Efficient Mining of Maximal Frequent Itemsets with the Closed and Intersection Approach

Xian-Hui Cheng
Dept. of Information Engineering and Computer Science, Feng Chia University
Taiwan
sanhue@soft.iecs.fcu.edu.tw

Don-Lin Yang
Dept. of Information Engineering and Computer Science, Feng Chia University
Taiwan
dlyang@fcu.edu.tw

ABSTRACT

The application of association rules becomes more and more popular in the field of data mining. In this research we propose a new approach of mining rules, the Bottom-up Closed and Top-down Intersection algorithm by discovering maximal frequent itemsets efficiently. Based on the Pincer-Search's bottom-up and top-down process, our method combines the closed set with the intersection operation. In the bottom-up process, our algorithm can accelerate the search speed by using the closed itemsets. In the top-down process, we use the intersection operation to generate minimal number of possible candidates. After scanning the source database at the beginning, we use a transaction frequency table to record the frequency of each transaction appearing in the database. To improve the efficiency of the intersection operation, the transaction frequency table is ordered by length and lexicographic sequence. Our method can not only find possible maximal frequent itemsets much faster and reduce the unnecessary database scanning time, but also use less memory space than the algorithms using the tree-structure. When the difference of the number of 1-itemsets and the length of transaction is larger, our algorithm has much better performance.

1: Introduction

Recently, data mining has been used frequently to discover valuable information from huge databases. Finding frequent patterns to produce useful rules is one of the most important parts of data mining. There are two aspects in achieving the above goal: pattern-growth and generate-and-test. In addition, finding maximal frequent itemset (MFI) provides a more efficient approach because a maximal frequent itemset involves subsets that can be extended to get all frequent itemsets. Some algorithm stores the data in a special form such as a tree structure [1, 3]. We can get all frequent itemsets by traversing the tree. But this method requires a lot of memory space when a large database is used.

Other algorithms use Apriori-based methods to solve this problem. One of the most famous algorithms is Pincer-Search[4]. The Pincer-Search uses a two-way search method to finding maximal frequent itemsets: bottom-up search and top-down search. This method can quickly complete all candidates checking. Like all Apriori-based algorithms, the Pincer-Search generates a lot of candidates. And the algorithm performs well when the maximal frequent itemset is short. The CMFI applies the concept of closed set to solve this problem. It performs well no matter if the length of maximal frequent itemsets is long or short.

Here we propose an algorithm that applies the concept of intersection when it generates the candidates in the top-down process. Our approach has all the advantages of the CMFI and the Pincer-search. However, it is not like the Pincer-search that generates the candidates from the longest itemset. Instead, it starts from the longest transaction such that it reduces the process time of candidate generation.

Although most of the stores may sell thousands of items, only a few people will buy different goods more than twenty of them in general. Thus, when there are many 1-itemsets and the maximum length of the transactions is shorter than the number of frequent 1-itemsets, our approach will have very good result. Therefore, it is very suitable to apply our method in the market basket analysis.

In Section 2, we will describe related work on mining maximal frequent itemsets. The detail concept of our new algorithm is explained in Section 3. Performance evaluation is presented in Section 4. Section 5 concludes the paper.

2: Related Work

2.1: The Pincer-Search Algorithm

Pincer-Search algorithm[4] uses a two-way search—bottom-up and top-down search. The primary direction is still bottom-up where top-down is giving information for pruning candidate. One of the algorithm's important characteristics is that it does not require detail examination of every frequent itemset. Therefore, the algorithm performs well even when some maximal frequent itemsets are long. The output of the algorithm is the maximal frequent set, i.e., the set containing all maximal frequent itemsets, thus providing immediately all frequent itemsets.

2.2: The Closed Algorithm

The Closed algorithm[5, 6, 9, 11] discoveries closed sets for mining association rules in very large databases. The Close is based on the pruning of closed itemset lattice, thus it is often much smaller. Such a structure is much likely related to Wille's concept lattice [10] in formal concept analysis. Figure 1 is a lattice structure of closed set.

This structure can be used for discovering association rules given the property that all sub-closed itemsets of a frequent closed itemset are frequent. In addition, we know that all sub-closed itemsets of an infrequent closed itemset are infrequent and the set of maximal frequent itemsets is identical to the set of maximal frequent closed itemsets. In fact, the closed itemset is a simplified lattice of the original lattice as shown in Figure 2. Closed set is a maximal set of items common to a set of objects. Take Figure 2 for example, the itemset {*B*, *C*, *E*} is a closed itemset since it is the maximal set of items common to the transactions {2, 3, 5}. {*B*, *C*, *E*} is called a frequent closed itemset for minimum support = 2 as support ({*B*, *C*, *E*}) = ||{2, 3, 5}|| = 3 ≥ *minsup*. In a market basket database, this means that 60% of the customers (3 customers on a total of 5) purchase at most the items *B*, *C*, *E*. The itemset {*B*, *C*} is not a closed itemset since it is not a maximal group of items common to some objects: all customers purchasing the items B and C also purchase the item E.

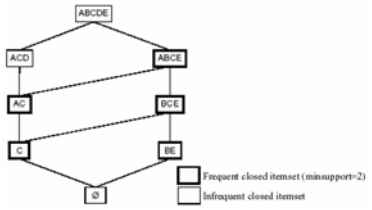

Figure 1. Lattice structure of closed set

Database *D*

TID	Items
1	A C D
2	B C E
3	A B C E
4	B E
5	A B C E

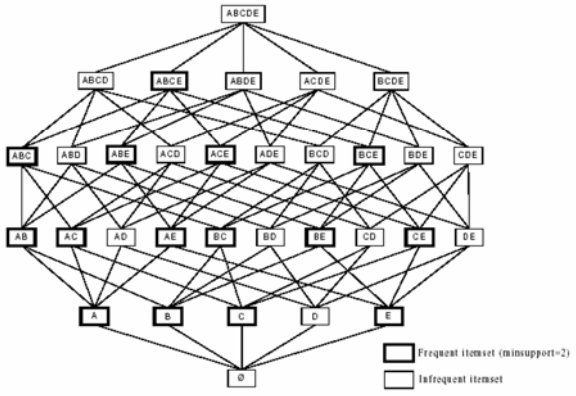

Figure 2. Lattice structure of database D

3: Our Proposed BCTI Algorithm

3.1: Overview

Most of the algorithms for mining the maximal frequent itemset can be classified into two types—Apriori-based[4, 7, 8] and tree-based[1-3]. In this study we propose an algorithm that can efficiently reduce the number of database scanning, called Bottom-Up Closed and Top-Down Intersection (BCTI) mining algorithm. We combine the concepts of Pincer-Search, Close, and Intersection.

Just like the Pincer-Search, the process of BCTI algorithm is divided into two steps—bottom-up and top-down. In addition, we introduce a sorted transaction frequency table (*TFT*) at the first database scan such that each transaction is recorded as $T = \{TID, t-itemset, frequency\}$. This data structure can help us complete the bottom-up and top-down process quickly. In the bottom-up process, the concept of Close is used. We use these closed sets to accelerate the speed of the bottom-up process. In the top-down process, we introduce a method that uses the intersection to find the maximal frequent candidate itemset. Using the *TFT*, we can be sure that an itemset not in the *TFT* can never become a maximal frequent itemset if it cannot be produced by the transactions on the top of the table. In these processes, we can quickly find the maximal frequent itemset.

The process of BCTI can be divided into four steps as shown below:

(1) Step 1: we set the minimum support and scan the database to get 1-itemsets. In this step, we also count each item's support, record its closed set and the number of transactions in the database into the *TFT*. If the support of any item does not meet the minimum support we specify, it is pruned. We combine the items to generate the candidate itemset. By using the properties of closed set, we can prune the items that have the same closed set. Only one of these items will be kept. And we join items to form the candidate 2-itemsets if the closed set of an itemset does not include another's. Then we have all candidate 2-itemsets. We record the information and go on to the next step.

(2) Step 2: we get itemset information from the previous step, including frequent 1-itemsets and the *TFT*. Before we start the top-down process for the first time, we must sort the *TFT*. When passing through the table, we can prune the infrequent items from the transactions and then sort the table by the length of transactions. This *TFT* will be used in next two steps.

(3) Step 3: this step is the top-down intersection process. We get the maximal frequent candidate itemsets from the *TFT* to check the longest itemsets in the table first. Since the support in the *TFT* is not an actual count of the itemset appearing in the database, we must compute the real support for the itemset. When we check the support of the itemset, we only need to check the support of the itemset whose length is longer than the current one. If the support of an itemset meets the minimum support, we keep the information for the bottom-up process. If not, we intersect any two

transactions that are infrequent to generate the candidate set. After checking all the itemsets with the longest length, we can continue to the next step.

(4) Step 4: we know that both the bottom-up search and the top-down search may move many levels in one database pass. If we cannot generate a new candidate in the bottom-up process, we finish the maximal frequent itemset search. Otherwise, we keep on doing bottom-up and top-down search in turns.

In the above process of BCTI, our method reduces the number of database scans and each pass may terminate earlier than other methods by using the *TFT*.

3.2: Top-down Intersection Mining

In the top-down process, we introduce the intersection operation. Our method is based on the use of the sorted *TFT* which records transaction information and is ordered lexicographically when transactions have the same length. Table 1 shows the example of an original database and its *TFT* after pruning infrequent items.

Table 1 Example of TFT

Database D	
TID	Itemsets
1	ABCDEFGI
2	BCDEIKL
3	CDIJ
4	BCDI
5	ABIKL
6	CDEF
7	ABIH

TFT	
itemsets	count
ABCDEFI	1
BCDEIKL	1
ABIKL	1
BCDI	1
CDEF	1
ABI	1

With the *TFT*, we can use the intersection operation to generate the maximal frequent candidate itemset. We know that for an itemset *X* not in the database, it may become frequent if there are two itemsets containing *X* at the same time. On the other hand, for a candidate itemset not in the database, if it cannot be generated from the intersection of two other itemsets, this candidate itemset will never be frequent. For example, if there are two itemsets {*ABCDEFI*} and {*BCDEIKL*} in the database, a candidate itemset {*AK*} cannot be frequent since they cannot generate it by intersection.

The main reason we introduce top-down intersection search is due to the problem of candidate generation. General algorithms like the Pincer-Search generate the candidate by decomposition, thus may result in a lot of candidates. As shown in Figure 3, the itemsets {*ABCD*} and {*BCDE*} are decomposed into {*ABC*, *ABD*, *ACD*, *BCD*} and {*BCD*, *BCE*, *BDE*, *CDE*} respectively in the general top-down search approach. It is very expensive to scan the database for every one of these candidate itemsets. However, if the intersection is used to generate the candidate, we will only have an itemset {*BCD*}. Clearly, using the top-down intersection approach can greatly reduce the number of candidates and the database scanning.

Combining the concept of top-down intersection, bottom-up closed search and *TFT*, we devise the BCTI algorithm to mining maximal frequent itemsets efficiently.

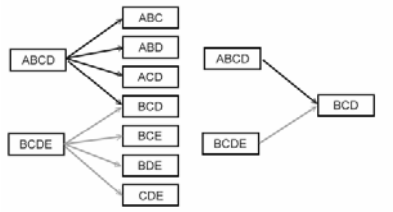

Figure 3. Comparison of decomposition and intersection

3.3: The Concept of the BCTI Algorithm

BCTI is an algorithm for mining the maximal frequent itemset. We use the closed itemset to reduce the candidate in the bottom-up process and use the intersection to generate less number of maximal frequent candidate itemsets in the top-down process. This two-way search method has two closure properties.

Property 1. *If an itemset is infrequent, all its supersets must be infrequent and we need not to examine them.*

Property 2. *If an itemset is frequent, all its subsets must be frequent and we need not to examine them.*

Using these two properties, the time of mining maximal frequent itemsets can be reduced. First, we scan the database once to get all frequent 1-itemsets, their closed itemsets, and the unsorted *TFT*. After pruning the infrequent items from the *TFT*, we sort the table. Second, we start the intersection top-down process from the longest transactions in the *TFT*. If the transaction is infrequent, we use the intersection to generate the maximal frequent candidate itemset; otherwise we send the frequent massage to the closed bottom-up process. This process repeats until no more candidate set can be generated. Using the concepts of closed and intersection, we not only use two–way search to speed up the process, but also reduce the I/O time of scanning the database in our approach.

We will show that the number of database scans in the top-down intersection is less than that of the Pincer-Search. A brief description is as follows. Suppose that there are *n* items in the database and the number of the frequent 1-itemsets is *m*. The top-down process of traditional algorithms starts from the *m*-itemset. Since the *TFT* is used, we can start the top-down process from the longest transaction in the table. Our top-down process starts from the *k*-itemset where *k* is the length of the longest transaction.

We define the parameters used for the performance analysis. Let *n* be the number of items in the database, *m* as the number of frequent 1-itemsets, and *k* as the length of the longest transaction without including infrequent items in the *TFT*. We can easily find that $n \geq m \geq k$, and the number of all candidate sets in the database is $\sum_{i=1}^{n} C_i^n$.

While the number of candidates generated by using the Pincer-Search is $\sum_{i=1}^{m} C_i^m$, the number of candidate sets in the BCTI is $\sum_{i=1}^{k} C_i^k$. We can compute the difference between the candidate sets in our method and the Pincer-Search as shown in the following Eq. (1).

$$\sum_{i=1}^{m} C_i^m - \sum_{i=1}^{k} C_i^k = \sum_{i=0}^{k} C_{m-i}^m \quad (1)$$

From Eq. (1), we know that if the parameter k is much smaller than m, the difference $\sum_{i=0}^{k} C_{m-i}^m$ between our method and the Pincer-Search is larger. Especially, if k gets smaller, the number of database scans becomes much less. If k is equal to m, the number of candidate sets generated from our method is equal to that of the Pincer-Search in the worst case.

Generating a candidate means adding one checking of the itemset or one pass of the database scan. The cost of checking an itemset is low if it is a subset of a frequent itemset or a superset of an infrequent itemset. But the cost of database scanning is usually high.

Table 2. Example of candidates and large 1-itemsets in bottom-up

Item	Closed set	Sup
A	ABI	3
B	BI	5
C	CD	5
D	CD	5
E	CDE	3
F	CDEF	3
~~G~~	~~ABCDEFGI~~	~~1~~
~~H~~	~~ABIH~~	~~1~~
I	I	6
~~J~~	~~CDIJ~~	~~1~~
K	KL	2
L	KL	2

→

Item	Closed set	Sup
A	ABI	3
B	BI	5
C	CD	5
E	CDE	3
F	CDEF	3
I	I	6
K	KL	2

In the closed bottom-up process, the closed itemsets are used to generate less number of candidate sets than the Pincer-Search. If a closure of an itemset equals to another one, then we need to keep just one of them because these itemsets have the same maximal itemset. If the closure of an itemset includes another one, then we do not need to join these itemsets together to produce redundant candidate itemsets. For example, for the database of Table 1, the closed itemset is shown in Table 2. Since items *C* and *D* have the same closed set {*CD*}, we only keep one of them. So do the items *K* and *L*. A's closed set {*ABI*} includes B's closed set {*BI*}, such that we do not need to join items *A* and *B* since the support of {*AB*} is contained in *A*'s support.

As explained above, using the properties of closed itemset, the feature of the *TFT* and top-down intersection in our method, we not only generate a fewer number of candidates, but also reduce the cost of I/O.

3.5: An Example of the BCTI Application

Here we show an example to present the process of the BCTI. We use the database as shown in Table 1. There are 7 transactions and 12 items in the database. The minimum support specified by the user is 2 meaning that an itemset is frequent if its count is greater than or equal to 2. The execution steps of the BCTI are shown below.

Step 1: we scan the database to find the closed sets and the support of each item. At the same time, we generate the unsorted *TFT*. As explained before, infrequent itemsets {*G*}, {*H*}, and {*J*} are pruned. Since {*C*} and {*D*} have the same closed set, we only keep itemset {*C*} in the large 1-itemset—L_1 table. Then we generate candidate 2-itemsets. Because the closed set of itemset {*A*} contains the closed set of itemset {*B*}, we do not generate {*AB*} as a candidate itemset. The execution of Step 1 is shown in Figure 4.

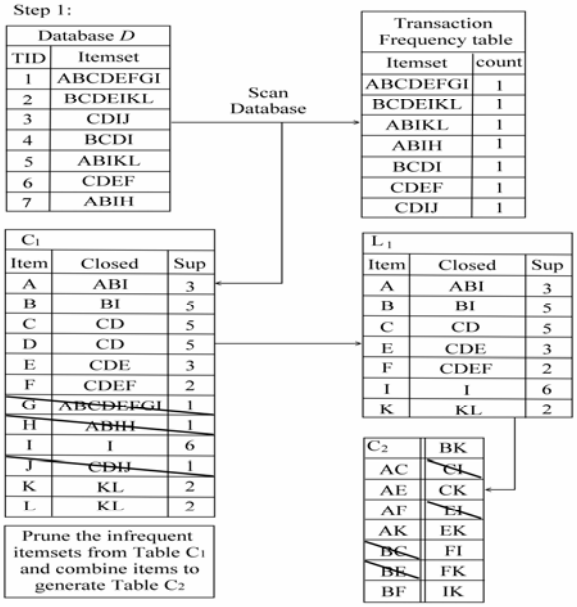

Figure 4. The execution result after Step 1 of BCTI

Step 2: we sort the transaction frequency table. First, the original transaction {*ABCDEFGI*} is changed to {*ABCDEFI*} because {*G*} is an infrequent itemset, such that the length of the new transaction is 7. Figure 5 shows the execution result of Step 2.

Step 3: after transforming the transaction frequency table, we can start the top-down intersection process. Because itemsets {ABCDEFI} and {BCDEIKL} are infrequent, we can generate the itemset {BCDEI} by using the intersection operation. Then we check if the itemset is frequent or not, and pass the result to the next step. We show the execution result of Step 3 in Figure 6.

Step 4: we continue the bottom-up and top-down process in turns. From the candidate set generated in Step 1, we first prune the itemset that is the subset of a frequent itemset {*BCDEI*} from Step 3 as shown in Figure 7. The itemsets with gray background in the table, i.e., {*BC*}, {*BE*}, {*CI*} and {*EI*}, are frequent without

any checking. As described before, we now have the large 2-itemsets. Then, the algorithm terminates because itemsets {*BK*} and {*IK*} have the same closed set {*BIKL*}.

Step2:

Transaction Frequency table	
Itemset	count
ABCDEFGI	1
BCDEIKL	1
ABIKL	1
ABIH	1
BCDI	1
CDEF	1
CDIJ	1

Transaction Frequency table after pruning	
Itemset	count
ABCDEFI	1
BCDEIKL	1
ABIKL	1
BCDI	1
CDEF	1
ABI	1
CDI	1

Prune the infrequent itemsets from the *TFT* and soft it by the transaction length

Figure 5. The execution result after Step 2 of BCTI

Step 3:

Maximal ferquent candidate set		
Itemsets	count	frequent?
ABCDEFI	1	N
BCDEIKL	1	N

If the itemsets are infrequent , intersect them

Maximal ferquent candidate set		
Itemsets	count	frequent?
BCDEI	2	Y

check if the itemset is frequent

Figure 6. The execution result after Step 3 of BCTI

Step4:

C_2

Itemsets	Closed	Sup
~~AC~~	~~ABCDEFI~~	1
~~AE~~	~~ABCDEFI~~	1
~~AF~~	~~ABCDEFI~~	1
~~AK~~	~~ABIKL~~	1
BC	BCDI	3
BE	BCDEI	2
~~BF~~	~~ABCDEFI~~	1
BK	BIKL	2
CI	CDI	3
~~CK~~	~~BCDEIKL~~	1
EI	ABCDEFI	2
~~EK~~	~~BCDEIKL~~	1
~~FI~~	~~ABCDEFI~~	1
~~FK~~		0
IK	BIKL	2

L_2

Itemsets	Closed	Sup
BK	BIKL	2
IK	BIKL	2

Figure 7. The execution result after Step 4 of BCTI

4: Experimental Result

4.1: Environment and Data

We implement the BCTI algorithm using Java programming language. In order to compare with the Pincer-Search, we also implement this algorithm in Java. We perform our experiments on a personal computer of Intel Pentium 4 640 series, 3.2GHZ processor and DDR2 533MHz 2GB main memory. NetBean 5.0 is used to develop our algorithm.

The test data for the experiments are produced by using the IBM dataset generator. We define the parameter D as the number of transactions, T the average size of transaction, I the average size of maximal potentially-large itemsets, L the number of potentially large itemsets and N the total number of items. By setting these parameters, we can generate proper datasets to evaluate the performance of our algorithm.

The test datasets generated are T5I2D10K, T10I4D10K and T15I8D10K. We set the average length of the transaction T as 5, 10, 15, and the average size of the transaction of maximal potentially-large itemsets I as 2, 4, and 8. Since we want to show the difference of the Pincer-Search and our method, we generate the databases having the number of items N = 50 and the number of potentially-large itemsets L = 1000.

4.2: Experimental Result and Discussion

First, we use the databases T5I2D10K and T10I4D10K to run the Pincer-Search and the BCTI. Figure 8 shows that when the average transaction size of potentially-large itemset is set to I = 2 and I = 4 for the lower minimum support 0.25%, the performance of our method is much better than the Pincer-Search. When the minimum support is 0.5%, the execution time of the Pincer-Search is about 3 times of our method. Although our performance is close to the Pincer-Search when the minimum support is 0.75% or 1%, the execution time of the Pincer-Search is still about 1.4 times of our method in average. Table 3 has the detail numbers.

Table 3. Execution time for T5I2D10K and T10I4D10K

T5I2D10K				
Minimum Support (%)	0.25	0.5	0.75	1
Pincer-Search (Sec.)	5623.66	1253.13	324.29	254.62
BCTI (Sec.)	1698.62	431.02	236.58	167.60
T10I4D10K				
Pincer-Search (Sec.)	6851.21	3674.89	948.25	356.84
BCTI (Sec.)	2865.21	1138.65	562.84	324.12

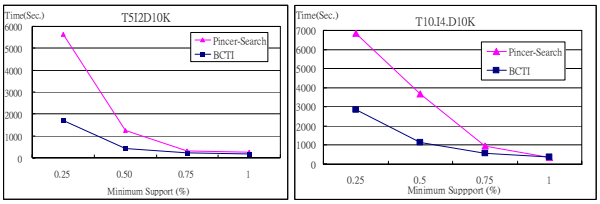

Figure 8. The experiment result of T5I2D10K and T10I4D10K

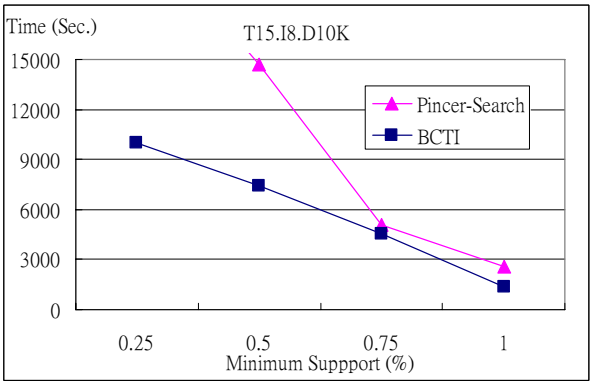

Figure 9. T15I8D10K

In the same way, we set T = 15, I = 8, and D = 10K for analysis. Figure 9 shows that BCTI performs about two times better than the Pincer-Search when the

minimum support is set to 0.25% or 0.5%. When the support is 0.75% or 1%, the performance of the Pincer Search and our method are very close. However, in the worse case, we can find that our performance is still 10% better than the Pincer-Search as shown in Table 4. Therefore, our method performs better when the minimum support is smaller.

Table 4. Execution time for T15I8D10K

T15I8D10K				
Minimum Support (%)	0.25	0.5	0.75	1
Pincer-Search (Sec.)	23253.11	14685.54	5096.38	2601.01
BCTI (Sec.)	10006.68	7432.14	4532.95	1350.52

Finally, we show the scalability test between the BCTI and the Pincer-Search in Figure 10. We set T = 5 and I = 2 in this experiment with the database size from 1K to 100K. In this figure, we can see that both algorithms have good scalability as the database size increases linearly while the BCTI performs better as the database size gets larger.

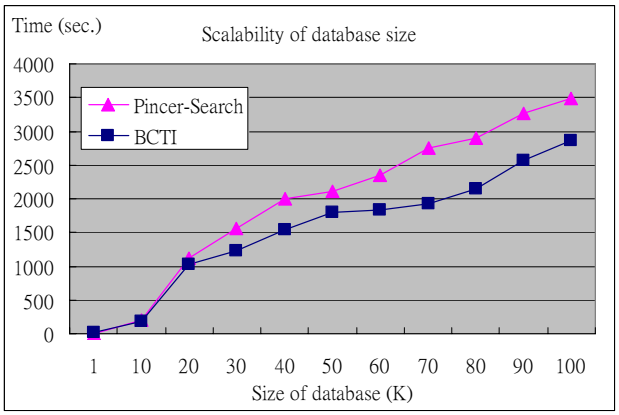

Figure 10.T5I2D10K

5: Conclusions and Future Work

An efficient way to mining maximal frequent itemsets can be very useful in various data mining applications, such as the discovery of the association rules, strong rules, and minimal keys. The maximal frequent itemset provides an efficient representation of the frequent itemsets.

In this paper, we propose the BCTI algorithm for mining the maximal frequent itemsets. We introduce a new concept of top-down intersection process with the transaction frequency table *TFT*. Our approach focuses on mining the maximal frequent itemsets in an efficient way. We combine the concepts of closed set, intersection, and the Pincer-Search. In the bottom-up process, we use closed set to reduce the generation of redundant candidate sets. In the top-down process, we use the intersection with the sorted *TFT*. Thus we can find all the maximal frequent itemsets much more quickly.

Our algorithm has better performance when the difference of the number of frequent 1-itemsets and the length of the longest transaction in the *TFT* is large. In fact, the number of frequent 1-itemsets is mostly larger than the length of the transaction in the real database. For example, a super market may have thousands of goods where the top 100 of them are popular with customers. However, most customers may only buy ten to twenty goods in each trip to the super market. Our approach is very suitable and useful for the market basket analysis, for example, in dealing with the sales database. We can find the maximal frequent itemsets to make better sale strategies. Since our approach can find who buy what products the most, we can analyze the purchase behaviors or habits of the customers in any particular regions. However, the final call of any decision still needs to be made by people.

When the length of the discovered pattern is medium, the performance of our approach is not as good as that of the shorter one. We will continue study the relation between the distribution of datasets and our algorithm. Another future work includes a better use of the transaction frequency table to record more useful information with efficient data structure.

Reference

[1] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. M. Yiu, "MAFIA: A maximal frequent itemset algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 1490-1504, 2005.

[2] K. Gouda and M. J. Zaki, "GenMax: An efficient algorithm for mining maximal frequent itemsets," *Data Mining and Knowledge Discovery*, vol. 11, pp. 223-242, 2005.

[3] G. Grahne and J. F. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 1347-1362, 2005.

[4] D. I. 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, pp. 553-566, 2002.

[5] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 21-36, 2006.

[6] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient mining of association rules using closed itemset lattices," *Information Systems*, vol. 24, pp. 25-46, 1999.

[7] K. Srikymar and B. Bhasker, "Efficiently mining maximal frequent sets for discovering association rules," *presented at Proceedings of the Seventh International Database Engineering and Applications Symposium*, 2003.

[8] H. Wang, Q. H. Li, C. X. Ma, and K. L. Li, "A maximal frequent itemset algorithm," *presented at Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, 2003.

[9] J. Y. Wang, J. W. Han, Y. Lu, and P. Tzvetkov, "TFP: An efficient algorithm for mining top-K frequent closed itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 652-664, 2005.

[10] R. Wille, "Concept lattices and conceptual knowledge systems," *Computers Math. Application*, vol. 23, pp. 493-515, 1992.

[11] M. J. Zaki and C. J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 462-478, 2005.