Pruning closed itemset lattices for association rules

Nicolas Pasquier, Yves Bastide, Rafik Taouil, Lotfi Lakhal

Pruning Closed Itemset Lattices for Association Rules

Nicolas Pasquier, Yves Bastide, Rafik Taouil, Lotfi Lakhal
{pasquier,bastide,taouil}@libd1.univ-bpclermont.fr
lakhal@ucfma.univ-bpclermont.fr

Laboratoire d'Informatique (LIMOS)
Université Blaise Pascal - Clermont-Ferrand II
Complexe Scientifique des Cézeaux
24, av. des Landais, 63177 Aubière Cedex France

Résumé

La découverte des règles d'association est l'un des principaux problèmes de l'extraction de connaissances dans les bases de données. De nombreux algorithmes efficaces ont été proposés, dont les plus remarquables sont Apriori, l'algorithme de Mannila, Partition, Sampling et DIC. Ces derniers sont tous basés sur la méthode de recherche de Apriori: l'élagage du treillis des parties (treillis des itemsets). Dans cet article, nous proposons un algorithme efficace basé sur une nouvelle méthode de recherche: l'élagage du treillis des fermés (treillis des itemsets fermés). Ce treillis qui est un sous-ordre du treillis des parties est étroitement lié au treillis de concepts de Wille dans son analyse formelle de concepts. Nous avons comparé expérimentalement Close à une version optimisée de Apriori et les résultats obtenus montrent la grande efficacité de Close dans le traitement des données denses et/ou corrélées telles que les données de rescensement (cas difficile). Nous avons également pu observer que Close donne des temps de réponse corrects dans le traitement des bases de données de ventes.

Mots-Clef: *extraction de connaissances; règles d'association; treillis; algorithmes.*

Abstract

Discovering association rules is one of the most important task in data mining and 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 of the subset lattice (itemset lattice). In this paper we propose an efficient algorithm, called Close, based on a new mining method: pruning of 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 data, and performs reasonably well for market basket style data.

Keywords: data mining; knowledge discovery; association rules; 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 the 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 have stiffnesses and 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, $I \Rightarrow I'$, where itemsets $I, I' \subset \mathcal{I}$ and $I \cap I' = \emptyset$. The *confidence* of an association rule $r : I \Rightarrow I'$ is the conditional probability that a transaction contains I', given that it contains I: $confidence(r) = support(I \cup I') \ / \ support(I)$. The support of r is defined as: $support(r) = support(I \cup I')$.

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 the association rules that hold with more than the given *minsupport* and *minconfidence*. This problem can be broken into two subproblems [1]:

1. Finding 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, generating all association rules $I_2 \Rightarrow I_1 - I_2 \mid I_2 \subset I_1$, with confidence greater or equal to *minconfidence*.

The second subproblem 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, 3, 9, 8, 11, 12, 13]. Although they are very different from each other, they are all based on the Apriori mining method [2]: pruning of the subset lattice for finding 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 perfomances 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 much smaller. Such a structure is closely related to Wille's concept lattice in formal concept analysis [5, 14, 15]. 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 the 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[1] and census data using the PUMS file for Kansas USA[2] and Section 6 concludes the paper.

[1] `http://www.almaden.ibm.com/cs/quest/syndata.html`
[2] `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

Figure 1: The transaction database $\mathcal{D}$

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 because 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 $\mathcal{I}$ with height equal to 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 6. 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 of 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.

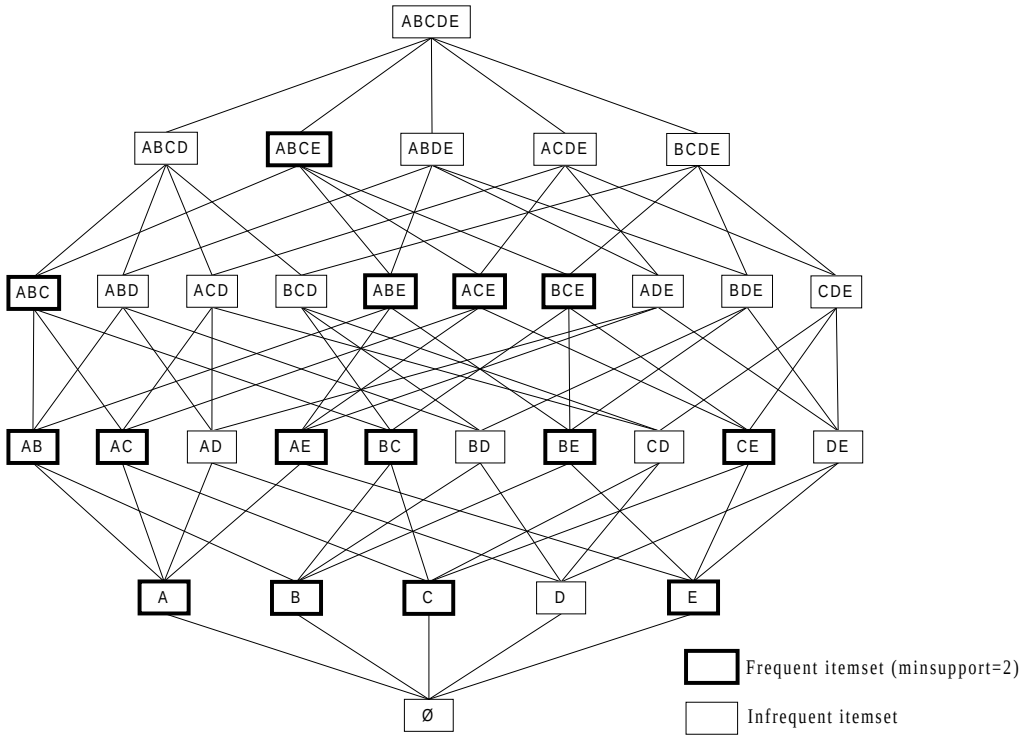


Figure 2: Itemset lattice of $\mathcal{D}$

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

Algorithm Apriori

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 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. Next, 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.

1) L_1 = {Large 1-itemsets};
2) **for** (k=2; $L_{k-1} \neq \emptyset$; k++) **do begin**
3) C_k = Apriori-Gen(L_{k-1}); // Generates candidates k-itemsets
4) **forall** transactions $t \in \mathcal{D}$ **do begin**
5) C_t = Subset(C_k,t); // Candidates contained in t
6) **forall** candidates $c \in C_t$ **do**
7) c.count++;
8) **end**
9) $L_k = \{ c \in C_k \mid c.\text{count} \geq minsupport \}$
10) **end**
11) Answer = $\bigcup_k L_k$;

Algorithm 1: Apriori frequent itemset discovery

Apriori-Gen Candidate Generation The function takes as argument the set L_{i-1} of frequent $(i-1)$-itemsets. It returns the set C_i of candidate i-itemsets, which is a superset of the set of all frequent i-itemsets. Two frequent itemsets of size $i-1$ with the same first $i-2$ items are joined, generating a new candidate itemset of size i:

insert into C_i
select $p.\text{item}_1$, $p.\text{item}_2$, ..., $p.\text{item}_{i-1}$, $q.\text{item}_{i-1}$
from L_{i-1} p, L_{i-1} q
where $p.\text{item}_1 = q.\text{item}_1$, ... , $p.\text{item}_{i-2} = q.\text{item}_{i-2}$, $p.\text{item}_{i-1} < q.\text{item}_{i-1}$;

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}:

forall candidate itemsets $c \in C_i$ **do begin**
forall $(i-1)$-subsets s of c **do begin**
if ($s \notin L_{i-1}$) **then**
delete c from C_i;

Example 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.

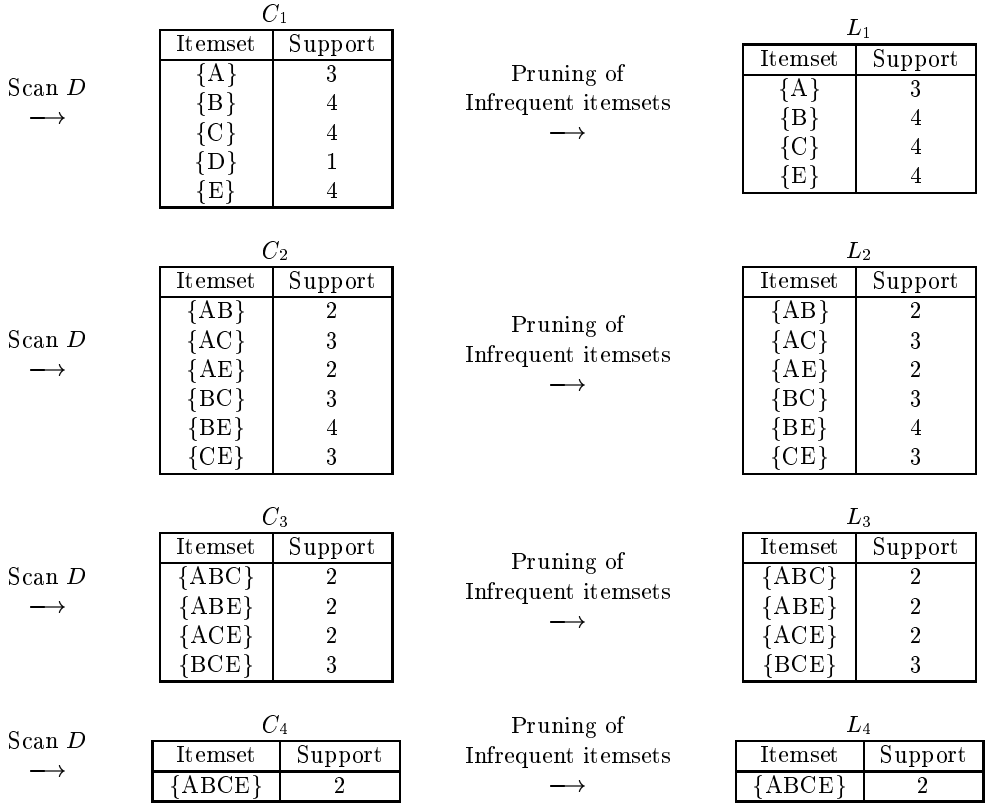


Figure 3: Discovering frequent itemsets with Apriori for $minsupport = 2(40\%)$

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 (generaly 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 the CPU overhead they involve can affect their performances.

2.2 Closed Itemset Lattices

In this section, we define *data mining context*, *Galois connection*, *closed itemset* and *closed itemset lattice*. Interested readers should consult [5] for further details on the order and lattice theory.

Data Mining Context A data mining context (a database) is a triple $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$. $\mathcal{O}$ and $\mathcal{I}$ are finite sets of objects and database 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 $o \in \mathcal{O}$ has the item $i \in \mathcal{I}$. Data mining context can be a relation, a class, or the result of an SQL/OQL query.

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:

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

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

$f(O)$ associates with O all items common to all objects $o \in O$ and $g(I)$ associates with I all objects containing 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 $g \circ f$ in $\mathcal{I}$ and $f \circ g$ in $\mathcal{O}$[3]

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

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 $O, O_1, O_2 \subseteq \mathcal{O}$ [5, 15]:

(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) $I \subseteq g \circ f(I)$	(2') $O \subseteq f \circ g(O)$
(3) $g \circ f(g \circ f(I)) = g \circ f(I)$	(3') $f \circ g(f \circ g(O)) = f \circ g(O)$
(4) $I_1 \subseteq I_2 \Longrightarrow g \circ f(I_1) \subseteq g \circ f(I_2)$	(4') $O_1 \subseteq O_2 \Longrightarrow f \circ g(O_1) \subseteq f \circ g(O_2)$
(5) $f \circ g(g(I)) = g(I)$	(5') $g \circ f(f(O)) = f(O)$

(6) $O \subseteq g(I) \Longleftrightarrow I \subseteq f(O)$

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

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. Having a lattice structure implies two properties:

i) 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 $C_1 \subseteq C_2$[4].

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

Below, we give the definitions of the *Join* and *Meet* elements extracted from Wille's basic theorem on concept lattices [5, 14]. For all $S \subseteq \mathcal{L}_\mathcal{C}$:

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

2.3 The Close Mining Method

The Close algorithm is fundamentally different from existing algorithms, since it is based on the pruning of the closed itemset lattice for finding 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 $\{2, 3, 4\}$. $\{B, C, E\}$ is called a frequent closed itemset for *minsupport* $= 2$ as support$(\{B, C, E\}) = \|\{2, 3, 4\}\| = 3 \geq$ *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 grouping 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 isomorphic to the concept lattice [14, 15], also called Galois lattice [7]. Figure 1 gives the closed itemset lattice of $\mathcal{D}$ with frequent closed itemsets for *minsupport* $= 2$ outlined.

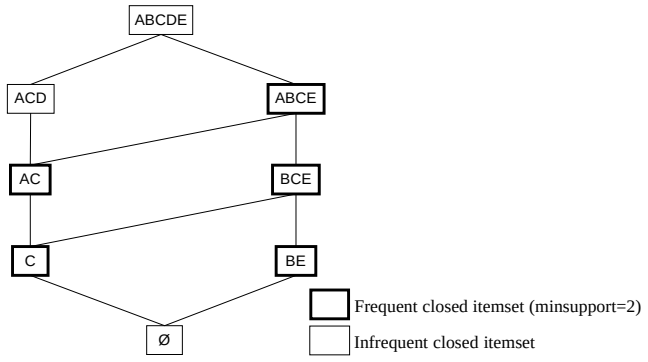


Figure 4: Closed itemset lattice of $\mathcal{D}$

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

Using the closed itemset lattice, which is a sub-order of the subset lattice, for finding frequent itemsets can improve the efficiency of the association rule discovery. Indeed, the proportion of itemsets that are closed and frequent is 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, $\|\mathcal{L}_{\mathcal{S}}\| = 2^{\|\mathcal{I}\|}$. Although in the worst case, the closed itemset lattice may grow exponentially, the growth is linear with respect to $\|\mathcal{D}\|$ when it exists an upper bound K on the object size ($\|o\|$). Then, the size of the closed itemset lattice is $\|\mathcal{L}_{\mathcal{C}}\| \leq 2^K \|\mathcal{D}\|$. Moreover, experimental applications and theoretical results based on a uniform distribution hypothesis showed that the average growth factor is far less than the 2^K bound. Actually, we can observe that $\|\mathcal{L}_{\mathcal{C}}\| \leq \mu \|\mathcal{D}\|$, where μ is the mean value for $\|o\|$ [7]. Using the closed itemset lattice framework we can deduce 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[5] of a frequent **closed** itemset are frequent.

iv) All sup-closed itemsets[6] 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.

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

1. Discovering all **frequent closed itemsets** in $\mathcal{D}$, i.e. itemsets that are closed and have support greater or equal to *minsupport*.
2. Deriving 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 frequent closed itemset supports .
3. For each frequent itemset I found in phase 2, generating all **association rules** that can be derived from I and have confidence greater or equal to *minconfidence*.

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

3 Semantics of Association Rules

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 $\mathcal{D} = (\mathcal{O}, \mathcal{I}, \mathcal{R})$. Then, we define *association rules* and *valid association rules* using frequent closed itemsets.

3.1 Frequent Itemsets

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:

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

[5]Closed subsets of a closed itemset.
[6]Closed supersets of a closed itemset.

Frequent Itemsets The itemset I is said to be frequent if the support of I in $\mathcal{D}$ is at least *minsupport*. We define the set L of frequent itemsets in $\mathcal{D}$ as:

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

Maximal Frequent Itemsets Let L be the set of frequent itemsets. We define the set M of maximal frequent itemsets as:

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

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: $I' \subseteq I \Longrightarrow g(I') \supseteq g(I) \Longrightarrow \text{support}(I') \geq \text{support}(I) \geq minsupport$. 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: $I \supseteq I' \Longrightarrow g(I) \subseteq g(I') \Longrightarrow \text{support}(I) \leq \text{support}(I') \leq minsupport$. So, we get: $I \notin L$.

3.2 Frequent Closed Itemsets

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:

$$FC = \{C \subseteq \mathcal{I} \mid C = g \circ f(C) \text{ and } support(C) \geq minsupport\}$$

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.

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 $g \circ f(I)$, the closure of I. Since $g(I)$ is closed and by consequence $f \circ g(g(I)) = g(I)$ (according to Property (5) of the Galois connection), we have:

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

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. $I = g \circ f(I)$. Let $I \in M$ be a maximal frequent itemset. By definition, $\forall I' \supset I$, I' is not frequent, i.e. $I' \notin M$. According to Property (2) of the Galois connection $I \subseteq g \circ f(I)$ and, since I is maximal and $support(g \circ f(I)) = support(I) \geq minsupport$, we can conclude $I = g \circ f(I)$. I is a maximal frequent closed itemset. Since all maximal frequent itemsets are also a maximal frequent closed itemsets, we get: $M = MC$.

3.3 Association Rules

Association Rules An association rule is an implication between itemsets of the form $I \Rightarrow I'$ where $I, I' \subset \mathcal{I}$ and $I \cap I' = \emptyset$. The itemset I is called the *antecedent* of the rule, and the itemset I' 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 $r : I \Rightarrow I'$ are:

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

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:

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

4 Algorithm Close

In Section 4.1 we describe our discovery of the frequent closed itemsets. In Section 4.2 we give our method for deriving 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 2. For each iteration, the algorithm constructs a set of candidate frequent closed itemsets, determines the frequent closed itemsets using the *minsupport* threshold and then computes the generator itemsets that will be used during the next iteration for constructing the set of candidate frequent closed itemsets. In each iteration, one pass over the database is necessary, for constructing the set of candidate frequent closed itemsets (closures of generators).

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

Table 2: Notation

The first operation of the algorithm (step 1) initializes the set of generator itemsets in FCC_1 with the items present in the data mining context, i.e. elements of the set $\mathcal{I}$, needing no database pass. Each of the following iterations consists of three phases. First, the closure function is applied to each generator in FCC_i, determining the candidate frequent closed itemsets and their support. 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: the closed itemsets with sufficient support

1) generators in FCC_1 = {1-itemsets};
2) **for** (i=1; FCC_i.generator$\neq \emptyset$; i++) **do begin**
3) closures in $FCC_i = \emptyset$;
4) supports in $FCC_i = 0$;
5) FCC_i = Gen-Closure(FCC_i); // Produces closures of generators (see Section 4.1.1)
6) **forall** candidate closed itemsets $c \in FCC_i$ **do begin**
7) **if** (c.support $\geq minsupport$) **then** // Pruning infrequent closures
8) $FC_i = FC_i \cup c$; // Insert c in FC_i
9) **end**
10) FCC_{i+1} = Gen-Generator(FC_i); // Creates generators of iteration i+1
11) **end** (see Section 4.1.2)
12) Answer $FC = \bigcup_{j=1}^{j=i-1}(FC_j.\text{closure}, FC_j.\text{support})$;

Algorithm 2: Algorithm Close

value are inserted in the set of frequent closed itemsets FC_i. Finally, the generators of the set FCC_{i+1} are determined by applying the function *Gen-Generator* (described in Section 4.1.2) to the generators of the 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 supports are known.

4.1.1 Gen-Closure Function

The closure function *Gen-Closure* takes as argument the set of generator itemsets in FCC_i. It updates FCC_i with, for each generator p, the closed itemset p.closure and its support count p.support obtained by applying the closure operator $g \circ f$ to p. Algorithm 3 gives the pseudo-code of the function. The method used for computing closed itemsets is based on Proposition 1.

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

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

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

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

We can conclude that $S = S'$, thus $g \circ f(I) = H$.

Using Proposition 1, only one database pass is necessary for computing the closures of the generators of an iteration i, and their support. The function works as follows. For each object o in $\mathcal{D}$, we create the set G_o containing all generators in FCC_i that are subsets of the object itemset $f(\{o\})$ (step 2). Then, for each generator p in G_o, we update the associated closed itemset p.closure (step 3 to 7). If the object o is the first one containing the generator, p.closure is empty and we assign to it the object itemset (step 4). Otherwise, the intersection between p.closure and the object itemset gives us the new p.closure (step 5). Then, we increment the closed itemset support p.support (step 6). At the end of the function, we have for each generator p in FCC_i, the closed itemset p.closure corresponding to the intersection of all objects containing p and its associated support count p.support corresponding to the number of objects containing p.closure (support count of the generator and its closure are equal according to Property 5).

4.1.2 Gen-Generator Function

The function *Gen-Generator* takes as argument the set of frequent closed itemsets FC_i. Based on Proposition 2, it returns the set FCC_{i+1} containing all generator $(i+1)$-itemsets that will be

1) **forall** objects $o \in O$ **do begin**
2) $G_o = \text{Subset}(FCC_i.\text{generator}, f(\{o\}))$; // Generators subsets of f({o}) (Section 4.1.3)
3) **forall** generators $p \in G_o$ **do begin**
4) **if** (p.closure $= \emptyset$) **then** p.closure $= f(\{o\})$;
5) **else** p.closure $= p$.closure $\cap f(\{o\})$;
6) p.support++;
7) **end**
8) **end**
9) Answer $= \bigcup \{ c \in FCC_i \mid c.\text{closure} \neq \emptyset \}$;

Algorithm 3: Function Gen-Closure

used during iteration $i+1$ for constructing the set of candidate frequent closed itemsets. The function first generates all potential generator $(i+1)$-itemsets using generator i-itemsets in FC_i. Then, based on Proposition 3, the potential generators produced that will lead to useless computing (infrequent closed itemsets) or redundancies (frequent closed itemsets already produced) are deleted from FCC_{i+1}.

Proposition 2: Let I_1, I_2 be two generator itemsets. We have:

$$g \circ f(I_1 \cup I_2) = g \circ f(g \circ f(I_1) \cup g \circ f(I_2))$$

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

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

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

$$g \circ f(g \circ f(I_1) \cup g \circ f(I_2)) \subseteq g \circ f(g \circ f(I_1 \cup I_2)) \implies g \circ f(g \circ f(I_1) \cup g \circ f(I_2)) \subseteq g \circ f(I_1 \cup I_2) \quad (2)$$

From (1) and (2), we can conclude $g \circ f(I_1 \cup I_2) = g \circ f(g \circ f(I_1) \cup g \circ f(I_2))$.

Proposition 3: Let I be a generator i-itemset 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 $I \subseteq g \circ f(s_a)$, then $g \circ f(I) = g \circ f(s_a)$.

Proof: Let I be an i-itemset and $s_a \in \mathcal{S}$, an $(i-1)$-subset of I.

$$I \subseteq g \circ f(s_a) \implies g \circ f(I) \subseteq g \circ f(g \circ f(s_a)) \implies g \circ f(I) \subseteq g \circ f(s_a) \quad (1)$$

$$s_a \in \mathcal{S} \implies s_a \subseteq I \implies g \circ f(s_a) \subseteq g \circ f(I) \quad (2)$$

From (1) and (2), we deduce $g \circ f(I) = g \circ f(s_a)$.

The function Gen-Generator works as follows. We first apply the combinatorial phase of Apriori-Gen [2] to the set of generators in FC_i giving us 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 $i+1$.

1) **insert into** FCC_{i+1}.generator
2) **select** $p.\text{item}_1$, $p.\text{item}_2$, ..., $p.\text{item}_i$, $q.\text{item}_i$
3) **from** FC_i.generator p, FC_i.generator q
4) **where** $p.\text{item}_1 = q.\text{item}_1$, ... , $p.\text{item}_{i-1} = q.\text{item}_{i-1}$, $p.\text{item}_i < q.\text{item}_i$;

Then, we prune the resulting set using two strategies. First, like in Apriori-Gen, for all potential generator p created we test the presence of all its i-subsets in FC_i.generator. Indeed, if one of them is absent from FC_i and according to Property 2 we deduce that p is not frequent and we can remove it from FCC_{i+1}.

Let's take an example. Suppose that the set of candidate frequent closed itemsets FC_2 contains the generator itemsets AB, AC with respective closures ABC, ACD all frequent. The Gen-Generator

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

function will create $ABC = AB \cup AC$ as a new potential generator in FCC_3. The first pruning strategy will remove ABC from FCC_3 since $BC \notin FC_2$.generator as it is infrequent.

The second pruning strategy is 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 Proposition 3), so we can remove p from FCC_{i+1}.

11) **forall** generators $p \in FCC_{i+1}$.generator **do begin**
12) $S_p = \text{Subset}(FCC_i.\text{generator},p)$; // Subsets of p that are existing generators in FCC_i
13) **forall** $s \in S_p$ **do begin**
14) **if** ($p \subseteq s$.closure) **then**
15) **delete** p from FCC_{i+1}.generator;
16) **end**
17) **end**

Let's give another example. Suppose that the set of frequent closed itemsets FC_2 contains generators AB, AC, BC with respective closures AB, ABC, BC all 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 can deduce that $closure(ABC) = closure(AC)$ and the computation of the closure of ABC is useless.

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 [10]. 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 itemset 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.

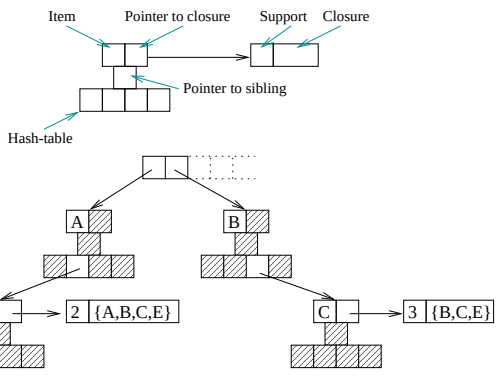


Figure 5: Prefix-tree of the set FCC_2 in Figure 6

The subset function takes as arguments a set of generators G and an itemset c. 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 are at 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 on the data mining context $\mathcal{D}$ given in Figure 1. We assume that *minsupport* is 2 (40%). Step 1 initializes the set of generators in FCC_1 with the list of items in $\mathcal{D}$. Calling Gen-Closure at step 3 gives for every generator p the candidate closed itemset p.closure and its support count p.support. In step 4 through 7 we generate FC_1 by pruning FCC_1 according to *minsupport*. In step 9 we produce the generators in FCC_2 by applying the function Gen-Generator 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 $C \subseteq AC$, so the generator $A \cup C$ is deleted by the second pruning step of the function Gen-Generator.

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.

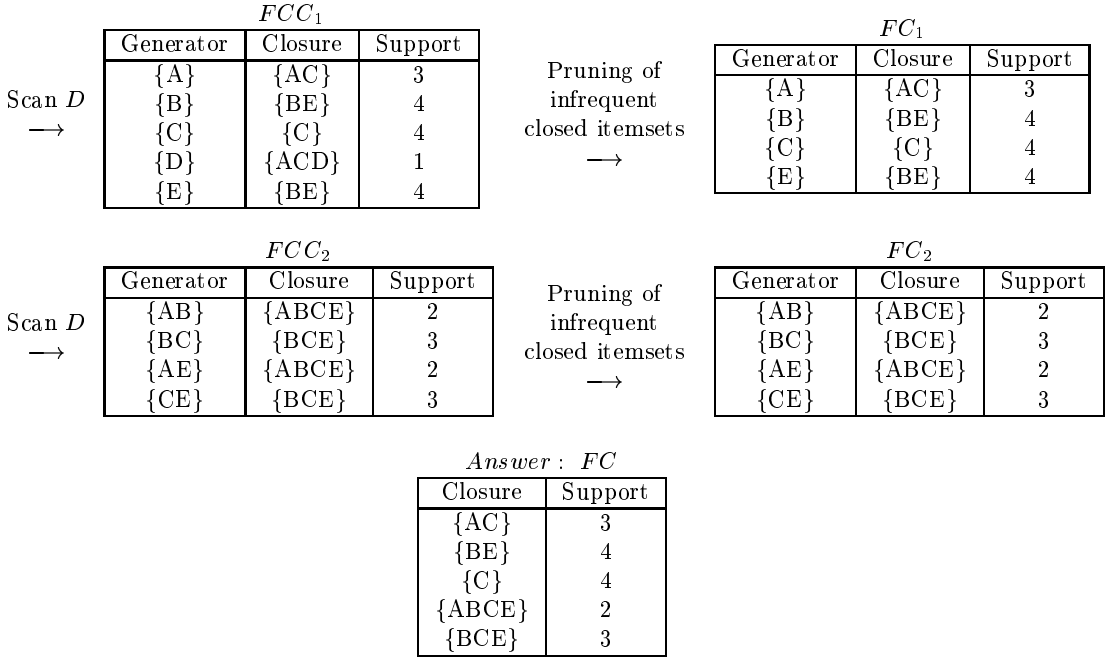


Figure 6: Discovering frequent closed itemsets with Close for *minsupport* = 2 (40%)

Correctness We must ensure that all frequent closed itemsets have been produced. Starting with the set of items in the data mining context and making successive closures of the unions of closed itemsets found in the previous iteration gives the complete closed itemset lattice, based on the *Join* operator of Wille's basic theorem on concept lattices (see section 2.2). According to Proposition 2, working on generator itemsets is identical to working on their closures. The method we use for creating the generators (similar to the one in Apriori-Gen) is an efficient union method. This method yields all possible unions, avoiding redundancies, given the lexicographic

order among items. The two pruning steps of the set of generators avoid useless computations by removing infrequent generators and generators which closure was already found (Proposition 3).

4.2 Deriving Frequent Itemsets

The pseudo-code for deriving frequent itemsets is given in Algorithm 4. It uses as its input the set of frequent closed itemsets $FC = \bigcup_i FC_i$ and gives the set of frequent itemsets $L = \bigcup_k L_k$ as the output. In step 1 through 5, we put each frequent closed itemset c from FC in the set of frequent itemset $L_{\|c\|}$ corresponding to the size of c, and we determine the size k of the largest frequent itemsets. Then, during step 6 to 15 we construct all sets L_i, starting from L_k down to L_1. In each iteration, we complete the set L_{i-1} using the itemsets in L_i. For each i-itemset c in L_i, we generate all $(i-1)$-subsets of c. 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.

```
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;
```

Algorithm 4: Deriving frequent itemsets

Example 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 $(i-1)$-subset 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 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 Proposition 3, 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

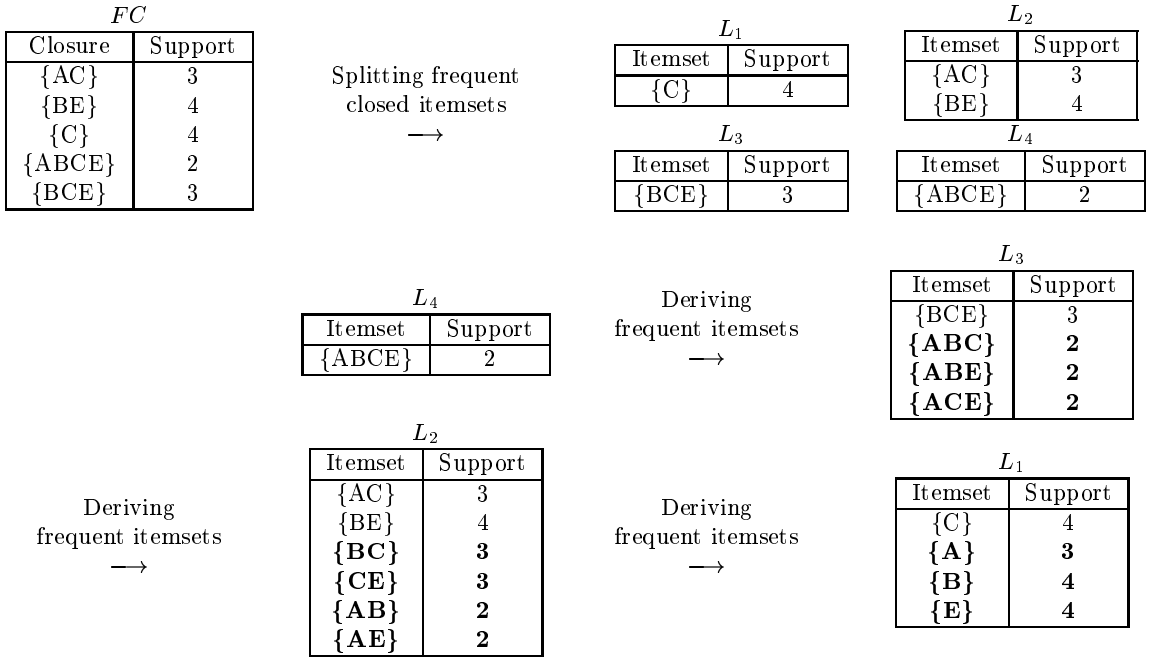


Figure 7: Deriving frequent itemsets for *minsupport* = 2 (40%)

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.

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 support are known. In this section, we describe an adapted version of Apriori rule generation algorithm [2] (implemented for our experiments). For every frequent itemset I_1 we derive all subsets I_2 of I_1 and we compute the ratio $support(I_1)/support(I_2)$. 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 5.

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

```
1)  forall frequent k-itemsets l_k ∈ L_k | 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 : (l_k-h_1) ⇒ h_1
5)          if ( confidence ≥ minconfidence ) then
6)              AR = AR ∪ {r : (l_k − h_1) ⇒ h_1};
7)          else H_1 = H_1 − {h_1}                          // H_1 = {1-item consequents of valid
8)      end                                                              rules from l_k}
9)      call GenRules(l_k, H_1);
10) end

11) Procedure GenRules(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});    // Confidence of
16)             if ( confidence ≥ minconfidence ) then                r : (l_k-h_{m+1}) ⇒ h_{m+1}
17)                 AR = AR ∪ {r : (l_k − h_{m+1}) ⇒ h_{m+1}};
18)             else
19)                 delete h_{m+1} from H_{m+1};
20)         end
21)         call GenRules(l_k, H_{m+1});
22)     end
23) end
```

Algorithm 5: Generating valid association rules

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

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

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 1, we describe the datasets used for the experiments. We then compare relative performances of the two algorithms in Section 2.

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 dataset, we used the program described in [2]. This dataset, called T10I4D100K, contains 100,000 objects for an average object size of 10 items and an average size of the maximal potentially frequent itemsets of 4.

The census data were extracted from the Kansas 1990 PUMS file (Public Use Microdata Samples), in the same way as [3] for the PUMS file of Washington (unavailable through Internet at the time

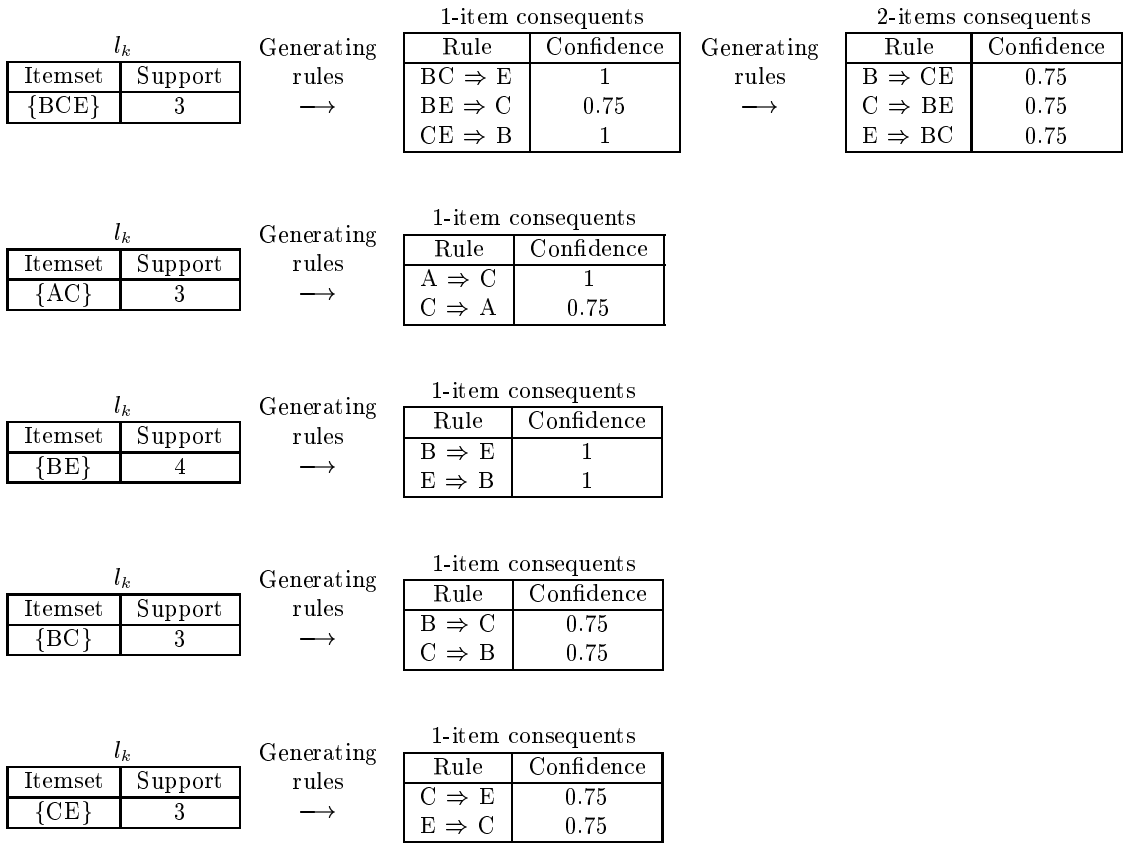


Figure 9: Generating valid association rules for *minsupport*=3 (60%) and *minconfidence*=0.5 (50%)

of the experiments). Unlike in [3] 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).

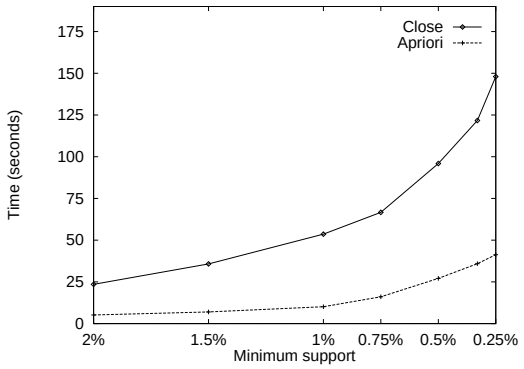
5.2 Relative Performance of Apriori and Close

5.2.1 Synthetic Data

We used the same values for *minsupport* as the ones used in [2], ranging from 2% to 0.25%. The figure beside shows the execution times of Apriori and Close on the dataset T10I4D100K. 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: two minutes and a half for the longest execution.

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 the execution times can be counted in hours. It should furthermore be noted that Apriori could not be run for *minsupport*



Execution times on T10I4D100K

Figure 10: Performance of Apriori and Close on synthetic data

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.

5.2.3 Scale up on Census Data

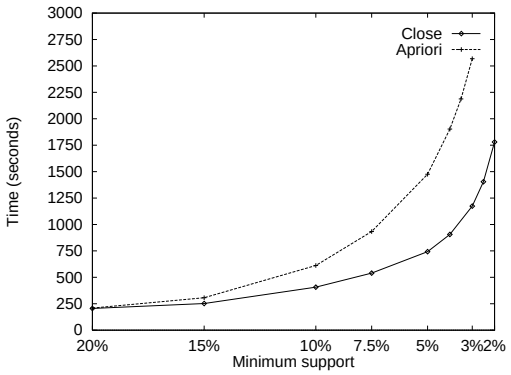
We finally examined how Apriori and Close behave as the object size is increased in census data. The number of objects was fixed to 10,000 and the *minsupport* level was set to 10%. The object size varied from 10 (281 total items) up to 24 (408 total items). Apriori could not be run for higher object sizes. Results are shown in Figure 13. We can see here that, as expected, Close outperforms Apriori both in execution times and in memory space requirements.

6 Conclusion

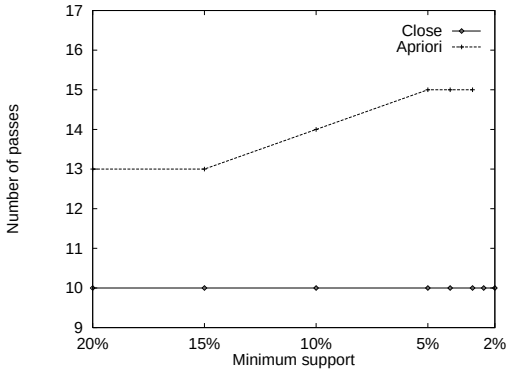
We presented a new algorithm, called Close, for mining association rules in large databases. Close is based on the pruning of the closed itemset lattice, unlike existing algorithms that are all based on the pruning of the itemset lattice. As the number of itemsets and the height of the closed itemset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

In addition to the discovery of association rules, our algorithm has another important feature: Close gives an efficient computing of the Dedekind-MacNeille completion of an order [5], which is the smallest lattice associated with an order and is isomorphic to the closed itemset lattice [14]. The closest works are algorithms [6, 7] which work only in main memory. Using such a structure, Close supplies an efficient data clustering technic (unsupervised classification), another important task in data mining [4, 16] and in machine learning [7].

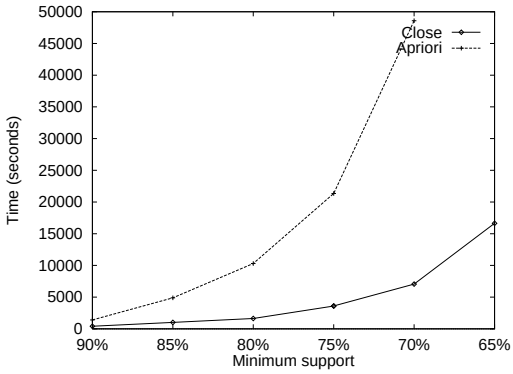


Execution times

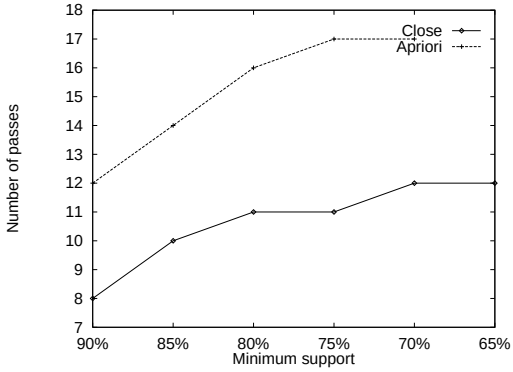


Number of database passes

Figure 11: Performance of Apriori and Close on census data C20D10K



Execution times



Number of database passes

Figure 12: Performance of Apriori and Close on census data C73D10K

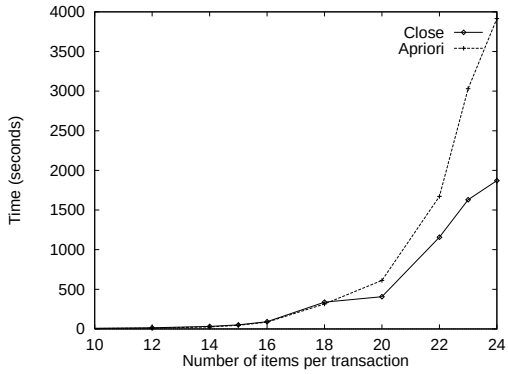


Figure 13: Scale-up properties of Apriori and Close on census data

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*, pages 207–216, May 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*, pages 478–499, June 1994. Expanded version in IBM Research Report RJ9839.

[3] 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*, pages 255–264, May 1997.

[4] 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, December 1996.

[5] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

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

[7] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. In N. Cercone and G. McCalla, editors, *Computational Intelligence*, volume 11, pages 246–267. Blackwell Publishers, May 1995.

[8] H. Mannila. Methods and problems in data mining. *Proceedings of the Int'l Conference on Database Theory*, pages 41–55, January 1997.

[9] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. *Proceedings of the Int'l Conference on Knowledge Discovery in Databases*, pages 181–192, July 1994.

[10] 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.

[11] 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*, pages 175–186, May 1995.

[12] 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*, pages 432–444, September 1995.

[13] H. Toivonen. Sampling large databases for association rules. *Proceedings of the 22th Int'l Conference on Very Large Data Bases*, pages 134–145, September 1996.

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

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

[16] 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*, pages 103–114, June 1996.