International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:4, 2007

Multiple-Level Sequential Pattern Discovery from Customer Transaction Databases

An Chen and Huilin Ye

***Abstract*—**Mining sequential patterns from large customer transaction databases has been recognized as a key research topic in database systems. However, the previous works more focused on mining sequential patterns at a single concept level. In this study, we introduced concept hierarchies into this problem and present several algorithms for discovering multiple-level sequential patterns based on the hierarchies. An experiment was conducted to assess the performance of the proposed algorithms. The performances of the algorithms were measured by the relative time spent on completing the mining tasks on two different datasets. The experimental results showed that the performance depends on the characteristics of the datasets and the pre-defined threshold of minimal support for each level of the concept hierarchy. Based on the experimental results, some suggestions were also given for how to select appropriate algorithm for a certain datasets.

***Keywords*—**Data Mining, Multiple-Level Sequential Pattern, Concept Hierarchy, Customer Transaction Database.

I. Introduction

Data Mining, which is also referred to as knowledge discovery from databases, means a process of nontrivial extraction of implicit, previously unknown and potentially useful information from databases. The discovered knowledge can be applied to information management, query processing, decision making, process control, and many other applications [1]. Since association rule mining between items over basket databases was first introduced [2], there has been considerable work devoted to the development of efficient algorithms for mining sequential patterns and association rules [3] [4]. Data records often contain customer information (e.g., customer-id, transaction-time etc.), particularly when the purchases have been made using credit cards or other kinds of customer cards. It is useful to find the sequential patterns that most frequently occur in a customer transaction databases to find some rules of the purchases. For example, in a share market, if many customers bought AT&T share, followed by IBM share, and followed by DEC share in one month in a share transaction database, then <AT&T, IBM, DEC> is a sequential pattern with large possibility.

A customer transaction database records the items purchased by the customers. Usually these items can be organized into a concept hierarchy according to a taxonomy. Based on the hierarchy, associate patterns can be found not only from the leaf nodes (i.e. the items) of the hierarchy, but also can be found at any level of the hierarchy. This is called multiple-level sequential patterns discovery, or mining generalized association patterns. Previous work has been focused on mining sequential patterns at a single concept level [5]. Now the necessity for mining multiple-level association patterns using concept hierarchies has been observed as finding sequential patterns at multiple concept levels is useful in many applications. The sequential patterns at lower concept levels often carry more specific and concrete information and those at higher concept levels carry more general information. This requires progressively deepening the mining process to multiple concept levels. In many cases, concept hierarchies over items are available. Given a set of transactions and a concept hierarchy over items contained in the transactions, association patterns at any level of the hierarchy can be found by developing appropriate algorithms.

In this paper, a method for mining multiple-level sequential patterns is proposed. This method includes three major steps:

(1) Transforming the original customer transaction database into a customer-sequence table: This sequence table will be used as the data resource for the mining. Each sequence in the table consists of all the transactions of a customer while each transaction consists of a set of items purchased at one time. All the items are encoded based on the hierarchical structure.

(2) Finding large sequences at each concept level using a top-down, progressively deepening process: The identification of large sequences depends on the pre-defined threshold of minimum support for each level. For a certain level, if the number of a certain kind sequence is not less than the threshold, this kind sequence is called large sequence.

(3) Identifying sequential patterns and sequential rules: Based on the results from Step 2, sequential patterns and rules can be identified.

Step 2 is crucial. We developed three algorithms for identifying large sequences. An experiment has been conducted to compare the performances of the algorithms. The

Manuscript received October 23, 2003.

Dr. A. Chen is with the Institute of Policy and Management, Chinese Academy of Sciences, Beijing, 100080, P. R. China and was a visiting scholar at School of Electrical Engineering and Computer Science, the University of Newcastle, 2308, Australia (e-mail: anchen@otcaix.iscas.ac.cn)

Dr. H. Ye is with the School of Electrical Engineering and Computer Science, the University of Newcastle, 2308, Australia (phone: +61 2 4921 6167; fax: +61 2 4921 6929; e-mail: hye@cs.newcastle.edu.au).

performances were measured by the relative time spent on completing the mining tasks on two different datasets. The experimental results showed that the performance depends on the characteristics of the datasets and the pre-defined threshold of minimal support for each level of the concept hierarchy. Different algorithms may have different performances for different datasets.

The paper is organized as follows. Section II gives the definitions of the concepts used in mining sequential patterns. Based on these definitions the problem of mining sequential patterns can be formally characterized. In Section III, a method for mining multiple-level sequential patterns is developed and an example is step-by-step shown to help understand the procedure. One algorithm for find large sequences at every concept level, called MLSeq_T2L1, is presented in pseudo code. Section IV presents two variant algorithms of MLSeq_T2L1 and the experimental results. The performances of the three algorithms are compared and discussed. Finally, Section V concludes this study.

II. Concept Definitions for Mining Sequential Patterns

In a given customer transactions database, each transaction consists of the following fields: customer-id, transaction-time, and the items purchased in the transaction. No customer has more than one transaction at the same transaction-time. We don't consider quantities of items bought in a transaction; each item is a binary variable representing whether an item was bought or not. Let $I = \{i_1, i_2, ..., i_n\}$ be a set of literals. An itemset is a non-empty set of items. A sequence is a non-empty and ordered list of itemsets. We denote an itemset by $(i_1, i_2, ..., i_m)$, where i_j is an item. The length of an itemset is the number of items in it. An itemset of length k is called a k-itemset. We denote a sequence S by $<s_1, s_2, ..., s_n>$, where s_j is an itemset. The length of a sequence is the number of itemsets in it. A sequence of length k is called a k-sequence. The sequence formed by the concatenation of two sequences A and B is denoted as $<A, B>$.

Definition 2.1: All the transactions of a customer can together be viewed as a sequence, where each transaction corresponds to a set of items, and the list of transactions, ordered by increasing transaction-time, corresponds to a sequence. A transaction made at transaction-time *Ti* can be denoted as *itemset* (T_i). Thus, the sequence of the transactions made by a customer, ordered by increasing transaction-time $T_1, T_2, \ldots, T_n$, can be denoted by $<itemset(T_1), itemset(T_2), \ldots, itemset(T_n)>$ which is called customer-sequence.

Definition 2.2: An itemset X is contained in a transaction T if $X \subseteq T$. A sequence $A = <a_1, a_2, ..., a_m>$ is contained in another sequence $B = <b_1, b_2, ..., b_n>$ (i.e., A is a subsequence of B) if there exist integers $i_1 < i_2 < \ldots < i_m$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}, \ldots, a_m \subseteq b_{im}$. In a set of sequences, a sequence S is maximal if S is not contained in any sequences in the set.

Definition 2.3: A customer supports an itemset X if X is contained in at least one transaction of the customer-sequence for this customer. The support for X is defined as the fraction of total customers who support X. The support count for X, denoted by *X.support*, is defined as the number of customers who support X. A customer supports a sequence S if S is contained in the customer-sequence for this customer. The support for S is defined as the fraction of total customers who support S. The support count for S, denoted by *S.support*, is defined as the number of customers who support S.

Definition 2.4: A concept hierarchy is a tree describing the relation of the concepts from the most generalized level concept to primitive level. Each node in the tree represents a concept and an edge represents an *is-a* relationship between two concepts. The root, called the first level of the tree, is the most generalized concept and the leaves, called the last level of the tree, are the most concrete concepts. Let x and y be nodes in the concept tree. If there is path from x to y, we call x an ancestor of y or y a descendant of x. If x is the nearest ancestor of y (i.e., there is an edge directly from x to y), we call x a parent of y or y a child of x. Concept hierarchies are given by domain experts and stored in the database or automatically produced by the system.

Definition 2.5: Different minimum support and confidence can be specified at different levels for a concept hierarchy to find multiple-level sequential patterns and rules. Let *minsup*[p] be the minimum support count at level p, an itemset X is large at level p if $X.support \geq minsup[p]$. Large itemset is also called Litemset. Similarly, a sequence S is large at level p if $S.support \geq minsup[p]$. Since each itemset in a large sequence must have minimum support, any large sequence must be a list of Litemsets. A sequence pattern is the maximal sequences in the set of large sequences.

Definition 2.6: A sequential rule is an implication of the form $A \Rightarrow B$, where A and B are sequences and the support count of the rule is $<A, B>.support$, defined as the number of customers who support $<A, B>$. The confidence of the rule is defined as $<A, B>.support / A.support$. Confidence denotes the strength of implication and support indicates the occurring frequency of the rule. Let *minconf*[p] be the minimum confidence at level p. A sequential rule at level p is strong if the support and confidence of the rule are not less than the *minsup*[p] and *minconf*[p] respectively.

Some concepts of single-level sequential patterns can be extended to multiple-level sequential patterns.

Definition 2.7: An item x is contained in an itemset X if $x \in X$ or $x' \in X$, where x' is a descendant of x, i.e. x is in X and x is an ancestor of some items in X. An itemset X is contained in another itemset Y if every item of X is contained in Y. A sequence $A = <a_1, a_2, \ldots, a_m>$ is contained in another sequence $B = <b_1, b_2, \ldots, b_n>$, if there exist a set of integers $i_1 < i_2 < \ldots < i_m$, such that a_j contained in b_{ij}.

Definition 2.8: An itemset X' is an ancestor of another itemset X if we can get X' from X by replacing one or more items in X with their ancestors and deleting the identical items. A sequence $S' = <y_1, \ldots, y_m>$ is an ancestor of another

sequence $S=<x1, \ldots, x_m>$ if for $k = 1, \ldots, m$, $y_k = x_k$ or y_k is an ancestor of x_k and S and S' have the same length.

Some properties can be reached based on the above definitions and set theory.

Property 1: If an itemset Y contains another itemset X, then Y also contains Z where Z is an ancestor of X.

Property 2: If a sequence B contains a sequence A, then B also contains C where C is an ancestor of A.

Property 3: If X is a large itemset, then its ancestor X' is also large.

Property 4: If S is a large sequence, then its ancestor S' is also large.

Based on the above concept definitions, the problem of mining multiple-level sequential patterns can be characterized as follows:

Problem Statement: Given a customer transaction database and a concept hierarchy, the problem of mining multiple-level sequential patterns is to discover all maximal sequences that have support not less than the user-specified minimum support and confidence at the corresponding level of the concept hierarchy. In addition, we can find the sequential rules that have support and confidence not less than the user-specified minimum support and minimum confidence at each level of the hierarchy.

III. An Algorithm for Mining Multiple-level Sequential Patterns

In this section, we will present an algorithm of mining multiple-level sequential patters from large customer transaction databases. An example will be shown to help understand the mining procedure.

A. Overview of the Algorithm

The method for mining multiple-level sequential patterns uses customer-sequence tables encoded by the hierarchy information rather than the original customer transaction table. We split the problem of mining multiple-level sequential patterns into the following steps:

1. Coding a concept hierarchy

A customer transaction database records the items purchased by the customers. Usually these items can be organized into a concept hierarchy. Each leaf node in a hierarchy represents an item and the items can be classified into categories from more general levels to more specific levels. We code each node in a concept hierarchy using a top-down coding method starting from the root and gradually down to the leaves. The root of a hierarchy is coded first by being assigned an integer of zero. For a hierarchy of m levels, any non-root node in the hierarchy can be coded based on the following formula:

$$\text{code}(p, i) = \text{COP}(p, i) \times 10 + i$$

where p ($p = 0, 1, \ldots, m-1$) represents the level where the node resides; i ($i = 1, 2, \ldots,$ number of nodes at level p) is the location number of a node at level p, (a set of contiguous integers starting from 1 is assigned to the nodes from left to right as location number); the code (p, i) denotes the code for the i^{th} node at level p, COP (p, i) is the code of the parent of the i^{th} node at level p. An example of a hierarchy and the code for each node are shown in Fig. 1. After the coding, each item recorded in a customer transaction database will be represented by its code.

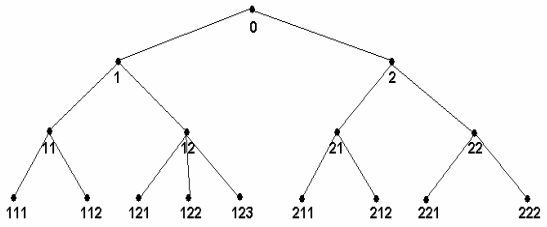

FIGURE 1 AN EXAMPLE OF CONCEPT HIERARCHY

2. Sorting customer transaction databases

A customer transaction database D can be sorted with *customer-id* as the major key and *transaction-time* as the minor key. After the sorting, all the records having the same customer-id and increased transaction-time can be converted to a customer-sequence. Thus the original customer transaction database can be converted to a customer sequence table. Table I is an example of such customer sequence tables and we will take this example to demonstrate the remaining steps of the mining process.

TABLE I
CUSTOMER-SEQUENCE TABLE T[1]

Customer_id	Customer-sequence
1	<(111, 211, 611), (111, 222)>
2	<(112, 431), (111, 231, 712)>
3	<(111, 211, 423), (222, 412)>
4	<(211, 323), (323, 431, 534)>
5	<(211, 222, 311), (323, 411, 712)>
6	<(111, 211, 324), (222, 321,534), (611)>
7	< (222, 324), (421, 612)>
8	<(531), (711)>

3. Finding large sequences at all levels:

A top-down, progressively deepening process which collects large sequences from level 1 to *max_level* (the maximal level of a concept hierarchy) based on a customer-sequence table, called T[1] and a set of pre-defined minimum support called *minsup*[p].

(i) Finding large sequences at level 1

To find large sequences at level 1, first we must fine all the large itemsets at level 1, denoted by L[1] by scanning T[1]. For each generalized candidate itemset at this level, such as 1**, 2**, …, if T[1] contains such an item, the support count for this itemset will be increased by 1. Those itemsets whose accumulated support count is greater than or equal to *minsup*[1] will be selected as large itemsets. Those items whose corresponding 1-itemset's support count is less than

minsup[1] will be removed from T[1]. After the removal, if any customer sequence becomes empty, it will be removed from T[1] too. This results in a filtered transaction table T[2] (shown in Table II) that will be used later for finding large sequences at level 2.

Let's look at the example shown in Table I. The customer number in this example is 8 and the minimal support at level 1 *minsup*[1] is set to 4. Based on the T[1] shown in Table I, the following results can be derived:

Filtered table T[2] is shown in Table II.

Large 1-itemsets at level 1 L[1, 1] = {(1**), (2**), (3**), (4**)}

Large 2-itemsets at level 1 L[1, 2] = {(1**, 2**), (2**, 3**)}

Large 3-itemsets at level 1 L[1, 3] = Ø

Large itemsets at level 1 $L[1]=\bigcup_{k=1}^{3} L[1, k]$ = {(1**), (2**), (3**), (4**), (1**, 2**), (2**, 3**)};

L[1] is shown in Table III where each large itemset is assigned an index starting from 1.

TABLE II
T[2] (FILTERED TABLE FROM T[1])

Customer_id	Customer-sequence
1	<(111, 211), (111, 222)>
2	<(112, 431), (111, 231)>
3	<(111, 211, 423), (222, 412)>
4	<(211, 323), (323, 431)>
5	<(211, 222, 311), (323, 411)>
6	<(111, 211, 324), (222, 321)>
7	<(222, 324), (421)>

TABLE III
L[1] (LARGE ITEMSETS AT LEVEL 1)

Index	Sequence	Support
1	(1**)	4
2	(2**)	7
3	(1**, 2**)	4
4	(3**)	4
5	(4**)	5
6	(2**, 3**)	4

Based on L[1], large 1-sequence at level 1 LL[1,1] ={<*iset*> | *iset* ∈ L[1]} can be identified as:

LL[1,1] = {<(1**)>, <(2**)>, <(3**)>, <(4**)>, <(1**, 2**)>, <(2**, 3**)>}.

Next, we try to find large 2-sequences at level 1. An intermediate table T' is derived from T[2] by replacing each itemset in each customer-sequence contained in T[2] with a set of large itemsets represented by their index in L[1] contained in it. For example, an itemset (111, 211) contains large itemsets (111), (211) and (111, 211) which are indexed as 1, 2, 3 respectively in L[1] (see Table III). It can then be transformed to an indexed representation (1, 2, 3). The transformed table T' is shown in Table IV. Then the set of large 2-sequence at level 1 LL [1, 2] (see Table V) is generated using the similar method to AprioriAll algorithm by scanning T'.

TABLE IV
TRANSFORMED TABLE T' AT LEVEL1

Customer_id	Transformed customer-sequence
1	<(1, 2, 3), (1, 2, 3)>
2	<(1, 5), (1, 2, 3)>
3	<(1, 2, 3, 5), (2, 5)>
4	<(2, 4, 6), (4, 5)>
5	<(2, 4), (4, 5)>
6	<(1, 2, 3, 4, 6), (2, 4, 6)>
7	<(2, 4, 6), (5)>

TABLE V
LL[1, 2] (LARGE 2-SEQUENCES AT LEVEL 1)

Indexed sequence	Actual sequence	Support
<1, 2>	< (1**), (2**)>	4
<2, 5>	<(2**), (4**)>	4

There is no large 3-sequences at level 1, thus LL[1,3]=∅;

All the large sequences at level 1 $LL[1] = \bigcup_{k=1}^{3} LL[1, k]$ = {<(1**)>, <(2**)>, <(3**)>, <(4**)>, <(1**, 2**)>, <(2**, 3**)>, <(1**), (2**)>, <(2**), (4**)>};

(ii) Finding large sequences at level 2

The minimum support at level 2, minsup[2], is set to 3 for the example. Similar to finding large sequences at level 1, a filtered table T[3] generated from T[2] (shown in Table VI) and the large itemsets at level 2 L[2] (shown in Table VII) can be derived.

TABLE VI
T[3] (FILTERED TABLE FROM T[2])

Customer_id	Customer-sequence
1	(111, 211), (111, 222)
2	<(112), (111)>
3	<(111, 211), (222)>
4	<(211, 323), (323)>
5	<(211, 222), (323)>
6	<(111, 211, 324), (222, 321)>
7	<(222, 324)>

TABLE VII
L[2] (LARGE ITEMSETS AT LEVEL 2)

Index	Sequence	Support
1	(11*)	4
2	(21*)	5
3	(11*, 21*)	3
4	(22*)	5
5	(32*)	4

Based on L[2], large 1-sequences at level 2 LL [2, 1] can be obtained based on the following formula:

LL[2,1] ={<*iset*> | *iset* ∈ L[2]} = {<(11*)>, <(21*)>, <(11*, 21*)>, <(22*)>, <(32*)>}

Based on T[2] and the L[2], the transformed table T' for

level 2 (shown in Table VIII) and the large 2-sequences at level 2, LL [2, 2] (shown in Table IX), can be derived.

There is no large 3-sequences at level 2, thus LL[2,3]=∅.

All the large sequences at level 2 LL[2] = $\bigcup_{k=1}^{3}$ LL[2, k] = {<(11*)>, <(21*)>, <(22*)>, <(32*)>, <(11*, 21*)>, <(11*), (22*)>, <(21*), (22*)>, <(21*), (32*)>, <(11*, 21*), (22*)>}.

TABLE VIII
TRANSFORMED TABLE T' AT LEVEL 2

Customer_id	Transformed customer-sequence
1	<(1, 2, 3), (1, 4)>
2	<(1), (1)>
3	<(1, 2, 3), (4)>
4	<(2, 5), (5)>
5	<(2, 4), (5)>
6	<(1, 2, 3, 5), (4, 5)>
7	<(4, 5)>

TABLE IX
LL[2, 2] (LARGE 2-SEQUENCES AT LEVEL 2)

Indexed sequences	Sequences	Support
<1, 4>	< (11*), (22*)>	3
<2, 4>	<(21*), (22*)	3
<2, 5>	<(21*), (32*)>	3
<3, 4>	<(11*, 21*), (22*)>	3

(iii) Finding large sequences at level 3

The processes of finding large sequences at level 3, level 4, …, *max_level* are similar. For the above example, *max_level* = 3 and the minimum support at level 3, *minsup*[3], is set to 3. The following tables, a filtered table T[4] generated from T[3] (shown in Table X), large 1-itemsets at level 3 LL[3, 1] (shown in Table XI), the transformed table T' at level 3 (shown in Table XII), and large 2-sequences LL[3,2] (shown in Table XIII) can be derived.

TABLE X
T[4] (FILTERED TABLE FROM T[3])

Customer_id	Customer-sequence
1	<(111,211), (111,222)>
2	<(111)>
3	<(111,211), (222)>
4	<(211)>
5	<(211, 222)>
6	<(111,211), (222)>
7	<(222)>

TABLE XI
L[3] (LARGE ITEMSETS AT LEVEL 3)

Index	Sequence	Support
1	(111)	4
2	(211)	5
3	(111, 211)	3
4	(222)	5

Based on L[3], large 1-sequences at level 3 LL [3, 1] can be obtained based on the following formula:

LL[3,1] ={<*iset*> | *iset* ∈ L[3]} = {< (111) >, < (211) >, < (111, 211) >, < (222) >}.

TABLE XII
TRANSFORMED TABLE T' AT LEVEL 3

Customer_id	Transformed customer-sequence
1	<(1, 2, 3), (1,4)>
2	<(1)>
3	<(1, 2, 3), (4)>
4	<(2)>
5	<(2, 4)>
6	<(1, 2, 3),(4)>
7	<(4)>

TABLE XIII
LL[3, 2] (LARGE 2-SEQUENCES AT LEVEL 3)

Indexed sequences	Sequences	Support
<1,4>	<(111), (222)>	3
<2,4>	<(211), (222)>	3
<3,4>	<(111, 211), (222)>	3

There is no large 3-sequences at level 3, thus LL[3, 3]=∅;

All the large sequences at level 3 LL[3] = $\bigcup_{k=1}^{3}$ LL[3, k] = {< (111) >, < (211) >, < (222) >, < (111, 211) >, < (111), (222) >, < (211), (222) >, < (111, 211), (222) >}.

4. Identifying sequential patterns and sequential rules

(i) Sequential patterns (maximal sequences of large sequences)

Having found the set of all large sequences LL[*p*] (*p* = 1, 2, …, *max-level*) in Step 3, we use the following algorithm (shown in Fig. 2) to identify maximal sequences from the set of large sequences. Let the length of the longest sequence of LL[p] is *n*. The following function will delete the non-maximal sequences that are contained in other sequence of LL[p]

```
maximal_seq()
{for (p := 1; p <= max_level; p++)  do
      for (k := n; k >= 1; k --)  do
            for each k-large sequence S do
                  delete all subsequences of S from LL[p]

}
```

FIGURE 2 MAXIMAL_SEQ ( ) FUNCTION

Data structure and algorithm to quickly identify all the

subsequences for a given sequence can be found in [6].

(ii) Sequential rules:

We can also use the identified large sequences to generate the desired sequential rules. For every large sequence S at level p, find all non-empty prefix subsequences of S. For every such subsequence A, a rule of the form $A \Rightarrow B$ ($<A, B> = S$) can be identified if *S.support* / *A.support* is great than or equal to *minconf[p]*.

B. Algorithm MLSeq_T2L1: (An algorithm for finding multiple-level large sequences)

Based on the steps presented in the preceding section, the most crucial step of mining multiple level sequential patters and rules is Step 3, i.e. finding large sequences at all levels. The previous discussion leads to the following algorithm *MLSeq_T2L1* for mining multiple-level large sequences. This algorithm is an extension of *ML_T2L1* algorithm [7]. The major variables used in the algorithm and their semantics are listed in Table XIV.

TABLE XIV
NOTATIONS USED IN MLSEQ_T2L1

Variables	Description
L[p, k]	Set of large *k*-itemsets at level *p* Each member of this set has two fields: (i) itemset and (ii) support count
C[p, k]	Set of candidate *k*-itemsets at level *p*
L[p]	Set of all large itemsets at level *p*, $L[p]=\bigcup_{k=1}^{n} L[p,k]$, where *n* is the maximum length of large itemsets at level *p*
LL[p, k]	Set of large *k*-sequences at level *p* Each member of this set has two fields: (i) sequence and (ii) support count
CL[p, k]	Set of candidate *k*-sequences at level *p*
LL[p]	Set of all large sequences at level *p*, $LL[p]=\bigcup_{k=1}^{n} LL[p,k]$, where *n* is the maximum length of large sequences at level *p*
T[p]	Filtered customer-sequence table derived from L[p-1, 1] at level *p-1*

The Inputs of the algorithm are:

(1) T[1]: a customer-sequence table (encoded based on a concept hierarchy) and

(2) minimum support thresholds minsup[*p*] for each concept level *p*.

The output of the algorithm will be the large sequences LL[*p*] at every level *p*. Fig. 3 presents the algorithm describing the process of how to generate the large sequences LL[*p*] for all levels (p =1, 2, …, *max_level*).

```
(0) L[1] = get_large_itemset (T[1], 1);
(1) for (p:=1; L[p,1] ≠ ∅ and p <= max_level; p++) do
        {   (2) T[p+1] = get_filtered_table (T[p], L[p, 1]);
            (3) if (p>1) then
                    (3.1) L [p] = get_large_itemset (T[p], p);
            (4) T' = transform_table (T[p+1]);
            (5) LL[p, 1] = { <iset> | iset ∈ L[p] };
            (6) for (k:=2; LL[p,k-1] ≠ ∅; k++) do
                {
                    (6.1) CL[p,k] = get_candidate_set (LL[p,k-1]);
                    (6.2) for each t∈T' do
                            {
                                (6.3) Ct = get_subsets (CL[p,k],t);
                                (6.4) for each c∈ Ct do
                                (6.5) c.support ++;
                            }
                    (6.6)LL[p,k] := {c∈CL[p,k] | c.support ≥ minsup[p]}
                    }
            (7) LL[p] := ∪k LL[p,k];
    }
```

FIGURE 3 ALGORITHM MLSEQ_T2L1

This algorithm consists of two parts: (a) generating large itemsets, and (b) generating large sequences based on the identified large itemsets. During this process, some intermediate tables will also be derived.

At any level p, large k-itemsets (k=1, 2, …, n) L[p] is derived from T[p] by invoking `get_large_itemset(T[p], p)` function (see Statement (1) and (3) in Fig.3). The filtered customer-sequence table T[p+1] can be derived by invoking `get_filtered_table (T[p], L[p, 1])` function, which uses L[p, 1] as a filter to filter any small items from customer-sequences and to remove the sequences that contain only small items from T[p] (see Statement (2)).

In Statement (4), an intermediate table T' at level p is generated from the transformation from the filtered table T[p+1]. This intermediate table will be used to derive large sequences.

Large 1-sequences at any level LL[p, 1] can be generated based on Statement (5) while large k-sequences (k>1) at level p are derived in two steps (see Statement (6) and its Sub-statements (6.1)- (6.6)):

(a) The candidates of k-sequences are generated from LL[p, k-1] by invoking `get_candidate_set (LL[p, k-1])` function. The function takes LL[p, k-1] as a parameter and returns a superset of the set of all large k-sequences at level p, CL[p, k].

(b) For each customer-sequence t in T', increment the support count of $S \in$ CL[p, k] if S is contained in t. Then LL[p, k] can be derived from those sequences in CL[p, k] whose support is not less than *minsup*[p].

Finally, the large sequences at any level p, LL[p], is the union of LL[p, k] for all k (see Statement (7)).

This algorithm uses several functions, `get_large_itemset()`, `get_filtered_table()`, `transform_table()`, and `get_candidate_set()`. We will give details for one of the functions, `get_large_itemset ()`, in Fig. 4 (the algorithms for other

functions can be derived based on the corresponding description presented in Subsection A):

```
get_large_itemset (T, p)
{  L[p,1] = set of large 1-itemsets at level p;
   for (k:=2; L[p,k-1] ≠∅; k++ ) do {
      C[p,k] = apriori_gen(L[p,k-1]);
      for each customer-sequence c∈T do {
         C_t = get_subsets (C[p,k], c);
         for each c∈C_t do c.support ++;
      }
     L[p,k] = {c∈C[p,k] | c.support ≥ minsup[p]};
   }
  L[p]:=∪_k L[p,k];
}
```

FIGURE 4 `get_large_itemset()` FUNCTION

The `apriori_gen()` function used in the function was borrowed from [8]. This functions takes L[*p*, *k*-1] as an parameter and returns a superset of the set of all large *k*-itemsets.

IV. Some Variant Algorithms and Performance Comparison

Based on the algorithm presented in preceding section, we can derive several variant algorithms by exploring different ways to share the data structures and intermediate results. The performance of the algorithms will be discussed based on the results from an experiment.

A. Algorithm MLSeq_TML1

The major difference between *MLSeq_T2L1* and *MLSeq_TML1* is that *MLSeq_TML1* only derive the filtered table T[2] at level 1 rather than generating all the filtered tables T[*p*+1] at any other level *p* (*p*>1) as did *MLSeq_T2L1*. T[2] is repeatedly used as a filtered table for all the processing at lower levels. The algorithm for *MLSeq_TML1* can be obtained by slightly modifying the *MLSeq_T2L1* algorithm; In Fig. 3 Statement (2) will be removed and the following statement will be inserted between the original Statements (1) and (2).

```
T[2] = get_filtered_table (T[1], L[1, 1]);
```

Statement (3) and (4) will be replaced by the following new statements:

```
(3) if (p>1) then
 (3.1) L[p] = get_large_itemset (T[2], p);
(4) T' = transform_table (T[2]);
```

This algorithm only generates filtered time T[2] and saves the effort of generating all the other filtered tables. However, as the T[2] is only filtered once the processing effort at lower level *p* (*p*>1) may be larger than using the filtered table T[*p*].

B. Algorithm MLSeq_T1LA

MLSeq_T1LA uses only T[1] and no other filtered table is generated. All large 1-itemsets L[*p*, 1] for every level *p* can be generated in parallel in one scan of T[1]. Then each item in L[*p*, 1] (p>1) whose parent is not large in the higher level large 1-itemsets, or whose support is lower than *minsup*[*p*], will be removed from L[*p*, 1]. Large *k*-itemsets L[*p*, *k*] (*k*>1) for each level is generated in parallel by scanning T[1]. After all large itemsets at each level have been identified, T[1] is transformed into T' according to L[*p*]. All large sequences LL[*p*] can then be ientified by scanning T' at every level *p*. *MLSeq_T1LA* generates large *k*-itemsets for each level in parallel, which save the effort for repeat scans of T[1]. However, it has to spend time on examining small itemsets and small items in T[1].

C. Performance Comparison

An experiment was conducted to assess the performance of the proposed three algorithms. Two customer transaction datasets coded by a pre-defined concept hierarchy were generated by using the method specified in [9] [10]. The relevant parameters for the experiment are listed below:

|C|=Average number of transactions per customer
|T|=Average number of items per transaction
|S|=Average length of maximal potentially large sequences
|I|=Average size of itemsets in maximal potentially large sequences

We changed the value of the average items per transaction to generate two datasets and defined two different minimum support thresholds for the two datasets. The three algorithms were applied to the datasets and the experiment run on a personal computer (Pentium-II 866MHz, 64MB). The performances of the three algorithms based on the two datasets are compared in Figure 5 and Figure 6 in terms of the relative time spent for the execution of the algorithms. Both figures show that the execution times increase almost linearly along with the customer transaction number increased from 25,000 to 200,000. As we set different values for the average number of items per transaction and different minimum support thresholds, we will discuss how these two parameters influence the corresponding performances of the algorithms.

In Figure 5 (|C| = 10, |T|=5, |S|= 4, |I|= 1.25, *minsup*[*p*]=0.25), *MLSeq_T1LA* achieved the best performance, while *MLSeq_T2L1* had the worst performance. In this dataset, as the average number of items per transaction |T| and the minimum support threshold are relatively small, not many items have been filtered out at each level. Therefore, the effort devoted to generating the filtered tables is not compensated by the reduced effort for processing the filtered tables at lower levels because the filtered tables are mot significantly smaller than T[1] at all. As *MLSeq_T1LA* did not generate any filtered table and used the T[1] for all the levels, it saved the time for generating the filtered tables and completed the task in the shortest time. *MLSeq_T2L1* generated filtered tables for all levels. Thus, its performance is worse than *MLSeq_TML1* who only generated one filtered table at level 1.

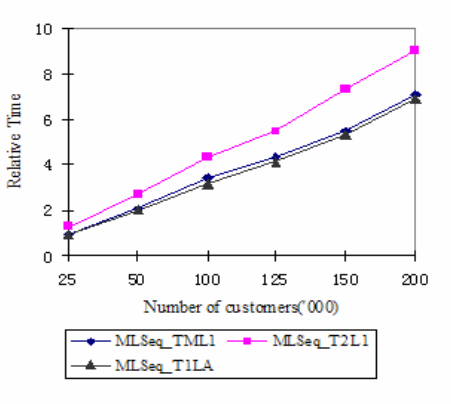

FIGURE 5 PERFORMANCE COMPARISON

In Figure 6 (|C| = 10, |T| = 10, |S|= 4, |I|= 1.25, minsup[p]=0.5), *MLSeq_TML1* achieved the best performance, while *MLSeq_T1LA* has the worst performance. In this dataset, as the average number of items per transaction |T| and the minimum support threshold are larger than the first dataset, a relatively large amount of items may be filtered out at each level. Therefore, the effort devoted to generating the filtered tables is compensated by the reduced effort for processing the much smaller filtered tables at lower levels. As *MLSeq_T1LA* did not generate any filtered table and used the T[1] for all level's processing and T[1] is much larger than the filtered tables, it took the longest time to complete the task.. The performances of *MLSeq_T2L1* and *MLSeq_TML1* are very similar. This may indicate that the effect of the filtration at level 1 is more significant in comparison with the filtration at other levels. As *MLSeq_T2L1* spent some time to generate filtered table for all the levels, its performance is slightly worse than that of *MLSeq_TML1*.

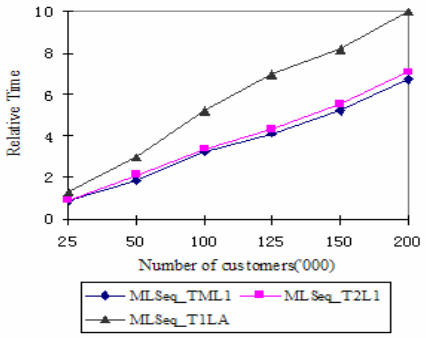

FIGURE 6 PERFORMANCE COMPARISON

Based on the above discussion, some suggestions may be given to the selection of appropriate algorithm for a specific dataset:

- If minimal support for each level is small, *MLSeq_T1LA* may be a good choice. As the number of filtered items is relatively small and the effort for generating the filtered tables is saved, using T[1] for all levels may be more efficient than using the filtered tables.
- *MLSeq_TML1* suits best for the datasets where a large number of items can be filtered at level 1. It is a good idea to check how many items have been filtered out at level 1 and then decide which algorithm should be selected. If the number is relatively large, *MLSeq_TML1* is a good choice. Otherwise, further filtrations may be required for the lower levels, i.e. *MLSeq_T2L1* may be better.
- When using *MLSeq_T2L1*, especially in the case that the value of minimum support for higher levels are larger than that at lower levels, some items that should be retained at lower level may already be filtered out at higher level. Thus, some associate pattern at lower level may be missed.
- If the values of minimum support increase from high level to low level, *MLSeq_T2L1* is a good choice. As a number of items will be filtered out at each level, the effort used for processing the filtered table at each level will be reduced.

V. CONCLUSIONS

Sequential patterns mined from large customer transaction databases can discover implicit and potential useful knowledge. Multiple-level sequential patterns provide this kind of knowledge at different concept levels, from general levels to more specific levels. Therefore multiple-level sequential patterns are more helpful than the single level sequential patterns for the decision-makers in sale management. In this study, we characterized the problem of mining multiple-level sequential patterns by formally defining a set of concepts used in the mining process and developed three algorithms to solve the problem.

An experiment was conducted to assess the performance of the proposed three algorithms. Two customer transaction datasets coded by a pre-defined concept hierarchy were generated and applied to the algorithms. The performances of the algorithms were measured by the relative time spent on the mining tasks based on the two datasets. The experimental results showed that the performance depends on the characteristics of the datasets and the pre-defined threshold of minimal support for each level of the concept hierarchy. Based on the experimental results, some suggestions were given for how to select appropriate algorithm for a certain dataset.

Applying the algorithms to large, real customer transaction databases to further test the algorithms may be required in future.

REFERENCES

[1] Chen, M.S., Han, J. and Yu, P.S., "Data Mining: An Overview from a Database Perspective," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 6, Dec, 1996, pp. 866-883.

[2] Rakesh, A., Tomasz, I. and Arun, S., "Mining Association Rules Between Sets of Items in Large Databases," *ACM SIGMOD*, May 1993, pp. 207-216.

[3] Park, J.S., Chen, M.S., and Yu, P.S., "An Effective Hash Bashed Algorithm for Mining Association Rules," in *Proceedings of ACM*

SIGMOD, May 1995, pp. 175-186.

[4] Rakesh, A. and Ramakrishnan, S., "Fast Algorithm for Mining Association Rules," in *Proceedings of 20th VLDB Conference*, Santiago, Chile, 1994, pp. 487-499.

[5] Chen, N. and Chen, A., "Discovery of Multiple-Level Sequential Patterns from Large Databases," in *Proceedings of the 4th International Symposium On Future Software Technology (ISFST-1999)*, Oct. 27-29, 1999, Nanjing, P. R. China, pp. 169-174.

[6] Rakesh, A. and Ramakrishnan, S., "Mining Sequential Patterns," Research Report, RJ 9910, IBM Almaden Research Center, San Jose, California, October 1994.

[7] Han, J., and Fu, Y., "Discovery of Multiple-Level Association Rules from Large Databases," in *Proceedings of 21st VLDB Conference*, Zurich, Switzerland, 1995, pp. 420-431.

[8] Ramakrishnan, S. and Rakesh, A., "Mining Generalized Association Rules," in *Proceedings of 21st VLDB Conference, Zurich, Switzerland*, 1995, pp. 407-419.

[9] Rakesh, A. and Ramakrishnan, S., "Mining Sequential Patterns," in *Proceedings of the 11th International Conference on Data Engineering*, March 1995, Taipei, Taiwan, IEEE Computer Society, pp. 3-14.

[10] Chen, R.S., Tzeng, G.H., Chen, C.C. and Hu, Y.C., "Discovery of Fuzzy Sequential Patterns for Fuzzy Partitions in Quantitative Attributes," in *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'01)*, June, 2001, Beirut, Lebanon, pp. 144-150.