A Parameterized Algorithm for Exploring Concept Lattices

Peggy Cellier[1], Sébastien Ferré[1], Olivier Ridoux[1] and Mireille Ducassé[2]

[1]IRISA/University of Rennes 1 and [2]IRISA/INSA,
Campus universitaire de Beaulieu, 35042 Rennes, France
firstname.lastname@irisa.fr
http://www.irisa.fr/LIS/

Abstract. Kuznetsov shows that Formal Concept Analysis (FCA) is a natural framework for learning from positive and negative examples. Indeed, the results of learning from positive examples (respectively negative examples) are sets of frequent concepts with respect to a minimal support, whose extent contains only positive examples (respectively negative examples). In terms of association rules, the above learning can be seen as searching the premises of exact rules where the consequence is fixed. When augmented with statistical indicators like *confidence* and *support* it is possible to extract various kinds of concept-based rules taking into account exceptions. FCA considers attributes as a non-ordered set. When attributes of the context are ordered, Conceptual Scaling allows the related taxonomy to be taken into account by producing a context completed with all attributes deduced from the taxonomy. The drawback of that method is concept intents contain redundant information. In a previous work, we proposed an algorithm based on Bordat's algorithm to find frequent concepts in a context with taxonomy. In that algorithm, the taxonomy is taken into account during the computation so as to remove all redundancy from intents. In this article, we propose a parameterized generalization of that algorithm for learning rules in the presence of a taxonomy. Simply changing one component, that parameterized algorithm can compute various kinds of concept-based rules. We present applications of the parameterized algorithm to find positive and negative rules.

1 Introduction

Learning from example is often the best approach when it is not possible to design a model a priori. This has been mainly tried for classification purpose. In that case, one proposes to represent classes by examples, or counter-examples, and a formal model of the classes is learned by a machine and applied to further inputs.

Kuznetsov shows that Formal Concept Analysis (FCA) [GW99] is a natural framework for learning from positive and negative examples [Kuz04,Mit97]. Indeed, the results of learning from positive examples (respectively negative examples) are sets of frequent concepts with respect to a minimal support, whose

extent contains only positive examples (respectively negative examples). In terms of *association rules* of Agrawal et al. [AIS93,AS94], the above learning can be seen as searching the premises of exact rules where the consequence is fixed. When augmented with statistical indicators like *confidence* and *support* it is possible to extract various kinds of concept-based rules taking into account exceptions [PBTL99,Zak04].

The input of FCA is a formal context that relates objects and attributes. FCA considers attributes as a non-ordered set. When attributes of the context are ordered, Conceptual Scaling [GW99] allows the related taxonomy to be taken into account by producing a context completed with all attributes deduced from the taxonomy. The drawback of that method is that concept intents contain redundant information. In a previous work [CFRD06], we proposed an algorithm based on Bordat's algorithm [Bor86] to find frequent concepts in a context with taxonomy. In that algorithm, the taxonomy is taken into account during the computation so as to remove all redundancies from intents.

There are several kinds of association rules, and several problems on this domain: e.g. finding all association rules with respect to some criteria, computing all association rules with a given conclusion or premises. It can be easily foreseen that these different problems will be solved by variants of a generic algorithm. Thus, it is important that that generic algorithm be designed and implemented generically. We propose a parameterized generalization of Bordat's algorithm to learn rules in the presence of taxonomy. Simply changing one component, that parameterized algorithm can compute various kinds of concept-based rules. We present applications of the parameterized algorithm to find positive and negative rules. Positive rules predict some given target (e.g. poisonous), while negative rules predict its opposite (e.g. edible). The advantage of taking into account the taxonomy is to reduce the size of the results. For example, in a context about Living Things, for a the target "suckling" the rule "Living Things"$\wedge$ "Animalia" $\wedge$ "Chordata" $\wedge$ "Vertebrata" $\wedge$ "Mammalia" $\rightarrow$ "suckling" is less relevant than the equivalent rule "Mammalia" $\rightarrow$ "suckling" where all redundant elements have been eliminated.

Our approach is formalized using the Logical Concept Analysis (LCA) framework [FR04], the taxonomy is taken into account as a specialized logic.

The contribution of this article is a generalization of the exploration of frequent concepts in a context with taxonomy in order to compute concept-based rules where statistical indicators are taken into account. Two applications of that generalization are presented.

In the following, Section 2 briefly recalls the framework, Logical Concept Analysis (LCA), used to define FCA with taxonomy (FCA-Tax). Section 3 presents the generalization of the algorithm described in [CFRD06] to filter frequent concepts in a formal context with taxonomy. Section 4 shows how to learn rules. Section 5 illustrates that the algorithm produces, as efficiently as with the completed context, sets of rules where rules are more compact. Section 6 concludes this paper.

2 A Logical Framework for FCA with Taxonomy

In this section, we formally describe Formal Concept Analysis with Taxonomy (FCA-Tax) using the Logical Concept Analysis (LCA) framework. We first present an example of context with taxonomy. Then we briefly introduce LCA, concept-based rules, and we instanciate LCA to FCA-Tax. The taxonomy describes that the attributes of the context are ordered and thus a taxonomy is a kind of logic where the subsumption relation represents this order relation. A detailed description of LCA, as well as a precise discussion on how it generalizes FCA can be found in [FR04].

2.1 Example of Context with a Taxonomy

	Pacific	*Southwest*	*Southeast*	*Region8*	*Hawai*	*Washington*	*Arizona*	*Michigan*	*Mississipi*	*Florida*	*Utah*	*Alaska*	*Oregon*	*California*	*Nevada*	*Endangered*	*Threatened*
Accipitridae										•						•	
Alcedinidae	•															•	
Alcidae						•							•	•			•
Anatidae												•					•
Cathartide														•		•	
Charadriidae								•		•				•		•	•
Corvidae					•											•	
Drepanidinae	•															•	
Emberizidae				•													•
Gruidae		•							•							•	
Icteridae			•													•	
Muscicapidae				•													•
Strigidae						•	•				•		•	•			•
Tyrannidae							•				•			•	•	•	
Vireonidae														•		•	

Table 1. *Context of threatened or endangered bird families*

The context given Table 1 represents the observations of threatened and endangered bird families. Data come from the web site of USFWS[1] (U.S Fish and Wildlife Service). The objects are bird families and each family is described by a set of states and regions where specimens have been observed and by a status: *threatened* or *endangered*. Note that in this context objects are elements of what could be a taxonomy in another context. We have chosen this context because it refers to a knowledge, (e.g. *Hawai* $\rightarrow$ *Pacific*) we believe is which

[1] http://ecos.fws.gov/tess_public/CriticalHabitat.do?listings=0&nmfs=1

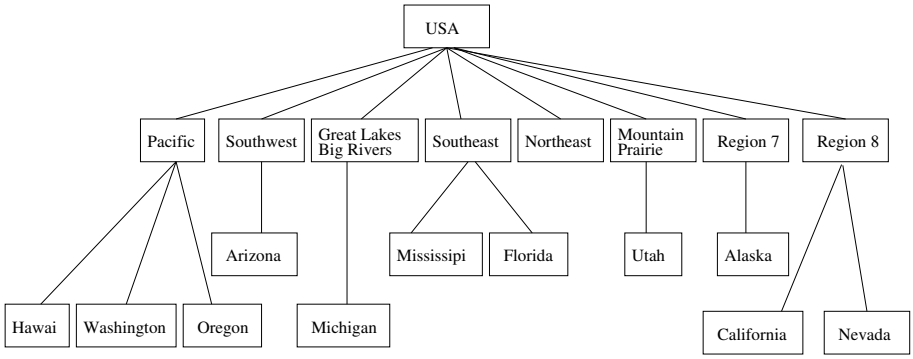


Fig. 1. Example: taxonomy of USA states and regions.

better known than the taxonomy of Living Things. Figure 1 shows the taxonomy of states and regions of USA. USA states are gathered in regions and this forms a taxonomy.

The context given in Table 1 is the straightforward transcription of the information found on the USFWS site. Note that it is not completed with respect to FCA. For example object *Accipitridae* which has in its description the attribute *Florida* has not the attribute *Southeast*. The information that an object has the attribute *Florida* implies that it has the attribute *Southeast*, is given implicitely by the taxonomy. We have observed that this situation is quite common.

2.2 Logical Concept Analysis (LCA)

In LCA the description of an object is a logical formula instead of a set of attributes as in FCA.

Definition 1 (logical context). *A* logical context *is a triple* $(\mathcal{O}, \mathcal{L}, d)$ *where* $\mathcal{O}$ *is a set of objects,* $\mathcal{L}$ *is a logic (e.g. proposition calculus) and* d *is a mapping from* $\mathcal{O}$ *to* $\mathcal{L}$ *that describes each object by a formula.*

The definition of a logic is given in Definition 2.

Definition 2 (logic). *A* logic *is a 6-tuple* $\mathcal{L} = (L, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$ *where*

- L *is the language of formulas,*
- $\sqsubseteq$ *is the subsumption relation,*
- $\sqcap$ *and* $\sqcup$ *are respectively conjunction and disjunction,*
- $\top$ *and* $\bot$ *are respectively tautology and contradiction.*

Definition 3 defines the logical versions of *extent* and *intent*. The extent of a logical formula f is the set of objects in $\mathcal{O}$ whose description is subsumed by f. The intent of a set of objects O is the most precise formula that subsums all descriptions of objects in O. Definition 4 gives the definition of a *logical concept*.

Definition 3 (extent, intent). *Let $\mathcal{K} = (\mathcal{O}, \mathcal{L}, d)$ be a logical context. The definition of extent and intent are:*

- $\forall f \in \mathcal{L} \quad ext(f) = \{o \in \mathcal{O} \mid d(o) \sqsubseteq f\}$
- $\forall O \subseteq \mathcal{O}\ int(O) = \bigsqcup_{o \in O} d(o)$

Definition 4 (logical concept). *Let $\mathcal{K} = (\mathcal{O}, \mathcal{L}, d)$ be a logical context. A* logical concept *is a pair $c = (O, f)$ where $O \subseteq \mathcal{O}$, and $f \in \mathcal{L}$, such that $int(O) \equiv f$ and $ext(f) = O$. O is called the extent of the concept c, i.e. ext_c, and f is called its intent, i.e. int_c.*

The set of all logical concepts is ordered and forms a *lattice*: let c and c' be two concepts $c < c'$ iff $ext_c \subset ext_{c'}$. Note also that $c < c'$ if $int_c \sqsubseteq int_{c'}$ but the converse is false.

The fact that the definition of a logic is left so abstract makes it possible to accommodate non-standard types of logics. For example, attributes can be valued (e.g. integer intervals, string patterns), and each domain of value can be defined as a logic. The subsumption relation allows to order integers and intervals, strings and substrings, and the terms of a taxonomy.

2.3 Concept-Based Rules

Definition 5 (concept-based rules). *The concept-based rules consider a target $T \subset \mathcal{L}$ such that T represents a set of objects that are positive (respectively negative) examples. Concept-based rules have the form: $X \rightarrow T$ where X is the intent of a frequent concept. A rule can have exceptions. These exceptions are measured with statistic indicators like support and confidence, defined below.*

There exists several statistical indicators like *support*, *confidence*, *lift* and *conviction* [BMUT97]. We have chosen support and confidence to measure the relevance of the rules, because they are the more widespread. However, the algorithm presented on this article does not depend on this choice.

Definition 6 (support). *The support of a formula f is the number of objects described by that formula. It is defined by:*

$$sup(f) = \|ext(f)\|$$

where $\|X\|$ denotes the cardinal of a set X.

The support of a rule is the number of objects described by both f_1 and f_2. It is given by:

$$sup(f_1 \rightarrow f_2) = \|ext(f_1) \cap ext(f_2)\|$$

Definition 7 (confidence). *The confidence of a rule $f_1 \rightarrow f_2$ describes the probability to have f_2 in a description which contains f_1. It is defined by:*

$$conf(f_1 \rightarrow f_2) = \frac{\|ext(f_1) \cap ext(f_2)\|}{\|ext(f_1)\|}$$

The lift is a variant of confidence which measure to what extent the probability to have f_2 is augmented by f_1.

The support indicator applies as well to concepts as to rules. In the case of concept it introduces the notion of *frequent concept*.

Definition 8 (frequent concept). *A concept is called frequent with respect to a min_sup threshold if $sup(int_c)$ is greater than min_sup.*

2.4 Formal Concept Analysis with Taxonomy

FCA-Tax is more general than FCA and can be formalized in LCA. Indeed, the attributes of the context are ordered.

Definition 9 (taxonomy). *A taxonomy is a partially ordered set of terms:*

$$TAX =< T, \leq >$$

where T is the set of terms and leq is the partial ordering. Let x and y be attributes in T, $x \leq y$ means that y is more general than x.

Example 1. In Figure 1: $Hawai \leq Pacific \leq USA$.

Definition 10 (predecessors). *Let $TAX =< T, \leq >$ be a taxonomy. For a set of attributes X in T, its predecessors in the taxonomy TAX are denoted by*

$$\uparrow_{tax} (X) = \{\, t \in T \mid \exists x \in X : x \leq t \,\}$$

Example 2.

$$\uparrow_{tax} (\{Mississipi, Arizona\}) = \{Mississipi, Southeast, Arizona, Southwest, USA\}$$

Definition 11 (successors). *Let $TAX =< T, \leq >$ be a taxonomy. For a set of attributes X in T, its succesors in the taxonomy TAX are denoted by*

$$succ_{tax}(X) = \{\, t \in T \mid \exists x \in X : x > t \wedge (\nexists t' \in T x > t' > t) \,\}$$

Example 3.

$$succ_{tax}(\{USA\}) = \{Pacific, Southeast, Southwest, GreatLakes/BigRivers, Northeast, Mountains/Prairie, Region7, Region8, Southwest, USA\}$$

Definition 12 ($\mathbf{Min_{tax}}$). *Let $TAX =< T, \leq >$ be a taxonomy. $Min_{tax}(X)$ is the set of minimal elements with respect to TAX.*

$$Min_{tax}(X) = \{\, t \in X \mid \nexists x \in X : x < t \,\}$$

Example 4. In fact, $Min_{tax}(X)$ is X minus its elements that are redundant with respect to TAX.

$$Min_{tax}(\{Florida, Southeast, USA\}) = \{Florida\}$$

Southeast and *USA* are not needed because they are implicitly present with *Florida*, thanks to the taxonomy.

Definition 13 ($\mathcal{L}_{\mathbf{tax}}$). *The specialization of the logic of LCA for FCA-Tax* $TAX =< T, \leq >$ *is* $\mathcal{L}_{tax}$ *defined by:*

- $L = 2^T$
- $\sqsubseteq$ *such that* $X \sqsubseteq Y$ *iff* $\uparrow_{tax}(X) \supseteq \uparrow_{tax}(Y)$
- $\sqcap$ *is* $\cup_{tax}$ *such that* $X \cup_{tax} Y = Min_{tax}(X \cup Y)$,
- $\sqcup$ *is* $\cap_{tax}$ *such that* $X \cap_{tax} Y = Min_{tax}(\uparrow_{tax}(X) \cap \uparrow_{tax}(Y))$,
- $\top = \{x \in T | ext(\{x\}) = \mathcal{O}\}$
- $\bot = Min_{tax}(T)$.

$$L = \{\{Hawai\}, \{Oregon\}, \{Pacific\}, \{Southwest\}, ..., \{USA\}, \{Hawai, Southwest\}, ...\} \quad (1)$$

$$\{Michigan, Florida\} \sqsubseteq \{Michigan, Southeast\} \quad (2)$$

$$\{Michigan, Southeast\} \sqcap \{Florida\} = \{Michigan, Florida\} \quad (3)$$

$$\{Mississipi\} \sqcup \{Florida\} = \{Southeast\} \quad (4)$$

$$\top = \{USA\} \quad (5)$$

$$\bot = \{Hawai, Washington, Oregon, Arizona, Michigan, Mississipi, Florida, Utah, Alaska, California, Nevada\} \quad (6)$$

Fig. 2. Examples of operations on bird families of Table 1.

Example 5. Figure 2 shows examples of operations on the context of bird families. The language, L, allows redundancy in descriptions. However, rules that our algorithm will compute will contain no redundancy (Equation (1)). Equation (2) illustrates the subsumption relation. *Florida* implies *Southeast* in the taxonomy, thus all bird families observed in Michigan and Florida can also be said to have been observed

in Michigan and Southeast. Equation (3) represents the conjunction operation. To be observed in Florida is an information more precise than to be observed in Southeast, thus only the attribute *Florida* is kept. Equation (4) represents the disjunction operation. The fact that birds are observed in Michigan or Florida can be generalized by birds which are observed in Southeast. The top concept contains only one attribute *USA* (equation (5)), and the bottom concept all minimal attributes (equation (6)).

All notions defined in LCA are kept in FCA-Tax, in particular extent, intent, and concept lattice. FCA-Tax differs from FCA in that the intents of concepts in FCA-Tax are without redundancy.

3 A Parameterized Algorithm to Find Concept-Based Rules

In a previous article [CFRD06] we described an algorithm for finding frequent concepts in a context with taxonomy. The algorithm is a variant of Bordat's algorithm which takes care of the taxonomy for avoiding redundant intents. In this section, a generalization of this algorithm is described in order to search concept-based rules. In a first step, the relation between frequent concepts and rules is presented. The configurable filter function, called **FILTER**, is introduced. In a second step, the parameterized algorithm with this filter function is described. Finally, we discuss the differences with Bordat's algorithm.

3.1 From Frequent Concepts to Rules

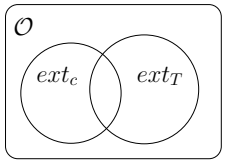
Relevant rules are rules that are frequently observed. In general (e.g. in [PBTL99]), all frequent rules are searched, without any constraints on the premises and conclusions. However, the learning case makes one search for frequent rules where either the premise or the conclusion is fixed by the learning target. So, one searches for conclusions or premises that match the target. For instance, learning sufficient conditions for a target T is to search frequent implications like $X \rightarrow T$. As we have seen in Section 2, the frequency of a rule is evaluated by a statistical indicator: the support. Thus, a rule cannot be more frequent than its premise or its conclusion, because let c be a frequent concept then $sup(int_c) \geq sup(int_c \cup T) = sup(int_c \rightarrow T)$. This implies that only the intents of frequent concepts are good candidates to be the premises of the searched rules.

Therefore, in order to learn rules, the frequent concepts that are computed by the algorithm presented in [CFRD06] be filtered. Only must frequent concepts that form relevant rules with respect to statistical indicators are kept.

In addition, some statistical indicators like the support, are monotonous. For instance, given two concepts c and c', $c < c' \Rightarrow sup(c) < sup(c')$ holds. It means that if the support of a concept c is lower than min_sup, all subconcepts of c have

a support lower than min_sup. Thus, it is not relevant to explore subconcepts of c.

A filter function, called **FILTER**, is defined in order to take into account this property. **FILTER** takes two parameters: ext_c and ext_T. These two parameters are sufficient to compute all statistical indicators like support, confidence, lift and conviction as it is illustrated in Figure 3. Using ext_c and ext_T, we can compute $ext_c \cup ext_T$, $ext_c \cap ext_T$, and all complements like $\mathcal{O} \setminus ext_c$, $\mathcal{O} \setminus (ext_c \cup ext_T)$. **FILTER**($ext_c$, ext_T) returns two booleans: **KEEP** and **CONTINUE**. **KEEP** tells whether $int_c \rightarrow T$ is a relevant rule with respect to statistical indicators. **CONTINUE** allows monotonous properties to be considered. **CONTINUE** tells whether there are may be some subconcept c' of c such that $int'_c \rightarrow T$ is a relevant rule. Thus **FILTER** gives four possibilities: 1) keep the current concept and explore subconcepts, 2) keep the current concept and do not explore subconcepts, 3) do not keep the current concept and explore subconcepts, 4) do not keep the current concept and do not explore subconcepts. In Section 4, we see on examples how these four possibilities are used.



$$sup(int_c \rightarrow T) = \|ext_c \cap ext(T)\|$$

$$conf(int_c \rightarrow T) = \frac{\|ext_c \cap ext(T)\|}{\|ext_c\|}$$

$$lift(int_c \rightarrow T) = \frac{\|ext_c \cap ext(T)\|}{\|ext_c\| * \|ext(T)\|}$$

$$conv(int_c \rightarrow T) = \frac{\|ext_c\| * \|\mathcal{O} \setminus ext(T)\|}{\|ext_c \cap (\mathcal{O} \setminus ext(T))\|}$$

Fig. 3. Computing indicators using the extents of a frequent concept c and a target T.

3.2 Algorithm Parameterized with Function FILTER

The algorithm uses two data structures: $incr_c$ of a concept c and **Exploration**. Appart from the top concept, each concept s is computed from a concept $p(s)$, which we call the predecessor of s. This predecessor is such that there exists a set of attributes, X, such that $ext_s = ext_{p(s)} \cap ext(X)$. We call X an increment of $p(s)$, and we say that X *leads* from $p(s)$ to s. All known increments are kept in a data structure $incr_c$ which is a mapping from subconcepts to increments: s maps to X iff X leads to s. Notation $incr_c[s \rightarrow X]$ means that the mapping is modified so that s maps to X. **Exploration** contains a set of frequent subconcepts that are to be explored. In **Exploration**, each concept s to explore is represented by a triple: $((ext_s \rightarrow X), int_{p(s)}, incr_{p(s)})$ where $ext_s = ext(int_{p(s)} \cup X)$ and $ext_s \geq min\_sup$, which means that X is an increment from $p(s)$ to s. $incr_{p(s)}$ are increments of the predecessor concept of s.

An invariant for the correction of the algorithm is that

$$incr_c \subseteq \{(s \rightarrow X) \mid ext_s = ext_c \cap ext(X) \wedge \|ext_s\| \geq min\_sup\} \ .$$

Algorithm 1 Explore_concepts

Require: $\mathcal{K}$, a context with a taxonomy TAX; and *min_sup*, a minimal support
Ensure: **Solution**, a set of all frequent concepts of $\mathcal{K}$ with respect to *min_sup*, and check **FILTER**

1: **Solution** := $\emptyset$
2: **Exploration**.add($(\mathcal{O} \rightarrow \top, \emptyset, \emptyset)$
3: **while Exploration** $\neq \emptyset$ **do**
4: **let** $((ext_s \rightarrow X), int_{p(s)}, incr_{p(s)}) = max_{ext}$(**Exploration**) **in**
5: (**KEEP**, **CONTINUE**) := **FILTER**(ext_s, ext_T)
6: **if KEEP** or **CONTINUE then**
7: $\text{int}_s := (\text{int}_{p(s)} \cup_{tax} X) \cup_{tax} \{y \in succ^+_{tax}(X) \mid ext_s \subseteq ext(\{y\})\}$
8: **if CONTINUE then**
9: $\text{incr}_s := \{(c \rightarrow X) \mid \exists c' : (c' \rightarrow X) \in incr_{p(s)} \wedge c = ext_s \cap c' \wedge \|c\| \geq min\_sup\}$
10: **for all** $y \in succ_{tax}(X)$ **do**
11: **let** $c = ext_s \cap ext(\{y\})$ **in**
12: **if** $\|c\| \geq min\_sup$ **then**
13: $incr_s := incr_s[c \rightarrow (incr_s(c) \cup \{y\})]$
14: **end if**
15: **end for**
16: **for all** $(ext \rightarrow Y)$ in $incr_s$ **do**
17: **Exploration**.add($(ext \rightarrow Y)$, int_s, incr_s)
18: **end for**
19: **end if**
20: **if KEEP then**
21: **Solution**.add(ext_s, int_s)
22: **end if**
23: **end if**
24: **end while**

Thus, all elements of $incr_c$ are frequent subconcepts of c.

An invariant for completeness is that

$$\begin{aligned} & ext_c \supset ext_s \wedge \|ext_s\| \geq min\_sup \wedge \neg\exists X : (s \rightarrow X) \in incr_c \\ \Longrightarrow\ & \exists s' : ext_c \supset ext_{s'} \supset ext_s \wedge \exists X : (s' \rightarrow X) \in incr_c \ . \end{aligned}$$

Thus, all frequent subconcepts of c that are not in $incr_c$ are subconcepts of a subconcept of c which is in $incr_c$. See [CFRD06] for more details.

Algorithm Explore_concepts allows frequent concepts of a context with taxonomy to be filtered with the configurable function **FILTER** previously introduced. The exploration of the concept lattice is top-down and starts with the top of the lattice, i.e. the concept labelled by $root_{tax}$ (step 2).

At each iteration of the while loop, an element of **Exploration** with the largest extent is selected (step 4): $((ext_s \rightarrow X), int_{p(s)}, incr_{p(s)})$. This element represents a concept s which is tested with the function **FILTER** (step 5). Then at step 7, the intent of s is completed with respect to the taxonomy and thanks to $\cup_{tax}$ without redundancy, defined at Section 2. $incr_s$ is computed with elements of $incr_{p(s)}$ (step 9) and successors of X in the taxonomy (steps 10-15). Note

that increments of $p(s)$ can still be increments of s. For instance, if $p(s) > s$, $p(s) > s'$, and $s > s'$, the increment of $p(s)$ which leads to s' is also an increment of s that leads to s'. If these increments are relevant with respect to **FILTER**, they are added to **Exploration** (step 16-18). Finally if s is relevant with respect to **FILTER**, it is added to **Solution** (step 21).

The difference between the algorithm in [CFRD06] and the algorithm previously presented is the addition of the function **FILTER** that allows frequent concepts to be filtered and exploration to be stopped. If function **FILTER** always returns (**KEEP** = true, **CONTINUE**=true), then all frequent concepts are eventually computed.

3.3 Comparison with Bordat's algorithm

The algorithm presented in the previous section is based on Bordat's algorithm. Like in Bordat's version, our algorithm starts by exploring the top concept. Then for each concept s explored, the subconcepts of s are computed; this corresponds to the computation of $incr_s$. Our algorithm uses the same data structure to represent **Solution**, i.e. a trie. The differences between our algorithm and Bordat's are: 1) the strategy of exploration, 2) the data structure of **Exploration** and 3) taking into account the taxonomy.

In our algorithm,

1. we first explore concepts in **Exploration** with the largest extent. Thus, it is a top-down exploration of the concept lattice. Contrary to Bordat, it avoids to test if a concept has already been found when it is added to **Solution**.
2. Whereas in Bordat's algorithm **Exploration** is represented by a queue that contains subconcepts to explore, in our version the data structure of **Exploration** is a triple: $((ext_s \rightarrow X), int_{p(s)}, incr_{p(s)})$ where $incr_{p(s)}$ avoids to test all attributes of the context to compute increments of s. Indeed, if an increment X is not relevant for c it cannot be relevant for s, because $ext_s \subset ext_{p(s)}$ and thus $(ext_s \cap ext(X)) \subset (ext_{p(s)} \cap ext(X))$. Thus, the relevant increments of the predecessor of s are stored in $incr_{p(s)}$ and are potential increments of s. In **Exploration**, a concept cannot be represented several times. When a triple representing a concept c is added to **Exploration**, if c is yet represented in **Exploration**, the new triple erases the previous.
3. The attributes can be structured in a taxonomy. This taxonomy is taken into account during the computation of the increments and the intent. Using $\cup_{tax}$ instead of the plain set-theoretic union operation allows computed intents to be without redundancy. Another benefit is that it makes computation more efficient. For instance, when the taxonomy is deep, a lot of attributes can be pruned without testing.

Example 6. If we consider the taxonomy of Living Things[2] with attributes like the kingdom (e.g. Animalia, Plantae, ...), the phylum (e.g. chordata), subphylum

[2] http://anthro.palomar.edu/animal/table_humans.htm

(e.g. Vertebrata), ..., family (e.g. hominidae), genus (e.g. homo), and species (e.g. sapiens). If a concept c with "Animalia" in its intent is unfrequent, it is useless to explore subconcepts of c. Using the algorithm, all subconcepts of c due to the taxonomy are pruned.

4 Specializations of the Parameterized Algorithm

In the previous section, the algorithm presented is parameterized with a filter function in order to permit the computation of concept-based rules in FCA-Tax. The advantage of computing the premises of rules in FCA-Tax is to reduce the size of the rules.

In this section, we show two instanciations of this filter function to learning. The first one allows *sufficient conditions* to be computed. In this case, a learning objective is expressed as positive examples, which are themselves expressed by their intent. Thus, sufficient conditions are premises of rules where the conclusion (the target), T, represents the positive examples. In other words, we search rules like: $X \rightarrow T$. The second one allows *incompatible conditions* to be computed. In this case, a learning objective is expressed as negative examples, which are themselves expressed by their intent. Thus, incompatible conditions are premises of rules where the conclusion (the target), T, is the opposite of the intent of the negative examples. In other words, we search rules like: $X \rightarrow \neg T$.

4.1 Computing sufficient conditions

In the generation of sufficient conditions, the learning objective, T, is the target of the rules to be learned. For computing sufficient conditions, one must determine what conjunctions of attributes, P, implies T, i.e. the intents P of concepts such that the rule $P \rightarrow T$ has a support and a confidence greater than the thresholds min_sup and min_conf. Sufficient conditions can be computed with the algorithm described in Section 3 that filters all frequent concepts of a context, by instanciating the filter function with:

$$\begin{aligned}\textbf{FILTER}_{sc}(ext_s, ext_T) = (\textbf{KEEP} = sup(int_s \rightarrow T) \geq min\_sup \wedge \\ conf(int_s \rightarrow T) \geq min\_conf, \\ \textbf{CONTINUE} = sup(int_s \rightarrow T) \geq min\_sup)\end{aligned}$$

It implies three possible behaviours of the algorithm.

1. The concept s is relevant, i.e. the rule $int_s \rightarrow T$ has a support and a confidence greater than the thresholds. s is a solution and has to be kept, and its subconcepts are potential solutions and have to be explored. Thus $\textbf{FILTER}_{sc}(ext_s)$ returns $(true, true)$.
2. The concept s has a support greater than min_sup but a confidence lower than min_conf. s is not a solution but the confidence is not monotonous and thus subconcepts of s can be solutions. $\textbf{FILTER}_{sc}(ext_s)$ returns $(false, true)$.

3. The concept s has a support lower than min_sup. s is not a solution. As the support is monotonous, i.e. the support of a subconcept of s is lower than the support of s, subconcepts of s cannot be solutions. $\mathbf{FILTER}_{sc}(ext_s)$ returns $(false, false)$.

The definition of $\mathbf{FILTER}_{sc}(ext_s, ext_T)$ given previously can be expressed in terms of extent by simply applying the definitions of support and confidence seen in Section 2. The filter function can be defined by :

$$\begin{aligned}\mathbf{FILTER}_{sc}(ext_s) = (\mathbf{KEEP} = \|ext_s \cap ext_T\| \geq min\_sup \wedge \\ \frac{\|ext_s \cap ext_T\|}{\|ext_s\|} \geq min\_conf, \\ \mathbf{CONTINUE} = \|ext_s \cap ext_T\| \geq min\_sup)\end{aligned}$$

Note that it is easy to use others statistical indicators like lift or conviction, by adding conditions in the evaluation of **KEEP**.

4.2 Computing incompatible conditions

In the case of incompatible conditions, the learning objective, T, is the negation of the target of the rules to be learned. For computing incompatible conditions, one must determine what conjunctions of attributes, P, implies $\neg T$, i.e. the intents P of concepts such that the rule $P \rightarrow \neg T$ has a support and a confidence greater than the thresholds min_sup and min_conf. Incompatible conditions can be computed with the algorithm described in Section 3 that filters all frequent concepts of a context, by instanciating the filter function with:

$$\begin{aligned}\mathbf{FILTER}_{ic}(ext_s, ext_T) = (\mathbf{KEEP} = sup(int_s \rightarrow \neg T) \geq min\_sup \wedge \\ conf(int_s \rightarrow \neg T) \geq min\_conf, \\ \mathbf{CONTINUE} = sup(int_s \rightarrow \neg T) \geq min\_sup)\end{aligned}$$

This can be expressed in terms of extents, using the definitions of support and confidence. In terms of extent, the filter function to compute incompatible conditions can be defined by :

$$\begin{aligned}\mathbf{FILTER}_{ic}(ext_s, ext_T) = (\mathbf{KEEP} = \|ext_s \cap (\mathcal{O} \setminus ext_T)\| \geq min\_sup \wedge \\ \frac{\|ext_s \cap (\mathcal{O} \setminus ext_T)\|}{\|ext_s\|} \geq min\_conf, \\ \mathbf{CONTINUE} = \|ext_s \cap (\mathcal{O} \setminus ext_T)\| \geq min\_sup)\end{aligned}$$

The behaviour of this algorithm is the same as for sufficient conditions.

5 Experiments

The algorithm is implemented in CAML (a functional programming language of the ML family) inside the Logic File System LISFS [PR03]. LISFS implements

the notion of Logical Information Systems (LIS) as a native Linux file system. Logical Information Systems (LIS) are based on LCA. In LISFS, attributes can be ordered to create a taxonomy (logical ordering). The data structures which are used allow taxonomies to be easily manipulated. For more details see [PR03]. We ran experiments on an Intel(R) Pentium(R) M processor 2.00GHz with Fedora Core release 4, 1GB of main memory.

The experimental objectives are 1) to measure the gain in space due to the elimination of redundancy, 2) to measure the overhead in time of taking care of the taxonomy, 3) to assess the relevance of the computed rules, and 4) to compare the overall algorithm with respect to other algorithms that do not take care of taxonomies.

- The first and second objectives have been first examined in a previous article on computing frequent concepts. Indeed, redundancy in rules is the same as redundancy in frequent concepts. To this aim we have programmed a variant of our algorithm which does not take care of the taxonomy. We have measured the gain in space using a context of Java methods in which the type hierarchy is the main source of the taxonomy. In the following we recall figures when necessary.
- The third objective needs a qualitative evaluation of the computed rules. This is a difficult question, for which we have preleminary results. We have replayed this experiments with the parameterized algorithm for computing rules and evaluating them.
- The last objective needs to compare our parameterized algorithm with other, efficient, algorithms. We have chosen to compare with Pasquier *et al.*'s algorithm using the contexts they are using in their article. In particular, we present data computed for the context MUSHROOMS.

Java context

We test the parameterized algorithm with the Java context[3] taken from [SR06]. This context contains 5 526 objects which are the methods of java.awt. Each method is described by its input and output types, visibility modifiers, exceptions, keywords extracted from its identifiers, and keywords from its comments. The context has 1 624 properties, and yields about 135 000 concepts. The taxonomy is derived for the largest part from the class inheritance graph, and for a smaller part from a taxonomy of visibility modifiers that is predefined in Java.

For the first experiment on the Java context, we choose to measure the computation time of the frequent concepts, in order to show the impact of the taxonomy on the computation. For instance, when $min\_sup = 5\%$, 189 frequent concepts are computed in 8s taking care of the taxonomy, whereas they are computed in 16s without taking it into account. When $min\_sup = 2.5\%$, 2299 frequent concepts are computed in 55s taking care of the taxonomy, whereas they are computed in 54s without taking it into account. It is explained by the fact that the Java context has few very frequent concepts. Thus pruning using

[3] Available on the web at http://lfs.irisa.fr/demo-area/awt-source/

the taxonomy when $min\_sup = 5\%$ occurs very soon. Indeed, it occurs before exploring the details of the Java class hierarchy. With this context taking into account the taxonomy allows the number of irrelevant attributes in the intents to be reduced of 39%

In a second experiment, we compute the premises of two rules about the set and get methods (e.g. *setValue*, *getValue*, *setName*, *getName*). The target is the set of methods containing the keyword "set" in their name. For $min\_sup = 5\%$ and $min\_conf = 80\%$ we find 6 rules that underline that the methods with the keyword "set" in their name, generally have the properties: to be in public access, to return void and to contain the keyword "sets" in the comments. The second target is methods containing the keyword "get" in their name. For $min\_sup = 5\%$ and $min\_conf = 80\%$ we find 5 rules that underline that the methods with the keyword "get" in their name, generally have the properties: to be in public access, to return integer and to contain the keyword "returns" in the comments. Note that it is not possible to choose freely the support of a rule. This is because the support of a rule cannot be greater than the support of its premises or conclusions.

Mushroom context

In order to compare our algorithm with Pasquier *et al.*'s algorithm, we evaluate it on the mushroom benchmark[4]. It contains description of mushrooms with properties like whether the mushroom is edible or poisonous, the ring number, the veil color, etc. With this kind of data, we can mine the properties implying that a muhsroom is poisonous. We find 7 premises with min_conf greater than 50%, min_sup greater than 40% for the target class=POISONOUS. We can deduce that we have to be very careful about mushrooms with partial veil, free gill attachement and only one ring.

For this context, execution times to computing frequent concepts are of the same order than the execution times on the same context of Pasquier's algorithm. For this comparison, we used the figures published by Pasquier in his PhD thesis ([Pas00]).

6 Conclusion

In this article we have proposed a parameterized algorithm with a filter function to explore frequent concepts in a context with taxonomy in order to learn association rules. We have described how to instantiate the filter function in order to find premises of rules where the target are positive examples (sufficient conditions) and negative examples (incompatible conditions). Then, experiments have shown that, in practice, taking a taxonomy into account does not negatively impact the performance and can make the computation more efficient.

The advantage of the presented method is to avoid redundancy in the intent of the computed frequent concepts thanks to the taxonomy. The resulting rules are therefore shorter and more relevant.

[4] Available at ftp://ftp.ics.uci.edu/pub/machine-learning-databases/mushroom/

References

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining associations between sets of items in massive databases. In *Proc. of the ACM-SIGMOD 1993 Int. Conf. on Management of Data*, pages 207–216, May 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499, 12–15 1994.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Joan Peckham, editor, *SIGMOD 1997, Proc. ACM SIGMOD Int. Conf. on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 255–264. ACM Press, 05 1997.

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

[CFRD06] P. Cellier, S. Ferré, O. Ridoux, and M. Ducassé. An algorithm for find frequent concepts of a formal context with taxonomy. In *CLA*, 2006.

[FR04] S. Ferré and O. Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3):383–419, 2004.

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

[Kuz04] Sergei O. Kuznetsov. Machine learning and formal concept analysis. In *ICFCA*, pages 287–312, 2004.

[Mit97] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Pas00] Nicolas Pasquier. *Data Mining : Algorithmes d'extraction et de réduction des règles d'association dans les bases de données*. Computer science, Université Blaise Pascal - Clermont-Ferrand II, January 2000.

[PBTL99] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT '99: Proc. of the 7th Int. Conf. on Database Theory*, pages 398–416. Springer-Verlag, 1999.

[PR03] Yoann Padioleau and Olivier Ridoux. A logic file system. In *Proc. USENIX Annual Technical Conference*, 2003.

[SR06] Benjamin Sigonneau and Olivier Ridoux. Indexation multiple et automatisée de composants logiciels. *Technique et Science Informatiques*, 2006.

[Zak04] Mohammed J. Zaki. Mining non-redundant association rules. *Data Min. Knowl. Discov.*, 9(3):223–248, 2004.