An Effective Algorithm for Mining Association Rules with Multiple Thresholds

Yi Siou Lin, Kun Yuan Chang, Cheng Chen
Institute of Computer Science and Information Engineering National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
Email:{linys, kychang, cchen}@csie.nctu.edu.tw
Tel: +886-3-5712121 ext 54734

Abstract

Catering the buying behaviors of customers becomes more and more important by the popularization of E-Commerce recently. How to find the association rules efficiently from the transaction records is one of the most interesting topics to be investigated. In this paper, at first, we propose en efficient algorithm, named Early Pruning Partition algorithm (EPP), with extending the concept of Partition algorithm and using an early pruning technology to improve the performance of mining frequent itemsets under single minimum support. Then we add the checking of multiple thresholds in EPP algorithm to construct our Multiple Thresholds Early Pruning Partition algorithm (MTEPP). Our MTEPP algorithm can find more effective frequent itemsets corresponding to some events of buying behavior. For evaluating our algorithm, we also implement a simulation environment to verify it. According to our evaluations, our algorithms outperform than that of previous methods and find the more useful frequent itemsets indeed. The detailed descriptions of our algorithms will be given in the contents.

Keywords

Data Mining, Association Rules, Frequent Itemset, E-Commerce, Multiple Thresholds

1. Introduction

Recently, with the popularization of E-Commerce, how to predict consumer psychology and tendency correctly becomes more and more important [14][8]. One of the most interesting topics in data mining is the problem of *discovering association rules* over *basket data* [1][12-15]. An association rule consists of two subsets, *antecedent* and *consequent*, such that each subset may include several items. And these two subsets apply to an implied relation [6]. The possibility of this situation is the *confidence* of this association rule [1][2][6]. We can generate some recommendations from association rules. The search of frequent itemsets is one of the most important procedures of discovering association rules. How to develop an algorithm to gather frequent itemsets quickly is quite meaningful.

We use an *early pruning* technique to construct our *Early Pruning Partition algorithm* (EPP). It is based on the concept of Partition algorithm embedded with early pruning technique to achieve better performance. However, if there are many the same transactions generated suddenly in a short time interval such as a week or a month and then disappear, which is called *sudden event*,

traditional single threshold algorithms are not suitable to solve this case. To aim at this weakness, we add the checking of multiple thresholds in EPP to construct the *Multiple Thresholds Early Pruning Partition algorithm* (MTEPP). Our EPP algorithm outperforms at most 37% than that of Apriori algorithm and 13% than that of Partition algorithm from the simulation results. Comparing with the total execution time, our MTEPP algorithm outperforms averagely 50% than that of Apriori algorithm and 28% than that of Partition algorithm.

The remaining part of our paper is organized as follows. In section 2, we give a formal description of discovering frequent itemsets problem and give an overview of the previous algorithms. In section 3, we describe early pruning technology and our EPP algorithm in some detail. The benefits of using multiple thresholds and our MTEPP algorithm are described in section 4. In section 5, our simulation environment is introduced and some preliminary evaluations of our algorithms are collected and compared with other previous methods. Finally, some concluding remarks are given in section 6.

2. Background and Related Work

2.1 Problem Description [1-3] [6-7]

Let $I=\{i_1, i_2,\ldots, i_m\}$ be a set of literals called items. Let D be a set of transactions. Each transaction T contains a set of items $i_i, i_j,\ldots, i_k \subset I$. An association rule is an implication of the form $X \Rightarrow Y$, where $X,Y \subset I$, $X \cap Y = \varnothing$. A set of items is called an *itemset*. An itemset with k items in it is referred to as *k*-itemset. Each itemset $X \subset I$ has a measurement called *support*. The *support*(X) equals the fraction of transactions in D containing X. A rule has a measure of its strength called the *confidence*, where *confidence*($X \Rightarrow Y$) = *support*($X \cup Y$)/*support*(X) .

The problem of mining association rules is to find all the association rules of the form $X \Rightarrow Y$ that satisfy the following two conditions, *support*($X \cup Y$) ≥ ***minsup*** and *confidence*($X \Rightarrow Y$) ≥ ***minconf***. Here, *minsup* and *minconf* are user specified thresholds. All itemsets that have support above the user specified minimum support are generated. These itemsets are called the *large* itemsets (or *frequent* itemsets). For each large itemset X and any $Y \subset X$, check if rule $(X-Y) \Rightarrow Y$ has confidence above *minconf*. These rules are called the *strong* rules. Generating strong association rules from frequent itemsets is relatively straightforward. However, finding all frequent itemsets is nontrivial if the size of the set of items, $|I|$, and the database, D, are large. We can conclude two following properties by observation:

1. Any subset of a frequent itemset must also be frequent. (2.1-1)
2. Any superset of an infrequent itemset must also be infrequent. (2.1-2)

In the following subsections, we will introduce some basic algorithms based on the above problem description. For easy description, we usually notate the set of the frequent k-itemsets as L_k and the set of the candidate k-itemsets as C_k.

2.2 Apriori Algorithm [1][2]

Apriori algorithm is one of the most important level-wise algorithms for discovering frequent itemsets. Items are sorted in

lexicographic order. Frequent itemsets are computed iteratively in the ascending order of size. Assume the largest frequent itemsets contain k items, it takes k iterations for mining all frequent itemsets. Initial iteration computes the frequent 1-itemsets L_1. Then, for each iteration i≤ k, all frequent i-itemsets are computed by scanning database once. Iteration i consists of two phases. First, the set C_i of candidate i-itemsets are created by joining the frequent (i-1)-itemsets in L_{i-1} found in the previous iteration. Next, the database is scanned for determining the support of all candidates in C_i and the frequent i-itemsets L_i are extracted from these candidates. This iteration is repeated until no more candidates can be generated. The Apriori algorithm needs to take k database passes to generate all frequent itemsets. For disk resident databases, this requires reading the database completely for each pass resulting in a large number of disk reads. It means that the Apriori algorithm takes a huge I/O operations. This is the major weakness of it.

2.3 Partition Algorithm [12][10]

The Partition algorithm focuses on times of scanning database. Initially the database *D* is logically partitioned into *n* partitions. Basically the Partition algorithm can be divided into two phases. In phase I, it takes n iterations. During iteration i, only partition p_i is considered. Then take a partition p_i as input and generate **local large itemsets** of all lengths, L^i_2, L^i_3, …, L^i_m as the output. In the merge phase the local large itemsets with the same lengths from all *n* partitions are combined to generate the **global candidate itemsets**. The set of global candidate itemsets of length j is computed as $C^G_j = \cup_{i=1,\ldots,n} L^i_j$. In phase II, the algorithm sets up counters for each global candidate itemset and counts their support in all *n* partitions. It outputs **global large itemsets** that have the minimum **global support** along with their support. Unlike the Apriori algorithm, the Partition algorithm just needs scan the entire database twice. Thus, we will adopt the principle of Partition algorithm combined with early pruning technique to develop a more efficient method, called EPP, and then extend it to handle multiple thresholds with mining association rules. In the following, we will describe them in some detail.

3. Early Pruning Partition algorithm

3.1 Early Local Pruning [9]

The idea of *early local pruning* is as follows. When reading a partition p_i to generate L^{pi}_1, we record and accumulate the number of occurrences for each item. Thus, after reading partition p_i, we know both the number of occurrences for each item in partition p_i, and the number of occurrences for each item in the *big* partition $\cup_{j=1,2,\ldots,i} p_j$ denoted by $BP_{1\ldots i}$. With this information, we can get frequent 1-itemsets of partition p_i and partition $BP_{1\ldots i}$. We define B^{pi}_1 as the set of all better local frequent 1-itemsets in partition p_i, where $B^{pi}_1 = L^{pi}_1 \cap L^{BP1\ldots i}_1$. With early local pruning, we use B^{pi}_1 instead of L^{pi}_1 to start the level-wise algorithm to construct the set B^{pi} of all better local frequent itemsets in partition p_i. Since the size of B^{pi}_1 is usually smaller than that of L^{pi}_1, B^{pi} can be constructed faster than L^{pi}. With Partition algorithm introduced before ensures that $C^G = \cup_{i=1,2,\ldots,n} L^{pi}$ is a super set of the set L^G of all global large itemsets. With early local pruning, Lemma 1 shows that $C^G = \cup_{i=1,2,\ldots,n} B^{pi}$ is also a superset of L^G.

Lemma 1 *Let the database D be divided into n partitions, $p_1,p_2,\ldots,p_n$. Let $C^G = \cup_{i=1,2,\ldots,n} B^{pi}$. Then, C^G is a superset of L^G.[9]*

Proof: see [9].

3.2 EPP algorithm

In previous subsection, we have introduced the idea of early local pruning. Then we will describe how to combine it with Partition algorithm. We propose a method named *Early Pruning Partition algorithm* (EPP algorithm) for mining frequent itemsets quickly. Figure 1 is the complete steps of EPP algorithm. We use an array of object named "*item*" to keep the accumulative result such that item[j] means the item this site sold which id is equal to j. The object "item" consists of three elements: *BPcount, count* and *tidlist* to stores the occurrence of this item from the first reading partition to the big partition, current partition and the occurrent transaction id. Here we also record the length account of each transaction in each partition and the length account in the whole database. When we have read the partition p_i completely, we generate the better frequent 1-itemsets by using early local pruning. We collect all items which count are larger than the local minimum support = *minsup*/n as L^{pi}_1, and check BPcount of all items with the big partition minimum support = i**minsup*/n. If it is larger, put the item into L^{BPi}_1. Then we get B^{pi}_1 by union L^{pi}_1 and L^{BPi}_1.

After getting the better frequent 1-itemsets B^{pi}_1, we calculate the length of local maximum frequent transactions. Finally, we use the procedure *gen_frequent_itemset* to generate all local frequent itemsets from B^{pi}_1.

Algorithm EPP
// Phase I
1. **for** i = 1 to n **do begin**
2. **forall** transaction t ∈ p_i **do begin**
3. **forall** item j ∈ t **do begin**
4. accumulate the occurrence of item[j]
5. **end**
6. accumulate the local and global maximum transaction count
7. **end**
8. calculate the values of local minimum support and big partition support
9. generate the better local frequent 1-itemset B^{pi}_1 of partition p_i
10. calculate the length of local maximum frequent transaction M
11. B^{pi} = *gen_frequent_itemset*(M, Sup, B^{pi}_1, item)
12. reset all local variables
13. **end**
14. $C^{G'} = B^{p1} \cup B^{p2} \cup \ldots \cup B^{pn}$
15. calculate the global minimum support M^G
16. $C^G = \{ c \in C^{G'} \mid c.length \le M^G\}$

// Phase II
17. **forall** transaction t ∈ D **do**
18. **forall** subset s ∈ t **do begin**
19. **if** s ∈ C^G **then**
20. s.count ++
21. **end**
22. $L^G = \{ c \in C^G \mid c.count \ge minsup\}$
23. **Answer** = L^G

Figure 1. Early Pruning Partition algorithm

Procedure gen_frequent_itemset(*M*, Sup, B^{pi}_1, item)

1. **for** (k=2; $B^{pi}_k \neq \varnothing \wedge k \leq M$; k++) **do**
2. **forall** distinct itemsets $l_1, l_2 \in B^{pi}_{k-1}$ **do**
3. **if** $l_1[1]=l_2[1] \wedge \ldots \wedge l_1[k-2]=l_2[k-2] \wedge l_1[k-1]<l_2[k-1]$ **then begin**
4. c = ($l_1[1], l_1[2], \ldots l_1[k-1], l_2[k-1]$)
5. **forall** (k-1)-subset s of c **do**
6. **if** all s $\in B^{pi}_{k-1}$ **then begin**
7. c.tidlist = l_1.tidlist $\cap$ l_2.tidlist
8. **if** |c.tidlist| $\geq$ Sup **then**
9. $B^{pi}_k = B^{pi}_k \cup c$
10. **end**
11. **end**
12. $B^{pi} = B^{pi}_1 \cup B^{pi}_2 \cup \ldots \cup B^{pi}_{k-1}$
13. **Answer** = B^{pi}

We inherit the step of Partition algorithm and use B^{pi}_1 instead of L^{pi}_1 as the level 1 frequent itemsets. Here we make an improvement, we add k ≤ *M* to the second term of "for loop". We will not check level larger than M because there is no possible frequent transaction which length is larger than *M*. Then we use the join step of Partition algorithm to generate all new candidate itemsets. We do some check before joining these two tidlists. We check all (k-1)-subsets of c to confirm that all (k-1)-subsets of c belong to better local frequent (k-1)-itemset of partition p_i. With the observation 2.1-2 we introduced in section 2.1, if itemset j is infrequent, all superset of j is also infrequent. After joining the tidlist of c, we can easily compute the support of c by calculating the length of tidlist. Finally we can calculate B^{pi}_k, the set of better frequent k-itemset in partition p_i. At the end of procedure gen_frequent_itemset, we join all sets of better frequent itemset in each level to B^{pi}, the better local frequent itemsets of p_i. After finishing to process all partitions in phase I of EPP algorithm, we can get all better local frequent itemsets B^{p1}, B^{p2}, … B^{pn} of each partition. Before joining all local frequent itemsets, we can use the length of global maximum transactions to eliminate some useless candidates. Finally we can join all remainder candidates from B^{pi} to C^G to get the set of global candidate itemsets. Then we continue to process the phase II of EPP algorithm by rescanning each transaction t in the whole database *D*, and checking all subsets of t. The set of all candidate itemset c whose count is larger than *minsup* is the global frequent itemsets L^G. This is the output of our EPP algorithm.

4. Multiple Thresholds Early Pruning Partition algorithm

4.1 Event latency

When a new popular purchasing behavior occurs, it implies some new association rules will be found. If this kind of behaviors will disappear in a while, we call the appearance of these behaviors as *event*. The new association rules will not be found until the occurrence of purchasing behavior achieves the minimum support. There is latency between the first customer bought and the new frequent itemsets were found. We call this latency as *event latency*. It is intuitively better to reduce the event latency as short as possible because we can make some appropriate recommendations from those association rules quickly. In the previous discussions of frequent itemsets, there is only one global minimum support . It is impossible to reduce the latency unless we reduce the corresponding local minimum support. Before showing how to solve this problem,

Table 1 Notation

H_j	the j^{th} threshold of user specification
$H_i.s$	the local minimum support of threshold H_j
$H_i.v$	the time interval of threshold H_j
H^G	the traditional threshold consist of the global minimum support.
L_{Hi}	Set of local frequent itemset within threshold H_i
C_{Hi}	Set of local candidate itemset within threshold H_i
p_i	The partition I
L^{pi}	Set of all local frequent itemset in partition p_i
L^{pi}_k	Set of all local frequent k-itemsets in partition p_i

we first denote some symbols using in our algorithm. These symbols are listed in Table 1.We propose a multiple thresholds concept as follows. Assume that each threshold consists of one local minimum support and a time interval, denoted by H = (*s*, *v*), where *s* is local minimum support and *v* is time interval. For example, H_1 = (10%, 30) denotes "*the itemsets which appear in more than 10% of transactions happened in the last 30 days*". We can also denote H^G=(*minsup, total days during the store was operating*) as a global threshold. This threshold H^G is used to filter traditional frequent itemsets within the whole database in our algorithm. And then we can use these thresholds to filter frequent itemsets. Thus, we will get several sets L_{H1}, L_{H2}, … L_{Hk} and L^G of frequent itemsets

Algorithm MTEPP
// Phase I
1. **for** i = n **to** 1 **do begin**
2. **for** t = latest transaction in p_i **to** earliest transaction **do begin**
3. **forall** item j ∈ t **do begin**
4. accumulate the occurrence of item[j]
5. **end**
6. accumulate the local and global maximum transaction count
7. **end**
8. calculate the current local and big partition support by gen_current_support()
9. generate the better local frequent 1-itemset B^{pi}_1 of partition p_i
10. calculate the length of local maximum frequent transaction M
11. B^{pi} = *gen_frequent_itemset*(M, Sup, B^{pi}_1, item)
12. reset all local variables
13. **end**
14. $C^G = B^{pn} \cup B^{p(n-1)} \cup \ldots \cup B^{p1}$

// Phase II
15. **for** t = latest transaction in *D* **to** earliest transaction **do begin**
16. **forall** subset s ∈ t **do**
17. **if** s ∈ C^G **then**
18. s.count ++
19. **if** the count of reading transactions fits to $|H_z.v|$ **then**
20. L_{Hz} = { c ∈ C^G | c.count ≥ H_z.s }
21. **end**
22. L^G = { c ∈ C^G, | c.count ≥ H^G.s }
23. **Answer** = L_{H1}, L_{H2}, … L_{Hh}, L^G

Figure 2 Multiple Thresholds Early Pruning Partition algorithm

corresponding to each threshold respectively. The newly additional association rules gathered from these frequent itemsets often correspond to some events. Thus we can make more suitable recommendations according to these rules.

The previous methods like Apriori and Partition algorithms can also generate the additional association rules by run more k+1 times, which will be inefficient obviously.

4.2 MTEPP algorithm

In this subsection, we will describe our *Multiple Thresholds Early Pruning Partition algorithm* (MTEPP) in some detail. Basically MTEPP algorithm is generalized from EPP algorithm. In fact, EPP algorithm is a special case of MTEPP algorithm, with only one threshold. At first, we specify h local thresholds H_1, H_2, … H_h and a global threshold H^G. Figure 2 shows the complete steps of MTEPP algorithm. Unlike the traditional algorithms like Apriori, Partition algorithm, MTEPP algorithm scans database with inverse direction, from the latest transaction to the earliest transaction. While finishing to read a partition pi, we also calculate the better local frequent 1-itemset. We use another procedure *get_current_minsup* to compute the current local minimum support and current big partition minimum support. The procedure get_current_minsup solves some problems in supporting multiple thresholds because a partition p_i may be included in many thresholds. Which support of these thresholds should we use to filter local frequent itemsets? Our solution is to use the minimum value. The reason can be explained in Lemma 2.

Lemma 2 *Let the database D be divided into n partitions, $p_1,p_2,\ldots,p_n$. Assume in partition pi, L_1 is the set of local frequent itemsets with local minimum support s_1. L_2 is the set of local frequent itemsets with local minimum support s_2. If $s_2 \geq s_1$, then $L_2 \subseteq L_1$.*

Proof: It's easy to prove it below.

$\forall$ local frequent itemset t $\in$ L2, t.support $\geq$ s2

$\because$ s2 $\geq$ s1 $\therefore$ t.support $\geq$ s1

$\Rightarrow$ t $\in$ L1 $\Rightarrow$ L2 $\subseteq$ L1 - **Q.E.D** -

The step of procedure get_current_minsup are described below:

Procedure get_current_minsup(P , *i*)

1. Locate H_{cur} in all thresholds H_1, … H_h and H^G where size of transactions in $H_{cur-1}.v < |P| \leq$ size of transactions in $H_{cur}.v$
2. $Sup=\min\left(\frac{|P|}{|H_{cur}.v|}\times H_{cur}.s,\ \frac{|P|}{|H_h.v|}\times H_h.s,\ \frac{H^G.s}{n}\times(n-i+1)\right)$
3. **Answer** = Sup

After finding the better local frequent 1-itemset B^{pi}_1 in partition p_i, we use the same steps as EPP algorithm to calculate the maximum frequent transaction and then finally compute the local frequent itemsets B^{pi}. Then we join all local frequent itemsets to global candidate itemsets C^G and continue to process phase II. In the second scan, we also reverse the direction of scan. While finishing to read all transactions specified in threshold H_i, we calculate the frequent itemsets L_{Hi}. For the previous assumption $|H_1.v| < |H_2.v| < \ldots < |H_h.v| < |H^G.v|$, we can easily check this from H_1 to H_h and H^G. After reading all transactions in database D, we can finally calculate the global frequent itemsets L^G. The sets L_{H1}, L_{H2}, … L_{Hh}, L^G are the final outputs of our MTEPP algorithm.

5. Simulation Environment and Performance Evaluation

5.1 Simulation Environment

The whole simulation environment is implemented in a PC with a 1000MHz AMD ThunderBird processor and 512 MB main memory. All simulation data resided on a 13 GB IDE Ultra DMA 66 hard disk. The operating system is FreeBSD 4.3-stable. We use C++ language to implement and g++ compiler to compile these algorithms. The overall architecture of our simulator is shown in Figure 3.

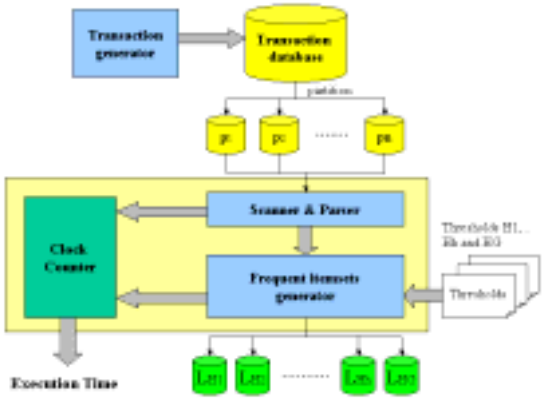

Figure 3 The overall architecture of our simulator

The synthetic data generation procedure is described in [2], whose parameters are shown in Table 2. We use the generator developed by **IBM Quest project** to generate the synthetic database [16]. The length of a transaction is determined by a *Poisson distribution* with mean μ to $|T|$. We have generated four different databases by using the generator and used them for performance evaluation. Table 3 shows the names and parameter settings for each database.

5 .2 Performance Evaluation

5.2.1 Evaluation of Single Threshold Algorithms

Table 2 Parameters of synthetic database generator

$\|D\|$	Number of transactions
$\|T\|$	Average number of items per transactions
$\|I\|$	Average number of items of maximal potentially frequent itemsets
$\|L\|$	Number of maximal potentially frequent itemsets
N	Number of items

Table3 Parameters setting of our input databases

Name	|T|	|I|	|D|	|L|	N	Size(MB)
T10.I4.D100K	10	4	100K	2000	1000	31
T20.I2.D100K	20	2	100K	2000	1000	63
T20.I4.D100K	20	4	100K	2000	1000	63
T20.I6.D100K	20	6	100K	2000	1000	63

In this subsection we will compare our EPP with Apriori and Partition algorithms. We use two databases, T10.I4.D100K and T20.I4.D100K, to test the variation with changing minimum support. We divide the database into 10 partitions for EPP and Partition algorithms. The results for the T10.I4.D100K and T20.I4.D100K databases are shown in Figure 4 and Figure 5 respectively. Our EPP algorithm always significantly outperforms Apriori and Partition algorithm at total execution time. For the case of database T10.I4.D100K, EPP algorithm outperforms Apriori from 21% to 44.9% with minimum support of 2% to 0.25% respectively. The average efficiency of EPP is better than Apriori with 31.2%. Our EPP algorithm is also better than Partition algorithm with average efficiency of 12%.

In the case of data base T20.I4.D100K, our EPP performs more efficient than that of T10.I4.D100K, It outperforms Apriori with average efficiency of 32.4% and Partition

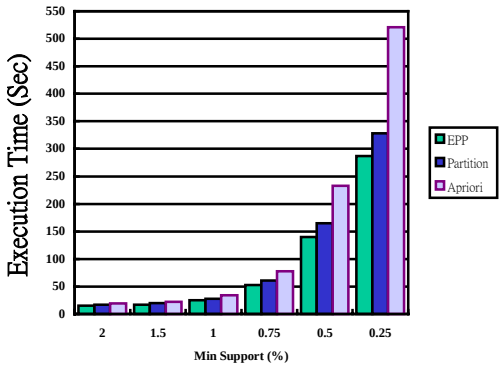

Figure 4 The execution time of database T10.I4.D100k

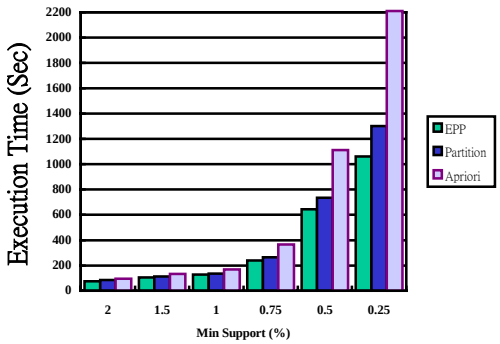

Figure 5 The execution time of database T20.I4.D100

algorithm with average efficiency of 10.7%.

With these results, we can derive that our EPP algorithm performs a better execution time than that of Apriori algorithm when the minimum support is changed smaller and smaller. The reason is that there are more candidate itemsets in smaller minimum support, and we use the early pruning and partition technology to reduce the size of candidate itemsets. With smaller minimum support the improvement of early pruning and partition technology will be more obvious.

5.2.2 Evaluation of Multiple Thresholds

Our interesting evaluation is about the relation of execution time and number of thresholds. We use two databases, T10.I4.D100K and T20.I4.D100K, and set the global minimum supports as 0.5%. And then we adopt 1 to 6 thresholds using in our MTEPP algorithm and other algorithms as six conditions. The first condition is just using the global threshold H^G and the second condition is using H_1 and H^G as thresholds, and so on. Table 4 shows the parameters of these six thresholds.

The evaluation result is illustrated in Figure 6 and Figure 7 corresponding to database T10.I4.D100K and T20.I4.D100K respectively. When the number of thresholds increases, the total execution time of all algorithms grows. In the case of input database as T10.I4.D100K, our MTEPP algorithm outperforms 39.9% to 60.3% than that of Apriori algorithm, the average value is 50.1%. And MTEPP is also faster than that of Partition about 9.1% to 33.7%, average value is 24.4%. In another database T20.I4.D100K, our MTEPP algorithm still has a better performance. It outperforms average 51% than that of Apriori algorithm. And the execution time of MTEPP is also faster than that of Partition about 31.6% in average. Basically the increasing rate of MTEPP is much lower than that of Apriori and Partition. Especially in Apriori algorithm, the growing curve looks like an exponential function. Although the growing rate of Partition algorithm is not very large, it is still larger than that of our MTEPP algorithm. According to the results of our simulation, we can claim that our MTEPP algorithm is a not only effective but also efficient algorithm for mining more useful association rules.

6. Concluding Remarks

In this paper we presented a more efficient method, EPP algorithm, for mining global

Table 4 The global threshold and five additional thresholds

Threshold	H^G	H_1	H_2	H_3	H_4	H_5
Minimum Support	0.5%	0.075%	0.125%	0.175%	0.225%	0.275%
Number of transactions in time interval	100K	10K	20K	30K	40K	50K

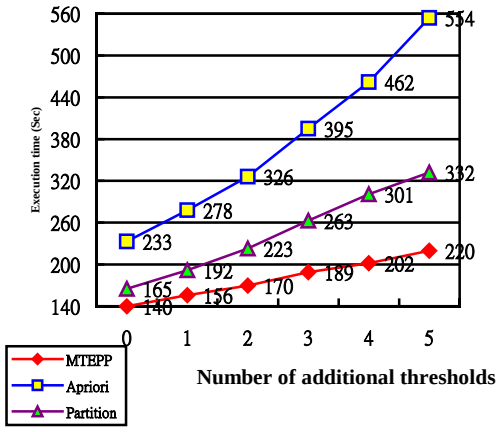

Figure 6 Execution time and number of thresholds within database T10.I4.D100K

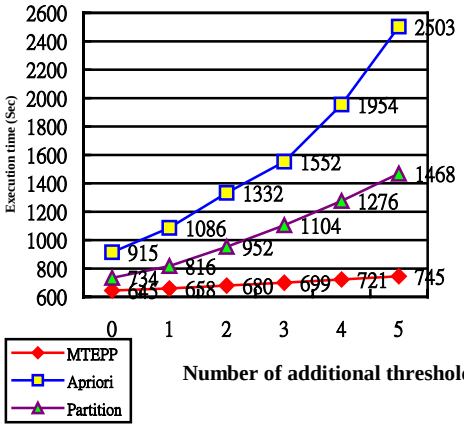

Figure 7 Execution time and number of thresholds within database T20.I4.D100k

frequent itemsets faster. The EPP algorithm outperforms average 31.8% than that of the Apriori algorithm. For comparing the Partition algorithm, the EPP algorithm also outperforms an average value 13%. The second interested concept of event latency is proposed in our algorithm. How to discover this kind of events quickly is very important. We extend the EPP by using multiple thresholds, named MTEPP, to solve this problem effectively. According to our simulation results, our MTEPP algorithm outperforms averagely 50.6% than that of Apriori algorithm with multiple operations and 28% than that of Partition algorithm. In addition to our previous features, there are still several promising issues in future research. First, how to generate the local frequent itemsets of each partition more quickly is worth to be considered. If we can join some more efficient algorithm such as fast intersection or additional candidate pruning, the whole performance of our algorithm can be improved further [7]. Another approach is that we can combine the association rules with the issue of clustering [11]. Generally the concept of clustering is used to group customers. We can gather the association rules from these subdatabases corresponding to groups from clustering. We believe that it is more suitable for the corresponding group of customers. Furthermore, cooperating with incremental mining algorithms will make the whole recommendation system more complete. The incremental mining algorithm tries to decrease the probability of scanning the whole database by analyzing those new coming transactions first [4][5].

Reference

[1] R. Agrawal, T. Imielinski, A. Swami, "Mining Association Rules between Sets of Items in Large Databases", *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp.207-216, May, 1993.

[2] R. Agrawal, R. Srikant, "Fast Algorithms

for Mining Association Rules", *Proceedings of the 20th International Conference on Very Large Databases*, pp.487-499, Sep. 1994.

[3] M. Chen, J. Han, P. S. Yu, "Data mining: An overview from a database perspective", *IEEE Transactions on Knowledge and Data Engineering 1996*, 8(6): pp. 866-833, Dec. 1996.

[4] D. W. Cheung, J. Han, V. Ng, et al. "Maintenance of discovered association rules in large databases:an incremental updating technique", *In Proceedings of the 1996 International Conference on Data Engineering*, New Orleans,Louisiana,1996.

[5] D. W. Cheung, S. D. Lee, B. Kao, "A general incremental technique for maintaining discovered association rules", *In Proceedings of the 5th International Conference on Database Systems for Advanced Applications*, Melbourne, Australia, 1997.

[6] Jiawei Han, Micheline Kamber, **Data Mining: Concepts and Techniques**, *Morgan Kaufmann Publishers*, Chapter 6, pp. 225-229, 2001.

[7] J. Hipp, U. Guntzer and G. Nakhaeizadeh, "Algorithms for Association Rule Mining –A General Survey and Comparison", *SIGKDD Explorations, ACM SIGKDD 2000*, Volume 2, Issue 1, pp. 58-64, Aug. 2000.

[8] Hsiangchu Lai, Tzyy-Ching Yang, "A System Architecture of Intelligent-Guided Browsing on the Web", *Proceeding of 31st Annual Hawaii International Conference on System Sciences*, pp.423-432, Jan. 1998.

[9] Jun-Lin Lin, Margaret H. Dunham, "Mining association rules: anti-skew algorithms", *Proceedings of 14th International Conference on Data Engineering*, pp. 486-493, Feb. 1998.

[10] A. Mueller, **Fast sequential and parallel algorithms for association rule mining: A comparison**, *Technical Report CS-TR-3515, Dept. of Computer Science, Univ. of Maryland*, College Park, MD, Aug. 1995.

[11] Olfa Nasraoui, Raghu Krishnapuram, Anupam Joshi, "Relational clustering based on a new robust estimator with application to Web mining", *Proceeding of 18th International Conference of the, NAFIPS*, pp. 705–709, Dec.1999.

[12] A. Savasere, E. Omiecinski, S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases", *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95)*, pp. 432-444, Sep, 1995.

[13] H. Toivonen, "Sampling Large Databases for Association Rules", *Proceedings of the 22nd VLDB Conference*, pp. 134-145, Sep. 1996.

[14] Don-Lin Yang, Hsiao-Ping Lee, "The CPOG Algorithm for Mining Association Rules", *Proceedings of the 5th Conference on Artifficial Intelligence and Applications (TAAI 2000)*, pp. 65-72, Taipei Taiwan, Nov. 2000.

[15] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New Algorithms for Fast Discovery of Association Rules", *Proceeding of the 3rd International*

Conference on KDD and Data Mining(KDD'97), pp. 283-286, Aug. 1997.

[16] http://www.almaden.ibm.com/cs/quest/syndata.html, *Quest Synthetic Data Generation Code*, IBM Quest project at IBM Almaden Research Center.