Interactive Mining and Knowledge Reuse for the Closed-Itemset Incremental-Mining problem

Luminita Dumitriu
Department of Computer Science and Engineering, "Dunarea de Jos" University,
str. Domneasca nr. 47,
Galati 6200, Romania.

Luminita.Dumitriu@ugal.ro

ABSTRACT

Using concept lattices as a theoretical background for finding association rules [11] has led to designing algorithms like Charm [10], Close [7] or Closet [8]. While they are considered as extremely appropriate when finding concepts for association rules, due to the smaller amount of results, they do not cover a certain area of significant results, namely the pseudo-intents that form the base for global implications. We have proposed an approach that, besides finding all proper partial implications, also finds the pseudo-intents. The way our algorithm is devised, it allows certain important operations on concept lattices, like adding or extracting items, meaning we can reuse previously found results. It is a well-known fact that mining association rules may lead to a large amount of results. Since, the mining results are meant to be understood by the user, we have come to the conclusion that he will benefit more from starting small, with some of the items in the data base, understand a small amount of results, and then add items receiving only the extra-results. This way the number of human interventions during the "full" mining process is increased and the process becomes user-driven.

Keywords

Frequent itemsets, association rules, formal concept analysis.

1. INTRODUCTION

Lately, corporations have made serious investments into using information technology to increase their management quality. Also, large amounts of important business data are now stored in database systems and their quantum is expected to grow. An important goal of data mining is to extract valuable implicit patterns from large quantities of data.

Association rules, introduced by [1], provide useful means to discover associations in data. Let $\boldsymbol{I} = \{i_1, i_2, \dots, i_m\}$ be a set of m literals called *items*. Let the database $\boldsymbol{D} = \{t_1, t_2, \dots, t_n\}$ be a set of n transactions, each transaction consisting of a set of items I of $\boldsymbol{I}$ and associated with a unique transaction identifier called *tid*. I is called a *k-itemset* if k is the size of I. A transaction $t \in \boldsymbol{D}$ is said to contain an itemset I if $I \subseteq t$. The *support* of an itemset I is the percentage of transactions in $\boldsymbol{D}$ containing I: $support(I) = |\{t \in \boldsymbol{D} \mid I \subseteq t\}| / |\{t \in \boldsymbol{D}\}|$. An association rule is a conditional implication among itemsets, $I \Rightarrow I'$, where itemsets $I, I' \subset \boldsymbol{I}$ 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 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 is defined as finding all the association rules that hold with more that a user-given support threshold, *minsup*, and a user-given confidence threshold, *minconf*. According to [1] this problem is solved in two steps:

1. Finding all *frequent* itemsets in the database, meaning itemsets with support higher than or equal to *minsup*.
2. For each frequent itemset I, generating all association rules $I' \Rightarrow I \setminus I'$, where $I' \subset I$, with confidence greater than or equal to *minconf*.

The second problem can be solved in a straightforward manner after the first step is completed. Hence, the problem of mining association rules is reduced to the problem of finding all frequent itemsets. This is not a trivial problem, since the number of possible frequent itemset is equal to the size of the power set of $\boldsymbol{I}$, $2^{|\boldsymbol{I}|}$. There are many algorithms proposed in the literature, most of them based on the Apriori mining method [2], that relies on a basic property of frequent itemsets: *all subsets of a frequent itemset are frequent*. This property can also be said as *all supersets of an infrequent itemset are infrequent*. This approach works well on weakly correlated data such as market basket data. Over correlated data, such as census data, there are other approach, as Close [7], CHARM [10] and Closet [8], which are more appropriate. These approaches search for closed itemsets, structured in lattices that are closely related with the concept lattice in formal concept analysis [9]. The main advantage of a closed itemset approach is the smaller size of the resulting concept lattice versus the number of frequent itemsets, *i.e.* a search space reduction.

In this paper, we propose a new approach that is also a closed itemset approach, but is more user-oriented; it takes into account the fact that an association rule mining process leads to a large amount of results, that is, most of the time, difficult to understand by the user. To prevent this to happen, we add a user-selection to the process: the interesting frequent items. This way we build a partial, easier to understand, model of the data. We still have to provide means to model all the data. To achieve this goal, our approach allows the user to pick a previously found partial data model (expressed as a concept lattice and a data context) and some new frequent items to be added to it, thus obtaining an extended or even a full data model. The results of an extension operation consist of the supplementary results, the *δ-model*. The reverse operation to the extension of a model is reducing a model with some frequent items; we have considered it to be used whenever a large data model is incomprehensible to the user. The main advantages of our approach are:

- a smaller size of results leads to a higher comprehensibility of a data model or a δ-model;
- the data models can be reused, when extending or reducing them with sets of items, thus sparing the time spent building them;
- the mining process becomes more user-driven than data-driven.

We have also considered the data model to be extended with the base for generating rules with confidence = 1, the *pseudo-intents*.

The rest of the paper is organized as follows. Section 2 reviews related work and compares it with the contribution of the paper. Section 3 describes the new mining process. In section 4 we give some experimental results. The paper is concluded in section 5.

2. RELATED WORK AND CONTRIBUTIONS

In this section we first describe the use of Apriori-type approaches and the advantage of using closed itemset lattices as theoretical framework for the closed itemset-type approach. In the end, we briefly describe our method.

2.1 Apriori and closed itemset approaches

Let's consider the example database in Figure 1.

Generating all possibly frequent itemsets and testing their support is clearly an impracticable solution when *m*, the cardinal of ***I***, is large. The Apriori methodology is described as follows.

First, the items in ***I*** are sorted in lexicographic order. The frequent items are computed in one pass over the database. They are stored in the set L_1 of frequent 1-itemsets. For each iteration *i*, a candidate set, C_i, is computed joining L_{i-1} with itself. In a database scan the support for each candidate in C_i is computed. The frequent itemsets found in C_i are stored in L_i, the set of frequent *i*-itemsets, and C_i is discarded. The process continues until all candidates are infrequent.

The join operation in iteration *i* selects pairs of *i-1*-itemsets having *i-2* items in common, reuniting their elements into a candidate of size *i*. Afterwards, the candidates that do not have all their subsets of *i-1* items in L_{i-1} are pruned.

The number of effective iterations is equal to the height of the frequent itemset lattice.

Let's see how Apriori does on the example database in Figure 2.

Iteration 1:

- database scan results into L_1 = { A, B, C, D, E} with *support*(A) = 6, *support* (B) = *support*(D) = *support*(E) = 4 and *support*(C) = 5.

Iteration 2:

- join phase $L_1 * L_1$ = {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}
- no pruning, $C_2 = L_1 * L_1$
- database scan results into L_2 = {AB, AC, AD, AE, BC, BD, CD, CE}, where the support of AC is 5, the support of AB, AD, AE, BC is 4 and the support of BD, CE, CD is 3.

Iteration 3:

- join phase $L_2 * L_2$ = {ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE, CDE}
- pruning ABE, ADE, BCE, BDE, CDE, C_3={ABC, ABD, ACD, ACE, BCD}
- database scan results into L_3 = { ABC, ABD, ACD, ACE, BCD }, where the support of ABC is 4 and the support of ABD, BCD, ACE, ACD is 3.

Iteration 4:

- join phase $L_3 * L_3$ = {ABCD, ABCE, ACDE }
- pruning ABCE, ACDE, C_4={ABCD }
- database scan results into L_4= { ABCD}, where the support of ABCD is 3.

Iteration 5:

- join phase $L_4 * L_4 = \emptyset$.
- no pruning, $C_4 = \emptyset$.

In the end, Apriori finds the 19 frequent itemsets we can see in Figure 1.

The closed itemset approach is described in the following.

We define a context, the Galois connection of a context, a concept of the context, a lattice of concepts. For a more details on lattice theory see [9].

A *context* is a triple (***T***, ***I***, ***D***) where ***T*** and ***I*** are sets and $\boldsymbol{D} \subseteq \boldsymbol{T} \times \boldsymbol{I}$. The elements of ***T*** are called *objects* and the elements of ***I*** are called *attributes*. For any $t \in \boldsymbol{T}$ and $i \in \boldsymbol{I}$, we note $t\boldsymbol{D}i$ when *t* is

Database items

Bread	Butter	Milk	Jam	Cereals
A	B	C	D	E

Transaction	Items
1	A B C D
2	A C E
3	A B C D
4	A B C E
5	A B C D E
6	A D E

All frequent itemsets, for minsup=50%

Itemsets	Support
A	100%;
C, AC	83%;
B, D, E, AB, AD, AE, BC, ABC	67%;
BD, CE, CD, ABD, BCD, ACE, ACD, ABCD	50%.

Figure 1: Example database and associated frequent itemsets.

related to *i*, *i.e.* (*t*, *i*) ∈ ***D***.

Let (***T***, ***I***, ***D***) be a context. Then the mappings

$$s: \wp(\boldsymbol{T}) \rightarrow \wp(\boldsymbol{I}),\ s(X) = \{\ i \in \boldsymbol{I} \mid (\forall t \in X)\ t\boldsymbol{D}i\ \}$$

$$t: \wp(\boldsymbol{I}) \rightarrow \wp(\boldsymbol{T}),\ s(Y) = \{\ t \in \boldsymbol{T} \mid (\forall i \in Y)\ t\boldsymbol{D}i\ \}$$

define a *Galois connection* between $\wp(\boldsymbol{T})$ and $\wp(\boldsymbol{I})$, the power sets of ***T*** and ***I***, respectively.

A concept in the context (***T***, ***I***, ***D***) is a pair (X, Y), where $X \subseteq \boldsymbol{T}$, $Y \subseteq \boldsymbol{I}$, s(X)=Y and t(Y)=X. X is called the *extent* and Y the *intent* of the concept (X,Y). This leads to s°t(Y) = s(X) =Y and t°s(X) = t(Y)=X being closure operators.

A concept (X', Y') is a *subconcept* of (X,Y), denoted (X',Y') ≤ (X,Y), iff $X' \subseteq X$ (iff $Y' \supseteq Y$).

Let ***C*** be the set of concepts derived from ***D*** using the Galois connection. The pair *L*=(***C***, ≤) is a complete lattice called a lattice of concepts.

The closed itemset approach uses from the concepts only their intent. Some of the frequent itemsets found in the previous section are not closed itemsets. For example, the itemset ABD, where t(ABD) = { 1, 3, 5}, but s({1, 3, 5})=ABCD. The frequent closed itemsets for the example database in Figure 2 are ***C*** = {A, AC, AD, AE, ABC, ACE, ABCD}. The lattice *L* is represented in Figure 2.

As we can observe, there are 7 frequent closed itemsets and 19 frequent itemsets. Ignoring the itemsets that are not closed is a significant pruning criterion for the search space. A closed itemset in the context of mining association rules is the maximal itemset of a collection of itemsets that share the same transaction set.

We will present the CHARM method for finding closed itemsets. First, the frequent items are found, along with their transaction sets. In the order defined on ***I***, the first frequent item i_1 is picked. The algorithm attempts extending this item with i_2. The support of the new itemset is computed while intersecting the transaction sets

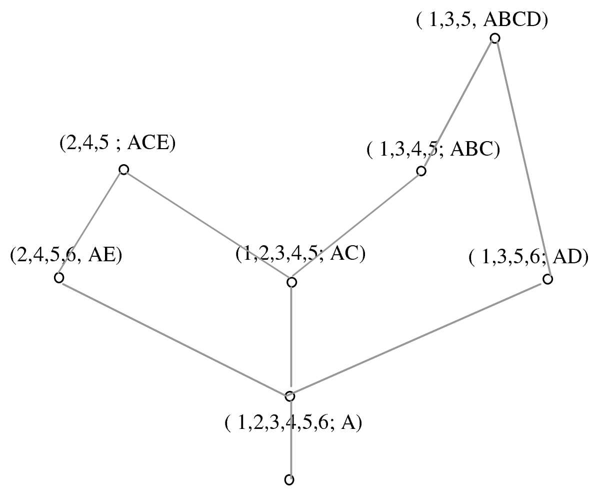

Figure 2: The closed-itemset lattice for the example database

of the two items. There are 4 cases to take into account:

- if the new itemset is infrequent, i_1 is extended with i_3, continuing in a depth-first manner;
- if the transaction set of the new itemset is equal to the one of i_1, then i_1 is replaced with the new itemset, and we continue extending the new itemset with the next frequent item;
- if the transaction set of the new itemset is equal to the one of i_2, then i_2 is discarded and the new itemset is added as a child of i_1, and we continue extending the new itemset with the next frequent item;
- otherwise, the new itemset is added as a child of i_1, and we continue extending it with the next frequent item;

When there are no more items to extend the current itemset with, the algorithm picks the next itemset in a depth first manner.

We will demonstrate how CHARM works on the example database, considering the lexicographic order.

Step 1. CurrentNode = root;

children(CurrentNode) = { (A; 1,2,3,4,5,6), (B; 1,3,4,5), (C; 1,2,3,4,5), (D; 1,3,5,6), (E; 2,4,5,6)}

Step 2. CurrentNode= A, ItemToExtendWith= B, NewItem = (AB; 1,3,4,5) - same as B

children(root) = { (A; 1,2,3,4,5,6), (C; 1,2,3,4,5), (D; 1,3,5,6), (E; 2,4,5,6)};

children(A) ={(AB; 1,3,4,5)}

Step 3. CurrentNode= AB, ItemToExtendWith= C, NewItem = (ABC; 1,3,4,5) - same as AB

children(A) ={(ABC; 1,3,4,5)}

Step 4. CurrentNode= ABC, ItemToExtendWith= D, NewItem = (ABCD; 1,3,5) - different

children(ABC) ={(ABCD; 1,3,5)}

Step 5. CurrentNode= ABCD, ItemToExtendWith= E, NewItem = (ABCDE; 5) - infrequent

Step 6. CurrentNode= ABC, ItemToExtendWith= E, NewItem = (ABCE; 5) - infrequent

Step 7. CurrentNode= A, ItemToExtendWith= C, NewItem = (AC; 1,2,3,4,5) - same as C

children(root) = { (A; 1,2,3,4,5,6), (D; 1,3,5,6), (E; 2,4,5,6)};

children(A) ={(ABC; 1,3,4,5), (AC; 1,2,3,4,5) }

Step 8. CurrentNode= AC, ItemToExtendWith= D, NewItem = (ACD; 1,3,5) - different

children(AC) ={(ACD; 1,3,5)}

Step 9. CurrentNode= ACD, ItemToExtendWith= E, NewItem = (ACDE; 3,5) - infrequent

Step 10. CurrentNode= AC, ItemToExtendWith= E, NewItem = (ACE; 2,4,5) - different

children(AC) ={(ACD; 1,3,5), (ACE; 2,4,5)}

Step 11. CurrentNode= A, ItemToExtendWith= D, NewItem = (AD; 1,3, 5, 6) - same as D children(root) = { (A; 1,2,3,4,5,6), (E; 2,4,5,6)};

children(A) ={(ABC; 1,3,4,5), (AC; 1,2,3,4,5) (AD; 1,3,5, 6)}

Step 12. CurrentNode= AD, ItemToExtendWith= E, NewItem = (ADE; 5, 6) - infrequent

Step 13. CurrentNode= A, ItemToExtendWith= E, NewItem = (AE; 2, 4, 5, 6) - same as E

children(root) = { (A; 1,2,3,4,5,6)}

children(A) ={(ABC; 1,3,4,5), (AC; 1,2,3,4,5) (AD; 1,3,5, 6), (AE; 2, 4, 5, 6) }.

The algorithm ends, finding the following closed itemsets: (A; 1,2,3,4,5,6), (AC; 1,2,3,4,5) (AD; 1,3,5, 6), (AE; 2, 4, 5, 6), (ABC; 1,3,4,5), (ACE; 2,4,5), (ABCD; 1,3,5), as in Figure 2. The (ACD; 1,3,5) element is discarded, since ABCD is the closed itemset for this transaction set. The CHARM algorithm does not determine the association rules.

2.2 Our approach

The common part of our approach with CHARM, for example, is the extension idea. The difference is that we extend a lattice built on a subset of ***I*** with some new frequent items. Other approaches, for example the one described in [5], that are operating on concept lattices are row-by-row oriented

The basic observation for our approach relies on a property of a non closed itemset. An itemset *i* is not a closed itemset, because there is a closed itemset *c*, where $i \subset c$ and t(*i*) = t(*c*). If we select from t(*c*) the subset of transactions containing some itemset *i'*, where *i'* is disjoint with *c*, we find that the resulting transaction set is one and the same for both the $i \cup i'$ and $c \cup i'$ itemsets. Thus, $i \cup i'$ will never be a closed itemset, any *i'* disjoint with *c*. We will call this property the *non-closure up-propagation*. This means that no closed itemset can originate in a non-closed itemset through extension with a new item. Thus, extending all closed itemsets in the lattice built on a subset of ***I*** with some new frequent items, we will obtain a cover set for all the closed itemsets involving the new frequent items.

We consider the basic operation of extending the closed itemset lattice with one frequent item at a time.

Lets say we already have the lattice for {A, B, C} and we want to extend it with {D, E}. As we can see in Figure 3, we practically make a copy of the initial lattice with new, extended itemsets. We eliminate itemsets D and ACD, since they are not closed, and we have the lattice for the { A, B, C, D} set of items. We extend it with E. There are only a few frequent itemsets from the extended ones and the itemset E is not closed because of AE. After eliminating E, we obtain the same frequent closed itemset lattice as CHARM does.

Until now, the main difference between our approach and CHARM is that we can start from a previously found lattice.

Now, we can find all rules with confidence smaller than 1. The association rules with confidence equal to 1 can be expressed through a base from which every rule can be generated. As shown in [4]: *The set* $\{X \rightarrow c(X) \setminus X \mid X$ *is a pseudo-intent*$\}$ *is a base for all global implications, where c is the closure operator, and X is a pseudo-intent if* $X \neq c(X)$, *and for all pseudo-intents* $Q \subset X$, $c(Q) \subseteq X$.

The fact that $X \neq c(X)$, tells us that X is not a closed itemset. For the initial lattice built on {A, B, C} we have 3 itemsets that are frequent, but not closed: {B, C, BC}. Since c(B) = ABC, c(C)=AC and they do not include any other itemset, we can say that B and C are pseudo-intents, while BC is not, since $B \subset BC$, but $ABC \not\subset BC$. Thus, the base for global implications consists of $B \rightarrow AC$ and $C \rightarrow A$ rules of confidence 1. What happens when we extend the initial lattice with D is that we find some new potential pseudo-intents {D, ACD} and after extending with E, we add E to this set, too. We have c(D)=AD, c(ACD)=ABCD and c(E)=AE. All of them are pseudo-intents in the context of the new lattice and the old ones remain pseudo-intents as well.

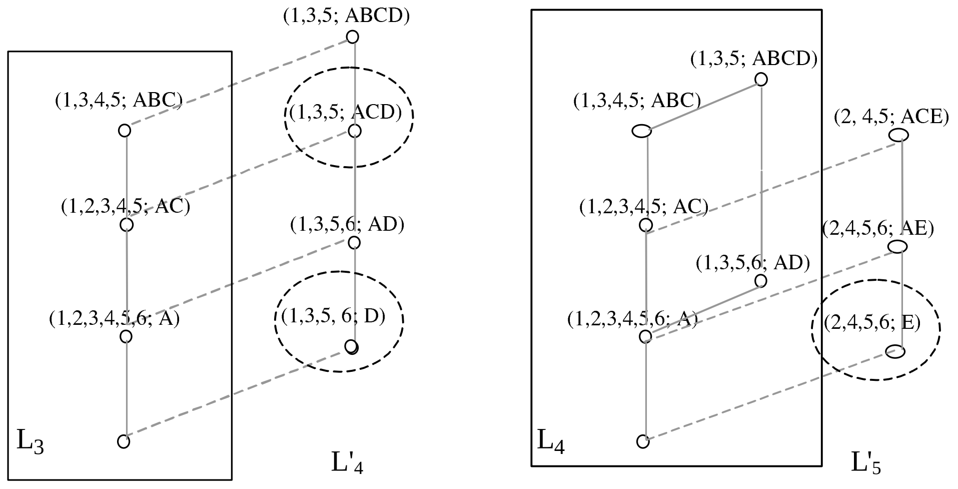

Figure 3: Extending the previously found lattice with frequent items D and E

In fact, the situation is not always this simple. Let's consider another initial lattice, namely the one built on {B, C, D, E}, and let's extend it with the A item. As we can see in Figure 4, the initial lattice contains the frequent closed sets {C, D, E, BC, CE, BCD}. The frequent itemsets that are not closed are B and CD. For this lattice we have c(B)=BC and c(CD)=BCD, both of them being pseudo-intents.

When we extend the lattice with A, the itemsets C, D and E become not closed, due to AC, AD and AE, respectively, and this relationship propagates over their subconcepts when extending them with A. Thus, all the previously closed itemsets become not closed. We can prove that storing only the rules C→A, D→A and E→A, called *generating rules*, along with the old pseudo-intents, we can calculate the pseudo-intents for the final lattice. The list of corresponding closed itemsets is: c(B)=ABC, c(CD)=ABCD, c(C)=AC, c(D)=AD, c(E)=AE. The 1-frequent itemsets in the list are pseudo-intents since there are no other pseudo-intents to cancel their status. CD is not a pseudo-intent since there are C and D to contradict it. If we add to CD the missing items from c(C), namely c(C)\C = A, we get ACD, that is still an itemset that is not closed, belongs to the same closed itemset as CD, and respects the definition of a pseudo-intent. If the newly constructed itemset was a closed itemset, we could not calculate a pseudo-intent starting from that itemset.

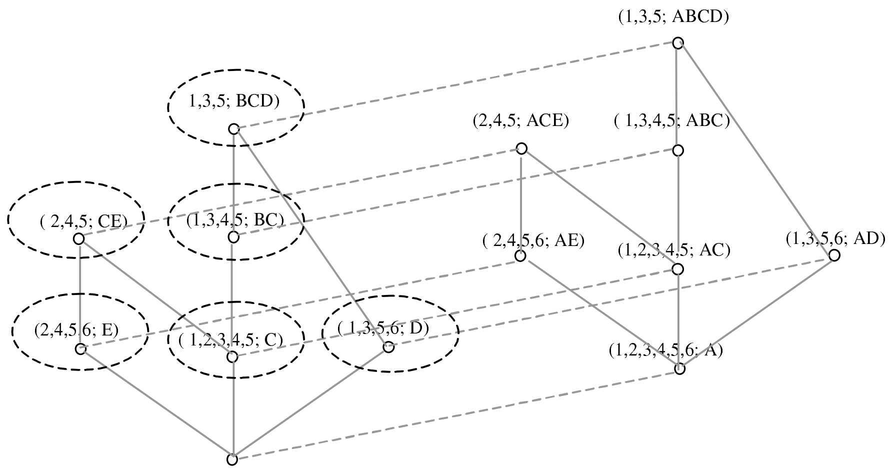

Figure 4: Extending a previously found lattice with the frequent item A

An important observation is referring to situations as the one in Figure 4. If we find a rule with confidence equal to 1, for all the supersets of the itemset that becomes not closed, we don't need to calculate the support of their extended itemsets, since they are equal to one another. Thus, *confidence*(C→A) = 1 leads to *confidence*(BC→A) = 1, *confidence*(CE→A) = 1, *confidence*(BCD→A) = 1, so *support*(C) = *support*(AC), *support*(BC) = *support*(ABC), *support*(CE) = *support*(ACE). We consider this situation, whenever it occurs, as a significant pruning criterion to calculating the support of an extended itemset.

Another operation we have considered is reducing a lattice of closed itemsets with some items. The only way the initial lattice is affected during the extension operation is as in Figure 4, when the original closed itemset becomes not closed and disappears. Keeping this observation in mind, we conclude that, for a closed itemset involving an item to be eliminated, we have to check the existence of the reduced itemset and, if it does not exist, we create it with the same support. If we keep track of the base of global implications we deal with two steps:

- we eliminate the current item from the left-side or from right-side of the rule, if it exists;
- if either the left-side or the right-side of the rule becomes empty, the rule is discarded.

2.3 Implementing extension and reduction of concept lattices

2.3.1 Representation

In our approach, one concept is represented by an index to its contents, that consists of the concept intent, the support information and the list of indices to its adjacent subconcepts.

The concept indices are numbers. When stored in a data model the numbers are consecutive. The itemsets generated when extending a data model are numbered corresponding to their occurrence, starting from the number of concepts in the data model. One major advantage of this numbering is the possibility to quickly identify the new itemsets from the old concepts.

The list of adjacent subconcepts of a concept is needed in order to find all non-concepts, while extending the model. The lists are updated when removing non-concepts from the newly generated itemsets. The lists also represent association rules, when *minconf* is exceeded.

Whenever it is the case and definitely when saving a model, the concepts are renumbered, to consecutive indices and their adjacent subconcept lists are updated.

2.3.2 Pseudo-intents and generating rules

A pseudo-intent is represented as a pair of its description and the index of its associated concept. We do not have to store all global implication rules, only the so-called *generating rules* are needed in order to compute the pseudo-intents. There are two kinds of global implication rules that can lead to generating rules, ones between concepts and their extended itemset, as C→A in Figure 4, and ones between new itemsets as E→A in Figure 3. A global implication rule X→X'\X, found while extending a data model, is a generating rule if any other implication rule Y→ Y'\Y found, with Y⊂X, has Y'⊂X.

In the case of Figure 4, C→A is a generating rule and all global implications concerning the subconcepts of C are not, because AC cannot be included in any of C subconcepts in the original lattice. We have proved, for this kind of generating rules, that all the global implications concerning the subconcepts of the antecedent cannot be pseudo-intents.

In the case of Figure 3, when extending the concept lattice with D, we have two global implications: D→A and ACD→B. They are both generating rules because both D and AD are included in ACD. We have proved, for this kind of rules, X→X', that all global implications concerning super-itemsets of X, that are not super-itemsets of X∪X', can not be pseudo-intents.

We have also proved that any pseudo-intent is either the antecedent of a generating rule or can be computed from it.

2.3.3 Extending a data model

We will consider extending a data model with a frequent item as a basic operation. If several items are involved we, step-by-step, extend the model with each of the items.

First, we build all new frequent itemsets, extending original concepts with the current item. We start with the empty concept in a queue of concepts, and while it is not empty, we extend the head of the queue; if the new itemset is frequent, we number and store it along with the original concept index and we place its adjacent subconcepts in the queue.

When we extend the current concept in the queue, we check the partial list of generating rules corresponding to the extension in process if the concept includes one of the antecedents in the list; in this case, we will have a non-closure up-propagation, and we don't need to compute the support information for the new itemset, because it is the same with the concept's support. If it is not the case, we compute the support information and, if frequent, we check for a new generating rule case. If so, we record the generating rule in the current list.

Computation of support information can be done as in CHARM or in Closet, depending on the nature of the database, sparse or dense. The transactions that count for the support computation are only the ones concerning the items we extend the model with.

Afterwards, for all new itemsets we build adjacent super-itemset lists, based on the itemsets extended from the adjacent subconcepts of the original concept, as in Figure 5. We also check the new itemsets for being non-concepts, and we store generating rules, if needed.

All concepts and new itemsets found as non-concepts are marked when found. For any itemset or concept that is not marked we build the correct adjacent subconcept list (consisting of unmarked subconcepts checked for adjacency) and we remove all non-concepts.

In the end, we store the partial list of generating rules along with the original pseudo-intents.

When extension is completed, we compute the pseudo-intents associated to the final data model.

2.3.4 Reducing a data model

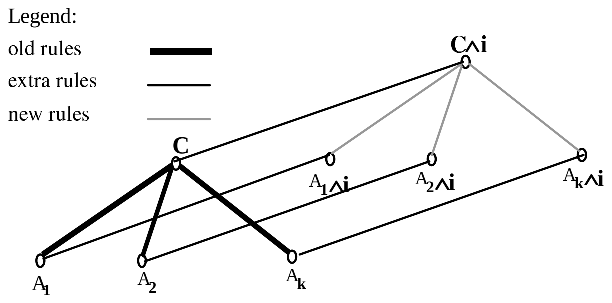

Figure 5: New concepts and new rules when extending a data model

Reducing a data model is much simpler than extending one. We will also consider reducing a model with one item, *i*. For all concepts in the model that include the item, we check if the reduced itemset is a concept in the current model or not. If it is, we simply remove those concepts. If it is not, we keep all the concept information, including the index, except for the intent, replacing it with the reduced itemset. The reason behind this decision is that, when extending with the same item, only the reduced itemset would lead to the generation of that concept, thus the reduced itemset had to be a concept. The lists of adjacent subconcepts remain the same, because if the reduced itemset became a non-concept while extending, all its subconcepts became non-concepts, so they will all be retrieved as this one.

For all pseudo-intents or associated concepts that contain *i*, we remove the item. If the antecedent or the consequent of the global implication become empty, we discard the pseudo-intent. If the reduced antecedent is also concept, we have to check if itemsets that include the reduced antecedent and are include by the reduced associated concept can be pseudo-intents.

2.4 Defining and operating with data models

For our approach, a data model for the association rule problem consists of three components:

– the mining context;

– the corresponding results;

– the observations added by user.

There is no question on the necessity of the results in the data model. The only issue is whether we keep track of the pseudo-intents or not. We represent the results as the set of closed itemsets and their associated support information and, when needed, the set of pseudo-intents and their associated closed itemsets.

The mining context stores information on the set of frequent items and the transactions involved in building the model. We also have to store the *minsup* value; otherwise the results have no relevance.

The user added information is optional and it is meant to record his remarks on the contents of the data model, as a remainder.

When choosing a data model to be extended, we have to check the mining context in order to refer the same transactions, with at least *minsup* for the support threshold, while extending the model with new frequent items. If we look for pseudo-intents we can only choose a previously found model containing associated pseudo-intents. The *minconf* parameter does not affect choosing the model, since we store the set of frequent closed itemsets, not the association rules.

The correctness of the data model extension can be proved as follows:

P_1. We extend the lattice L_k with i_{k+1}, obtaining L'_{k+1}. We say that $L_k \cup L'_{k+1} \supset L_{k+1}$.

P_2. We eliminate non-closed itemsets from $L_k \cup L'_{k+1}$. We say the result is L_{k+1}. If we keep track of pseudo-intents, we add to their set all generating rules found during the elimination phase.

P_3. Processing the set of potential pseudo-intents, we can calculate all pseudo-intents associated to the lattice of closed itemsets.

The correctness of the data model reduction can be proved as follows:

P'_1. We eliminate from L_k all closed itemsets involving i_k; when the original closed itemset does not exist, we create it with the same support as the extended one. We say that the result is L_{k-1}.

P'_2. If we keep track of pseudo-intents, we eliminate i_k from the left-side or from the right-side of the global implication; whenever a rule has an empty antecedent or consequent we discard it. We say this new set is the base for global implications associated with L_{k-1}.

We have mentioned in the first section the δ–model. This kind of model is used for result displaying reasons only. Whenever we extend a previously found model we display to the user only the differences between the initial model and the final one, in order to reduce the amount of presented results to the new ones. We have not yet dealt with presenting to the user the closed itemsets that become not closed during the extension.

The way we assume the data model processing will be useful is as follows:

– first, the user can mine a few items, getting a fast response and a small amount of results;

– if the user is satisfied by a certain data model he can store it and extend it, afterwards, with some other items;

– if the user is not satisfied with the model, he can reduce it by eliminating some items, until he can understand the results.

More, we can use the extension operation on an empty data model, building data models from scratch.

3. THE NEW MINING PROCESS

According to the definition of a data model and of the operations that can be performed on data models, a new mining process can be designed. This new process consists of the following stages:

S'_1 - definition of mining process requirements

S'_2 - data selection

S'_3 - data cleaning and transformation

S'_4 - pre-mining selection of data model, of operation to perform (extending, reducing the model), of frequent items to operate with;

S'_5 - performing selected operation;

S'_6 - presenting results;

S'_7 - interpreting results, if non-satisfactory results go to S'_4;

S'_8 - evaluating results; if important, save new data model.

A major advantage of our association rule mining process is the increased number of human interventions during a complete mining process.

The fact that a previously found model can be extended/reduced with a set of frequent items ensures knowledge reuse.

Let's consider extending a data model. The problem complexity depends on the difference in size between the final data model and the original one. The complexity of reducing a data model depends on the number of non-concepts that become concepts in the final data model.

In a hypothetical case, when no non-concepts and no infrequent itemsets appear, extending a concept lattice, consisting in C concepts and R rules, with a new item will lead to 2C concepts and 2R+C rules. The resulting concepts consist of the original ones and new concepts, one for each of the old one. The resulting rules comprise the old rules, the same number of rules between the extended concepts, and one extra rule for each concept, as shown in Figure 5 (C is a concept, and A_i its adjacent superconcepts, while *i* is the new item).

Following the same reasoning, when reducing a data model with one item, in the hypothetical case, one should get C/2 concepts and (R-C/2)/2 rules. In this case, there will only be a simple filtering operation to perform. In the real case, some extra processing is involved to recover the rules of previous non-concepts, becoming concepts in the new mining context.

4. EXPERIMENTAL RESULTS

We have performed several experiments on different data collections from the public domain. We have used a Pentium II computer, 350 MHz, 128 MB RAM. The first experiment that is revealing for this paper is about generating non-concepts. We have used the http://lib.stat.cmu.edu/ datasets/boston_corrected data, as it is more revealing for our experiment. We have run an

Minsup	A1		A2		
	Closed itemsets	Non-closed itemsets	Closed itemsets	Generating rules	Pseudo-intents
4%	49	1113	49	57	29
3%	73	1627	73	91	46
2%	102	5490	102	197	98
1.5%	144	11047	144	299	164
1%	202	18559	202	449	251

Table 1: Comparative amount of results between an Apriori-type algorithm and our approach

exhaustive Apriori-type algorithm, we will call it A1, that finds all frequent itemsets, closed itemsets as well as non-closed itemsets, and the proposed algorithm, A2, for *minsup*= 1, 1.5, 2, 3, 4 %. The results as shown in Table 1.

We have limited the experiment in what concerns the minimum support, due to the long response time of A1. As we can see from Table 1, our algorithm builds some non-closed itemsets, but significantly less than an exhaustive algorithm. More, it does not store all non-closed itemsets, but pseudo-intents. We have also devised an exhaustive procedure for finding the pseudo-intents of a lattice, in order to validate the results of the extension and reduction algorithms.

Data Model	Items no.	Concepts no.	Pseudo-intents no.	T	Longest itemset
1	21	25	0	1	2
2	46	56	19	1	3
3	55	73	49	2	4
4	56	79	51	2	5
5	93	106	92	5	6
6	95	138	120	8	7
7	119	138	145	9	8
8	128	173	170	14	8
9	135	187	175	18	8
10	142	187	184	18	8
11	175	202	218	23	9

Data models	ΔItems	ΔConcepts	ΔPseudo-intents	ΔT	T↑	T↓
1, 2	25	21	19	0	1	0
2, 3	9	17	30	1	2	2
3, 4	1	6	2	0	1	1
4, 5	37	27	41	3	3	2
5, 6	2	32	28	3	5	2
6, 7	24	0	25	1	5	3
7, 8	9	35	25	5	6	3
8, 9	7	14	5	4	1	1
9, 10	7	0	9	0	2	1
10, 11	33	15	38	5	9	4

Table 2: Finding data models from scratch (upper table) and from previously found data models (lower table)

In what concerns the time response of the algorithm when mining from scratch or from previous results, we will show a relevant one, run on the same data collection with *minsup*= 1%.

In Table 2 (upper table) we describe some characteristics of 11 larger and larger data models and the time required to build them from scratch. The data model characteristics are: the number of selected frequent items, the number of resulting concepts and pseudo-intents, along with the length of the longest frequent itemset. In Table 2 (lower table) we show the time needed to extend, in T↑, and reduce, in T↓, one data model to the next. We also show the number of items used for extension and reduction, the number of concepts and pseudo-intents built or removed, and we compute, in ΔT, the difference between the time corresponding to building those models from scratch.

We can see that sometimes the computed time, ΔT, is bigger than T↑ time, and sometimes is smaller. We know that, theoretically, adding an item depends on the initial number of closed itemsets. The basic algorithm considers attributes in a specific order, namely in the increasing order of their support (all our tests show it is a convenient order, but we have no theoretical support for this statement). Adding items, means considering them last. This is the reason why T↑ is not equal to ΔT. Anyway, even if different, T↑ is considerably smaller than mining from scratch. We can also see from Table 2, that T↓ is significantly smaller than T↑. More, our tests show that while T↑ grows with the number of added items, when extending the same data model, T↓ is smaller when reducing the same data model with more items.

5. CONCLUDING REMARKS

In this paper, we have introduced the idea of data model, expressed as frequent closed itemset lattice, with a base for global implications, if needed. We have also described two important operations applicable on data models: extension and reduction with several items. The main advantages of our approach are:

- Constructing small models of data, makes them more understandable for the user; also, the time of response is small;
- Extending data models with a set of new items returns to the user only the supplementary results, hence a smaller amount of results; the response time is considerably smaller than building the model from scratch;
- Whenever data models are incomprehensible, some of the items can be removed, thus obtaining an easier to understand data model;
- Extending or reducing a model spares the time spent building it, thus reusing knowledge;
- The mining process becomes more interactive and flexible, due to the increased number of human interventions.

We are considering applying this approach to the more complex mining process of finding quantitative association rules. For example, for the quantitative attributes mapping problem, the user-driven character of this approach offers the user means to generate-and-test different mappings of the same quantitative attribute in the context of a certain, pre-acquired, data model.

6. REFERENCES

[1] Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. *In Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207-216, Washington D.C., May (1993).

[2] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. *In Proc. of the VLDB Conference*, Santiago, Chile (1994).

[3] Fayyad U., Data Mining Knowledge Discovery: Making Sense Out of Data *IEEE Expert Special issue on Data Mining*, 1996.

[4] Ganter B.: Algoritmen zur formalen begriffsanalyse, *Beitrage zur Beigriffanalyse (Ganter, Wille, Wolf, eds)*, Wissenschaft-Verlag (1987).

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

[6] Gupta, S. K., Bhatnagar, V., Wasan, S. K., Somayajulu, D., Intension Mining: A New Paradigm in Knowledge Discovery. *Technical Report No. IITD/CSE/TR2000/001*, Indian Institute of Technology, Delhi, INDIA, (2000)

[7] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. *In 7th Intl. Conf. On Database Theory* (1999).

[8] Pei, J., Han, J., Mao, R.: CLOSET: An efficient algorithm for mining frequent closed itemsets. *In Proc. of DMKD 2000*, pp. 11--20 (2000).

[9] Wille, R.: Restructuring lattice theory: an approach based on hierarchies of concepts, *Ordered Sets*, pp. 445-470, (1982).

[10] Zaki, M.J., Hsiao, C.J.: CHARM: An Efficient Algorithm for Closed Association Rule Mining, *RPI Technical Report 99-10* (1999).

[11] Zaki, M.J., Ogihara, M.: Theoretical Foundations of Association Rules, *in Proc. of the 3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, Seattle, WA, pp 7:1-7:8 (1998).