Short Papers

A Graph-Based Approach for Discovering Various Types of Association Rules

Show-Jane Yen and
Arbee L.P. Chen, *Member*, *IEEE*

Abstract—Mining association rules is an important task for knowledge discovery. We can analyze past transaction data to discover customer behaviors such that the quality of business decision can be improved. Various types of association rules may exist in a large database of customer transactions. The strategy of mining association rules focuses on discovering *large itemsets*, which are groups of items which appear together in a sufficient number of transactions. In this paper, we propose a graph-based approach to generate various types of association rules from a large database of customer transactions. This approach scans the database once to construct an *association graph* and then traverses the graph to generate all large itemsets. Empirical evaluations show that our algorithms outperform other algorithms which need to make multiple passes over the database.

Index Terms—Data mining, knowledge discovery, association rule, association pattern, graph-based approach.

1 Introduction

From a large amount of data, potentially useful information may be discovered. Techniques have been proposed to find knowledge from databases [1], [3], [4], [5]. Data mining has high applicability in the retail industry. The effective management of business is significantly dependent on the quality of its decision making. Therefore, it is important to improve the quality of business decisions by analyzing past transaction data to discover customer purchasing behaviors. In order to support this analysis, a sufficient amount of transactions needs to be collected and stored in a database. A transaction in the database typically consists of customer identifier, transaction date (or transaction time), and the items purchased in the transaction. Because the amount of these transaction data can be very large, an efficient algorithm needs to be designed for discovering useful information.

An *association rule* describes the associations among items in which when some items are purchased in a transaction, the others are purchased, too. In order to find association rules, we need to discover all *large itemsets* from a large database of customer transactions. A large itemset is a set of items which appear often enough within the same transactions.

The following definitions are adopted from [2]. A transaction t *supports* an item x if x is in t. A transaction t supports an itemset X if t supports every item in X. The *support for an itemset* is defined as the ratio of the total number of transactions which support this itemset to the total number of transactions in the database. To make the discussion easier, occasionally, we also let the total number of transactions which support the itemset denotes the support for the itemset. Hence, a large itemset is an itemset whose support is no less than a certain user-specified *minimum support*. An itemset of length k is called a k-itemset and a large itemset of length k a large k-itemset.

After discovering all large itemsets, the association rules can be generated as follows: If the large itemset $Y = I_1 I_2 \ldots I_k, k \geq 2$, all rules that reference items from the set $\{I_1, I_2, \ldots, I_k\}$ can be generated. The antecedent of each of these rules is a subset X of Y and the consequent $\mathrm{Y} - \mathrm{X}$. The *confidence* of $\mathrm{X} \Rightarrow \mathrm{Y} - \mathrm{X}$ in database D is the probability that when itemset X occurs in a transaction in D, itemset $\mathrm{Y} - \mathrm{X}$ also occurs in the same transaction. That is, the ratio of the support for itemset Y to the support for itemset X. A generated rule is an association rule if its confidence achieves a certain user-specified *minimum confidence*. Various algorithms [2], [6], [7], [8], [9], [10] have been proposed to generate all large itemsets from a large amount of transaction data. These algorithms generate candidate k-itemsets for large k-itemsets, scan each transaction in a database to count the supports for these candidate k-itemsets, and find all large k-itemsets in the kth iteration based on a predetermined minimum support. However, because the size of the database can be very large, it is very costly to repeatedly scan the database to count supports for candidate itemsets.

In this paper, we propose a graph-based approach to discovering various types of association rules, that is, *primitive association rules*, *generalized association rules*, and *multiple-level association rules*. A *primitive association rule* is an association rule which describes the association among *database items* which appear in the database. A *primitive association pattern* is a large itemset in which each item is a database item.

A *concept hierarchy* of the items can usually be derived. An example of the concept hierarchy is shown in Fig. 1, in which the terminal nodes are database items, and the nonterminal nodes are *generalized items*. If there is a path between nodes y and x, where y is a "higher concept" of x, then y is called an ancestor of x and x a descendant of y. In [10], the concept of "support" is extended such that a transaction t supports an item x if x is in t or x is an ancestor of some items in t. Association rules may exist at higher level concepts if the itemsets at the lower level concepts cannot reach the minimum support. Hence, significant association rules may not be discovered if we only consider database items which are the lowest level concepts in the concept hierarchy. A *generalized association rule* is introduced in [10], which describes the association among items which can be generalized items or database items. A *generalized association pattern* is a large itemset in which each item is a generalized item or database item.

Another type of association rule is called *multiple-level association rule* [6]. The multiple-level association rules are discovered from a large database of customer transactions in which all items are described by a set of relevant attributes. Each attribute represents a certain concept and these relevant attributes form a set of multiple-level concepts. The concept level for an attribute is defined by domain experts. For example, food items can be described by the relevant attributes "category," "content," and "brand," and attribute "category" represents the first-level concept (i.e., the highest level concept), attribute "content" the second-level concept and attribute "brand" the third-level concept. There is a set of domain values for an attribute. Each item in the database contains a domain value for each relevant attribute. For example, if the "category," "content," and "brand" of an item have the domain values "bread," "wheat," and "Wonder," respectively, then this item is described as "Wonder wheat bread" in the database.

- *S.-J. Yen is with the Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taipei, Taiwan, R.O.C. E-mail: sjyen@csie.fju.edu.tw.*
- *A.L.P. Chen is with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, R.O.C. E-mail: alpchen@cs.nthu.edu.tw.*

Manuscript received 7 Aug. 1997; revised 19 Mar. 2000; accepted 19 Apr. 2000; posted to Digital Library 6 Apr. 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 105484.

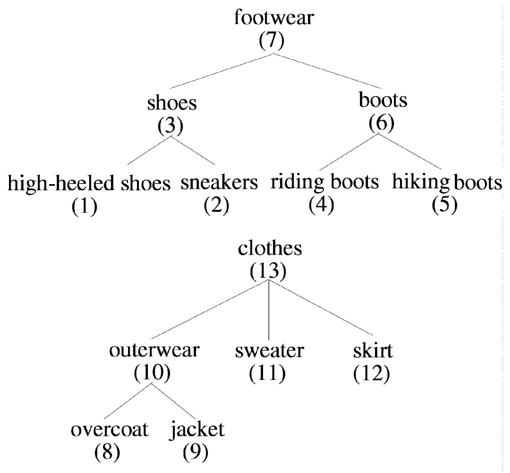


Fig. 1. An example of concept hierarchies.

From the items in the database, we can derive other items at different concept levels. The domain values of the attribute at the first (i.e., the highest) concept level are the items at the first concept level. An item at the $k\text{th}$ concept level can be formed by combining a domain value of the attribute at the $k\text{th}$ concept level with an item at the $(k-1)\text{th}$ concept level. Hence, item "bread" is at the first concept level, item "wheat bread" at the second concept level, and item "Wonder wheat bread" at the third concept level. For an item, the items at the corresponding higher (lower) concept level of this item are more general (specific) than this item. For example, items "bread" and "wheat bread" are the items at the corresponding higher level of item "Wonder wheat bread."

In general, items at higher concept levels have a larger support than those of items at lower concept levels. If we want to find associations among items at relatively low concept levels, many uninteresting associations among items at higher concept levels may also be generated. Hence, the minimum supports specified at higher concept levels should be larger than the minimum supports specified at lower concept levels. A multiple-level association rule [6] is an association rule which describes the associations among items at the same concept level. For each concept level, both minimum support and minimum confidence are specified. A *multiple-level association pattern* is a large itemset in which all items are at the same concept level. For an item I to be in a multiple-level association pattern, the items at the corresponding higher concept levels of the item I need to be large at their corresponding concept levels. This is to avoid the generation of many meaningless combinations formed by the items at the corresponding lower concept level of the nonlarge items. For example, if "bread" is not a large item, item "wheat bread" which is at the corresponding lower concept level of item "bread" need not be further examined.

In our previous work [11], we proposed a graph-based approach to analyze a large amount of transaction data and to generate primitive association rules. In this paper, we extend the graph-based approach to generate generalized association rules and multiple-level association rules. We propose a uniform data mining framework to discover various types of association rules:

1. Numbering phase: In this phase, all items are assigned an integer number.
2. *Large item* generation phase: This phase generates large items and records related information. A large item is an item whose support is no less than a user specified minimum support.
3. Association graph construction phase: This phase constructs an association graph to indicate the associations between large items.

TABLE 1
A Database TDB1 of Transactions

TID	Itemset
100	CAD
200	ECB
300	ABCE
400	EB

4. Association pattern generation phase: This phase generates all association patterns by traversing the constructed association graph.
5. Association rule generation phase: The association rules can be generated directly according to the corresponding association patterns.

This paper focuses on the association pattern generation, because after generating the association patterns, the association rules can be generated from the corresponding association patterns. In this paper, we present three algorithms for generating primitive association patterns, generalized association patterns, and multiple-level association patterns.

The rest of this paper is organized as follows: The algorithms to discovering primitive association patterns, generalized association patterns and multiple-level association patterns are presented in Section 2, 3, and 4, respectively. Section 5 evaluates the performance of our data mining algorithms. Finally, we conclude the paper and present directions for future research in Section 6.

2 Mining Primitive Association Rules

In this section, an algorithm PAPG (Primitive Association Pattern Generation) is presented to generate primitive association patterns, which is the same as the algorithm DLG [11]. Because we focus on the association pattern generation, in the following, we describe the first four phases discussed in Section 1 for the algorithm PAPG.

2.1 Association Graph Construction

In the numbering phase, the algorithm PAPG arbitrarily assigns each item a unique integer number. In the large item generation phase, PAPG scans the database and builds a bit vector for each item. The length of each bit vector is the number of transactions in the database. If an item appears in the $i\text{th}$ transaction, the $i\text{th}$ bit of the bit vector associated with this item is set to 1. Otherwise, the $i\text{th}$ bit of the bit vector is set to 0. The bit vector associated with item i is denoted as BV_i. The number of 1s in BV_i is equal to the number of transactions which support the item i, that is, the support for the item i.

Example 2.1. Consider the database TDB1 in Table 1. Each record is a <TID, Itemset> pair, where TID is the identifier of the corresponding transaction, and Itemset records the items purchased in the transaction. Assume that the minimum support $\Im$ is 50 percent (i.e., 2 transactions).

After the numbering phase, the numbers of the items A, B, C, D, and E are 1, 2, 3, 4, and 5, respectively. In the large item generation phase, the large items found in the database TDB1 are items 1, 2, 3, and 5, and BV_1, BV_2, BV_3, and BV_5 are (1010), (0111), (1110), and (0111), respectively. In the following, we use the number of an item to represent this item.

Property 2.1. *The support for the itemset $\{i_1, i_2, \ldots, i_k\}$ is the number of 1s in $\mathrm{BV}i_1 \wedge \mathrm{BV}i_2 \wedge \ldots \wedge \mathrm{BV}i_k$, where the notation "$\wedge$" is a logical AND operation.*

In the association graph construction phase, PAPG applies the algorithm AGC (Association Graph Construction) to construct an

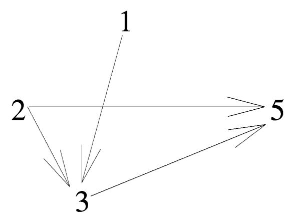


Fig. 2. The association graph for Example 2.1.

association graph. The AGC algorithm is described as follows: For every two large items i and $j(i < j)$, if the number of 1s in $\mathrm{BV}_i \wedge \mathrm{BV}_j$ achieves the user-specified minimum support, a directed edge from item i to item j is created. Also, itemset (i, j) is a large 2-itemset. The association graph for the Example 2.1 is shown in Fig. 2.

2.2 Primitive Association Pattern Generation

The large 2-itemsets are generated after the association graph construction phase. In the association pattern generation phase, the algorithm LGDE (Large itemset Generation by Direct Extension) is proposed to generate large k-itemsets ($k > 2$), which is described as follows: For each large k-itemsets ($k \geq 2$), the last item of the k-itemset is used to extend the large itemset into $k+1$-itemsets.

Lemma 2.1. *If an itemset is not a large itemset, then any itemset which contains this itemset cannot be a large itemset.*

Lemma 2.2. *For a large itemset $(i_1, i_2, \ldots, i_k)$, if there is no directed edge from item i_k to an item v, then itemset $(i_1, i_2, \ldots, i_k, v)$ cannot be a large itemset.*

Suppose $(i_1, i_2, \ldots, i_k)$ is a large k-itemset. If there is no directed edge from item i_k to an item v, then the itemset need not be extended into $k+1$-itemset, because $(i_1, i_2, \ldots, i_k, v)$ must not be a large itemset according to Lemma 2.2. If there is a directed edge from item i_k to an item u, then the itemset $(i_1, i_2, \ldots, i_k)$ is extended into $k+1$-itemset $(i_1, i_2, \ldots, i_k, u)$. The itemset $(i_1, i_2, \ldots, i_k, u)$ is a large $k+1\text{-itemset}$ if the number of 1s in $\mathrm{BV}i_1 \wedge \mathrm{BV}i_2 \wedge \ldots \wedge \mathrm{BV}i_k \wedge \mathrm{BV}_u$ achieves the minimum support. If no large $k+1\text{-itemsets}$ can be generated, the algorithm LGDE terminates. For example, consider Example 2.1. After the association graph construction phase, the large 2-itemsets (1, 3), (2, 3), (2, 5), (3, 5) are generated. For the large 2-itemset (2, 3), there is a directed edge from the last item 3 of the itemset (2, 3) to item 5 in the association graph shown in Fig. 2. Hence, the 2-itemset (2, 3) is extended into 3-itemset (2, 3, 5). The number of 1s in $\mathrm{BV}_2 \wedge \mathrm{BV}_3 \wedge \mathrm{BV}_5$ (i.e., (0110)) is 2. Hence, the 3-itemset (2, 3, 5) is a large 3-itemset, since the number of 1s in its bit vector is no less than the minimum support threshold. The LGDE algorithm terminates because no large 4-itemsets can be further generated.

3 Mining Generalized Association Rules

We propose the algorithm GAPG (Generalized Association Pattern Generation) to discover all generalized association patterns. In the following, we also describe the four phases for the algorithm GAPG.

3.1 A Numbering Method

To generate generalized association patterns, one can add all ancestors of each item in a transaction to the transaction and then apply the algorithm PAPG on the extended transactions. However, because if an item is a large item, then the 2-itemset which contains the item and its ancestor is also a large 2-itemset, the number of the edges in the association graph can be very large, and the LGDE algorithm needs to take much more time to traverse the association graph to generate all large itemsets.

Lemma 3.1 [10]. *The support for an itemset X that contains both an item $\mathrm{x_i}$ and its ancestor χ_i will be the same as the support for the itemset $\mathrm{X} - \chi_\mathrm{i}$.*

Rationale. Suppose the itemset $\mathrm{X} = \{\mathrm{x_1}, \ldots, \mathrm{x_i}, \chi_\mathrm{i}, \mathrm{x_{i+1}}, \ldots, \mathrm{x_n}\}$. The support for itemset X is the number of 1s in

$$\mathrm{BVx_1} \wedge \ldots \wedge \mathrm{BVx_i} \wedge \mathrm{BV}_{\chi_\mathrm{i}} \wedge \mathrm{BVx_{i+1}} \wedge \ldots \wedge \mathrm{BVx_n},$$

and the support for itemset $\mathrm{X} - \chi_\mathrm{i}$ is the number of 1s in

$$\mathrm{BVx_1} \wedge \ldots \wedge \mathrm{BVx_i} \wedge \mathrm{BVx_{i+1}} \wedge \ldots \wedge \mathrm{BVx_n},$$

according to Property 2.1. Because the set of the transactions which contain an item $\mathrm{x_i}$ is the subset of the set of the transactions which contain the ancestor χ_i of item $\mathrm{x_i}$, $\mathrm{BVx_i} \wedge \mathrm{BV}\chi_\mathrm{i} = \mathrm{BVx_i}$. Hence, the support for X is the same as the support for the itemset $X - \chi_i$.

From Lemma 3.1, when an itemset X contains both an item x and its ancestor χ, if the itemset $\mathrm{X} - \chi$ is a large itemset, then itemset X is also a large itemset. Lemma 3.1 can be employed to reduce the cost for large itemset generation. Hence, the problem of mining generalized association patterns becomes to find all generalized association patterns which do not contain both an item and its ancestor.

In the numbering phase, GAPG applies the numbering method PON (POstorder Numbering method) to number items at the concept hierarchies. For each concept hierarchy, PON numbers each item according to the following order: For each item at the concept hierarchy, after all descendants of the item are numbered, PON numbers this item immediately, and all items are numbered increasingly. After all items at a concept hierarchy are numbered, PON numbers items at another concept hierarchy.

Lemma 3.2. *If the numbering method PON is adopted to number items, and for every two items i and j ($i < j$), item ϑ is an ancestor of item i but not an ancestor of item j, then $\vartheta < j$.*

Rationale. According to PON numbering method, after all descendants of an item are numbered, this item is numbered immediately, and these items are numbered increasingly. Hence, for an item i, if it is numbered, then its ancestor ϑ must

TABLE 2
A Database TDB2 of Transactions

TID	Itemset
100	{high-heeled shoes, riding boots, overcoat, sweater, skirts}
200	{high-heeled shoes, sneakers, skirts}
300	{high-heeled shoes, skirts}
400	{hiking boots, jackets, sweater}
500	{overcoat, sweater}

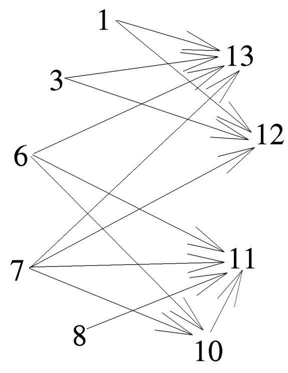


Fig. 3. The generalized association graph for Example 3.1.

be numbered before the other item j which is not a descendant of item ϑ is numbered. So, $\vartheta < j$.

Example 3.1. Consider the database TDB2 in Table 2 and the concept hierarchies in Fig. 1. Assume that the minimum support $\Im$ is 40 percent (i.e., 2 transactions).

After applying the PON method on the concept hierarchies in Fig. 1, all items at the concept hierarchies are numbered, where the number within the parentheses below each item in Fig. 1 is the number of the item.

3.2 Large Item Generation

In the large item generation phase, GAPG builds a bit vector for each database item, and finds all large items (include database items and generalized items). Here, we assume that all database items are specific items.

Lemma 3.3. *Suppose items $i_1, i_2, \ldots,$ and i_m are all specific descendants of the generalized item i_n. The bit vector $\mathrm{BV}i_n$ associated with item i_n is $\mathrm{BV}i_1 \vee \mathrm{BV}i_2 \vee \ldots \vee \mathrm{BV}i_m$, and the number of 1s in $\mathrm{BV}i_1 \vee \mathrm{BV}i_2 \vee \ldots \vee \mathrm{BV}i_m$ is the support for item i_n, where the notation "$\vee$" is a logical OR operation.*

From Lemma 3.3, the bit vector associated with a generalized item is obtained by performing logical OR operations on the bit vectors associated with all specific descendants of the generalized item.

3.3 Generalized Association Graph Construction

In the association graph construction phase, GAPG applies the algorithm GAGC (Generalized Association Graph Construction) to construct a *generalized association graph* to be traversed. The algorithm GAGC is described as follows: For every two large items i and j $(i < j)$, if item j is not an ancestor of item i and the number of 1s in $\mathrm{BV}_i \wedge \mathrm{BV}_j$ achieves the user-specified minimum support, a directed edge from item i to item j is created. Also, itemset (i, j) is a large 2-itemset.

Lemma 3.4. *If an itemset X is a large itemset, then any itemset generated by replacing an item in itemset X with its ancestor is also a large itemset.*

Lemma 3.5. *If (the number of 1s in $\mathrm{Bv}_i \wedge \mathrm{BV}_j$) $\geq$ minimum-support, then for each ancestor u of item i and for each ancestor v of item j, (the number of 1s in $\mathrm{BV}_u \wedge \mathrm{BV}_j$) $\geq$ minimum-support and (the number of 1s in $\mathrm{BV}_i \wedge \mathrm{BV}_v$) $\geq$ minimum-support.*

From Lemma 3.5, if an edge from item i to item j is created, the edges from item i to the ancestors of item j, which are not ancestors of item i, are also created. According to Lemma 3.2, the numbers of the ancestors of item i, which are not the ancestors of item j, are all less than j. Hence, if an edge from item i to item j is created, the edges from the ancestors of item i, which are not ancestors of item j, to item j are also created.

After applying the GAGC algorithm in the association graph construction phase, the generalized association graph for Example 3.1 is constructed in Fig. 3, where there are no edges between an item and its ancestors.

TABLE 3
A Database TDB3 of Transactions

TID	Itemset
100	{111, 121, 211, 221}
200	{111, 211, 222, 323}
300	{112, 122, 221, 411}
400	{111, 121}
500	{111, 122, 211, 221, 413}
600	{211, 323, 524}
700	{323, 411, 524, 713}

3.4 Generalized Association Pattern Generation

In the association pattern generation phase, GAPG applies the LGDE algorithm to generate all generalized association patterns by traversing the generalized association graph.

Theorem 3.1. *If the numbering method PON is adopted to number items and the algorithm GAGC is applied to construct a generalized association graph, then any itemset generated by traversing the generalized association graph (i.e., performing the LGDE algorithm) will not contain both an item and its ancestor.*

Proof. We use mathematical induction to prove this theorem.

- **Basis of induction.** By the GAGC algorithm, because there is no edge between an item and its ancestor, any large 2-itemset does not contain an item and its ancestor.
- **Inductive hypothesis.** We assume that any large k-itemset $(i_1, i_2, \ldots, i_k)$ does not contain both an item and its ancestor.
- **Inductive step.** Suppose there is a directed edge from item i_k to item w in the generalized association graph constructed by applying GAGC algorithm. By the LGDE algorithm, the large k-itemset $(i_1, i_2, \ldots, i_k)$ is extended into $k+1$-itemset $(i_1, i_2, \ldots, i_k, w)$. Suppose items $\vartheta_1, \vartheta_2, \ldots,$ and ϑ_{k-1} are the ancestors of items $i_1, i_2, \ldots,$ and i_{k-1}, respectively, but none are ancestors of item i_k. Because items are numbered by the PON method, $i_k > \vartheta_j (1 \leq j \leq k-1)$ (Lemma 3.2). Hence, there are no edges from item i_k to the ancestors of items $i_1, i_2, \ldots,$ and i_k. So, item w cannot be an ancestor of item $i_1, i_2, \ldots,$ or i_k. □

4 Mining Multiple-Level Association Rules

In this section, we present the algorithm MLAPG (Multiple-Level Association Pattern Generation) to generate all multiple-level association patterns. In Section 1, we have mentioned that each item in the database contains domain values of the relevant attributes for the problem of mining multiple-level association rules. For an attribute, each domain value is arbitrarily given a unique number in the numbering phase. Besides, each item in a transaction is numbered according to its domain values.

After the numbering phase, MLAPG performs the remaining three phases for each concept level (from the highest concept level to the lowest concept level), which are similar to the associated three phases in the PAPG algorithm. The large item generation phase scans the database once to build a bit vector for each item and find large items at the current concept level. Also, the size of

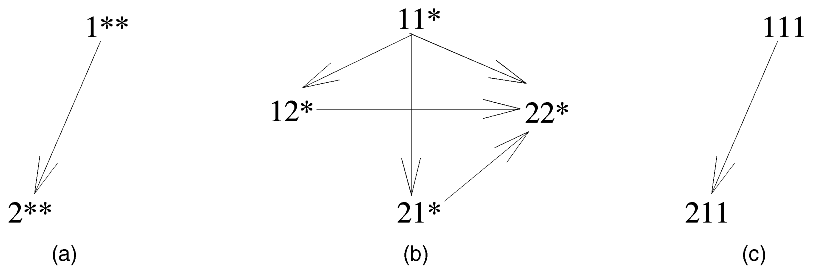


Fig. 4. The association graphs for Example 4.1. (a) Level-1 association graph. (b) Level-2 association graph. (c) Level-3 association graph.

the database is progressively trimmed by eliminating the items which are not large items at the previous concept level. Notice that, for the last concept level, the database need not be further trimmed. The association graph construction phase constructs an association graph by applying the AGC algorithm for the large items found at the current concept level. The association pattern generation phase performs the algorithm LGDE on the constructed association graph to generate all large itemsets at the current concept level.

Example 4.1. Consider the database TDB3 in Table 3 in which there are three concept levels defined, and the items at each concept level are numbered. For example, in Table 3, item "221" can be the item "Wonder wheat bread," where the first number "2" represents the domain value "bread" of the attribute "category" at level-1, the second number "2" for the domain value "wheat" of the attribute "content " at level-2, and the third number "1" for the domain value "Wonder" of the attribute "brand" at level-3. Assume that the minimum supports $\Im_1$, $\Im_2$, and $\Im_3$ specified at the levels 1, 2, and 3 are 4, 3, and 3 transactions, respectively.

Because there are three concept levels defined, MLAPG needs to perform the above three phases three times to generate all multiple-level association patterns. For the first level, only the level-1 items in the transactions are considered. After the large item generation phase, the found level-1 large items are "1**" and "2**" and the associated bit vectors are (1111100) and (1110110), respectively, where the notation "*" represents any item. In the association graph construction phase, the association graph for the level-1 large items is constructed, as shown in Fig. 4a and the level-1 large 2-itemset (1**, 2**) is generated.

For the second level, MLAPG scans the database TDB3 to build a bit vector for each level-2 item and find level-2 large items in the large item generation phase. The level-2 large items found are 11*, 12*, 21*, and 22*, and the associated bit vectors are (1111100), (1011100), (1100110), and (1110100), respectively. By the way, MLAPG eliminates nonlarge items at the first level from the database TDB3. The trimmed database is shown in Table 4.

In the association graph construction phase, MLAPG performs the AGC algorithm for the level-2 large items. The constructed association graph is shown in Fig. 4b. After the association pattern generation phase, the level-2 large 3-itemsets (11*, 21*, 22*) and (11*, 12*, 22*) are generated by performing the algorithm LGDE.

For the last level, after scanning the trimmed database in Table 4, the found level-3 large items are 111, 211, and 221, and the associated bit vectors are (110110), (110011), and (101010), respectively. The association graph constructed for the level-3 large items is shown in Fig. 4c, and there is only one level-3 large 2-itemset (111, 211) generated.

TABLE 4
The Trimmed Database of TDB3

TID	Itemset
100	{111, 121, 211, 221}
200	{111, 211, 222}
300	{112, 122, 221}
400	{111, 121}
500	{111, 122, 211, 221}
600	{211}

5 Performance Evaluation

In this section, we evaluate the performance of the three algorithms PAPG, GAPG, and MLAPG. In [11], we have evaluated the performance of PAPG (called DLG in [11]) and demonstrated that PAPG has a better performance than other approaches [2], [7], [8], [9]. In the following sections, we analyze the performance of the two algorithms, GAPG and MLAPG.

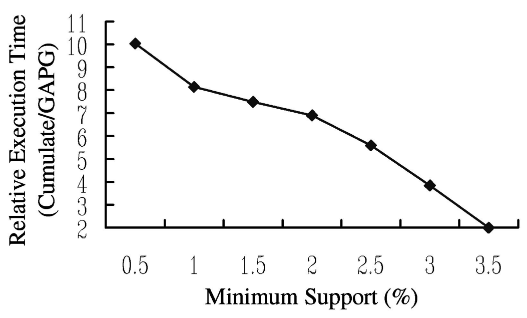
5.1 Performance Evaluation for GAPG Algorithm

In this section, we evaluate the performance of the GAPG algorithm by comparing this algorithm with the algorithm Cumulate [10].

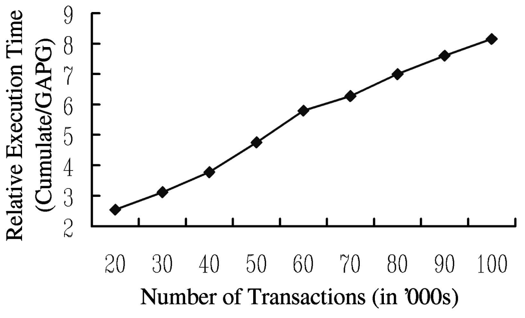
We first generate the synthetic data for the experiment by applying the method described in [10]. The parameters for the synthetic data are set as follows: The number of transactions is 100,000, the number of items is 100,00, the number of concept hierarchies is 100, the fanout is 5, the number of the potentially large itemsets is 5,000, the average size of the transactions is 10, and the average size of the potentially large itemsets is 5.

Suppose in the kth iteration, the set GL_k of the large k-itemsets is generated. In the first iteration, Cumulate scans the database to add the ancestors of each item in a transaction to the transaction, and count the support for each item in the extended database. For GAPG, it first applies the PON method to number items at the concept hierarchies, and then scans the database to count the support and build a bit vector for each item in the database. The support for the generalized items can be obtained by performing logical OR operations on the bit vectors associated with some specific items. Hence, in the first iteration, the two algorithms, Cumulate and GAPG, takes a similar time to generate large items.

In the second iteration, Cumulate generates candidate 2-itemsets by combining every two large items and deletes any candidate 2-itemset that consists of an item and its ancestor. For these remaining candidate 2-itemsets, Cumulate adds the ancestors of each item in a transaction, which are present in any of the candidates, to the transaction and count the support for each candidate 2-itemset by scanning each extended transaction. However, the number of the candidate 2-itemsets to be counted and the size of the extended database both are very large. It is very time consuming to search such a large number of candidates and scan the large extended database. For the GAPG algorithm, it applies GAGC algorithm to construct a generalized association



(a)



(b)

Fig. 5. Relative Execution Time (Cumulate/GAPG).

graph. Suppose the average number of the ancestors of each large item is p. GAGC algorithm at most needs to perform

$$\frac{|GL_1|(|GL_1|-1)}{2} - |GL_1| \times p$$

logical AND operations on the bit vectors to construct a generalized association graph and generate large 2-itemsets.

In the $k\mathrm{th}$ $(k > 2)$ iteration, Cumulate generates candidate k-itemsets by applying join-based algorithm [2]. The execution time of Cumulate still depends on the number of generated candidate itemsets and the amount of data that has to be scanned. For the GAPG algorithm, it applies LGDE algorithm to generate large k-itemsets. LGDE extends each large $k-1$-itemset into k-itemsets according to the generalized association graph and performs logical AND operations. Suppose the average out-degree of each item in the generalized association graph is q. LGDE performs $(k-1) \times |GL_{k-1}| \times q$ logical AND operations to find all large k-itemsets. Hence, as the minimum support decreases, the number of logical AND operations performed increases because the two values $|GL_{k-1}|$ and q increase.

Since the number of the candidate itemsets to be counted and the size of the extended database to be scanned by Cumulate are much larger than the logical AND operations performed by GAPG, and GAPG needs only one database scan but Cumulate needs to scan the extended database in each iteration, GAPG always outperforms Cumulate for various minimum supports which is shown in Fig. 5a. Fig. 5b shows the relative execution time for Cumulate and GAPG for various database sizes, in which the minimum support is set to 1 percent.The GAPG algorithm outperforms the Cumulate algorithm significantly and the performance gap increases as the minimum support decreases or the database size increases because the number of candidate itemsets and the number of database scans increases for Cumulate.

5.2 Performance Evaluation for MLAPG Algorithm

Han and Fu [6] presented four similar algorithms to generate multiple-level association patterns, in which the algorithm ML_T2L1 is used to compare with our algorithm MLAPG.

The synthetic data generation method is the same as the method described in [6]. The parameters used to generate synthetic data are set as follows: The number of transactions is 100,000, the number of items is 1,000, the number of potentially large itemsets is 2,000, the average size of the transactions is 5, and the average size of the potentially large itemsets is 10. Besides, the number of the concept levels is set to 4 and the fanouts for levels 2, 3, and 4 are set to 5, 5, and 5, respectively. Hence, the number of nodes at level-1 is 8.

Fig. 6 shows the relative execution time of MLAPG and ML_T2L1 for various database sizes. In this experiment, we set the minimum supports for levels 1, 2, 3, and 4 to 50 percent, 10 percent, 5 percent, and 2 percent, respectively, because in this case, the algorithm ML_T2L1 has the best performance among the four algorithms presented in [6].

For the concept level i, ML_T2L1 needs to generate level-i candidate k-itemsets $(k \geq 1)$ and scan the database to count the support for each level-i candidate k-itemset in the $k\mathrm{th}$ iteration. If there are n concept levels defined and Apriori [2] needs to perform r_i iterations to find all level-i large itemsets, then there are

$$\sum_{i=1}^{n} r_i$$

database scans needed to generate all large itemsets at each concept level. However, MLAPG needs only n database scans to generate all large itemsets at every concept level. Because MLAPG and ML_T2L1 apply algorithms PAPG and Apriori, respectively, for each concept level, we analyze the performance of the two algorithms PAPG and Apriori.

Suppose in the $k\mathrm{th}$ iteration, the set L_k of the large k-itemsets is generated. For the first iteration, PAPG and Apriori both need to scan the database to count the support for each item. By the way, PAPG builds the bit vector for each item.

In the second iteration, PAPG performs

$$\frac{|L_1|(|L_1-1|)}{2}$$

logical AND operations on the bit vectors to construct an association graph and generate large 2-itemsets. However, Apriori needs to generate

$$\frac{|L_1|(|L_1-1|)}{2}$$

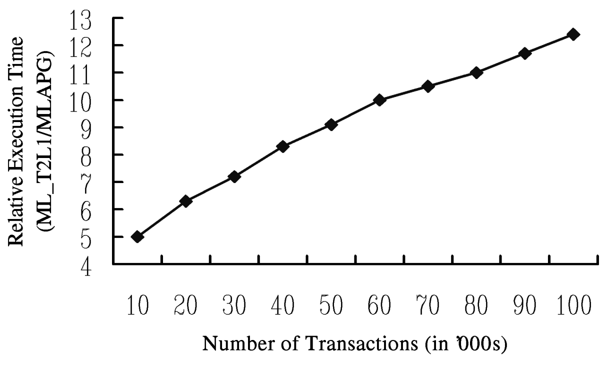


Fig. 6. Relative Execution Time (ML_T2L1/MLAPG).

candidate 2-itemsets. After generating candidate 2-itemsets, a priori scans the database to combine every two items in each transaction and search these candidate 2-itemsets to count their supports.

In the kth iteration $(k > 2)$, PAPG applies the LGDE algorithm to generate large k-itemsets. Suppose the average out-degree of each item in the association graph is q. LGDE performs $(k-1) \times |L_{k-1}| \times q$ logical AND operations to find all large k-itemsets. However, Apriori needs to generate candidate k-itemsets from large $k-1$-itemsets. After generating candidate k-itemsets, Apriori scans the database to combine every k items in each transaction and search these candidate k-itemsets to count their supports. Hence, the execution time of Apriori depends on the number of generated candidate itemsets and the amount of data that has to be scanned.

Since for each concept level, the number of logical AND operations performed by MLAPG is much less than the number of the candidate itemsets to be counted and the sizes of the databases to be scanned by ML_T2L1, ML_T2L1 takes much more time than MLAPG for each concept level. Fig. 6 shows that MLAPG outperforms ML_T2L1 significantly and the performance gap increases as the size of the database increases because the number of candidate itemsets and the number of database scans increase for ML_T2L1 algorithm.

6 Conclusion and Future Work

We propose a uniform graph-based approach to discover the three types of association rules: primitive association rules, generalized association rules, and multiple-level association rules. The approach includes the five phases: numbering phase, large item generation phase, association graph construction phase, association pattern generation phase, and association rule generation phase. We present the three algorithms: PAPG, GAPG, and MLAPG to generate the three types of association patterns: primitive association patterns, generalized association patterns, and multiple-level association patterns.

In [11], the algorithm PAPG has been demonstrated to have a better performance than other approaches [2], [7], [8], [9]. In this paper, we compare GAPG and MLAPG to the previously known algorithms, Cumulate [10] and ML_T2L1 [6], respectively. The experimental results show that GAPG and MLAPG outperform Cumulate and ML_T2L1, respectively. When the minimum support decreases or the size of the database increases, the performance gap increases because the number of candidate itemsets generated by GAPG or MLAPG increases and the number of database scans also increases.

For our approach, the related information may not fit in the main memory when the size of the database is very large. In the future, we shall consider this problem by reducing the memory space requirement. Also, we shall apply our approach on different applications, such as document retrieval and resource discovery in the World Wide Web environment.

Acknowledgments

This work was partially supported by the Republic of China National Science Council under contract no. 89-2213-E-030-022. The authors would like to thank the Societas Verrbi Divini for their financial support.

References

[1] Y. Aumann et al., "Borders: An Efficient Algorithm for Association Generation in Dynamic Databases," *J. Intelligent Information Systems,* pp. 61-73, 1999.

[2] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Association Rules," *Proc. Int'l Conf. Very Large Data Bases,* pp. 487-499, 1994.

[3] R.J. Bayardo Jr., "Efficiently Mining Long Patterns from Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 85-93, 1998.

[4] C.L. Carter and H.J. Hamilton., "Efficient Attribute-Oriented Algorithms for Knowledge Discovery from Large Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 10, no. 2, pp. 193-208, Mar./Apr. 1998.

[5] R.J. Hilderman et al., "Data Mining in Large Databases Using Domain Generalization Graphs," *J. Intelligent Information Systems,* pp. 195-234, 1999.

[6] J. Han and Y. Fu, "Mining Multiple-Level Association Rules in Large Databases," *IEEE Trans. Knowledge and Data Eng.,* pp. 798-805, 1999.

[7] M. Houtsma and A. Swami, "Set-Oriented Mining for Association Rules in Relational Databases," *Proc. Int'l Conf. Data Eng.,* pp. 25-33, 1995.

[8] H. Mannila et al., "Efficient Algorithm for Discovering Association Rules," *Proc. AAAI Workshop Knowledge Discovery in Databases,* pp. 181-192, 1994.

[9] J.S. Park, M.S. Chen, and P.S. Yu, "An Effective Hash-Based Algorithm for Mining Association Rules," *Proc. ACM SIGMOD,* vol. 24, no. 2, 1995.

[10] R. Srikant and R. Agrawal, "Mining Generalized Association Rules," *Proc. Int'l Conf. Very Large Data Bases,* pp. 407-419, 1995.

[11] S.J. Yen and A.L.P. Chen, "An Efficient Approach to Discovery Knowledge from Large Databases," *Proc. IEEE/ACM Int'l Conf. Parallel and Distributed Information Systems,* pp. 8-18, 1996.