SETM*-MaxK: An Efficient SET-Based Approach to Find the Largest Itemset

Ye-In Chang and Yu-Ming Hsieh

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

Abstract. In this paper, we propose the SETM*-MaxK algorithm to find the largest itemset based on a high-level set-based approach, where a large itemset is a set of items appearing in a sufficient number of transactions. The advantage of the set-based approach, like the SETM algorithm, is simple and stable over the range of parameter values. In the SETM*-MaxK algorithm, we efficiently find the L_k based on L_w, where L_k denotes the set of large k-itemsets with minimum support, $L_k \neq \emptyset, L_{k+1} = \emptyset$ and $w = 2^{\lceil log_2 k \rceil - 1}$, instead of step by step. From our simulation, we show that the proposed SETM*-MaxK algorithm requires shorter time to achieve its goal than the SETM algorithm.

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]. 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 [3]. 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 [1] and DHP [5] are based on the low-level approach, while the SETM algorithm [4] 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. 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 [4], 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 that it generates too many invalid candidate itemsets. In this paper, we design the SETM*-MaxK algorithm to find the largest itemset based on a high-level set-oriented approach. One of the applications to find the largest itemset is that in a grocery store, we may want to know the maximum set of data items which will be bought in one transaction by the most of customers. In the SETM*-MaxK algorithm, we efficiently find the L_k based on L_w, where L_k denotes the set of large k-itemsets with minimum support, $L_k \neq \emptyset, L_{k+1} = \emptyset$ and $w = 2^{\lceil log_2 k \rceil - 1}$, instead of step by step. From our simulation, we show that the proposed SETM*-MaxK algorithm needs shorter time to achieve its goal than the SETM algorithm.

2 The SETM*-MaxK Algorithm

Sometimes, we may only want to know the maximum set of data items which will be bought in one transaction by the most of customers. That is, we only want to find out the maximum k such that $L_k \neq \emptyset$ and $L_{k+1} = \emptyset$. In this Section, we present the *SETM*-MaxK* algorithm to achieve such a goal. In the proposed algorithm, we make use of the "jump" approach and the binary search approach to efficiently find the maximum k. We use the "jump" approach to efficiently construct L_k based on L_w, until $L_k = \emptyset$, where $w = k/2$.

Table 1. Variables used in the *SETM*-MaxK* algorithm

$R_k^{'}$	A database of candidate k-itemsets(i.e., a candidate DB)
L_k	Large k-itemsets
C_k	Candidate k-itemsets
R_k	A database of large k-itemsets(i.e., a filtered DB)
$half\_k$	Last k processed
eq_len	Record the length of the same items in the join step

Table 1 shows the variables used in the *SETM*-MaxK* algorithm. The complete algorithm are shown in Figures 2. In procedure *SETM*-MaxK*, the first step (i.e., the forward phase) is to generate $R_k^{'}$, L_k, and R_k, except $R_1^{'}$, until $L_k = \emptyset$ or $C_k = \emptyset$, where $k = 2^i$, $i \geq 0$. To generate counts for those patterns in $R_k^{'}$ that meet the minimum support constraint, we call procedure *gen-Litemset*. Before we go on to generate patterns of length $k + 1$, we first have to select the tuples from $R_k^{'}$ that should be extended; that is, those tuples that meet the minimum support constraint. We also wish the resulting relation to be sorted on $R_k(trans\_id, item_1, \ldots, item_k)$. This can be done by calling procedure *filter-DB*. After L_1 and R_1 are constructed, we then apply the "jump" approach. We repeat calling procedures *gen-2k-C, gen-2k-CDB, gen-Litemset* and *filter-DB* to generate $C_k, R_k^{'}$, L_k and R_k, respectively, until $L_k = \emptyset$ or $C_k = \emptyset$, where $k \geq 2$. In procedure *gen-2k-C*, we construct C_k based on L_{half_k}, where $half\_k = k$ div 2. In procedure *gen-2k-CDB*, we could generate all lexicographically ordered patterns of length k stored on $R_k^{'}(trans\_id, item_1, \ldots, item_k)$ based on R_{half_k} only. 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-2k-CDB*. Moreover, to avoid unnecessary construction of $R_k^{'}$, L_k and R_k, we will first construct C_k before we construct L_k. That is, $R_k^{'}, L_k$ and R_k will be generated only if $C_k \neq \emptyset$.

In step 2 of procedure *SETM*-MaxK*, we will keep changing the value of $targetK$ by considering the range between k and $half\_k$ $(= k/2)$, until $MaxK \neq 0$. We apply a variant (marked with **) of the binary search, in which when $L_{targetK} = \emptyset$, in additional to updating HK, we will update $LK = LK + 1$ and compute $R_{LK}^{'}$, L_{LK} and R_{LK} if $C_{LK} \neq \emptyset$. In this way, for the next loop, $R_{targetK}^{'}$ can be generated based on a new R_{LK}. For the example as shown in Figure 1-(a), Figure 1-(b) shows the process of procedure *SETM*-MaxK* (as shown in Figure 2), where the minimum support = 3.

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

(a)

L1

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

R1

TID	Items
1	ABCDE
2	ABCEF
3	ACDF
4	ABCDE
5	BCDH
6	DEF
7	ACDG
8	ABCDEH
9	BCEG
10	EFGH
11	AGH
12	BFH

R'2

TID	Itemsets
1	AB AC AD AE BC BD BE CD CE DE
2	AB AC AE AF BC BE BF CE CF EF
3	AC AD AF CD CF DF
4	AB AC AD AE BC BD BE CD CE DE
5	BC BD BH CD CH DH
6	DE DF EF
7	AC AD AG CD CG DG
8	AB AC AD AE AH BC BD BE BH CD CE CH DE DH EH
9	BC BE BG CE CG EG
10	EF EG EH FG FH GH
11	AG AH GH
12	BF BH FH

L2

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

R2

TID	Itemsets
1	AB AC AD AE BC BD BE CD CE DE
2	AB AC AE BC BE BF CE EF
3	AC AD CD
4	AB AC AD AE BC BD BE CD CE DE
5	BC BD BH CD
6	EF
7	AC AD CD
8	AB AC AD AE BC BD BE BH CD CE DE
9	BC BE CE
10	EF
12	BH

R'4

TID	Itemsets
1	ABCD ABCE ABDE ACDE BCDE
2	ABCE ABEF ACEF BCEF
4	ABCD ABCE ABDE ACDE BCDE
8	ABCD ABCE ABDE ACDE BCDE

L4

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

R4

TID	Itemsets
1	ABCD ABCE ABDE ACDE BCDE
2	ABCE
4	ABCD ABCE ABDE ACDE BCDE
8	ABCD ABCE ABDE ACDE BCDE

R'5

TID	Itemsets
1	ABCDE
4	ABCDE
8	ABCDE

L5

Itemset	Sup.
ABCDE	3

R5

TID	Itemsets
1	ABCDE
4	ABCDE
8	ABCDE

(b)

Fig. 1. (a) An example transaction database ; (b) The process of the SETM*-MaxK algorithm

```
procedure SETM*-MaxK;
begin
(* Step 1: Forward Phase *)
  $k := 1$;
  $L_1 := gen\text{-}Litemset(Sales, minsup)$;
  $R_1 := filter\text{-}DB(Sales, L_1)$;
  repeat
    $k := 2 * k$;
    $half\_k := k$ div 2;
    $eq\_len := 0$;
    $C_k := gen\text{-}2k\text{-}C(L_{half\_k}, L_{half\_k})$;
    if $C_k \neq \emptyset$ then
    begin
      $R'_k := gen\text{-}2k\text{-}CDB(R_{half\_k}, R_{half\_k})$;
      $L_k := gen\text{-}Litemset(R'_k, minsup)$;
      $R_k := filter\text{-}DB(R'_k, L_k)$;
    end;
  until ($L_k = \emptyset$ OR $C_k = \emptyset$);

(* Step 2: Find MaxK *)
  $HK := k$;
  $LK := half\_k$;
  $targetK := \lceil (HK + LK)/2 \rceil$;
  $MaxK := 0$;
  repeat
    $eq\_len := 2 * LK - targetK$;
    $C_{targetK} := gen\text{-}2k\text{-}C(L_{half\_k}, L_{half\_k})$;
    if $C_{targetK} \neq \emptyset$ then
    begin
      $R'_{targetK} := gen\text{-}2k\text{-}CDB(R_{LK}, R_{LK})$;
      $L_{targetK} := gen\text{-}Litemset(R'_{targetK}, minsup)$;
      $R_{targetK} := filter\text{-}DB(R'_{targetK}, L_{targetK})$;
    end;
    if ($L_{targetK} = \emptyset$) or ($C_{targetK} = \emptyset$) then
    begin
      $HK := targetK$;
      $LK := LK + 1$; (**)
      $eq\_len := LK - 2$; (**)
      $C_{LK} := gen\text{-}2k\text{-}C(L_{LK-1}, L_{LK-1})$; (**)
      if $C_{LK} \neq \emptyset$ then (**)
      begin (**)
        $R'_{LK} := gen\text{-}2k\text{-}CDB(R_{LK-1}, R_{LK-1})$; (**)
        $L_{LK} := gen\text{-}Litemset(R'_{LK}, minsup)$; (**)
        $R_{LK} := filter\text{-}DB(R'_{LK}, L_{LK})$; (**)
      end; (**)
      if ($L_{targetK} = \emptyset$) then
        $MaxK := LK - 1$;
    end
    else
      $LK := targetK$;
    if ($MaxK = 0$) then
    begin
      $targetK := \lceil (HK + LK)/2 \rceil$;
      if ($targetK = HK$) then
      begin
        if $L_{targetK} = \emptyset$ then
          $MaxK := targetK - 1$
        else
          $MaxK := targetK$;
      end;
    end;
  until MaxK $\neq 0$;
end;
```

```
procedure gen-Litemset($R'_k$, minsup);
begin
  insert into $L_k$
  select $p$.item$_1, \ldots, p$.item$_k$, COUNT(*)
  from $R'_k$ $p$
  group by $p$.item$_1, \ldots, p$.item$_k$
  having COUNT(*) $\geq$ :minsup;
end;

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;

procedure gen-2k-C($L_{half\_k}, L_{half\_k}$);
begin
  insert into $C_k$
  select $p$.item$_1$, ..., $p$.item$_{half\_k}$,
         $q$.item$_{eq\_len+1}$, ..., $q$.item$_{half\_k}$
  from $L_{half\_k}$ $p$, $L_{half\_k}$ $q$
  where $p$.item$_1 = q$.item$_1$ AND ...
        AND $p$.item$_{eq\_len} = q$.item$_{eq\_len}$
        AND $p$.item$_{half\_k} < q$.item$_{eq\_len+1}$;
end;

procedure gen-2k-CDB($R_{half\_k}, R_{half\_k}$);
begin
  insert into $R'_k$
  select $p$.tid, $p$.item$_1$, ..., $p$.item$_{half\_k}$,
         $q$.item$_{eq\_len+1}$, ..., $q$.item$_{half\_k}$
  from $R_{half\_k}$ $p$, $R_{half\_k}$ $q$
  where $p$.tid=$q$.tid
        AND $p$.item$_1 = q$.item$_1$ AND ...
        AND $p$.item$_{eq\_len} = q$.item$_{eq\_len}$
        AND $p$.item$_{half\_k} < q$.item$_{eq\_len+1}$;
end;
```

Fig. 2. The *SETM*-MaxK* procedure

3 Performance

In this Section, we study the performance of the proposed SETM*-MaxK algorithm, and make a comparison with the SETM [4] 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. Table 2 shows the parameters. The length of an

Table 2. Parameters

$\|D\|$	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

itemset in $\mathcal{F}$(potentially maximal large itemsets) is determined according to a Poisson distribution with 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. 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>. Let Case 1 denotes $|T| = 5, |MT| = 10, |I| = 4, |MI| = 8$, $|D|$ = 20K, $Size$ = 1.5MB, N = 1,000 and $|L|$ = 2,000. When we choose Case 1 the synthetic dataset and with a minimum support = 0.33%, the detailed information about $|R_k^{'}|, |L_k|$,

$|R_k|$ and the execution time in these two algorithms are shown in Tables 3-(a), and 3-(b). From these tables, similarly, we observe that the execution time of SETM*-MaxK algorithm is shorter than that of the SETM algorithm.

4 Conclusion

Discovery of association rules is an important problem in the area of data mining. In order to benefit from the experience with relational databases, a set-oriented approach to mining data is needed. In this paper, to find a large itemset of a specific size in relational database, we have efficiently found the L_k based on L_w, where $L_k \neq \emptyset, L_{k+1} = \emptyset$ and $w = 2^{\lceil log_2 k \rceil - 1}$, instead of step by step. From our simulation results, we have shown that the proposed SETM*-MaxK algorithm requires shorter time to achieve their goals than the SETM algorithm.

Table 3. A comparison of storage space and execution time (Case 1): (a) SETM; (b) SETM*-MaxK.

	L_1			L_2			L_3			L_4		
	R_1'	L_1	R_1	R_2'	L_2	R_2	R_3'	L_3	R_3	R_4'	L_4	R_4
SETM	116799	209	-	323149	714	130093	198789	384	50955	49139	121	11068

	L_5			L_6			Total Time
	R_5'	L_5	R_5	R_6'	L_6	R_6	(seconds)
SETM	7938	2	147	10	0	0	62.45

(a)

SETM*-MaxK												
L_1			L_2			L_4			L_5			Total Time
R_1'	L_1	R_1	R_2'	L_2	R_2	R_4'	L_4	R_4	R_5'	L_5	R_5	(seconds)
116799	209	113491	306160	714	130093	96673	121	10068	2464	2	147	46.63

(b)

References

1. 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.
2. F. Berzal, J. Cubero, N. Marin, and J Serrano, "TBAR: An Efficient Method for Association Rule Mining in Relational Databases," *Data and Knowledge Engineering,* Vol. 37, No. 1, pp. 47-64, April 2001.
3. 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.
4. 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.
5. J.-S. Park, M.-S. Chen, and P.S. Yu, "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules," *IEEE Trans. on Knowledge and Data Engineering,* Vol. 9, No. 5, pp. 813-825, Sept. 1997.
6. 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.