# Memory Cutback for FP-Tree Approach

D. P. Rana
SVNIT
Surat
India

N. J. Mistry
SVNIT
Surat
India

M. M. Raghuwanshi
RGCER
Nagpur
India

## ABSTRACT
The pattern growth approach of association rule mining is very efficient as avoiding the candidate generation step which was utilized in Apriori algorithm. Here, revisited of the pattern growth approaches are done to improve the performance using different criteria like item search order, conditional database representation and construction approach and tree traversal ways. The header table construction is the first part in almost all the approaches having constant number of dataset items. This research is representing the reduction in overall memory requirement of pattern growth approach by reducing the search space and processor operations time at the header table generation. It is proposed to achieve the memory cutback by only considering the items that are going to be frequent and ignoring the infrequent items at early stage of scan, by considering the boundary. Experimental analysis achieves cutback in memory consumption in the proposed approach Modified FP-Growth (MFP-Growth) compare to FP-Growth and CFP-Growth.

## General Terms
Data Mining, Association Rule Mining

## Keywords
Association rule mining, FP-Tree, pattern growth

## 1. INTRODUCTION
Association rule mining one of the data mining techniques finds the associations between items in a transaction dataset [1]. The association rule is represented as A→B, which means if customer buys item A then he also tends to buy item B. Any number of items can be there on antecedent and consequent of the rules. To perform so, two thresholds like support and confidence are used. For the rule A→B, the support is defined as the probability to contain the {A, B} and confidence is defined as the conditional probability that a transaction having item A also having the item B. The association rules which satisfies the support and confidence thresholds are considered as interesting association rules. A large number of association rule mining methods have been reported in the literature. From that commonly used association rule mining algorithms are Apriori and FP-Growth. From these two, FP-Growth has good approach of FP-Tree generation, so this paper is reviewing different FP-Growth algorithms to understand the innovative strategies used by them. From the study, it is found that there is a scope of reduction in time and memory usage by the introduction of boundary. Herewith proposed the modification to the traditional FP-Growth approach and experiments are tested and resulted in the betterment.

Accordingly, this paper is organized as follows: Section 2 is briefing the basic association rule mining approaches and discussing and summarizing the variations of FP-growth approach, Section 3 is describing the problem statement and approach used to reduce the execution time respectively, Section 4 is the experimental and result analysis and finally Section 5 is concluding with the future work area.

## 2. ASSOCIATION RULE MINING
To discover the association between items, the commonly used association rule mining algorithm is the Apriori algorithm which is of the type bottom-up and breadth-first search [2-3]. The basic approach of the Apriori algorithm is that "An itemset is frequent, if and only if all of its subsets are frequent". So, this approach generates k+1-candidate itemsets using k-frequent itemsets and then the algorithm tests the candidate k+1-itemsets to satisfy the given support threshold. This way, it is repeatedly mining all frequent superset of items. At the last, the algorithm provides frequent patterns of all size and the rule directions by filtering the rules using the confidence threshold. Though, the algorithm has limitations of number of candidate generation and multiple pass over the database, other approaches are introduced with different computational efficiencies, scanning of the database, the representation structure used for the transactions and memory utilization. One of the big improvement over Apriori algorithm is proposed by Han et al. achieved by introducing the Frequent Pattern (FP)-Growth approach by reducing the number of database scan and using the efficient storage structure called Frequent Pattern (FP)-tree that mines the frequent pattern without candidate generation [4]. This approach is executed in two phases: During the first phase, in the first database scan it is deriving a list of frequent items known as header table in which items are ordered by frequency descending order and in the second database scan, the complete frequent item database is compressed into an FP-tree. In the second phase, it is constructing the conditional pattern-bases which are the sets of prefix paths in the FP-tree from which conditional FP-tree is constructed to generate the patterns by the concatenation of the suffix pattern with the generated frequent patterns. And the process is recursively mining till the FP-tree is empty or it contains only a single path. The characteristics of this approach is using depth-first search techniques with only two data scans and also avoids the generation of a large number of candidate itemsets, that helps to reduce the execution time of this is quite less compare to Apriori. But, as the approach stores complete FP-tree into the memory that causes the limitation due to limited main memory size. The memory consumption of this approach is reduced by modification to FP-tree called Patricia Tries [5].

Even though, the FP-growth has limitation, it is found as more efficient than Apriori in case of dense datasets with number of long frequent itemsets and low support threshold. Related to this, different approaches are studied and found that the generation of header table is similar like Apriori first step that needs to scan a large number of 1-candidate items, many of which are proved to be infrequent at the end of the first database scan. This research is discussing the

modification before the creation of FP-Tree to reduce the execution time by considering the total operations time only for possible frequent items by using boundary and together with this, it also reducing the space required to store 1-candidate items.

Due to the novelty of FP-tree with restrain of memory size and recursive approach to mine the frequent patterns, the different research works have carried out to improvise the mining ways are briefed here.

Frequent Pattern (FP)-List approach was proposed by Tseng et al. using a simple linear list which stores item node and bit string node with partition [6]. Each item node has information like: frequent item, count and a link to a bit string node. A bit string node manages information to indicate the presence and absence of frequent items in a transaction, its count, a link to a sibling item node. So, the rightmost item node contains the smallest frequency information. This approach mines recursively with the help of simple list operations together with bit counting, transaction trimming and migration. The partitioning and transaction trimming reduce the memory requirement.

TD-FP-Growth proposed by K. Wang et al. searches the FP-tree in the top-down order as the header table is managed in ascending order to mine non recursive way [7]. To save amount of execution time and memory space, it is not generating conditional pattern bases and conditional FP-trees. To achieve this, it is preparing sub header tables of item and merging the information from one sub tree to another

Co-Occurrence Frequent Item (COFI) tree algorithm was introduced by M. El-Hajj et al. where for each frequent item in the header table of FP-tree the relatively small tree is prepared referred as COFI trees [8]. COFI tree is controlling bidirectional links to provide non recursive approach and is helping to avoid the generation of conditional sub-trees. COFI tree manages global frequent and local non frequent property built and mined independently one by one to reduce the memory consumption.

The Ascending Frequency Ordered Prefix Tree (AFOPT) algorithm is another pattern growth algorithm proposed by Liu et al. in which manages the header table of items in ascending order to minimize the number and the size of conditional database [9]. It utilizes the tree and array based structure depending on the dense and sparse dataset respectively. It is mining the first item's conditional database to retrieve all the patterns and then all of its subtrees are merged with its siblings which will be the complete representation of the second item's conditional database to mine, thus increasing the performance.

CT-PRO algorithm is the variation of FP-tree algorithm presented by Y.G. Sucahyo et al. which uses Compress Frequent Pattern tree (CFP-tree) to represent all the items of the transactions in the main memory [10]. Together with header table, it uses the field index with item-id in the descending order of the frequency. Approach is preparing first global CFP-tree as FP-tree but with index value. Then mining is done using projection for the local CFP-Tree from global CFP-tree.

SQL based approach was proposed by S. Xuequn et al. [11]. RDBMS provides the benefits of using the buffer management system freeing the user from the size consideration of the data and limitation of FP-growth tree. It is storing FP-tree as a flat table structure called FP table. To speed up the insertion and mining is overcome by Extended Frequent Pattern (EFP) table which has the same format as table FP but directly obtained by transforming frequent items in transactional table T' and thus the SQL based frequent Pattern mining approach using FP-Growth can get efficient result.

Fast Updated Frequent Pattern-tree (FUFP-tree) proposed by T. Hong et al. utilizing an FP-Tree with bidirectional link [12]. Initially author is preparing FP-tree with given data just like FP-tree approach. Then on arrival of some transactions separately preparing header table for these by considering relative support and merging these transactional items to the original FUFP-Tree incrementally.

To mine the association rules with multiple minimum supports is also an important problem of the association rule mining. First time, with the help of pattern growth approach, the author Y. Hu et al. proposed a MIS-tree (just alike FP-tree structure) based algorithm CFP-growth to mine all frequent itemsets with multiple support values in one database scan is proposed without managing any separate header table. The maintenance algorithm to update the MIS-tree is provided to tune the different supports of items at any stage without rescanning database [13].

Compact Pattern-tree (CP-tree) proposed by S. Tanbeer et al. prepares a compact prefix-tree structure in one database scan and provides the same mining performance as the FP-growth technique by efficient tree restructuring process [14]. The approach is not require extra database scan to generate header table, but on at the collection of user given transactions, inserting and updating the header table and pattern tree. Thus, it is an interactive and incremental approach which requires only single scan of database.

Tree-based Incremental Association Rule Mining (TIARM) approach is proposed by G. Pradeepini et al. [15] to reduce total database scan from 2 to 1. This approach uses INCremental (INC)-Tree data structure which is an extension of FP-Tree constructed by inserting every transaction in database one after another into it directly by sorting the transaction items according to item's appearance order in the database and together it manages the sort list. INC-Tree contains nodes representing items and total count of that item in the path up to that node and connecting each node containing same item. After construction of INC-Tree, before the mining of INC-tree it prunes the tree according to support threshold.

The author Lin et al. [16] proposed the Improved FP-growth (IFP-growth) algorithm which improves the performance of FP-growth in terms of memory reduction by utilizing an address-table structure together with FP-tree and preparing a new structure called FP-tree+ that will help to reduce the recursive building of conditional FP-trees.

## 3. PROBLEM AIM AND APPROACH

From number of approaches discussed here, derived that execution time can be also reduced by considering the operations needed to update the count of items and memory consumption. Here the approach is considering the particular boundary level from which one can grasp that after that boundary level if new item is seem then it will not become frequent from the remainder number of transactions. There is no need to consume memory for these items and thus it is avoiding the update operation for those items. Only to consider the memory for items those are going to be frequent and perform counting operations for the items which are appearing before the boundary.

The aim of the planned approach is to reduce the memory usage by ignoring the non-frequent items at early stage.

In order to achieve less execution time with less candidate items storage, here considered that to become the frequent item, the candidate item has to appear in x number of transactions during the dataset scanning, where x is the boundary. Otherwise candidate item is infrequent as the item appears in transactions less than the x number of transactions. Thus one can say that for the consideration of the item as frequent item, it must have to be occurred once in the boundary.

Here, the boundary is represented by the help of the given support Sup. For the given support Sup the item is frequent, if it is present in the boundary=(Total number of Transactions - Sup + 1) number of transactions. This boundary can be used during the dataset scanning. In the first database scan, at the confirmation boundary, appears the items which may be almost frequent and after the boundary the approach needs to scan the transactions and to perform the count updating operations only for these items and for rest new items just ignore them and even not to allocate memory for them. And then continue with the second database scan to prepare FP-Tree and mining as usual like FP-Growth.

The algorithm steps are as shown in the following Fig 1.

**Algorithm MFP-Growth:** MFP-Growth header table generation is by creating boundary
**Input:** Minimum support threshold Sup
**Output:** Full set of frequent patterns
Step 1: Calculate Boundary=TotalTransactions–Sup+1
Step 2: Scan dataset
Step 3: If CurrentTransactionNo <= Boundary
Step 4: If NewItem Add it to the CandidateListItem with 1Count
Step 5: Else Increment count of CandidateListItem
Step 6: Else Increment the count of CandidateListItem
Step 7: If Not End of the Dataset Go to Step 2
Step 8: Filter Infrequent Items and sort the header table
Step 9: Generate FP-Tree in 2nd dataset pass
Step 10: Mining of FP-Tree

**Fig 1: Algorithm Steps**

Here, Step 1 is calculating boundary for the transactions. Steps 3 to 5 are to add all items appearing within the boundary to candidate list as considered that item is appearing then only it will be frequent. Step 6 is utilized after boundary only increment of count already appeared candidates. Steps 8 to 10 are similar like FP-Growth.

Thus with the help of confirmation boundary found out items which are going to appear in frequent itemsets and helps to reduce the number of items to store at the scanning time and thus reducing storage space which is more explained with the given dataset example of 10 transactions as shown in following Table 1.

For the given sample dataset, here, explanation is provided for two cases for the analysis purpose: 1) Minimum Support greater than 50 and 2) Minimum Support less than 50.

**Table 1. Sample Dataset**

| Tid | Itemset | Tid | Itemset |
|---|---|---|---|
| 1 | 1, 2, 3, 4, 5 | 6 | 1, 2, 6, 7, 10, 11 |
| 2 | 1, 3, 5, 8 | 7 | 1, 8, 10, 11 |
| 3 | 2, 3, 4, 5, 7 | 8 | 1, 2, 3, 6, 8, 10 |
| 4 | 1, 3, 5, 8 | 9 | 2, 6, 8, 12 |
| 5 | 1, 2, 3, 5 | 10 | 1, 3, 8, 9, 12, 13 |

**Case Analysis 1: Minimum support = 60%**
Here, calculating support count=6 and Boundary=5.

The transaction numbers from 1 to 5 are scanned at the beginning of the first pass and only {1, 2, 3, 4, 5, 7, 8} items are considered for memory allotment and for count updating operation. Memory is not allotted to the items {6, 9, 10, 11, 12, 13} as according the proposed approach they are lying beyond the boundary as they cannot become frequent after transaction 5. At the end of the pass, candidate items {4, 6, 7, 10, 11} will be removed due to less support count and only {1, 2, 3, 8} will be available as 1-frequent items out of total unique 13 items.

**Case Analysis 2: Minimum support = 40%**
Here, calculating support count=4 and Boundary=7.

The transactions numbered 1 to 7 are considered for candidate memory allocation becomes the boundary. So, here also few candidate items are not considered after the boundary. But, new candidate items {9, 12, 13} are not included in the candidate items as they cannot become frequent items from the remaining transactions. And candidate items {4, 6, 7, 10, 11} will be removed at the end of first pass as their support is less compare to given support Sup and they cannot be frequent. But the number of candidate items which not to consider further are very less compare to higher support items. Rather here, candidate items to be considered as frequent are more compare to higher support items but which is less than the Apriori first pass candidate items and making the execution time of this modification comparatively equal to the FP-Growth for low support which is shown in further section.

The dataset used here is the example dataset, showing that some items are not considered after boundary, which will be more effective when the dataset is larger. This boundary is helpful to reduce the execution time by reducing the storage space of candidate items and their counting operations.

# 4. EXPERIMENTAL ANALYSIS

To assess the performance of MFP-growth used two other pattern growth algorithms, FP-growth and CFP-Growth to mine frequent itemsets for various support thresholds on real and synthetic datasets downloaded from [17] as described in the Table 2 are tested on Intel i5 CPU @3.10 GHz with 4 GB RAM and 64-bit OS. Furthermore, both FP-growth and CFP-Growth were downloaded from http://www.philippe-fournier-viger.com/spmf/index.php and were compared with MFP-growth in Java to present the performance comparison as shown in Fig 3. Since the CFP-algorithm is used for the multiple support values, here set all items' supports as equal when executing them.

The experiments were performed for mainly for three types of tests: 1) Number of 1-candidate items generated at the header table generation in 1st pass, 2) The time taken to generate the sorted header table and 3) The affect of overall execution

time. Numbers of 1-candidate items are considered not the actual memory mapping, as the reduction in number of items is same as the memory occupied by the candidates. The numbers of 1-candidates generated through different experiment is shown in the Fig 2. And other execution time results are as shown in the following Fig 3.

**Table 2. Datasets**

| Dataset | No. of Transactions | Unique Items | Transaction Avg. Length |
|---|---|---|---|
| T25I10D10K | 10000 | 929 | 25 |
| BMS | 59601 | 497 | 2.42 |
| Sign | 730 | 119 | 23 |

The T25I10D10K dataset is sparse and containing approximately 10000 transactions. In this case, the planned approach MFP-Growth shows overall memory required to store 1-candidate items is constant and on average memory consumption is reduced to 6% compare to FP-Growth and CFP-Growth. The reduction in average execution time to generate header table of 1-frequent itemset compare to FP-Growth is reduced to 4% and in the case of CFP-Growth it is reduced to 92% for support value 10% to 90% respectively.

| Dataset | T25I10D10K | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Support% | 90 | 80 | 70 | 60 | 50 | 40 | 30 | 20 | 10 |
| FP-Growth | 929 | 929 | 929 | 929 | 929 | 929 | 929 | 929 | 929 |
| CFP-Growth | 929 | 929 | 929 | 929 | 929 | 929 | 929 | 929 | 929 |
| MFP-Growth | *746* | *838* | *858* | *877* | *889* | *897* | *910* | *916* | *920* |
| Dataset | BMS | | | | | | | | |
| Support% | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| FP-Growth | 497 | 497 | 497 | 497 | 497 | 497 | 497 | 497 | 497 |
| CFP-Growth | 497 | 497 | 497 | 497 | 497 | 497 | 497 | 497 | 497 |
| MFP-Growth | *418* | *418* | *419* | *419* | *419* | *419* | *419* | *419* | *492* |
| Dataset | Sign | | | | | | | | |
| Support% | 90 | 80 | 70 | 60 | 50 | 40 | 30 | 20 | 10 |
| FP-Growth | 267 | 267 | 267 | 267 | 267 | 267 | 267 | 267 | 267 |
| CFP-Growth | 267 | 267 | 267 | 267 | 267 | 267 | 267 | 267 | 267 |
| MFP-Growth | *189* | *217* | *230* | *246* | *251* | *255* | *265* | 267 | 267 |

**Fig 2: No. of 1-Candidates**

As shown in Fig. 2 average execution time is reduced to 4% compare to FP-Growth and 87% from CFP-Growth and which may be even reduced more with more number of transactions.

More time is achieved in the CFP-Growth algorithm as it is generating the header table and MIS-tree together in one database scan only and later on approach is just adjusting the MIS-tree.

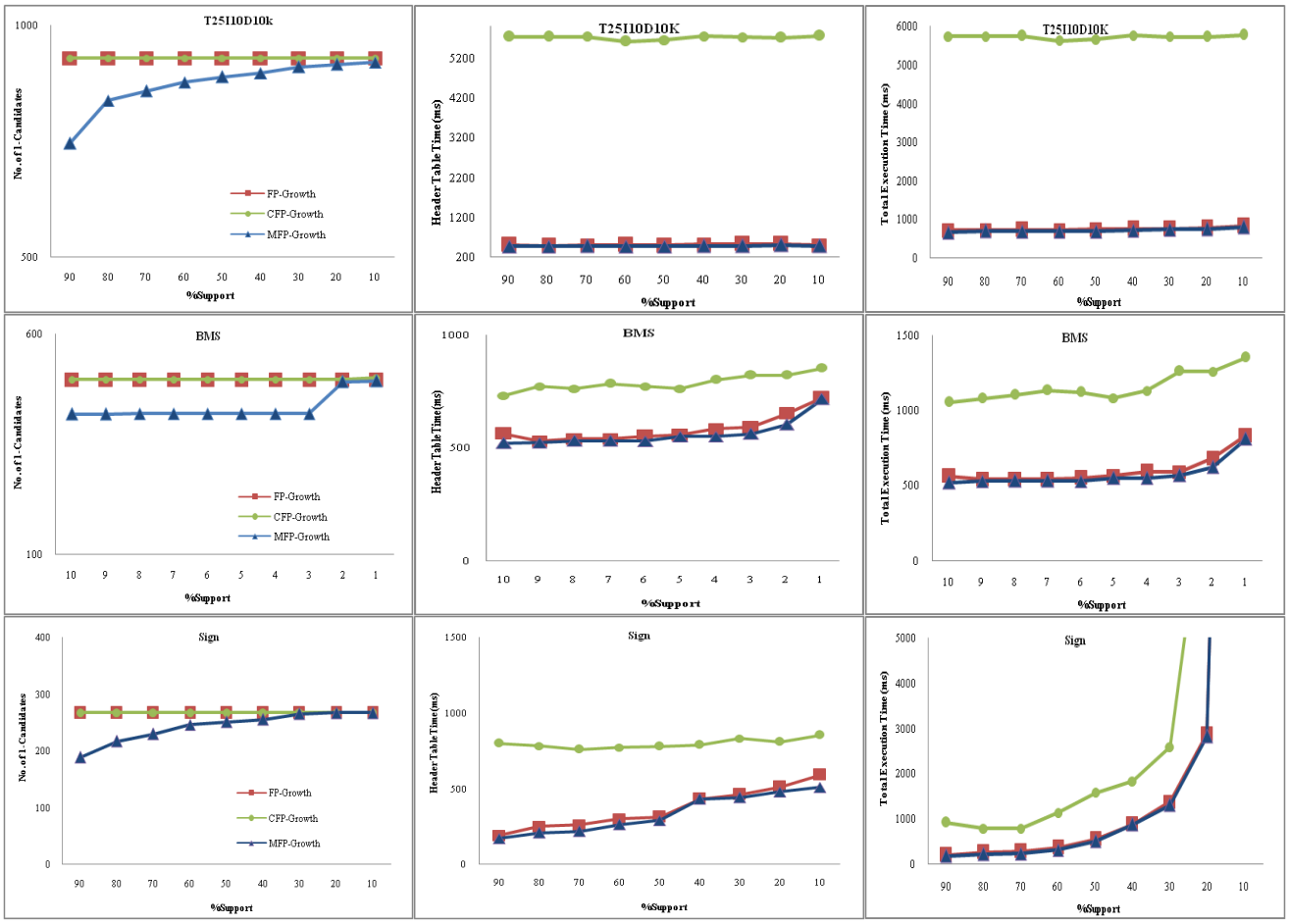


**Fig 3: Experiment Graphical Results**

For the BMS dataset, planned approach MFP-Growth shows overall memory required to store 1-candidate items is almost constant and on average memory consumption is reduced to 13% compare to both FP-Growth and CFP-Growth. But, here as 59,601 transactions are there the reduction in execution time to generate header table of 1-frequent itemset in average is 4% compare to FP-Growth and 29% compare to CFP-Growth for support value 1% to 10% respectively. As shown in Fig. 2 average overall execution time is reduced to approx. 4% from FP-Growth while in CFP-Growth it is 50%, which may be even reduced more with more number of transactions.

In the case of Sign dataset proposed approach MFP-Growth compare to FP-growth for different supports ranging from

10% to 90%, overall memory required to store 1-candidate items and average header table generation time both are reduced to 9%. This is because of all the distinct items are playing their role from 30% support and thus the average overall execution time is reduced to 1% as shown in Fig. 1. And compare to CFP-Growth memory required to store 1-candidate items, average header table generation time and overall execution time is reduced to 9%, 58% and 71% respectively.

In general, the MFP-Growth approach achieved the overall memory reduction compare to FP-Growth and CFP-Growth in all the datasets for different support thresholds. And as additional benefits the reduction in execution time is also achieved.

## 5. CONCLUSION

From the literature study, found the various approaches of pattern growth approach and through experiments, it is observed that in FP-Growth approach the memory consumption is constant but proposed approach is novel, it helps to reduce the memory utilization and thus also it helps to decrease the overall execution time. Cutback in memory is achieved by ignoring the non frequent items at early stage with proposing the boundary and in future it will be tested with other approaches of pattern growth and still to reduce execution time together with the integration of this concept to test at the recursive mining of trees.

## 6. REFERENCES

[1] J. Han and M. Kamber, "Data Mining Concepts and Techniques", Second Edition, Morgan Kaufmann Publishers, 2006.

[2] R. Agrawal, T. Imielinski and Swami, "Mining association rules between sets of items in large databases," In Proceeding of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207-216, 1993.

[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," Proceedings of 20th International Conference on Very Large Data Bases, pp. 487-499, 1994.

[4] J. Han, J. Pei and Y. Yin, "Mininin frequent patterns without candidate generation," Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 53-87, 2000.

[5] Pietracaprina and D. Zandolin, "Mining frequent itemsets using patricia tries", In B. Goethals and M. J. Zaki, editors, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Melbourne, FL, USA, November 2003.

[6] F. C. Tseng and C. C. Hsu, "Generating frequent patterns with the frequent pattern list," Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 376-386, 2001.

[7] K. Wang, L. Tang, J. Han and J. Liu, "Top Down FP-Growth for Association Rule Mining", PAKDD 2002, LNAI 2336, Springer-Verlag Berlin, pp. 334-340, 2002.

[8] M. El-Hajj and O. R. Zaıane, "COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation", In Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA, CEUR Workshop Proceedings, vol. 90, pp. 1-10, 19 December 2003.

[9] G. Liu. , H. Lu, J. Xu Yu, W. Wang and X. Xiao, "AFOPT: An Efficient Implementation of Pattern Growth Approach*", In Proceedings of IEEE ICDM'03 Workshop, FIMI'03, Melbourne, pp. 1-10, 2003.

[10] Y. G. Sucahyo and R. P. Gopalan, "CT-PRO: A Bottom Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tree Data Structure", In proceedings of IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI), Brighton UK, pp. 1-11, 2004.

[11] S. Xuequn, S. Kai-Uwe and G. Ingolf, "SQL Based Frequent Pattern Mining with FP-Growth", In Proceedings of 15th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2004, and 18th Workshop on Logic Programming, WLP, Germany, pp. 32-46, March 4-6, 2004.

[12] T. Hong, C. Lin and Y. Wu, "A fast updated frequent pattern tree", 2006 IEEE International Conference on Systems, Man and Cybernetics, Taiwan, 2006.

[13] Y. Hu and Y. Chen, "Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism", Decision Support Systems, Vol. 42(1), pp. 1-24, 2006.

[14] S. K. Tanbeer, C. F. Ahmed, B. Jeong and Y. Lee, "CP-Tree: A Tree Structure for Single-Pass Frequent Pattern Mining", PAKDD 2008, Springer LNAI 5012 - Verlag Berlin Heidelberg 2008, pp. 1022-1027, 2008.

[15] G. Pradeepini and S. Jyothi, "Tree-Based Incremental Association Rule Mining without Candidate Itemset Generation", In Trendz in Information Science & Computing (TISC), Chennai, pp. 78-81, Dec, 2010.

[16] K. Lin, I. Liao and Z. Chen, "An improved frequent pattern growth method for mining association rules", Expert Systems with Applications, Vol. 38, pp. 5154-5161, 2011.

[17] SPMF, A Sequential Pattern Mining Framework Available at: http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php