An Algorithm to Find Frequent Concepts of a Formal Context with Taxonomy

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. Formal Concept Analysis (FCA) considers attributes as a non-ordered set. This is appropriate when the data set is not structured. When an attribute taxonomy exists, existing techniques produce a completed context with all attributes deduced from the taxonomy. Usual algorithms can then be applied on the completed context for finding frequent concepts, but the results systematically contain redundant information. This article describes an algorithm which allows the frequent concepts of a formal context with taxonomy to be computed. It works on a non-completed context and uses the taxonomy information when needed. The results avoid the redundancy problem with equivalent performance.

1 Introduction

Formal Concept Analysis (FCA) [GW99] finds interesting clusters, called *concepts*, in data sets. FCA is based on a formal *context*, i.e. a binary relation describing a set of objects by a set of properties (*attributes*). A *formal concept* is defined by a pair $(extent, intent)$, where *extent* is the maximal set of objects that have in their description all attributes of *intent*, and *intent* is the maximal set of attributes common to the description of all objects of *extent*. Searching all concepts is, in general, costly and not always relevant. Thus some of these algorithms search for *frequent* concepts. A concept is called frequent, with respect to a threshold, if the cardinal of its extent is greater than the threshold. Algorithms have been designed in order to find frequent concepts ([STB+02]).

FCA considers attributes as a non-ordered set. There are, however, numerous cases where attribute taxonomies are genuinely available. For example, most corpus of knowledge in natural science are organized in rich taxonomies. *Conceptual Scaling* [GW99] can treat contexts with ordered attributes. A preprocessing step produces a completed context where new attributes deduced from the taxonomy are included. Namely, let o be an object with initial attribute a, if in the taxonomy a implies b, Conceptual Scaling adds attribute b to the description of o. After the transformation, usual data mining algorithms can be applied on the completed context for finding frequent concepts. However, the explicit links between initial attributes and deduced attributes are lost. As a consequence, the

resulting frequent concepts will systematically contain redundant information. This might be a problem. For example, it is not always relevant to recall that a nightingale is a Muscicapidae, order Passeriformes, class Aves, category Bird, phylum Chordata, kingdom Animalia.

In this paper, we propose an algorithm for finding frequent concepts in a context with taxonomy. The context needs not be completed, because the taxonomy is taken into account, when needed, during the computation. It is based on Bordat's algorithm which computes the concept lattice of a formal context [Bor86].

The algorithm is implemented into LISFS [PR03], a file system based on Logical Concept Analysis (LCA) [FR04], a version of FCA.

The contribution of this article is to describe, and experimentally validate, an algorithm which allows frequent concepts of a formal context with taxonomy to be computed. Thus, it is able to compute answers at the proper level of abstraction with respect to the taxonomy, without redundancy in the resulting frequent concepts.

In the following, Section 2 describes the algorithm. Section 3 gives experimental results. Section 4 concludes this paper.

2 Finding Frequent Concepts

Our algorithm is an adaptation of Bordat's algorithm [Bor86,KO02]. The differences are: firstly the strategy to explore the concept lattice; secondly the underlying data structures, and most importantly, the possibility to use a taxonomy to compute concepts.

The strategy of the method is top-down. The concept lattice is traversed by first exploring one non-explored concept whose extent has the greatest cardinal. The algorithm starts with the top concept. The taxonomy is taken into account, when needed, during the computation.

In the following, we first present the data structures used by the algorithm, then we give the details of the algorithm, its properties and we show the first 2 steps of computation on one example.

2.1 Data structures

The algorithm manages 2 data structures: a set of computed frequent concepts with respect to a threshold min_sup, called **Solution**, and a set of concepts to explore called **Exploration**.

Notation: given a concept c, ext_c (resp. int_c) is the extent of c (resp. its intent). Given an intent i (resp. an extent e), $ext(i)$ is the extent of i (resp. the intent of e).

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 $p(s)$. A concept c may have several increments, but we are only

Algorithm 1 Frequent_concepts

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

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

interested in increments that lead to different frequent immediate subconcepts of c. This is approximated by a data structure $incr_c$ which contains at least all frequent immediate subconcepts of c. In this data structure, every subconcept is associated with its increment with respect to c. Thus, $incr_c$ is a mapping from subconcepts to increments, and we write $incr_c[s \to X]$ to express that the mapping is modified so that c maps to X.

An invariant for the correction of the algorithm is that

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

All elements of $incr_c$ are frequent subconcepts of c.

An invariant for completness is that

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

All frequent subconcepts of c that are not in $incr_c$ are subconcepts of a subconcept of c which is in $incr_c$.

Structure $incr_c$ avoids to test all attributes at each step. Indeed, the set of increments is reducing when the lattice is explored top-down. Therefore, $incr_c$ avoids to test a lot of irrelevant attributes, by storing relevant choice points from the previous step in the computation. In practice, concepts are represented by their extent, so that $incr_c$ is represented by a trie indexed by extents.

2.2 Algorithm

Algorithm Frequent_concepts computes all frequent concepts, exploring the concept lattice top-down. **Solution** is initially empty (step 1). The top concept, la-

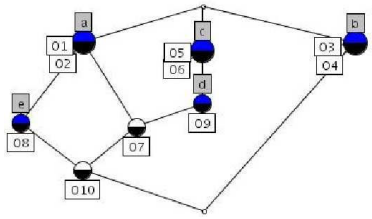

Fig. 1. Concept lattice without taxonomy.

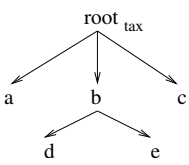

Fig. 2. Taxonomy of the example.

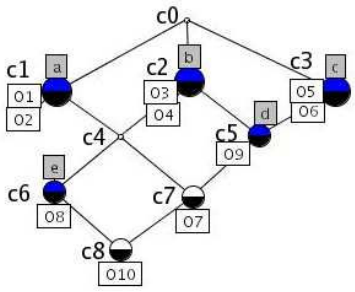

Fig. 3. Completed concept lattice.

belled by the root of the taxonomy ($root_{tax}$), is put in **Exploration** (step 2). At each iteration of the while loop, an element of **Exploration** with the largest possible extent is selected: $((ext_s \rightarrow X), int_{p(s)}, incr_{p(s)})$ (step 4), where $(ext_s \rightarrow X)$ is an element of $incr_{p(s)}$.

First, the intent of s is computed by completing $(int_{p(s)} \cup X)$ (step 5) with successors of X in the taxonomy. $succ_{tax}(X)$ returns immediate successors of attributes of X in the taxonomy and $succ^{+}_{tax}$ is the transitive closure. This is here that the elimination of redundant attributes takes place, thanks to $\cup_{tax}$. $\cup_{tax}$ is the union of two sets of attributes with elimination of redundancies due to the taxonomy.

Second, the increments of s are computed by exploring the increments of $p(s)$ (step 6) and the successors of the attributes of X in the taxonomy (steps 7-10). Indeed, the first are still possible increments for s. For each candidate increment, the algorithm checks whether it actually leads to a frequent subconcept. Finally, **Exploration** (steps 13-14) and **Solution** (step 16) are updated.

The context and the taxonomy of an example are given in Figure 1 and Figure 2. Figure 3 shows the completed context, i.e. the explored lattice.

For this example, we assume min_sup=3, and we give the first 2 steps of computation. Initially, **Solution** and **Exploration** are:

- **Solution** = $\emptyset$
- **Exploration** = $\{((\mathcal{O} \rightarrow \{root_{tax}\}), \emptyset, \emptyset)\}$.

First step: the top of the lattice is explored, i.e. s=c_0. Increments of s are computed from the taxonomy only, as there is no predecessor concept:

- $incr_{c_0}$ = { ($\{o_3, o_4, o_7, o_8, o_9, o_{10}\} \rightarrow \{\text{b}\}$), ($\{o_1, o_2, o_7, o_8, o_{10}\} \rightarrow \{\text{a}\}$), ($\{o_5, o_6, o_7, o_9, o_{10}\} \rightarrow \{\text{c}\}$)}
- **Solution** = $\{c_0\}$
- **Exploration** = {(($\{o_3, o_4, o_7, o_8, o_9, o_{10}\} \rightarrow \{\text{b}\}$), $\emptyset$, $incr_{c_0}$), (($\{o_1, o_2, o_7, o_8, o_{10}\} \rightarrow \{\text{a}\}$), $\emptyset$, $incr_{c_0}$), (($\{o_5, o_6, o_7, o_9, o_{10}\} \rightarrow \{\text{c}\}$), $\emptyset$, $incr_{c_0}$)}.

Second step: an element of **Exploration** with the largest possible extent is explored: $s = c_2$, $p(s) = c_0$. In order to compute $incr_{c_2}$, we have to consider the elements of $incr_{c_0}$ and the elements in the taxonomy.

- $incr_{c_2}$ = { ({o_7, o_8, o_{10}} → {a}), ({o_7, o_9, o_{10}} → {c, d}), ({o_8, o_{10}} → {e})}
- **Solution** = {c_0, c_2}
- **Exploration** = {(({o_1, o_2, o_7, o_8, o_{10}} → {a}), ∅, $incr_{c_0}$), (({o_5, o_6, o_7, o_9, o_{10}} → {c}), ∅, $incr_{c_0}$), (({o_7, o_9, o_{10}} → {c,d}), {b}, $incr_{c_2}$), (({o_7, o_8, o_{10}} → {a}), {b}, $incr_{c_2}$)}.

In the second step, attributes d and e are introduced as successors of attributes b, and attributes a and c are introduced as increments of c_0, the predecessor of c_2.

Increment {e} is eliminated because it leads to an unfrequent concept. Attributes c and d are grouped into a single increment because they lead to the same subconcept. This ensures that computed intents are complete.

2.3 Properties

The algorithm has two properties: 1) it computes all frequent concepts; 2) all intents of computed concepts are maximal and without redundancy according to the taxonomy.

The first property is given by the fact that every frequent concept is a subconcept of a frequent concept (except top) [PBTL99], and the concepts in **Exploration** are treated from the largest (with respect to the cardinal of the extent) to the smallest.

The second property (the intent of a computed concept is without redundancy), is given by the use of $\cup_{tax}$ which explicitly removes redundancy. In addition, when computing $incr_s$, the increments from $p(s)$ and from the taxonomy which lead to the same concept are grouped together.

3 Experiments

The algorithm is implemented in the CAML language inside LISFS [PR03]. In LISFS, attributes can be ordered to create a taxonomy (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.

We study a context with taxonomy about Java methods[1]. The context contains 5 526 objects which are the methods of java.awt. They are described by their input and output types, visibility modifiers, exceptions, keywords extracted from their identifiers, and keywords from their comments. The context has 1 624 properties. Due to the class inheritance, the context has a natural hierarchy, i.e. a taxonomy. There are 134 780 concepts but few of them are really frequent.

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

For this context, the execution time is proportional to the number of found concepts. For example, with a threshold min_sup of 5%, 189 frequent concepts are computed in 8s and taking into account the taxonomy to compute intent allows to reduce 39% of irrelevant attributes.

In order to study the impact on the performance, of taking into account the taxonomy, we test the method on a context without taxonomy, using the mushroom benchmark[2]. The mushroom context has 8 416 objects and 127 different properties. The computation time is similar to the results of the algorithms Close and A-Close on the same data [Pas00], for example with a threshold min_sup of 10%, 4 793 concepts are computed in 76s. This shows that in practice, taking into account the taxonomy does not negatively impact the performance.

4 Conclusion

We have proposed an algorithm to compute all frequent concepts in a context with taxonomy. The main advantage of the presented algorithm is to avoid redundancies due to the taxonomy, in the intents of the computed frequent concepts. The resulting concepts are therefore more relevant. Experiments have shown that, in practice, taking a taxonomy into account does not negatively impact the performance.

References

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

[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.

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

[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.

[STB+02] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Computing iceberg concept lattices with TITANIC. *Data Knowl. Eng.*, 42(2):189–222, 2002.

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