndFor
Z := Z U A
End
EndFor
End

Algorithm 5 systematically generates rules by exploring the whole lattice. Even though the If-tests within the second For loop help eliminate redundant rules, some ones may still exist since they are produced by two different iterations of the first For. They can be removed by using the minimal cover algorithm borrowed from the work done on functional dependencies as suggested earlier. A possible minimal cover for our example (Figure 3) is:

$f \rightarrow c$ (node #17)
If a student is excellent, then he/she has at least an average score $= B$

ag $\rightarrow i$ (node #7)
If a person is a management graduate student, then he/she has access to the laboratory $X$

ai $\rightarrow g$, gi $\rightarrow a$ (node #7)
h $\rightarrow f$ (node #6)
If a student has a scholarship, then he/she is excellent
bc $\rightarrow f$ (node #18)
e $\rightarrow bg$, bg $\rightarrow e$ (node #9)
af $\rightarrow h$ (node #10)
cg $\rightarrow a$, cg $\rightarrow i$ (node #11)
d $\rightarrow ai$ (node #12)
bi $\rightarrow f$, fi $\rightarrow b$ (node #19)

Algorithm 6: Deriving the Determinant Set for Q

Determinant(Q, $\Sigma_m$)
Input: Q, $\Sigma_m$
Output: the determinant P

Begin
$P' := \emptyset$
For each $P \rightarrow P_j$ in $\Sigma_m$ do
If $P_j = Q$ then
$P' := P' \cup \{P\}$
For each $R \rightarrow S$ in $\Sigma_m$ do
If $S \subseteq P$ and $Q \subset R$ then
$P' := P' \cup \{R \cup (P - S)\}$
EndIf
EndIf
EndFor
End

End

Example:
Determinant (({b}), $\Sigma_m$)= (fi, agf, hi, df, cgf, e). It represents the alternative properties that subsume b (i.e. eligible for a position at the university).

Algorithm 7: Minimize P under $\Sigma_m$

Input: P, $\Sigma_m$
Output: P

Begin
RES := P
For each property $p$ in P do
If RES $(\neg p) \rightarrow (p)$ holds then
RES := RES - (p)
EndIf
EndFor
End

Example:
$\Sigma_m$= $a \rightarrow b$, $bd \rightarrow e$
Minimize (P, $\Sigma_m$)= \{ad\}.

5 Conclusion

An incremental algorithm for updating the Galois lattice of a binary relation has been proposed and analyzed. A large experimental application was performed, showing that adding a new object to the lattice may be done in time proportional to the total number of objects. Theoretical analyses of the algorithm confirm the experimental data when a fixed upper bound on the number of properties related to an object is supposed, which is the case in practical applications. These results are very encouraging for practical use. Algorithms for generating and dealing with rules from the lattice were proposed.

Much work remains to be done, and we mention here some directions that we are currently exploring:

- Refining the algorithms by using some heuristics to cope with a potentially explosive set of rules. One such
heuristic is limiting the search to a subset of the lattice nodes.

- Generalizing the Galois lattice nodes structure to allow richer knowledge representation schemes such as conceptual graphs [17, 18].
- Testing the potential of these ideas in different application domains such as information retrieval [9], database design, and intensional database querying.

Acknowledgments

We would like to thank Eugene Saunders for his contribution to the implementation of the algorithms. This research has been supported by NSERC (the Natural Sciences and Engineering Research Council of Canada) under grants No OGP0041989, OGP0009184, EQP0092688 and by FCAR Funds (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) under grant No 91-NC-0446.

References