Pergamon

Information Systems Vol. 24, No. 1, pp. 25–46, 1999

0306-4379/99 $19.00 + 0.00

EFFICIENT MINING OF ASSOCIATION RULES USING CLOSED ITEMSET LATTICES†

Nicolas Pasquier, Yves Bastide, Rafik Taouil and Lotfi Lakhal

Laboratoire d'Informatique (LIMOS)
Université Blaise Pascal - Clermont-Ferrand II
Complexe Scientifique des Cézeaux
24, av. des Landais, 63177 Aubière Cedex France
{pasquier,bastide,taouil,lakhal}@libd1.univ-bpclermont.fr

(Received 13 May 1998; in final revised form 16 October 1998)

Abstract — Discovering association rules is one of the most important task in data mining. Many efficient algorithms have been proposed in the literature. The most noticeable are Apriori, Mannila's algorithm, Partition, Sampling and DIC, that are all based on the Apriori mining method: pruning the subset lattice (itemset lattice). In this paper we propose an efficient algorithm, called Close, based on a new mining method: pruning the closed set lattice (closed itemset lattice). This lattice, which is a sub-order of the subset lattice, is closely related to Wille's concept lattice in formal concept analysis. Experiments comparing Close to an optimized version of Apriori showed that Close is very efficient for mining dense and/or correlated data such as census style data, and performs reasonably well for market basket style data.

Key words: data mining, knowledge discovery, association rules, data clustering, lattices, algorithms.

1. INTRODUCTION

One of the most important task in data mining is the discovery of *association rules* first introduced in [1]. The aim of association rule discovery is to identify relationships between items in very large databases. For example, given a market basket database, it would be interesting for decision support to know the fact that 80% of customers who bought cereals and sugar also bought milk. In a census database, we should discover that 60% of persons who worked last year earned less than the average income, or in a medical database, that 70% of patients who are stiff and have fever also have headaches.

Agrawal's statement of the problem of discovering association rules in market basket databases is the following [1, 2]. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of m literals called *items*. Let the database $\mathcal{D} = \{t_1, t_2, \ldots, t_n\}$ be a set of n transactions, each one consisting of a set of items I from $\mathcal{I}$ and associated with a unique identifier called its TID. I is called a *k-itemset*, where k is the size of I. A transaction $t \in \mathcal{D}$ is said to *contain* an itemset I if $I \subseteq t$. The *support* of an itemset I is the percentage of transactions in $\mathcal{D}$ containing I: $support(I) = \|\{t \in \mathcal{D} \mid I \subseteq t\}\| \ / \ \|\{t \in \mathcal{D}\}\|$. An association rule is a conditional implication among itemsets, $I_1 \Rightarrow I_2$, where itemsets $I_1, I_2 \subset \mathcal{I}$ and $I_1 \cap I_2 = \emptyset$. The *confidence* of an association rule $r : I_1 \Rightarrow I_2$ is the conditional probability that a transaction contains I_2, given that it contains I_1: $confidence(r) = support(I_1 \cup I_2) \ / \ support(I_1)$. The support of an association rule is defined as: $support(r) = support(I_1 \cup I_2)$.

The problem of mining association rules in a database $\mathcal{D}$ is then traditionally defined as follows. Given user defined thresholds for the permissible minimum support and confidence, find all association rules that hold with more than the given *minsupport* and *minconfidence*. This problem can be broken into two sub-problems [1]:

1. Find all *frequent* itemsets in $\mathcal{D}$, i.e. itemsets with support greater or equal to *minsupport*. Frequent itemsets are also called *large* itemsets.

2. For each frequent itemset I_1 found, generate all association rules $I_2 \Rightarrow I_1 - I_2 \mid I_2 \subset I_1$, with confidence greater or equal to *minconfidence*.

†Recommended by Felipe Carino Jr.

The second sub-problem can be solved in main memory in a straightforward manner once all frequent itemsets and their support are known. Hence, the problem of mining association rules is reduced to the problem of finding frequent itemsets. Many algorithms have been proposed in the literature [2, 5, 12, 13, 15, 16, 18]. Although they are very dissimilar, they are all based on the Apriori mining method [2]: pruning the subset lattice in order to find frequent itemsets. This relies on the basic properties that *all subsets of a frequent itemset are frequent* and that *all supersets of an infrequent itemset are infrequent*. Algorithms based on this approach perform very well for weakly correlated data such as market basket data. However, performances drastically decrease for correlated data such as census data.

In this paper, we propose a new efficient algorithm called Close for mining association rules in very large databases. Close is based on the pruning of the closed itemset lattice which is a sub-order of the subset lattice, thus often much smaller. Such a structure is closely related to Wille's concept lattice in formal concept analysis [3, 7, 19, 20]. We show that this structure can be used as a formal framework for discovering association rules given the basic properties that *all sub-closed itemsets of a frequent closed itemset are frequent*, that *all sup-closed itemsets of an infrequent closed itemset are infrequent* and that *the set of maximal frequent itemsets is identical to the set of maximal frequent closed itemsets*. Empirical evaluations comparing Close to an optimized version of Apriori showed that Close performs reasonably well for weakly correlated data and performs very well for correlated data.

The rest of the paper is organized as follows. Section 2 reviews related work and exhibits the contribution of the paper. In Section 3, we define semantics of association rules based on the *Galois connection operators*. In Section 4, we describe the Close algorithm. Section 5 gives experimental results on synthetic data[†] and census data using the PUMS file for Kansas USA[‡] and Section 6 concludes the paper.

2. RELATED WORK AND CONTRIBUTION

In this section, we first present the subset lattice based approach for mining association rules. Then, we introduce the use of the closed itemset lattice as a formal framework in data mining and we briefly describe the Close mining method.

2.1. A Common Approach for Mining Association Rules

Finding all frequent itemsets is a nontrivial problem since the number of possible frequent itemsets is exponential in the size of the set of items $\mathcal{I}$ of the database. Given $\|\mathcal{I}\| = m$, there are possibly 2^m frequent itemsets, which form a *lattice of subsets* over $\mathcal{I}$ of height m. Consider the example transaction database $\mathcal{D}$ given in Figure 1. The lattice of subsets associated with $\mathcal{D}$ is represented in Figure 2. This lattice contains 32 itemsets and its height is 5. However, depending on the data and the *minsupport* value, only a small fraction of the whole lattice space is frequent. For instance, assuming that *minsupport* is 2 (40%), only 15 itemsets of $\mathcal{D}$ are frequent. A naive approach consists in testing the support of every itemset in the lattice, which can be done in a single pass over the database. Clearly, this approach is impractical for large values of m. In the following, we describe the Apriori mining method used by all existing algorithms for finding frequent itemsets. The notation is given in Table 1.

Apriori Algorithm

In the Apriori algorithm, items are sorted in lexicographic order. The pseudo-code of the Apriori frequent itemset discovery is given in Algorithm 1. Frequent itemsets are computed iteratively, in the ascending order of their size. The process takes k iterations, where k is the size of the largest frequent itemsets. For each iteration $i \leq k$, the database is scanned once and all frequent itemsets of size i are computed. The first iteration computes the set L_1 of frequent 1-itemsets. A

[†] `http://www.almaden.ibm.com/cs/quest/syndata.html`

[‡] `ftp://ftp2.cc.ukans.edu/pub/ippr/census/pums/pums90ks.zip`

TID	Items			
1	A	C	D	
2	B	C	E	
3	A	B	C	E
4	B	E		
5	A	B	C	E

Fig. 1: The transaction database $\mathcal{D}$

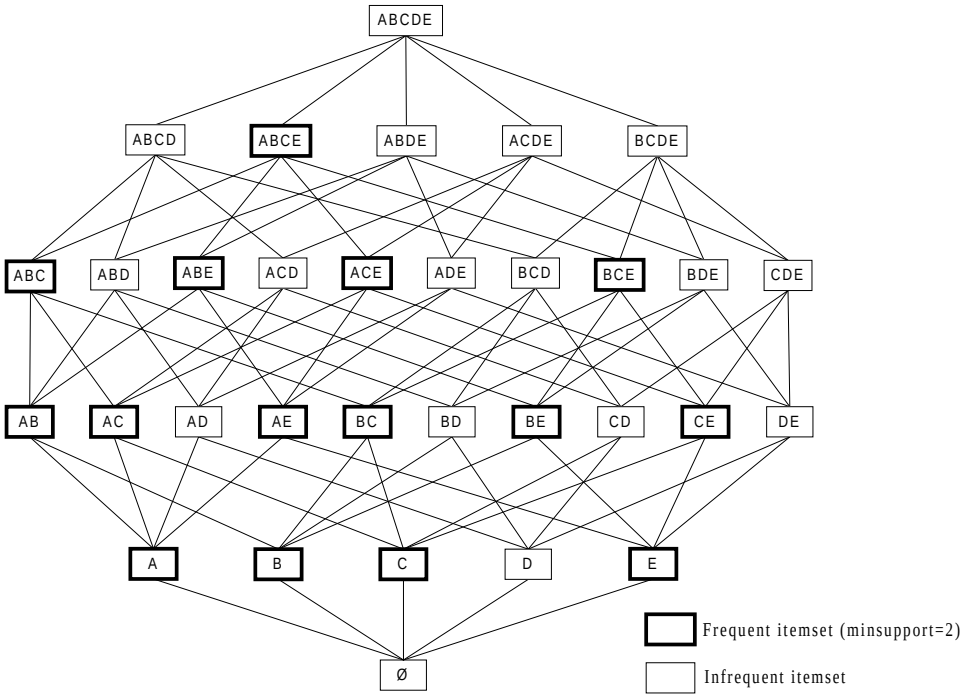


Fig. 2: Itemset lattice of $\mathcal{D}$

subsequent iteration i consists of two phases. First, a set C_i of candidate i-itemsets is created by joining the frequent $(i-1)$-itemsets in L_{i-1} found in the previous iteration. This phase is realized by the Apriori-Gen function described below. Then, the database is scanned for determining the support of the candidates in C_i and the frequent i-itemsets are extracted from the candidates. This process is repeated until no more candidate can be generated.

Algorithm 1 (Apriori frequent itemset discovery)

```
1)  L_1 ← {Frequent 1-itemsets};
2)  for (k ← 2; L_{k-1} ≠ ∅; k++) do begin
3)       C_k ← Apriori-Gen(L_{k-1});          // Generates candidate k-itemsets
4)       forall transactions t ∈ D do begin
5)            C_t ← Subset(C_k,t);            // Candidates contained in t
6)            forall candidates c ∈ C_t do
7)                 c.count++;
8)       end
9)       L_k ← {c ∈ C_k | c.count ≥ minsupport};
10) end
11) Answer ← ⋃_k L_k;
```

C_k	Set of candidate k-itemsets (potentially frequent itemsets). Each element of this set has two fields: i) itemset and ii) support count.
L_k	Set of frequent k-itemsets (itemsets with minimum support). Each element of this set has two fields: i) itemset and ii) support count.

Table 1: Notation

Apriori-Gen Function

The function takes the set L_{i-1} of frequent $(i-1)$-itemsets as argument. It returns the set C_i of candidate i-itemsets, which is a superset of the set of all frequent i-itemsets. The pseudo-code of the function is given in Algorithm 2. Two frequent itemsets of size $i-1$ with the same first $i-2$ items are joined, generating a new candidate itemset of size i (step 1 to 4). Then, the candidate set C_i produced is pruned by removing every candidate i-itemset c such that some $(i-1)$-subset of c is not in L_{i-1} (step 5 to 10).

Algorithm 2 (Apriori-Gen candidate generation)

```
1)  insert into C_i
2)  select p.item_1, p.item_2, ..., p.item_{i-1}, q.item_{i-1}
3)  from L_{i-1} p, L_{i-1} q
4)  where p.item_1 = q.item_1, ..., p.item_{i-2} = q.item_{i-2}, p.item_{i-1} < q.item_{i-1};
5)  forall candidate itemsets c ∈ C_i do begin
6)       forall (i-1)-subsets s of c do begin
7)            if (s ∉ L_{i-1}) then
8)                 delete c from C_i;
9)       end
10) end
11) Answer ← ⋃{c ∈ C_i};
```

Example 1 Figure 3 shows the execution of Apriori for a minimum support of 2 (40%) on the database $\mathcal{D}$. This process takes four iterations, computing four sets of candidates and frequent itemsets and performing four database passes. The frequent itemsets found are outlined in the itemset lattice given in Figure 2.

The algorithms based on this approach take k database passes to generate all frequent itemsets, where k is strongly linked to the height of the itemset lattice (generally the size of the maximal frequent itemsets). Recent algorithms like Partition, Sampling and DIC have attempted to improve the search efficiency by reducing the number of database passes. However, the efficiency of algorithms does not rely only on the I/O cost they incur (number of database passes), but also on the CPU overhead they involve.

2.2. Closed Itemset Lattices

In this section, we define *data mining context*, *Galois connection*, *closed itemset* and *closed itemset lattice*. Interested readers should read [3, 7, 20] for further details on order and lattice theory.

Definition 1 (Data mining context) A data mining context[†] is a triple $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$. $\mathcal{O}$ and $\mathcal{I}$ are finite sets of objects and items respectively. $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{I}$ is a binary relation between objects and items. Each couple $(o, i) \in \mathcal{R}$ denotes the fact that the object $o \in \mathcal{O}$ is related to the item $i \in \mathcal{I}$. A data mining context can be a relation, a class, or the result of an SQL/OQL query.

[†]By extension, well call database a data mining context afterwards.

Scan D $\longrightarrow$

C_1

Itemset	Support
{A}	3
{B}	4
{C}	4
{D}	1
{E}	4

Pruning Infrequent itemsets $\longrightarrow$

L_1

Itemset	Support
{A}	3
{B}	4
{C}	4
{E}	4

Scan D $\longrightarrow$

C_2

Itemset	Support
{AB}	2
{AC}	3
{AE}	2
{BC}	3
{BE}	4
{CE}	3

Pruning Infrequent itemsets $\longrightarrow$

L_2

Itemset	Support
{AB}	2
{AC}	3
{AE}	2
{BC}	3
{BE}	4
{CE}	3

Scan D $\longrightarrow$

C_3

Itemset	Support
{ABC}	2
{ABE}	2
{ACE}	2
{BCE}	3

Pruning Infrequent itemsets $\longrightarrow$

L_3

Itemset	Support
{ABC}	2
{ABE}	2
{ACE}	2
{BCE}	3

Scan D $\longrightarrow$

C_4

Itemset	Support
{ABCE}	2

Pruning Infrequent itemsets $\longrightarrow$

L_4

Itemset	Support
{ABCE}	2

Fig. 3: Discovering frequent itemsets with Apriori for $minsupport = 2$ (40%)

Definition 2 (Galois connection) Let $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$ be a data mining context. For $O \subseteq \mathcal{O}$ and $I \subseteq \mathcal{I}$, we define:

$$f(O) : 2^{\mathcal{O}} \to 2^{\mathcal{I}} \qquad f(O) = \{i \in \mathcal{I} \mid \forall o \in O, (o,i) \in \mathcal{R}\}$$

$$g(I) : 2^{\mathcal{I}} \to 2^{\mathcal{O}} \qquad g(I) = \{o \in \mathcal{O} \mid \forall i \in I, (o,i) \in \mathcal{R}\}$$

$f(O)$ associates with O all items common to all objects $o \in O$ and $g(I)$ associates with I all objects related to all items $i \in I$. The couple of applications (f, g) is a Galois connection between the power set of $\mathcal{O}$ (i.e. $2^{\mathcal{O}}$) and the power set of $\mathcal{I}$ (i.e. $2^{\mathcal{I}}$). The operators $h = f \circ g$ in $2^{\mathcal{I}}$ and $h' = g \circ f$ in $2^{\mathcal{O}\dagger}$ are *Galois closure operators*. Given the Galois connection (f, g), the following properties hold for all $I, I_1, I_2 \subseteq \mathcal{I}$ and $O, O_1, O_2 \subseteq \mathcal{O}$ [3, 7, 20]:

(1) $I_1 \subseteq I_2 \Longrightarrow g(I_1) \supseteq g(I_2)$ (1') $O_1 \subseteq O_2 \Longrightarrow f(O_1) \supseteq f(O_2)$
(2) $I \subseteq h(I)$ (2') $O \subseteq h'(O)$
(3) $h(h(I)) = h(I)$ (3') $h'(h'(O)) = h'(O)$
(4) $I_1 \subseteq I_2 \Longrightarrow h(I_1) \subseteq h(I_2)$ (4') $O_1 \subseteq O_2 \Longrightarrow h'(O_1) \subseteq h'(O_2)$
(5) $h'(g(I)) = g(I)$ (5') $h(f(O)) = f(O)$
(6) $O \subseteq g(I) \Longleftrightarrow I \subseteq f(O)$

Definition 3 (Closed itemsets) Let $C \subseteq \mathcal{I}$ be a set of items from $\mathcal{D}$. C is a closed itemset iff $h(C) = C$. The smallest (minimal) closed itemset containing an itemset I is obtained by applying h to I.

Definition 4 (Closed itemset lattice) Let $\mathcal{C}$ be the set of closed itemsets derived from $\mathcal{D}$ using the Galois connection. The pair $\mathcal{L}_{\mathcal{C}} = (\mathcal{C}, \leq)$ is a complete lattice called closed itemset lattice. The lattice structure implies two properties:

†Here, we use the following notation: $f \circ g(I) = f(g(I))$ and $g \circ f(O) = g(f(O))$.

i) There exists a partial order on the lattice elements such that, for every elements $C_1, C_2 \in \mathcal{L}_\mathcal{C}$, $C_1 \leq C_2$, iff $C_1 \subseteq C_2$[‡].

ii) All subsets of $\mathcal{L}_\mathcal{C}$ have one greatest lower bound, the *Join* element, and one lowest upper bound, the *Meet* element.

Below, we give the definitions of the *Join* and *Meet* elements extracted from Wille's basic theorem on concept lattices [3, 7, 19]. For all $S \subseteq \mathcal{L}_\mathcal{C}$:

$$Join\ (S) = h(\bigcup_{C \in S} C), \qquad Meet\ (S) = \bigcap_{C \in S} C$$

2.3. Close Mining Method

The Close algorithm is fundamentally different from existing algorithms, since it is based on the pruning of the closed itemset lattice in order to find frequent itemsets. A closed itemset is a maximal set of items common to a set of objects. For example, in the database $\mathcal{D}$, the itemset $\{B, C, E\}$ is a closed itemset since it is the maximal set of items common to the objects $\{2, 3, 5\}$. $\{B, C, E\}$ is called a frequent closed itemset for $minsupport = 2$ as support$(\{B, C, E\}) = \|\{2, 3, 5\}\| = 3 \geq minsupport$. In a basket database, this means that 60% of customers (3 customers on a total of 5) purchase **at most** the items B, C, E. The itemset $\{B, C\}$ is not a closed itemset since it is not a maximal group of items common to some objects: all customers purchasing the items B and C also purchase the item E. The closed itemset lattice of a finite relation (the database) is dually isomorphic to the concept lattice [19, 20], also called Galois lattice [11]. Figure 4 gives the closed itemset lattice of $\mathcal{D}$ with frequent closed itemsets for $minsupport = 2$ outlined.

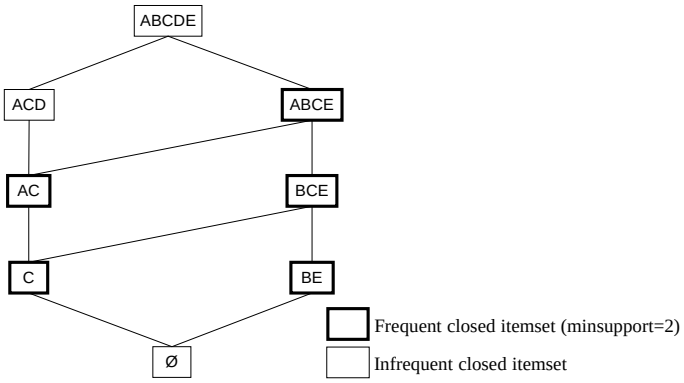


Fig. 4: Closed itemset lattice of $\mathcal{D}$

Using the closed itemset lattice, which is a sub-order of the subset lattice, to find frequent itemsets can improve the efficiency of the association rule discovery. Indeed, the proportion of itemsets that are both closed and frequent is often much smaller than the proportion of frequent itemsets. By minimizing the search space, we reduce both the number of database passes and the CPU overhead incurred by the generation of frequent itemsets. Indeed, the size of the itemset lattice is exponential in the size of the set of items, $\|\mathcal{L}_\mathcal{S}\| = 2^{\|\mathcal{I}\|}$. In the worst case, the closed itemset lattice may grow exponentially. However, the growth is linear with respect to $\|\mathcal{D}\|$ if an upper bound K on the object size ($\|f\{o\}\|$) exists. In such a case, the size of the closed itemset lattice is $\|\mathcal{L}_\mathcal{C}\| \leq 2^K \|\mathcal{D}\|$. Moreover, experimental applications and theoretical results showed that the average growth factor is far less than the 2^K bound. Based on a uniform distribution hypothesis, we can observe that $\|\mathcal{L}_\mathcal{C}\| \leq \mu \|\mathcal{D}\|$, where μ is the mean value for $\|f\{o\}\|$ [11].

[‡] C_1 is a sub-closed itemset of C_2 and C_2 is a sup-closed itemset of C_1.

The closed itemset lattice framework leads us to the following properties (see Section 3):

i) All subsets of a frequent itemset are frequent.

ii) All supersets of an infrequent itemset are infrequent.

iii) All sub-closed itemsets[†] of a frequent closed itemset are frequent.

iv) All sup-closed itemsets[‡] of an infrequent closed itemset are infrequent.

v) The set of maximal frequent itemsets is identical to the set of maximal frequent closed itemsets.

vi) The support of a frequent itemset I which is not closed is equal to the support of the smallest frequent closed itemset containing I (thus, the closure of a frequent itemset is frequent).

Based on these properties, Close generates all association rules from a database $\mathcal{D}$ through three successive phases:

1. Discover all frequent closed itemsets in $\mathcal{D}$, i.e. itemsets that are closed and have support greater or equal to *minsupport*.
2. Derive all frequent itemsets from the frequent closed itemsets found in phase 1. This phase consists in generating all subsets of the maximal frequent closed itemsets and deriving their support from the supports of frequent closed itemsets.
3. For each frequent itemset I found in phase 2, generate all association rules that can be derived from I and have confidence greater or equal to *minconfidence*.

The first phase is the most computationally intensive part of the algorithm. After this phase, no more database access is necessary and the second and third phases can be solved in main memory in a straightforward manner. Indeed, the first phase has given us all the information needed for the next ones, particularly the support of the frequent closed itemsets used to determinate the support of the frequent itemsets without any more database access.

3. ASSOCIATION RULE SEMANTICS

In this section, we propose new semantics for association rules using the Galois connection (f, g). We first define *frequent itemsets*, *frequent closed itemsets* and their properties, in a data mining context $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$. We then define *association rules* and *valid association rules* using the frequent closed itemsets.

3.1. Frequent Itemsets

Definition 5 (Itemset support) Let $I \subseteq \mathcal{I}$ be a set of items from $\mathcal{D}$. The support count of the itemset I in $\mathcal{D}$ is:

$$support(I) = \frac{\|g(I)\|}{\|\mathcal{O}\|}$$

Definition 6 (Frequent itemsets) The itemset I is said to be frequent if the support of I in $\mathcal{D}$ is at least *minsupport*. The set L of frequent itemsets in $\mathcal{D}$ is:

$$L = \{I \subseteq \mathcal{I} \mid support(I) \geq minsupport\}$$

Definition 7 (Maximal frequent itemsets) Let L be the set of frequent itemsets. We define the set M of maximal frequent itemsets as:

$$M = \{I \in L \mid \nexists I' \in L,\ I \subset I'\}$$

[†]Closed subsets of a closed itemset.

[‡]Closed supersets of a closed itemset.

Property 1 *All subsets of a frequent itemset are frequent (intuitive in [2]).*

Proof. Let $I, I' \subseteq \mathcal{I}$, $I \in L$ and $I' \subseteq I$. According to Property (1) of the Galois connection: $I' \subseteq I \Longrightarrow g(I') \supseteq g(I) \Longrightarrow \text{support}(I') \geq \text{support}(I) \geq \textit{minsupport}$. So, we get: $I' \in L$. □

Property 2 *All supersets of an infrequent itemset are infrequent (intuitive in [2]).*

Proof. Let $I, I' \subseteq \mathcal{I}$, $I' \notin L$ and $I' \subseteq I$. According to Property (1) of the Galois connection: $I \supseteq I' \Longrightarrow g(I) \subseteq g(I') \Longrightarrow \text{support}(I) \leq \text{support}(I') \leq \textit{minsupport}$. So, we get: $I \notin L$. □

3.2. Frequent Closed Itemsets

Definition 8 (Frequent closed itemsets) The closed itemset C is said to be frequent if the support of C in $\mathcal{D}$ is at least *minsupport*. We define the set FC of all frequent closed itemsets in $\mathcal{D}$ as:

$$FC = \{C \subseteq \mathcal{I} \mid C = h(C) \text{ and } \text{support}(C) \geq \textit{minsupport}\}$$

Definition 9 (Maximal frequent closed itemsets) Let FC be the set of all frequent closed itemsets. We define the set MC of maximal frequent closed itemsets as:

$$MC = \{C \in FC \mid \nexists C' \in FC,\ C \subset C'\}$$

Property 3 *All sub-closed itemsets of a frequent closed itemset are frequent.*

Proof. Derived from Property 1. □

Property 4 *All sup-closed itemsets of an infrequent closed itemset are infrequent.*

Proof. Derived from Property 2. □

Property 5 *The support of an itemset I is equal to the support of the smallest closed itemset containing I: $support(I) = support(h(I))$.*

Proof. Let $I \subseteq \mathcal{I}$ be an itemset. The support of I in $\mathcal{D}$ is: $$support(I) = \frac{\|g(I)\|}{\|\mathcal{O}\|}$$

Now, we consider $h(I)$, the closure of I. Since $g(I)$ is closed and by consequence $h'(g(I)) = g(I)$ (according to Property (5) of the Galois connection), we have:

$$support(h(I)) = \frac{\|g(h(I))\|}{\|\mathcal{O}\|} = \frac{\|h'(g(I))\|}{\|\mathcal{O}\|} = \frac{\|g(I)\|}{\|\mathcal{O}\|} = support(I)$$

□

Property 6 *The set of maximal frequent itemsets M is identical to the set of maximal frequent closed itemsets MC.*

Proof. It suffices to demonstrate that $\forall I \in M$, I is closed, i.e. $I = h(I)$. Let $I \in M$ be a maximal frequent itemset. According to Property (2) of the Galois connection $I \subseteq h(I)$ and, since I is maximal and $support(h(I)) = support(I) \geq minsupport$, we conclude that $I = h(I)$. I is a maximal frequent closed itemset. Since all maximal frequent itemsets are also maximal frequent closed itemsets, we get: $M = MC$. □

3.3. Association Rules

Definition 10 (Association rules) An association rule is an implication between itemsets of the form $I_1 \Rightarrow I_2$ where $I_1, I_2 \subset \mathcal{I}$ and $I_1 \cap I_2 = \emptyset$. The itemset I_1 is called the *antecedent* of the rule, and the itemset I_2 is called the *consequent*. Below, we define the support and confidence of an association rule using the Galois connection applications f and g. The support and confidence of an association rule $r : I_1 \Rightarrow I_2$ are:

$$support(r) = \frac{\|g(I_1 \cup I_2)\|}{\|\mathcal{O}\|}, \qquad confidence(r) = \frac{support(I_1 \cup I_2)}{support(I_1)} = \frac{\|g(I_1 \cup I_2)\|}{\|g(I_1)\|}$$

Definition 11 (Valid association rules) The task of mining association rules consists in generating all valid association rules, i.e. association rules with support and confidence greater or equal to the *minsupport* and *minconfidence* thresholds. Let $\mathcal{AR}$ be the set of valid association rules in $\mathcal{D}$. We define $\mathcal{AR}$ using the set MC of maximal frequent closed itemsets as:

$$\mathcal{AR}(\mathcal{D}, minsupport,\ minconfidence) = \{r : I_2 \Rightarrow (I_1 - I_2) \quad | \quad I_2 \subset I_1,\ I_1 \in L = \bigcup_{C \in MC} 2^C \text{ and } confidence(r) \geq minconfidence\}$$

4. CLOSE ALGORITHM

In Section 4.1 we describe our search of the frequent closed itemsets. In Section 4.2 we give our method for deriving all the frequent itemsets from the frequent closed itemsets. In Section 4.3 we present an efficient algorithm for finding valid association rules using frequent itemsets. This algorithm is adapted from the one described in [2].

4.1. Discovering Frequent Closed Itemsets

As in the Apriori algorithm, items are sorted in lexicographic order. The pseudo-code for discovering frequent closed itemsets is given in Algorithm 3. The notation is given in Table 2. In each iteration, the algorithm constructs a set of candidate frequent closed itemsets. It then determines the frequent closed itemsets using the *minsupport* threshold. Finally, it computes the generator[†] that will be used in the next iteration to construct the set of candidate frequent closed itemsets. In each iteration, one pass over the database is necessary, in order to construct the set of candidate frequent closed itemsets (closures of generators) and count their support.

Set	Field	Contains
FCC_i	*generator*	A generator of size i.
	closure	Candidate closed itemset produced by the closure of *generator*: $closure = h(generator)$.
	support	Support count of the closed itemset: $support = count(closure) = count(generator)$ (according to Property 5).
FC_i	*generator*	Generator of the frequent closed itemset.
	closure	Frequent closed itemset (closed itemset with support greater or equal to *minsupport*).
	support	Support count of the frequent closed itemset.

Table 2: Notation

[†] A generator p of a closed itemset c is one of the smallest itemsets (there can be more than one) that will give c by application of the Galois closure operator: $h(p) = c$.

Algorithm 3 (Close algorithm)

```
1)  generators in FCC_1 ← {1-itemsets};
2)  for (i ← 1; FCC_i.generator≠ ∅; i++) do begin
3)       closures in FCC_i ← ∅;
4)       supports in FCC_i ← 0;
5)       FCC_i ← Gen-Closure(FCC_i)†;          // Produces generator closures
6)       forall candidate closed itemsets c ∈ FCC_i do begin
7)            if (c.support ≥ minsupport) then          // If c is frequent
8)                 FC_i ← FC_i ∪ {c};                   // Append c to FC_i
9)       end
10)      FCC_{i+1} ← Gen-Generator(FC_i);    // Creates generators of iteration i+1
11) end
12) Answer FC ← ⋃_{j=1}^{j=i-1} {FC_j.closure, FC_j.support};
```

The first operation of the algorithm (step 1) initializes the set of generators in FCC_1 with items present in the data mining context, i.e. elements of the set $\mathcal{I}$. No database pass is needed for this step. Each of the following iterations consists of three phases. First, the closure function is applied to each generator in FCC_i, determining the candidate closed itemsets and their support (step 5). The closure function *Gen-Closure* used for this purpose is described in Section 4.1.1. Next, the set of candidate closed itemsets obtained is pruned: closed itemsets with sufficient support value are inserted in the set of frequent closed itemsets FC_i (step 6 to 9). Finally, generators in the set FCC_{i+1} are determined by applying the function *Gen-Generator* (described in Section 4.1.2) to the generators of frequent closed itemsets in FC_i. This process takes place until FCC_{i+1} is empty. Then, all frequent closed itemsets have been produced and their support is known (see Theorem 3).

4.1.1. Gen-Closure Function

The closure function *Gen-Closure* takes as argument the set of generators in FCC_i. It updates FCC_i with, for each generator p, the closed itemset p.closure obtained by applying the closure operator h to p and their support count p.support. Algorithm 4 gives the pseudo-code of the function. The method used to compute closed itemsets is based on Proposition 1.

Proposition 1 *The closed itemset $h(I)$ corresponding to the closure by h of the itemset I is the intersection of all objects in the database that contain I:*

$$h(I) = \bigcap_{o \in O} \{f(\{o\}) \mid I \subseteq f(\{o\})\}$$

Proof. We define $H = \bigcap_{o \in S} f(\{o\})$ where $S = \{o \in O \mid I \subseteq f(\{o\})\}$. We have $h(I) = f(g(I)) = \bigcap_{o \in g(I)} f(\{o\}) = \bigcap_{o \in S'} f(\{o\})$ where $S' = \{o \in O \mid o \in g(I)\}$. Let's show that $S' = S$:

$$\begin{aligned} I &\subseteq f(\{o\}) \iff o \in g(I) \\ o &\in g(I) \iff I \subseteq f(g(I)) \subseteq f(\{o\}) \end{aligned}$$

We conclude that $S = S'$, thus $h(I) = H$. □

Using Proposition 1, only one database pass is necessary to compute the closures of the generators in an iteration i, and their support. The function works as follows. For each object o in $\mathcal{D}$, the set G_o is created (step 2). G_o contains all generators in FCC_i that are subsets of the object itemset $f(\{o\})$ (see Section 4.1.3). Then, for each generator p in G_o, the associated closed itemset p.closure and their support count p.support are updated (step 3 to 7). If the object o is the first one containing the generator, p.closure is empty and the object itemset $f(\{o\})$ is assigned to it

† An optimization (implemented version) consists in keeping in FC_1 only generators with dissimilar closures.

(step 4 and 5). Otherwise, the intersection between p.closure and the object itemset gives the new p.closure (step 6). Then, the closed itemset support p.support is incremented (step 7). At the end, the function returns for each generator p in FCC_i, the closed itemset p.closure and its associated support count p.support. The closed itemset p.closure corresponds to the intersection of all objects containing[‡] p. The support count p.support corresponds to the number of objects containing p.closure (supports counts of the generator and its closure are equals according to Property 5).

Algorithm 4 (Gen-Closure function)

```
1)  forall objects o ∈ O do begin
2)       G_o ← Subset(FCC_i.generator, f({o})); //Gen. that are subsets of f({o})
3)       forall generators p ∈ G_o do begin
4)            if (p.closure = ∅) then
5)                 p.closure ← f({o});
6)            else p.closure ← p.closure ∩ f({o});
7)            p.support++;
8)       end
9)  end
10) Answer ← ⋃ {c ∈ FCC_i | c.closure ≠ ∅};
```

4.1.2. Gen-Generator Function

The *Gen-Generator* function takes the set of frequent closed itemsets FC_i as argument. Based on Lemma 1, it returns the set FCC_{i+1} containing all $(i+1)$-generators[†] that will be used during iteration $i+1$ to construct the set of candidate frequent closed itemsets. The function first generates all potential $(i+1)$-generators using i-generators in FC_i. Then, based on Corollary 2, the potential generators produced that will lead to useless computations (infrequent closed itemsets) or redundancies (frequent closed itemsets already produced) are deleted from FCC_{i+1}. The pseudo-code of the function is given in Algorithm 5.

Lemma 1 *Let I_1, I_2 be two itemsets. We have:*

$$h(I_1 \cup I_2) = h(h(I_1) \cup h(I_2))$$

Proof. Let I_1 and I_2 be two itemsets. According to Property (2) of the Galois connection:

$$\begin{aligned} I_1 \subseteq h(I_1) \text{ and } I_2 \subseteq h(I_2) &\implies I_1 \cup I_2 \subseteq h(I_1) \cup h(I_2) \\ &\implies h(I_1 \cup I_2) \subseteq h(h(I_1) \cup h(I_2)) \end{aligned} \quad (1)$$

Obviously, $I_1 \subseteq I_1 \cup I_2$ and $I_2 \subseteq I_1 \cup I_2$. So $h(I_1) \subseteq h(I_1 \cup I_2)$ and $h(I_2) \subseteq h(I_1 \cup I_2)$. According to Property (3) of the Galois connection:

$$h(h(I_1) \cup h(I_2)) \subseteq h(h(I_1 \cup I_2)) \implies h(h(I_1) \cup h(I_2)) \subseteq h(I_1 \cup I_2) \quad (2)$$

From (1) and (2), we conclude that $h(I_1 \cup I_2) = h(h(I_1) \cup h(I_2))$. □

Lemma 2 *Let I_1 be an itemset and I_2 a subset of I_1 where $I_1 \subseteq h(I_2)$. Then we have $h(I_1) = h(I_2)$ and $\forall I_3 \subseteq \mathcal{I}$, $h(I_1 \cup I_3) = h(I_2 \cup I_3)$.*

Proof. Let I_1, I_2 be two itemsets and $I_2 \subset I_1 \subseteq h(I_2)$. Then according to the properties of monotonicity and idempotency of the Galois closure operator h (Properties (3) and (4)) we deduce

[‡] We say that an object o contains an itemset I if o is related to all items $i \in I$.

[†] We call i-generator a generator of size i.

that $h(I_2) \subseteq h(I_1) \subseteq h(h(I_2)) = h(I_2)$ and we conclude that $h(I_1) = h(I_2)$. Let $I_3 \subseteq \mathcal{I}$ be an itemset. Then according to Lemma 1:

$$h(I_1 \cup I_3) = h(h(I_1) \cup h(I_3)) = h(h(I_2) \cup h(I_3)) = h(I_2 \cup I_3)$$

□

Corollary 2 *Let I be an i-generator and $\mathcal{S} = \{s_1, s_2, \ldots, s_j\}$ a set of $(i-1)$-subsets of I where $\bigcup_{s \in \mathcal{S}} s = I$. If $\exists s_a \in \mathcal{S}$ such as $I \subseteq h(s_a)$, then $h(I) = h(s_a)$.*

Proof. Derived from Lemma 2. □

The Gen-Generator function works as follows. We first apply the combinatorial phase of Apriori-Gen [2] to the set of generators in FC_i in order to obtain a set of new potential generators: two generators of size i in FC_i with the same first $i-1$ items are joined, producing a new potential generator of size $i+1$ (step 1 to 4). Then, the resulting set is pruned using two strategies.

First, like in Apriori-Gen, for all potential generator p created we check the presence of all its i-subsets in FC_i (step 5 to 10). Using this strategy, we prune two kinds of itemsets: first, all supersets of infrequent itemsets (that are also infrequent according to Property 2); second, all generators that are included in one of their subset closures, which has already been determined (see Theorem 3). Let's take an example. Suppose that the set of frequent closed itemsets FC_2 contains the generators AB, AC. The Gen-Generator function will create $ABC = AB \cup AC$ as a new potential generator in FCC_3 and the first pruning will remove ABC since $BC \notin FC_2$.

The second pruning strategy works as follows. For each potential generator p in FCC_{i+1}, we test if the closure of one of its i-subsets s is a superset of p. In that case, the closure of p will be equal to the closure of s (see Corollary 2), so we remove p from FCC_{i+1} (step 11 to 17). Let's give another example. Suppose that the set of frequent closed itemsets FC_2 contains generators AB, AC, BC and their respective closures AB, ABC, BC and that all of them are frequent. The Gen-Generator function will create $ABC = AB \cup AC$ as a new potential generator in FCC_3. The second prune step will remove ABC from FCC_3 since $ABC \subseteq closure(AC)$. Indeed, we deduce that $closure(ABC) = closure(AC)$ and the computation of the closure of ABC is useless.

Algorithm 5 (Gen-Generator function)

```
1)  insert into FCC_{i+1}.generator
2)  select p.item_1, p.item_2, ..., p.item_i, q.item_i
3)  from FC_i.generator p, FC_i.generator q
4)  where p.item_1 = q.item_1, ..., p.item_{i-1} = q.item_{i-1}, p.item_i < q.item_i;

5)  forall generators p ∈ FCC_{i+1}.generator do begin
6)       forall i-subsets s of p do begin
7)            if (s ∉ FC_i.generator) then
8)                 delete p from FCC_{i+1}.generator;
9)       end
10) end

11) forall generators p ∈ FCC_{i+1}.generator do begin
12)      S_p ← Subset(FC_i.generator,p); // Generators that are subsets of p
13)      forall s ∈ S_p do begin
14)           if (p ⊆ s.closure) then
15)                delete p from FCC_{i+1}.generator;
16)      end
17) end
18) Answer ← ⋃{c ∈ FCC_{i+1}};
```

4.1.3. Subset Function

Candidate frequent closed itemsets are stored in a *prefix-tree* structure to quickly find all generators associated with an object. Our structure is derived from the one proposed in [14]. Figure 5 shows the Prefix-tree structure for the set FCC_2 given in Figure 6. Each edge in the tree is labeled with an item. A generator is represented as a path in the tree, starting from the root node. The closure of a generator is stored in the leaf (terminal node) of the path representing it. Each node contains a pointer to a sibling node, a hash-table towards the children of the node and, if the node is a leaf, a pointer to the closure of the generator represented. For a node representing an i-itemset c, a sibling node represents another i-itemset with the same first $i-1$ items and a hash collision on the i^{th} item. For performance reasons, if the size of such a linked list exceeds a given threshold, instead of adding a new sibling node, the size of the hash-table of the parent node is doubled and the i^{th} nodes are rebalanced.

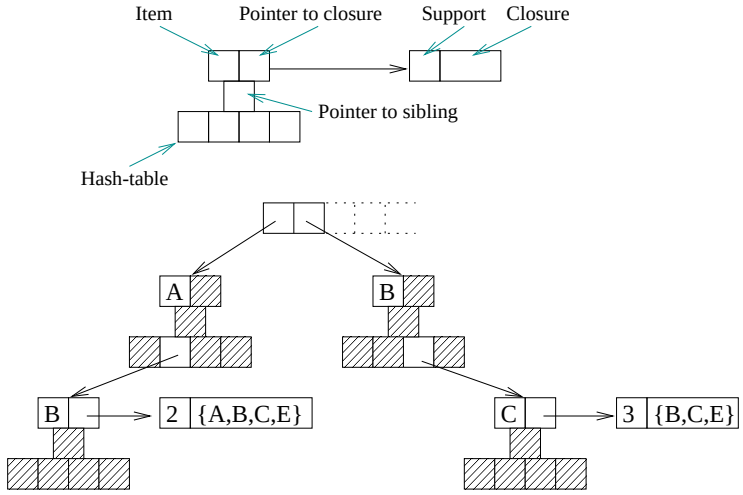


Fig. 5: Prefix-tree of the set FCC_2 in Figure 6

The subset function takes a set of generators G and an itemset c as arguments. It determines which generators $p \in G$ are subsets of the itemset c. The function starts from the root node and hashes successively on each item of the itemset c down through the tree. Having reached a node by hashing on item i, we hash on each item that comes after i in c and recursively apply this procedure to the node in the corresponding bucket of the hash-table. At the root node, we hash on every item in c. When we reach a leaf, we add the reference to the generator to the answer set.

4.1.4. Example and Correctness

Figure 6 shows the execution of the algorithm for a minimum support of 2 (40%) on the data mining context $\mathcal{D}$ given in Figure 1. This execution includes the first iteration optimization previously mentioned: we keep only the first 1-generator (in lexicographic order) from those that have the same closure. Thus, since $h(B) = h(E) = BE$, the generator E is pruned and the generator B is kept in FC_1.

Step 1 initializes the set of generators in FCC_1 with the list of items in $\mathcal{D}$. Calling Gen-Closure at step 5 gives for every generator p the candidate closed itemset p.closure and its support count p.support. In step 6 through 9, the set FC_1 is generated by pruning FCC_1 according to *minsupport*. In step 10, generators in FCC_2 are produced by applying the Gen-Generator function to FC_1. As we can see in Figure 6, calling Gen-Generator with FC_1 produces two new generators: AB and BC. Generators A and C in FC_1 do not give a new generator AC since the closure of A is AC and the closure of C is C. Obviously $AC \subseteq h(A)$, so the generator $A \cup C$ is deleted by the second pruning step of the Gen-Generator function.

Calling Gen-Closure with FCC_2 produces the closures of the generators in FCC_2 and their support. After the pruning of the candidate closed itemsets, FCC_2 and FC_2 are identical since all candidate closed itemsets in FCC_2 are frequent. The set of generators in FCC_3 constructed by calling Gen-Generator with FC_2 is empty as no generator in FC_2 have the same first item, and the algorithm terminates. We can observe that the number of database passes is reduced by half compared to the execution of Apriori on the same example.

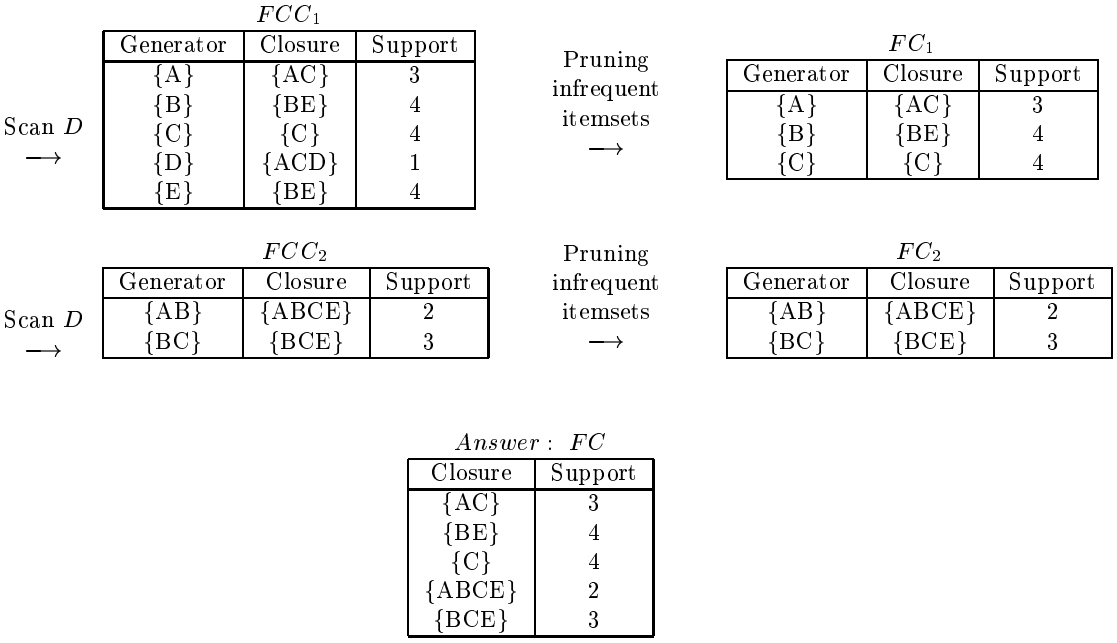


Fig. 6: Discovering frequent closed itemsets with Close for *minsupport* = 2 (40%)

Lemma 3 *For $p \subseteq \mathcal{I}$ such as $\|p\| > 1$, if $p \notin FC_{\|p\|}$ and support(p) $\geq$ minsupport then $\exists s_1, s_2 \subseteq \mathcal{I}$, $s_1 \subset s_2 \subseteq p$ and $\|s_1\| = \|s_2\| - 1$ such as $h(s_1) = h(s_2)$ and $s_1 \in FC_{\|s_1\|}$.*

Proof. We show this using a recurrence. For $\|p\| = 2$, we have $p = s_2$ and $\exists s_1 \in FC_1$ where $s_1 \subset s_2$ and $h(s_1) = h(s_2)$ (Lemma 3 is obvious). Then, supposing that Lemma 3 is true for $\|p\| = i$, let's show that it is true for $\|p\| = i + 1$. Let $p \subseteq \mathcal{I} \mid \|p\| = i + 1$ and $p \notin FC_{\|p\|}$. There are two possible cases:
(1) $\exists p' \subset p$ where $\|p'\| = i$ and $p' \notin FC_{\|p'\|}$
(2) $\exists p' \subset p$ where $\|p'\| = i$ and $p' \in FC_{\|p'\|}$ and $p \subseteq h(p') \Rightarrow h(p) = h(p')$ (Lemma 2)
If (1) then according to the recurrence hypothesis, $\exists s_1 \subset s_2 \subseteq p' \subset p$ such as $h(s_1) = h(s_2)$ and $s_1 \in FC_{\|s_1\|}$. If (2) then we identify s_1 to p' and s_2 to p. □

Theorem 3 *The Close algorithm generates all frequent closed itemsets.*

Proof. Using a recurrence, we show that $\forall p \subseteq \mathcal{I}$ where support$(p) \geq$ minsupport we have $h(p) \in FC$. We first demonstrate the property for the 1-itemsets: $\forall p \subseteq \mathcal{I}$ where $\|p\| = 1$, if support$(p) \geq$ minsupport then $h(p) \in FC_1 \Rightarrow h(p) \in FC$. Let's suppose that $\forall p \subseteq \mathcal{I}$ with $\|p\| = i$ we have $h(p) \in FC$. We then demonstrate that $\forall p \subseteq \mathcal{I}$ such as $\|p\| = i + 1$ we have $h(p) \in FC$. If $p \in FC_{\|p\|}$ then $h(p) \in FC_{\|p\|}$. Else, if $p \notin FC_{\|p\|}$ and according to Lemma 3, we have: $\exists s_1 \subset s_2 \subseteq p$ with $s_1 \in FC_{\|s_1\|}$ and $h(s_1) = h(s_2)$. Now $h(p) = h(s_2 \cup p - s_2) = h(s_1 \cup p - s_2)$ and $\|s_1 \cup p - s_2\| = i$, therefore in conformity with the recurrence hypothesis we conclude that $h(s_1 \cup p - s_2) \in FC_i$ and so $h(p) \in FC$. □

4.2. Deriving Frequent Itemsets

The pseudo-code for deriving frequent itemsets is given in Algorithm 6. It uses as input the set of frequent closed itemsets $FC = \bigcup_i FC_i$ and returns the set of frequent itemsets $L = \bigcup_k L_k$. In step 1 through 5, we put each frequent closed itemset c from FC in the set of frequent itemsets $L_{\|c\|}$ corresponding to the size of c and we determine the size k of the largest frequent itemsets. During step 6 to 15 we construct all sets L_i, starting from L_k down to L_1. In each iteration, the set L_{i-1} is completed using itemsets in L_i. For each i-itemset c in L_i, all $(i-1)$-subsets of c are generated. All subsets that are not present in L_{i-1} are added to the end of L_{i-1} with support value equal to the support of c. This process takes place until L_1 has been completed.

Algorithm 6 (Deriving frequent itemsets)

```
1)  k ← 0;
2)  forall frequent closed itemsets c ∈ FC do begin
3)        L_||c|| ← L_||c|| ∪ {c};                  // Splitting frequent closed itemsets
4)        if (k < ||c||) then k ← ||c||;
5)  end
6)  for (i ← k; i>1; i−−) do begin
7)        forall itemsets c ∈ L_i do begin
8)              forall (i−1)-subsets s of c do begin
9)                    if (s ∉ L_{i−1}) then begin
10)                         s.support ← c.support;
11)                         L_{i−1} ← L_{i−1} ∪ {s};        // Put s at the end of L_{i−1}
12)                   end
13)             end
14)       end
15) end
16) Answer ← ⋃_{i=1}^{i=k} L_i;
```

Example 2 Figure 7 shows the execution of the algorithm using as input the sets FC_1 and FC_2 given in Figure 6. The first phase of the algorithm simply splits frequent closed itemsets of FC_1 and FC_2 in sets L_1 to L_4 according to their size, and determines that $k = 4$. During the first iteration of the loop (step 6 to 15), the closed itemset $ABCE$ in L_4 is examined and generates ABC, ABE and ACE in L_3 with the same support value as $ABCE$. The closed itemset BCE is not generated since it is already present in L_3. During the second iteration, we first examine the closed itemset BCE, generating BC in L_2 with support = 3. If we had first examined the itemset ABC, we would have generated BC in L_2 with support = 2 which is incorrect. At the end of the second iteration, L_2 is complete. The third iteration generates L_1 and ends the algorithm.

Correctness The correctness of the algorithm for deriving frequent itemsets relies on Properties 5 and 6, and on the fact that we first examine closed i-itemsets in L_i during the $(i-1)$-subsets generation. For an iteration i, let c be a frequent closed i-itemset in L_i and s an $(i-1)$-subset of c. If s is a closed itemset then it has already been inserted in L_{i-1} during the first phase of the algorithm (step 1 to 5). If s is not a closed itemset, then according to Property 5 it is correct to insert s in L_{i-1} with the same support value as c. Now, consider that s is not a closed $(i-1)$-itemset and that we are completing L_{i-2}. The support of s is equal to the support of c which is the smallest closed itemset containing s. Let s' be a $(i-2)$-subset of s. If s' is not a closed itemset and has not already been generated in L_{i-2}, then, given Property 5, its support is equal to the support of the smallest closed itemset containing s' which is c. Hence it is correct to insert s' in L_{i-2} with the support value of s. Since the set of maximal frequent itemsets is the same as the set of maximal frequent closed itemsets (see Property 6), the set L_k is complete, where k is the size of the largest frequent itemsets (obviously all maximal frequent itemsets). Given the properties that all subsets of a frequent itemset are frequent and all supersets of an infrequent itemset are infrequent, by generating all subsets of the maximal frequent closed itemsets we generate all frequent itemsets, and the result is correct.

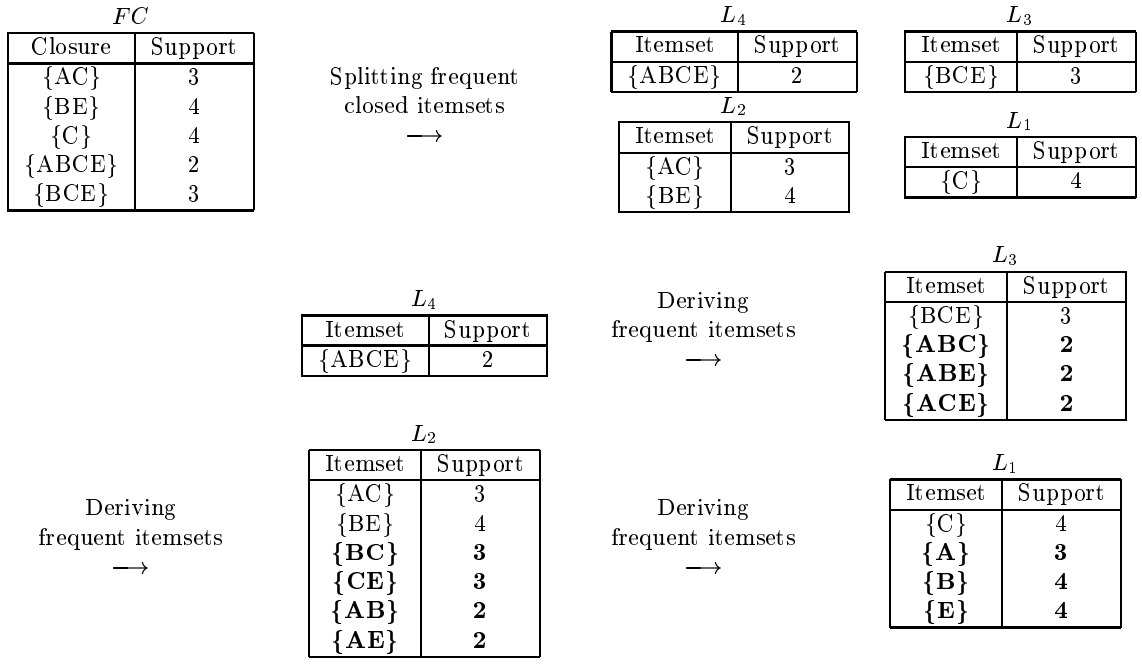


Fig. 7: Deriving frequent itemsets for *minsupport* = 2 (40%)

4.3. Generating Valid Association Rules

The problem of generating valid association rules can be solved in a straightforward manner once all frequent itemsets and their supports are known. In this section, we describe an adapted version of Apriori rule generation algorithm [2] (implemented for our experiments). The general principle is the following: for every frequent itemset I_1, all subsets I_2 of I_1 are derived and the ratio $support(I_1)/support(I_2)$ is computed. If the result is at least *minconfidence*, then the rule $I_2 \Rightarrow (I_1 - I_2)$ is generated.

The support of any subset I_3 of I_2 being greater or equal to the support of I_2, the confidence of the rule $I_3 \Rightarrow (I_1 - I_3)$ is necessarily less than the confidence of the rule $I_2 \Rightarrow (I_1 - I_2)$. Hence, if the rule $I_2 \Rightarrow (I_1 - I_2)$ does not hold, neither will the rule $I_3 \Rightarrow (I_1 - I_3)$. Conversely, if the rule $(I_1 - I_2) \Rightarrow I_2$ holds, then all rules of the form $(I_1 - I_3) \Rightarrow I_3$ also hold. For example, if the rule $A \Rightarrow BC$ holds, then the rules $AB \Rightarrow C$ and $AC \Rightarrow B$ also hold.

Using this property for efficiently generating valid association rules, the algorithm works as follows. For every frequent itemset I_1, all rules with one item in the consequent that have a confidence at least equal to *minconfidence* are generated. We then create all consequents with 2 items that are possible in a rule generated from I_1. This generation is performed by applying the Apriori-Gen function given in Section 2.1 to the set of one item consequents found in the first step. Next the set of rules with 2 items in the consequent generated is pruned with respect to *minconfidence*. The 2 items consequents of the rules that hold are used for generating consequents with 3 items, etc. The pseudo-code is given in Algorithm 7.

Example 3 Figure 8 gives the result of the frequent itemset discovery for the data mining context $\mathcal{D}$, assuming that *minsupport* is 3 (60%). Figure 9 shows the valid association rule generation for *minconfidence* = 0.5 (50%) using previous result.

L_3

Itemset	Support
{BCE}	3

L_2

Itemset	Support
{AC}	3
{BC}	3
{BE}	4
{CE}	3

L_1

Itemset	Support
{A}	3
{B}	4
{C}	4
{E}	4

Fig. 8: Frequent itemsets extracted from $\mathcal{D}$ for *minsupport* = 3 (60%)

Algorithm 7 (Generating valid association rules)

```
1)  forall frequent k-itemsets l_k ∈ L_k where k ≥ 2 do begin
2)       H_1 ← {itemsets of size 1 that are subsets of l_k};
3)       forall h_1 ∈ H_1 do begin
4)            confidence ← support(l_k)/support(l_k − h_1); // Confidence of r
5)            if (confidence ≥ minconfidence) then
6)                 AR ← AR ∪ {r : (l_k − h_1)⇒h_1};
7)            else H_1 ← H_1 − {h_1};
8)       end
9)       call Gen-Rules(l_k, H_1); // H_1 = 1-item consequents of valid rules from l_k
10) end
11) Procedure Gen-Rules(l_k:frequent k-itemset, H_m:set of m-item consequents)
12)      if (k > m + 1) then do begin
13)           H_{m+1} ← Apriori-Gen(H_m);
14)           forall h_{m+1} ∈ H_{m+1} do begin
15)                confidence ← support(l_k)/support(l_k − h_{m+1});
16)                if (confidence ≥ minconfidence) then
17)                     AR ← AR ∪ {r : (l_k − h_{m+1})⇒h_{m+1}};
18)                else delete h_{m+1} from H_{m+1};
19)           end
20)           call Gen-Rules(l_k, H_{m+1});
21)      end
22) end
```

5. EXPERIMENTAL RESULTS

We implemented the Apriori and Close algorithms in C++ on several Unix platforms, to assess their relative performances. Both used the same data structure (as described in Section 4.1.3) that improves Apriori efficiency. Our experiments were realized on a 43P240 bi-processor IBM Power-PC running AIX 4.1.5 with a CPU clock rate of 166 MHz, 1GB of main memory and a 9GB disk. Only one processor was used since the application was single-threaded. The test program was allowed a maximum of 128MB. We did not implement swapping; also, the system buffers were not flushed between each database pass of the algorithms. In Section 5.1, we describe the datasets used for the experiments. In Section 5.2, we compare the relative performances of Apriori and Close, and in Section 5.3, we address the problem of determining which of the two algorithms will be the most efficient according to the dataset.

5.1. Test Data

The algorithms were tested on two types of datasets: synthetic data, which mimic market basket data, and census data, which belong to the domain of statistical databases. For generating the synthetic datasets, we used the program described in [2]. These datasets, called T10I4D100K and

l_k

Itemset	Support
{BCE}	3

Generating rules $\longrightarrow$

1-item consequents

Rule	Confidence
BC $\Rightarrow$ E	1
BE $\Rightarrow$ C	0.75
CE $\Rightarrow$ B	1

Generating rules $\longrightarrow$

2-items consequents

Rule	Confidence
B $\Rightarrow$ CE	0.75
C $\Rightarrow$ BE	0.75
E $\Rightarrow$ BC	0.75

l_k

Itemset	Support
{AC}	3

Generating rules $\longrightarrow$

1-item consequents

Rule	Confidence
A $\Rightarrow$ C	1
C $\Rightarrow$ A	0.75

l_k

Itemset	Support
{BE}	4

Generating rules $\longrightarrow$

1-item consequents

Rule	Confidence
B $\Rightarrow$ E	1
E $\Rightarrow$ B	1

l_k

Itemset	Support
{BC}	3

Generating rules $\longrightarrow$

1-item consequents

Rule	Confidence
B $\Rightarrow$ C	0.75
C $\Rightarrow$ B	0.75

l_k

Itemset	Support
{CE}	3

Generating rules $\longrightarrow$

1-item consequents

Rule	Confidence
C $\Rightarrow$ E	0.75
E $\Rightarrow$ C	0.75

Fig. 9: Generating valid association rules for *minsupport* = 3 (60%) and *minconfidence* = 0.5 (50%)

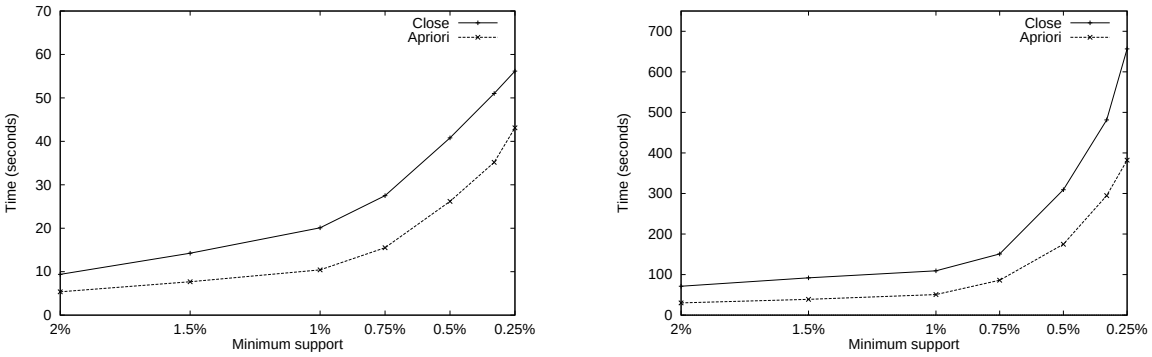
T20I6D100K, contains 100,000 objects for an average object size of 10 and 20 items respectively and an average size of the maximal potentially frequent itemsets of 4 and 6 respectively.

The census datasets were extracted from the Kansas 1990 PUMS file (Public Use Microdata Samples), in the same way as [5] for the PUMS file of Washington (unavailable through Internet at the time of the experiments). Unlike in [5] though, we did not put an upper bound on the support, as this distorts each algorithm's results in different ways. We therefore took smaller datasets containing the first 10,000 persons. Dataset C20D10K contains 20 attributes (20 items per object and 386 total items), and C73D10K, 73 attributes (73 items per object and 2178 total items).

5.2. Relative Performance of Apriori and Close

5.2.1. Synthetic Data

We used the same values for *minsupport* as the ones used in [2], ranging from 2% to 0.25%. Figure 10 shows the execution times of Apriori and Close on the datasets T10I4D100K and T20I6D100K. We can observe that Apriori performs better than Close on these data. The reason is that, in such datasets, data are weakly correlated and sparse; furthermore, nearly all frequent itemsets are closed. For an identical number of database passes, Close performs more operations to compute the closure of the generators. Response times remain however acceptable: less than one minute for the longest execution on T10I4D100K and eleven minutes for the longest execution on T20I6D100K, that is to say about once and a half the execution times of Apriori. We can also observe that both algorithms scale up similarly as the minimum support is decreased.



Execution times on T10I4D100K

Execution times on T20I6D100K

Fig. 10: Performance of Apriori and Close on synthetic data

5.2.2. Census Data

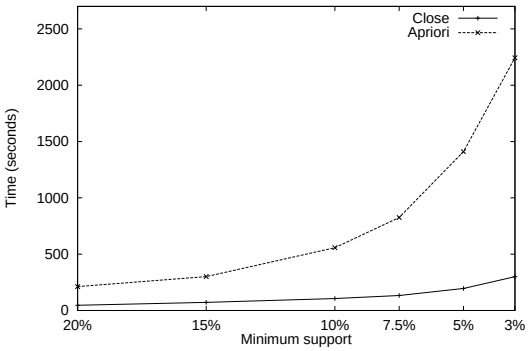
Experiments were conducted on the two census datasets using different *minsupport* ranges to get meaningful response times and to accommodate with the memory space limit. Results for the C20D10K and C73D10K datasets are plotted on Figure 11 and Figure 12 respectively. Close always significantly outperforms Apriori, for execution times as well as number of database passes. Here, contrarily to the experiments on synthetic data, the differences between execution times can be measured in minutes for C20D10K and in hours for C73D10K. It should furthermore be noted that Apriori could not be run for *minsupport* lower than 3% on C20D10K and lower than 70% on C73D10K as it exceeds the memory limit. Census datasets are typical of statistical databases: highly correlated and dense data. Many items being extremely popular, this leads to a huge number of frequent itemsets from which few are closed.

5.2.3. Scale up on Census Data

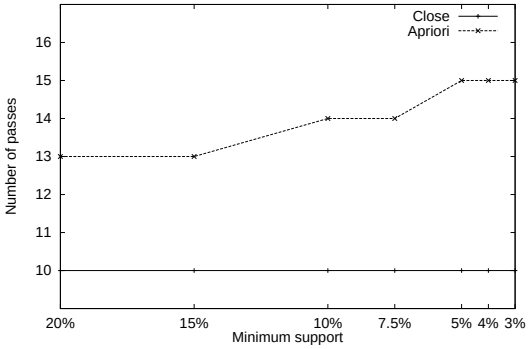
We finally examined how Apriori and Close behave as the object size and the number of objects are increased in census data. Results are shown in Figure 13. We first fixed the object size to 20 attributes (386 total items) and the *minsupport* level to 10%, and we increased the total number of objects from 20,000 up to 100,000. We can see here that both algorithms scale up linearly in the number of objects. Then, the number of objects was fixed to 10,000 and the *minsupport* level to 10%, and the object size varied from 10 (281 total items) up to 24 (408 total items). Apriori could not be run for higher object sizes. The graph shows that Close clearly has better sacle-up properties than Apriori, even if both algorithms are exponential in the size of the objects (number of items per object).

5.3. Algorithm choice

The problem of determining which algorithm will be more efficient according to the dataset is hard. In practice, however, some methods seem to give good results. First, the domain from which data come from can give us information on density and correlation of the dataset. In many domains, these characteristics have been studied, e.g. statistical data that are always highly correlated [17], text data that are correlated [4] and synthetic data weakly correlated [5]. Second, by determining the proportion of frequent 1-itemsets that are closed we can give a good indication on the efficiency of Close on the dataset. Our experiments have shown that if nearly all frequent 1-itemsets are closed (e.g. synthetic data) then many of all frequent itemsets are closed and the extra cost of closure computations will lead to better results of Apriori like algorithms. If few frequent 1-itemsets are closed (e.g. census data) then many items are correlated and few of all frequent itemsets are closed. In that case, the steps that the closure mechanism helps us to jump lead to better results of Close.

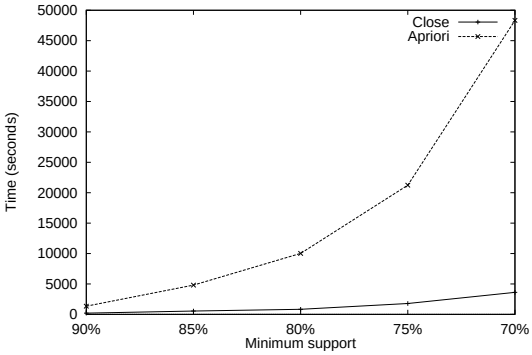


Execution times

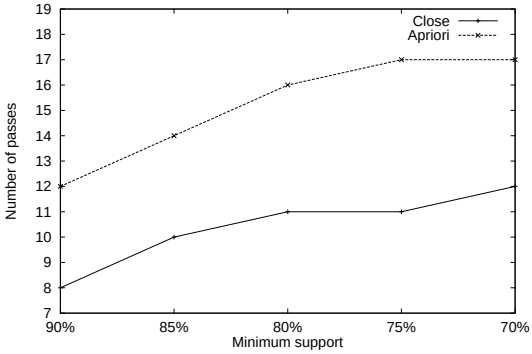


Number of database passes

Fig. 11: Performance of Apriori and Close on census data C20D10K

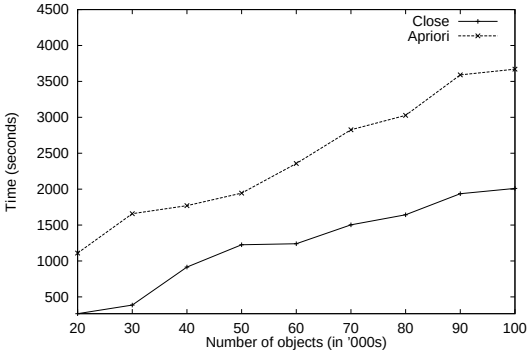


Execution times

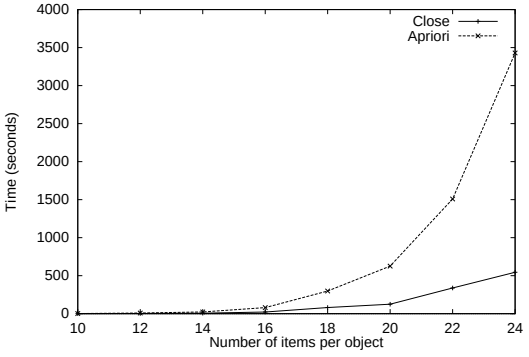


Number of database passes

Fig. 12: Performance of Apriori and Close on census data C73D10K



Scale-up increasing the number of objects



Scale-up increasing the number of items

Fig. 13: Scale-up properties of Apriori and Close on census data

6. CONCLUSION

We presented a new algorithm, called Close, for mining association rules in large databases. Close is based on the pruning of the closed itemset lattice, unlike existing algorithms that are all based on the pruning of the itemset lattice. As the size and the height of the closed itemset lattice of a database are often much smaller than those of the itemset lattice, Close can reduce both the number of database passes and the CPU overhead incurred by frequent itemsets search.

We conducted performance evaluations to compare Close to an optimized version of Apriori using prefix-tree, which corresponds to the basic approach for finding association rules by pruning the itemset lattice. Experiments were carried out using two types of databases: synthetic data (often used as a benchmark for mining market basket data) and census data (a real-life statistical database application). Experimental results showed that Close is less efficient, but gives nonetheless acceptable response times, for mining synthetic data. On the contrary, Close clearly outperforms Apriori in the case of census data, in particular for large problems (that are more significant of real-life datasets). The number of database passes is reduced from a quarter to a half in comparison with the number of passes Apriori needs. Moreover, in all the cases, Close was able to discover association rules for low *minsupport* values that Apriori cannot treat because of its memory space requirements. Close is particularly well suited to statistical database applications that are considered as difficult problems.

Besides the discovery of association rules, Close has another important feature: it provides an efficient computation of the Dedekind-MacNeille completion of an order [3, 7]. The DM completion is the smallest lattice associated with an order and is dually isomorphic to the closed itemset lattice of this order [19]. The closest works are algorithms [10, 11] which work only in main memory. Using the closed itemset lattice framework, Close supplies an efficient item clustering technic (unsupervised classification), another leading task in data mining [6, 21] and in machine learning [11].

The complete set of association rules is often huge and unmanageable. We thus seek a method to construct a basis of implication rules (exact association rules), using the Duquenne-Guigues theorem [8, 9] in order to have an useable set of frequent implication rules. We think this theorem is particularly well suited to Close as it uses the Galois closure operator h.

Acknowledgements — The authors would like to gratefully acknowledge the referees for their valuable comments and suggestions.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proceedings of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 207–216 (1993).

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proceedings of the 20th Int'l Conference on Very Large Data Bases*, pp. 478–499, Expanded version in IBM Research Report RJ9839 (1994).

[3] G. Birkhoff. Lattices theory. In *Coll. Pub. XXV*, volume 25. American Mathematical Society, Third edition (1967).

[4] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlation. *Proceedings of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 265–276 (1997).

[5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *Proceedings of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 255–264 (1997).

[6] M.-S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, **8**(6):866–883 (1996).

[7] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Fourth edition (1994).

[8] V. Duquenne. Contextual implications between attributes and some representation properties for finite lattice. In B. Ganter, R. Wille, and K. Wolff, editors, *Beiträge zur Begriffsanalyse*, pp. 213–240. B.I. Wissenschaftsverlag (1987).

[9] V. Duquenne and L.-L. Guigues. Famille minimale d'implication informatives résultant d'un tableau de données binaires. *Math. Sci. Hum.*, **24**(95):5–18 (1986).

[10] B. Ganter and K. Reuter. Finding all closed sets: A general approach. In *Order*, pp. 283–290. Kluwer Academic Publishers (1991).

[11] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, **11**(2):246–267 (1995).

[12] H. Mannila. Methods and problems in data mining. *Proceedings of the 6th Int'l Conference on Database Theory*, pp. 41–55 (1997).

[13] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pp. 181–192 (1994).

[14] A. M. Mueller. Fast sequential and parallel algorithms for association rules mining: A comparison. Technical report, Faculty of the Graduate School of The University of Maryland (1995).

[15] J. S. Park, M.-S. Chen, and P. S. Yu. An efficient hash based algorithm for mining association rules. *Proceedings of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 175–186 (1995).

[16] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in larges databases. *Proceedings of the 21th Int'l Conference on Very Large Data Bases*, pp. 432–444 (1995).

[17] A. Shoshani and H. K. T. Wong. Statistical and scientific database issues. *IEEE Tansactions on Software Engineering*, **11**(10):1071–1081 (1985).

[18] H. Toivonen. Sampling large databases for association rules. *Proceedings of the 22nd Int'l Conference on Very Large Data Bases*, pp. 134–145 (1996).

[19] R. Wille. Restructuring lattices theory: an approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pp. 445–470. Reidel, Dordrecht-Boston (1982).

[20] R. Wille. Concept lattices and conceptual knowledge systems. *Computers and Mathematics with Applications*, **23**:493–515 (1992).

[21] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. *Proceedings of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 103–114 (1996).