8th International Conference on Advanced Computational Intelligence
Chiang Mai, Thailand; February 14-16, 2016

Discovering Long Maximal Frequent Pattern

Shu-Jing Lin
National Chung-Shan Institute of Science & Technology
Taoyuan, Taiwan

Yi-Chung Chen and Don-Lin Yang
Department of Information Engineering and Computer Science
Feng Chia University
Taichung, Taiwan

Jungpin Wu
Department of Statistics
Feng Chia University
Taichung, Taiwan

***Abstract*—Association rule mining, the most commonly used method for data mining, has numerous applications. Although many approaches that can find association rules have been developed, most utilize maximum frequent itemsets that are short. Existing methods fail to perform well in applications involving large amounts of data and incur longer itemsets. Apriori-like algorithms have this problem because they generate many candidate itemsets and spend considerable time scanning databases; that is, their processing method is bottom-up and layered. This paper solves this problem via a novel hybrid Multilevel-Search algorithm. The algorithm concurrently uses the bidirectional Pincer-Search and parameter prediction mechanism along with the bottom-up search of the Parameterised method to reduce the number of candidate itemsets and consequently, the number of database scans. Experimental results demonstrate that the proposed algorithm performs well, especially when the length of the maximum frequent itemsets are longer than or equal to eight. The concurrent approach of our multilevel algorithm results in faster execution time and improved efficiency.**

Keywords—data mining; association rule; maximum frequent itemset; long itemset; multilevel search

I. INTRODUCTION

In data mining or knowledge discovery [1, 2], an analytic method is applied automatically or semi-automatically to look for meaningful rules or patterns in large amounts of data. The main goal of data mining is to extract useful information from huge databases in an effective and efficient manner [3]. Data mining is very useful in both data management and decision making. Association rule mining [4], the most frequently used data mining technique, uses the Apriori algorithm to discover the relationships between various points of data in a database. Many mining algorithms are based on the Apriori algorithm as it is simple and straightforward. The Apriori algorithm has two phases, for finding frequent itemsets: candidate generation and verification. A frequent itemset is a set of items that appear together in a number of database records and their occurrence frequency meets a pre-defined threshold. To find all frequent m-itemsets for m starting from 1 to the maximal length of frequent itemsets, the algorithm must produce all 2m of its subsets and scan the database m times. With their exponential complexity, Apriori-like algorithms are restricted to only short frequent itemsets.

To overcome this limitation, this paper proposes the Multilevel-Search algorithm that efficiently extracts the maximal frequent itemsets, where an itemset is maximal frequent when it has no superset that is frequent. Experimental results demonstrate that the proposed approach is most efficient when maximum frequent itemsets are long. This paper's contribution to literature is its novel approach to effectively seeking longer association rules [5] using bottom-up and top-down searches concurrently with a multilevel approach. The remainder of this paper is organized as follows. Section II explores relevant methods related to literature. Section III introduces the concept and pseudo code of the proposed Multilevel-Search algorithm. Section IV presents detailed implementation and explanatory examples. Section V provides comparisons, experimental analysis, and efficiency evaluations, and Section VI contains concluding remarks and future directions for research.

II. RELATED WORK

The number of organizations utilizing mining association rules [6] to discover useful information is increasing.

Definition of association rules: Let I = {i1, i2,..., ik} be a set of k distinct items. A transaction T is a set of items in I and $T \subseteq I$. A transaction T can represent items purchased by a customer from a supermarket. A database D is a set of transactions. An itemset is a set X of items ($X \subseteq I$). The number of items in an itemset is its length. Itemsets of length k are referred to as k-itemsets. A transaction T is said to support an itemset $X \subseteq I$ if and only if $X \subseteq T$. The fraction of transactions in D that support X is the support of X, denoted as support(X). A user can define a minimum support threshold, which is a fraction (or percentage).

An association rule has the form R: $X \Rightarrow Y$; X, $Y \subseteq I$ and $X \cap Y = \varnothing$. The support for rule R is defined as support($X \cup Y$). A confidence factor for such a rule (customarily represented by percentage) is defined as 100% × support($X \cup Y$) / support(X), and is used to evaluate the strength of an association rule. An itemset is frequent when its support meets the minimum support; otherwise, it is infrequent. A frequent itemset is interesting (or strong) when its confidence meets the minimum confidence level.

This paper focuses on finding the maximal frequent itemsets [7-9]. An itemset is a maximal frequent itemset when it is frequent and no proper superset of it is frequent. The maximal frequent set (MFS) is the set of all maximal frequent itemsets. To find maximal frequent itemsets (MFI) efficiently, Lin and Kedem (2002) [10] applied the Pincer-Search algorithm that quickly searched for maximal frequent itemsets in two directions: bottom-up and top-down. Denwattana and

Getta (2001) [11] proposed the Parameterised algorithm based on the Apriori algorithm. Hong et al. (2009) [12], who proposed a mining algorithm based on the Parameterised algorithm, applied the idea of a bottom-up search with multiple levels to reduce the number of candidate itemsets and the number of database scans to accelerate the process of finding maximal frequent itemsets. The method proposed in this paper mainly integrates concepts from both the Parameterised and Pincer-Search algorithms.

III. Our Efficient Algorithm for Discovering Maximal Frequent Itemsets

A. The Concept Informing the Proposed Approach

The proposed Multilevel-Search algorithm is based on bottom-up and top-down searches of maximal frequent itemsets of the Pincer-Search and Parameterised algorithms. This algorithm can go up or down many levels in one pass. First, the Multilevel-Search algorithm constructs a statistics table and sets the required parameters, as does the Parameterised algorithm. For instance, a parameter n indicates how many lattice levels [13] can be traversed at a time. Parameter k is the start level of the mining process. The next step is to predict the candidate from k to $(k+n-1)$-itemsets and scan the database once to verify them. When the prediction is incorrect, the database is scanned again to correct the error. The process continues and works efficiently by using a bidirectional search until the end condition is met. Thus, the Multilevel-Search algorithm can eliminate more redundant itemsets than other methods and also reduces the number of database scans.

B. No Missing Candidate Itemsets

The algorithm starts from the lattice level of the parameter $k = 2$; the value of k is increased by parameter n. Then predicted-frequent $(k+n-1)$-itemsets are produced and used to eliminate lower-level candidate itemsets. Section IV.D describes the process in detail.

Lemma 1. Pruning of redundant candidate itemsets in the Multilevel-Search algorithm is based on the prediction that a lattice level does not result in any candidate missing from the next level.

Proof: When the prediction is correct, the algorithm finds the correct result without the need to recover any candidate itemset; otherwise, the recovery procedure in Section IV.D is used to make the necessary correction. Here is the end of the proof.

C. Multilevel-Search Algorithm

The definition of symbols:
MFCI := Maximal frequent candidate itemsets;
MFI := Maximal frequent itemsets;
C_k := Candidate k-itemsets;
L_k := Frequent k-itemsets;
inf := Infrequent itemsets;
RC := Remaining candidate itemsets;
ST := Statistics table;
n := The number of lattice levels traversed at a time;
minsup := The minimum support;
t_m := The user specified threshold of itemset m-support;
t_l := The user specified threshold of transaction length;;
C_k^+ := Predicted-frequent k-itemsets;
C_k^- := Predicted-infrequent k-itemsets;

```
Algorithm: Multilevel-Search algorithm
Input: A database and user-defined parameters minsup, k, n, tm , tl
Output: MFI contains all maximal frequent itemsets
Results := $\emptyset$; k := 2;
Call the Statistics table procedure to generate a statistics table ( i.e., ST );
Scan ST and count supports for every 1-item to generate $L_1$;
Join $L_1$ to generate $C_2$;
MFCI :={ {i} | $i \in L_1$ } ; MFI := $\emptyset$;
1. while ( $L_{k-1} \notin \emptyset$ and $C_k \notin \emptyset$ ) do
2.  Call the Predict_candidates procedure to generate
3.   predicted-frequent and predicted-infrequent itemsets;
4.  Scan database and count supports for predicted-candidate
    itemsets and MFCI;
5.  /* both in the bottom-up and top-down approach */
6.  Move frequent itemsets from MFCI to MFI;
7.  inf := { infrequent itemsets in candidate itemsets };
8.  Update the MFCI if inf $\notin \emptyset$;
9.  /* in the top-down approach */
10.  if there is an incorrect prediction
11.    Call the Recovery procedure to generate remaining
       candidate itemsets (RC);
12.    Scan database and count supports for RC and MFCI;
13.    /* both in the bottom-up and top-down approaches */
14.    Move frequent itemsets from MFCI to MFI;
15.    inf := { infrequent itemsets in RC };
16.    Update the MFCI if inf $\notin \emptyset$;
17.    /* in the top-down approach */
18.  end-if
19.  Join frequent $(n+k-1)$-itemsets to generate $C_{n+k}$;
20.  /* in the bottom-up approach */
21.  $C_{n+k}$ := { $C_{n+k}$ } $\notin$ {subsets of MFI };
22.  $L_{n+k-1}$ := { frequent$(n+k-1)$-itemsets } $\notin$ {subsets of MFI };
23.  Results := {Results $\cup$ { $L_k$ , $L_{k+1}$ , ….. , $L_{k+n-1}$ }} $\notin$ {subsets of MFI };
24.  k := k+n;
25. end-while;
26. MFI := Results $\cup$ MFI;
27. return MFI;
```

Fig. 1. The pseudo code of the Multilevel-Search algorithm.

Here, MFCI is the set of the candidate-to-be-frequent itemsets with the maximal length; MFI is the set of the frequent itemsets with the maximal length; the m-support of tm is the support of an itemset appearing in the transactions of length m; t_l <= the maximal transaction length based on the available resource.

Fig. 1 shows the pseudo code of the proposed Multilevel-Search algorithm. A detailed description of the algorithm is provided in Section IV, in which various examples are used to explain its implementation.

IV. The Implementation of the Multilevel-Search Algorithm

Implementation of the Multilevel-Search algorithm has the following four tasks.

A. Generate a Statistics Table

A statistics table (ST) is built by scanning the database once (Fig. 2). The ST records the number of times every 1-item appears in database transactions. The ST is then used to find frequent 1-itemsets (i.e., L_1). The default *k* value of the *k*-level is 2. The MFCI is the combination of all items in L_1. If L_{k-1} and C_k are not empty, the Predict_candidates procedure is used. The range of traversed levels to be predicted is from level $_k$ to level $_{k+n-1}$ and a bottom-up approach is used.

```
Algorithm: Statistics table procedure
Input: A database
Output: A statistics table
1. Scan the database to find the support of 1-item
   appearing in transactions of m-length and m ranges
   from 1 to the maximal length of a transaction;
2. For each value of m, find the total number of m-
   length transactions in the database.
```

Fig. 2. The pseudo code of the statistics table procedure.

TABLE I
EXAMPLE DB

TID	Itemset
1	ABCEF
2	ABCEF
3	BCDEF
4	ABCD
5	ABCE
6	ABC
7	ABF
8	ACE
9	BCE
10	BDE

Example 1. To elucidate how the proposed approach evolves from the Parameterised algorithm, Denwattana and Getta's dataset [11] is used (Tables I and II). With the use of this dataset, this paper demonstrates that the proposed is more efficient than their approach. Each record in the example database (DB) contains a transaction's TID and a set of items. A total of six unique items exist in the database with letters A~F. Table II shows the occurrences of each 1-item in transactions of various lengths. The values of m start from the minimal length of a transaction to the maximal length of a transaction (i.e., *m* = 3, 4, and 5). Specifically, among the ten transactions, five are 3-length, two are 4-length, and three are 5-length.

TABLE II
STATISTICS TABLE [11]

Items	*3-length*	*4-length*	*5-length*	*Total sup.*
A	3	2	2	7
B	4	2	3	9
C	3	2	3	8
D	1	1	1	3
E	3	1	3	7
F	1	0	3	4
# of *m*-length transactions	5	2	3	10

B. Predict Candidate Itemsets

The Predict_candidates procedure (Fig. 3,) generates predicted-candidate itemsets according to the statistics table, L_{k-1}, k-level, and the user-defined parameters. The parameters include the user-specified threshold of 1-item in any length (i.e., *tm*), the user-specified threshold of transaction length (i.e., *tl*) and the number of levels to traverse in each pass (i.e., *n*). Here, *tm* is the support of 1-item appearing in transactions of *m*-length and *m* ranges from the minimal length of a transaction to the *tl*. Additionally, *tl* must be less than or equal to the maximal length of a transaction based on the available resource. The Predict_candidates procedure begins its prediction from *k*-level and uses L_{k-1} to generate C_k.

When the candidate *k*-itemsets belong to predicted-frequent itemsets, they are marked C_k^+ . Predicted-infrequent itemsets are marked C_k^- . The support calculation for a 1-item appearing in *m*-length transactions is defined as follows:

$$\text{An item } x\text{'s support value in the } m\text{-length transactions} = \frac{\text{The number of x item appearing in the } m\text{-length transactions}}{\text{The number of the } m\text{-length transactions}} \quad (1)$$

The calculation scope is from *m*-length transactions to *tl*-length transactions. Qualified items that satisfy *tm* are combined, becoming MC_k itemsets. If C_k is a subset of MC_k, C_k belongs to the predicted-frequent itemsets. Otherwise, it belongs to the predicted-infrequent itemsets. L_k is used to produce C_{k+1}. This prediction procedure is repeated until it reaches the (*k+n-1*)-level in a bottom-up search. While the predicted-frequent (*k+n-1*)-itemsets are generated at the (*k+n-1*)-level, they are used to eliminate other candidate itemsets at lower levels.

Example 2. To simplify the discussion, parameters are set as *minsup* = 20%, *n* = 3, *tm*= 80%, and *tl* = 5. Based on the statistics table (Table II), the frequent 1-itemsets are {A}, {B}, {C}, {D}, {E}, and {F}. These 1-itemsets are combined to generate C2 ={{A, B}, {A, C}, {A, D}, {A, E}, {A, F}, {B, C}, {B, D}, {B, E}, {B, F}, {C, D}, {C, E}, {C, F}, {D, E}, {D, F}, {E, F}}. Thus, MFCI is L_1={A, B, C, D, E, F}. If

```
Algorithm: Predict_candidates procedure
Input: A statistics table (ST), L_{k-1}, k-level and the user
defined parameters ($t_m$, $t_l$, $n$)
Output: Predicted-frequent and predicted-infrequent
itemsets from $k$ to $k+n-1$ level
1.  Scan statistics table;
2.  $C_k$ is generated from the $L_{k-1}$;
3.  if $C_k \in$ (subsets of $MC_k$) move $C_k$ to $C_k^+$;
4.  else move $C_k$ to $C_k^-$;
5.  for $i$ from $k+1$ to $n+k-1$
6.      $C_{k+1}$ generated from the $C_k^+$;
7.      if $C_{k+1} \in$ (subsets of $MC_{k+1}$) move $C_{k+1}$ to $C_{k+1}^+$;
8.      else move $C_{k+1}$ to $C_{k+1}^-$;
9.  end-for;
10. if predicted itemsets $\in$ (subsets of $C_{k+n-1}^+$) then delete;
```

Fig. 3. The pseudo code of the Predict_candidates procedure.

some i infrequent 1-itemsets exist, the cardinality of the MFCI would be reduced by i. In this case, the top-down search goes down i levels in one pass, as with the Pincer-Search algorithm. Initially, the MFI is an empty set. Next, the prediction procedure finds MC_2 for C_2 to obtain C_2^+ and C_2^-. Because no 2-length transaction exists in the example, the process starts from 3-length transactions. Only one 1-item {B} satisfies t_m= 80% because {B} item's support value in 3-length transactions is $\frac{4}{5} = 80\% \geq 80\%$.

Three 1-itemsets {A}, {B}, and {C} satisfy the threshold in 4-length transactions (Table III). The user-specified value of t_l is 5, so this value is used to consider transactions up to 5-length. In the last column of PC_2, all 1-itemsets under different m-length transactions are merged as MC_2 = {A, B, C, E, F}. As {A, D} is not a subset of MC_2, it is predicted as infrequent and belongs to C_2^-. Consequently, C_2^+ = {{A, B}, {A, C}, {A, E}, {A, F}, {B, C}, {B, E}, {B, F}, {C, E}, {C, F}, {E, F}} and C_2^- = {{A, D}, {B, D}, {C, D}, {D, E}, {D, F}}. Then the itemsets in C_2^+ are joined to generate C_3 = {{A, B, C}, {A, B, E}, {A, B, F}, {A, C, E}, {A, C, F}, {A, E, F}, {B, C, E}, {B, C, F}, {B, E, F}, {C, E, F}}. These procedures are repeated to obtain C_3^+ = {{A, B, C}, {A, B, E}, {A, B, F}, {A, C, E}, {A, C, F}, {A, E, F}, {B, C, E}, {B, C, F}, {B, E, F}, {C, E, F}} and C_3^- is an empty set; and C_4^+ = {{A, B, C, E}, {A, B, C, F}, {A, B, E, F}, {A, C, E, F}, {B, C, E, F}}, and C_4^- is an empty set.

TABLE III
THE ITEMSETS SATISFYING THE 1-ITEM THRESHOLD (T_M)

m-length	3-length trans.	4-length trans.	5-length trans.	Merged candidate itemsets
PC_2	B	A,B,C	B,C,E,F	MC_2={A,B,C,E,F}
PC_3	B	A,B,C	B,C,E,F	MC_3={A,B,C,E,F}
PC_4		A,B,C	B,C,E,F	MC_4={A,B,C,E,F}
PC_5			B,C,E,F	MC_5={B,C,E,F}

As k = 2, n = 3, and k+n-1 = 4 in this example, the prediction process has completed a pass of C_2^+, C_2^-, C_3^+, C_3^-, C_4^+ and C_4^-. Since this work wants to find maximal frequent itemsets, their sub-itemsets can be pruned from lower-level candidates using the predicted-frequent 4-itemsets and predicted-infrequent 4-itemsets in the top-down approach.

After eliminating itemsets, the predicted candidate itemsets, MFCI, and MFI are generated as follows: C_2^+ ={}; C_2^- ={A, D}, {B, D}, {C, D}, {D, E}, {D, F}; C_3^+ ={}; C_3^- ={} C_4^+ ={A, B, C, E}, {A, B, C, F}, {A, B, E, F}, {A, C, E, F}, {B, C, E, F}; C_4^- ={} and MFCI={A, B, C, D, E, F}; MFI={}

C. Update the Maximal Frequent Candidate Itemsets

The database is scanned again to find the actual support of predicted-frequent, predicted-infrequent itemsets, and MFCI. If the frequent-candidate itemsets in MFCI are verified after counting their frequencies, they are moved to the MFI. The MFCI is updated according to the infrequent itemsets *inf*. Then, an examination is required to check whether any incorrect prediction occurs.

Example 3. After scanning the database to count support, the frequent k-itemsets and infrequent itemset *inf* are found:

L_2={B, D}, {C, D}, {D, E}; *inf*={A, D}, {D, F}; L_3={}

L_4={A, B, C, E}, {A, B, C, F}, {A, B, E, F}, {A, C, E, F}, {B, C, E, F}

Then, it utilizes the infrequent itemset *inf* to update the MFCI using the top-down approach. Considering {A, D}, the MFCI becomes {{B, C, D, E, F}, {A, B, C, E, F}}. Finally, MFCI = {{A, B, C, E, F}, {B, C, D, E}} after processing {D, E} to generate {B, C, D, E} and {B, C, E, F}. However, {B, C, E, F} is removed from the MFCI because it is a subset of {A, B, C, E, F}. Both the MFCI and MFI are updated as follows:

MFCI={ {A, B, C, E, F}, {B, C, D, E} } and MFI={}

D. Recover Candidate Itemsets

There are two prediction error types: (1) a predicted-frequent itemset becomes an infrequent itemset after counting its support; and (2) a predicted-infrequent itemset becomes a frequent itemset after counting its support. To rectify these wrongly predicted itemsets, additional candidate itemsets must be generated, which are called Remaining Candidate k-

itemsets (*RC*). Fig. 4, shows the recovery procedure. If any predicted-frequent *j*-itemset (*i.e.*, C_j^+) turns out to be an infrequent itemset after calculation, all of its subsets from the *k*-level to (*j-1*)-level will be produced. Similarly, if any predicted-infrequent *j*-itemset (*i.e.*, C_j^-) becomes a frequent itemset after calculation, all of its supersets from (*j+1*)-level to (*k+n-1*)-level will be produced. After the *RC* from *k*-level to (*k+n-1*)-level is produced, the database is scanned again to count the support of the *RC* and the MFCI for correcting prediction errors.

When a frequent-candidate itemset in the MFCI is qualified after counting its support, it will be moved to the MFI. The MFCI will be updated according to the *inf* found in the *RC*. As in Section II, all supersets in an infrequent itemset must be infrequent and all subsets of a frequent itemset must be frequent. These two properties are used to produce a new MFCI. By eliminating the found frequent (*n+k-1*)-itemsets using the MFI, any subset belonging to the MFI is not listed in the frequent itemsets of $L_{(n+k-1)}$. Then it joins $L_{(n+k-1)}$ itemsets to produce the candidate itemsets of C_{n+k}. If any itemset of $L_{(n+k-1)}$ in the above elimination procedure is deleted, candidate itemsets are then recovered to produce complete candidate (*n+k*)-itemsets of C_{n+k}. Then, the approach uses the MFI to prune C_{n+k}. The value *k* of the *k*-level is increased to *n+k*. It repeats the above steps until $L_{(k-1)}$ and C_k are empty. The final result of the MFI is then returned.

```
Algorithm: Recovery procedure
Input: Incorrect predicted-frequent and predicted-infrequent
itemsets, k-level and the user defined parameters ( n )
   Output: Remaining candidate itemsets (RC)
1. Take apart incorrect predicted-frequent itemsets to
   generate remaining candidate itemsets (RC)
2. until k-level;
3. Join incorrect predicted-infrequent itemsets to generate
   remaining candidate itemsets (RC)
4. until (k+n-1)-level;
```

Fig. 4. The pseudo code of the Recovery procedure

Example 4. From Example 3, some prediction errors are likely because three itemsets {B, D}, {C, D}, and {D, E} in C_2^- become frequent after support verification. For correction, these itemsets are combined to produce *RC* itemsets as follows:

RC_2 ={}; RC_3 ={B, C, D}, {B, D, E}, {C, D, E}

RC_4 ={B, C, D, E}; MFCI={A, B, C, E, F}, {B, C, D, E}

The database is scanned again to count the support of the *RC* itemsets and the MFCI is updated. Thus, MFI = {A, B, C, E, F} after support is counted. The infrequent itemsets {C, D, E} and {B, C, D, E} are used to update the MFCI. The MFCI then becomes an empty set. The frequent 4-itemsets are combined to yield C_5 ={A, B, C, E, F}. The MFI is used to eliminate the frequent itemsets (which are not maximal) and candidate 5-itemsets via the top-down approach, yielding the following results:

L_2={}; L_3={B, C, D}, {B, D, E}; L_4={};

C_5={}; MFCI={}; MFI = {A, B, C, E, F};

Final result = {{B, C, D}, {B, D, E}, {A, B, C, E, F}};

Because the frequent 4-itemsets and candidate 5-itemsets are empty, the entire procedure ends here.

V. Experimental Results and Discussion

A. Experimental Environment

Experiments were performed on an Intel® Xeon™ MP CPU at 2.00GHz with 3800 MB RAM running Windows 2000. The programs were developed using Microsoft Visual Basic 6.0. The Pincer-Search and Parameterised algorithms were also implemented for comparison. The IBM dataset generator was used to produce test databases. Table IV defines parameters. Table V lists the parameters used to generate the databases for the experiments. We generated twenty databases in total and each type of database in Table V had five databases.

TABLE IV
Definitions of Parameters

D	The number of transactions
T	The average length of transactions
L	The number of maximal potentially-frequent
N	The number of distinct items

TABLE V
Parameters of Databases

Type of Database	*T*	*L*	*N*	*D*
T2.L10.N500.D100K	2	10	500	100000
T8.L10.N500.D100K	8	10	500	100000
T10.L10.N500.D100K	10	10	500	100000
T12.L10.N500.D100K	12	10	500	100000

The types of maximal frequent itemsets in these databases are mainly short, moderate, and long in length. The short maximal frequent itemsets range at 1~4; the moderate maximal frequent itemsets range at 5~8; and the long maximal frequent itemsets range at 9~12. All are used in the Multilevel-Search, Pincer-Search, and Parameterised algorithms to examine their respective performance.

B. Data Analysis and Efficiency Assessment

Three parameters, *n*, t_m, and t_l in the Multilevel-Search algorithm affect efficiency and execution time. For instance, when parameter t_m is set too large, many infrequent itemsets will be found. Conversely, when parameter t_m is too small, many frequent itemsets will be found. To demonstrate its simplicity and ease-of-use, multiple experiments are performed with various parameters settings for the proposed approach without complex analysis of datasets. Suitable settings for the three parameters, *n*, *tm*, and *tl*, are adopted to

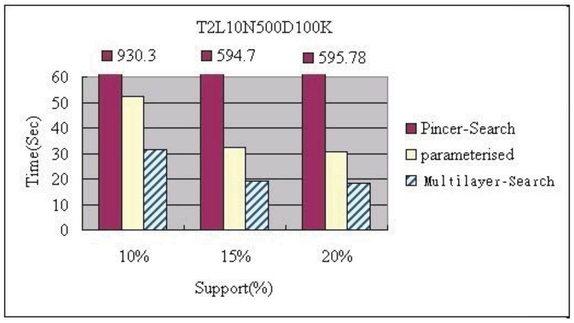

Fig. 5. Comparison of the average execution time for three algorithms with T2.L10.N500.D100K.

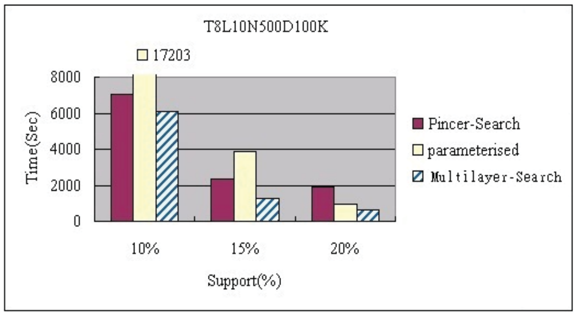

Fig. 6. Comparison of the average execution time for three algorithms with T8.L10.N500.D100K.

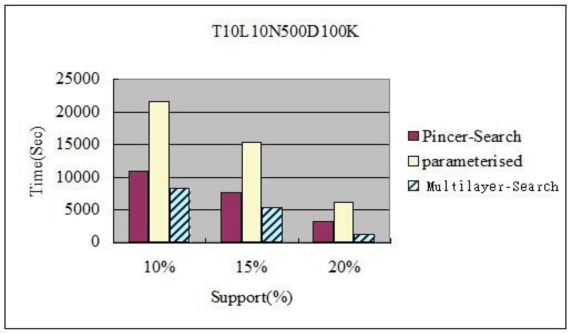

Fig. 7. Comparison of the average execution time for three algorithms with T10.L10.N500.D100K.

perform an efficiency assessment of target algorithms. Consequently, these parameters are set as follows.

1. If the average transaction length is 2, *n* is set to 2 and t_m is 65%.
2. If the average transaction length is 8~10, *n* = 4 and t_m = 80%.
3. The user-specified threshold of transaction length t_l depends on the database used (Table V).

In the first set of experiments, T2.L10.N500.D100K databases are used to compare the efficiency of the Multilevel-Search, Pincer-Search, and Parameterised algorithms.

The average transaction length is 2. Fig. 5 shows that the Multilevel-Search algorithm performs better than the other two algorithms for short maximal frequent itemsets. Here, multilevel and multilayer are used interchangeably.

TABLE VI
THE AVERAGE TIMES OF DATABASE SCAN (*MINSUP*=20%)

Databases	**Apriori**	**Pincer-search**	**Parameterised**	**Multilevel-Search**
T2.L10.N500.D100	4	2	3	2
T8.L10.N500.D100	8	5	4	3
T10.L10.N500.D100	11	6	7	4
T12.L10.N500.D100	13	7	7	4

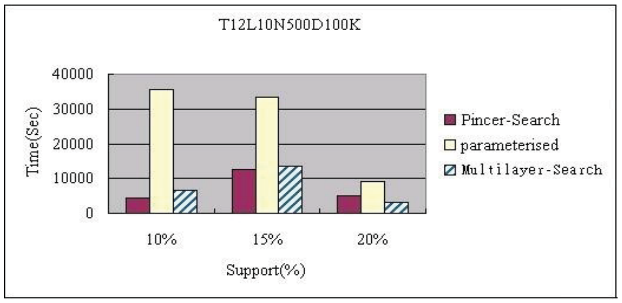

Fig. 8. Comparison of the average execution time for three algorithms with T12.L10.N500.D100K.

The average length of transactions in T8.L10.N500.D100K datasets is 8 (Fig. 6). With the minimal support of 10%, the difference in execution time is not significant between Multilevel-search and Pincer-search algorithms, while the Parameterised algorithm performs poorly. When minimal support increases, fewer candidates of frequent itemsets are found, and the performance of the Multilevel-search and Parameterised algorithms increases. When minimal support is doubled to 20%, execution time of the Parameterised algorithm is less than that of the Pincer-Search algorithm while the Multilevel-search algorithm remains best.

The average length of transactions in T10.L10.N500. D100K dataset is 10 (Fig. 7). When minimum support is 20%, the Multilevel-Search algorithm is over two times faster than the other two algorithms. Due to the efficiency of going up or down multiple levels in one pass, the Multilevel-search algorithm is the overall winner.

Using the databases T12.L10.N500.D100K (Fig. 8), experiment results show that the efficiency of the Multilevel-Search and Pincer-Search algorithms are similar, as both use the top-down approach to find the maximal frequent itemsets and improve their efficiency.

Since the goal in this paper is to find efficient solutions for long patterns, the times taken by database scans and the number of candidate itemsets for databases when T is 8, 10, and 12 are examined in detail. Experiment results (Tables VI and VII) show that the Multilevel-Search algorithm has the best performance for total times the database is scanned and the total number of candidate itemsets. Although the Pincer-search generates less candidate itemsets in two instances (Table VII), the Multilevel-Search algorithm is still the overall

TABLE VII

THE AVERAGE NUMBER OF CANDIDATE ITEMSETS (*MINSUP* =20%)

The average length of transactions	**Pincer-search**	**Parame-terised**	**Multilevel-Search**
8	258	338	280
10	913	1356	782
12	1117	1778	1421

TABLE VIII

COMPARISON OF AVERAGE EXECUTION TIME (SECONDS)

Algorithm (*minsup*=20%)	**The average length of**			**Average time**
	8	***10***	***12***	
Pincer-Search	1885.48	3155.97	5155.97	3399.14
Parameterised	924.30	6157.23	9157.23	5412.92
Multilevel-Search	642.29	1290.64	3290.64	1741.19

TABLE IX

THE RATIO OF IMPROVEMENT IN TERMS OF EXECUTION TIME

Algorithm (*minsup*=20%)	**The average length of transactions**			**Average ratio**
	8	***10***	***12***	
Pincer-Search/ Multilevel-Search	2.94	2.45	1.57	2.32
Parameterised/ Multilevel-Search	1.44	4.77	2.78	2.99

winner because the extra candidate itemsets come from the recovery procedure and no data scan is involved.

To assess the performance of these algorithms, this paper compares the average execution times for three kinds of databases with the minimal support of 20% and the experiments are performed fifteen times for each algorithm. Table VIII shows the average execution time with the transaction lengths of T = 8, 10, and 12. To elucidate the data in Table VIII, Table IX shows clearly that the Multilevel-Search algorithm is 2.32 times faster than the Pincer-Search algorithm and 2.99 times faster than the Parameterised algorithm. Separate data for the ratio of improvement in terms of transaction lengths of T = 8, 10, and 12 are also presented.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposes a novel Multilevel-Search algorithm to mine efficiently long association rules. Using the concept of predicting frequent and infrequent itemsets by parameter settings, this Multilevel-Search algorithm combines the top-down search mechanism from the Pincer-Search algorithm and bottom-up multilevel searching from the Parameterised algorithm to rapidly identify maximal frequent itemsets. After finding the maximal frequent itemsets, the algorithm can also generate all frequent itemsets. This method is especially useful when an itemset is long. Experimental results verify that the proposed algorithm is faster than the Parameterised and Pincer-Search algorithms.

However, the Multilevel-Search algorithm has room for improvement in its prediction mechanism. The effects of settings for parameters n, t_m, t_l can be determined with further study. Experiments showed that prediction accuracy declines when too many levels are scanned in one pass. This also produces a large number of candidate itemsets. Better prediction accuracy will reduce the number of redundant candidate itemsets and the number of database scans. In the future, it is planned to use a statistical approach to improve the prediction mechanism and apply parallel computing [14] for better performance.

ACKNOWLEDGMENT

This work was supported in part by the Ministry of Science and Technology, Taiwan, under grants MOST 104-2221-E-035-089 and MOST 104-2218-E-035-012.

REFERENCES

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

[2] J. Han and M. Kamber, "Data Mining: Concepts and Techniques," Burlington: Morgan Kaufmann, 2011.

[3] J. Dong and M. Han, "BitTableFI: An Efficient Mining Frequent Itemsets Algorithm," *Knowledge-Based Systems*, vol. 20, no. 4, pp. 329-335, 2007.

[4] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Association Rules in Large Databases," in *Proc. of the 20th International Conference on Very Large Data Bases*, Santiago, 1994, pp. 487-499.

[5] Y. Tsay and J. Chiang, "CBAR: An Efficient Method for Mining Association Rules," *Knowledge-Based Systems*, vol. 18, no. 2-3, pp. 99-105, 2005.

[6] Irina Tudor, Association rule mining as a data mining technique, BULETINUL universitatii Petrol-Gaze din. Ploiesti, vol. LX, no. 1, page 49-56, 2008.

[7] P. Dong and B. Chen, "New Algorithm of Maximum Frequent Itemsets for Mining Multiple-Level Association Rules," in *Proc. of the 3rd International Conference on Innovative Computing Information and Control (ICICIC'08 Proceedings)*, 2008, pp. 332-335.

[8] D. Burdick, M. Calimlim, and J. Gehrke, "Mafia: A maximal frequent itemset algorithm for transactional databases," in Proc. of the 17th International Conference on Data Engineering (ICDE 2001), Heidelberg, Germany, 2001, pp. 443-452.

[9] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: A Performance Study of Mining Maximal Frequent Itemsets," in *Proc. of the 2003 Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, Melbourne, Florida, November 2003.

[10] D. Lin and Z.M. Kedem, "Pincer-search: An Efficient Algorithm for Discovering the Maximum Frequent Set," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 553-566, 2002.

[11] N. Denwattana and J.R. Getta, "A Parameterised Algorithm for Mining Association Rules," in *Proc. of the 12th Australasian Database Conference*, 2001, pp. 45-51.

[12] T. P. Hong, C. Y. Horng, C. H. Wu, and S. L. Wang, "An improved data mining approach using predictive itemsets," *Expert Systems with Applications*, vol. 36, no. 1, pp. 72-80, 2009.

[13] Z. Abdullah, T. Herawan, and M. M. Deris, "Mining Highly Correlated Least Association Rules Using Scalable Trie-Based Algorithm," *Journal of the Chinese Institute of Engineers*, vol. 35, no. 3, pp. 547-554, 2012.

[14] C. Yeh, "Big Data Mining with Parallel Computing: A Comparison of Distributed and MapReduce Methodologies," *Master thesis*, Dept. of Information Management, National Central University, Taiwan, 2015