Preknowledge-based generalized association rules mining

Yin-Fu Huang* and Chieh-Ming Wu
Graduate School of Engineering Science and Technology, National Yunlin University of Science and Technology, Taiwan, R.O.C.

Abstract. The subject of this paper is the mining of generalized association rules using pruning techniques. Given a large transaction database and a hierarchical taxonomy tree of the items, we attempt to find the association rules between the items at different levels in the taxonomy tree under the assumption that original frequent itemsets and association rules have already been generated in advance. The primary challenge of designing an efficient mining algorithm is how to make use of the original frequent itemsets and association rules to directly generate new generalized association rules, rather than re-scanning the database. In the proposed algorithms GMAR (Generalized Mining Association Rules) and GMFI (Generalized Mining Frequent Itemsets), we use join methods and/or pruning techniques to generate new generalized association rules. After several comprehensive experiments, we find that both algorithms are much better than BASIC and Cumulate algorithms, since they generate fewer candidate itemsets, and furthermore the GMAR algorithm prunes a large amount of irrelevant rules based on the minimum confidence.

Keywords: Data mining, generalized association rules, taxonomy trees, frequent itemsets, maximal itemsets, pruning techniques

1. Introduction

In the recent ten years, vast data have been propagated and recorded in the databases of different applications, due to developments in information technology and advances of the Internet. Currently, data mining is arising to understand, analyze, and use these data. One of the major tasks of data mining is to find association rules for helping retail industries to understand consumer behaviors [1, 2, 7, 13].

Generalized association rule mining plays a very important role in Knowledge discovery in Databases (KDD) [3]. Generalized association rule mining is an extension of Boolean association rules. In [12], use the approach (MOSSFRA) for selecting import variables to apply fuzzy association rule mining. The three-step method is proposed based on Łukasiewicz logic for the description of the rules and the fuzzy memberships to construct concise and highly comprehensible fuzzy rules [4]. Various efficient algorithms have been proposed by researchers, including the new parallel algorithm with a classification hierarchy by Shintani and Kitsuregawa [14], different minimum supports on a multiple level tree by Lui and Chung [10], the Partial-Match Retrieval method by Matsumoto et al. [11], some novel pruning techniques by Huang [8] and Wu [17, 18]. The association rule mining at multiple concept levels, proposed by Han [6] and Yen [19] and Gaul and Schmidt-Thieme [5], leads to the discovery of more concrete knowledge.

The motivation of our research was initiated by two situations. One was that although mining association rules had been completed, it did not consider the taxonomy tree at that time. Another was that although the taxonomy tree was considered, it may have needed...
to be adjusted as time went by, and thus the generalized association rules mined before were not useful any more. Until now, all proposed researches can only mine all association rules from scratch. In this paper, our goal is to find generalized association rules without re-scanning the original database. Here the pre-knowledge, including the taxonomy tree, the original frequent itemsets and association rules, all generated beforehand, can be used to mine new generalized association rules. Our paper differs from other ones [9, 16], in that we assume the minimum supports of all the items are the same. Finally, in our algorithms GMAR (generalized mining association rules) and GMFI (generalized mining frequent itemsets), we use join methods and/or pruning techniques to generate new generalized association rules. In the experiments, we find that the GMAR and GMFI algorithms are much better than BASIC and Cumulate algorithms [15], since they generate fewer candidate itemsets, and furthermore the GMAR algorithm prunes a large amount of irrelevant rules based on the minimum confidence.

In Section 2 below, the mining problem for generalized association rules is defined. We then propose 1) the GMAR algorithm using the original association rules and 2) the GMFI algorithm using the original frequent itemsets respectively to find generalized association rules (Section 3). In Section 4, several experiments are undertaken and the results show the superiority of GMAR and GMFI algorithms over BASIC and Cumulate algorithms.

2. Problem descriptions

The problem investigated here is to mine generalized association rules according to a large transaction database D and a hierarchical taxonomy tree H of the purchased items. Transaction database D is a set of transactions where each transaction T consists of a set of items, called an itemset, such that \( T \subseteq I \). Here \( I = \{ i_1, i_2, \ldots, i_m \} \) is a set of the items purchased by all the transactions. Besides, the hierarchical taxonomy tree H is a directed acyclic graph on the items where an edge in H represents an “is-a” relationship between items. For example, if there is an edge from p to c in H, we call p a parent of c and c a child of p. In other words, we can say p represents a generalization of c. Further, if there is a path from \( \hat{e} \) to e in the transitive-closure of H, we call \( \hat{e} \) an ancestor of e, and e a descendant of \( \hat{e} \). Since H is acyclic, any item cannot be an ancestor of itself. Furthermore, all the leaf items in H must come from I.

We can definitely find generalized association rules from scratch, given a large transaction database D and a hierarchical taxonomy tree H of the purchased items. However it is not avoidable to scan the original database once more, and this will be very inefficient. Thus an efficient mining algorithm proposed here is to make use of the original frequent itemsets and association rules that have already been generated beforehand, to produce new generalized association rules, rather than re-scanning the original database. In addition, we also propose some pruning techniques to speed up the mining process.

3. Mining algorithms

3.1. Processing flow of mining algorithms

The processing flow of our mining algorithm for finding generalized association rules is shown in Fig. 1. Suppose that the components shown inside the dotted box, such as vertical-TID-vector table, frequent itemsets, and association rules, were generated before our mining algorithm is executed. Rather than scanning the original database, we make use of the vertical-TID-vector table transformed from the original database to mine generalized association rules. Since the bit storage is used in the vertical-TID-vector table, the memory space and processing time used to get the database information will be saved. Besides the original frequent itemsets and association rules generated beforehand should not be thrown away, since we need them to produce generalized association rules.

At first, we create maximal itemsets from the original frequent itemsets, and then store them in an array to reduce disk space and speed up the mining operation. Although no subsets of a maximal itemset are stored, we can use a hash function to calculate the position of each subset in a maximal itemset and find its support. Next, given a hierarchical taxonomy tree, we must calculate the supports of the non-leaf items in the taxonomy tree from the maximal itemsets or VTV table. Basically, for a non-leaf item t being the ancestor of \( \{ i_1, i_2, \ldots, i_k \} \) where items \( i_1, i_2, \ldots, i_k \) are leaf items, the support of item t can be found in the array of maximal itemset I if \( \{ i_1, i_2, \ldots, i_k \} \) is a subset of maximal itemset I. Then, we use the original frequent itemsets \( L_1 \) and \( L_2 \), and non-leaf items with the supports \( \geq \) the minimum support to create an association graph. It facilitates the check whether k-itemsets (\( k \geq 3 \)) involving non-leaf and leaf items are frequent or
not. Finally, in our algorithm GMAR, we use join methods and pruning techniques to generate new generalized association rules. The join methods used in the GMAR algorithm can directly produce generalized association rules from the original association rules, and the pruning techniques are used to prune irrelevant rules, thereby speeding up the production of generalized association rules.

3.2. The vertical-TID-vector table

The vertical-TID-vector table was generated before the mining algorithm is executed. It uses bit vectors to record the transaction information. Each item is represented with a bit vector where the length of the bit vector is the total number of transactions in the database. If an item appears in the jth transaction, the jth bit of the corresponding bit vector is set to 1; otherwise, the bit is set to 0. For example, suppose item i3 appears in transactions 1, 8, 9, and 10, and there are a total of 10 transactions in the database, the bit vector for item i3 can be expressed as [1000000111]. The bit vector facilitates the support computation of an itemset \{i_1, i_2, \ldots, i_k\}; i.e., using bitwise operations AND for the corresponding bit vectors of item i_1, i_2, \ldots, and i_k. For the transaction database and vertical-TID-vector table as shown in Fig. 2 the support of itemset \{i_1, i_3, i_5\} can be computed through [0100000111] AND [1000000111] AND [0110011111] = [0000000111], and its support is the total number of 1’s in the resulting bit vector divided by 10 (i.e., the number of transactions).

3.3. Creating maximal itemsets

A maximal itemset is defined as a frequent itemset not contained in any other frequent itemset. For example, given a set of frequent itemsets \{\{i_1\}, \{i_2\}, \{i_3\}, \{i_4\}, \{i_5\}, \{i_1, i_3\}, \{i_1, i_5\}, \{i_2, i_5\}, \{i_3, i_5\}, \{i_1, i_3, i_5\}\}, the maximal itemsets are \{i_1\}, \{i_2\}, \{i_3\}, \{i_4\}, \{i_5\}, \{i_1, i_5\}, \{i_2, i_5\}, \{i_3, i_5\}, \{i_1, i_3, i_5\}. The procedure to find the maximal itemsets from a set of frequent itemsets is shown in Fig. 1 and can be described as follows:
set.

non-empty subsets of maximal itemset as shown in Fig. 3. In the figure, the supports of all reduce disk space and speed up the mining operation, 4

Y.-F. Huang and C.-M. Wu / Preknowledge-based generalized association rules mining

follows:

0.5. The hash function used here can be expressed as

\begin{align*}
\text{hf}1 &= \sum_{i=1}^{L-1} t_{i1} t_{i2} \\
\text{hf}2 &= \sum_{i=1}^{L-1} (t_{i1} - X(i) + 1) \end{align*}

where hf1 and hf2 calculate the relative position of a target among the subset itemsets with the same length L. Finally, HF = hf1 + hf2 + 1 can calculate the precise position among all the 2^L - 1 subset itemsets.

3.4. Calculating the supports of non-leaf items

Given the original frequent itemsets and association rules already generated beforehand, and a hierarchical taxonomy tree, we can produce new generalized association rules, rather than re-scanning the original database. However, the items (except those at the leaf level in the taxonomy tree) do not appear in the original frequent itemsets and association rules. Therefore we must calculate the supports of the non-leaf items in the taxonomy tree from the maximal itemsets or VTV table. For a non-leaf item t being the ancestor of {i1, i2, ..., in} where items i1, i2, ..., in are leaf-items, the support of item t can be calculated as follows:

\begin{align*}
\text{Support}(t) &= \text{Support}(\{i1, i2, ..., in\}) \\
&= \text{Support}(i1) \times \text{Support}(i2) \times \ldots \times \text{Support}(in)
\end{align*}

Here we take an example to illustrate how the supports of non-leaf items are calculated. Given the VTV table and the taxonomy tree as shown in Figs 2 and 4, we can calculate the supports of non-leaf items 1001, 1002, and 1003 as follows:

Support(1001) = vector(1) OR vector(2) OR vector(3) / no. of transactions = 1002 / 10 = 0.1

Support(1002) = vector(1) OR vector(2) / no. of transactions = 1003 / 10 = 0.3

Support(1003) = vector(4) OR vector(5) / no. of transactions = 1001 / 10 = 0.1

Fig. 3. The maximal itemsets.

Here we only need to store the maximal itemsets to reduce disk space and speed up the mining operation, as shown in Fig. 3. In the figure, the supports of all non-empty subsets of maximal itemset {i1, i2, i3} are 0.5 for {i1}, 0.4 for {i2}, 0.7 for {i3}, 0.3 for {i1, i2}, 0.5 for {i1, i3}, 0.7 for {i2, i3}, and 0.3 for {i1, i2, i3}. Instead of storing all non-empty subsets of a maximal itemset, we can use the hash function to calculate the position of any subset in a maximal itemset and find its support. For example, using the hash function, we can compute the value 5 for the position of {i1, i3} in maximal itemset {i1, i2, i3}, and find its support value 0.5. The hash function used here can be expressed as follows:

\begin{align*}
\text{HF} &= \sum_{i=1}^{L-1} t_{i1} t_{i2} + \sum_{i=1}^{L-1} (t_{i1} - X(i) + 1) + 1
\end{align*}

Fig. 4. Taxonomy tree.

Rationale:

\begin{align*}
\text{HF} &= \text{hf}1 + \text{hf}2 + 1
\end{align*}
3.5. Creating association graphs

In our mining algorithm differs from the Apriori algorithm in that we only use the original frequent itemsets \(L_1\) and \(L_2\) to generate \(k\)-itemsets \((k \geq 3)\) involving non-leaf and leaf items and check whether they are frequent or not. Our method is based on a graph called the association graph to search all these \(k\)-itemsets. The association graph is defined as follows:

A graph \(AG(V, E)\), called the association graph, consists a set of vertices or items \(V\) and a set of edges \(E\) where

1) \(V = L_1 \cup T\) where \(T = \{v \mid v\) is a non-leaf item and \(\text{support}(v) \geq \text{the minimum support}\}\), and
2) \(E = \{(u, v) \mid (u, v) \in L_2\} \cup \{(x, y) \mid \text{at least one item of } (x, y) \text{ is in } T, x \text{ and } y \text{ are not in the ancestor-offspring relationship, and } \text{support}(x, y) \geq \text{the minimum support}\}\)

Given the VTV table and the taxonomy tree as shown in Figs 2 and 4, the minimum support 0.3, and the original frequent itemsets \(L_1\) and \(L_2\), we can obtain the frequent itemsets \(T\), and then create the association graph as shown in Fig. 5.

\[L_1 = \{1, 2, 3, 4, 5\}, \quad L_2 = \{1, 3, 2, 5, 3, 5\}\]
\[T = \{1001, 1002, 1003\}\]
\[\text{Support}(1, 1003) = \{0010011110\} \text{ AND } \{1110011111\}/10 = 0.5\]
\[\text{Support}(2, 1003) = \{0001100001\} \text{ AND } \{0110011111\}/10 = 0.4\]
\[\text{Support}(3, 1001) = \{0100000111\} \text{ AND } \{1010011111\}/10 = 0.3\]
\[\text{Support}(4, 1001) = \{0001100000\} \text{ AND } \{0110011111\}/10 = 0.1\]

Fig. 5. Association graph. (Colours are visible in the online version of the article; http://dx.doi.org/10.5233/JISPS-2010-0469)

3.6. Pruning techniques

In the GMAR algorithm to be discussed in the next section, we use pruning techniques to generate new generalized association rules since pruning irrelevant rules can speed up the production of generalized association rules. Here we present six pruning techniques described as follows:

PT 1: For a frequent itemset \(I\) where \(i_{\text{min}}\) is the item with the minimum support count within it, if support(\(i_{\text{min}}\))/support(subset(I)) is less than the minimum confidence, then we can prune any rule \(I\Rightarrow P\) instead of support(I) in the confidence check.

Rationale: For any rule subset(I) \(\Rightarrow I\) \(-\) subset(I), the confidence check is support(I)/support(subset(I)). Since support(I) must be less than or equal to support(i_min), support(I)/support(subset(I)) is definitely less than the minimum confidence as long as support(i_min)/support(subset(I)) is less than the minimum confidence. In other words, we can use support(i_min) instead of support(I) in the confidence check.

PT 2: For a subset \(P\) in a frequent itemset \(I\), if the rule \(P\Rightarrow \text{NOT} P\) holds, then any rule whose antecedent containing \(P\), such as superset(P) \(\Rightarrow I\), is less than support(P) holds as well. 

Rationale: If the rule \(P\Rightarrow \text{NOT} P\) holds, it implies that support(P)/support(P) is less than or equal to the minimum confidence. Nevertheless, since support(P) is less than or equal to support(P)/support(P), support(P) is definitely more than or equal to the minimum confidence, and thus the corresponding rule holds as well.

PT 3: For two rules with the same antecedent, if 1) the rule \(P\Rightarrow \text{Q}_{1}\) holds and 2) the rule \(P\Rightarrow \text{Q}_{2}\) does not hold due to support(P \(\cup\) \(Q_{2}\)) \(<\) the minimum support, then all the rules corresponding to the itemset \(I = P \cup \text{Q}_{1} \cup \text{Q}_{2}\) do not hold.
Rationale: If one of the rules corresponding to the itemset I holds, then I must be a frequent itemset and all the subsets are frequent itemsets as well. However we know P∪Q2 is not a frequent itemset since support(P∪Q2) is less than the minimum support. Thus all these rules corresponding to the itemset I do not hold. For example, if 1) the rule i1⇒i2 holds and 2) the rule i1⇒i3 does not hold due to support([i1, i3])<the minimum support, then all the rules, such as i2⇒i1 ∧ i3, i2 ⇒ i1 ∧ i3, i2 ∧ i3 ⇒ i1, i1 ∧ i3 ⇒ i2, and i1 ∧ i3 ⇒ i1, do not hold.

PT 4: If the rule P⇒Qk holds, then the rule P⇒Q2 always holds where Q2 is derived from Q1 by replacing some items in Q1 with their ancestors.

Rationale: If the rule P⇒Q1 holds, it implies that support(P∪Q1)/support(P) is more than or equal to the minimum confidence. Since support(Q2) is more than or equal to support(Q1), support(P∪Q2)/support(P) is definitely more than or equal to the minimum confidence and thus the rule P⇒Q2 always holds.

PT 5: For two rules with the same antecedent, if 1) the rule P⇒Q1 holds and 2) the rule P⇒Q2 does not hold due to not satisfying the minimum confidence although it satisfies the minimum support, then the rule P⇒Q1∪Q2 does not hold.

Rationale: For the confidence check support(P∪Q2)/support(P)<the minimum confidence, support(P∪Q1∪Q2)/support(P) is definitely less than the minimum confidence since support(P∪Q1) is less than or equal to support(P∪Q2). Thus the rule P⇒Q1∪Q2 does not hold.

PT 6: If any item in a frequent itemset I does not appear in association graph AG or all the items of I do not form a complete connection in AG, then, for a subset P of I, the rule P⇒I−P does not hold.

Rationale: If the rule P⇒I−P holds, support(I)/support(P) is not only more than or equal to the minimum confidence, but also support(I) should be more than or equal to the minimum support. In other words, all the items in I would appear in AG and form a complete connection. This contradicts the assumption, and therefore the rule P⇒I−P does not hold.

3.7. GMFI and GMAR algorithms

The Apriori algorithm proposed by Agrawal and Srikant is a two-step process which consists of join and pruning actions to find frequent itemsets, and then uses the frequent itemsets to derive association rules. In this section, apart from the GMAR (generalized mining association rules) algorithm, we also propose another algorithm GMFI (generalized mining frequent itemsets). They both generate generalized association rules not directly based on the raw data from the database, but based on the original frequent itemsets and association rules.

The GMFI algorithm also has two steps to generate generalized association rules: i.e., finding all new frequent itemsets and generating generalized association rules from the new frequent itemsets. Here we show the detailed steps of the GMFI algorithm as follows:

GMFI Algorithm:
Input: VTV table, original frequent itemsets (Lo), taxonomy tree, min_sup, min_conf
Output: new frequent itemsets

Math:
1. For each (k=2, Ln−1(k−1)∈.characteristics),
   \[ C_k = \text{GMFI}(k, \text{Ln−1(k−1)}, \text{Lo}, \text{taxonomy tree}, \text{min_sup}, \text{min_conf}) \]

Procedure GMFI(C, L1, L2, k)
1. C = C ∪ \{\}
2. For each itemset l1 ∈ L1
   For each itemset l2 ∈ L2
     \[ l = l1 \cup \{i\} \]
     \[ \text{if } \text{Check}(l, k) \]
     \[ \text{Add } l \text{ to } C \]
3. Return C

Procedure Check(l, k)
1. For each (k−1)-subset s of l
   If s \( \neq l_{k−1} \)
     \[ \text{Return false; } \]
   2. Return true;

The detailed algorithm is described as follows:

GMFI Algorithm:
Input: VTV table, original frequent itemsets (Lo), taxonomy tree, min_sup, min_conf
Output: new frequent itemsets

Math:
1. Generate new frequent 1-itemsets Ln1 using VTV table and taxonomy tree:
   \[ L1 = \text{GMFI}(1, \text{Lo}, \text{taxonomy tree}, \text{min_sup}, \text{min_conf}) \]

Procedure GMFI(C, L1, L2, k)
1. C = C ∪ \{\}
2. For each itemset l1 ∈ L1
   For each itemset l2 ∈ L2
     \[ l = l1 \cup \{i\} \]
     \[ \text{if } \text{Check}(l, k) \]
     \[ \text{Add } l \text{ to } C \]
3. Return C

Procedure Check(l, k)
1. For each (k−1)-subset s of l
   If s \( \neq l_{k−1} \)
     \[ \text{Return false; } \]
   2. Return true;
Given the VTV table and the original frequent itemsets $L_{o1}, L_{o2},$ and $L_{o3}$ (as shown in Fig. 2), the taxonomy tree (as shown in Fig. 4), the minimum support 0.3, and the minimum confidence 0.5, we can generate new frequent itemsets (Fig. 7) and then produce new generalized association rules using the GMFI algorithm.

In contrast, the GMAR algorithm directly generates generalized association rules from the original association rules. Here an association rule is called weak when it satisfies the minimum support threshold, but not minimum confidence threshold. In the GMAR algorithm, we need both strong and weak association rules in the current level ($k$) to generate the generalized association rules for from the next level $k$ to the next 2($k-1$) level, as shown in Fig. 8. Therefore, the generation of the generalized association rules is not based on a step-by-step manner. The detailed algorithm is described as follows:

**GMAR Algorithm:**

**Input:** VTV table, original association rules ($R_o$), taxonomy tree, $min_{sup}$, $min_{conf}$

**Output:** newly generated association rules

**Method:**

Step 1: Generate new frequent 1-itemsets $L_{n1}$ using VTV table and taxonomy tree.

Step 2: Create association graph AG using original frequent itemsets $L_{o1}$ and $L_{o2}$, and new frequent 1-itemsets $L_{n1}$.

Step 3: For each edge $x\rightarrow y$ in association graph AG where $\{x, y\}$ is not in $L_{o2}$

- Add the rules $x \Rightarrow y$ and $y \Rightarrow x$ to $R_{n2}$.
- If support($\{x, y\}$)/support($x$) $\geq min_{sup}$, set the rule $x \Rightarrow y$ strong.
- If support($\{x, y\}$)/support($y$) $\geq min_{sup}$, set the rule $y \Rightarrow x$ strong.

Step 4: For ($k = 3; R_{n(k-1)} = \emptyset; k++)$

- Generate weak ($k$-1)-association rules using the original frequent ($k-1$)-itemsets $L_{o(k-1)}$ and add them to $R_{n(k-1)}$.

**Procedure GMARGen($R_o(k-1), R_{n(k-1)})$**

1. For each rule $r_1 \in R_o(k-1)$, for each rule $r_2 \in R_{n(k-1)}$:

   - If ($r_1.\text{antecedent} \neq r_2.\text{antecedent}$) and ($r_1.\text{consequent} \neq r_2.\text{consequent}$) and ($\text{conf}(r_1) \geq min_{conf}$) and ($\text{conf}(r_2) \geq min_{conf}$)

     - Using PT 5 and PT 6

     - If all the items in $\{r_1.\text{antecedent} \cup r_1.\text{consequent} \cup r_2.\text{consequent}\}$ can form a complete connection in association graph AG

     - Using PT 2 and PT 4

     - $r = r_1.\text{antecedent} \Lambda r_2.\text{antecedent} \Rightarrow r_1.\text{consequent}$

   - If $r_1.\text{antecedent} = r_2.\text{antecedent}$ and ($r_1.\text{consequent} = r_2.\text{consequent}$)

     - Using PT 3 and PT 6

     - If all the items in $\{r_1.\text{antecedent} \cup r_1.\text{consequent} \cup r_2.\text{consequent}\}$ can form a complete connection in association graph AG

     - Using PT 2 and PT 4

     - $r = r_1.\text{antecedent} \Rightarrow r_1.\text{consequent}$

   - If ($r_1.\text{antecedent} \neq r_2.\text{antecedent}$) and ($r_1.\text{consequent} = r_2.\text{consequent}$)

     - Using PT 5 and PT 6

     - If all the items in $\{r_1.\text{antecedent} \cup r_1.\text{consequent} \cup r_2.\text{consequent}\}$ can form a complete connection in association graph AG

     - Using PT 2 and PT 4

     - $r = r_1.\text{antecedent} \Lambda r_2.\text{consequent} \Rightarrow r_1.\text{consequent}$

   - Set the rule $r$ strong.

Step 5: $R_n = \cup_{k \geq 1} \{r \in R_{n(k)}$ and $r$ is a strong rule.$\}$

Given the same database as the example for the GMFI algorithm and original association rules $R_o$ (including the strong rules in $R_{o2}$ and $R_{o3}$), new generalized association rules $R_n$ (including the strong rules in $R_{n2}$ and $R_{n3}$) can be generated using the GMAR algorithm as follows. Among the rules, a rule is marked if it is a strong one.
Fig. 7. New frequent itemsets.

Fig. 8. GMAR algorithm to generate new generalized association rules.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Confidence</th>
<th>Confidence</th>
<th>Confidence</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ⇒ 2, 1003</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 2, 1004</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 2, 1005</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 3, 1001</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 3, 1002</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 3, 1003</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 3, 1004</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 3, 1005</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 4, 1001</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 4, 1002</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 4, 1003</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 4, 1004</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 4, 1005</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 5, 1001</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 5, 1002</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 5, 1003</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 5, 1004</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>1 ⇒ 5, 1005</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 3, 1001</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 3, 1002</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 3, 1003</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 3, 1004</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 3, 1005</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 4, 1001</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 4, 1002</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 4, 1003</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 4, 1004</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 4, 1005</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 5, 1001</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 5, 1002</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 5, 1003</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 5, 1004</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2 ⇒ 5, 1005</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 4, 1001</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 4, 1002</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 4, 1003</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 4, 1004</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 4, 1005</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 5, 1001</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 5, 1002</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 5, 1003</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 5, 1004</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>3 ⇒ 5, 1005</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>4 ⇒ 5, 1001</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>4 ⇒ 5, 1002</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>4 ⇒ 5, 1003</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>4 ⇒ 5, 1004</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>4 ⇒ 5, 1005</td>
<td>0.333333</td>
<td>1</td>
<td>0.333333</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 4, 1001</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 4, 1002</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 4, 1003</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 4, 1004</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 4, 1005</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 5, 1001</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 5, 1002</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 5, 1003</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 5, 1004</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5 ⇒ 5, 1005</td>
<td>0.75</td>
<td>1</td>
<td>0.75</td>
<td>1</td>
</tr>
</tbody>
</table>
Finally, the new generalized association rules are the same as those shown in Fig. 7.

4. Performance evaluations

4.1. Simulation model

In this section, we evaluate the performances of the four algorithms, including BASIC [15], Cumulate [15], GMAR, and GMFI. All the experimental data are generated from the IBM synthetic data generator and stored on a local 30GB SCSI Disk (Ultra 160) with a RAID controller. The relative simulation parameters are shown in Table 1. To make our data representative, we generated two types of databases in the experiments; i.e., DENSE databases and SPARSE databases. Each item in the DENSE database was randomly generated from a pool P called potentially frequent itemsets with size 300, while each item in the SPARSE database was randomly generated from a pool N (i.e., the set of all the items) with size 1000. Since the items in the DENSE database are more clustered than those in the SPARSE database, larger frequent itemsets will probably be produced in the DENSE database for the same minimum support. In addition, we use the notation T for the average number of items per transaction, I for the average number of items in a frequent itemset, and D for the number of transactions.

Table 1: Simulation parameters with default values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>1000-500,000</td>
</tr>
<tr>
<td>T</td>
<td>5-15</td>
</tr>
<tr>
<td>P</td>
<td>300</td>
</tr>
<tr>
<td>I</td>
<td>2-5</td>
</tr>
<tr>
<td>N</td>
<td>1000</td>
</tr>
<tr>
<td>R</td>
<td>31-75</td>
</tr>
<tr>
<td>L</td>
<td>3</td>
</tr>
</tbody>
</table>

4.2. Experimental results

4.2.1. Experiment 1

In the experiment, we explored the execution time of BASIC, Cumulate, GMAR, and GMFI algorithms for the environment T10.I3.D1K under different minimum support and minimum confidence pairs (Fig. 9). In this figure, we find that our algorithms GMAR (or GMFI) are almost 2–16 (or 4.5–7) times faster than BASIC, especially for larger minimum support and minimum confidence pairs. Cumulate is only 1.3–1.5 times faster than BASIC, although R. Srikant and R. Agrawal claimed that Cumulate runs 2–5 times than faster than BASIC [15]. In general, the larger the minimum support and minimum confidence pair is, the faster the execution time of the four algorithms becomes. Besides, GMFI performs better than GMAR for smaller minimum support and minimum confidence pairs, but the situation is different for larger pairs. This is because GMAR uses join methods to produce rules in higher levels, and also uses pruning techniques to eliminate irrelevant rules. When the minimum confidence is
higher than a threshold, a large amount of rules will be pruned and the execution time is dramatically reduced. To be fair to all algorithms, we have added the additional time of generating original frequent itemsets and association rules for GMAR and GMFI. However, the time is below 1% of total execution time and therefore we do not show it in the figure.

Moreover, we also explored the relationship between the number of candidate itemsets generated in the algorithms and their performances (Table 2). From Table 2, we can understand why many researchers are trying to propose new techniques to reduce the size of candidate itemsets. In this experiment, GMAR generated the fewest candidate itemsets.

4.2.2. Experiment 2

In this experiment, we extend Experiment 1 by fixing the minimum support 1.5%, and observed their variations. For the minimum support 1.5%, all the algorithms except GMAR were not sensitive to the changes of the minimum confidences. This is shown in Fig. 10. The reason for this is that larger minimum confidences will make GMAR prune more irrelevant rules. Nevertheless, GMAR is still in the first rank and GMFI is in the second one.

4.2.3. Experiment 3

In this experiment, we explored the execution time of the four algorithms for the environment T10.I3.DxK (i.e., different numbers of transactions) generated in the SPARSE database and in the DENSE database (see Fig. 11(a) and (b), respectively). Both cases have the same minimum confidence 0.3. However, to get a comparable number of frequent itemsets, we set a smaller minimum support 1% in the SPARSE case, and a larger minimum support 2% in the DENSE case. As expected, GMAR is still the best one among them in the SPARSE case, especially when there are a huge amount of transactions. However, GMFI becomes the best one in the DENSE case. From both cases, we find that many more frequent itemsets are generated in the DENSE database.

<table>
<thead>
<tr>
<th>min_sup (%)</th>
<th>min_conf</th>
<th>GMAR</th>
<th>GMFI</th>
<th>Cumulate</th>
<th>BASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3</td>
<td>0.26</td>
<td>893</td>
<td>11158</td>
<td>18395</td>
<td>6107930</td>
</tr>
<tr>
<td>1.4</td>
<td>0.28</td>
<td>789</td>
<td>9728</td>
<td>15389</td>
<td>5325862</td>
</tr>
<tr>
<td>1.5</td>
<td>0.3</td>
<td>689</td>
<td>8535</td>
<td>12760</td>
<td>4668166</td>
</tr>
<tr>
<td>1.6</td>
<td>0.32</td>
<td>631</td>
<td>7682</td>
<td>11203</td>
<td>4314549</td>
</tr>
<tr>
<td>1.7</td>
<td>0.34</td>
<td>591</td>
<td>7552</td>
<td>10475</td>
<td>4138531</td>
</tr>
<tr>
<td>1.8</td>
<td>0.36</td>
<td>545</td>
<td>7201</td>
<td>9638</td>
<td>3953027</td>
</tr>
</tbody>
</table>

Fig. 10. Execution time for different minimum confidences.
than in SPARSE database, so that BASIC and Cumulate are not practicable candidates there.

4.2.4. Experiment 4

In this experiment, we explored the execution time of the four algorithms in very large databases, such as the environment T5.I2.D500K and T15.I5.D500K under different minimum support and minimum confidence pairs (Fig. 12). In the figures, we find that our algorithms GMAR (or GMFI) still perform much better than BASIC and Cumulate algorithms, especially for large T and large I.

5. Conclusions

In this paper, we tried to find the association rules between items at different levels in the taxonomy tree under the assumption that original frequent itemsets and association rules have already been generated in advance. The primary challenge is how to make use of the original frequent itemsets and association rules to directly generate new generalized association rules, rather than rescanning the database. In the proposed algorithms GMAR and GMFI, we used join methods and/or pruning techniques to generate new generalized association rules.
Through several comprehensive experiments, we found that GMAR and GMFI algorithms were much better than BASIC and Cumulate algorithms, since they generated fewer candidate itemsets, and furthermore the GMAR algorithm pruned a large amount of irrelevant rules based on the minimum confidence. Concerning GMAR and GMFI algorithms, we find that GMAR is always better than GMFI in a SPARSE database. However, GMFI can be better than GMAR in a DENSE database if the amount of frequent itemsets is large enough (i.e., the minimum support is small enough).

References


