Volume 1, No. 2, September 2010

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

A Study on the Performance of CT-APRIORI and CT-PRO Algorithms using Compressed Structures for Pattern Mining

Mrs. A.B. Dhivya*[1] and Dr. (Mrs.) B.Kalpana[2]

*[1]Avinashilingam Deemed University for Women, Coimbatore, India.
Email: divya21385@yahoo.com[1]
[2]Avinashilingam Deemed University for Women, Coimbatore, India.
Email: kalpanabsekar@yahoo.com[2]

Abstract: Many algorithms have been proposed to improve the performance of mining frequent patterns from transaction databases. Pattern growth algorithms like FP-Growth based on the FP-tree are more efficient than candidate generation and test algorithms. In this paper, we propose a new data structure named Compressed FP-Tree (CFP-Tree) and an algorithm named CT-PRO that performs better than the current algorithms including FP-Growth and Apriori. The number of nodes in a CFP-Tree can be up to 50% less than in the corresponding FP-Tree. CT-PRO is empirically compared with FP-Growth and Apriori. CT-PRO is also extended for mining very large databases and its scalability evaluated experimentally. All these results point CT-PRO as the right candidate for generating a compact version of the original transaction database, which is small in size and which performs frequent pattern mining in a fast and efficient manner.

Key Words: Frequent Patterns; Transaction Databases; FP-Growth; Apriori; CFP-Tree; CT-PRO.

INTRODUCTION

In a scenario where WWW has become more important every day, to have a clear and well organized web site has become one of the vital goals of enterprises and organizations. Association Rule Mining (ARM) [1] has been the focus of research in many communities (e.g. data mining, artificial intelligence, machine learning) for a decade. Traditionally, ARM has been defined on market basket data. However, it has been used in many other application areas and also extended to data mining tasks of classification [2] and clustering [3]. However, the existing algorithms rely on expensive computations using large amounts of memory or require many I/O scans over the database.

ARM algorithms typically divide the problem into two parts: find the frequent patterns and then use them to form the rules. The general performance of ARM is determined by the first part. Once frequent patterns are found, generating the association rules is straightforward. Constraints such as support and confidence are used to reduce the search space during mining. The Apriori property (if a pattern is infrequent then its supersets can never be frequent) is the foundation for reducing the cost of all algorithms in ARM.

The Apriori algorithm uses the candidate generation and test approach [4]. The main drawback of this approach is the many traversals over the database required to enumerate a significant part of the possible 2n frequent

patterns where n is the number of items. Another factor contributing to the efficiency of FP-Growth is its compact representation of the database in memory using a variant of the prefix tree named FP-Tree.

The use of prefix tree itself was introduced first in [6]. The performance gain from using variants of the prefix tree for representing transactions was previously demonstrated in [5] [7] [8] and [9].

In this paper, we propose a new data structure named Compressed FP-Tree (CFP-Tree for short) that is even more compact than FP-Tree. In this paper, we present a new algorithm named CT-PRO that divides the database into several projections and then mines each projection independently. The projections are also represented as CFP-Trees. The performance of CT-PRO is compared against other known efficient algorithms.

To study the feasible performance range of the algorithm, we carried out extensive testing using a set of databases with varying number of both transactions and average number of items per transaction.

RELATED WORK

Given a set of items I= {I1, I2, I3, …, In} and a database D as a set of transactions T, each transaction is a subset of I (TÍ I) and is identified by a TID. An itemset X is a subset of items (XÍ I), and an itemset of length k is called a k-itemset. The support of an itemset X is the percentage of transactions in D that contains X. If the support of an itemset is greater than or equal to a given support threshold s, it is called a frequent itemset or frequent pattern otherwise it is infrequent. The objective of frequent pattern extraction is to find all frequent patterns, given an input database D and a support threshold s. The input database D can be represented as an m x n matrix where m is the number of transactions and n is the number of items. We can denote the presence or absence of an item in each transaction by a binary value (1 if present, else 0). Counting the support for an item is the same as counting the number of 1s for that item in all the transactions. The sparseness or denseness can be determined by a density measure defined as the percentage of 1s in the total of 1s and 0s. If a dataset contains more 1s than 0s, it can be considered as a dense dataset, otherwise, as a sparse dataset.

Tid	ITEMS

$\Rightarrow$

1	2	3	4	5

1	1	2	3	5			1	1	1	0	1
2	2	3	4	5			0	1	1	1	1
3	3	4	5				0	0	1	1	1
4	1	2	3	4	5		1	1	1	1	1
5	1	2	4	5			1	1	0	1	1

Figure 1. Binary representation of a transaction database

METHODOLOGY

The amount of data stored in databases has increased tremendously with the widespread use of databases and the rapid adoption of information systems and data warehousing technologies. An important type of database that contains huge knowledge of a business is the transaction database. A transaction database contains information about frequently used patterns of potential customers. The process of obtaining this information is called Frequent Pattern Mining and can be discovered using various data mining techniques, like clustering, classification, prediction and association analysis. In this research work two solutions are compared for this purpose. The first is to use a CT-Apriori (Compact Tree-Apriori) algorithm and the second is to use CFP-Tree (Compressed FP-Tree) algorithm. Both the algorithms are based on association rules and the working of both these algorithms is explained in this chapter. The algorithms are analyzed based on memory usage, time and scalability.

Ct-Apriori

Association rule mining algorithms consists of two tasks. The first task focus on generating all frequent itemsets that satisfy the user specified minimum support, while the second uses the frequent itemsets generated in the first task to discover all the association rules that meet a user defined confidence threshold.

Compact Transaction Database

Let I = {i_1, i_2, … , i_m} be a set of m items. A subset $X \subseteq I$ is called an itemset. A k-itemset is an itemset that contains k items.

Definition 3.1: A transaction database TDB = {T_1, T_2, …, T_N} is a set of N transactions, where each transaction T_n ($n \in \{1, 2, \dots, N\}$) is a set of items such that $T_n \subseteq I$. A transaction T contains an itemset X if and only if $X \subseteq T$.

Table I. Example Transaction Database TDB

TID	LIST OF ITEMIDs
001	A, B, C, D
002	A, B, C
003	A, B, D
004	B, C, D
005	C, D
006	A, B, C
007	A, B, C
008	B, C
009	B, C, D
010	C, D

Table II. Compact transaction database of TDB

HEAD				
ITEM	C	B	D	A
COUNT	9	8	6	5
BODY				
COUNT	LIST OF ITEMIDS			
3	C, B, A			
1	C, B, D, A			
1	B, D, A			
1	C, B			
2	C, B, D			
2	C, D			

Algorithm description

CT-Apriori Algorithm: The Apriori algorithm is one of the most popular algorithms for mining frequent patterns and association rules.

Algorithm: CT-Apriori algorithm

Input: CTDB (Compact transaction database) and min sup (minimum support threshold).

Output: F (Frequent itemsets in CTDB)

```
1: $F_1 \leftarrow \{\{i\} \mid i \in$ items in the head of CTDB$\}$
2: **for** each $X,Y \in F_1$, **and** $X<Y$ **do**
3:     $C_2 \leftarrow C_2 \cup \{X \cup Y\}$
4: **end for**
5: $k \leftarrow 2$
6: **while** $C_k \neq \theta$ **do**
7:     **for** each transaction T in the body of CTDB **do**
8:         **for** each candidate itemsets $X \in C_k$ **do**
9:             **if** $X \subseteq T$ **then**
10:                $count[X] \leftarrow count[X] + count[T]$
11:            **end if**
12:        **end for**
13:    **end for**
14:    $F_k \leftarrow \{X \mid support[X] \geq$ min sup$\}$
15:    **for** each $X,Y \in F_k$, $X[i]=Y[i]$ for $1 \leq i \leq k$ **and** $X[k]<Y[k]$ **do**
16:        $L \leftarrow X \cup \{Y[k]\}$
17:        **if** $\forall J \subset L, |J| = k : J \in F_k$ **then**
18:            $C_{k+1} \leftarrow C_{k+1} \cup L$
19:        **end if**
20:    **end for**
21:    $k \leftarrow k+1$
22: **end while**
23: **return** $F = U_k F_k$
```

There are two essential differences between this method and the Apriori algorithm.

1. The CT-Apriori algorithm skips the initial scan of database in the Apriori algorithm by reading the head part of the compact transaction database and inserting the frequent 1-itemsets into F_1. Then candidate 2-itemset C_2 is generated from F1 directly.

2. In the Apriori algorithm, to count the supports of all candidate k-itemsets, the original database is scanned, during which each transaction can add at most one count to a candidate k-itemset. In contrast, in CT-Apriori, as shown in step 10, these counts are incremented by the occurrence count of that transaction stored in the body of the compact transaction database, which is, in most of the time, greater than 1.

Compressed FP-Tree (CFP-Tree): The main objective of compressed FP-Tree (CFP-Tree) is to reduce the size of the FP-Tree size by half. The items are in descending order of their frequency in a CFP-Tree and there is a link to the next node with the same-item-node. FP-Tree stores the item id in the tree while in CFP-Tree the item ids are mapped to an ascending sequence of integers that is actually the array index in HeaderTable. The frequency of each item is also stored in HeaderTable. The FP-Tree is compressed by removing identical subtrees of a complete FP-Tree and by succinctly storing the information from them in the remaining nodes. All subtrees of the root of the FP-Tree (except the leftmost branch) are collected together at the leftmost branch to form the CFP-Tree

Table III. Sample Database

Tid	ITEMS					Tid	ITEMS					Tid	ITEMS				
1	1	2	3	4		6	2					11	1				
2	2	4				7	1	4				12	2	3	4		
3	1	3	4			8	1	2	3			13	1	2			
4	3					9	3	4				14	1	2	4		
5	2	3				10	4					15	1	3			

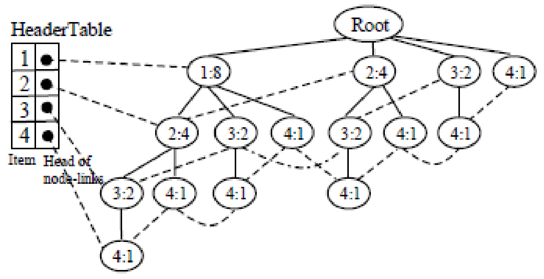


Figure 3. FP-Tree

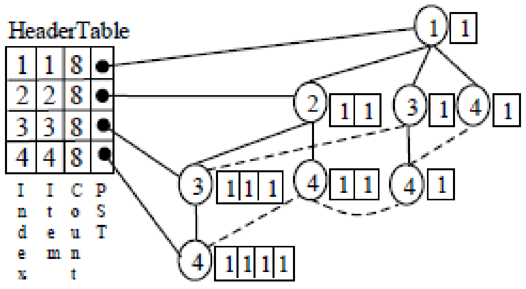


Figure 4. CFP-Tree

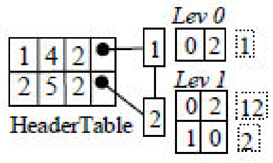
To mine the frequent patterns from the transaction, two additional columns are compared to the HeaderTable of the FP-Tree. These are the frequency count of each item and an index that renames the items arranged in the descending order of frequency. Each node of the CFP-Tree contains an array of counts for item subsets in the path from the root to that node. The index of the cells in the array corresponds to the level numbers of the nodes above. The number of nodes in the FP-Tree is twice that of the corresponding CFP-Tree.

Mining the CFP-Tree using CT-PRO: To mine all frequent patterns in the transaction tree using the CFP-Tree, the pointers in the HeaderTable are used as the starting points.

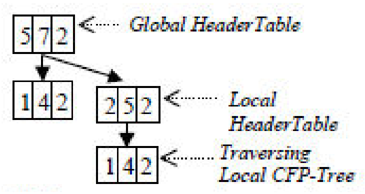
```
1.  /*Input: database Output: HeaderTable*/

2.  Procedure ConstructHeaderTable
3.  For each transaction in the database
4     For each item in a transaction
5       If item in HeaderTable
6         Increment count of item in HeaderTable
7       Else
8         Insert item into HeaderTable with count = 1
9       End If
10    End For
11  End For
12  Delete infrequent items and sort HeaderTable in
    descending order
13  Assign an index for each frequent item


14  /* Input: database, HeaderTable, min_sup Output:
    Global CFP-Tree */
15  Procedure ConstructTree
16  Build_LeftMost_Branch_of_the_Tree()
17  For each transaction in the database
18    Initialize mappedTrans
19    For each frequent item in the trans
20    /*get index of items from HeaderTable*/
21      mappedTrans = mappedTrans È GetIndex(item)
22    End For
23    Sort(mappedTrans)
24    InsertToTree(mappedTrans)
25  End For


26  Procedure InsertToTree(mappedTrans)
27  firstItem = mappedTrans[1]
28  currNode = root of subtree pointed by
    HeaderTable[firstItem]
29  For each subsequent item i in mappedTrans
30    If currNode has child representing i
31      increment count[firstItem-1] of the child node
32    Else
33      create child node and set its count[firstItem-
    1]=1
34      Organise the same-item-node-link
35    End If
36  End For
```

Figure 5. Algorithms for Constructing *CFP-Tree*

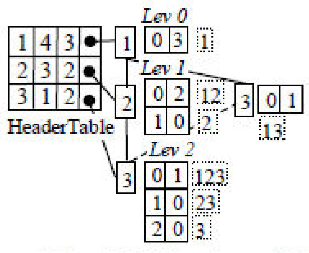
In CT-PRO, for each frequent item f, only one local CFP-Tree is created and traversed non-recursively to extract all frequent patterns beginning with f. By doing this, the cost of creating conditional FP-Trees is avoided as in FP-Growth.



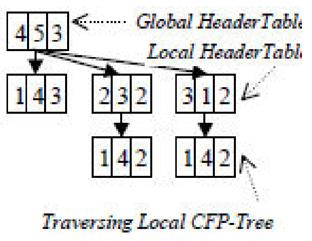
(a) Local FP-Tree 5



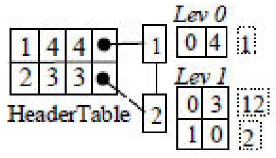
(b) Frequent Patterns in Projection 5



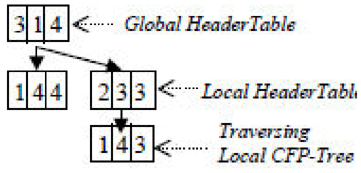
(c) Local FP-Tree 4



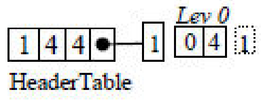
(d) Frequent Patterns in Projection 4



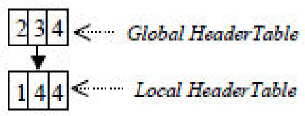
(e) Local FP-Tree 3



(f) Frequent Patterns in Projection 3



(g) Local FP-Tree 2



(h) Frequent Patterns in Projection 2

Figure 6. Local CFP-Tree during Mining Process

Frequent Pattern Mining In Web Log Files: While considering web log files, the main aim is to find the frequent pages visited at the same time, and to discover the page sequences visited by users. The results obtained by the application can be used to form the structure of a portal, satisfactorily for advertising reasons and to provide a more personalized Web portal.

RESULTS AND DISCUSSION

The results obtained for the research work entitled "A Study on the Applicability of Compact Transaction Database using CT-Apriori and Compressed Trees using CT-PRO for Pattern Mining" is discussed in this section.

Test Datasets

The two models selected were tested with two types of datasets. One is the synthetic data which mimic the market basket database and other is the web data which belong to a web log databases. The synthetic data sets used in experiments were generated using the procedure described by [10]. These transactions mimic the actual transactions in a retail environment.

Table IV. Parameters Used In The Synthetic Data Generation Program

PARAMETERS	MEANING
|D|	Total number of transactions
|T|	Average size of transactions
|I|	Average size of maximal potentially frequent itemsets
|L|	Number of maximum potentially frequent itemsets
N	Total number of items

Table V. Parameter Settings of Synthetic Data Sets

TRANSACTION DATABASE	|T|	|I|	|D|
T5I4D50K	5	4	50k
T10I8D100K	10	8	100k
T15I10D100K	15	10	100k
T20I12D200K	20	12	200k
T20I12D300K	20	12	300K

Performance Metrics

While evaluating the algorithms used, compression ratio was considered to be the most important performance metric. Compression ratio is defined as the ratio between the original transaction database size to the compact database size. The compression results with regard to number of association rules were also analyzed.

Apart from storage space required to store the resultant database, the amount of memory utilized during execution also plays a vital role during evaluation. The result of this metric can be used to evaluate the memory utilization complexity of the proposed algorithms.

Time taken to generate the association rules and mine frequent patterns was another parameter that was considered during evaluation.

Results

The results of the various experiments are presented and discussed in this section.

Compression Ratio: The compression result in terms of storage size is shown in Tables VI Table VII shows the compression performance in terms of number of association rules generated.

Table VI. Compression ratio in terms of database size

Transaction Database	Original Size (KB)	CT-Apriori		CT-PRO	
		Compressed Size (KB)	Ratio (%)	Compressed Size (KB)	Ratio (%)
T5I4D50K	1,786	1,430	80.07	1,321	73.96
T10I8D100K	5,013	4,799	95.73	4,672	93.20
T15I10D100K	8,642	7,652	88.54	7,109	82.26
T20I12D200K	16,948	13,987	82.53	13,045	76.97
T20I12D300K	21,315	17,009	79.80	16,178	75.90
Web Data	545	344	63.12	287	52.66
Average Compression Ratio		81.63		75.83	

Table VII. Compression Ratio In Terms Of Number Of Transactions

Transaction Database	Original Size	CT-Apriori		CT-PRO	
		Compressed Size	Ratio	Compressed Size	Ratio
T5I4D50K	50,000	37,878	75.76	36,077	72.15
T10I8D100K	1,00,000	86,928	86.93	85,765	85.77
T15I10D100K	1,00,000	89,347	89.35	87,621	87.62
T20I12D200K	2,00,000	1,62,421	81.21	1,46,411	73.21
T20I12D300K	3,00,000	2,41,931	80.64	2,11,113	70.37
Web Data	32,711	11,233	34.34	9,519	29.10
Average Compression Ratio		74.70		69.70	

From the results, it could be seen that both the algorithms are efficient in generating a compact version of the original database.While considering the web log data the algorithms were able to achieve more compression when compared to synthetic dataset. The results show that the compact transaction databases provide effective data compression.

Execution Time: The overall system performance is analyzed by comparing the average time taken by the selected algorithms. Table VIII presents the average time taken for synthetic datasets and web log data for various support thresholds.

Table VIII. Average Time Taken(Seconds)

Dataset	Apriori	CT-Apriori	FP-Growth	CT-PRO
T5I4D50K	214.75	180.75	186.75	159.00
T10I8D100K	302.25	265.50	272.5	237.75
T15I10D100K	343.00	315.75	297.5	268.50
T20I12D200K	396.75	362.50	350.5	325.50
T20I12D300K	439.75	409.75	394.50	351.75
Web Data	3.65	2.88	3.23	2.53

From the data projected in Table, showing the execution speed performance curves, it is evident that CT-PRO performs better than all the algorithms in all situations. Both CT-Apriori and CT-PRO outperforms their base algorithms Apriori and FP-Growth. The performance gap between CT-Apriori and CT-PRO is more prominent at lower thresholds.

Execution time while using synthetic database is shown in Figures 7, 8, 9, 10 and Figure 11 shows the execution time for web log data.

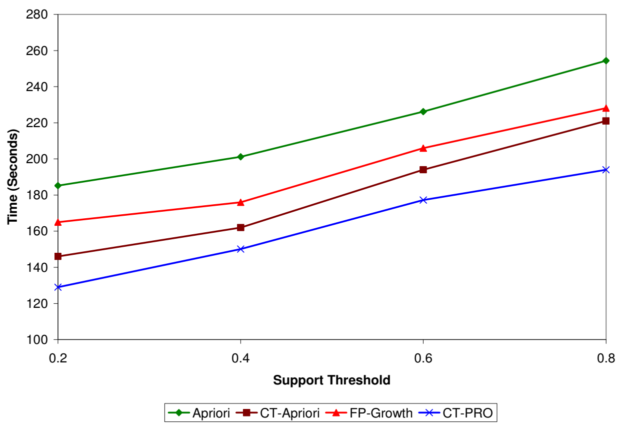


Figure 7: T5I5D50K

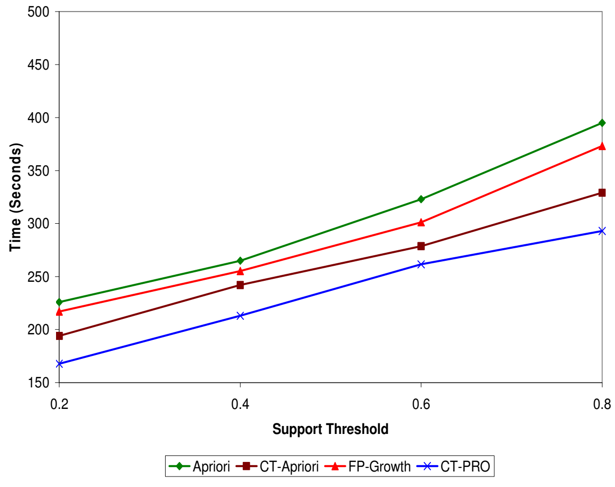


Figure 8: T10I8D100K

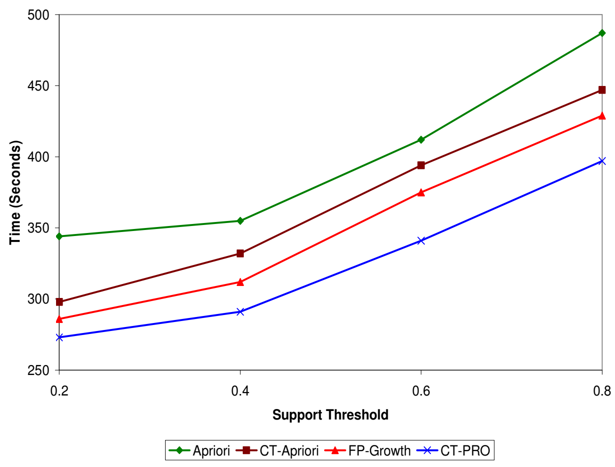


Figure 9. T20I12D200K

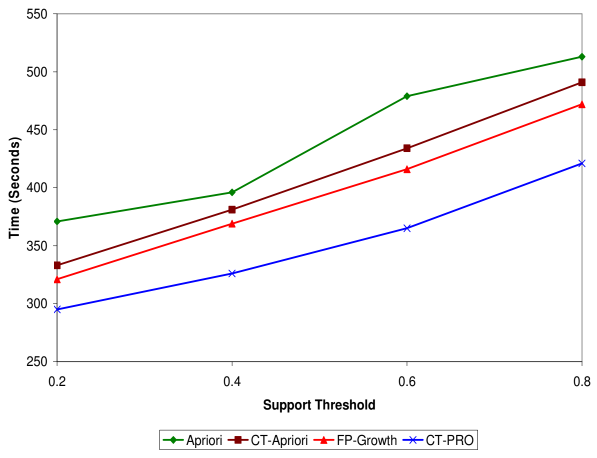


Figure 10. T20I12D300K

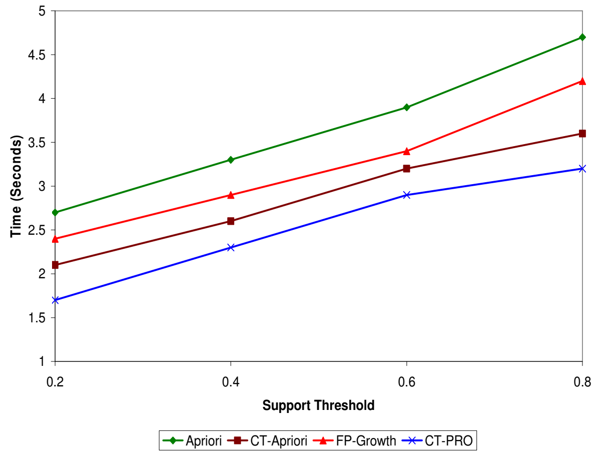


Figure 11. Web log Data

These results indicate that the performance of CT-PRO algorithm in terms of compactness achieved, in terms of storage size, number of transactions and execution speed with different datasets is efficient when compared with all the other algorithms.

SUMMARY AND CONCLUSION

The CT-Apriori algorithm uses a compact tree structure, called CT-tree, to compress the original transactional data. The tree representation allows the CT-Apriori algorithm, which is revised from the Apriori algorithm, to generate frequent patterns quickly by skipping the initial database scan and reducing a great amount of I/O time per database scan. The CT-PRO algorithm uses a compact tree structure called CFP-Tree, which is more compact than the FP-Tree of the FP-Growth algorithm. An algorithm called CT-PRO is used to mine frequent patterns from CFP-Tree. The CT-PRO algorithm divides the CFP-Tree into several projections represented by CFP-Trees. Then CT-PRO conquers the CFP-Tree for mining all frequent patterns in each projection. The execution speed results also indicated that the CT-PRO algorithm was the fastest among all the algorithms. All these results point CT-PRO as the right candidate for generating a compact version of the original transaction database, which is small in size and which performs frequent pattern mining in a fast and efficient manner.

REFERENCE

1. Agrawal, R., Imielinski, T., and Swami, A. (1993) Mining association rules between sets of items in large databases, Buneman, P. and Jajodia, S., editors, Proceedings of the ACMSIGMOD International Conference on Management of Data, ACM Press, Pp. 207–216,Washington.
2. B. Liu, W. Hsu, and Y. Ma, "Integrating Classification and Association Rule Mining," Proceedings of ACM SIGKDD, New York, NY, 1998.
3. K. Wang, X. Chu, and B. Liu, "Clustering Transactions Using Large Items," Proceedings of ACM CIKM, USA, 1999.
4. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proceedings of the 20th International

Conference on Very Large Data Bases, Santiago, Chile, 1994.

5. J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequentpattern Tree Approach," Data Mining and Knowledge Discovery: An International Journal, Kluwer Academic Publishers, vol. 8, pp. 53-87, 2004.
6. R. Agarwal, C. Aggarwal, and V. V. V. Prasad, "A Tree Projection Algorithm for Generation of Frequent Itemsets," Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining), 2000.
7. C. Borgelt, "Efficient Implementations of Apriori and Eclat," Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Melbourne, USA, 2003.
8. J. Liu, Y. Pan, K. Wang, and J. Han, "Mining Frequent Item Sets by Opportunistic Projection," Proceedings of ACM SIGKDD, Edmonton, Alberta, Canada, 2002.
9. W. Cheung and O. R. Zaiane, "Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint," Proceedings of Seventh International Database Engineering and Applications Symposium (IDEAS2003), Hong Kong, China, 2003.
10. Agrawal, R. and Srikant, R. (1994) Fast algorithms for mining association rules in large databases, In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, Santiago, Chile, Pp. 487-499.
11. Bayardo, R.J. (1998) Efficiently mining long patterns from databases, Proceeding of the 1998 ACM-SIGMOD International Conference on Management of Data (SIGMOD'98), Seattle, WA, Pp 85–93.
12. Bell, T., Witten, I.H. and Cleary, J.G. (2009) Modelling for Text Compression, ACM Computing Surveys, Vol. 21, No.4, P.557.