

Iterative Mining for Rules with Constrained Antecedents

Zheng Sun*

Philip S. Yu†

Xiang-Yang Li‡

Abstract

In this study we discuss the following association rule mining problem: for a user-defined set \mathcal{A} of items, the objective is to compute all association rules (satisfying suitable support and confidence thresholds) induced by \mathcal{A} , where an association rule is said to be induced by \mathcal{A} if its antecedent (*i.e.*, LHS) is a subset of \mathcal{A} while the consequent (*i.e.*, RHS) contains no items in \mathcal{A} . In particular, we are interested in a multi-step scenario where in each step \mathcal{A} is incremented by one item and all association rules induced by the updated \mathcal{A} are to be computed. We propose an efficient iterative algorithm that can exploit mining information gained in previous steps to efficiently answer subsequent queries.

1 Introduction

The problem of discovering association rules is to compute, given a set of items $\mathcal{I} = \{I_1, I_2, \dots, I_n\}$ and a set \mathcal{T} of transactions each of which is a subset of \mathcal{I} , all association rules in the form of $X \Rightarrow Y$, where $X, Y \subset \mathcal{I}$ and $X \cap Y = \emptyset$. Here X is called the *antecedent* (or the LHS) of the rule, while Y is the *consequent* (or the RHS). The merit of the rule $r : X \Rightarrow Y$ may be measured by its *support* $\text{Supp}(r)$, the fraction of transactions containing both X and Y out of all transactions in \mathcal{T} , and *confidence* $\text{Conf}(r)$, the fraction of transactions containing Y that also contain X . Usually, we are interested in those *good rules* that satisfy some user-defined support and confidence thresholds, Supp_{\min} and Conf_{\min} .

Traditionally, association rule discovery is conducted in an *offline* manner, and the rules being sought are *general* ones without any constraint on either the antecedent or the consequent part. In this paper we study the problem of iterative mining with constrained antecedents. Each query of our problem is specified with Supp_{\min} , Conf_{\min} , and a set $\mathcal{A} \subset \mathcal{I}$ of *antecedent items* selected by the user. Let $\mathcal{C} = \mathcal{I} \setminus \mathcal{A}$ be the set of *consequent items*. The objective is to find association rules induced by \mathcal{A} , where a rule $r : X \Rightarrow Y$ is said to be induced by \mathcal{A} if $X \subseteq \mathcal{A}$ and $Y \subseteq \mathcal{C}$. In particular, we are interested in a multi-step scenario where in each step \mathcal{A} is incremented by one item and all association rules induced by the updated \mathcal{A} are to be computed. Therefore, it is critical to efficiently handle the previous mining results as new antecedent items are introduced.

To facilitate efficient computation of association rules in the query phase, we use an *adjacency lattice* (see [2]), denoted by $\mathcal{L}_{\text{itemset}}$, to store frequent itemsets with respect to a minimum support picked such that the entire adjacency lattice can be stored in the main memory.

Several different forms of constrained mining have been discussed in previous literature. Srikant *et al.* [8] considered the problem of computing association rules under constraints that are boolean expressions over the presence or absence of items. Bayardo *et al.* [4] described an algorithm that exploits user-defined constraints on support, confidence, and predicative advantage. Lakshmanan *et al.* [6] studied a rich class of anti-monotone and succinct constraints for itemsets and provided mining algorithms that achieve a significant degree of pruning for these constraints. Pei *et al.* [7] studied several types of constraints that are difficult to be handled using traditional techniques. Cong and Liu [5] provided an efficient technique to utilize previous mining information to compute frequent itemsets when the minimum support constraint is relaxed.

Aggarwal and Yu [2] proposed an online mining technique that can efficiently answer queries with different minimum support constraints using a pre-computed adjacency lattice of itemsets. Later Aggarwal *et al.* [1] extended this technique to mining profile association rules with quantitative attributes. Thomas *et al.* [9] studied interactive rule mining with constraint relaxations. In a sense, our iterative mining method complements the existing works by providing a more complete mechanism for iterative mining, with constraints defined on support, confidence, and antecedent/consequent.

2 Preliminaries

Let $\mathcal{A} = \{I_{i_1}, I_{i_2}, \dots, I_{i_k}\}$ be a set of items. For a given \mathcal{A} , an itemset X is called an *antecedent itemset* if $X \subseteq \mathcal{A}$. We use \mathcal{X} to denote the set of all *frequent antecedent itemsets* with supports no less than Supp_{\min} . An itemset Z is called a *rule itemset* if $Z \setminus \mathcal{A} \neq \emptyset$; such an itemset uniquely corresponds to a rule $Z \cap \mathcal{A} \Rightarrow Z \setminus \mathcal{A}$ (without considering any support and confidence thresholds) induced by \mathcal{A} . We call $Z \cap \mathcal{A}$ the *associated antecedent itemset* of Z . In this paper, we assume that the goal is to compute all rule itemsets corresponding to good rules (including those with null antecedents). With minor modifications, our algorithm can also be used to compute rules with some other properties.

We define the following lexicographic order $O(\cdot)$ on items: i) $O(I_j) < O(I_m)$ for any $I_j \in \mathcal{A}$ and $I_m \in \mathcal{C}$; ii) $O(I_{i_j}) < O(I_{i_m})$ if $I_{i_j}, I_{i_m} \in \mathcal{A}$ and $j < m$;

*Hong Kong Baptist University, sunz@comp.hkbu.edu.hk. The author was supported in part by Grant FRG/03-04/II-21.

†IBM T.J. Watson Research Center, psyu@us.ibm.com.

‡Illinois Institute of Technology, xli@cs.iit.edu. The author was supported in part by NSF under Grant CCR-0311174.

iii) $O(I_j) < O(I_m)$ if $I_j, I_m \in \mathcal{C}$ and $j < m$. We say that an item I_j is *ordered before* (after, respectively) I_k if $O(I_j)$ is less than (more than, respectively) $O(I_k)$.

For any itemset Z and any $I_m \in \mathcal{C}$, if $Z' = Z \cup \{I_m\}$ is a good rule itemset and $O(I_m) > O(I_j)$ for all $I_j \in Z$, we say that Z' is a *consequent extension* of Z . We use $\mathcal{E}_{con}(Z)$ to denote the set of all consequent extensions of Z . We define *antecedent extensions* $\mathcal{E}_{ant}(Z)$ analogously. We say that I_m is the *generating item* of Z' and denote it by $I_{gen}(Z')$. We use $\mathcal{I}_{gen}(Z)$ to denote the set of generating items of all itemsets in a set \mathcal{Z} of itemsets.

For the purpose of comparison, we first describe an Apriori-like offline algorithm, FindRulesOffline (see Algorithm 1), that computes all rule itemsets induced by \mathcal{A} assuming that the entire \mathcal{A} is given at once. This algorithm recursively calls Procedure FindRulesForItemset (see Procedure 1), which computes for a given itemset Z all rules that are extensions of Z .

Algorithm 1 FindRulesOffline(\mathcal{A})

- 1: $\mathcal{R} \leftarrow \emptyset; Z \leftarrow \emptyset$.
 - 2: $\mathcal{I}_{cc}(Z) \leftarrow \mathcal{C}; \mathcal{I}_{ca}(Z) \leftarrow \mathcal{A}$.
 - 3: FindRulesForItemset(Z).
-

Procedure 1 FindRulesForItemset(Z)

- 1: **for all** $I_t \in \mathcal{I}_{ca}(Z)$ **do**
 - 2: **if** $\text{Supp}(Z \cup \{I_t\}) \geq \text{Supp}_{min}$ **and** $O(I_t) > O(I_m)$ for all $I_m \in Z$ **then**
 - 3: $\mathcal{I}_{ca}(Z \cup \{I_t\}) \leftarrow \mathcal{I}_{gen}(\mathcal{E}_{ant}(Z)); \mathcal{I}_{cc}(Z \cup \{I_t\}) \leftarrow \mathcal{C}$.
 - 4: FindRulesForItemset($Z \cup \{I_t\}$).
 - 5: **for all** $I_t \in \mathcal{I}_{cc}(Z)$ **do**
 - 6: **if** $\text{Supp}(Z \cup \{I_t\}) \geq \max\{\text{Supp}_{min}, \text{Conf}_{min} \cdot \text{Supp}(X)\}$ **and** $O(I_t) > O(I_m)$ for all $I_m \in Z$ **then**
 - 7: add itemset $Z \cup \{I_t\}$ into \mathcal{R} .
 - 8: $\mathcal{I}_{ca}(Z \cup \{I_t\}) \leftarrow \emptyset; \mathcal{I}_{cc}(Z \cup \{I_t\}) \leftarrow \mathcal{I}_{gen}(\mathcal{E}_{con}(Z))$.
 - 9: FindRulesForItemset($Z \cup \{I_t\}$).
-

Similar to Apriori Algorithm, for any (antecedent or rule) itemset Z and any $Z' \in \mathcal{E}_{ant}(Z) \cup \mathcal{E}_{con}(Z)$, we let the *candidate consequent extension list* $\mathcal{I}_{cc}(Z')$ of Z' be the list $\mathcal{I}_{gen}(\mathcal{E}_{con}(Z))$ of all generating items of $\mathcal{E}_{con}(Z)$. We construct the list $\mathcal{I}_{ca}(Z')$ of *candidate antecedent extensions* in a similar manner, with the exception that for any rule itemset Z' , $\mathcal{I}_{ca}(Z')$ is always empty.

Next, we suppose that antecedent items are added into \mathcal{A} one at a time. Let $\mathcal{A}_k = \{I_{i_1}, I_{i_2}, \dots, I_{i_k}\}$ be the set of antecedent items in Step k , and let \mathcal{R}_k be the list of all good rules itemsets induced by \mathcal{A}_k . In Step $(k+1)$ a new antecedent item I_d is added, and therefore $\mathcal{A}_{k+1} = \mathcal{A}_k \cup \{I_d\}$.

We first examine each rule itemset $X \cup Y$ (corresponding to rule $X \Rightarrow Y$) in \mathcal{R}_k after I_d becomes an antecedent item. There are three cases:

- Type A.** $I_d \notin Y$: $X \cup Y$ is intact, as it corresponds to the same rule with support and confidence unchanged.
- Type B.** $\{I_d\} \subset Y$: $X \cup Y$ now corresponds to a different (but still good) rule $X \cup \{I_d\} \Rightarrow Y \setminus \{I_d\}$.
- Type C.** $\{I_d\} = Y$: $X \cup Y$ is no longer a good rule

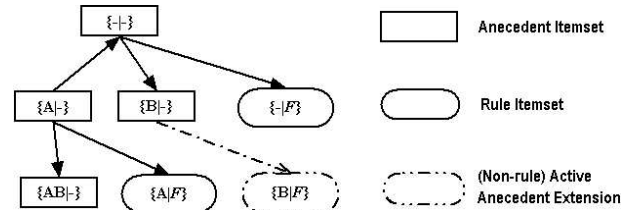


Figure 1: Active antecedent itemsets with respect to item F.

itemset as now the corresponding rule $X \cup \{I_d\} \Rightarrow \emptyset$ is no longer meaningful.

To construct \mathcal{R}_{k+1} from \mathcal{R}_k , we first need to delete all Type C rule itemsets from \mathcal{R}_k . Further, we need to compute all rules in the form of $X \cup \{I_d\} \Rightarrow Y$, where $X \subseteq \mathcal{A}_k$ and $Y \cap \mathcal{A}_{k+1} = \emptyset$. If the corresponding rule itemset $X \cup Y \cup \{I_d\}$ is not in \mathcal{R}_k (i.e., rule $X \Rightarrow Y \cup \{I_d\}$ is not a good rule discovered in Step k), it shall be added into \mathcal{R}_k . We call these rule itemsets *Type D itemsets*. The resulting set of rule itemsets is \mathcal{R}_{k+1} .

To better utilize mining information gained in previous steps, we maintain a *local rule itemset tree* \mathcal{T}_{rule} , which is a lexicographic tree (according to the lexicographic order defined in the previous section) and contains all frequent antecedent itemsets as well as all good rule itemsets. Naturally, there is a one-to-one mapping between nodes in \mathcal{T}_{rule} and nodes in $\mathcal{L}_{itemset}$. There are two types of nodes in \mathcal{T}_{rule} , *antecedent nodes* and *rule nodes*, corresponding to antecedent itemsets and rule itemsets respectively. For any $Z \in \mathcal{T}_{rule}$, we use $\mathcal{T}_{rule}(Z)$ to denote the subtree of \mathcal{T}_{rule} rooted at Z .

3 Terminologies and Data Structures

Support- and confidence-constrained antecedent itemsets Let \mathcal{X}_k be the set of frequent antecedent itemsets with respect to \mathcal{A}_k . Any $X \in \mathcal{X}_k$ is said to be *support-constrained* if $\text{Conf}_{min} \cdot \text{Supp}(X) \leq \text{Supp}_{min}$; otherwise, it is said to be *confidence-constrained*. It is easy to see that, if X is support-constrained, there will be no Type D rule itemset in \mathcal{R}_{k+1} that is a descendent of X . This is because in Step k a rule itemset Z associated with X can only be disqualified (from being a good rule) due to low support and hence it will remain a “bad rule” even after the introduction of a new antecedent item I_d .

Active antecedent extensions A confidence-constrained $X \in \mathcal{X}_k$ is said to be *active with respect to a consequent item* I_m if $X \cup \{I_m\}$ is frequent. For each I_m , we maintain an *active antecedent extension list* $\mathcal{X}_{act}(I_m)$ containing all $X \cup \{I_m\}$ such that X is active with respect to I_m .

As a running example throughout the paper, we assume that we have $\mathcal{I} = \{A, B, \dots, M, N\}$ (in this exact lexicographic order). After two steps, items A and B are specified as antecedent items, and the next item to be specified as an antecedent item is F (i.e., $I_d = F$). In the following discussion as well as in the figures, we

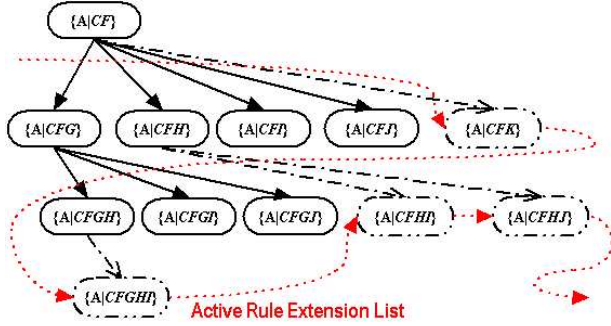


Figure 2: Recursive active rule extensions for rule itemset $\{A|CF\}$.

use $\{A_1 A_2 \dots A_s | C_1 C_2 \dots C_t\}$ to represent an itemset corresponding to rule $\{A_1, A_2, \dots, A_s\} \Rightarrow \{C_1, C_2, \dots, C_t\}$. We use “-” to denote that the antecedent (or consequent) part of an itemset is empty. Figure 1 shows that for consequent item F, there are three active antecedent extensions: itemsets $\{-|F\}$ and $\{A|F\}$ are good rule itemsets, while itemset $\{B|F\}$ does not have enough confidence to be a good rule itemset (but it still has a support no less than Supp_{min}).

It is important to note that there is no additional computation required to acquire such information, as for each frequent antecedent itemset X and each consequent item I_m , we need to check $\text{Supp}(X \cup \{I_m\})$ in order to determine whether $X \Rightarrow \{I_m\}$ is a good rule.

Active rule extensions A good rule itemset Z associated with a confidence-constrained antecedent itemset X is said to be *active with respect to* a consequent item I_m if it is *judged* that the $Z' = Z \cup \{I_m\}$ is frequent (with respect to Supp_{min}) and yet is not a good rule itemset. We call the list containing all such Z' the *active rule extension list of Z* , and denote it by $\mathcal{E}_{act}(Z)$. For example, rule itemset $\{A|CFH\}$ has two active rule extensions, $\{A|CFHI\}$ and $\{A|CFHJ\}$, as shown in Figure 2. If Z is an antecedent itemset or a rule itemset associated with a support-constrained antecedent, its active rule extension list is defined to be empty.

We shall emphasize here that the $\mathcal{E}_{act}(Z)$ may not contain all $Z \cup \{I_m\}$ that are frequent but not good. For example, since itemset $\{A|CFK\}$ is not a good rule itemset (see Figure 2), in previous steps we would not have attempted to extend $\{A|CFH\}$ (or any of the descendant rule itemsets of $\{A|CF\}$ other than $\{A|CFK\}$ itself) by item K, as K is even not in the candidate consequent extension list of $\{A|CFH\}$. Therefore, we have no prior information regarding the support of itemset $\{A|CFHK\}$; it may or may not be a frequent itemset.

However, it is easy to see that, if Z is a good rule item and yet $Z \cup \{I_m\}$ is a frequent but not a good rule itemset, either Z is active with respect to I_m , or there exists an ancestor Z'' of Z such that Z'' is active with respect to I_m . This property can be used to identify all potential Type D rule itemsets in the event of an $I_m \in Z$ switching from a consequent item to an antecedent item.

To facilitate finding these rule itemsets in an efficient manner, we recursively merge the active rule extension lists into a single list \mathcal{E}_{act} . For any good rule itemset Z , we define the *recursive active rule extension list* $\mathcal{E}_{act}^r(Z)$ of Z as the following: $\mathcal{E}_{act}^r(Z) = \mathcal{E}_{act}(Z) \cup (\bigcup_{k=1}^t \mathcal{E}_{act}^r(Z \cup \{I_{j_k}\}))$. Here $Z \cup \{I_{j_1}\}, Z \cup \{I_{j_2}\}, \dots, Z \cup \{I_{j_t}\}$ are the frequent extensions of Z such that $j_1 < j_2 < \dots < j_t$. Therefore, $\mathcal{E}_{act}^r(Z)$ contains all active rule extensions of Z as well as those of rule itemsets extending Z (by one or multiple consequent items). Figure 2 shows the recursive active rule extension list for rule itemset $\{A|CF\}$. \mathcal{E}_{act} is then defined to be $\mathcal{E}_{act}^r(\emptyset)$, which contains all active rule extensions.

4 Iterative Mining: The Algorithm

In this section, we describe an iterative mining algorithm that will update \mathcal{T}_{rule} every time a new antecedent item is specified. The goal is to manipulate \mathcal{T}_{rule} in such an efficient manner that the total computation time of iterative mining is close to the time required if all items in \mathcal{A} are known at once.

We first give the description of the main loop of our algorithm (see Algorithm 2).

Algorithm 2 FindRulesOnline(I_d)

- 1: $\mathcal{A}_{k+1} \leftarrow \mathcal{A}_k \cup \{I_d\}$; $\mathcal{R}_{k+1} \leftarrow \mathcal{R}_k$; $\mathcal{X}_{k+1} \leftarrow \mathcal{X}_k$.
 - 2: **for all** $X' \in \mathcal{X}_{act}(I_d)$ **do**
 - 3: $\mathcal{X}_{k+1} \leftarrow \mathcal{X}_{k+1} \cup \{X'\}$.
 - 4: $X \leftarrow X' \setminus \{I_d\}$; $\mathcal{I}_{cc}(X') \leftarrow \mathcal{I}_{gen}(\mathcal{E}_{con}(X) \cup \mathcal{E}_{act}(X))$.
 - 5: **if** X' was already a good rule itemset in Step k **then**
 - 6: change X' to an antecedent itemset.
 - 7: ComputeSubTree(X', I_d).
 - 8: **else**
 - 9: $\mathcal{I}_{ca}(X') \leftarrow \emptyset$.
 - 10: FindRulesForItemset(X').
-

Notice that, in Step $(k+1)$ we only need to search for good rule itemsets associated with an antecedent $X' = X \cup \{I_d\}$ for each $X \in \mathcal{X}_k$. If X is supported-constrained, there will be no Type D rule itemsets underneath X , meaning that no new good rule itemsets associated with X' will be found. Therefore, we can skip the subtree $\mathcal{T}_{rule}(X')$ of \mathcal{T}_{rule} rooted at X' entirely.

Moreover, for a confidence-constrained X , we need to search for good rule itemsets associated with $X' = X \cup \{I_d\}$ only if $\text{Supp}(X') \geq \text{Supp}_{min}$, meaning that X was active with respect to I_d in the previous step, when I_d was a consequent item. Therefore, we just need to go through $\mathcal{X}_{act}(I_d)$, and add each $X' \in \mathcal{X}_{act}(I_d)$ into \mathcal{X}_{k+1} , as indicated in Line 3 of Algorithm 2.

Among these new antecedent itemsets, some were not good rule itemsets in the previous step. For each of these antecedent itemsets X' , to build the subtree $\mathcal{T}_{rule}(X')$ rooted at X' , we can only use Procedure 1 (as indicated in Line 10 of Algorithm 2) as there is no prior information to utilize. The remaining new antecedent itemsets are the Type C ones; they were converted from good rule itemsets found in the previous step. For each such X' , it would be wasteful to compute

the subtree $\mathcal{T}_{rule}(X')$ from scratch in case X' already has some descendants in \mathcal{T}_{rule} , as these itemsets shall be retained in this step; they are the Type B ones. Instead, we use `ComputeSubTree` (see Procedure 2) to compute $\mathcal{T}_{rule}(X')$, as indicated in Line 7 of Algorithm 2.

To illustrate how Procedure 2 works, we use itemset $X' = \{AF|\cdot\}$ (which was $\{AF\}$ in the previous step) as an example. The new antecedent item F divides all consequent items into two groups; Group 1 contains those ordered before F in the previous step (*i.e.*, C, D, and E), and Group 2 contains those ordered after F (*i.e.*, G, H, \dots , N). In turn, the subtree $\mathcal{T}_{rule}(X')$ is divided into two partial trees: the left partial tree, which contains all the immediate children (as well as their descendants) that extend $\{AF|\cdot\}$ by items in Group 1, and the right partial tree, which contains all immediate children (as well as their descendants) that extend $\{AF|\cdot\}$ by items in Group 2. In the following, we show how to compute the two partial trees.

Procedure 2 `ComputeSubTree(Z, I_d)`

```

1: ExpandPartialSubTree( $Z, \mathcal{E}_{act}^r(Z), \emptyset$ ).
2: if  $Z$  is an antecedent node then
3:    $i_{min} \leftarrow 0$ .
4: else
5:    $I_m \leftarrow I_{gen}(Z); i_{min} \leftarrow m$ .
6: for all  $I_t \in C_{k+1}$  such that  $i_{min} < t < d$  do
7:   if  $\text{Supp}(Z \cup \{I_t\}) \geq \max\{\text{Supp}_{min}, \text{Conf}_{min} \cdot \text{Supp}(X)\}$ 
   then
8:     if  $Z \cup \{I_t\}$  was already a good rule itemset in Step  $k$ 
       then
9:       change the parent of  $Z \cup \{I_t\}$  from  $Z \setminus \{I_d\} \cup \{I_t\}$  to
          $Z$ .
10:    ComputeSubTree( $Z \cup \{I_t\}, I_d$ ).
11:   else
12:     add  $Z \cup \{I_t\}$  as a new child of  $Z$ .
13:    $\mathcal{I}_{cc}(Z \cup \{I_t\}) \leftarrow \mathcal{I}_{gen}(\mathcal{E}_{con}(Z)); \mathcal{I}_{ca}(Z \cup \{I_t\}) \leftarrow \emptyset$ .
14:   FindRulesForItemset( $Z \cup \{I_t\}$ ).

```

Reconstruct the Left Partial Tree In the previous step, we would not have tried to extend $\{A|F\}$ by any of C, D and E, because they were ordered before F. In this step, to compute the left partial tree of $\{AF|\cdot\}$, we first need to check whether $\{AF|C\}$, $\{AF|D\}$, and $\{AF|E\}$ are good rule itemsets; if they are, we need to further compute the subtrees rooted at those itemsets.

We may find Type D children of $\{AF|\cdot\}$, *e.g.*, $\{AF|E\}$. Before F becomes an antecedent item, $\{AF|E\}$ was $\{A|EF\}$ (see Figure 3), who did not satisfy the minimum confidence constraint. We may also find Type B children of $\{AF|\cdot\}$, *e.g.*, $\{AF|C\}$, which was $\{A|CF\}$ in the previous step. However, at that time $\{A|CF\}$ was a child of $\{A|C\}$. Now itemset $\{A|CF\}$ becomes $\{AF|C\}$ and thus should be changed to become a child of $\{AF|\cdot\}$ to maintain the lexicographic order, as indicated in Line 9 of Procedure 2.

For any Type D child Z' of $\{AF|\cdot\}$ found in the left partial tree, we use Procedure 1 (see Line 14 of Procedure 2) to compute the subtree rooted at Z' . If Z'

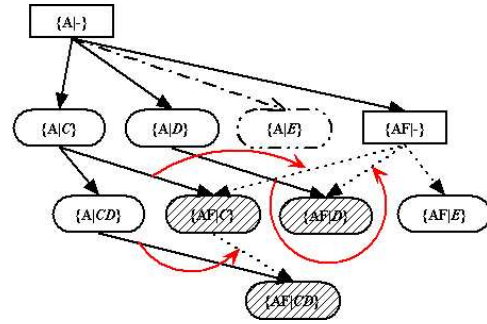


Figure 3: Recursively reconstructing the left partial tree for $\{AF|\cdot\}$. The shaded rule nodes are the ones to be moved.

is a Type B one, however, we recursively call Procedure 2, as indicated in Line 10 of Procedure 2, so that we can “reuse” the descendants of Z' already in \mathcal{T}_{rule} .

Expand the Right Partial Tree To compute the right partial tree for a Type B or C itemset Z , Procedure 2 invokes, as indicated in Line 1, the procedure `ExpandPartialSubTree` (see Procedure 3). We now describe how Procedure 3 works by using the example of $Z = \{AF|C\}$. Our goal is to give a chance to all the rule itemsets underneath Z that have been disqualified as good rule itemsets due to low confidence, while avoiding revisiting those ones who would never become good rule itemsets due to low support.

Procedure 3 `ExpandPartialSubTree(Z, L_{act})`

```

1: if  $\mathcal{I}_{cc}(Z) = \emptyset$  then
2:   if  $L_{act} = \emptyset$  then
3:     return.
4:    $Z' \leftarrow$  the first active rule extension in  $L_{act}$ .
5:    $Z'' \leftarrow$  the parent node of  $Z'$ .
6:   if  $Z'' \neq Z$  then
7:      $Z \leftarrow Z''; \mathcal{I}_{cc}(Z) \leftarrow \emptyset$ .
8:    $L'_{act} \leftarrow \emptyset$ .
9:   remove from  $L_{act}$  all active rule extensions that extend  $Z$ 
     and add them into  $L'_{act}$ .
10:   $L_{can} \leftarrow \mathcal{I}_{gen}(L'_{act}) \cup \mathcal{I}_{cc}(Z); L_{new} \leftarrow \emptyset$ .
11:  for all  $I_t \in L_{can}$  such that  $t > d$  do
12:    if  $\text{Conf}(Z \cup \{I_t\}) \geq \text{Conf}_{min}$  (after  $I_d$  becomes an
      antecedent item) then
13:       $\mathcal{E}_{con}(Z) \leftarrow \mathcal{E}_{con}(Z) \cup \{Z \cup \{I_t\}\}$ .
14:       $L_{new} \leftarrow L_{new} \cup \{Z \cup \{I_t\}\}$ .
15:  for all  $Z' \in \mathcal{E}_{con}(Z)$  ordered by generating item do
16:    if  $Z' \in L_{new}$  then
17:       $\mathcal{I}_{cc}(Z') \leftarrow \mathcal{I}_{gen}(\mathcal{E}_{con}(Z)); \mathcal{I}_{ca}(Z') \leftarrow \emptyset$ .
18:      FindRulesForItemset( $Z'$ ).
19:    else
20:       $\mathcal{I}_{cc}(Z') \leftarrow \mathcal{I}_{gen}(L_{new})$ .
21:      ExpandPartialSubTree( $Z', L_{act}$ ).

```

In the previous step, itemset Z (then denoted by $\{A|CF\}$) was already in \mathcal{T}_{rule} and thus we have computed the subtree of good rule itemsets underneath Z , as shown in Figure 2. Further, we have identified four active rule extensions in $\mathcal{E}_{act}^r(Z)$, namely, $\{A|CFK\}$, $\{A|CFGHI\}$, and $\{A|CFHJ\}$, $\{A|CFHI\}$. Now F becomes an antecedent item, and thus each of these rule itemsets shall have an increased confidence. Therefore, we shall

check each of them to see if it now satisfies the minimum confidence constraint.

Further, for any Type B itemset Z' in the subtree $\mathcal{T}_{rule}(Z)$, we should try to extend Z' if either Z' itself has active rule extension(s), or Z' has new “younger siblings.” For example, if $Z' = \{A|CH\}$, we should try to extend Z' by: i) items I and J, as previously $\{A|CFH\}$ was active with respect to I and J; and ii) item K, as it is the generating item of $\{A|CK\}$, a younger sibling of Z' . (Here we assume that the confidence of $\{A|CK\}$, originally $\{A|CFK\}$, grows enough to make it a good rule itemset.) The trick is, however, that we should avoid revisiting each $Z' \in \mathcal{T}_{rule}(Z)$ to see if it is “expandable,” which could be costly if $\mathcal{T}_{rule}(Z)$ contains a large number of nodes (in case we use relatively relaxed thresholds for support and confidence).

In Procedure 3, we add all immediate children of Z in a list L_{new} (see Line 14). We use the generating items of all itemsets in L_{new} as the candidate consequent extensions $\mathcal{I}_{cc}(Z')$ for each existing child Z' of Z , and recursively call Procedure 3 to revisit Z' so that we can expand the subtree rooted at Z' . Therefore, if a revisited Z' is found to have no candidate consequent extensions, we do not need to revisit any of its existing children. We can skip $\mathcal{T}_{rule}(Z')$ entirely if Z' does not have any recursive active rule extensions (*i.e.*, $\mathcal{E}_{act}^r(Z') = \emptyset$); or, we go directly to the first recursive active rule extension of Z' (see Lines 1-7 of Procedure 3). For any Type D itemset Z' found, we invoke Procedure 1 to compute the subtree $\mathcal{T}_{rule}(Z')$, as indicated in Line 18 of Procedure 3.

5 Experimental Results

To conduct experiments, we generated using the method first proposed by Agrawal and Srikant [3] a synthetic data set T20.I8.D100K containing 100,000 transactions. We generated using a minimum support of 0.15% an adjacency lattice with 968,820 nodes. We compare Algorithm 2 and Algorithm 1 using a number of combinations of support and confidence thresholds. For each combination, we ran 30 different test cases, each with a distinct set \mathcal{A} of 300 antecedent items (out of a total of 1000 items) randomly picked, and recorded the running time averaged over the 30 test cases.

Figure 4 illustrates the “competitive ratio” of Algorithm 2, which is defined to be the ratio of its average running time to that of Algorithm 1. Here for Algorithm 1 the running time is defined to be the time for answering a *single* query with all 300 antecedent items known in advance. For Algorithm 2, the running time is the total time for finishing 300 queries, each of which was performed after one antecedent item was added. Figure 4 shows that the performance of our online algorithm is very close to that of the offline algorithm. Further, for Algorithm 2, answering a single query is almost instantaneous. This demonstrates the advantage of pre-

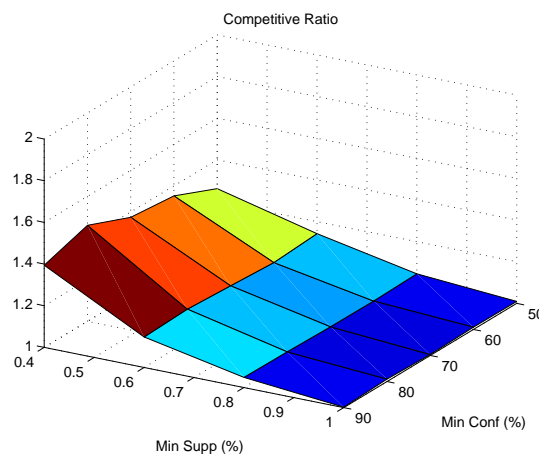


Figure 4: Competitive ratio.

computing an adjacency lattice for mining association rules with constrained antecedents.

6 Conclusion and Future Works

In this paper we provided a novel approach for online mining of association rules with constrained antecedents and in multiple steps. By carefully maintaining a tree of rules as well as some auxiliary data structures, our algorithm performs nearly as well as the offline algorithm, as shown by the preliminary experimental results. For future works, we plan to study how this algorithm can be effectively combined with other iterative mining algorithms to answer queries with both tightened/relaxed support and confidence constraints, as well as increased/decreased antecedent set. Further, we are interested in studying an alternative approach that does not require the pre-computation of an adjacency lattice.

References

- [1] C. C. Aggarwal, Z. Sun, and P. S. Yu, *Fast algorithms for online generation of profile association rules*, TKDE, 14 (2002), pp. 1017–1028.
- [2] C. C. Aggarwal and P. S. Yu, *A new approach to online generation of association rules*, TKDE, 13 (2001), pp. 527–540.
- [3] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules*, VLDB'94, pp. 487–499.
- [4] R. Bayardo, R. A. Jr., and D. Gunopulos, *Constraint-based rule mining in large, dense databases*, ICDE'99, pp. 188–197.
- [5] G. Cong and B. Liu, *Speed-up iterative frequent itemset mining with constraint changes*, ICDE'02, pp. 107–114.
- [6] L. V. S. Lakshmanan, R. T. Ng, J. Han, and A. Pang, *Optimization of constrained frequent set queries with 2-variable constraints*, SIGMOD'99, pp. 157–168.
- [7] J. Pei, J. Han, and L. V. S. Lakshmanan, *Mining frequent item sets with convertible constraints*, ICDE'01, pp. 433–442.
- [8] R. Srikant, Q. Vu, and R. Agrawal, *Mining association rules with item constraints*, KDD'97, pp. 67–73.
- [9] S. Thomas and S. Chakravarthy, *Incremental mining of constrained associations*, HiPC'00, pp. 547–558.