Galois Connections and Data Mining[1]

Dana Cristofor

Laurentiu Cristofor

Dan A. Simovici
University of Massachusetts at Boston, Department of Mathematics and Computer Science, Boston, Massachusetts 02125[2]

Abstract: We investigate the application of Galois connections to the identification of frequent item sets, a central problem in data mining. Starting from the notion of closure generated by a Galois connection, we define the notion of extended closure, and we use these notions to improve the classical `Apriori` algorithm. Our experimental study shows that in certain situations, the algorithms that we describe outperform the `Apriori` algorithm. Also, these algorithms scale up linearly.

Key Words: Galois connection, closure, extended closure, support, frequent set of items

Category: H.2.0, E.5

1 Introduction

Galois connections are algebraic constructions which play an important role in lattice theory, universal algebras and, more recently, in computer science (see [4]). We demonstrate their usefulness as an algebraic and conceptual tool in designing efficient algorithms for the identification of frequent sets of items, as defined in data mining.

Let $(P, \leq)$ and $(Q, \leq)$ be two partially ordered sets. *A Galois connection* between P and Q is a pair of mappings (Φ, Ψ) such that $\Phi : P \longrightarrow Q, \Psi : Q \longrightarrow P$ and:

$$x \leq x' \text{ implies } \Phi(x) \geq \Phi(x'),$$
$$y \leq y' \text{ implies } \Psi(y) \geq \Psi(y'),$$
$$x \leq \Psi(\Phi(x)) \text{ and } y \leq \Phi(\Psi(y)),$$

for $x, x' \in P$ and $y, y' \in Q$.

It is easy to verify (see [4]) that $\Phi(\Psi(\Phi(x))) = x$ and $\Psi(\Phi(\Psi(y))) = y$ for $x \in X$ and $y \in Y$.

Let $(P, \leq)$ be a complete lattice, that is, a poset such that for any $K \subseteq P$ there exist both $\sup K$ and $\inf K$. A *closure* is a mapping $\mathtt{cl} : P \longrightarrow P$ such that the following conditions are satisfied:

1. $x \leq \mathtt{cl}(x)$,
2. $\mathtt{cl}(x) = \mathtt{cl}(\mathtt{cl}(x))$ and

[1] C. S. Calude and G. Ştefănescu (eds.). Special issue dedicated to *Professor Sergiu Rudeanu Festschrift.*

[2] e-mail: `dana,laur,dsim@cs.umb.edu`

3. if $x \leq x'$ then $\mathtt{cl}(x) \leq \mathtt{cl}(x')$,

for all $x, x' \in P$. An element $x \in P$ is $\mathtt{cl}$-*closed* if $\mathtt{cl}(x) = x$. The set of $\mathtt{cl}$-closed elements will be denoted by $C_{\mathtt{cl}}$.

For any Galois connection $\mathsf{c} = (\Phi, \Psi)$ between the complete lattices $(P, \leq)$ and $(Q, \leq)$ the mapping $\mathtt{cl}_{\mathsf{c}} = \Psi\Phi$ is a closure on P while the mapping $\mathtt{cl}'_{\mathsf{c}} = \Phi\Psi$ is a closure on Q. It is easy to see that in this case the mapping $\beta_{\mathsf{c}} = \Phi \lceil C_{\mathtt{cl}_{\mathsf{c}}}$ is a bijection between $C_{\mathtt{cl}_{\mathsf{c}}}$ and $C_{\mathtt{cl}'_{\mathsf{c}}}$.

A standard method for generating Galois connections is through the notion of *polarity*. Let X, Y be two sets and let $R \subseteq X \times Y$ be a relation. Define the mappings $\Phi : \mathcal{P}(X) \longrightarrow \mathcal{P}(Y)$ and $\Psi : \mathcal{P}(Y) \longrightarrow \mathcal{P}(X)$ by

$$\Phi(K) = \{y \mid y \in Y, (x, y) \in R \text{ for all } x \in K\},$$

for $K \subseteq X$ and

$$\Psi(H) = \{x \mid x \in X, (x, y) \in R \text{ for all } y \in H\},$$

for $H \subseteq Y$. The pair $\mathsf{c} = (\Phi, \Psi)$ introduced above is a Galois connection and is usually referred to as *the polarity on X and Y determined by the relation R*.

Definition 1.1 *Let $\mathsf{c} = (\Phi, \Psi)$ be a polarity on the sets X and Y.*

The semidistances *generated by c are the mappings $d_{\mathsf{c}} : \mathcal{P}(X) \times \mathcal{P}(X) \longrightarrow \mathbf{N}$ and $d'_{\mathsf{c}} : \mathcal{P}(Y) \times \mathcal{P}(Y) \longrightarrow \mathbf{N}$ defined by $d_{\mathsf{c}}(U_0, U_1) = |\Phi(U_0) \oplus \Phi(U_1)|$ for $U_0, U_1 \in \mathcal{P}(X)$, where $\oplus$ is the symmetric difference operation, and $d'_{\mathsf{c}}(V_0, V_1) = |\Psi(V_0) \oplus \Psi(V_1)|$ for $V_0, V_1 \in Y$.*

The proximities *generated by c are the mappings $p_{\mathsf{c}} : \mathcal{P}(X) \times \mathcal{P}(X) \longrightarrow \mathbf{N}$ and $p'_{\mathsf{c}} : \mathcal{P}(Y) \times \mathcal{P}(Y) \longrightarrow \mathbf{N}$ defined by $p_{\mathsf{c}}(U_0, U_1) = |\Phi(U_0) \cap \Phi(U_1)|$ for $U_0, U_1 \in \mathcal{P}(X)$, and $p'_{\mathsf{c}}(V_0, V_1) = |\Psi(V_0) \cap \Psi(V_1)|$ for $V_0, V_1 \in Y$.*

The weight functions *are the mappings $w_{\mathsf{c}} : \mathcal{P}(X) \longrightarrow \mathbf{N}$ and $w'_{\mathsf{c}} : \mathcal{P}(Y) \longrightarrow \mathbf{N}$ given by $w_{\mathsf{c}}(U) = |\Phi(U)|$ and $w_{\mathsf{c}}(V) = |\Psi(V)|$ for every $U \in \mathcal{P}(X)$ and $V \in \mathcal{P}(Y)$.*

If the Galois connection c is clear from context then the subscript c may be omitted. Also, if a set consists of only one element ℓ we may write simply ℓ instead of $\{\ell\}$.

Proposition 1.2 *Let c be a polarity on the sets X and Y. We have*

1. $d_{\mathsf{c}}(U_0, U_0 \cup U_1) + d_{\mathsf{c}}(U_1, U_0 \cup U_1) = d_{\mathsf{c}}(U_0, U_1)$,
2. $p_{\mathsf{c}}(U_0, U_0 \cup U_1) = p_{\mathsf{c}}(U_1, U_0 \cup U_1) = p_{\mathsf{c}}(U_0, U_1)$,
3. $d_{\mathsf{c}}(U_0, \mathtt{cl}_{\mathsf{c}}(U_0)) = 0$,

for every $U_0, U_1 \in \mathcal{P}(X)$.

Proof. The definition of d implies that

$$\begin{aligned}
d_{\mathsf{c}}(U_0, U_0 \cup U_1) &= |\Phi(U_0) \oplus \Phi(U_0 \cup U_1)| \\
&= |\Phi(U_0) \oplus (\Phi(U_0) \cap \Phi(U_1))| \\
&= |\Phi(U_0) - \Phi(U_1)| \\
d_{\mathsf{c}}(U_1, U_0 \cup U_1) &= |\Phi(U_1) \oplus \Phi(U_0 \cup U_1)| \\
&= |\Phi(U_1) \oplus (\Phi(U_0) \cap \Phi(U_1))| \\
&= |\Phi(U_1) - \Phi(U_0)|,
\end{aligned}$$

which imply

$$\begin{aligned}
d_{\mathsf{c}}(U_0, U_1) &= |\Phi(U_0) \oplus \Phi(U_1)| \\
&= |\Phi(U_0) - \Phi(U_1)| + |\Phi(U_1) - \Phi(U_0)| \\
&= d_{\mathsf{c}}(U_0, U_0 \cup U_1) + d_{\mathsf{c}}(U_1, U_0 \cup U_1).
\end{aligned}$$

For the second part of the proposition we note that

$$\begin{aligned}
p_{\mathsf{c}}(U_0, U_0 \cup U_1) &= |\Phi(U_0) \cap \Phi(U_0 \cup U_1)| \\
&= |\Phi(U_0) \cap (\Phi(U_0) \cap \Phi(U_1))| \\
&= |\Phi(U_0) \cap \Phi(U_1)| \\
&= p_{\mathsf{c}}(U_0, U_1)
\end{aligned}$$

for every $U_0, U_1 \in \mathcal{P}(X)$. The proof of $p_{\mathsf{c}}(U_1, U_0 \cup U_1) = p_{\mathsf{c}}(U_0, U_1)$ is entirely similar.

Finally, note that $d_{\mathsf{c}}(U_0, \mathtt{cl}_{\mathsf{c}}(U_0)) = |\Phi(U_0) \oplus \Phi(\Psi(\Phi(U_0)))| = |\Phi(U_0) \oplus \Phi(U_0)| = 0$. □

Proposition 1.3 *The weight function w_{c} generated by a polarity $\mathsf{c} = (\Phi, \Psi)$ on X and Y has the following properties:*

1. $\max\{w_{\mathsf{c}}(U_0), w_{\mathsf{c}}(U_1)\} \leq d_{\mathsf{c}}(U_0, U_1) + p_{\mathsf{c}}(U_0, U_1)$,
2. $p_{\mathsf{c}}(U_0, U_1) = w_{\mathsf{c}}(U_0 \cup U_1) \leq \min\{w_{\mathsf{c}}(U_0), w_{\mathsf{c}}(U_1)\}$,
3. $d_{\mathsf{c}}(U_0, U_1) + p_{\mathsf{c}}(U_0, U_1) \leq w_{\mathsf{c}}(U_0 \cap U_1)$,
4. $w_{\mathsf{c}}(U_0) + w_{\mathsf{c}}(U_1) = d_{\mathsf{c}}(U_0, U_1) + 2p_{\mathsf{c}}(U_0, U_1)$,
5. $w_{\mathsf{c}}(U_0) = w_{\mathsf{c}}(\mathtt{cl}_{\mathsf{c}}(U_0))$,

for every $U_0, U_1 \in \mathcal{P}(X)$.

Proof. We give the argument for the third part of the proposition. Note that

$$\begin{aligned}
d_{\mathsf{c}}(U_0, U_1) + p_{\mathsf{c}}(U_0, U_1) &= |\Phi(U_0) \oplus \Phi(U_1)| + |\Phi(U_0) \cap \Phi(U_1)| \\
&= |\Phi(U_0) \cup \Phi(U_1)| \\
&\leq |\Phi(U_0 \cap U_1)| \\
&= w_{\mathsf{c}}(U_0 \cap U_1).
\end{aligned}$$

The rest of the proof is elementary and is left to the reader. □

Proposition 1.4 *The proximity function p_{c} generated by a polarity $\mathsf{c} = (\Phi, \Psi)$ on X and Y has the following properties:*

1. $p_{\mathsf{c}}(U, U) = w_{\mathsf{c}}(U)$
2. $p_{\mathsf{c}}(U_0, U_1) = p_{\mathsf{c}}(U_1, U_0)$
3. $p_{\mathsf{c}}(U_0, U_1) = p_{\mathsf{c}}(\mathtt{cl}_{\mathsf{c}}(U_0), \mathtt{cl}_{\mathsf{c}}(U_1))$
4. $p_{\mathsf{c}}(U_0, U_2) \geq p_{\mathsf{c}}(U_0, U_1) + p_{\mathsf{c}}(U_1, U_2) - w_{\mathsf{c}}(U_1)$,

for every $U, U_0, U_1, U_2 \in \mathcal{P}(X)$.

Proof. The argument is elementary and is omitted. □

Proposition 1.5 *Let $U_0, U_1 \subseteq X$ and let c be a polarity on the sets X and Y. For every sets U', U'' such that $U_0 \subseteq U' \subseteq \mathtt{cl}_{\mathsf{c}}(U_0)$ and $U_1 \subseteq U'' \subseteq \mathtt{cl}'_{\mathsf{c}}(U_1)$ we have $d_{\mathsf{c}}(U', U'') = d_{\mathsf{c}}(\mathtt{cl}(U_0), \mathtt{cl}(U_1))$.*

Proof. Note that if $U_0 \subseteq U' \subseteq \mathtt{cl}_{\mathsf{c}}(U_0)$, then $\Phi(U_0) \supseteq \Phi(U') \supseteq \Phi(\Psi(\Phi(U_0))) = \Phi(U_0)$, so $\Phi(U') = \Phi(U_0)$. This implies $d_{\mathsf{c}}(U', V') = |\Phi(U') \oplus \Phi(V')| = |\Phi(U_0) \oplus \Phi(U_1)| = d_{\mathsf{c}}(U_0, V_0)$. □

2 Frequent Sets of Items and Closures of Sets of Items

The notion of frequent sets of items is formulated starting from two finite, nonempty sets, a set of transactions T and a set of items I, and from a relation $R \subseteq T \times I$. The mappings of the polarity determined by the relation R are denoted by $\mathsf{ti}_R : \mathcal{P}(T) \longrightarrow \mathcal{P}(I)$ and $\mathsf{it}_R : \mathcal{P}(I) \longrightarrow \mathcal{P}(T)$. If R is clear from context, the subscript will be omitted.

The table τ associated to this Galois connection is a table whose heading is the set of items I. Each item, regarded as an attribute has the binary domain $\{0, 1\}$. If t is a transaction in T, then T is represented in the table τ by a tuple (denoted by the same letter) such that $t[i] = 1$ if and only if $(t, i) \in R$.

Definition 2.1 *Let* c *be a polarity on the set of transactions* T *and the set of items* I*. The* support of a set of items K*,* $K \subseteq I$*, is the number* $\mathtt{supp}_{\mathsf{c}}(K) = w_{\mathsf{c}}(K)/|T|$.

A set K *is* ϵ-frequent *if* $\mathtt{supp}_{\mathsf{c}}(K) \geq \epsilon$ *and is* ϵ-maximal *if it is* ϵ*-frequent and there is no* ϵ*-frequent set* L *such that* $K \subset L$.

The weight of a set of items K is given by

$$w_{\mathsf{c}}(K) = |\mathsf{ti}(K)| = |t \in T \mid t[k] = 1 \text{ for every } k \in K|.$$

In other words, the weight of the set of items K equals the number of transactions that are associated with every item $k \in K$.

In view of Proposition 1.3 we have

Theorem 2.2 *If* c *is a polarity on the set of transactions* T *and the set of items* I*, then for every two sets of items* $K_0, K_1 \subseteq I$ *we have:*

1. $\max\{\mathtt{supp}_{\mathsf{c}}(K_0), \mathtt{supp}_{\mathsf{c}}(K_1)\} \leq (d_{\mathsf{c}}(K_0, K_1) + p_{\mathsf{c}}(K_0, K_1))/|T|$;
2. $\mathtt{supp}_{\mathsf{c}}(K_0 \cup K_1) \leq \min\{\mathtt{supp}_{\mathsf{c}}(K_0), \mathtt{supp}_{\mathsf{c}}(K_1)\}$;
3. $d_{\mathsf{c}}(K_0, K_1) + p_{\mathsf{c}}(K_0, K_1) \leq \mathtt{supp}_{\mathsf{c}}(K_0 \cap K_1) \cdot |T|$;
4. $\mathtt{supp}_{\mathsf{c}}(K_0) + \mathtt{supp}_{\mathsf{c}}(K_1) = (d_{\mathsf{c}}(K_0, K_1) + 2p_{\mathsf{c}}(K_0, K_1)) \cdot |T|$;
5. $\mathtt{supp}_{\mathsf{c}}(K_0) = \mathtt{supp}_{\mathsf{c}}(\mathtt{cl}_{\mathsf{c}}(K_0))$;
6. $K_0 \subseteq K_1$ *implies* $\mathtt{supp}_{\mathsf{c}}(K_1) \leq \mathtt{supp}_{\mathsf{c}}(K_0)$.

Proof. The arguments for the first five parts follow immediately from Definition 2.1 and the corresponding parts of Proposition 1.3. The last part follows from Part (ii). □

Thus, if K_1 is an ϵ-frequent set of items and $K_0 \subseteq K_1$, then K_0 is also ϵ-frequent.

Corollary 2.3 *For every sets of items* K, H *such that* $K \subseteq H \subseteq \mathtt{cl}_{\mathsf{c}}(K)$ *we have* $\mathtt{supp}_{\mathsf{c}}(K) = \mathtt{supp}_{\mathsf{c}}(H)$.

Definition 2.4 *Let* c *be a polarity on the set of transactions* T *and the set of items* I *and let* $L \subseteq I$*. The* family of (ℓ, ϵ)-extended closures of L *relative to* c *is the collection of sets*

$$\mathtt{CL}_{\mathsf{c}}^{\ell,\epsilon}(L) = \{\mathtt{cl}_{\mathsf{c}}(L) \cup H \mid |H| = \ell, H \cap \mathtt{cl}_{\mathsf{c}}(L) = \emptyset, \text{ and } \mathtt{supp}(L \cup H) \geq \epsilon\}.$$

Note that

$$\mathtt{CL}_{\mathsf{c}}^{0,\epsilon}(L) = \begin{cases} \emptyset & \text{if } \mathtt{supp}(L) < \epsilon, \\ \{\mathtt{cl}_{\mathsf{c}}(L)\}, & \text{otherwise.} \end{cases}$$

Further,

$$\mathtt{CL}_{\mathsf{c}}^{1,\epsilon}(L) = \{\mathtt{cl}_{\mathsf{c}}(L) \cup \{i\} \mid i \notin \mathtt{cl}_{\mathsf{c}}(L) \text{ and } \mathtt{supp}(L \cup \{i\}) \geq \epsilon\}.$$

We refer to any member of $\mathtt{CL}_{\mathsf{c}}^{\ell,\epsilon}(L)$ as an (ℓ,ϵ)-*extended closure*. Unless stated otherwise we always work with $(1,\epsilon)$-extended closures; so, we will just refer to them as *extended closures*.

We will often refer to a set of items using the term itemset.

Definition 2.5 *An itemset I* cannot be extended by closure *if it is* $\mathtt{cl}$-closed *and its family of extended closures is $\emptyset$.*

To compute the family of $(1,\epsilon)$-extended closures for a set of items we need to define the k-matrix $M^{[k]}$ of the Galois connection c. Every row of this matrix corresponds to a set of items of cardinality k. We assume that the set of items is $I = \{i_0, \ldots, i_{n-1}\}$ and that the sets of items of length k are arranged in lexicographical order. The columns of the matrix $M^{[k]}$ correspond to the items of I. If C is a set of items such that $|C| = k$ and $i_p \in I$, then $M^{[k]}_{C,i_p}$, the element located in the C-row and p-th column is

$$M^{[k]}_{C,i_p} = p_{\mathsf{c}}(C, \{i_p\}) = |\{t \in T \mid t[C, i_p] = (1, \ldots, 1)\}|.$$

Note that the matrix M^1 is symmetric. Also, the largest value of the entries of the C-line of the matrix $M^{[k]}$ will be found in the columns that correspond to the members of C and possibly in other columns.

The closure and family of extended closures of a set of items K can be computed starting from the table τ associated to the Galois connection c, as shown in the next section.

Theorem 2.6 *Let $\mathsf{c} = (\mathsf{ti}_R, \mathsf{it}_R)$ be the polarity associated to the relation $R \subseteq T \times I$. We have $i \in \mathtt{cl}_{\mathsf{c}}(K)$ if and only if $M^{[k]}_{K,i} = \max\{M^{[k]}_{K,j} \mid j \in I\}$, where $k = |K|$.*

Proof. Note that $\max\{M^{[k]}_{K,j} \mid j \in I\} = \mathtt{supp}_{\mathsf{c}}(K)$. For any $i \in I$ such that $M^{[k]}_{K,i} = \mathtt{supp}_{\mathsf{c}}(K)$ and $i \notin K$ we have to prove that $i \in \mathtt{cl}_{\mathsf{c}}(K)$. This follows from the definition of the value $M^{[k]}_{K,i}$ which tells us that i appears in all transactions in which K appears so i belongs to the closure of the itemset K. □

Finding all ϵ-frequent sets of items for a specified ϵ is an important data mining problem as it represents the most computationally expensive step in finding association rules in a database [1]. In literature ([1], [2]) the value of ϵ is called minimum support and ϵ-frequent sets of items are called large itemsets.

In the following sections we present one algorithm (`Closure`) for finding all ϵ-frequent itemsets and two algorithms (`MaxClosure`, `AltMaxClosure`) for finding all maximal ϵ-frequent itemsets.

3 The Closure Algorithm

We use the notions of closure and extended closure to improve on the number of database scans done by the `Apriori` algorithm. Our algorithm is based on the following result:

Theorem 3.1 *Let $F_{\epsilon,k}$ be the collection of ϵ-frequent k-itemsets. Define*

$$\mathtt{CF}_{\epsilon,k} = \bigcup_{F \in F_{\epsilon,k}} \{\mathtt{cl}_{\mathtt{c}}(F)\} \cup \mathtt{CL}_{\mathtt{c}}^{1,\epsilon}(F).$$

Then, for every $G \in F_{\epsilon,k+1}$ there is a set $C \in \mathtt{CF}_{\epsilon,k}$ such that $G \subset C$.

Proof. Suppose $G \in F_{\epsilon,k+1}$; then, all subsets of G that have cardinality k are included in $F_{\epsilon,k}$. We can write $G = F \bigcup \{i\}$ where $F \in F_{\epsilon,k}$ and i is an item. We now have two cases:

1. If the item i appears in all transactions in which F appears, then $i \in \mathtt{cl}_{\mathtt{c}}(F)$ and since $\mathtt{cl}_{\mathtt{c}}(F)$ is one of the elements of $\mathtt{CF}_{\epsilon,k}$ it follows that G must be included in one of the elements of $\mathtt{CF}_{\epsilon,k}$.
2. If i does not appear in all transactions in which F appears, but it appears in a fraction of transactions greater than ϵ, then this will ensure that one $(1,\epsilon)$-extended closure of F will contain i which means that G must be included in one of the elements of $\mathtt{CF}_{\epsilon,k}$.

□

The algorithm based on this idea uses the matrix M presented before and works as follows:

Let `Maximal` be the collection of all ϵ-maximal itemsets.

```
Candidate = all 1 itemsets
KFrequent = empty
Maximal = empty

while (true) do
  initialize the elements of matrix M with 0;
  count the support for all elements of Candidate
        and update their corresponding rows in M;
  for all itemsets C in Candidate do
    if (C is not frequent)
      continue to next itemset;
    add C to KFrequent;
    compute closure of C in cl(C);
    compute all extended closures of C and add them to Maximal;
    if no extended closures were found
      add cl(C) to Maximal;
  od
  empty Candidate;
  generate new candidate itemsets in Candidate
                          based on KFrequent;
  if Candidate is empty
```

```
    break;
  empty KFrequent;
  for all itemsets L in Maximal
    mark as frequent all itemsets from Candidate
                        that are contained in L;
  for all itemsets C in Candidate
    if C is marked as frequent
      add C to KFrequent;
  empty Candidate;
  generate new candidate itemsets in Candidate
                           based on KFrequent;
  if Candidate is empty
    break;
  empty KFrequent;
od
return Maximal
```

Note that the `Closure` algorithm requires $(\lfloor \frac{n}{2} \rfloor + 1)$ scans of the database where n is the dimension of the longest frequent itemset.

4 The MaxClosure Algorithm

We wrote the `MaxClosure` algorithm in order to provide a fast algorithm for finding the maximal frequent sets. As mentioned in [3], finding all frequent sets is an extremely expensive computation for some databases, so it makes sense to have instead a fast algorithm that would allow one to know what the maximal itemsets are (which also means what the frequent itemsets are) without finding the support for all frequent candidates. Based on this knowledge, one could guide the mining process to extract only the rules one is interested in.

The `MaxClosure` algorithm is a specialization of the `Closure` algorithm. It uses a similar matrix structure for computing the closures and extended closures and it makes use of the fact that if at any point a frequent itemset has an empty extended closure, then that itemset is a maximal frequent itemset (it belongs to the positive border [6]). The matrix used in this algorithm differs from the one described before in the fact that its rows can correspond to itemsets of different cardinality.

Theorem 4.1 *If a frequent itemset F cannot be extended by closure then it is a maximal frequent itemset.*

Proof. Suppose that F cannot be extended by closure but is not a maximal frequent itemset. Then, there exists a frequent itemset G such that $F \subset G$ and $G - F \neq \emptyset$. Let i be an item such that $i \in G$ and $i \notin F$. Since G is frequent it folows that i is frequent and appears with F in a fraction of transactions greater than ϵ. Then i should either belong to the closure of F or to an extended closure of F and we have a contradiction since F could not be extended by closure. □

The algorithm's psudocode is presented below. It uses three collections: `Candidate` to keep the candidate maximal frequent itemsets, `Frequent` which is used for intermediate storage, and `Maximal` which contains the maximal frequent itemsets found at any point.

```
Candidate = all 1-itemsets;
Frequent = empty;
Maximal = empty;

while (true) do
  initialize the elements of matrix M with 0;
  count the support for all elements of Candidate
        and update their corresponding rows in M;
  for all itemsets C in Candidate do
    if C is not frequent
      continue to next itemset;
    compute closure of C in cl(C);
    for all extended closures xcl(C) of C
      if xcl(C) has not been already added to Frequent
        add xcl(C) to Frequent;
    if no extended closures were found
      if cl(C) has not been already added to Maximal
        add cl(C) to Maximal;
  od
  if Frequent is empty
    break;
  empty Candidate;
  move into Candidate all elements from Frequent;
od
return Maximal;
```

Note that the `MaxClosure` algorithm requires in the worst case $n-1$ scans of the database where n is the dimension of the longest frequent itemset. The reason is that in the worst case, a candidate itemset can be extended with only one item during one scan of the database so it will take $n-1$ scans of the database to generate an n-itemset.

5 An Extension of the MaxClosure Algorithm

The algorithm presented in this section is designed to guarantee fewer passes over the database, as compared to `MaxClosure`. `AltMaxClosure` requires $(\lfloor \frac{n}{2} \rfloor + 1)$ scans of the database where n is the dimension of the longest frequent itemset, the same number that `Closure` does.

`AltMaxClosure` is an extension of `MaxClosure`. The idea is that instead of starting with a candidate C, computing the extended closure of C and then using this extended closure as a candidate, we can try to extend even further the extended closure and use this extension as a candidate. This ensures that we will do half the number of passes required by `MaxClosure`.

The pseudocode for the algorithm is presented below. It uses one additional collection: `XPrevCandidate` to keep the extended closures generated from the previous candidate itemsets.

```
Candidate = all 1-itemsets;
Frequent = empty;
```

```
XPrevCandidate = empty;
Maximal = empty;

while (true) do
  initialize the elements of matrix M with 0;
  count the support for all elements of Candidate
            and update their corresponding rows in M;
  if XPrevCandidate is not empty do
    for all itemsets C in Candidate
      mark as not being maximal all itemsets from
           XPrevCandidate that are contained in C;
    for all itemsets L in Maximal
      mark as not being maximal all itemsets from
           XPrevCandidate that are contained in L;
    for all itemsets P in XPrevCandidate
      if P is marked as being maximal
        add P to Maximal;
  od
  empty XPrevCandidate;
  for all itemsets C in Candidate
    do
    if C is not frequent
      continue to next itemset;
    compute closure of C in cl(C);
    for all extended closures xcl(C) of C
      if xcl(C) has not been already added to XPrevCandidate
      do
        add xcl(C) to XPrevCandidate;
        for all items i not in xcl(C) such that M(C,i) >= epsilon
        do
          create itemset C' from the union of xcl(C) and i;
          if C' has not been already added to Frequent
            add C' to Frequent;
        od
      od
    if no extended closures were found
      if cl(C) has not been already added to Maximal
        add cl(C) to Maximal;
  od
  if Frequent is empty
    break;
  empty Candidate;
  move into Candidate all elements from Frequent;
od
return Maximal;
```

Given an extended closure of a candidate C, we try to extend it by looking at all items that appear with C in a number of transactions greater or equal to ϵ. We also store xcl(C) in XPrevCandidate in the eventuality that none of these extensions prove to be frequent which would mean that xcl(C) can not be further extended so it is maximal. A new step has been introduced for verifying whether any of the elements of XPrevCandidate are maximal.

6 Experimental results

We implemented `Apriori`, `Closure`, `MaxClosure`, and `AltMaxClosure` in C++. We have used the hash-tree structure introduced in [2] in order to efficiently test for itemset inclusions. The data was generated using the IBM Quest synthetic data generator and was stored in binary files. As observed by other researchers [7], when working with files the bottleneck is in the CPU computations, not in accessing the disk. For this reason, the difference between `Closure` and `Apriori` is not as big as one would expect. In a previous experiment in which we ran a preliminary version of the `Apriori` and `Closure` algorithms using Oracle databases, the results showed a 50% improvement in time on behalf of the `Closure` algorithm.

Our first experiments used databases with an average number of items per transaction equal to 10, and a total of 100 items. We have generated a 50 thousand, a 100 thousand and a 1 million rows database in order to check for the scalability of each algorithm. The results of the experiments, presented in Figure 1, show clearly that all algorithms scale linearly with the size of the database.

The graphs in Figure 2 show that at reasonable levels of support, our algorithms outperform the `Apriori` algorithm.

Similar results hold for `AltMaxClosure` when compared to `Apriori`, as shown in Figure 3.

On the other hand, under our current implementations, which are based on raw files rather than on relational databases, `AltMaxClosure` is always slower than `MaxClosure`, despite the fact that it does fewer passes over the data. The reason is that `AltMaxClosure` has to do several equality and inclusion tests in order to ensure that no duplicates are added to its various collections and those operations prove to be quite expensive.

In Figure 4 we present the results obtained when we ran the algorithms on a 100,000 thousand rows database having an average of 25 items per transaction.

In this experiment `Closure` performs slightly worse than `Apriori` because it does more computations which are not compensated by the decreased number of passes over the database. `MaxClosure` continues to perform better than the other two although the improvement comes from counting the support for fewer itemsets rather than from doing fewer scans over the database. For 5% minimum support the algorithms discover 16949 maximal frequent itemsets.

We also did a test on a database with 100,000 transactions and an average of 50 items per transaction. We ran the algorithms for 50% minimum support and we obtained 2243 maximal frequent itemsets:

	`Apriori`	`Closure`	`MaxClosure`	`AltMaxClosure`
Time	1573	2905	3169	4500
Database passes	8	4	7	4

We conclude that as the average number of items per transaction increases, the performance of the `Closure` algorithm decreases. `MaxClosure` performs its worst

Support %	Time in seconds			
	`Apriori`	`Closure`	`MaxClosure`	`AltMaxClosure`
50 thousand records				
1	105	100	16	195
2	29	19	6	33
3	16	4	3	9
5	11	1	1	2
10	4	1	1	1
15	1	0	0	0
20	0	0	0	0
100 thousand records				
1	201	186	33	348
2	58	38	12	64
3	33	9	6	18
5	22	3	3	4
10	8	2	2	2
15	2	1	1	1
20	1	1	1	1
1 million records				
1	2033	1874	328	3139
2	573	367	123	611
3	324	92	66	180
5	226	34	37	45
10	85	19	20	19
15	24	16	16	16
20	15	16	16	16

Figure 1: Performance of Algorithms

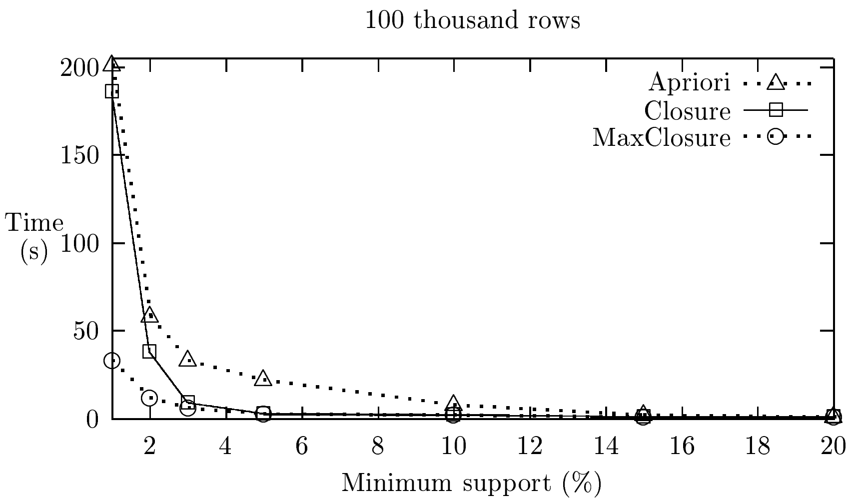


Figure 2: Performance Graphs for Apriori, Closure, and MaxClosure

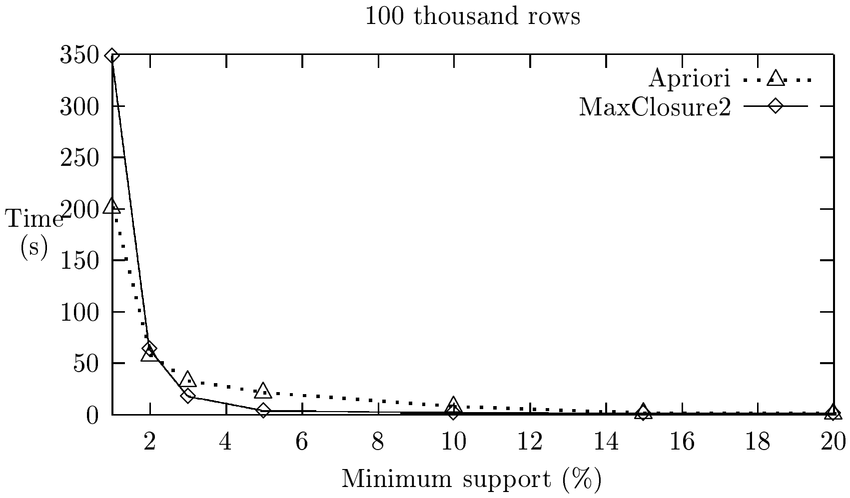


Figure 3: Performance Graphs for Apriori and AltMaxClosure

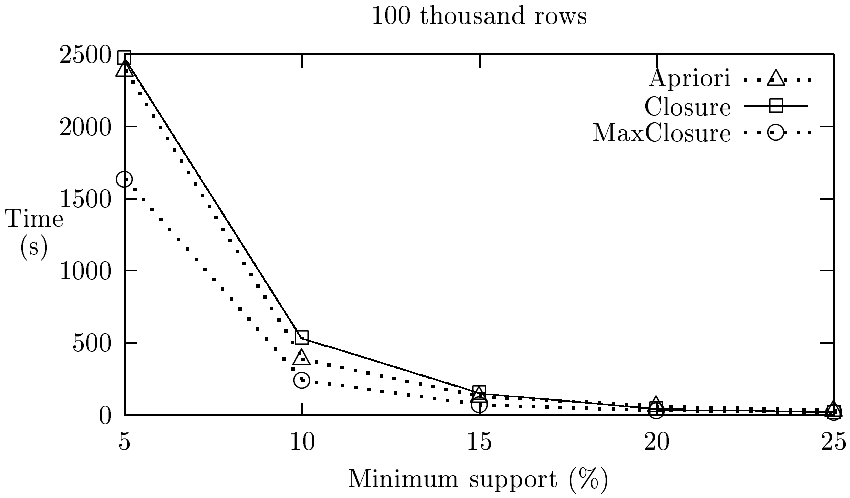
case number of passes but it comes pretty close to `Closure` because it does not compute all frequent itemsets.

7 Conclusions

We introduced three new algorithms: `Closure`, `MaxClosure` and `AltMaxClosure` that make use of Galois connections. The experiments we have done show that `Closure` and `MaxClosure` perform faster than the standard `Apriori` algorithm when running on databases with an average number of items per transaction of less than 25% of the total number of items. For higher averages (like 50%), our algorithms are outperformed by `Apriori`. However, if database access is expensive, `Closure` might still perform better than `Apriori` due to the fewer number of passes over the database it requires. `AltMaxClosure` may be an alternative for `MaxClosure` when database access is expensive; however, its performance decreases very fast as the minimum support takes smaller values.

References

1. Agrawal R., Imielinski T., Swami A.:*Mining Association rules between Sets of Items in Large Databases*, SIGMOD 1993, pages 207-216.
2. Agrawal R., Srikant R.:*Fast Algorithms for mining association rules*, VLDB 1994.
3. Bayardo R. J.:*Efficiently Mining Long Patterns from Databases*, SIGMOD 1998, pages 85-93.
4. Birkhoff G.:*Lattice Theory*, American Mathematical Society, Colloquium Publications, 3rd edition, 1973.
5. Gierz G.*et al.*:*A Compendium of Continuous Lattices*, Springer-Verlag, Berlin, 1980.



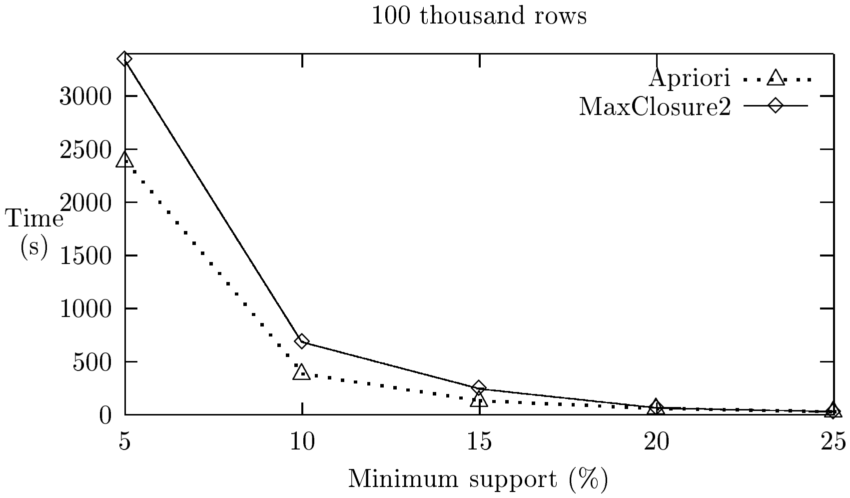


Figure 4: Performance Graphs at 25 Items per Transaction

6. Gunopulos D., Khardon R., Mannila H., Toivonen H.:*Data mining, Hypergraph Transversals, and Machine Learning*, PODS 1997, pages 209-216.
7. Mueller A.:*Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison*, CS-TR-3515, University of Maryland-College Park, 1995.