Information retrieval by on-line navigation in the latticial space-search of a database, with limited objects access

Ch. Demko♦★, K. Bertet♦

♦ L3I - Université de La Rochelle - av Michel Crépeau - 17042 La Rochelle
`cdemko,kbertet@univ-lr.fr`
★ Joomla! Production Leadership Team
`christophe.demko@joomla.org`

Abstract. We propose in this paper basic operations with limited access to the objects in the table, which can improve the computation time. Experiments were conducted with Joomla!, a content management system based on relational algebra, and located on a MySQL database. This work follows the results presented in [5].

keywords: concept lattice ; databases ; algorithm ; closure operator

1 Introduction

Galois lattice is a graph providing a representation of all the possible correspondences between a set of *objects* (or examples) O and a set of binary *attributes* (or features) I. *Galois lattices* (or *concept lattices*) were first introduced in a formal way in the graph and ordered structures theory [2,1,4].

The concept lattice is a rich and flexible navigation structure automatically derived from the context, and can therefore be considered as a dynamic and complete space search enables data description while preserving its diversity. Querying and navigation can be freely combined: to each user request corresponds a concept of the lattice as answer ; the user can then improve its search either by amending its request, or by on-line browsing arround the concept in the lattice structure.

In this paper, we propose an implementation of these basic operations with Limited Object Access, aiming to improve time computation for a large amount of objects in large databases.

This paper is organized as follows. In section 2, we describe the concept lattice and the closed set lattice. In section 3, we present the motivations that have conducted this word. In section 4, we describe our basic operations with limited object access. In section 5, we present some experiments.

2 Concept lattice: definition and generation

Definition. The *concept lattice* is a particular graph defined and generated from a a *binary table* (also denoted a *formal context*) C described by a relation R between a set of objects O and a set of attributes I. We associate to a set of objects $A \subseteq O$ the set $f(A)$ of attributes in relation R with the objects of A:

$$f(A) = \{y \in I \mid xRy \ \forall\ x \in A\}$$

Dually, to a set of attributes $B \subseteq I$, we define the set $g(B)$ of objects in relation with the attributes of B:

$$g(B) = \{x \in O \mid xRy \ \forall\ y \in B\}$$

These two functions f and g defined between objects and attributes form a *Galois correspondence*. A *formal concept* represents maximal objects-attributes correspondences (following relation R) by a pair (A, B) with $A \subseteq O$ and $B \subseteq I$, which verifies $f(A) = B$ and $g(B) = A$. The whole set of formal concepts thus corresponds to all the possible maximal correspondences between a set of objects O and a set of attributes I. Two formal concepts (A_1, B_1) and (A_2, B_2) are in relation when they verify the following inclusion property:

$$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow \left\| \begin{array}{l} A_2 \subseteq A_1 \\ \text{(equivalent to } B_1 \subseteq B_2) \end{array} \right.$$

3 Motivations

The existence of a concept lattice underlying a data table allows to consider an information retrieval strategy combining querying and navigation by a browsing in this lattice as in an area of research. Indeed, the user request is a concept of a lattice, and the user can then improve its search either by amending its request, or by browsing in the lattice structure. From a computational point of view, such a mechanism of information retrieval by request and by navigation requires two main operations:

1. Generation of the smallest concept $(g(B), f(g(B))$ containing a given subset B of attributes: B is the request, the objects part $f(g(B))$ of the concept is the answer, $g(B)$ are inferred attributes.
2. Generation of the immediates successors of a given concept (A, B) for a browsing in the lattice by computing the inclusion-maximal in the set system $\mathcal{F}_A$ defined on O by $\mathcal{F}_A = \{g(x + B) \ : \ x \in I \backslash B\}$.

Large data are often described by a huge amount of objects, as in databases for example where the number of recordings (i.e. objects) can be huge, indexed using sophisticated key-indexation techniques. We propose in this paper an improvement of these basic operations with two limited objects access strategies:

A storage improvment by considering the restriction of the concept lattice to the attributes, namely the *closed set lattice.*

A computation improvment by considering in the operations the cardinality of the subset of objects instead of the subset itself, using the *count function.*

```
Name: Immediates_Successors_LOA
Data: A context $K$ ; A closed set $B$ of the closed set lattice $(\mathbb{C}_I, \subseteq)$ of $K$
Result: The immediate successors of $B$ in the lattice
begin
    initialize the $Succ_B$ family to an empty set;
    foreach $x \in I \setminus B$ do
        add = true;
        foreach $X \in Succ_B$ do
            \\ Merge x and X in the same potential successor
            if $c(B + x) = c(B + X)$ then
                if $c(B + X + x) = c(B + x)$ then
                    replace $X$ by $X + x$ in $Succ_B$; add=false; break;
                end
            end
            \\ Eliminate x as potential successor
            if $c(B + x) < c(B + X)$ then
                if $c(B + X + x) = c(B + x)$ then add=false; break;
            end
            \\ Eliminate X as potential successor
            if $c(B + x) > c(B + X)$ then
                if $c(B + X + x) = c(B + X)$ then delete $X$ from $Succ_B$
            end
        end
        \\ Insert x as a new potential successor ;
        if add then add $\{x\}$ to $Succ_B$
    end
    return $Succ_B$;
end
```

Algorithm 1: Generation of the immediate successors of a closed set in the Hasse diagram of the lattice $(\mathbb{C}_I, \subseteq)$

Closed set lattice. Instead of a concept lattice, it is possible to consider its restriction to the attributes in order to limit the storage of huge ammount of objects in each concept. A nice result establishes that any concept lattice $(\mathbb{C}, \leq_C)$ is isomorphic to the lattice $(\mathbb{C}_I, \subseteq)$ defined on the set I of attributes, with $\mathbb{C}_I$ the restriction of $\mathbb{C}$ to the attributes in each concept. The lattice $(\mathbb{C}_I, \subseteq)$ is also known as the closed sets lattice on the attributes I of a context (O, I, R), where the set system $\mathbb{C}_I$ is composed of all closed set - i.e. fixed points - for the closure operator $\varphi = g \circ f$ - i.e. a map that is isotone, extensive and idempotent):

$$\mathbb{C}_I = \{\varphi(X) \ : \ X \subseteq I\} \tag{1}$$

The closed sets $\bot = \varphi(\emptyset) = f(O)$ and $\top = I$ respectively correspond to the bottom and the top of the closed set lattice. See the survey of Caspard and Monjardet [3] for more details about closed set lattices.

Therefore, each smallest concept $(g(B), f(g(B))$ containing a given set B of attributes is replaced by the closure $\varphi(B) = f(g(B)$ on the attributes. Thus, only the attributes part is stored.

The count function. Moreover, we propose to reinforce the object access limitation by considering the cardinality of the subset $g(B)$ instead of the subset itself in the treatment. The count function c associates to any subset X of attributes the cardinality of the subset $g(X)$: $c(X) = |g(X)|$

It corresponds to the notion of support introducing in rules extraction from data-bases, and is in particular used by Titanic algorithm [6]. We use the count function c instead of the closure operator φ since c and φ possesses together nice properties, $\forall X, X' \subseteq I$:

$$X \subseteq Y \Rightarrow c(X) \geq c(Y) \tag{2}$$

$$\varphi(X) = \varphi(Y) \Rightarrow c(X) = c(Y) \tag{3}$$

$$X \subseteq Y \text{ and } c(X) = c(Y) \Rightarrow \varphi(X) = \varphi(Y) \tag{4}$$

4 Basic operations in a lattice with limited objects access

Using the closed sets lattice $(\mathbb{C}_I, \subseteq)$ instead of the whole concept lattice $(\mathbb{C}, \leq_C)$ gives raise to a storage improvement since only the attributes part is stored. Using the count function, the two main operations can therefore be reformulated as follows:

Generation of the closed-set $\varphi(B)$ of a subset B of attributes. It can be performed using the following equality:

$$\varphi(B) = B + \{x \in I \backslash B \ : \ c(B) = c(B + x)\} \tag{5}$$

This equality is a direct consequence of the third property of c together with the isotone property of φ (i.e. $X \subseteq X' \Rightarrow \varphi(X) \subseteq \varphi(X')$).

Generation of the immediates successors of a closed set B. A closed sets lattice can be generated using an algorithm similar to Bordat's algorithm, where the immediate successors of a closed set B in the Hasse diagram of the lattice are the inclusion-minimal subsets of

$$\mathcal{F}_B = \{\varphi(B + x) \ : \ x \in I \backslash B\} \tag{6}$$

In our previous work [5], we present an incremental algorithm by testing, for each attribute x of $I\backslash B$ and each already inserted potential successor $X \subseteq I\backslash B$, the inclusion between $\varphi(B+X)$ and $\varphi(B+x)$:

1. Merge x with X when $\varphi(B+x) = \varphi(B+X)$.
2. Eliminate X as potential successor of B when $\varphi(B+x) \subset \varphi(B+X)$
3. Eliminate x as potential successor of B when $\varphi(B+X) \subset \varphi(B+x)$
4. Insert x as potential successor of B when x is neither eliminated or merged with X.

The inclusion test between $\varphi(B+X)$ and $\varphi(B+x)$ can easily be performed using the count function c and the following proposition deduced from Prop. 1 in [5]:

Proposition 1. $\varphi(B+X) \subseteq \varphi(B+x) \iff c(B+X+x) = c(B+X)$

$\Rightarrow$: Consider that $\varphi(B+X) \subseteq \varphi(B+x)$. The equivalence between inclusion and intersection set operations ($C \subseteq D \iff C = C \cap D$) allows to deduce that $\varphi(B+X) = \varphi(B+X) \cap \varphi(B+x)$. Since $\varphi(B+X) \cap \varphi(B+x) = \varphi(B+X) \wedge \varphi(B+x) = \varphi(B+X+x)$, then $\varphi(B+X) = \varphi(B+X+x)$. We conclude by $c(B+X+x) = c(B+X)$ using the second property of the count function c (see Eq. 3).
$\Leftarrow$: Consider that $c(B+X+x) = c(B+X)$. By Eq. 4, and since $B+X \subseteq B+X+x$, we deduce that $\varphi(B+X+x) = \varphi(B+X)$, and we conclude by $\varphi(B+X) \subseteq \varphi(B+x)$ as above.

In the case where $\varphi(B+X) \subseteq \varphi(B+x)$, the strict inclusion has then to be tested in order to decide if x has to be deleted as potential successor, or merged with X. Using Eq. 2 and Eq. 3, this test can be performed by checking if $c(B+X) > c(B+x)$ or $c(B+X) = c(B+x)$. The case where $\varphi(B+x) \subseteq \varphi(B+X)$ is dualy tested in order to decide if X has to be deleted or not as potential successor.

The complexity of computing the immediate successors of a closed set B using the `Immediates_Successors_LOA` algorithm is:

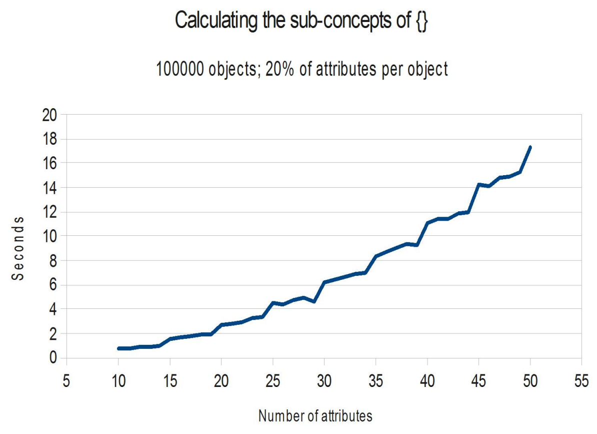
$$\frac{(|I|-|B|)(|I|-|B|)}{2} * O(c(B+X))$$

which leads to

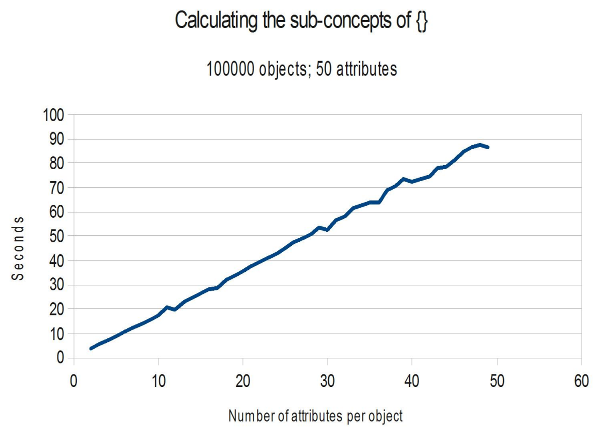
$$O((|I|-|B|)^2 * O(c(B+X)))$$

using the big O notation.

This has to be compared with $O(|I|^2 * |O|)$ of the Bordat's algorithm. In addition the cost $O(c_(B+x))$ of computing the cardinality of objects satisfying the required properties can be based on multiple keys and robust algorithms used in databases that do not need to load all data for computing a cardinality [5].



(a) 100.000 objects and attributes varying from 10 to 50, each object randomly described by 20 % of the attributes

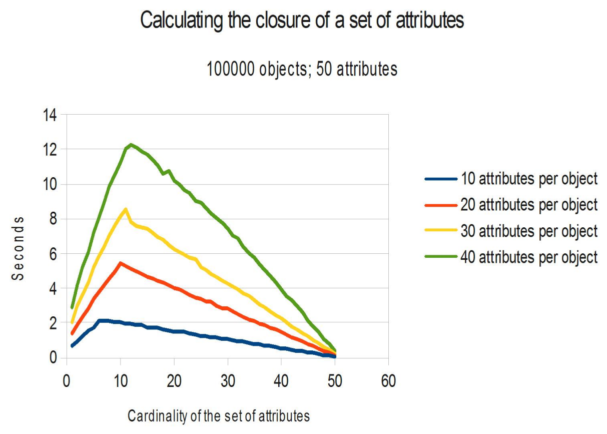


(b) 100.000 objects and 50 attributes, each object described by random attributes varying from 2 to 49

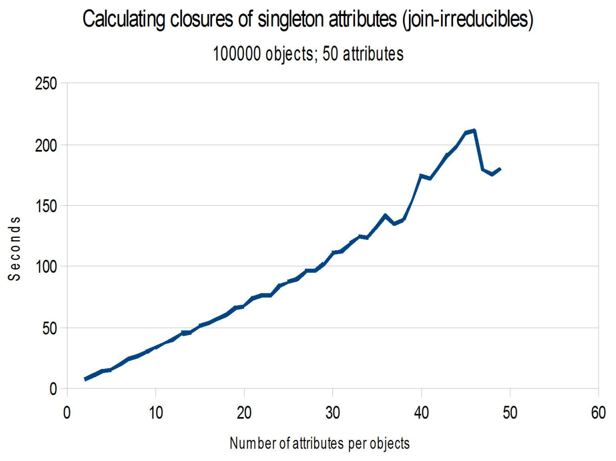
Fig. 1. Calculating the immediate successors of $\emptyset$

5 Experimentations

In the experiment, we use a dataset composed of 100.000 objects described by a random set of attributes varying from 10 to 50, each objects described by a random set of attributes. The dataset is stored in a database MySQL 5.5.17. We have implemented our algorithms using PhP 5.3.8 using a laptop with 8 processors clocked at 1.73GHz and 8Gb of memory. The counting of objects



(a) Average time of a closure generation for attributes varying from 1 to 50 with 100.000 objects and 50 attributes, each object described by 10, 20, 30 then 40 random attributes



(b) closure of attributes with 100 000 objects, 50 attributes, each object described by random attributes from 2 to 49

Fig. 2. Closures computation

satisfying a set of properties is realised by the SQL request comparing indexes with a constant:

```
select count (*) from att1=1 and att2=1
```

We compare the processing time of our algorithms in the following cases:

Immediate successors generation: we compute the immediate successors of the bottom concept in the two following cases:

1. 100.000 objects and attributes varying from 10 to 50, each object randomly described by 20% of the attributes (see Fig. 1(a))
2. 100.000 objects and 50 attributes, each object described by random attributes varying from 2 to 49 (see Fig. 1(b))

Closure generation: We compute closures in the following cases:

1. We compute the average time of a closure generation for attributes varying from 1 to 50 in the following case, with 100.000 objects and 50 attributes, each object described by 10, 20, 30 then 40 random attributes. (see Fig. 2(a))
2. We compute the closure of a singleton attribute with 100 000 objects, 50 attributes, each object described by random attributes from 2 to 49.

Trends of our results are consistent with theoretical complexities of the algorithms. Deporting a part of the calculation in the SQL engine, we believe that the index dedicated to counting object can improve performance. This results renforces those obtained in [5] on the efficience of the key-indexation techniques in SQL. In addition, new technologies related to cloud computing will also divide the workload of the SQL engine on demand.

6 Conclusion

In this paper, we described two basic algorithms for browsing in a concept lattice with limited objects access. By separating the counting from the rest of the algorithm, new systems for exploring concept lattices can now rely on optimization algorithms used in relational databases. If the tests we will realize on PostgreSQL and MySQL databases are successfull in terms of manipulating a huge amounts of data, we plan to propose a library for extending content management system such as Joomla!.

References

1. M. Barbut and B. Monjardet. *Ordres et classifications : Algèbre et combinatoire.* Hachette, Paris, 1970. 2 tomes.
2. G. Birkhoff. *Lattice theory.* American Mathematical Society, 3d edition, 1967.
3. N. Caspard and B. Monjardet. The lattice of closure systems, closure operators and implicational systems on a finite set: a survey. *Discrete Applied Mathematics*, 127(2):241–269, 2003.
4. B.A. Davey and H.A. Priestley. *Introduction to lattices and orders.* Cambridge University Press, 2nd edition, 1991.
5. C. Demko and K. Bertet. Generation algorithm of a concept lattice with limited object access. In *Proc. of Concept lattices and Applications (CLA'11)*, pages 113–116, Nancy, France, October 2011.
6. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with TITANIC. *Data and Knowledge Engineering*, 42(2):189–222, August 2002.