A lattice-based approach for mining most generalization association rules

Bay Vo a,*, Tzung-Pei Hong b,c, Bac Le d

a Scientific Research and IT Department, Information Technology College, Ho Chi Minh, Viet Nam
b Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan, ROC
c Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, ROC
d Department of Computer Science, University of Science, Ho Chi Minh, Viet Nam

A R T I C L E   I N F O

Article history:
Received 19 December 2011
Received in revised form 31 January 2013
Accepted 1 February 2013
Available online 13 February 2013

Keywords:
Association rule
Data mining
Frequent closed itemset
Lattice
Most generalization association rule

A B S T R A C T

Traditional association rules consist of some redundant information. Some variants based on support and confidence measures such as non-redundant rules and minimal non-redundant rules were thus proposed to reduce the redundant information. In the past, we proposed most generalization association rules (MGARs), which were more compact than (minimal) non-redundant rules in that they considered the condition of equal or higher confidence, instead of only equal confidence. However, the execution time for generating MGARs increased with an increasing number of frequent closed itemsets. Since lattices are an effective data structure widely used in data mining, in this paper, we thus propose a lattice-based approach for fast mining most generalization association rules. Firstly, a new algorithm for building a frequent-closed-itemset lattice is introduced. After that, a theorem on pruning nodes in the lattice for rule generation is derived. Finally, an algorithm for fast mining MGARs from the lattice constructed is developed. The proposed algorithm is tested with several databases and the results show that it is more efficient than mining MGARs directly from frequent closed itemsets.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Mining association rules is an important task in data mining and knowledge discovery [1]. They have wide applications, such as basket data analysis [1], semantic web mining [13], text mining [32], and among others. In the past, some methods have been proposed for mining association rules, such as traditional association rules [1], non-redundant association rules [29,30], and minimal non-redundant association rules [2]. Given a transaction database, mining traditional association rules is to generate all rules that their supports satisfy minimum support threshold (minSup) and their confidences satisfy minimum confidence threshold (minConf). An association rule \( R: X \rightarrow Y \) is called a minimal non-redundant association rule if and only if there does not exist an association rule with the same support value and confidence value as \( R \), but with a more specific antecedent part and a more general consequent part. Although the methods for mining association rules are different, their processing is nearly the same. Their mining processes are usually divided into the following two phases:

(i) Mining frequent itemsets (FIs) or frequent closed itemsets (FCIs).

(ii) Mining traditional association rules or (minimal) non-redundant association rules from FIs or FCIs.

Traditional association rules generated a lot of redundant. Some approaches have thus been proposed to reduce the number of rules and increase the rule usefulness for users [2,14,15,29,30]. Although these approaches may generate fewer rules than the traditional approach, the number of rules is still large. For example, for the Chess database with \( \text{minSup} = 70\% \) and \( \text{minConf} = 0\% \), the number of rules generated by Zaki’s method [30] is 152,074 and that by Bastide et al.’s method [2] is 3,373,625. In fact, in a mined knowledge set, some rules may be inferred from some other rules.

For illustrating this problem, we consider an example database [31] as in Table 1:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Table 1: Example Database

If minimal non-redundant rules are mined with \( \text{minSup} = 50\% \) and \( \text{minConf} = 80\% \), the results are nine rules as \( \{ D \supseteq C, T \supseteq C, W \supseteq C, A \supseteq CW, DW \supseteq C, AT \supseteq W, TW \supseteq A, W \supseteq AC, W \supseteq AC \} \). We can see that some of them are redundant according to their supports and confidences. For example, the rule \( DW \supseteq C \) is a weaker rule because there exist some stronger rules such as \( D \supseteq C \) and \( W \supseteq C \). The rule \( AT \supseteq W \) is also a weaker rule because there is a stronger rule \( A \supseteq CW \). Similarly, the rule \( W \supseteq AC \) will not be kept since its more general rule \( W \supseteq AC \) has been generated. The two previous methods (non-redundant association rules and minimal non-redundant association rules) will not prune the above rules.

Based on the consideration above, it is thus necessary to have a method for pruning these weak rules. We thus proposed an approach called mining the most generalization association rules [25] to generate a compact rule set. A most generalization
association rule is different from a minimal non-redundant rule in that the former considers the condition of equal or higher confidence, instead of only equal confidence. That is, an association rule \( R: X \rightarrow Y \) is a most generalization rule if and only if there does not exist an association rule with a higher confidence value than \( R \), but with a more specific antecedent part and a more general consequent part. They showed that the number of MGARs was smaller than those of the non-redundant association rules [30] and the minimal non-redundant association rules [2]. We also developed some theorems for fast pruning a lot of rules directly. The remaining rules differ from the previous hybrid ones in that they do not check subsumption. Unlike CHARM [31], they do not use the hash-table technique. DCI-Close [11], LCM [20], and PGMiner [12] belong to this category.

### 2.2. Mining minimal non-redundant association rules

Mining minimal non-redundant association rules (MNARs) was first proposed by Pasquier et al. in 1999 [14,15]. To better illustrate their approach, a generator of a frequent closed itemset \( X \) is first defined as follows. Let \( X \) be a frequent closed itemset. An itemset \( X \neq \emptyset \) is called a generator of \( X \) if and only if \( X \subseteq X \) and \( \sigma(X) = \sigma(X') \). Let \( G(X) \) denotes the set of all the generators of \( X \). An itemset \( X' \subseteq G(X) \) is called a minimal generator of \( X \) if \( X' \) has no proper subset in \( G(X) \). Let \( mG(X) \) denotes the set of all the minimal generators of \( X \).

An association rule is represented by \( R: X \rightarrow Y \), where \( X, Y \) are frequent itemsets \( (X \subseteq Y) \), sup is the support of \( R \), and conf is the confidence of \( R \). The scores of sup and conf are calculated as follows:

\[
\text{sup}(R) = \sigma(Y), \quad \text{conf}(R) = \frac{\sigma(X)}{\sigma(Y)}.
\]

The rule is also usually represented by \( R: X \rightarrow Y \setminus X \) for convenience. For example, an association rule \( AB \rightarrow ABC \) is usually represented as \( AB \rightarrow C \).

An association rule \( R_1: X_1 \rightarrow Y_1 \) is a minimal non-redundant association rule if and only if there does not exist an association rule \( R_2: X_2 \rightarrow Y_2 \) with \( \sigma(R_1) = \sigma(R_2) \), \( \text{conf}(R_1) = \text{conf}(R_2) \), \( X_2 \subseteq X_1 \), and \( Y_1 \subseteq Y_2 \). Based on this definition, their approach first mines all FCIs by computing the closure of each minimal generator, which is the largest itemset with the same support as the minimal generator. It then finds MNARs from the FCIs. There are two kinds of MNARs obtained:

1. Exact rules (confidence = 100%): the rules have the form \( X' \rightarrow X \), where \( X \) is an FCI and \( X' \subseteq mG(X) \).
2. Approximate rules (confidence < 100%): the rules have the form \( X' \rightarrow Y \), where X and Y are FCIs, and \( X' \subseteq mG(X), X \subseteq Y \).

In 2000, Zaki proposed a method to mine non-redundant association rules (NARs) based on FCIs and their \( mG \) [29]. There are two kinds of candidate NARs:

1. NARs with their confidence = 100%: the following rules are generated from this kind of NAR.
   - Self-rules: the rules have the form \( X' \rightarrow X \), \( X' \subseteq mG(X) \), and \( X \) is an FCI. This kind of rules is the same as the exact rules of MNARs;
   - Down-rules: the rules have the form \( Y \rightarrow X' \), where \( Y \subseteq mG(Y), X' \subseteq mG(X), X \) and Y are FCIs, and \( X \subseteq Y \).
2. NARs with their confidence < 100%: the rules have the form \( X' \rightarrow Y \), where \( X' \subseteq mG(X), X' \subseteq mG(Y), X \) and Y are FCIs, and \( X \subseteq Y \).

The interested readers may refer to [30] for more details.
2.3. Mining the most generalization association rules

We defined the concept of MGARs and proposed an algorithm for generating MGARs based on FCIs [25]. A generalization association rule is introduced as follows. Assume that there are two rules $R_1: X_1 \rightarrow Y_1$ and $R_2: X_2 \rightarrow Y_2$. Rule $R_1$ is said to be more general than $R_2$, denoted as $R_1 \prec R_2$, if and only if $X_1 \subseteq X_2$ and $Y_2 \subseteq Y_1$. We then defined the precedence order among the rules. Let $R = \{R_1, R_2, \ldots, R_n\}$ be the set of rules that satisfy the conditions of a minimum support and a minimum confidence. A rule $R_i$ is said to have a higher precedence than another rule $R_j$, denoted as $R_i \succ R_j$, if $R_i \succ R_j$ and one of the following conditions holds:

1. the confidence of $R_i$ is greater than that of $R_j$;
2. their confidence values are the same, but the support of $R_i$ is greater than or equal to that of $R_j$.

The set $(R_{MC})$ of the most generalization association rules of $R$ is then defined as follows:

$$R_{MC} = \{R_i \in R | \forall R_j \in R : R_i \succ R_j\}.$$

According to the above definition, only the association rules that have the highest precedence will be kept. That is, a rule $R_i$ in $R$ will be pruned if $R$ has another rule $R_k$ such that $R_i \succ R_k$. Our previous algorithm for mining MGARs considers each frequent closed itemset $X$ with all the frequent closed itemsets having the highest precedence will be kept. This list is not sorted according to length. Since the mining approach proposed here for mining MGARs must traverse FCIs in an increasing order of length, an appropriate lattice has to be built up from the FCIs. Let the FCIs be first sorted in an increasing order of length. A simple approach for building the lattice is to consider each FCI $X$ and its succeeding FCIs. If $Y$ is a succeeding FCI of $X$ and $X \subseteq Y$, $Y$ is then added to the set of child nodes of $X$. The complexity of the approach is $O(|FCIs|^2)$. When the number of frequent closed itemsets is large, the approach requires a lot of computational time.

2.4. Building lattices

There are a lot of algorithms for building lattices [4,7–9,17,18,20,22,23,26,27,31]. They can be divided into two categories:

(i) Concept lattice: Kourie et al. presented an incremental method and two variants for building concept lattices [7]. Kuznetsov and Obiedkov gave an overview of formal concept analysis (FCA) and compared the performance of some algorithms for generating concept lattices [8]. Liu et al. proposed a reduction method for concept lattices based on the rough-set theory [9]. Priss presented an application of concept lattices to information retrieval [17]. More details about FCA and concept lattices are given in [4,18].

(ii) Frequent-itemset lattice (FIL) and frequent-closed-itemset lattice (FCIL): Zaki and Hsiao proposed CHARM-L [31], which is an extension of CHARM, for building a frequent-closed-itemset lattice. Vo and Le presented an extension of the Eclat algorithm [31] for building a frequent-itemset lattice (FIL) [27]. A modification of the frequent-itemset lattice for mining MNARs was presented in [26].

In this paper, we extend the lattice-based approach for quickly mining MGARs. An algorithm for building FCIL is described in the next section. After the FCIL is obtained, a mining approach for MGARs based on the obtained FCIL is designed.

2.5. Mining rules from frequent-itemset lattice

In the past, Vo and Le proposed an algorithm for mining association rules from frequent itemsets lattice [27]. The algorithm utilized the relation between two nodes in a lattice for fast traversing all child nodes of a given node. It was more efficient than direct mining from frequent itemsets (using hash table) [5]. Vo and Le then modified the frequent-itemset lattice and used it for mining MNARs [26].

3. Building a lattice from frequent closed itemsets

3.1. Concept and algorithm

As mentioned above, Zaki and Hsiao [31] proposed the CHARM-L algorithm for building an FCI. When an FCI is traversed to generate rules, the order of itemsets may not be listed according to length. For example, consider the database shown in Table 1. It consists of six transactions and five items.

The FCI built from the database in Table 1 with $\minSup = 50\%$ is shown in Fig. 1. If we traverse it using the depth-first search (DFS), the list of FCIs is $\{C, CD, CDW, CT, ACTW, CW, ACW\}$. If we traverse it using the breadth-first search (BFS), the list of itemsets is not guaranteed to be sorted. This concept can be further illustrated by the example in Fig. 2 [22].

In Fig. 2, if we traverse the lattice using the BFS, the result is $\{d, c, g, f, cd, bc, cf, fgh, fh, ef, bcd, cdgh, abc, abcef, efgh, efh, abcd, abcdefgh\}$. This list is not sorted according to length. Since the mining approach proposed here for mining MGARs must traverse FCIs in an increasing order of length, an appropriate lattice has to be built up from the FCIs. Let the FCIs be first sorted in an increasing order of length. A simple approach for building the lattice is to consider each FCI $X$ and its succeeding FCIs. If $Y$ is a succeeding FCI of $X$ and $X \subseteq Y$, $Y$ is then added to the set of child nodes of $X$. The complexity of the approach is $O(|FCIs|^2)$. To build the lattice faster, another approach is designed here. Assume that there is an FCI $Y$ being inserted into a temporary lattice $L_r$, which has been built by the FCIs proceeding $Y$. Only the set of nodes in $L_r$ with each node $X \subseteq Y$ has to be traversed to find the correct positions for linking node $\{Y\}$ to the set. Based on the above idea, the proposed algorithm for building a lattice from FCIs is shown in Fig. 3.

There is a main procedure in the algorithm, BUILD-LATTICE(), which incrementally inserts all FCIs into the lattice by another algorithm and two variants for building concept lattices [7]. Kuznetsov and Obiedkov gave an overview of formal concept analysis (FCA) and compared the performance of some algorithms for generating concept lattices [8]. Liu et al. proposed a reduction method for concept lattices based on the rough-set theory [9]. Priss presented an application of concept lattices to information retrieval [17]. More details about FCA and concept lattices are given in [4,18].

Table 1

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, C, T, W</td>
</tr>
<tr>
<td>2</td>
<td>C, D, W</td>
</tr>
<tr>
<td>3</td>
<td>A, C, T, W</td>
</tr>
<tr>
<td>4</td>
<td>A, C, D, W</td>
</tr>
<tr>
<td>5</td>
<td>A, C, D, T, W</td>
</tr>
<tr>
<td>6</td>
<td>C, D, T</td>
</tr>
</tbody>
</table>

Fig. 1. The FCIL built from the database in Table 1 with $\minSup = 50\%$. 

procedure INSERT_LATTICE(Y, l), where Y is an FCI to be inserted and l is a node in the lattice from which the procedure begins execution. The attribute isTraverse is used for determining whether the node l is traversed, and the variable flag is used to decide whether the node \{Y\} can be directly connected to the current node l. In Fig. 3, there are two possible cases in the procedure INSERT_LATTICE(Y, l).

(i) If Y is the superset of any X with \{X\} ∈ L.Children (lines 8 and 9), then \{Y\} is a child node of \{X\}. The procedure will be recursively called to insert \{Y\} into the lattice with \{X\} as the root node (line 11).

(ii) If Y is not a superset of any X with \{X\} ∈ L.Children, then \{Y\} is a direct child node of the node l. \{Y\} is then directly added into the list of the child nodes of l (lines 10, 12, and 13).

Procedure RESET_LATTICE(L_r) resets the value of the variable isTraverse to be false for all nodes in the temporary lattice L_r.

3.2. An example

An example is given here to illustrate the above lattice building algorithm. Assume that the FCIs shown in Table 2 are found using the CHARM algorithm [31] for the example in Table 1 with minSup = 50%.

According to the lattice-building algorithm shown in Fig. 3, the FCIL from the FCIs in Table 2 is constructed as shown in Fig. 4.

The process is as follows. Firstly, the root node of the lattice is assigned to ∅ (or \{\}). Then, the first FCI, C, in Table 2 is processed. It is directly connected to the root node {} (Fig. 4a) according to the algorithm. The next FCI CD is then processed. Since the root node has child nodes \{C\} and \{CD\} and \{CD\} is not \{C\}, the algorithm then recursively calls the procedure to insert \{CD\} into the sub-lattice with the root node of \{C\}. \{CD\} is then directly connected to the node of \{C\} (Fig. 4b) because \{C\} has no children. Similar processing is applied to CT, CW, ACW, and CDW (Fig. 4c–f). When the last FCI, which is \{ACTW\}, is inserted into the lattice in Fig. 4f, the child node of \{C\} is considered and the procedure is recursively called again. \{ACTW\} is then inserted into the sub-lattice with the root node \{C\}. The three child nodes (\{CD\}, \{CT\}, and \{CW\}) of \{C\} are then checked. Since \{CT\} is the branch that stops, \{ACTW\} is then connected to \{CT\} as its child. At last, the branch \{CW\} is considered. It has two child nodes, \{CDW\} and \{ACW\}. According to the algorithm, \{ACTW\} is connected to \{ACW\} as its child.
Theorem 1. Given three nodes l₁, l₂, and l₃ in FCIL, if l₁ is the parent node of l₂, l₂ is the parent node of l₃, and l₃ is the parent node of {l₁, l₂}, then \( \frac{\text{sup}(l₁)}{\text{sup}(l₂)} < \text{minConf} \). Thus, \( \frac{\text{sup}(l₁)}{\text{sup}(l₂)} \geq \frac{\text{sup}(l₂)}{\text{sup}(l₃)} \). Since \( \frac{\text{sup}(l₁)}{\text{sup}(l₂)} < \text{minConf} \), it implies \( \frac{\text{sup}(l₂)}{\text{sup}(l₃)} < \text{minConf} \).

Proof. Since l₁ is the parent node of l₂, l₂ is the parent node of l₃, and l₃ is the parent node of \{l₁, l₂\}, this implies that \( \frac{\text{sup}(l₁)}{\text{sup}(l₂)} \geq \frac{\text{sup}(l₂)}{\text{sup}(l₃)} \). Since \( \frac{\text{sup}(l₁)}{\text{sup}(l₂)} < \text{minConf} \), it implies \( \frac{\text{sup}(l₂)}{\text{sup}(l₃)} < \text{minConf} \). □

According to Theorem 1, if a lattice node \{Y\} is a child node of \{X\} in the FCIL and \( \frac{\text{sup}(X)}{\text{sup}(Y)} \) < minConf, then the child nodes of \{Y\} cannot form rules with \{X\}. For example, consider the two nodes \{C\} and \{CW\} in Fig. 1. Assume that \( \text{minConf} = 90\% \). Since \( \sigma(CW)/\sigma(C) = 5/6 < \text{minConf} \), the child nodes \{ACW\} and \{ACTW\} of \{CW\} in the FCIL will not be considered as candidates.

4.1. Algorithm for generating MGARs from FCIL

The MG-CHARM algorithm [24] is adopted to efficiently mine FCIs and their minimal generators. The algorithm in Fig. 3 is used to build the FCIL. The proposed algorithm, shown in Fig. 5, then finds MGARs based on the FCIL.

Firstly, the algorithm traverses all the FCI ∈ FCIs. For each FCI C, it then initializes the RHS (right-hand side) to ∅ (line 3). It then generates rules from the minimal generators of FCI C to C by calling the FIND_RULES procedure (line 4). Then it calls the EXTEND_MGARs_LATTICE procedure (line 5). This procedure uses a queue to traverse all the child nodes of \{C\} (and marks all the visited nodes to avoid coincidence). For each child node \( Lₖ \) of \{C\} (line 13), the confidence (line 14) of all rules with the form \( X' \rightarrow Lₖ \times \{X' \in \text{mg}(C)\} \) is then calculated. If the confidence satisfies \( \text{minConf} \) and \( Lₖ \) is not marked (line 15), then \( Lₖ \) is added to Queue (line 16) for generating all rules from C to \( Lₖ \) (line 12). After \( Lₖ \) is added to Queue, it is marked to avoid coincidence in the future (line 17).

4.2. An example

Consider the FCIL in Fig. 4.g with \( \text{minConf} = 60\% \). The process of generating MGARs from node \{C\} of the lattice is as follows. First, the RHS is initialized to ∅. The algorithm then calls FIND_RULES([C], [C], RHS) for generating rules with confidence 100%. Because \( Z = C \setminus C = ∅ \), no rule is added to \( \text{RMSG} \). Next, the child nodes of \{C\} are \{[CD], [CT], [CW]\}, and they are added to Queue and marked. Because Queue ≠ ∅, an element is obtained to be Z from Queue. Thus \( Z = \{CD\} \) (and Queue = \{[CT], [CW]\}). Next, because \( \text{conf}(C \rightarrow CD) = \sigma(CD)/\sigma(C) = 4/6 > \text{minConf} \), the procedure FIND_RULES([C], [CD], RHS) is called. This procedure

### Table 3

<table>
<thead>
<tr>
<th>Database</th>
<th>minSup (%)</th>
<th># of FCIs</th>
<th>k – [avg subsets]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>85</td>
<td>1885</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>5083</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>11,525</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>23,892</td>
<td>141</td>
</tr>
<tr>
<td>Mushroom</td>
<td>30</td>
<td>427</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>688</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1200</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>2265</td>
<td>27</td>
</tr>
<tr>
<td>Connect</td>
<td>96</td>
<td>481</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>1237</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>92</td>
<td>2212</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>3486</td>
<td>91</td>
</tr>
<tr>
<td>Accidents</td>
<td>80</td>
<td>149</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>529</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>2074</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>8057</td>
<td>47</td>
</tr>
<tr>
<td>Pumsb</td>
<td>80</td>
<td>68</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>55</td>
<td>116</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>248</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>713</td>
<td>16</td>
</tr>
<tr>
<td>Pumsb</td>
<td>94</td>
<td>216</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>92</td>
<td>610</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>1465</td>
<td>20</td>
</tr>
<tr>
<td>Retail</td>
<td>1</td>
<td>159</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>243</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>417</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>831</td>
<td>3</td>
</tr>
</tbody>
</table>
generates the rule $C^4\bar{4}/D$ and adds it to $R_{MG}$. $R_{MG}$ becomes $\{C^4\bar{4}/D, C^4\bar{4}/T\}$ and $RHS = DT$. $(CT)$ has one child node $(ACTW)$ but $conf(C\rightarrow ACTW) = \sigma(CTW)/\sigma(C) = 3/6 < minConf$, so it is not added to $Queue$. Because $Queue = \emptyset$, $[Y] = [CW]$, and $conf(C\rightarrow CW) = \sigma(CW)/\sigma(C) = 5/6 \geq minConf$, the procedure $FIND\_RULES([C], [CW], RHS)$ is called. This procedure generates the rule $C^4\bar{4}/W$, and adds it to $R_{MG}$. Thus, $R_{MG} = \{C^4\bar{4}/D, C^4\bar{4}/T, C^5\bar{4}/W\}$ and $RHS = DTW$. $(CW)$ has one child node $(ACW)$ and $conf(C\rightarrow ACW) = 4/6 > minConf$. It is then added to $Queue$, and $Queue = ([ACW])$. Because $Queue = \emptyset$, $[Y] = [ACW]$, and $conf = \sigma(ACW)/\sigma(C) = 4/6 \geq minConf$, the procedure $FIND\_RULES([C], [ACW], RHS)$ is executed. This procedure generates the rule $C^3\bar{4}/ACW = (C\downarrow DTW)$, and adds it to $R_{MG}$. $R_{MG} = \{C^4\bar{4}/D, C^4\bar{4}/T, C^5\bar{4}/W, C^4\bar{4}/A\}$, and $RHS = DTWA$. $(ACW)$ has one child node $(ACT)$, but it is marked. It is thus not added to $Queue$. Finally, because $Queue = \emptyset$, the procedure $(EXTRACT\_MG\_FROM\_LATTICE)$ is stopped.

5. Experimental results

Experiments were conducted to show the performance of the algorithms. They were implemented on a Centrino Core 2 Duo $(2 \times 2.53$ GHz) PC with 4 GB of RAM and running Windows 7. The algorithms were coded in C# 2008. Seven databases from [6] were used for the experiments; their features are shown in Table 4.

5.1. The execution time

Experiments were made to compare the execution time of the following two algorithms $MG\_FCIL$ (mining $MG\_ARes$ using $FClIs$ [25]), and $MG\_FCIL$ (mining $MG\_ARes$ using $FClIL$). The $minConf$ was set to 50%. The results of the seven databases for various $minSup$ values are shown in Figs. 6–13.

The results from the above figures show that the execution of $MG\_FCIL$ was faster than $MG\_ARes$-$FCIL$ [25] in all the above cases. For example, consider the Chess database with $minSup = 70\%$. The mining time of $MG\_FCIL$ and $MG\_ARes$-$FCIL$ were 73.06(s) and 314(s), respectively. The time ratio is $73.06/314 = 0.2327$. Besides, when we decrease the $minSup$, the time ratio will reduce as well. For example, consider the Mushroom database with $minSup$ set at 30%, 25%, 20% and 15%
respectively. The speed-ups were 93.94, 72.32, 70.09 and 56.07, respectively.

If only the time of mining rules is measured (without considering the time of mining FCIs and building lattice), the results for the seven databases are shown in Figs. 13–19.

The results show that the execution time of mining rules from FCIL was very efficient. For example, consider the Chess database with $\text{minSup} = 70\%$. The mining time of MGARs-FCIL was 19.63, and of MGARs-FCIs was 313.03(s), respectively. The time ratio was only 6.27%.

The experiment then was made to compare the proposed algorithm with the algorithm for mining minimal non-redundant association rules using frequent itemsets lattice (FIL) [25]. Figs. 20–26 show the results without FCIs mining and building lattice.

Mining MGARs from FCIL is faster than mining MNARs from FIL in some results such as Mushroom and Connect. This is because the
ratio between the number of frequent itemsets and frequent closed itemsets is large. For example, consider the Mushroom database with \( \text{minSup} = 20\% \), the number of frequent itemsets is 53,583, while the number of frequent closed itemsets is 1200. The ratio is \( \frac{1200}{53,583} \times 100\% \approx 2.24\% \). However, when the ratio is small, mining MNARs based on FIL is faster than mining MGARs based on FCIL. For example, consider the Chess database with \( \text{minSup} = 70\% \), the number of frequent itemsets is 48,731, while the number of frequent closed itemsets is 23,892. The ratio is \( \frac{23,892}{48,731} \times 100\% \approx 49.03\% \).

Fig. 13. Execution time of the two algorithms in Chess without FCIs mining and building lattice.

Fig. 14. Execution time of the two algorithms in Mushroom without FCIs mining and building lattice.

Fig. 15. Execution time of the two algorithms in Connect without FCIs mining and building lattice.

Fig. 16. Execution time of the two algorithms in Accidents without FCIs mining and building lattice.

Fig. 17. Execution time of the two algorithms in Pumsb without FCIs mining and building lattice.

Fig. 18. Execution time of the two algorithms in Pumsb without FCIs mining and building lattice.
5.2. The memory usage

Because mining MGARs is based on FCIs, both FCIs-based and FCIL-based have the same rule set. The memory for storage them (FCIs and rules) is the same, too. Besides, using lattice must store the link between parent and child nodes. Therefore, mining MNARs using FCIL will consume more memory than only using FCIs. Assume that each link used 4 bytes, the memory usage for each experimental result is shown in Table 5.

Results from Table 5 show that the memory usage for mining MGARs using FCIL is little higher than that of using FCIs. The
maximum difference gets to 0.532 MBs (Chess database with minSup = 70%).

Note that association rule mining seldom considers target attributes; instead, it finds the association relationships between items or itemsets. If we use the items in the if-part of an association rule as the condition and in the then-part as the target, then the precision is just the confidence of the rule. It is certainly equal or larger than the given minimum confidence value.

5.3. Numbers of rules

We compare the numbers of MGARs with those obtained by traditional association rules [1], non-redundant association rules (NARs) [30], and minimal non-redundant association rules (MNARs) [2] in Table 6 (We do not compare them in Accidents and Retail databases because authors in [30] did not report the numbers of rules in these databases). The results show that the number of MGARs is always smaller than those of NARs, MNARs, and traditional association rules. Consider Pumsb database with minSup = 95% and minConf = 0% as an example, the number of MGARs is 101, whereas the number of NARs is 267, the percentage of #MGARs per #NARs is 37.83%; the number of MNARs is 690, and the percentage is 14.64%; the number of traditional association rules is 1170, and the percentage is 8.63%.

6. Conclusions and future work

In this paper, we have proposed an effective approach for mining most generalization association rules from transaction databases. It utilizes FCIL to quickly find all pairs \( \{X, Y\} \), in which both \( X \) and \( Y \) are FCIs and \( X \land C \subseteq Y \). An algorithm for building FCIL has been proposed as well. Experimental results show that the lattice-based approach is more efficient than the FCI-based one in most cases. Besides, the proposed lattice-based approach is quite suitable for reuse. If association rules under various minConf values are to be mined, the lattice needs to be built only once. The re-mining time can thus be significantly reduced.

It could also be observed from the experimental results that building a FCIL from FCIs is quite time-consuming, especially when the number of FCIs is large. Therefore in future, we will study how to reduce the time for building a FCIL by using a hash table to fast check the subsets.

Recent years, some methods for mining association rules from quantitative databases have been proposed such as mining weighted association rules, mining weighted utility association rules. Therefore, we will study how to apply lattices to these kinds of rules.

Acknowledgment

This work was supported by Vietnam’s National Foundation for Science and Technology Development (NAFOSTED) under Grant Number 102.01-2012.47.
Table 6

Comparison of numbers of MGARs, NARs, MNARs and traditional ARs with minConf = 0%.

<table>
<thead>
<tr>
<th>Database</th>
<th>minSup (%)</th>
<th>#Of rules</th>
<th>MGARs (1)</th>
<th>NARs (2)</th>
<th>MNARs (3)</th>
<th>Traditional ARs (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>80</td>
<td>951</td>
<td>27,711</td>
<td>316,057</td>
<td>552,564</td>
<td>3.43</td>
</tr>
<tr>
<td>Connect</td>
<td>97</td>
<td>101</td>
<td>1116</td>
<td>4600</td>
<td>8002</td>
<td>0.3</td>
</tr>
<tr>
<td>Mushroom</td>
<td>40</td>
<td>283</td>
<td>475</td>
<td>1168</td>
<td>7020</td>
<td>0.17</td>
</tr>
<tr>
<td>Pumsb*</td>
<td>60</td>
<td>126</td>
<td>192</td>
<td>611</td>
<td>12,840</td>
<td>0.08</td>
</tr>
<tr>
<td>Pumsb</td>
<td>95</td>
<td>101</td>
<td>267</td>
<td>690</td>
<td>1170</td>
<td>0.98</td>
</tr>
</tbody>
</table>

References