GenAll Algorithm: Decorating Galois lattice with minimal generators

Sondess Ben Tekaya, Sadok Ben Yahia and Yahya Slimani

Département des Sciences de l'Informatique, Faculté des Sciences de Tunis
Campus Universitaire, 1060 Tunis, Tunisie.
sondess.bentekaya@laposte.net,{sadok.benyahia,yahya.slimani}@fst.rnu.tn

Abstract. The problem of relevance and the usefulness of extracted association rules is becoming of primary importance, since an overwhelming number of association rules may be derived. This paper proposes an algorithm, called GenAll, to build a formal concept lattice, in which each formal concept is "decorated" by its minimal generators. The main characteristic of this algorithm is to use a refinement process of upper cover lists to determine, in a simultaneous manner, the set of formal concepts, their underlying partial order and the set of minimal generators associated to each formal concept. Experimental results have showed that the proposed algorithm is specially efficient for dense formal contexts compared to that of Nourine et *al.*. Response times pointed out by GenAll algorithm largely outperform those of Nourine et *al.*.
Keywords: Data Mining, Formal Concept Analysis, Generic association rules, Generator.

1 Introduction

Data Mining is a discipline which aims to discover regularities, in voluminous datasets, commonly expressed by association rules [1]. Several studies in particular underlined the prohibitive number of association rules drawn from even reasonably sized datasets [2]. This fact bootstrapped the development of more acute techniques or methods to reduce the size of the reported rule sets. In this context, the battery of results provided by the Formal Concept Analysis (FCA) permitted to define "irreductible" nucleus of association rule subset better known as generic bases. These bases constitute reduced sets of informative rules allowing to preserve the most relevant rules, without loss of information. The generation of these informative association rules relies on the extraction of formal concepts, its associated minimal generators and the underlying partial order [3]. In this paper, our main claim is that there is no single algorithm allowing to obtain the generic base of association rules. In fact, a critical survey of dedicated literature pointed out the following remarks:

1. **Oriented Data Mining algorithms** generate the set of closed itemsets[1] and the set of associated minimal generators [4–6]. The underlying order

[1] or equivalently the set of formal concept intents.

Radim Bělohlávek, Václav Snášel (Eds.): CLA 2005, pp. 166–178, ISBN 80–248–0863–3.

between closed itemsets is badly missing or not of interest. To obtain the underlying partial relation, one can use the algorithm proposed by Valtchev *et al.* [7] to construct the covering graph.

2. **Oriented formal concept algorithms** generate the set of formal concepts and the underlying order [8]. Here, the set of associated minimal generators is missing. In this case, we can apply the JEN algorithm [9] to catch out minimal generators given that the covering graph is already generated.

In this paper, we propose a new algorithm, called GenAll, merging the above mentioned views. Indeed, aiming to derive generic bases of association rules, we propose to build the graph in which each formal concept is decorated by its associated minimal generators. As a starting point, we choose to improve the algorithm presented by Nourine *et al.* [10]. This choice can be justified by the fact that the latter presents the best theoretical complexity, even though practical experiments highlighted that it is the worst compared to other algorithms [8, 11]. Hence, a careful review of Nourine *et al.* algorithm showed that it operates in two steps: An original approach, based on a special trie, to discover and store formal concepts. A costly repeated access to originally resident disk to compute the underlying order between formal concepts. However, the bad results of Nourine *et al.* algorithm are not a fatality and we feel that its bad performances can be largely improved using the GenAll algorithm proposed in this paper.

The remainder of the paper is organized as follows: Section 2 briefly sketches the basic FCA constructs and stresses on the link between the notion of minimal blocker and minimal generator. Section 3 discusses in depth the GenAll algorithm. Results of experiments carried out on benchmarking datasets are reported in Section 4. Finally, section 5 concludes this paper and points out some research directions for future work.

2 Basic notions

Formal Concept: Let us consider an *formal context* $\mathcal{K} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$, where $\mathcal{O}$ is a set of objects, $\mathcal{I}$ is a set of attributes (or items) and $\mathcal{R}$ is a binary relation between the objects and the attributes ($\mathcal{R} \subseteq \mathcal{O} \times \mathcal{I}$). Each couple $(o, a) \in \mathcal{R}$ expresses that the transaction $o \in \mathcal{O}$ contains the attribute $a \in \mathcal{I}$. Within a context, objects are denoted by numbers and attributes by letters.

We define two functions summarizing links between subsets of objects and subsets of attributes induced by $\mathcal{R}$, that map sets of objects to sets of attributes and *vice versa*.
Thus, for a set $O \subseteq \mathcal{O}$, we define: $\phi(O) = \{a | \forall o, o \in O \Rightarrow (o, a) \in \mathcal{R}\}$ and for a set $A \subseteq \mathcal{I}$, $\psi(A) = \{o | \forall a, a \in A \Rightarrow (o, a) \in \mathcal{R}\}$. Both functions ϕ and ψ form a *Galois connection* between the sets $\mathcal{P}(\mathcal{I})$ and $\mathcal{P}(\mathcal{O})$ [12]. Consequently, both compound operators of ϕ and ψ are closure operators, in particular $\omega = \phi \circ \psi$ is a closure operator.

A *formal concept* is a pair $C_{\mathcal{K}} = (O, A)$, where O is called *extent*, and A is a closed itemset[2], called *intent*. Furthermore, both O and A are related through the Galois connection, *i.e.*, $\phi(O) = A$ and $\psi(A) = O$. Let an itemset $A \subseteq \mathcal{I}$, the support of the itemset A in the context $\mathcal{K}$ is defined by: $support(A) = \frac{|\psi(A)|}{|\mathcal{O}|}$.
Face: Let $C_{\mathcal{K}} = (O, A)$ be a formal concept and let $pred_i(C_{\mathcal{K}})$ be the i^{th} immediate predecessor of $C_{\mathcal{K}}$ in a Galois lattice extracted from a context $\mathcal{K}$. The i^{th} face of the formal concept $C_{\mathcal{K}}$ corresponds to the difference between its intent and the intent of its i^{th} predecessor [13].

Let p be the number of immediate predecessors of the formal concept $C_{\mathcal{K}}$. The family of faces $F_{C_{\mathcal{K}}}$ of the formal concept $C_{\mathcal{K}}$ is expressed by the following relation: $F_{C_{\mathcal{K}}} = \{A - Intent(pred_i(C_{\mathcal{K}})\}, i \in \{1, \ldots, p\}$ [13]. For example, according to the formal concept lattice presented on the figure 3, $F_{(abde,23)_{\mathcal{K}}} = \{b, e\}$.
Minimal blocker: Let $G = \{G_1, ..., G_n\}$ be a family of n sets. A *blocker* B of the family G is a set where the intersection with all sets $G_i \in G$ is not empty [13]. A blocker B of family of $G = \{G_1, ..., G_n\}$ is said to be minimal if $\nexists B_1 \subset B$ and $\forall G_i \in G, B_1 \cap G_i \neq \emptyset$ [13]. Given the family of faces $F_{(abde,23)_{\mathcal{K}}} = \{b, e\}$, the minimal blocker $B = \{be\}$ where the intersection with the two faces is always different of the empty set.
Generator: Let $C_{\mathcal{K}}$ be a formal concept and $F_{C_{\mathcal{K}}}$ its family of faces. The set G of the generators associated with the intent A of the formal concept $C_{\mathcal{K}}$, corresponds to the minimal blockers associated to the family of faces $F_{C_{\mathcal{K}}}$ [13]. Equivalently, An itemset $g \subseteq \mathcal{I}$ is *called minimal generator* of a formal concept $C_{\mathcal{K}} = (O, A)$, if and only if $\omega(g) = A$ and $\nexists g_1 \subset g$ such that $\omega(g_1) = A$ [14].
Minimal association rules: An association rule is an assorted statistical metric implication between itemsets of the form $r : X \Rightarrow (Y - X)$ in which X and Y are frequent itemsets (their supports are at least equal to a minimal threshold, called *minsup*), and $X \subset Y$. Itemsets X and $(Y - X)$ are called, respectively, *antecedent* and *conclusion* of the rule r. The valid association rules are those whose measure of confidence $Conf(r) = \frac{support(Y)}{support(X)}$ is greater than or equal to the minimal threshold of confidence, named $minconf$. An association rule whose confidence is equal to 1 is called *exact association rule*. Otherwise it is called *approximative association rule*. Bastide *et al.* characterized what they called "*the generic base for exact association rules*" (adapting the global implication base of *Guigues and Duquenne* [15]). The generic base for exact association rules is defined as follows :
Trie[3]: It is a research tree, whose elements are stored in a condensed way. The "trie" used by the Nourine algorithm presented in [10] is a special trie, whose edges are labelled by attributes of formal concepts, and only some nodes are labelled by the extents of formal concepts. The way going from the root node towards a labelled node by an extent form a formal concept. As shown in Figure 1, $(acde, 34)$ is a formal concept.

[2] an itemset is a set of items or attributes.
[3] From reTrieval.

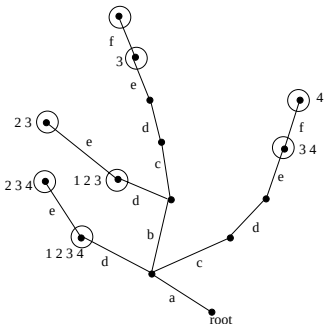

Fig. 1. Example of trie structure

3 The proposed GenAll algorithm

In this section, we present a new algorithm, called GenAll, to derive generic bases of association rules. To do so, it gathers in a **single pass** all the required informations which are : the set of formal concepts and their associated minimal generators, the underlying partial order.

The GenAll algorithm is inspired in particular from the first part of the Nourine *et al.* algorithm [10], stressing on the construction of the formal concept trie. The choice of this algorithm is motivated by the fact that it presents the best theoretical complexity [8, 10, 11]. In addition to its linear complexity, the originality of this algorithm resides in its way to discover formal concepts and their storage in a special trie.

However, this algorithm presents some disadvantages, that we can summarize as follows:

1. A costly systematic back to the dataset, originally resident disk, to compute the covering graph of the formal concepts,
2. Ignore the determination of minimal generators associated to each formal concept.

To avoid these disadvantages, we present a new algorithm, relying on the construction of a trie to calculate on the fly, the set of formal concepts, their associated minimal generators and the partial order between these formal concepts. Notations used by GenAll algorithm are given in Table 1, while the pseudo-code is illustrated by *Algorithm* 1. In each iteration, the algorithm builds the set of formal concepts, the set of candidate minimal generators and a potential order, which has to be later refined, between the already discovered formal concepts. Each transaction is passed only once. Each iteration consists of two steps. The first calculates the formal concepts and generates a potential order between formal concepts, whereas the second aims to refine the order and to compute the associated minimal generators. These two steps are discussed in the following.

Algorithm 1 Construction of formal concept lattice decorated by the minimal generators

1: **Algorithm** GenAll
Input : Formal context $\mathcal{K}$
Output : Lexicographic tree of $\mathcal{F}(trie), ImmSucc$ of each formal concept, $list\_gen$
Begin
2: $\mathcal{F} = (\emptyset, \mathcal{I})$
{/* step 1: Generation of formal concepts */}
3: **For** Each transaction $t \in \mathcal{K}$ **Do**
4: $\mathcal{L} = \emptyset$
5: **For** Each concept $C_i \in \mathcal{F}$ **Do**
6: $C.intent = C_i.intent \cap t$
7: **If** $C.intent \notin \mathcal{F}$ **Then**
8: $\mathcal{F} = \mathcal{F} \cup C$
9: $C.extent = C_i.extent \cup t.TID$
10: $C.ImmSucc = \{t\} \cup \{C_i\} \setminus \{C\}$
11: $\mathcal{L} = \mathcal{L} \cup C$ {/* Sorted insertion */}
12: **Else**
13: $C.extent = C.extent \cup C_i.extent \cup t.TID$
14: **If** $C \notin \mathcal{L}$ **Then**
15: $\mathcal{L} = \mathcal{L} \cup C$ {/* Sorted insertion */}
16: **End If**
{/* Updating $C.ImmSucc$ */}
17: $C.LP = \{t\} \cup \{C_i\} \setminus \{C\}$
18: **For** Each $P_i \in C.LP$ **Do**
19: **For** Each $Succ \in C.ImmSucc$ **Do**
20: Compare-concept$(Succ, P_i)$
21: **End For**
22: **End For**
23: **End If**
24: **End For**{/* Refinement of immediate successor list and determination of minimal generators*/}
25: **For** Each concept $C_i \in \mathcal{L}$ **Do**
26: $C_i.ImmSucc =$ find-Succ$(C_i, \mathcal{L})$
27: **For** Each $Succ \in C_i.ImmSucc$ **Do**
28: $face = Succ \setminus C_i$
29: **For** Each $face_i \in Succ.list\_face$ **Do**
30: $Succ.list\_face =$ compare-face$(face, face_i)$
31: **End For**
32: **End For**
33: **End For**
34: **End For**

Structure	Fields	Description
$\mathcal{F}$		Family of the formal concepts.
$\mathcal{L}$		Lists of formal concepts calculated in an iteration i.
LP		Intermediary list of formal concepts.
C		Formal concept.
	$intent$	Intent of the formal concept
	$extent$	Extent of the formal concept
	$ImmSucc$	List of immediate successors of the formal concept
	$list\_face$	List of faces of the formal concept
	$list\_gen$	List of the minimal generators of the formal concept

Table 1. Notations

3.1 Generation of formal concepts

In this step (lines 4-24), we start by initializing the $\mathcal{L}$ list with the empty set. This list will be useful for the refinement of the set of the immediate successors of each formal concept found in an iteration. To calculate the set of formal concepts, we perform an intersection between the intent of each formal concept of the family $\mathcal{F}$ and each transaction of the dataset. Two cases are distinguishable:

1. **The intent does not exist in the family $\mathcal{F}$ (a new formal concept is found):** it must then be added to the family $\mathcal{F}$. Then, the associated formal concept extent is calculated (line 9). We can notice that all the transactions of the formal context are formal concepts. Hence, the result of the intersection of the transactions with the formal concept where its intent is equal to the set of attributes of the formal context, is equal to the attributes of the transaction.
   Potential immediate successors of this new formal concept are firstly initialized with the formal concept utilized to generate it (line 10). Indeed, to determine potential immediate successor of a formal concept, we should distinguish two particular cases:
   If the generated formal concept is equal to the transaction, then only the formal concept used in this intersection can be an immediate successor. Otherwise both the transaction and the formal concept are considered as potential immediate successors of the formal concept. Thereafter, this new formal concept is added to the list $\mathcal{L}$ (line 11). It is important to mention that this insertion is performed by maintaining an ascending order of formal concept intents cardinality.
2. **The intent already exists in the family $\mathcal{F}$:** the formal concept extent (line 13) should be updated, and we check whether the concept is already existing in the list $\mathcal{L}$ (lines 14-15). Aiming to update the immediate successors list $ImmSucc$ of the formal concept, and given that for each formal concept we maintain an $ImmSucc$ list, we build a list LP containing the formal concepts used to generate it. This list is necessary, to be able to

make comparisons and to update the $ImmSucc$ list. Indeed, for each element in LP and for each element in $ImmSucc$ list, we test the inclusion of these formal concepts using the COMPARE-CONCEPT function (line 20). The COMPARE-CONCEPT function is applied to update the $ImmSucc$ list of the formal concepts under consideration. This list has to be modified in two cases: the first case, the element of LP is smaller (in term of inclusion) than one element of $ImmSucc$. In this case, the old immediate successor will be replaced by the new one. The second case, the two elements are incomparable: a new successor will then be added to the $ImmSucc$ list of the formal concept under concern.

3.2 Refinement of immediate successor list and determination of minimal generators

We notice that the formal concepts, found in an iteration, represent a branch of the lattice, *i.e.*, for each formal concept (except the largest one), we find another formal concept calculated in the same iteration, which covers it. For that, we traverse the $\mathcal{L}$ list of formal concepts found in an iteration and for each formal concept we call the FIND-SUCC function (lines 25-26). This function is based on the fact that a formal concept of cardinality n (*i.e.*, the length of its intent is equal to n) is at least covered by a formal concept of cardinality $(n + 1)$ or higher. For that, and since the $\mathcal{L}$ list is sorted according to the cardinality's intent ascending order, we seek for each formal concept of cardinality n, a formal concept which covers it in the list of cardinality $(n+1)$. In the event of defect, we pass to the cardinality $(n + 2)$, until we find a minimal covering formal concept. Then, we compare these formal concepts to update the $ImmSucc$ list.

Once the list of the immediate successors ($ImmSucc$) of the current formal concept is built, we traverse this list and for each immediate successor we have to compute the set of its minimal generators. To obtain such result, we calculate the associated faces of successors (line 28), then we compare them to the corresponding list of faces by calling the COMPARE-FACE function (line 30). The set of faces $list\_face$ of the formal concept (immediate successor) can be modified in two cases as illustrated by the following:

1. The $face$ is smaller (in term of inclusion) than an element of $list\_face$: in this case, we calculate the $difference\_face$, and we replace the old face by the new one. If the obtained $difference\_face$ does not exist in one of the faces of this formal concept, then we remove each generator containing this difference. This removal is necessary, since a non existing attribute in the faces cannot exist in the generators.
2. The $face$ is incomparable with all the faces of $list\_face$: in this case, it will be added to $list\_face$.

When the faces list is updated, the COMPUTE-MIN-BLOCKERS function of the JEN algorithm [9] is applied for determining the minimal generators.

Example
Let us consider the Formal context illustrated by Figure 3 with the set of attributes $\mathcal{I} = \{a, b, c, d, e, f\}$ and the set of transactions of the formal context, denoted from 1 to 4. Firstly, the family $\mathcal{F}$ is initialized with the set of attributes

$\mathcal{R}$	a	b	c	d	e	f
1	×	×		×		
2	×	×		×	×	
3	×	×	×	×	×	
4	×		×	×	×	×

Fig. 2. Formal context $\mathcal{K}$.

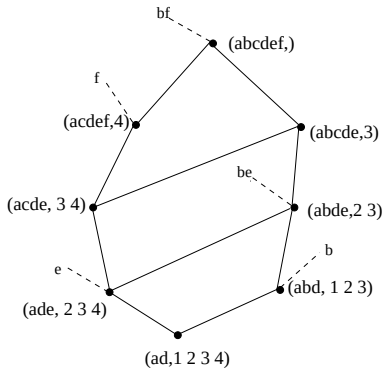

Fig. 3. Formal concept lattice extracted from the context $\mathcal{K}$ decorated by some minimal generators.

(line 2). It present the top formal concept of the lattice.

First step: Generation of formal concepts
Each formal concept of $\mathcal{F}$ is handled individually. Let us consider the first transaction $\{abd\}$:
Transaction 1:
The application of the intersection operation with the single formal concept of the family $\mathcal{F}$ gives raise to a new formal concept with an intent equal to $\{abd\}$. The extent of this new formal concept, denoted C_2, will be later computed.
Thus, the family $\mathcal{F}$ is updated by adding C_2:
$\mathcal{F} = \{(abcdef, \emptyset), (abd, \emptyset)\}$.
At first, the set of immediate successor of C_2 is initialized with $C_1 = (abcdef, \emptyset)$ and the extent of C_2 is initialized with union of $C_1.extent$ and the transaction ID under concern.
Hence, $C_2.extent = \{1\}$ and $C_2.ImmSucc = \{(abcdef, \emptyset)\}$ and the $\mathcal{L}$ list is initialized with $\{(abd, 1)\}$.
Second step: Refinement of immediate successor list and determination of minimal generators
For the single formal concept in $\mathcal{L} = \{(abd, 1)\}$ we haven't to update the $ImmSucc$ list of this concept, and $(abd, 1).ImmSucc = \{(abcdef, \emptyset)\}$. For each immediate successor we compute its minimal generators set using the Compute-Min-blockers function of the JEN algorithm [9].
The $face$ of the formal concept $(abcdef, \emptyset) = cef$. Then $C_1.list_gen = \{c, e, f\}$.

Transaction 2:
Let now,consider the second transaction $\{abde\}$:
$\{abcdef\} \cap \{abde\} = \{abde\}$, $\{abde\}$ is an intent of a new formal concept.
$\mathcal{F} = \{(abcdef, \emptyset), (abd, 1), (abde, \emptyset)\}$, $C_3.extent = \{2\}$, then:
$\mathcal{F} = \{(abcdef, \emptyset), (abd, 1), (abde, 2)\}$
$C_3.ImmSucc = \{(abcdef, \emptyset)\}$ and $\mathcal{L} = \{(abde, 2)\}$.
These process above is repeated for the second formal concept in the family $\mathcal{F}$: $\{abd\} \cap \{abde\} = \{abd\}$, this set is an intent of an existing formal concept. Then we update the extent of this formal concept (line 13): $C_2.extent = \{12\}$ and $\mathcal{L} = \{(abd, 12), (abde, 2), \}$
$C_2.LP = \{abde\}$. Using COMPARE-CONCEPT function we have $\{abde\} \subset \{abcdef\}$ then we update $C_2.ImmSucc = \{abde\}$.
The second step is performed and we obtain:
$C_2.ImmSucc = \{(abde, 2)\}$ and $C_3.ImmSucc = \{(abcdef, \emptyset)\}$.
At the end of the thirst transaction we obtain:
$\mathcal{F} = \{(abcdef, \emptyset), (abd, 123), (abde, 23), (abcde, 3)\}$

$C_2.ImmSucc = \{(abde, 23)\}$ $\quad$ $C_3.ImmSucc = \{(abcde, 3)\}$
$C_4.ImmSucc = \{(abcdef, \emptyset)\}$
$C_1.list\_gen = \{f\}$ $\quad$ $C_2.list\_gen = \{a, b, d\}$
$C_3.list\_gen = \{e\}$ $\quad$ $C_4.list\_gen = \{c\}$

Transaction 4$= \{acdef\}$
At the first step we obtain:
$\mathcal{F} = \{(abcdef, \emptyset), (abd, 123), (abde, 23), (abcde, 3), (acdef, 4), (ad, 1234),$
$(ade, 234), (acde, 34)\}$
$C_5.ImmSucc = \{(abcdef, \emptyset)\}$ $\quad$ $C_6.ImmSucc = \{(abd, 123), (acdef, 4)\}$
$C_7.ImmSucc = \{(abde, 23), ((acdef, 4)\}$ $\quad$ $C_8.ImmSucc = \{(abcde, 3), (acdef, 4)\}$
$\mathcal{L} = \{(ad, 1234), (ade, 234), (acde, 34), (acdef, 4)\}$. Let consider the first formal concept in $\mathcal{L}$ list, $(ad, 1234)$. The later formal concept, has as immediate successor the formal concept $(acdef, 4)$. But we find in $\mathcal{L}$ list, a formal concept with cardinality 3 who can cover the formal concept $(ad, 1234)$. Then we replace $(acdef, 4)$ by $(ade, 234)$ because $\{ade\} \subset \{acdef\}$. The same process is done for the other formal concepts. We have at the end:

$C_5.ImmSucc = \{(abcdef, \emptyset)\}$ $\quad$ $C_6.ImmSucc = \{(abd, 123), (ade, 234)\}$
$C_7.ImmSucc = \{(abde, 23), ((acde, 34)\}$ $\quad$ $C_8.ImmSucc = \{(abcde, 3), (acdef, 4)\}$
$C_1.list\_gen = \{bf\}$ $\quad$ $C_2.list\_gen = \{b\}$
$C_3.list\_gen = \{be\}$ $\quad$ $C_4.list\_gen = \{bc\}$
$C_5.list\_gen = \{f\}$ $\quad$ $C_6.list\_gen = \{a, d\}$
$C_7.list\_gen = \{e\}$ $\quad$ $C_8.list\_gen = \{c\}$

Figure 2 depicts the formal concept lattice associated to the formal context $\mathcal{K}$, presented in Figure 3. A sketch of some minimal generators, indicated by dotted lines, are decorating formal concepts nodes.

4 Experimental results

We implemented the Nourine *et al.* and GenAll algorithms in C on a Linux platform with a Mandrake distribution to assess their relative performances. Both algorithms used the same data structure. Our experiments were carried out on Pentium IV with CPU clock rate of 2Ghz and 512Mb of main memory. To examine the practical efficiency of our algorithm, we run experiments on real and synthetic datasets, whose characteristics are detailed in what follows.

4.1 Test data

Hence the results depend on the dataset density, algorithms were tested on two types of datasets: synthetic data, which mimic market basket, and dense datasets, which belong to the domain of statistical databases. All these test datasets are freely available on Internet[4]. *Chess* base is derived from steps of chess game. Typically, real datasets are very dense. We also chose some synthetic databases. Generally, synthetic datasets are sparse compared to real ones.

4.2 Computational experiments

In the sequel, we assess performances of Nourine *et al.* and GenAll algorithms both on sparse and dense datasets.
Case of sparse datasets: For this type of datasets, the list of successors associated to a given formal concept changes frequently. Thus, comparisons of all the list of immediate successors will be carried out. Given that the successor list of the formal concept is long, in sparse datasets compared to that of the dense datasets, the step of updating the immediate successor list consumes more time comparatively to the Nourine *et al.* algorithm [10], in which this comparison is not performed. Indeed, this algorithm calculates, in each step, only the immediate predecessor list of each formal concept. Hence,an important factor in the performance of the algorithm is the average length of the transaction. In fact, the smaller the number of items per transaction is, the faster the GenAll algorithm, for determining formal concepts and their underlying order.
Case of dense datasets: The number of reported formal concepts is by far more important than that obtained for sparse datasets. This will slacken the corresponding execution time compared to that of sparse datasets. From the point of view of execution time, we noticed that the list of immediate successors does not change frequently as it is the case for sparse datasets.

4.3 Relative performances of Nourine and GenAll algorithms

In what follows, we will put the focus on comparing performances pointed out by GenAll algorithm and those obtained by Nourine *et al.* algorithm [10], in

[4] *http://fimi.cs.helsinki.fi/data*

the context of formal concept discovery and their underlying partial order. The behavior of both algorithms is somehow different for different levels of density.

According to Figure 5, we note that the GenAll algorithm largely outperforms Nourine *et al.* algorithm, especially on dense datasets. Indeed, the time ratio is very large (the average is 40 and sometimes reaches 83). However, ac-

Fig. 4. Execution time of Nourine *et al.* and GenAll algorithms (case of sparse dataset)

cording to Figure 4, Nourine *et al.* algorithm and GenAll show similar performances on sparse datasets. The behavior of the sparse dataset *Kosarak*, compared to the remaining sparse datasets is noteworthy. In fact, the execution time of the GenAll algorithm [16] on the *Kosarak* base is slightly lower than that of Nourine *et al.* [10].

Fig. 5. Execution time of Nourine *et al.* and GenAll algorithms (case of dense data)

5 Conclusion

We proposed a new algorithm for the construction **at the same time** the set of formal concepts, its underlying order and the determination of the set of minimal generators associated to each formal concept. GenAll algorithm [16] performs only one scan on the datasets. We conduced a comparison of performances of two algorithms : GenAll and Nourine *et al.* algorithms. The results shows that GenAll outperforms largely Nourine *et al.* algorithm on dense datasets.

By using the algorithm output, the derivation of the generic bases for the rules of association can be performed in a straightforward manner. In the near future, we plan to tackle two issues. Firstly, we try to improve Genall performances by using advanced data structures to reduce the retrieval operations cost. Secondly, we propose to examine the potential benefits of a parallel version of GenAll algorithm on a MIMD machine (IBM SP2 with 32 processors).

References

1. Agrawal, R., Skirant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile. (1994) 478–499
2. Zaki, M.: Mining Non-Redundant Association Rules. Data Mining and Knowledge Discovery (2004) 223–248
3. BenYahia, S., Nguifo, E.M.: Approches d'extraction de règles d'association basées sur la correspondance de Galois. Ingénierie des Systèmes d'Information (ISI), Hermès-Lavoisier **3–4** (2004) 23–55
4. Zaki, M.J., Hsiao, C.J.: CHARM: An efficient algorithm for closed itemset mining. In: Proceedings of the 2nd SIAM International Conference on Data Mining, Arlington. (2002) 34–43
5. Pei, J., Han, J., Mao, R., Nishio, S., Tang, S., Yang, D.: Closet: An efficient algorithm for mining frequent closed itemsets. In: Proceedings of the ACM SIGMOD DMKD'00, Dallas,TX. (2002) 21–30
6. Pasquier, N.: Data Mining : algorithmes d'extraction et de réduction des règles d'association dans les bases de données. Doctorat d'université, Université de Clermont-Ferrand II, France (2000)
7. Valtchev, P., Missaoui, R., Lebrun, P.: A fast algorithm for building the hasse diagram of a galois lattice. In: Proceedings of the Colloque LaCIM 2000,Montréal (CA). (2000) 293–306
8. Kuznetsov, S., Obedkov, S.: Comparing performance of algorithms for generating concept lattices. JETAI **14** (2002) 189–216
9. Floc'h, A.L., Fisette, C., Missaoui, R., Valtchev, P., Godin, R.: JEN : un algorithme efficace de construction de générateurs pour l'identification des règles d'association. In: Numéro spécial de la revue des Nouvelles Technologies de l'Information, Vol. 1 No. 1, Editions Cépaduès. (2003) 135–146
10. Nourine, L., Raynaud, O.: A fast algorithm for building lattices. Information Processing Letters (1999) 199–214
11. Fu, H., Nguifo, E.M.: Etude et conception d'algorithmes de génération de concepts formels. Revue Ingénierie des Systèmes d'Information **9** (2004) 109–132
12. Barbut, M., Monjardet, B.: Ordre et classification. Algèbre et Combinatoire. Hachette, Tome II (1970)

13. Pfaltz, J.L., Taylor, C.M.: Scientific discovery through iterative transformation of concept lattices. In: Proceedings of Workshop on Discrete Applied Mathematics in conjunction with the 2nd SIAM International Conference on Data Mining, Arlington. (2002) 65–74
14. Bastide, Y., Pasquier, N., Taouil, R., Lakhal, L., Stumme, G.: Mining minimal non-redundant association rules using frequent closed itemsets. In: Proceedings of the International Conference DOOD'2000, Lecture Notes in Computer Sciences, Springer-Verlag. (2000) 972–986
15. Guigues, J., Duquenne, V.: Familles minimales d'implications informatives résultant d'un tableau de données binaires. Mathématiques et Sciences Humaines (1986) 5–18
16. BenTekaya, S., BenYahia, S., Slimani, Y.: Algorithme de construction d'un treillis des concepts formels et de détermination des générateurs minimaux. In: Proceedings of the 7th African Conference on Research in Computer Science. (2004, Tunis) 247–254