SETM*-Lmax: An Efficient Set-Based Approach to Find Maximal Large Itemsets

Ye-In Chang, and Yu-Ming Hsieh

Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
Republic of China
{E-mail: changyi@cse.nsysu.edu.tw}
{Tel: 886-7-5252000 (ext. 4334)}
{Fax: 886-7-5254301}

Abstract

Discovery of *association rules* is an important problem in the area of data mining. An association rule means that the presence of some items in a transaction will imply the presence of other items in the same transaction. For this problem, how to efficiently *count large itemsets* is the major work, where a large itemset is a set of items appearing in a sufficient number of transactions. In this paper, we propose an efficient SETM*-Lmax algorithm to find maximal large itemsets, based on a high-level set-based approach. The advantage of the set-based approach, like the SETM algorithm, is simple and stable over the range of parameter values. In the SETM*-Lmax algorithm, we use a forward approach to find all maximal large itemsets from L_k, and the w-itemset is not included in the w-subsets of the j-itemset, where $1 \leq k \leq MaxK$, $1 \leq w < j \leq MaxK$, $L_{MaxK} \neq \emptyset$ and $L_{MaxK+1} = \emptyset$. We conduct several experiments using different synthetic relational databases. The simulation results show that the proposed forward approach (SETM*-Lmax) to find all maximal large itemsets requires shorter time than the backward approach proposed by Agrawal.

(**Key Words:** association rules, data mining, knowledge discovery, relational databases, transactions)

1 Introduction

One of the important data mining tasks, mining *association rules* in transactional or relational databases, has recently attracted a lot of attention in database communities [1, 2, 5, 6, 8, 11, 12, 13, 16, 17, 19, 21]. The task is to discover the important associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction [7]. For example, one may find, from a large set of transaction data, such an association rules as if a customer buys (*one brand of*) milk, he/she usually buys (*another brand of*) bread in the same transaction. Since mining association rules may require to repeatedly scan through a large transaction database to find different association patterns, the amount of processing could be huge, and performance improvement is an essential concern [7].

Previous approaches to mining association rules can be classified into two approaches: low-level and high-level approaches, where a low-level approach means to retrieve one tuple from the relational database at a time, and a high-level approach means a set-based approach. For example, Apriori/AprioriTID [2], DHP [17] and Boolean algorithms [23] are based on the low-level approach, while the SETM algorithm [14] is based on the high-level approach. A set-based approach (i.e., a high-level approach) allows a clear expression of what needs to be done as opposed to specifying exactly how the operations are carried out in a low-level approach. The declarative nature of this approach allows consideration of a variety of ways to optimize the required operations. This means that the ample experience that has been gained in optimizing relational queries can directly be applied here. Eventually, it should be possible to integrate rule discovery completely with the database system. This would facilitate the use of the large amounts of data that are currently stored on relational databases. The relational query optimizer can then determine the most efficient way to obtain the desired results. Finally, the set-based approach has a small number of well-defined, simple concepts and operations. This allows easy extensibility to handling additional kinds of mining, e.g. relating association rules to customer classes.

In [14], based on a high-level approach, Houtsma and Swami proposed the SETM algorithm that uses SQL for generating the frequent itemsets. Algorithm SETM is simple and stable over the range of parameter values. Moreover, it is easily parallelized. But the

disadvantage of the SETM algorithm is similar to that of the AIS algorithm [1]. That is, it generates too many invalid candidate itemsets. In this paper, we design an efficient algorithm for mining association rules based on a high-level set-oriented approach. We propose the SETM*-Lmax algorithm to find maximal large itemsets. We conduct several experiments using different synthetic relational databases. The simulation results show that the proposed forward approach (SETM*-Lmax) to find all maximal large itemsets requires shorter time than the backward approach proposed by Agrawal.

The rest of the paper is organized as follows. In Section 2, we describe the background. In Section 3, we give a brief survey. In Section 4, we present the proposed SETM*-Lmax algorithm. In Section 5, we study the performance of proposed algorithm. Finally, Section 6 gives the conclusion.

2 Background

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of m distinct items [15]. A *transaction* T is defined as any subset of items in I. A database D is a set of transactions. A set of items is called an *itemset*. The number of items in an itemset is called the *length* of an itemset. Itemsets of some length k are referred to as k-itemsets.

A transaction T is said to *support* an itemset $X \subseteq I$ if it contains all items of X , i.e., $X \subseteq T$. The fraction of the transactions in D that support X is called the *support* of X, denoted as support(X). An itemset is *large* if its support is above some user-defined minimum support threshold [2].

An *association rule* has the from $R : X \Rightarrow Y$, where X and Y are two non-empty and non-intersecting itemsets [15]. The *support for rule* R is defined as support($X \cup Y$). The rule $X \Rightarrow Y$ has support s in the transaction set D if $s\%$ of transactions in D contain X. A *confidence* factor defined as support($X \cup Y$)/support(X), is used to evaluate the strength of such association rules. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if $c\%$ of transactions in D that contain X also contain Y. Consequently, the semantics of the confidence of a rule indicates how often it can be expected to apply, while its support indicates how trustworthy this rule is. The problem of the association rule mining is to discover all rules that have support and confidence greater than some user-defined minimum

support and minimum confidence thresholds, respectively.

In [1, 2, 17, 18, 20], the problem of mining association rules is decomposed into the following two steps:

1. Discover the *large itemsets*, i.e., the sets of itemsets that have transaction support above a predetermined minimum support s.

2. Use the large itemsets to generate the association rules for the database. The general idea is that if, say, $ABCD$ and AB are large itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by computing the ration $\tau = \text{support}(ABCD)/\text{support}(AB)$. The rule holds only if $\tau \geq$ minimum confidence. Note that the rule will have minimum support because $ABCD$ is a large itemset.

It is noted that the overall performance of mining association rules is determined by the first step. After the large itemsets are identified, the corresponding association rules can be derived in a straightforward manner. Efficient counting of large itemsets is thus the focus of most priorwork.

3 The Apriori Algorithm

Agrawal and Srikant proposed an algorithm, called Apriori [2], for generating large itemsets. Apriori constructs a candidate set of large itemsets, counts the number of occurrence of each candidate itemset, and then determines large itemsets based on a pre-determined minimum support. In the Apriori algorithm, the candidate *k-itemsets* is generated by a cross product of the large $(k-1)$-*itemsets* with itself. Then, the database is scanned for computing the count of the candidate *k-itemsets*. The large *k-itemsets* consist of only the candidate *k-itemsets* with sufficient support. This process is repeated until no new candidate itemsets is generated. It is noted that in the Apriori algorithm, each iteration requires a pass of scanning the database, which incurs a severe performance penalty [23].

Figure 1 shows the Apriori algorithm, and Table 1 summarizes the variables used in the algorithm [2]. Consider an example transaction database given in Figure 2. Figure 3 shows the process. In the first iteration, Apriori simply scans all the transactions to count the number of occurrences for each item, and generates candidate 1-itemsets, C_1. Assuming

k-itemset	An itemset has k items.
L_k	Set of large k-itemsets (those with minimum support). Each member of this set has two fields: (i) itemset and (ii) support count.
C_k	Set of candidate k-itemsets (potentially large itemsets). Each member of this set has two fields: (i) itemset and (ii) support count.

Table 1: Variables used in the Apriori algorithm

Procedure Apriori;
begin
  L_1 := large 1-itemsets;
  $k := 1$;
  repeat
    $k := k + 1$;
    C_k := apriori-gen(L_{k-1}); (* New candidates *)
    forall transactions $t \in D$ **do**
    begin
      C_t :=subset(C_k, t); (* Candidates contained in t *)
      forall candidates $c \in C_t$ **do**
        $c.count := c.count + 1$;
    end;
    $L_k := \{c \in C_k | c.count \geq$ minimum support$\}$;
  until $L_k = \emptyset$;
  Answer := $\bigcup_k \ L_k$;
end;

Figure 1: The Apriori algorithm

that the minimum transaction support required is two (i.e., $s = 50\%$), the set of large 1-itemsets, L_1, composed of candidate 1-itemsets with the minimum support required, can then be determined. To discover the set of large 2-itemsets, in view of the fact that any subset of a large itemset must also have minimum support, Apriori uses $L_1 * L_1$ to generate a candidate set of itemsets C_2 where * is an operation for concatenation in this case. Next, the four transactions in D are scanned and the support of each candidate itemset in C_2 is counted. The set of large 2-itemsets, L_2, is therefore determined based on the support of each candidate 2-itemset in C_2.

The set of candidate itemsets, C_3, is generated from L_2 as follows [7]. From L_2, two large

Database D

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Figure 2: A transaction database (Example 1)

2-itemsets with the same first item, such as $\{BC\}$ and $\{BE\}$, are identified first. Then, Apriori tests whether the 2-itemset $\{CE\}$, which consists of their second items, constitutes a large 2-itemset or not. Since $\{CE\}$ is a large itemset by itself, we know that all the subsets of $\{BCE\}$ are large and then $\{BCE\}$ becomes a candidate 3-itemset. There is no other candidate 3-itemset from L_2. Apriori then scans all the transactions and discovers the large 3-itemsets L_3. Since there is no candidate 4-itemset to be constituted from L_3, Apriori ends the process of discovering large itemsets.

In the Apriori algorithm as shown in Figure 1, the *apriori-gen* function takes an argument L_{k-1}, and returns a superset of the set of all large $k-$itemsets. Before exiting the *apriori-gen* function, a *prune* step is executed, which deletes all itemsets $c \in C_k$ such that some $(k-1)$-subset of c is not in L_{k-1}.

4 The SETM*-Lmax Algorithm

In this Section, we present the *SETM*-Lmax* algorithm to find all maximal large itemsets (denoted as $Lmax$) from L_k, and the w-itemset is not included in the w-subsets of the j-itemset, where $1 \leq k \leq MaxK$, $1 \leq w < j \leq MaxK$, $L_{MaxK} \neq \emptyset$ and $L_{MaxK+1} = \emptyset$.

4.1 An Example

For the sample input as shown in Figures 2 and 4, Figures 5 and 6 show the results (L_k), respectively. For the resulting large itemsets shown in Figures 5 and 6, Figures 7 and 8 show the corresponding all maximal large itemsets ($Lmax$), respectively. For example, in Figure 5, since BC $\in$ {BCE} (=L_3), where BC $\in L_2$, BC is removed in the result.

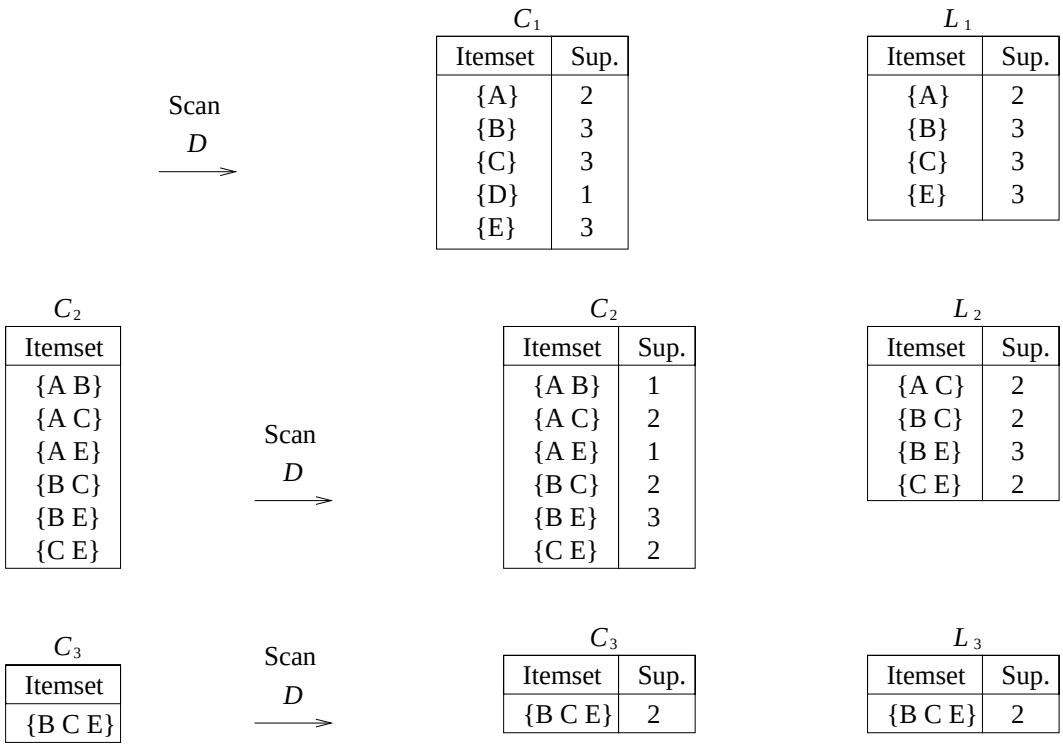

Figure 3: Generation of candidate itemsets and large itemsets

TID	Items
1	A B C D E I
2	A B C E F
3	A C D F
4	A B C D E
5	B C D H
6	D E F
7	A C D G
8	A B C D E H
9	B C E G I
10	E F G H
11	A G H
12	B F H

Figure 4: A transaction database (Example 2)

L_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{E}	3

L_2

Itemset	Sup.
{A C}	2
{B C}	2
{B E}	3
{C E}	2

L_3

Itemset	Sup.
{B C E}	2

Figure 5: Example 1: Large Itemsets ($s = 50\%$)

L_1

Itemset	Sup.
A	7
B	7
C	8
D	7
E	7
F	5
G	4
H	5

L_2

Itemset	Sup.
AB	4
AC	6
AD	5
AE	4
BC	6
BD	4
BE	5
BH	3
CD	6
CE	5
DE	4
EF	3

L_3

Itemset	Sup.
ABC	4
ABD	3
ABE	4
ACD	5
ACE	4
ADE	3
BCD	4
BCE	5
BDE	3
CDE	3

L_4

Itemset	Sup.
ABCD	3
ABCE	4
ABDE	3
ACDE	3
BCDE	3

L_5

Itemset	Sup.
ABCDE	3

Figure 6: Example 2: Large Itemsets ($s = 25\%$)

L_2

Itemset	Sup.
{A C}	2

L_3

Itemset	Sup.
{B C E}	2

Figure 7: Maximal Large Itemsets for Example 1

L_2

Itemset	Sup.
BH	3
EF	3

L_5

Itemset	Sup.
ABCDE	3

Figure 8: Maximal Large Itemsets for Example 2

$R_k^{'}$	A database of candidate k-itemsets (i.e., a candidate DB)
L_k	Large k-itemsets
R_k	A database of large k-itemsets (i.e., a filtered DB)
$MaxK$	The maximal length of the itemset
i	A loop index ($1 \leq i \leq MaxK - 1$)
j	A loop index ($1 \leq j \leq i + 1$)

Table 2: Variables used in the *SETM*-Lmax* algorithm

4.2 The Algorithm

Table 2 shows the variables used in the *SETM*-Lmax* algorithm. The complete algorithms are shown in Figures 9, 10, 11, 12, and 13. In procedure *SETM*-Lmax* as shown in Figure 9, the first step is to generate all large itemsets L_k based on the *SETM** algorithm, $1 \leq k \leq MaxK$, and the second step is to delete some element w ($\in L_k$) from L_k if $w \in k$-subset of L_{k+1} except $k = MaxK$. Note that the SETM algorithm constructs $R_k^{'}$ based on R_{k-1} and the original database $SALES$. Due to this reason, the SETM algorithm generates and counts too many candidates itemsets. To reduce the size of the candidate database $R_k^{'}$, we have a new strategy to construct $R_k^{'}$ in procedure *gen-CDB* (as shown in Figure 12). In Agrawal's algorithm [3] for finding sequential patterns, they have proposed a *backward* approach to process step 2 as shown in Figures 14 and 15, where procedure *comb(j, i)* (shown in Figure 15) is a function to compute C_i^j and stores all the possible combinations in a two dimensional array $COMBD$. (Note that in Figure 15, $COMBD[k][1] \ldots COMBD[k][i]$ denote the k'th combination pattern with a size = i.) For the example of *comb(6,4)*, i.e., C_4^6, the contents of $COMBD$ computed from function *comb(6,4)* is shown in Figure 16, in additional to return $C_4^6 = 15$. Take Figure 5 as an example, the backward approach will delete BC, BE and CE from L_2 in the first iteration by checking L_3, and delete A, B, C and E from L_1 at the second iteration by checking L_2 and L_3. While our step 2 is a forward approach. Take Figure 5 as example, our forward approach will delete A, B, C and E from L_1 in the first iteration by checking L_2, and delete BC, BE and CE from L_2 in the second iteration by checking L_3. Figures 17 and 18 show a simplified interpretation of the backward and the forward approaches, respectively.

```
procedure SETM*-Lmax;
begin
    (* Step 1: Finding all large itemsets *)
    (* the sort operation is optional *)
    k := 1;
    L_k := gen-Litemset(Sales, minsup);
    R_k := filter-DB(Sales, L_k);
    repeat
       k := k + 1;
       R'_k := gen-CDB(R_{k-1}, R_{k-1});
       L_k := gen-Litemset(R'_k, minsup);
       R_k := filter-DB(R'_k, L_k);
    until R_k = ∅;
    MaxK := k − 1;

    (* Step 2: Deleting items forwards *)
    for i := 1 to MaxK − 1 do
       for j:=1 to (i + 1) do
          del-Litemset(L_i, L_{i+1}, i, j);
    Answer := ⋃ L_k;
end;
```

Figure 9: The *SETM*-Lmax* procedure

In our forward approach, for the example shown in Figure 5 and 6, Tables 3 and 4 show the changes of the values of the variables for $MaxK = 3$ and $MaxK = 5$, respectively. For example, take Figure 5 as an example, when $i = 2$, we will first remove BC from L_2 by following the *else* part in procedure *del-Litemset*, since B ($= L_2.item_1 = L_3.item_1$) and C ($= L_2.item_2 = L_3.item_2$) appear at the first and second positions in L_3, where $j = 1$ and $(i + j - 1) = 2$. Next, we will remove CE from L_2 by following the *else* part in procedure *del-Litemset*, since C ($= L_2.item_1 = L_3.item_2$) and E ($= L_2.item_2 = L_3.item_3$) appear at the second and third positions in L_3, where $j = 2$ and $(i + j - 1) = 3$. Finally, we will remove BE from L_2 by following the *then* part in procedure *del-Litemset*, since B ($= L_2.item_1 = L_3.item_1$) and E ($= L_2.item_2 = L_3.item_3$) appear at the first and third positions in L_3, where $(j - 2 = 1)$ and $j = (i + 1) = 3$. Therefore, only the itemset AC remains in L_2. The change of L_2 is shown in Figure 19.

```
procedure gen-Litemset(R'_k, minsup);
begin
  insert into L_k
  select p.item_1, ..., p.item_k, COUNT(*)
  from R'_k p
  group by p.item_1, ..., p.item_k
  having COUNT(*) ≥ :minsup;
end;
```

Figure 10: The *gen-Litemset* procedure

```
procedure filter-DB(R'_k, L_k);
begin
  insert into R_k
  select p.tid, p.item_1, ..., p.item_k
  from R'_k p, L_k q
  where p.item_1 = q.item_1 AND ... AND p.item_k = q.item_k;
end;
```

Figure 11: The *filter-DB* procedure

```
procedure gen-CDB(R_{k-1}, R_{k-1});
begin
  insert into R'_k
  select p.tid, p.item_1, ..., p.item_{k-1}, q.item_{k-1}
  from R_{k-1} p, R_{k-1} q
  where p.tid=q.tid AND p.item_1 = q.item_1 AND ... AND
    p.item_{k-2} = q.item_{k-2} AND p.item_{k-1} < q.item_{k-1};
end;
```

Figure 12: The *gen-CDB* procedure

loop	i	j	$i+j-1$	$j-2$	$j-1$	$i+1$
1	1	1	1	-	-	-
2	1	2	2	-	-	-
3	2	1	2	-	-	-
4	2	2	3	-	-	-
5	2	3	-	1	2	3

Table 3: Changes of the values of the variables for $MaxK = 3$

```
procedure del-Litemset(L_i, L_{i+1}, i, j);
begin
   if (j > 2) then
   begin
      delete
      from L_i p
      where exists
      (select *
       from L_{i+1} q
       where p.item_1 = q.item_1 AND ... AND p.item_{j-2} = q.item_{j-2}
         AND p.item_{j-1} = q.item_j AND ... AND p.item_i = q.item_{i+1})
   else
   begin
      delete
      from L_i p
      where exists
      (select *
       from L_{i+1} q
       where p.item_1 = q.item_j AND ... AND p.item_i = q.item_{i+j-1})
   end;
end;
```

Figure 13: The *del-Litemset* procedure

```
(* Step 2: Deleting items backwards *)
   for i := (MaxK - 1) downto 2 do
      for j := (i + 1) to MaxK do
      begin
         (* comb(j, i) is a function to compute C_i^j and generate COMBD *)
         loop_times := comb(j, i);
         for k := 1 to loop_times do
         begin
            delete
            from L_i p
            where exists
            (select *
             from L_j q
             where p.item_1 = q.item_{COMBD[k][1]} AND ... AND p.item_i = q.item_{COMBD[k][i]};
         end;
      end;
```

Figure 14: Step 2 in Agrawal's Algorithm (denoted as BLmax)

```
function comb(j, i):integer;
begin
    (* compute C_i^j *)
    x:=1;
    y:=1;
    for k := j downto (i + 1) do
        x := x * k;
    for k := (j − i) downto 1 do
        y := y * k;
    total_times := x div y;
    (* generate COMBD *)
    for k := 1 to i do
        COMBD[1][k] := k;
    for k := 2 to total_times do
    begin
        for x := 1 to i do
            COMBD[k][x] := COMBD[k − 1][x];
        p := i;
        COMBD[k][p] := COMBD[k][p] + 1;
        while (COMBD[k][i] > j) do
        begin
            p := p − 1;
            COMBD[k][p] := COMBD[k][p] + 1;
            for x:= (p + 1) to i do
                COMBD[k][x] := COMBD[k][x − 1] + 1;
        end;
    end;
    (* return value *)
    comb:=total_times;
end;
```

Figure 15: The *comb* function

k	COMBD[k][i]
1	1234
2	1235
3	1236
4	1245
5	1246
6	1256
7	1345
8	1346
9	1356
10	1456
11	2345
12	2346
13	2356
14	2456
15	3456

Figure 16: The contents of COMBD computed from function *comb(6,4)* ($1 \leq i \leq 4$)

(* Deleting items backwards*)
for $i := (k-1)$ downto 1 do
  Delete all itemsets in L_i contained in some subsets of L_j, $j > i$;

Figure 17: A backward approach

(* Deleting items forwards *)
for $i := 1$ to $(k-1)$ do
  Delete all itemsets in L_i contained in some subsets of L_{i+1};

Figure 18: A forward approach

i	j	Deleted Item	Resulting L_i
2	1	BC	AC BE CE
2	2	CE	AC BE
2	3	BE	AC

Figure 19: Change of L_2 ($i = 2$)

loop	i	j	$i+j-1$	$j-2$	$j-1$	$i+1$
1	1	1	1	-	-	-
2	1	2	2	-	-	-
3	2	1	2	-	-	-
4	2	2	3	-	-	-
5	2	3	-	1	2	3
6	3	1	3	-	-	-
7	3	2	4	-	-	-
8	3	3	-	1	2	4
9	3	4	-	2	3	4
10	4	1	4	-	-	-
11	4	2	5	-	-	-
12	4	3	-	1	2	5
13	4	4	-	2	3	5
14	4	5	-	3	4	5

Table 4: Changes of the values of the variables for $MaxK = 5$

5 Performance

In this Section, we study the performance of the proposed SETM*-Lmax algorithm by simulation. Our experiments were performed on a PentiumIII Server with one CPU clock rate of 450 MHz, 128 MB of main memory, running Windows-NT 2000, and coded in Delphi. The data resided in the Delphi relational database and was stored on a local 8G IDE 3.5" drive.

5.1 Generation of Synthetic Data

We generated synthetic transactions to evaluate the performance of the algorithms over a large range of data characteristics. The synthetic data is said to simulate a customer buying pattern in a retail environment. The parameters used in the generation of the synthetic data are shown in Table 5. The length of a transaction is determined by a Poisson distribution with mean μ equal to $|T|$. The size of a transaction is between 1 and $|MT|$. The transaction is repeatedly assigned items from a set of potentially maximal large itemsets $\mathcal{F}$, until the length of the transaction does not exceed the generated length [2, 17, 19, 24].

The length of an itemset in $\mathcal{F}$ is determined according to a Poisson distribution with

$\|D\|$	Number of transactions
$\|T\|$	Average size of transactions
$\|MT\|$	Maximum size of the transactions
$\|I\|$	Average size of maximal potentially large itemsets
$\|MI\|$	Maximum size of the potentially large itemsets
$\|L\|$	Number of maximal potentially large itemsets
N	Number of items

Table 5: Parameters

mean μ equal to $|I|$. The size of each potentially large itemset is between 1 and $|MI|$. Items in the first itemset are chosen randomly from the set of items. To model the phenomenon that large itemsets often have common items, some fraction of items in subsequent itemsets are chosen from the previous itemset generated. We use an exponentially distributed random variable with mean equal to the *correlation level* to decide this fraction for each itemset. The remaining items are picked at random. In the datasets used in the experiments, the correlation level was set to 0.5. Each itemset in $\mathcal{F}$ has an associated weight that determines the probability that this itemset will be picked. The weight is picked from an exponential distribution with mean equal to 1. The weights are normalized such that the sum of all weights equals 1. For example, suppose the number of large itemsets is 5. According to the exponential distribution with mean equal to 1, the probabilities for those 5 itemsets with ID equal to 1, 2, 3, 4 and 5 are 0.43, 0.26, 0.16, 0.1 and 0.05, respectively, after the normalization process. These probabilities are then accumulated such that each size falls in a range, which is shown in Table 6. For each transaction, we generate a random real number which is between 0 and 1 to determine the ID of the potentially large itemset. To model the phenomenon that all the items in a large itemset are not always bought together, we assign each itemset in $\mathcal{F}$ a *corruption level* c. When adding an itemset to a transaction, we keep dropping an item from the itemset as long as a uniformly distributed random number (between 0 and 1) is less than c. The corruption level for an itemset is fixed and is obtained from a normal distribution with mean = 0.5 and variance = 0.1. Each transaction is stored in a file system with the form of <transaction identifier, item>.

Some different data sets were used for performance comparison. Table 7 shows the

Itemset ID	Range
1	$0 - 0.43$
2	$0.44 - 0.69$
3	$0.70 - 0.85$
4	$0.86 - 0.95$
5	$0.96 - 1$

Table 6: The probabilities of itemsets after normalization

Case	Name	$\|T\|$	$\|MT\|$	$\|I\|$	$\|MI\|$	$\|D\|$	Size
1	T5.MT10.I2.MI4.D20K	5	10	2	4	20K	1.5MB
2	T10.MT15.I6.MI10.D5K	10	15	6	10	5K	0.8MB

Table 7: Parameter values for synthetic datasets

names and parameter settings for each data set. For all data sets, N was set to 1,000 and $|L|$ was set to 2,000.

5.2 Experiments

In this Section, we compare the performance of our *SETM*-Lmax* algorithm based on a forward approach (denoted as FLmax) with the backward approach described in Agrawal's Algorithm [3] (denoted as BLmax). When we choose the synthetic dataset as T5.MT10.I2.MI4.D20K (Case 1), Figure 20 shows a comparison of execution time between the forward and the backward approaches. The detailed information is shown in Table 8. For this result, we show that our forward approach requires shorter time than the backward approach. Obviously, as the value of the minimum support is decreased, the execution time in both approaches is decreased. Figure 21 shows another simulation result where we choose the synthetic dataset as T10.MT15.I6.MI10.D5K (Case 2), which also shows that forward approach requires shorter time than the backward approach.

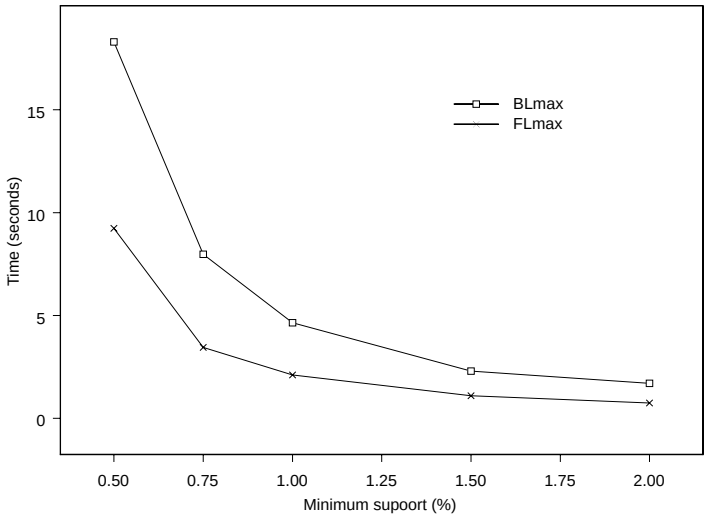

Figure 20: A comparison of execution time between BLmax (the backward approach) and FLmax (the forward approach) (T5.MT10.I2.MI4.D20K: Case 1)

Time	Minimum Support				
(seconds)	0.5	0.75	1	1.5	2
BLmax	18.3	7.98	4.65	2.3	1.7
FLmax	9.24	3.45	2.11	1.1	0.75

Table 8: A comparison of execution time based on different values of the minimum support (T5.MT10.I2.MI4.D20K: Case 1)

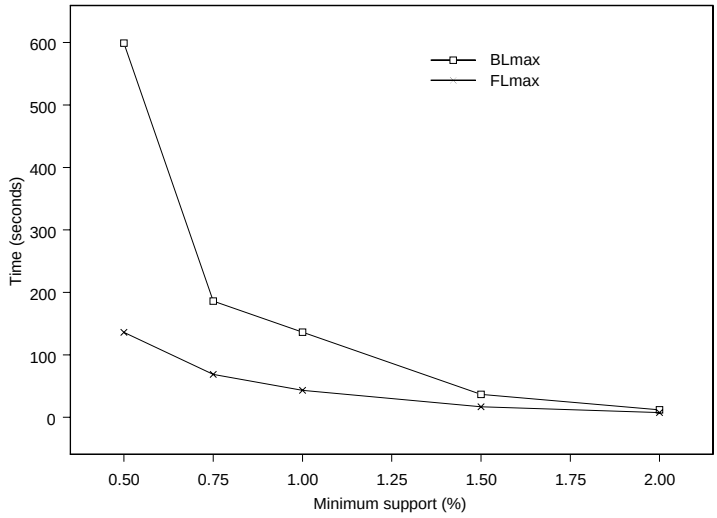

Figure 21: A comparison of execution time between BLmax (the backward approach) and FLmax (the forward approach) (T10.MT15.I6.MI10.D5K: Case 2)

6 Conclusion

Discovery of association rules is an important problem in the area of data mining. Since the amount of the processed data in mining association rules tends to be huge, it is important to devise efficient algorithms to conduct mining on such data [7]. In order to benefit from the experience with relational databases, a set-oriented approach to mining data is needed [14]. In such an approach, the data mining operations are expressed in terms of relational or set-oriented operations. In this paper, to find a large itemset of a specific size in relational database, we have proposed the *SETM*-Lmax* algorithm to find all maximal large itemsets from L_k. We have studied the performance of the proposed SETM*-Lmax algorithm. The simulation results have shown that the proposed forward approach (SETM*-Lmax) to find all maximal large itemsets requires shorter time than the backward approach. In the future, we plan to extend this work to the related problems of mining multiple-level association rules, mining sequential patterns, and mining path traversal patterns directions.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," *Proc. 1993 ACM SIGMOD Int'l Conf. Management of Data,* pp. 207-216, May 1993.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. 20th Int'l Conf. Very Large Data Bases,* pp. 490-501, Sept. 1994.

[3] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 11th IEEE Int'l Conf. Data Engineering,* pp. 3-14, March 1995.

[4] R. Agrawal and K. Shim, "Developing Tightly-Coupled Applications on IBM DB2/CS Relational Database System: Methodology and Experience," *IBM Research Report,* 1995.

[5] R. Agrawal, C. C. Aggarwal and V. V. V. Prasad, "A Tree Projection Algorithm for Generation of Frequent Item Sets," *Journal of Parallel and Distributed Computing,* Vol. 61, No. 3, pp. 350-371, March 2001.

[6] F. Berzal, J. Cubero, N. Marin, and J Serrano, "TBAR: An Efficient Method for Association Rule Mining in Relationall Databases," *Data and Knowledge Engineering,* Vol. 37, No. 1, pp. 47-64, April 2001.

[7] M.-S. Chen, J. Han, and P.S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Trans. on Knowledge and Data Engineering,* Vol. 8, No. 5, pp. 866-882, Dec. 1996.

[8] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P.Indyk, R. Motwani, J. Ullman, and C. Yang, "Finding Interesting Associations without Support Pruning," *IEEE Tansactions on Knowledge and Data Engineering,* Vol. 13, No. 1, pp. 64-78, Jan. 2001.

[9] Y. Fu, "Data Mining," *IEEE Potentials,* pp. 18-20, 1997.

[10] V. Ganti, J. Gehrke, and R. Ramakrishnan, "Mining Very Large Databases," *IEEE Computer,* Vol. 32, No. 8, pp. 38-45, 1999.

[11] J. Han and Y. Fu, "Mining of Multiple-level Association Rules from Large Databases," *IEEE Trans. on Knowledge and Data Engineering,* Vol. 11, No. 5, pp. 798-805, September/October 1999.

[12] J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. 2000 ACM SIGMOD Conf. on Management of Data,* pp. 1-11, May 2000.

[13] J. Han, and J. Pei , "Mining Frequent Patterns by Pattern-Growth: Methodology and Implications," *ACM SIGKDD Explorations (Special Issue on Scaleble Data Mining Algorithms),* Vol. 2, No. 4, pp. 14-20, December 2000.

[14] M. Houtsma and A. Swami, "Set-oriented Mining for Association Rules in Relational Databases," *Proc. 11th IEEE Int'l Conf. Data Engineering,* pp. 25-33, 1995.

[15] Dao-I Lin and Zvi M. Kedem, "Pincer Search: An Efficient Algorithm for Discovering the Maximum Frequent Set," *IEEE Trans. on Knowledge and Data Engineering, Vol. 14, No. 3, pp. 553-565, May/June 2002,* pp. 105-119, 1998.

[16] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Efficient Algorithms for Discovering Association Rules," *Proc. AAAI Workshop Knowledge Discovering in Databases,* pp. 181-192, July 1994.

[17] J.-S. Park, M.-S. Chen, and P.S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules," *Proc. 1995 ACM SIGMOD Int'l Conf. Management of Data,* pp. 175-186, May 1995.

[18] G. Piatetsky-Shapiro, "Discovery, Analysis, and Presentation of Strong Rules," G. Piatetsky-Shapiro and W.J. Frawley, eds., *Knowledge Discovery in Databases, AAAI/MIT Press.* pp. 229-238. 1991.

[19] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. 21th Int'l Conf. Very Large Data Bases,* pp. 432-444, Sept. 1995.

[20] S. Sarawagi, S. Thomas, and R. Agrawal, "Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications," *Proc. 1998 ACM SIGMOD Int'l Conf. Management of Data,* pp. 343-354, 1998.

[21] R. Srikant and R. Agrawal, "Mining Generalized Association Rules," *Proc. 21th Int'l Conf. Very Large Data Bases,* pp. 407-419, Sept. 1995.

[22] D. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal, "Query Flocks: A Generalization of Association-Rule Mining," *Proc. ACM SIGMOD Int'l Conf. on Management of Data,* pp. 1-12, June 2-4, 1998.

[23] S.-Y. Wur and Y. Leu, "An Effective Boolean Algorithm for Mining Association Rules in Large Databases," *Proc. 6th Int'l Conf. Database Systems for Advanced Applications,* pp. 179-186, April 1999.

[24] S.-J. Yen and A. Chen, "An Efficient Approach to Discovering Knowledge from Large Databases," *Proc. 4th Int'l Conf. Parallel and Distributed Information Systems,* pp. 8-18, 1996.