Discovery of Multiple-Level Association Rules from Large Databases *

Jiawei Han and Yongjian Fu
School of Computing Science
Simon Fraser University
British Columbia, Canada V5A 1S6
{han,yongjian}@cs.sfu.ca

Abstract

Previous studies on mining association rules find rules at single concept level, however, mining association rules at multiple concept levels may lead to the discovery of more specific and concrete knowledge from data. In this study, a top-down progressive deepening method is developed for mining multiple-level association rules from large transaction databases by extension of some existing association rule mining techniques. A group of variant algorithms are proposed based on the ways of sharing intermediate results, with the relative performance tested on different kinds of data. Relaxation of the rule conditions for finding "level-crossing" association rules is also discussed in the paper.

1 Introduction

With wide applications of computers and automated data collection tools, massive amounts of transaction data have been collected and stored in databases. Discovery of interesting association relationships among huge amounts of data will help marketing, decision making, and business management. Therefore, mining association rules from large data sets has been a focused topic in recent research into knowledge discovery in databases [1, 2, 3, 9, 12, 14].

Studies on mining association rules have evolved from techniques for discovery of functional dependencies [10], strong rules [14], classification rules [7, 15], causal rules [11], clustering [6], etc. to disk-based, efficient methods for mining association rules in large sets of transaction data [1, 2, 3, 12]. However, previous work has been focused on mining association rules at a single concept level. There are applications which need to find associations at multiple concept levels. For example, besides finding *80% of customers that purchase* milk *may also purchase* bread, it could be informative to also show that *75% of people buy* wheat bread *if they buy* 2% milk. The association relationship in the latter statement is expressed at a lower concept level but often carries more specific and concrete information than that in the former. This requires progressively deepening the knowledge mining process for finding refined knowledge from data. The necessity for mining multiple level association rules or using taxonomy information at mining association rules has also been observed by other researchers, e.g., [2].

To confine the association rules discovered to be *strong* ones, that is, the patterns which occur relatively frequently and the rules which demonstrate relatively strong implication relationships, the concepts of *minimum support* and *minimum confidence* have been introduced [1, 2]. Informally, the support of a pattern A in a set of transactions S is the probability that a transaction in S contains pattern A; and the confidence of $A \rightarrow B$ in S is the probability that pattern B occurs in S if pattern A occurs in S.

*This research was supported in part by the research grant NSERC-A3723 from the Natural Sciences and Engineering Research Council of Canada, the research grant NCE/IRIS-HMI5 from the Networks of Centres of Excellence of Canada, and a research grant from the Hughes Research Laboratories.

Proceedings of the 21st VLDB Conference
Zurich, Swizerland, 1995

For mining multiple-level association rules, concept taxonomy should be provided for generalizing primitive level concepts to high level ones. In many applications, the taxonomy information is either stored implicitly in the database, such as "*Wonder wheat bread is a wheat bread which is in turn a bread*", or computed elsewhere [7]. Thus, data items can be easily generalized to multiple concept levels. However, direct application of the existing association rule mining methods to mining multiple-level associations may lead to some undesirable results as presented below.

First, large support is more likely to exist at high concept levels, such as *milk* and *bread*, rather than at low concept levels, such as a *particular brand* of milk and bread. Therefore, if one wants to find strong associations at relatively low concept levels, the minimum support threshold must be reduced substantially. However, this may lead to the generation of many uninteresting associations, such as "toy $\rightarrow$ 2% milk" before the discovery of some interesting ones, such as "Dairyland 2% milk $\rightarrow$ Wonder wheat bread", because the former may occur more frequently and thus have larger support than the latter.

Second, it is unlikely to find many strong association rules at a primitive concept level, such as the associations among particular bar codes, because of the tiny average support for each primitive data item in a very large item set. However, mining association rules at high concept levels may often lead to the rules corresponding to prior knowledge and expectations [9], such as "milk $\rightarrow$ bread", or lead to some uninteresting attribute combinations, such as "toy $\rightarrow$ milk".

In order to remove uninteresting rules generated in knowledge mining processes, researchers have proposed some measurements to quantify the "usefulness" or "interestingness" of a rule [13] or suggested to "put a human in the loop" and provide tools to allow human guidance [4]. Nevertheless, automatic generation of relatively focused, informative association rules will be obviously more efficient than first generating a large mixture of interesting and uninteresting rules.

These observations lead us to examining the methods for mining association rules at multiple concept levels, which may not only discover rules at different levels but also have high potential to find nontrivial, informative association rules because of its flexibility at focusing the attention to different sets of data and applying different thresholds at different levels.

In this study, issues for mining multiple-level association rules from large databases are examined, with a top-down, progressive deepening method developed by extension of some existing algorithms for mining single-level association rules. The method first finds large data items at the top-most level and then progressively deepens the mining process into their large descendants at lower concept levels. Some data structures and intermediate results generated at mining high level associations can be shared for mining lower level ones, and different sharing schemes lead to different variant algorithms. The performance identifies the conditions that certain algorithms could be best suited for certain kinds of data distributions.

The paper is organized as follows. In Section 2, the concepts related to multiple-level association rules are introduced. In Section 3, a method for mining multiple-level association rules in large data sets is studied. In Section 4, a set of variant algorithms for mining multiple-level association rules are introduced, with their relative efficiency analyzed. In Section 5, a performance study is conducted on different kinds of data distributions, which identifies the conditions for the selection of algorithms. Section 6 is a discussion on mining "level-crossing" association rules and several other issues. The study is concluded in Section 7.

2 Multiple level association rules

To study the mining of association rules from a large set of transaction data, we assume that the database contains (1) a transaction data set, $\mathcal{T}$, which consists of a set of transactions $\langle T_i, \{A_p, \ldots, A_q\}\rangle$, where T_i is a transaction identifier, $A_i \in \mathcal{I}$ (for $i = p, \ldots, q$), and $\mathcal{I}$ is the set of all the data items in the item data set; and (2) the description of the item data set, $\mathcal{D}$, which contains the description of each item in $\mathcal{I}$ in the form of $\langle A_i, description_i\rangle$, where $A_i \in \mathcal{I}$.

Furthermore, to facilitate the management of large sets of transaction data, our discussion adopts an extended relational model which allows an attribute value to be either a single or a set of values (i.e., in non-first-normal form). Nevertheless, the method developed here is applicable (with minor modifications) to other representations of data, such as a data file, a relational table, or the result of a relational expression.

Definition 2.1 A pattern, A, is one item A_i or a set of conjunctive items $A_i \wedge \cdots \wedge A_j$, where $A_i, \ldots, A_j \in \mathcal{I}$. The **support** of a pattern A in a set S, $\sigma(A/S)$, is the number of transactions (in S) which contain A versus the total number of transactions in S. The **confidence** of $A \rightarrow B$ in S, $\varphi(A \rightarrow B/S)$, is the ratio of $\sigma(A \wedge B/S)$ versus $\sigma(A/S)$, i.e., the probability that pattern B occurs in S when pattern A occurs in S.

To find relatively frequently occurring patterns and reasonably strong rule implications, a user or an expert may specify two thresholds: *minimum support*,

σ', and *minimum confidence*, φ'. Notice that for finding multiple-level association rules, different minimum support and/or minimum confidence can be specified at different levels.

Definition 2.2 A pattern A is **large** in set S at level l if the support of A is no less than its corresponding minimum support threshold σ'_l. The confidence of a rule "$A \rightarrow B/S$" is **high** at level l if its confidence is no less than its corresponding minimum confidence threshold φ'_l.

Definition 2.3 A rule "$A \rightarrow B/S$" is **strong** if, for a set S, each ancestor (i.e., the corresponding high level item) of every item in A and B, if any, is large at its corresponding level, "$A \wedge B/S$" is large (at the current level), and the confidence of "$A \rightarrow B/S$" is high (at the current level).

The definition indicates that if "$A \rightarrow B/S$" is strong, then (1) $\sigma(A \wedge B/S) \geq \sigma'$, (and thus, $\sigma(A/S) \geq \sigma'$, and $\sigma(B/S) \geq \sigma'$), and (2) $\varphi(A \rightarrow B/S) \geq \varphi'$, at its corresponding level. It also represents a filtering process which confines the patterns to be examined at lower levels to be only those with large supports at their corresponding high levels (and thus avoids the generation of many meaningless combinations formed by the descendants of the small patterns). For example, in a sales_transaction data set, if milk is a large pattern, its lower level patterns such as 2% milk will be examined; whereas if fish is a small pattern, its descendants such as salmon will not be examined further.

Based on this definition, the idea of mining multiple-level association rules is illustrated below.

Example 2.1 Suppose that a shopping transaction database consists of two relations: (1) a *sales_item* (description) relation (Table 1), which consists of a set of attributes: *bar_code, category, brand, content, size, storage_period, price*, and (2) a *sales_transaction* table (Table 2), which registers for each transaction the transaction number and the set of items purchased.

Let the query be to find multiple-level strong associations in the database for the purchase patterns related to the foods which can only be stored for less than three weeks. The query can be expressed as follows in an SQL-like data mining language [7].

```
discover association rules
from sales_transactions T, sales_item I
where T.bar_code = I.bar_code and I.category =
      "food" and I.storage_period < 21
with interested attributes category, content, brand
```

The query is first transformed into a standard SQL query which retrieves all the data items within the

bar_code	category	brand	content	size	storage_pd	price
17325	milk	Foremost	2%	1 (ga.)	14 (days)	$3.89
...	...	...	...	...	...	...

Table 1: A sales_item (description) relation

transaction_id	bar_code_set
351428	{17325, 92108, 55349, 88157, ...}
982510	{92458, 77451, 60395, ...}
...	{..., ...}

Table 2: A sales_transaction table

"food" category (covers high level concepts: *beverage, fruit, vegetable, bread, milk, meat, fish, cereal*, etc.) and with the storage period less than 21 days.

GID	bar_code_set	category	content	brand
112	{17325, 31414, 91265}	milk	2%	Foremost
141	{29563, 77454, 89157}	milk	skim	Dairyland
171	{73295, 99184, 79520}	milk	chocolate	Dairyland
212	{88452, 35672, 31205}	bread	wheat	Wonder
...	{..., ...}	...	...	...
711	{32514, 78152}	fruit_juice	orange	Minute_maid

Table 3: A generalized sales_item description table

Since there are only three interested attributes, *category, content*, and *brand* in the query, the sales_item description relation is generalized into a generalized sales_item description table, as shown in Table 3, in which each tuple represents a generalized item which is the merge of a group of tuples which share the same values in the interested attributes. For example, the tuples with the same *category, content* and *brand* in Table 1 are merged into one, with their *bar codes* replaced by a bar_code set. Each group is then treated as an atomic item in the generation of the lowest level association rules. For example, the association rule generated regarding to milk will be only in relevance to (at the low concept levels) *brand* (such as Dairyland) and *content* (such as 2%) but not to *size, producer*, etc.

The taxonomy information is provided implicitly in Table 3. Let *category* (such as "milk") represent the first-level concept, *content* (such as "2%") for the second level one, and *brand* (such as "Foremost") for the third level one. The table implies a concept tree like Figure 1.

The process of mining association rules is expected to first discover large patterns and strong association rules at the top-most concept level. Let the minimum support at this level be 5% and the minimum confidence be 50%. One may find the following: a set of single large items (each called a large 1-itemset, with the support ratio in parentheses): "bread (25%), meat (10%), milk (20%), ..., vegetable (30%)", a set of pair-wised large items (each called a large 2-itemset): "⟨vegetable, bread (19%)⟩, ⟨vegetable, milk (15%)⟩, ..., ⟨milk, bread (17%)⟩", etc. and a set of strong association rules, such as "bread → vegetable (76%), ..., milk

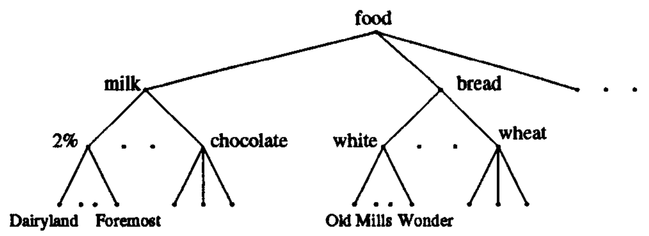

Figure 1: A taxonomy for the relevant data items

→ bread (85%)".

At the second level, only the transactions which contain the large items at the first level are examined. Let the minimum support at this level be 2% and the minimum confidence be 40%. One may find the following large 1-itemsets: "lettuce (10%), wheat bread (15%), white bread (10%), 2% milk (10%), chicken (5%), ..., beef (5%)", and the following large 2-itemsets: "⟨2% milk, wheat bread (6%)⟩, ⟨lettuce, 2% milk (4%)⟩, ⟨chicken, beef (2.1%)⟩", and the strong association rules: "2% milk → wheat bread (60%), ..., beef → chicken (42%)", etc.

The process repeats at even lower concept levels until no large patterns can be found. □

3 A method for mining multiple-level association rules

A method for mining multiple-level association rules is introduced in this section, which uses a hierarchy-information encoded transaction table, instead of the original transaction table, in iterative data mining. This is based on the following considerations. First, a data mining query is usually in relevance to only a portion of the transaction database, such as *food* instead of all the items. It is beneficial to first collect the relevant set of data and then work repeatedly on the task-relevant set. Second, encoding can be performed during the collection of task-relevant data, and thus there is no extra "encoding pass" required. Third, an encoded string, which represents a position in a hierarchy, requires less bits than the corresponding object-identifier or bar-code. Moreover, encoding makes more items to be merged (or removed) due to their identical encoding, which further reduces the size of the encoded transaction table. Thus it is often beneficial to use an encoded table although our method does not rely on the derivation of such an encoded table because the encoding can always be performed on the fly.

To simplify our discussion, an abstract example which simulates the real life example of Example 2.1 is analyzed as follows.

Example 3.1 As stated above, the taxonomy information for each (grouped) item in Example 2.1 is encoded as a sequence of digits in the transaction table $\mathcal{T}[1]$ (Table 4). For example, the item '2% Foremost milk' is encoded as '112' in which the first digit, '1', represents 'milk' at level-1, the second, '1', for '2% (milk)' at level-2, and the third, '2', for the brand 'Foremost' at level-3. Similar to [2], repeated items (i.e., items with the same encoding) at any level will be treated as one item in one transaction.

TID	Items
T_1	{111, 121, 211, 221}
T_2	{111, 211, 222, 323}
T_3	{112, 122, 221, 411}
T_4	{111, 121}
T_5	{111, 122, 211, 221, 413}
T_6	{211, 323, 524}
T_7	{323, 411, 524, 713}

Table 4: Encoded transaction table: $\mathcal{T}[1]$

The derivation of the large item sets at level 1 proceeds as follows. Let the minimum support be 4 transactions (i.e., *minsup*[1] = 4). (Notice since the total number of transactions is fixed, the support is expressed in an absolute value rather than a relative percentage for simplicity). The level-1 large 1-itemset table $\mathcal{L}[1,1]$ can be derived by scanning $\mathcal{T}[1]$, registering support of each generalized item, such as 1**, ..., 4**, if a transaction contains such an item (i.e., the item in the transaction belongs to the generalized item 1**, ..., 4**, respectively), and filtering out those whose accumulated support count is lower than the minimum support. $\mathcal{L}[1,1]$ is then used to filter out (1) any item which is not large in a transaction, and (2) the transactions in $\mathcal{T}[1]$ which contain only small items. This results in a filtered transaction table $\mathcal{T}[2]$ of Figure 2. Moreover, since there are only two entries in $\mathcal{L}[1,1]$, the level-1 large-2 itemset table $\mathcal{L}[1,2]$ may contain only 1 candidate item {1**, 2**}, which is supported by 4 transactions in $\mathcal{T}[2]$.

Level-1 minsup = 4

Level-1 large 1-itemsets: $\mathcal{L}[1,1]$

Itemset	Support
{1**}	5
{2**}	5

Level-1 large 2-itemsets: $\mathcal{L}[1,2]$

Itemset	Support
{1**, 2**}	4

Filtered transaction table: $\mathcal{T}[2]$

TID	Items
T_1	{111, 121, 211, 221}
T_2	{111, 211, 222}
T_3	{112, 122, 221}
T_4	{111, 121}
T_5	{111, 122, 211, 221}
T_6	{211}

Figure 2: Large item sets at level 1 and filtered transaction table: $\mathcal{T}[2]$

According to the definition of ML-association rules,

Level-2 minsup = 3

Level-2 large 1-itemsets:

$\mathcal{L}$[2,1]

Itemset	Support
{11*}	5
{12*}	4
{21*}	4
{22*}	4

Level-2 large 2-itemsets:

$\mathcal{L}$[2,2]

Itemset	Support
{11*, 12*}	4
{11*, 21*}	3
{11*, 22*}	4
{12*, 22*}	3
{21*, 22*}	3

Level-2 large 3-itemsets:

$\mathcal{L}$[2,3]

Itemset	Support
{11*, 12*, 22*}	3

Level-3 minsup = 3

Level-3 large 1-itemsets:

$\mathcal{L}$[3,1]

Itemset	Support
{111}	4
{211}	4
{221}	3

Level-3 large 2-itemsets:

$\mathcal{L}$[3,2]

Itemset	Support
{111, 211}	3

Figure 3: Large item sets at levels 2 and 3

only the descendants of the large items at level-1 (i.e., in $\mathcal{L}$[1,1]) are considered as candidates in the level-2 large 1-itemsets. Let *minsup*[2] = 3. The level-2 large 1-itemsets $\mathcal{L}$[2,1] can be derived from the filtered transaction table $\mathcal{T}$[2] by accumulating the support count and removing those whose support is smaller than the minimum support, which results in $\mathcal{L}$[2,1] of Figure 3. Similarly, the large 2-itemset table $\mathcal{L}$[2,2] is formed by the combinations of the entries in $\mathcal{L}$[2,1], together with the support derived from $\mathcal{T}$[2], filtered using the corresponding threshold. The large 3-itemset table $\mathcal{L}$[2,3] is formed by the combinations of the entries in $\mathcal{L}$[2,2] (which has only one possibility {11*, 12*, 22*}), and a similar process.

Finally, $\mathcal{L}$[3,1] and $\mathcal{L}$[3,2] at level 3 are computed in a similar process, with the results shown in Figure 3. The computation terminates since there is no deeper level requested in the query. Note that the derivation also terminates when an empty large 1-itemset table is generated at any level. □

The above discussion leads to the following algorithm for mining strong ML-association rules.

Algorithm 3.1 (ML_T2L1)
Find multiple-level large item sets for mining strong ML association rules in a transaction database.

Input: (1) $\mathcal{T}$[1], a hierarchy-information-encoded and task-relevant set of transaction database, in the format of $\langle TID, Itemset \rangle$, in which each item in the *Itemset* contains encoded concept hierarchy information, and (2) the minimum support threshold (*minsup*[l]) for each concept level l.

Output: Multiple-level large item sets.

Method: A top-down, progressively deepening process which collects large item sets at different concept levels as follows.

Starting at level 1, derive for each level l, the large k-items sets, $\mathcal{L}[l,k]$, for each k, and the large item set, $\mathcal{LL}[l]$ (for all k's), as follows (in the syntax similar to C and Pascal, which should be self-explanatory).

```
(1)  for ($l := 1$; $\mathcal{L}[l,1] \neq \emptyset$ and $l < max\_level$; $l$++) do {
(2)    if $l = 1$ then {
(3)      $\mathcal{L}[l,1] := get\_large\_1\_itemsets(\mathcal{T}[1], l)$;
(4)      $\mathcal{T}[2] := get\_filtered\_t\_table(\mathcal{T}[1], \mathcal{L}[1,1])$;
(5)    }
(6)    else $\mathcal{L}[l,1] := get\_large\_1\_itemsets(\mathcal{T}[2], l)$;
(7)    for ($k := 2$; $\mathcal{L}[l,k-1] \neq \emptyset$; $k$++) do {
(8)      $C_k := get\_candidate\_set(\mathcal{L}[l,k-1])$;
(9)      foreach transaction $t \in \mathcal{T}[2]$ do {
(10)       $C_t := get\_subsets(C_k, t)$;
(11)       foreach candidate $c \in C_t$ do c.support++;
(12)     }
(13)     $\mathcal{L}[l,k] := \{c \in C_k | c.support \geq minsup[l]\}$
(14)   }
(15)   $\mathcal{LL}[l] := \bigcup_k \mathcal{L}[l,k]$;
(16) } □
```

Explanation of Algorithm 3.1.

According to Algorithm 3.1, the discovery of large support items at each level l proceeds as follows.

1. At level 1, the large 1-itemsets $\mathcal{L}[l,1]$ is derived from $\mathcal{T}$[1] by "*get_large_1_itemsets*($\mathcal{T}$[1], l)". At any other level l, $\mathcal{L}[l,1]$ is derived from $\mathcal{T}$[2] by "*get_large_1_itemsets*($\mathcal{T}$[2], l)", and notice that when $l > 2$, only the item in $\mathcal{L}[l-1,1]$ will be considered when examining $\mathcal{T}$[2] in the derivation of the large 1-itemsets $\mathcal{L}[l,1]$. This is implemented by scanning the items of each transaction t in $\mathcal{T}$[1] (or $\mathcal{T}$[2]), incrementing the support count of an item i in the itemset if i's count has not been incremented by t. After scanning the transaction table, filter out those items whose support is smaller than *minsup*[l].

2. The filtered transaction table $\mathcal{T}$[2] is derived by "*get_filtered_t_table*($\mathcal{T}$[1], $\mathcal{L}$[1, 1])", which uses $\mathcal{L}$[1,1] as a filter to filter out (1) any item which is not large at level 1, and (2) the transactions which contain no large items.

3. The large k (for $k > 1$) item set table at level l is derived in two steps:

   (a) Compute the candidate set from $\mathcal{L}[l,k-1]$, as done in the *apriori candidate generation algorithm* [2], apriori-gen, i.e., it first generates a set C_k in which each item set consists of k items, derived by joining two $(k-1)$ items in $\mathcal{L}[l,k]$ which share $(k-2)$ items, and then removes a k-itemset c from C_k if there exists a c's $(k-1)$ subset which is not in $\mathcal{L}[l,k-1]$.

(b) For each transaction t in $\mathcal{T}[2]$, for each of t's k-item subset c, increment c's support count if c is in the candidate set C_k. Then collect into $\mathcal{L}[l,k]$ each c (together with its support) if its support is no less than *minsup*$[l]$.

4. The large itemsets at level l, $\mathcal{LL}[l]$, is the union of $\mathcal{L}[l,k]$ for all the k's. □

After finding the large itemsets, the set of association rules for each level l can be derived from the large itemsets $\mathcal{LL}[l]$ based on the minimum confidence at this level, *minconf*$[l]$. This is performed as follows [2]. For every large itemset r, if a is a nonempty subset of r, the rule "$a \rightarrow r - a$" is inserted into *rule_set*$[l]$ if *support*(r)/*support*$(a) \geq$ *minconf*$[l]$, where *minconf*$[l]$ is the minimum confidence at level l.

Algorithm ML_T2L1 inherits several important optimization techniques developed in previous studies at finding association rules [1, 2]. For example, *get_candidate_set* of the large k-itemsets from the known large $(k-1)$-itemsets follows *apriori-gen* of Algorithm Apriori [2]. Function *get_subsets*(C_k, t) is implemented by a hashing technique from [2]. Moreover, to accomplish the new task of mining multiple-level association rules, some interesting optimization techniques have been developed, as illustrated below.

1. Generalization is first performed on a given item description relation to derive a generalized item table in which each tuple contains a set of item identifiers (such as bar_codes) and is encoded with concept hierarchy information.

2. The transaction table $\mathcal{T}$ is transformed into $\mathcal{T}[1]$ with each item in the itemset replaced by its corresponding encoded hierarchy information.

3. A filtered transaction $\mathcal{T}[2]$ which filters out small items at the top level of $\mathcal{T}[1]$ using the large 1-itemsets $\mathcal{L}[1,1]$ is derived and used in the derivation of large k-items for any k $(k > 1)$ at level-1 and for any k $(k \geq 1)$ for level l $(l > 1)$.

4. From level l to level $(l+1)$, only large items at $\mathcal{L}[l,1]$ are checked against $\mathcal{T}[2]$ for $\mathcal{L}[l+1,1]$.

Notice that in the processing, $\mathcal{T}[1]$ needs to be scanned twice, whereas $\mathcal{T}[2]$ needs to be scanned p times where $p = \sum_l k_l - 1$, and k_l is the maximum k such that the k-itemset table is nonempty at level l.

4 Variations of the Algorithm for potential performance improvement

Potential performance improvements of Algorithm ML_T2L1 are considered by exploration of the sharing of data structures and intermediate results and maximally generation of results at each database scan, etc. which leads to the following variations of the algorithm: (1) ML_T1LA: using only *one* encoded transaction *table* (thus T1) and generating $\mathcal{L}[l,1]$ for *all* the *levels* at one database scan (thus LA), (2) ML_TML1: using *multiple* encoded transaction *tables* and generating $\mathcal{L}[l,1]$ for *one* corresponding concept *level*, and (3) ML_T2LA: using *two* encoded transaction *tables* ($\mathcal{T}[1]$ and $\mathcal{T}[2]$) and generating $\mathcal{L}[l,1]$ for *all* the *levels* at one database scan.

4.1 Algorithm ML_T1LA

The first variation is to use only one encoded transaction table $\mathcal{T}[1]$, that is, no filtered encoded transaction table $\mathcal{T}[2]$ will be generated in the processing.

At the first scan of $\mathcal{T}[1]$, large 1-itemsets $\mathcal{L}[l,1]$ for every level l can be generated in parallel, because the scan of an item i in each transaction t may increase the count of the item in every $\mathcal{L}[l,1]$ if its has not been incremented by t. After the scanning of $\mathcal{T}[1]$, each item in $\mathcal{L}[l,1]$ whose parent (if $l > 1$) is not a large item in the higher level large 1-itemsets or whose support is lower than *minsup*$[l]$ will be removed from $\mathcal{L}[l,1]$.

After the generation of large 1-itemsets for each level l, the candidate set for large 2-itemsets for each level l can be generated by the *apriori-gen* algorithm [2]. The *get_subsets* function will be processed against the candidate sets at all the levels at the same time by scanning $\mathcal{T}[1]$ once, which calculates the support for each candidate itemset and generates large 2-itemsets $\mathcal{L}[l,2]$. Similar processes can be processed for step-by-step generation of large k-item-sets $\mathcal{L}[l,k]$ for $k > 2$.

This algorithm avoids the generation of a new encoded transaction table. Moreover, it needs to scan $\mathcal{T}[1]$ once for generation of each large k-itemset table. Since the total number of scanning of $\mathcal{T}[1]$ will be k times for the largest k-itemsets, it is a potentially efficient algorithm. However, $\mathcal{T}[1]$ may consist of many small items which could be wasteful to be scanned or examined. Also, it needs a large space to keep all $C[l]$ which may cause some page swapping.

Example 4.1 The execution of the same task as Example 3.1 using Algorithm ML_T1LA will generate the same large item sets $\mathcal{L}[l,k]$ for all the l's and k's but in difference sequences (without generating and using $\mathcal{T}[2]$). It first generates large 1-itemsets $\mathcal{L}[l,1]$ for all the l's from $\mathcal{T}[1]$. Then it generates the candidate sets from $\mathcal{L}[l, 1]$, and then derives large 2-itemsets $\mathcal{L}[l, 2]$ by passing the candidate sets through $\mathcal{T}[1]$ to obtain the support count and filter those smaller than *minsup*$[l]$. This process repeats to find k-itemsets for larger k until all the large k-itemsets have been derived. □

4.2 Algorithm ML_TML1

The second variation is to generate multiple encoded transaction tables $\mathcal{T}[1], \mathcal{T}[2], \ldots, \mathcal{T}[max\_l+1]$, where *max_l* is the maximal level number to be examined in the processing.

Similar to Algorithm ML_T2L1, the first scan of $\mathcal{T}[1]$ generates the large 1-itemsets $\mathcal{L}[1, 1]$ which then serves as a filter to filter out from $\mathcal{T}[1]$ any small items or transactions containing only small items. $\mathcal{T}[2]$ is resulted from this filtering process and is used in the generation of large k-itemsets at level 1.

Different from Algorithm ML_T2L1, $\mathcal{T}[2]$ is not repeatedly used in the processing of the lower levels. Instead, a new table $\mathcal{T}[l+1]$ is generated at the processing of each level l, for $l > 1$. This is done by scanning $\mathcal{T}[l]$ to generate the large 1-itemsets $\mathcal{L}[l, 1]$ which serves as a filter to filter out from $\mathcal{T}[l]$ any small items or transactions containing only small items and results in $\mathcal{T}[l+1]$ which will be used for the generation of large k-itemsets (for $k > 1$) at level l and table $\mathcal{T}[l+2]$ at the next lower level. Notice that as an optimization, for each level $l > 1$, $\mathcal{T}[l]$ and $\mathcal{L}[l, 1]$ can be generated in parallel (i.e., at the same scan).

The algorithm derives a new filtered transaction table, $\mathcal{T}[l+1]$, at the processing of each level l. This, though seems costly at generating several transaction tables, may save a substantial amount of processing if only a small portion of data are large items at each level. Thus it may be a promising algorithm in this circumstance. However, it may not be so effective if only a small number of the items will be filtered out at the processing of each level.

Example 4.2 The execution of the same task as Example 3.1 using Algorithm ML_TML1 will generate the same large itemsets $\mathcal{L}[l, k]$ for all the l's and k's but in difference sequences, with the generation and help of the filtered transaction tables $\mathcal{T}[2], \ldots, \mathcal{T}[max\_l+1]$, where *max_l* is the maximum level explored in the algorithm. It first generates the large 1-itemsets $\mathcal{L}[1, 1]$ for level 1. Then for each level l (initially $l = 1$), it generates the filtered transaction table $\mathcal{T}[l+1]$ and the level-$(l+1)$ large 1-itemsets $\mathcal{L}[l+1, 1]$ by scanning $\mathcal{T}[l]$ using $\mathcal{L}[l, 1]$, and then generates the candidate 2-itemsets from $\mathcal{L}[l, 1]$, calculates the supports using $\mathcal{T}[l+1]$, filters those with support less than *minsup*$[l]$, and derives $\mathcal{L}[l, 2]$. The process repeats for the derivation of $\mathcal{L}[l, 3], \ldots, \mathcal{L}[l, k]$. □

4.3 Algorithm ML_T2LA

The third variation uses the same two encoded transaction tables $\mathcal{T}[1]$ and $\mathcal{T}[2]$ as in Algorithm ML_T2L1 but it integrates some optimization techniques considered in the algorithm ML_T1LA.

The scan of $\mathcal{T}[1]$ first generates large 1-itemsets $\mathcal{L}[1, 1]$. Then one more scan of $\mathcal{T}[1]$ using $\mathcal{L}[1, 1]$ will generate a filtered transaction table $\mathcal{T}[2]$ and all the large 1-itemset tables for all the remaining levels, i.e., $\mathcal{L}[l, 1]$ for $1 < l \leq max\_l$ by incrementing the count of every $\mathcal{L}[l, 1]$ at the scan of each transaction and removing small items and the items whose parent is small from $\mathcal{L}[l, 1]$ at the end of the scan of $\mathcal{T}[1]$.

Then the candidate set for the large 2-itemsets at each level l can be generated by the *apriori-gen* algorithm [2], and the *get_subsets* routine will extract the candidate sets for all the level l ($l \geq 1$) at the same time by scanning $\mathcal{T}[2]$ once. This will calculate the support for each candidate itemset and generate large 2-item-sets $\mathcal{L}[l, 2]$ for $l \geq 1$.

Similar processes proceed step-by-step which generates large k-item-sets $\mathcal{L}[l, k]$ for $k > 2$ using the same $\mathcal{T}[2]$.

This algorithm avoids the generation of a group of new filtered transaction tables. It scans $\mathcal{T}[1]$ twice to generate $\mathcal{T}[2]$ and the large 1-itemset tables for all the levels. Then it scans $\mathcal{T}[2]$ once for the generation of each large k-itemset, and thus scans $\mathcal{T}[2]$ in total $k - 1$ times for the generation of all the k-itemsets, where k is the largest such k-itemsets available. Since k-itemsets generation for $k > 1$ is performed on $\mathcal{T}[2]$ which may consist of much less items than $\mathcal{T}[1]$, the algorithm could be a potentially efficient one.

Example 4.3 The execution of the same task as Example 3.1 using Algorithm ML_T2LA will generate the same large itemsets $\mathcal{L}[l, k]$ for all the l's and k's. It first generates large 1-itemsets $\mathcal{L}[l, 1]$ from $\mathcal{T}[1]$, then T[2] and all the large 1-itemsets $\mathcal{L}[2, 1], \ldots, \mathcal{L}[max\_l, 1]$, where *max_l* is the maximum level to be explored. Then it generates the candidate sets from $\mathcal{L}[l, 1]$, and derives large 2-itemsets $\mathcal{L}[l, 2]$ by testing the candidate sets against $\mathcal{T}[2]$ to obtain the support count and filter those with count smaller than *minsup*$[l]$. This process repeats to find k-itemsets for larger k until all the large k-itemsets have been derived. □

5 Performance study

To study the performance of the proposed algorithms, all the four algorithms: *ML_T2L1*, *ML_T1LA*, *ML_TML1*, and *ML_T2LA*, are implemented and tested on a SUN/SPARC-2 workstation with 16 megabytes of main memory.

The testbed consists of a set of synthetic transaction databases generated using a randomized item set generation algorithm similar to that described in [2].

The following are the basic parameters of the generated synthetic transaction databases: (1) the total number of items, I, is 1000; (2) the total number of transactions is 100,000; and (3) 2000 potentially large itemsets are generated and put into the transactions based on some distribution. Table 5 shows the database used, in which S is the average size (# of items in a potential large itemset) of these itemsets, and T is the average size (# of items in a transaction) of a transaction.

Database	S	T	# of transactions	Size(MBytes)
DB1	2	5	100,000	2.7MB
DB2	4	10	100,000	4.7MB

Table 5: Transaction databases

Each transaction database is converted into an encoded transaction table, denoted as $\mathcal{T}[1]$, according to the information about the generalized items in the item description (hierarchy) table. The maximal level of the concept hierarchy in the item table is set to 4. The number of the top level nodes keeps increasing until the total number of items reaches 1000. The fan-outs at the lower levels are selected based on the normal distribution with mean value being $M2$, $M3$, and $M4$ for the levels 2, 3, and 4 respectively, and a variance of 2.0. These parameters are summarized in Table 6.

Item Table	#_nodes at level-1	M2	M3	M4
$I1$	8	5	5	5
$I2$	15	6	3	4

Table 6: Parameters settings of the item description (hierarchy) tables

The testing results presented in this section are on two synthetic transaction databases: one, $T10$ ($DB2$), has an average transaction size (# of item in a transaction) of 10; while the other, $T5$ ($DB1$), has an average transaction size of 5.

Two item tables are used in the testing: the first one, $I1$, has 8, 5, 5 and 5 branches at the levels 1, 2, 3, and 4 respectively; whereas the second, $I2$, has 15, 6, 3 and 4 branches at the corresponding levels.

Figure 4 shows the running time of the four algorithms in relevance to the number of transactions in the database. The test uses the database $T10$ and the item set $I1$, with the minimum support thresholds being $(50, 10, 4, 2)$, which indicates that the minimum support of level 1 is 50%, and that of levels 2, 3 and 4 are respectively 10%, 4%, and 2%.

The four curves in Figure 4 show that ML_T2LA has the best performance, while the ML_T1LA has the worst among the four algorithms under the current threshold setting. This can be explained as follows. Since the first threshold filters out many small 1-itemsets at level 1 which results in a much smaller filtered transaction table $\mathcal{T}[2]$, but the later filter is not so strong and parallel derivation of $\mathcal{L}[l,k]$ without derivation of $\mathcal{T}[3]$ and $\mathcal{T}[4]$ is more beneficial, thus leads ML_T2LA to be the best algorithm. On the other hand, ML_T1LA is the worst since it consults a large $\mathcal{T}[1]$ at every level.

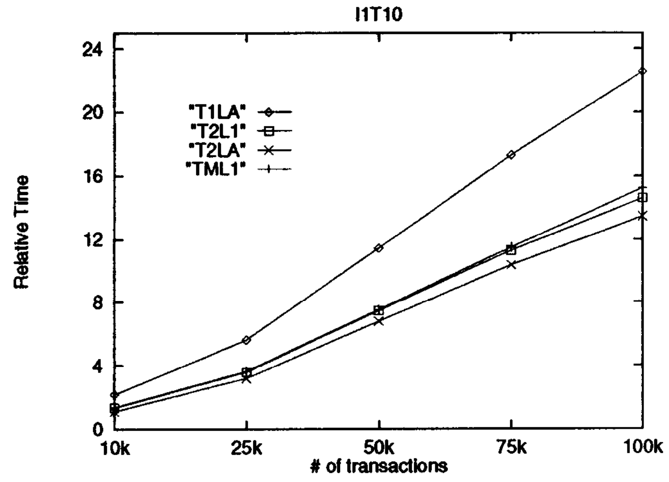

Figure 4: Threshold (50, 10, 4, 2)

Figure 5 shows that ML_T1LA is the best whereas ML_TML1 the worst among the four algorithms under the setting: a different test database $T5$, the same item set $I1$, and with the minimum support thresholds: $(20, 8, 2, 1)$. This is because the first threshold filters out few small 1-itemsets at level 1 which results in almost the same sized transaction table $\mathcal{T}[2]$. The generation of multiple filtered transaction tables is largely wasted, which leads the worst performance of ML_TML1. Thus parallel derivation of $\mathcal{L}[l,k]$ without derivation of any filtered transaction tables applied in ML_T1LA leads to the best performance.

Figure 6 shows that ML_T2L1 and ML_TML1 are closely the best whereas ML_T2LA and ML_T1LA the worst under the setting: a test database $T10$, an item set $I2$, and with the minimum support thresholds: $(50, 10, 5, 2)$. This is because the first threshold filters out relatively more 1-itemsets at level 1 which results in small transaction table $\mathcal{T}[2]$. Thus the generation of multiple filtered transaction tables is relatively beneficial. Meanwhile, the generation of multiple level large 1-itemsets may not save much because one may still obtain reasonably good sized itemsets in the current setting, which leads ML_T2L1 to be the best perfor-

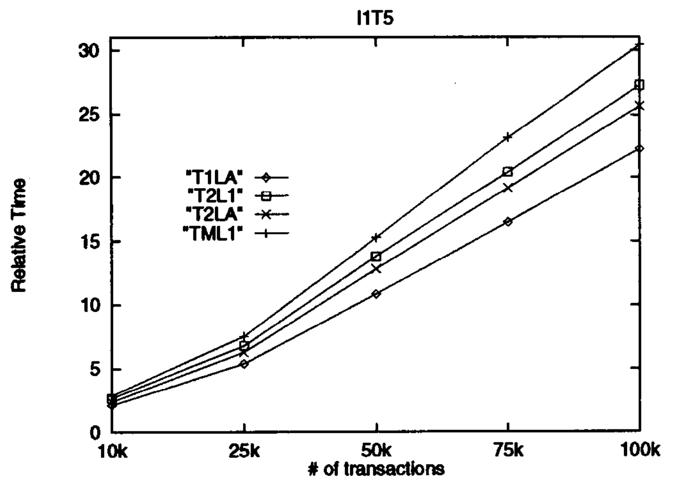

Figure 5: Threshold (20, 8, 2, 1)

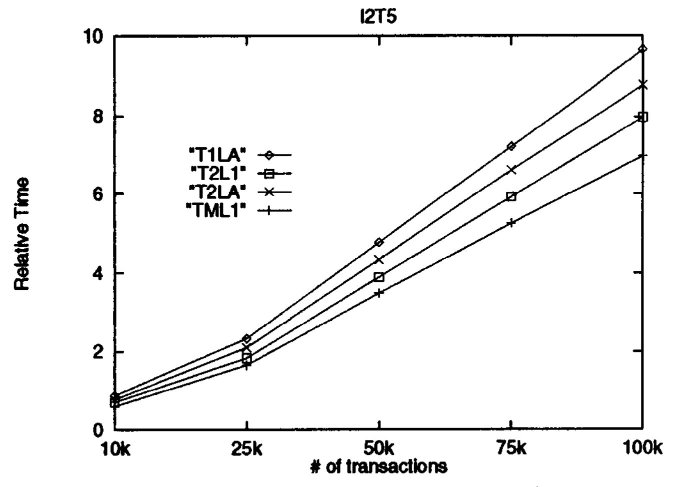

Figure 7: Threshold (30, 15, 5, 2)

mance algorithm.

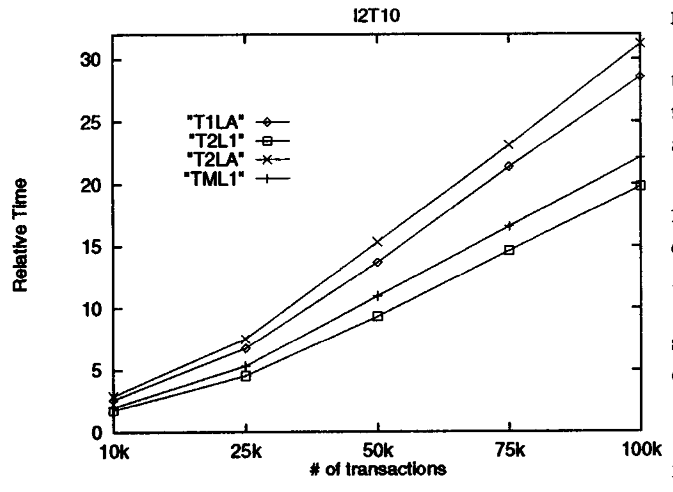

Figure 6: Threshold (50, 10, 5, 2)

Figure 7 shows that ML_TML1 is the best whereas ML_T1LA the worst under the setting: a test database $T5$, an item set $I2$, and with the minimum support thresholds: $(30, 15, 5, 2)$. This is because every threshold filters out relatively many 1-itemsets at each level which results in much smaller transaction tables at each level. Thus the generation of multiple filtered transaction tables is beneficial, which leads to ML_TML1 is the best, and then ML_T2L1, ML_T2LA and ML_T1LA in sequence.

The above four figures show two interesting features. First, the relative performance of the four algorithms under any setting is relatively independent of the number of transactions used in the testing, which indicates that the performance is highly relevant to the threshold setting (i.e., the power of a filter at each level). Thus based on the effectiveness of a threshold, a good algorithm can be selected to achieve good performance. Second, all the algorithms have relatively good "scale-up" behavior since the increase of the number of transactions in the database will lead to approximately the linear growth of the processing time, which is desirable in the processing of large transaction databases.

Figure 8 shows the running time of the four algorithms in relevance to the minimum support thresholds. The test uses the database $T10$ and the item set $I2$, with a sequence of threshold settings: *thre1*, ..., *thre6*. The setting of *thre1* is $(60, 15, 5, 2)$ (with the same notational convention). The remaining threshold settings are as follows: *thre2*: $(55, 15, 5, 2)$, *thre3*: $(55, 10, 5, 2)$, *thre4*: $(50, 10, 5, 2)$, *thre5*: $(50, 10, 5, 1)$, *thre6*: $(50, 5, 2, 1)$. The value-decreasing sequence of minimum support thresholds indicates that weaker filtering mechanism is applied to the later portion of the sequence.

The relative performance of the four algorithms shows the interesting trends of growth as indicated by the four curves in Figure 8. The stronger the filtering mechanism, the more 1-itemsets are filtered out at each level, and the smaller large 1-itemsets are resulted in. Thus ML_TML1, which generates a sequence of filtered transaction tables, has the lowest cost at *thre1*, *thre2* and also (but marginally) *thre3*, but the highest cost at *thre5* and *thre6* (since few items are filtered out). On the contrary, ML_T1LA, which uses only one encoded transaction table but generates the large 1-itemsets for each level at the beginning has the highest cost at *thre1*, *thre2* and *thre3*, but the lowest cost at *thre6*. The other two algorithms stand in the middle with ML_T2LA performs the best at *thre5* when the threshold is reasonable small, especially at the lower levels, and ML_T2L1 performs the best at *thre4* when the threshold is reasonable small but the lowest level is not as small as *thre5*. Since ML_T2LA scans $\mathcal{T}[1]$ twice and needs to maintain all large itemsets $\mathcal{L}[l,k]$ at the same time, it is outperformed by ML_T2L1 when the thresholds are big enough so that a substantial amount of $\mathcal{T}[1]$ is cut and the maximal length of large itemsets at each level is small. Moreover, one may observe the significant performance degradation from *thre4* to *thre5*. This, based on our speculation, is because of the limited size of main memory which may cause substantial page swapping when the support threshold is dropped significantly.

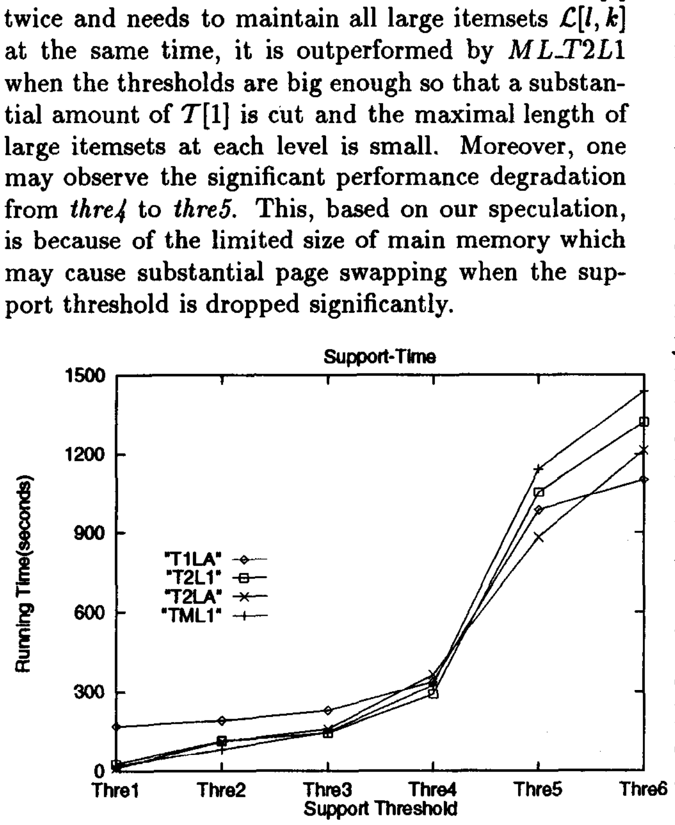

Figure 8: Different thresholds

6 Discussion

6.1 More about concept hierarchies

In our discussions, we have assumed desired concept hierarchies exist and are presented in the form of relational tables (e.g., *sales_item* in Table 1). However, there are often cases that portions of concept hierarchies do not exist. For example, the hierarchy relationships, such as "peanuts, pistachios, ..., walnuts $\subset$ nuts", may not be stored in the sales_item relation. Therefore, it is often necessary for experts or users to specify portions of hierarchies to facilitate mining multiple-level association rules. Specified hierarchies can be mapped into relations with the paths from high-level general concepts to low-level specific ones registered in tuples. Null values should be allowed in the mapped relational entries if there exist unbalanced nodes in a hierarchy.

Notice that there may often exist more than one possible way of mapping a relation into a concept hierarchy. For example, "2% Foremost milk $\subset$ 2% milk $\subset$ milk" and "2% Foremost milk $\subset$ Foremost milk $\subset$ milk" are both meaningful hierarchies, but "2% Foremost milk $\subset$ 2% Foremost $\subset$ Foremost" may not be. An expert or a user may provide mapping rules at the schema level (i.e., meta-rules) to indicate meaningful or desired mappings, such as "{content, brand, category} $\subset$ {content, category} $\subset$ category", etc.

Concept hierarchies may not exist for numerical valued attributes but can be automatically generated according to data distribution statistics [8, 5]. For example, a hierarchy for the price range of sales items can be generated based on the distribution of price values. Moreover, a given concept hierarchy for numerical or nonnumerical data can be dynamically adjusted based on data distribution [8]. For example, if there are many distinct country names in the attribute "place_made", countries can be grouped into continents, such as *Asia, Europe, South_America*, etc. Moreover, if most fresh food products are from *B.C.* and *Northwest America*, the geographic hierarchy can be automatically adjusted to reflect this distribution when studying fresh food products [8].

6.2 Generation of flexible association rules

Our study has been confined to mining association relationships level-by-level in a fixed hierarchy. However, it is often necessary or desirable to find flexible association rules not confined to a strict, pre-arranged concept hierarchies.

First, one may wish to find associations among the concepts associated with alternative, multiple hierarchies. For example, following the hierarchy given in Example 2.1, one may find relationships like "2% milk $\rightarrow$ wheat bread". Alternatively, one may like to find "Foremost milk $\rightarrow$ Wonder bread" or "2% milk $\rightarrow$ Wonder bread", which may require alternative concept hierarchy structures. It seems to be challenging to explore so many alternatives since there may exist only a small number of fixed hierarchies in a database. However, the algorithms presented in this study can be modified minorly to meet the challenge since the new requirement essentially associates the patterns in some alternative generalized forms, such as $\langle\{1{*}2\}, \{2{*}1\}\rangle$, $\langle\{12{*}\}, \{2{*}1\}\rangle$, etc.

Second, one may relax the restriction of mining strong associations among the concepts at the *same level* of a hierarchy to allow the exploration of "*level-crossing*" association relationships. This relaxation may lead to the discovery of associations like "2% Foremost milk $\rightarrow$ Wonder bread" in which the two concepts are at different levels of a hierarchy. This can be achieved by minorly modifying our algorithms since the new requirement associates the patterns like $\langle\{112, 2{*}1\}\rangle$, as demonstrated in the example below.

Example 6.1 For the same transaction tables and concept hierarchies given in Example 3.1, we examine the mining of strong multiple-level association rules which includes nodes at different levels in a hierarchy.

Let minimum support at each level be: minsup = 4 at level-1, and minsup = 3 at levels 2 and 3.

The derivation of the large itemsets at level 1 proceeds in the same way as in Example 3.1, which generates the same large itemsets tables $\mathcal{L}[1,1]$ and $\mathcal{L}[1,2]$ at level 1 and the same filtered transaction table $\mathcal{T}[2]$, as shown in Figure 2.

The derivation of level-2 large itemsets generates the same large 1-itemsets $\mathcal{L}[2,1]$ as shown in Figure 9. However, the candidate items are not confined to pairing only those in $\mathcal{L}[2,1]$ because the items in $\mathcal{L}[2,1]$ can be paired with those in $\mathcal{L}[1,1]$ as well, such as {11*, 1**} (for potential associations like "milk → 2% milk"), or {11*, 2**} (for potential associations like "2% milk → bread"). These candidate large 2-itemsets will be checked against $\mathcal{T}[2]$ to find large items (for the level-mixed nodes, the minimum support at a lower level, i.e., minsup[2], can be used as a default). Such a process generates the large 2-itemsets table $\mathcal{L}$[2,2] as shown in Figure 9.

Notice that the table does not include the 2-item pairs formed by an item with its own ancestor such as ⟨{11*, 1**}, 5⟩ since its support must be the same as its corresponding large 1-itemset in $\mathcal{L}$[2,1], i.e., ⟨{11*}, 5⟩, based on the set containment relationship: any transaction that contains {11*} must contain {1**} as well.

Similarly, the level 2 large 3-itemsets $\mathcal{L}[2,3]$ can be computed, with the results shown in Figure 9. Also, the entries which pair with their own ancestors are not listed here since it is contained implicitly in their corresponding 2-itemsets. For example, ⟨{11*, 12*}, 4⟩ in $\mathcal{L}$[2,2] implies ⟨{11*, 12*, 1**}, 4⟩ in $\mathcal{L}$[2,3].

Level-2 minsup = 3

Level-2 large 1-itemset: $\mathcal{L}$[2,1]

Itemset	Support
{11*}	5
{12*}	4
{21*}	4
{22*}	4

Level-2 large 2-itemset: $\mathcal{L}$[2,2]

Itemset	Support
{11*, 12*}	4
{11*, 21*}	3
{11*, 22*}	4
{12*, 22*}	3
{21*, 22*}	3
{11*, 2**}	4
{12*, 2**}	3
{21*, 1**}	3
{22*, 1**}	4

Level-2 large 3-itemset: $\mathcal{L}$[2,3]

Itemset	Support
{11*, 12*, 22*}	3
{21*, 22*, 1**}	3

Figure 9: Large Item Sets at Level 2

Finally, the large 1-itemset table at level 3, $\mathcal{L}$[3,1], should be the same as Figure 3. The large 2-itemset table includes more itemsets since these items can be paired with higher level large items, which leads to the large 2-itemsets $\mathcal{L}$[3, 2] and large 3-itemsets $\mathcal{L}$[3, 3] as shown in Figure 10. Similarly, the itemsets {111, 11*} and {111, 1**} have the same support as {111} in $\mathcal{L}$[3, 1] and are thus not included in $\mathcal{L}$[3,2].

Since the large k-itemset (for $k > 1$) tables do not explicitly include the pairs of items with their own ancestors, attention should be paid to include them at the generation of association rules. However, since the existence of a special item always indicates the existence of an item in that class, such as "2% milk → milk (100%)", such trivial rules should be eliminated. Thus, only nontrivial implications, such as "milk → 2% milk (70%)", will be considered in the rule generation. □

Level-3 minsup = 3

Level-3 large 1-itemset: $\mathcal{L}$[3,1]

Itemset	Support
{111}	4
{211}	4
{221}	3

Level-3 large 2-itemset: $\mathcal{L}$[3,2]

Itemset	Support
{111, 211}	3
{111, 21*}	3
{111, 22*}	3
{111, 2**}	4
{11*, 211}	3
{1**, 211}	3

Level-3 large 3-itemset: $\mathcal{L}$[3,3]

Itemset	Support
{111, 21*, 22*}	3

Figure 10: Large Item Sets at Level 3

6.3 User interface for mining association rules

In many applications, users may be only interested in the associations among a subset of items in a large database (e.g., associations among foods but not between foods and tires). It is important to provide a flexible interface for users to specify their interested set of data, adjust the thresholds, and interactively discover interesting association relationships.

The query in Example 2.1 is an example of specifying association rule mining tasks. Besides a general claim of mining association rules, a user may also like to specify the discovery of associations among or between specific groups of data. For example, the following query indicates that the user is interested only in discovering the association relationships *between* milk and bread.

```
discover association rules
between I.category = "milk" and I.category =
        "bread"
from sales_transactions T, sales_item I
where T.bar_code = I.bar_code
with interested attributes category, content, brand
```

Since the query requires to find multiple-level large

2-itemsets only, the rule mining algorithm needs to be modified accordingly, however, it will preserve the same spirit of sharing structures and computations among multiple levels.

Graphical user interface is recommended for dynamic specification and adjustment of a mining task and for level-by-level, interactive, and progressive mining of interesting relationships. Moreover, graphical outputs, such as graphical representation of discovered rules with the corresponding levels of the concept hierarchies may substantially enhance the clarity of the presentation of multiple-level association rules.

7 Conclusions

We have extended the scope of the study of mining association rules from single level to multiple concept levels and studied methods for mining multiple-level association rules from large transaction databases. A top-down progressive deepening technique is developed for mining multiple-level association rules, which extends the existing single-level association rule mining algorithms and explores techniques for sharing data structures and intermediate results across levels. Based on different sharing techniques, a group of algorithms, notably, ML_T2L1, ML_T1LA, ML_TML1 and ML_T2LA, have been developed. Our performance study shows that different algorithms may have the best performance for different distributions of data.

Related issues, including concept hierarchy handling, methods for mining flexible multiple-level association rules, and adaptation to difference mining requests are also discussed in the paper. Our study shows that mining multiple-level association rules from databases has wide applications, and efficient algorithms can be developed for discovery of interesting and strong such rules in large databases.

Extension of methods for mining single-level knowledge rules to multiple-level ones poses many new issues for further investigation. For example, with the recent developments on mining single-level sequential patterns [3] and metaquery guided data mining [16], mining multiple-level sequential patterns and metaquery guided mining of multiple-level association rules are two interesting topics for future study.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pp. 207–216, Washington, D.C., May 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pp. 487–499, Santiago, Chile, Sept. 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, Taipei, Taiwan, March 1995.

[4] A. Borgida and R. J. Brachman. Loading data into description reasoners. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pp. 217–226, Washington, D.C., May 1993.

[5] W. W. Chu and K. Chiang. Abstraction of high level concepts from numerical values in databases. In *AAAI'94 Workshop on Knowledge Discovery in Databases*, pp. 133–144, Seattle, WA, July 1994.

[6] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pp. 461–465, Seattle, Washington, July 1987.

[7] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

[8] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *AAAI'94 Workshop on Knowledge Discovery in Databases*, pp. 157–168, Seattle, WA, July 1994.

[9] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int'l Conf. on Information and Knowledge Management*, pp. 401–408, Gaithersburg, Maryland, Nov. 1994.

[10] H. Mannila and K-J. Raiha. Dependency inference. In *Proc. 1987 Int. Conf. Very Large Data Bases*, pp. 155–158, Brighton, England, Sept. 1987.

[11] R. S. Michalski and G. Tecuci. *Machine Learning, A Multistrategy Approach, Vol. 4*. Morgan Kaufmann, 1994.

[12] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, San Jose, CA, May 1995.

[13] G. Piatesky-Shapiro and C. J. Matheus. The interestingness of deviations. In *AAAI'94 Workshop on Knowledge Discovery in Databases*, pp. 25–36, Seattle, WA, July 1994.

[14] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pp. 229–238. AAAI/MIT Press, 1991.

[15] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1992.

[16] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1995.