A SURVEY ON INCREMENTAL PATTERN MINING AND THEIR TECHNIQUES

[1]J.Dhakshayani, [2]Dr.S.Sivasathya, [3]S.Sharmiladevi, [4]S.LourduMarie Sophie
[1]Student, [2]Associate Professor, [3]Research Scholar, [4]Student
[1]Department of Computer Science,
[1]Pondicherry University, Pondicherry, India

Abstract: Finding frequent patterns are the essential stuff in data mining and knowledge discovery. A huge part of the existing data mining techniques is static in nature, while the real-world databases are dynamic and keep increasing its content in the form of transaction. The static traditional data mining techniques suffer from obsolete results in the long run as dynamic forms are not considered for the mining process. The domain of incremental mining is becoming more alive and growing as newer algorithms are developed day by day. This paper deals about the existing algorithms working on incremental data mining, with respect to Apriori and Tree based approaches.

***IndexTerms* - Incremental Mining, Frequent Patterns, Apriori-based approach, Tree-based Approach.**

I. INTRODUCTION

In the recent days, data mining is serving as a powerful tool in extracting meaningful data from the vast arena of database. Data analysis is used to solve the issues related to the decision-making process involving large datasets by building relationship and identifying patterns within them. The identified patterns occur frequently in nature, which may be of particular interest. The frequent patterns are substructures or itemsets that emerge in a dataset with frequency value greater than the threshold level specified by the user, and mining of the above said pattern is termed as frequent pattern mining.

Majority of the studies deal with high utility pattern mining that is concerned about the static database. However, the highly dynamic real-world databases make the existing methods unfit for processing as they are efficient in the way that they work through initial pruning process which can cause pattern losses. They also lose efficiency whenever any changes occur to the original database. The re-application of algorithms can solve the problems of the static approaches yet it is highly time consuming and CPU power utilizing. Thus, it is apt to switch to incremental mining approaches that can process only the incremental part to find the frequent patterns, when the new data is dynamically added to the initial database.

The incremental data mining falls under two classes viz., the Apriori-based and the Tree-based algorithms, which have been studied extensively and discussed in this paper.

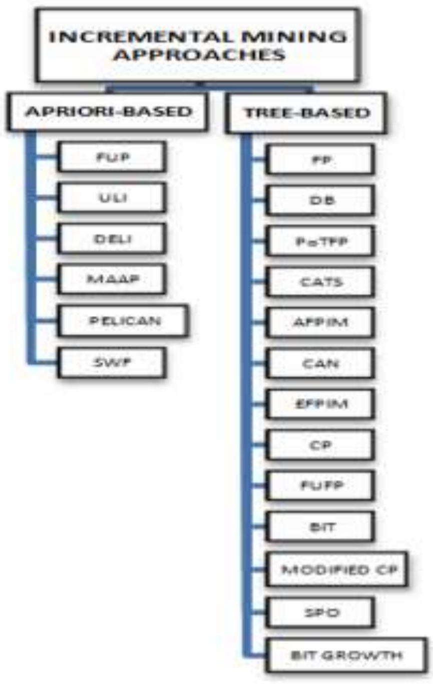

fig 1. Classification of Incremental Mining Approaches

This paper is structured as follows: Section II discuss the incremental mining application and its needs, Section III deals with the approaches for incremental mining followed by conclusion in Section IV and the references in Section V.

II. INCREMENTAL DATA MINING – APPLICATION AND NEEDS

The recent applications of today call upon the necessity of incremental mining wherein, the probable reason is the increased utilization of the record-based databases that includes continuous addition of data. For instance, the cases of stock market analysis, transaction in e-commerce, regular weather/traffic reports, grocery sales data and web logs. The prime aim of all these applications is to extract the transaction from the database for the recent data. The incremental mining directs one to add not only new data but also to remove the obsolete data. Incremental mining that deals with dynamic database has been very beneficial in several applications like decision making, selective commerce and business management, web data analysis, knowing customer trends etc.

III. INCREMENTAL MINING APPROACHES

Incremental mining approaches are of two types. This section deals with the technique under Apriori-based and tree-based approaches.

3.1Apriori-based approaches

The Apriori-based approaches work with two main objectives viz., to find all the frequent items based on the threshold minimum support value which incorporate a significant number of transactions and to produce all rules associated with the frequent itemsets. This is based on another threshold value called minimum confidence. The Apriori-based approaches and the Direct Hashing and Pruning (DHP) algorithms are the two superior algorithms which execute many iterations and find the frequent itemsets.
There are many algorithms/ techniques which can be categorized under this type. They are briefly summarized here.

3.1.1 FUP- Fast Update

The work carried out by Cheung et al [1] proposed an incremental updating technique for maintaining discovered associations rules efficiently whenever new incoming transaction data are added to the transaction database. They came up with FUP (Fast Update algorithm) to compute the frequent itemsets from the new updated database as well as a technique to reduce the database size during updation. The FUP performance was compared with that of Apriori and DHP and found that the former is two to sixteen times faster than the latter. FUP deals only with the insertion part, an improved version called FUP2 was proposed which deals with both insertion as well as deletion of transactions. The advantage of this algorithm is that the cost of finding new rules in the new database is reduced by using the earlier mined results.

3.1.2 ULI- Update Large Itemsets

Cheung et al [1] in his work does only the addition of new transactions to the database for updating the frequent itemsets while, Thomas et al [2] proposed an incremental updating technique called ULI (Update Large Itemsets) that involves either addition or deletion of transaction from a database. Besides the large itemsets, the negative borders were also maintained. Itemsets in the border that are not frequent are said to be negative borders. This can be accomplished by repeating the join and prune steps of the Apriori-gen function in the Apriori algorithm and they are used to determine when to perform scanning of database as well as can be used in any algorithm such as Apriori or partition. The special feature of the algorithm is that it requires complete scan of the full database, only if the database update cause negative border of all frequent itemsets to expand. The comparison study of both FUP and ULI revealed that the size of the candidate set could be potentially be much smaller than that of FUP.

3.1.3 DELI- Difference Estimation for Large Itemsets

An algorithm was devised by Lee et al [3] which is capable of evaluating the differences in database before and after updation in terms of association rules and this estimate indicate the need of the update, i.e. larger the difference, then it is time to update. The earlier algorithms such as FUP and FUP2 are used in the updation process of database, yet they face a performance overhead in the long run since they are applied too frequently. It is mandatory to use the FUP and FUP2 algorithms at suitable times for better results. This challenge was met out by a new algorithm DELI (Difference Estimation for Large Itemsets) that yields a difference estimate which in turns gives an indication to apply the FUP2 algorithm to database. The major function of this technique is to first find the amount of changes that has occurred by the updation process. If those changes are not significant, then it is necessary to ignore and wait for more changes in order to apply the FUP2 algorithm.

3.1.4 UWEP- Update With Early Pruning

Ayan et al [4] put forward an algorithm termed Updated With Early Pruning (UWEP), an enhanced FUP2. The merit of UWEP is that, the scanning of existing database occurs no more than once and new database just once. UWEP also utilizes a dynamic look-ahead strategy for identifying and eliminating items that become smaller once the updation of the new set of transaction is complete.
Steps involved in UWEP:

1. Support value of the itemsets are counted and a ‘tidlist’ is created for each of the I- itemset in the old database.
2. Check the frequent itemsets in the new database whose items are not in the old database and their supersets in updated database.
3. Check the frequent itemsets in the old database for largeness in the updated database.
4. Check the frequent itemsets in new database which are not considered in the old database for largeness in the updated database.
5. Generate the candidate list of itemsets from the obtained list of frequent itemsets from the previous step.

3.1.5 MAAP- Maintaining Association rules with Apriori Property

The study made by Ezeife et al [5] presented an algorithm for Maintaining Association rules with Apriori Property (MAAP) which is efficient in maintaining the discovered association rules in the new updated transactional database, in order to determine the highly frequent itemsets in the new database with the help of available highly frequent itemsets in the old database.

Steps involved in MAAP:

1. Compute new itemsets which are frequent in old database.
2. Compute another itemsets which are frequent in old database and not considered in step 1. This is because they are smaller superset itemset but still can be large in the new database.
3. Compute the remaining itemsets present in the candidate list that can be frequent in the new updated database.
4. Adjust candidate sets so that the new large itemsets can be added which are previously small in the old database.

This approach becomes advantageous as some low level frequent itemsets are not generated thus it eliminates the overhead. The superiority of MAAP over the FUP2 is that MAAP doesn't compute some frequent itemsets that have the longest list of candidates itemsets.

3.1.6 PELICAN

Veloso et al [6] proposed an algorithm names PELICAN that outperforms most of the existing algorithm by handling all the five knowledge managements such as reuse of previous knowledge, updation of knowledge base, tolerance to noisy input, management of resource usage and discarding of obsolete knowledge, while others only the first two issues. For instance, PELICAN does the scanning of database only when needed. The main objective of PELICAN is to minimize the number of candidates examined to decide whether they are frequent.

ECLAT algorithm:

The proposed PELICAN algorithm is based on ECLAT algorithm is used to determine the frequent itemsets with two features namely vertical database format and lattice decomposition. Vertical format is used to maintain an ordered list of transaction while lattice decompositions are used to divide the search space into sub-lattices and further smaller. Thus, the itemsets are determined by bottom up search along with merging the tidlist.

3.1.7 SWF-Sliding Window Filtering

FUP-based algorithms are high in I/O cost since they require multiple scans of database. To overcome this menace, a new algorithm termed Sliding-Window Filtering (SWF) was proposed by Lee et al [7] which involves the fragmentation of a database into many parts, employing a filtering threshold method in each part in order to handle the candidate itemset generation. These candidate itemsets are generated during partition process carried out by SWF. Thus, the output of this SWF called cumulative jilter CF consists of a successive candidate list of itemsets, their event counts and the respected partial support. Thus, the execution time of SWF is smaller than those needed by other algorithms and also efficiently controls memory utilization.

3.2 Tree-based approaches

Though Apriori-based approach remains to be the oldest technique in the field of pattern mining, and also being efficient, they are likely to be cumbersome due to numerous candidate patterns generation, time complexity and more database scans. To remediate these problems, Han et al [8] came up with the frequent pattern tree (FP- tree) that is capable of scanning the database only twice and cut the necessity of candidate itemsets generation.

FP tree representation:

A FP-tree is a compressed data structure that represents the data set as a tree design. Each item in a transaction is read and then mapped as FP-tree. The Fig 2 is an example of a best-case scenario that occurs when all transactions have exactly the same itemsets and a single branch of nodes denotes the size of FP tree. for instance, A:5 denotes the item A with its support value 5.

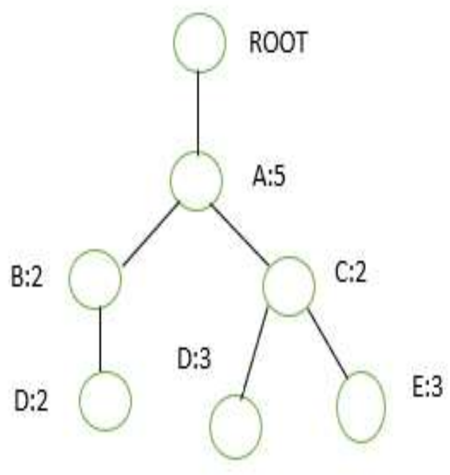

fig 2. Best-case scenario that occurs when all transactions have exactly the same itemset

FP tree construction:

The FP construction is divided into three major steps.

1. Scan the transactions to calculate the support count for each item, eliminate the infrequent items and sort the frequent items in decreasing order.
2. Scan the database for each transaction one at a time to build the tree and create a root node as null

For each transaction:

a. Select and sort the frequent itemsets in the descending order and call the function for insertion.
b. If the transaction is unique then form a new path and set the counter for each node to 1.
c. If a common itemset are shared then increment the common itemset node counters and construct new nodes if necessary.

3. Continue until each transaction has been mapped into the tree.

There are many algorithms/ techniques which can be categorized under this type. They are briefly summarized here.

3.2.1 DB and PotFP

Two algorithms namely DB-tree and PotFP-tree, that use the FP-tree as a basic structure in order to reduce the number of database scans were proposed by Ezeife et al [9]. Both the algorithms aim at efficient mining of the association rules thereby eliminating the need to re-scan the entire new database and reconstruct the FP-tree. The salient feature of DB-tree is that it keeps all the information regarding the database in the form of an FP-tree structure and doesn't require the original database to be scanned and that of the PotFP-tree is that it reduces the scanning level using a prediction of available frequent itemsets to which the original database is to be subjected. It was concluded that the performance of this proposed algorithm was superior over that of the FP and Apriori algorithms with different support values.

3.2.2 CATS-Compressed and Arranged Transaction Sequences

A tree structure improvised over the FP-tree, called the CATS tree which means Compressed and Arranged Transaction Sequences along with an enhanced storage compression that allows mining without candidate generation was developed by Cheung et al [10]. The objective of CATS-tree is to generate frequent patterns without any candidate itemsets. Thus, it employs an algorithm FELINE that fragments the growth method. In contrast to the FP-tree, once the CATS-tree is built, the mining of frequent patterns can be repeated without the help of re-building the entire tree. However, this is conscious to the order and the items of transaction.

3.2.3 AFPIM- Adjusting FP-tree for Incremental Mining

Koh et al [11] designed a strategy to maintain the association rules that are based on the adjustments in FP-tree structure. A dynamic algorithm termed Adjusting FP-tree for Incremental Mining (AFPIM) was proposed for supporting the frequent itemsets. Earlier another algorithm called DUP algorithm was framed by Lee et al [12], which also focus on solving the problems in maintaining the association rules but it suffered from the storage cost due to huge database. Meanwhile, AFPIM algorithm enables the FP-tree structure accomplished by modifying the existing FP-tree and then the frequent itemsets are excavated from the new FP-tree structure and their respective association rules are also identified.

3.2.4 CAN-Canonical Order

Though new algorithms such as FELINE and AFPIM were framed to overcome the existing problems, they also had some weakness. For instance, FELINE algorithm needs huge computation to search frequent items and mergeable paths during CATS tree construction. Thus, a new tree structure called CAN-tree (Canonical Order) was proposed by Leung et al [13], where the items are already organized based on some canonical form that is insensitive to item frequency, thus making the above searches convenient. Similarly, the AFPIM algorithm needs an added mining constant called 'preMinsup', whose amount estimation is critical and challenging while, the construction of CAN-tree is not dependent on the threshold values 'preMinsup'.

3.2.5 EFPIM-Extending FP-tree for Incremental Mining

The AFPIM plays a role in identifying new frequent itemsets on adjusting FP-tree structure, however adjusting FP-tree of the original database is highly expensive. On considering this problem. Li et al [14] proposed an algorithm, named as EFPIM (Extending FP-tree for Incremental Mining) that employs EFP-tree, a similar variant of FP-tree.

EFP-TREE:

EFP-tree is a structure that is similar to FP-tree structure in the execution of the algorithm at initial level. Items that are not frequent in the updated database are removed after database is updated, then they decrease the EFP-tree with both old and updated transactions, which are sorted in the form of pre-frequent itemsets. This tree is considered as a prefix-tree structure for arranging an essential transaction with no data loss but not as constrictive as FP-tree.

This algorithm also focusses on efficient finding of frequent large itemsets with minimal re-computation. Two forms of EFPIM algorithm viz., EFPIM1, a simple version to execute and EFPIM2, a fast algorithm, which mine frequent items on the basis of EFP-tree have been implemented.

3.2.6 CP- Compact Pattern

CAN-tree had reached a familiar status as it employs only one database scan, yet the drawback is that it possesses poor mining performance. In order to remedy this issue, Tanbeer et al [15] came up with a new structure, namely the CP-tree (Compact Pattern tree), which is similar to FP-tree as it works through efficient tree restructuring. The CP-tree consist of two phases viz., insertion phase, in which current item orders are inserted and updated in the restructuring phase, wherein the items are arranged according to the frequency and the tree is restricted through Branch Sorting Method (BSM). BSM is an array-based method which executes the process of restructuring node by node from the root. The rearrangement of items in the list results in a sorted list and then performs the restructuring mechanism on tree until all the nodes are processed. Thus, the final restructured tree is produced.

3.2.7 FUFP-Fast Updated FP

One of the demerits of FP-tree is that it is required to perform all transactions in batch manner. A try was made by Hong et al [16] which involves a modification of the batch method of FP-tree algorithm along with the FUP concept. A FUFP-tree, a similar structure to FP-tree, but the relation between the leaf and root are bidirectional. It also facilitates easy updation of tree. the salient feature of the FUFP-tree is that it generates a less concise tree, compared to that of the former as they do not follow any sorted order to build the tree.

Steps involved in FUFP:

1. Scan the current transaction to get all counts and items, also verify that those items are frequent enough by comparing with minimum count.
2. If the items are frequent both in current and original database.
   Set the count and update header table and insert the frequent items in a set of "to be inserted item" list.
3. If the items are small in new database but frequent in original database.
   Set the count and if count is less than the number of transactions in old and new database, then add the items to the updated database.
4. If the items are frequent in current database but less frequent in original database,
   Then rescan database to find frequent transactions
5. Sort and insert the items in decreasing order of their new counts
6. For each item in New/updated Itemset I, if I is not present at the respective level of the FUFP-tree, then insert I, set count as 1. Else increase the count of I by 1.
7. For each item in existing itemset J, if J is not present at the respective level of the FUFP-tree, then insert J and make count as 1. Else increase the count of J by 1

3.2.8 BIT-Batch Incremental Tree

The earlier algorithms such as AFPIM, CATS tree and CAN-tree performs mining by processing one transaction at a time and then do the updation. Totad et al [17] proposed an algorithm, named BIT (Batch Incremental tree) algorithm which is designed to merge two small consecutive FP-trees that are similar to the one obtained through processing the entire database. The distinct difference of the BIT algorithm from the others is that it is required to perform periodical mining of transaction database. BIT considers FP-tree as two periodical datasets, read the itemsets of FP-tree 1, searches for mergeable prefix path of FP-tree 2 and merges the former with latter.

3.2.9 Modified CP

The CP-tree, though efficient, is time consuming as regards to tree construction. To overcome this wastage in time, Vishnu et al [18] framed a modified CP-tree structure by constructing the prefix tree with dynamic rearrangements of the nodes in the old tree, based on the sorted list. The tree is built, though the items are non-frequent. It takes less mining time which is an advantage.

3.2.10 SPO-Single Pass Ordered

Koh et al [19] proposed an algorithm called SPO-tree (Single Pass Ordered tree) which retrieves information by scanning the tree only once. The special character of this algorithm is that it makes transaction of all inserted/ sorted items, on the basis of their frequency, where the re-scan of database is not necessary. The SPO-tree consist of two phases namely tree construction phase, which is further sub divided into two phases, the insertion and reorganization phase and the tree mining phase, which uses the FP-growth technique. It has been proved that SPO-tree is faster than other tree by 7.4%, on average.

3.2.11 BIT_FPGrowth

Totad et al [20] proposed a modified form of BIT algorithm. The existing algorithms such as AFPIM, CATS, CP-tree and SPO-tree performs mining by processing one transaction at a time while, the proposed BIT_FPGrowth reconstructs and merges two small consecutive FP-trees for obtaining a tree using the FPGrowth algorithm. It employs FP-tree as pre-processed data repository to get itemsets. Here, as a modification from the former BIT algorithm, it employs the FP-Growth algorithm for constructing FP-tree. It is advantageous since multiple occurrence of itemsets are processed only once.

Thus, the comparison of major parameters for tree-based approach is depicted in fig 3.

PARAMETRS DISCUSSED	FP- TREE	DB-PotFP TREE	CATS TREE	AFPIM TREE	CAN TREE	EFPIM TREE	CP TREE	FUFP TREE	BIT TREE	MODIFIED CP TREE	SPO TREE	BIT_GROWTH TREE
TREE STRUCTURE	extended prefix-tree	FP tree	FP tree	Adjusted FP tree	Canonical order tree	EFP tree	Compact pattern tree	Fast updated FP tree	Batch incremental tree	Modified compact pattern tree	Single Pass Ordered Tree	Batch incremental tree
SUPPORT THRESOLD/MINIMUM SUPPORT (%)	1.5	0.1 - 6	0.15	0.5	0.05	3	5	4	0.01	10	0.10	0.25
BASIC METHOD/CONCEPT USED	partitioning-based, divide-and-conquer method	-	divide-and-conquer and fragment growth method	pre-large sequences	-	divide-and-conquer method	Path adjusting and branch sorting method	-	-	-	-	Pre-computation and data reduction
ADVANTAGE	Constructs high compact tree and avoids costly candidate generation	Eliminates the need of re-scan	Enhanced storage compression	Minimal re-computation	Insensitive to item frequency thus making the search convenient	Minimal re-computation	Efficient restructuring	Generates less concise tree	Performs periodic mining	Dynamic rearrangement of tree and less mining time	Takes information from a single scan	Multiple occurrence of itemsets are processed only once
DISADVANTAGE	Process all transaction in batch	-	required to swap and merge the nodes during the updates	Highly expensive	Poor mining performance compared to FP	Unchanged old arrangements of nodes	Time consuming regards to tree construction	Process one transaction at the time	-	-	Process one transaction at the time	-

fig 3. Comparison Table of Tree-Based Approach

IV. CONCLUSION

In the current global scenario, large sets od distributed data are being efficiently dealt by incremental data mining. The incremental data mining techniques for maintaining association rules are becoming enormous in number as the existing techniques needs to be improvised as they possess few drawbacks. The current survey studies these new techniques extensively for better understanding the reasons for the new updates and the necessity for the new techniques.

V. ACKNOWLEDGMENT

We express our sincere gratitude to Dr.S. Sivasathya for her guidance and kind motivation throughout the period of study.

REFERENCES

[1] Cheung, D. W., Han, J. H. J., Ng, V. T., & Wong, C. Y. (1996). Maintenance of discovered association rules in large databases: an incremental updating technique. Proceedings of the Twelfth International Conference on Data Engineering, 106–114. https://doi.org/10.1109/ICDE.1996.492094
[2] Thomas, S., & Ranka, S. (1997). An Efficient Algorithm for the Incremental Updation Rules in Large Databases of Association, 263–266.
[3] Lee, S. D., & Cheung, D. W. (1997). Maintenance of Discovered Association Rules: When to update? ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'97).

[4] Ayan, N. F., Tansel, A. U., & Arkun, E. (1999). An efficient algorithm to update large itemsets with early pruning. Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’99, (X), 287–291. https://doi.org/10.1145/312129.312252

[5] Ezeife, C. I., & Zhou, Z. (n.d.). A Low-Scan Incremental Association Rule Maintenance Method Based on the Apriori Property 1 Introduction.

[6] Veloso, A., Pôssas, B., Meira Jr, W., & others. (2001). Knowledge management in association rule mining. In Integrating Data Mining and Knowledge Management, Held in Conjunction with the 2001 IEE International Conference on Data Mining (ICDM. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.63.9197

[7] Lee, C. H., Lin, C. R., & Chen, M. S. (2001). Sliding-window filtering: An efficient algorithm for incremental mining. In Proceedings of the tenth international conference on information and knowledge management (pp. 263–270). Nov. 2001.

[8] Han, J., Pei, J., & Yin, Y. (2000). Frequent Pattern Tree: Design and Construction. Networks, 1–12. https://doi.org/10.1145/342009.335372

[9] Ezeife, C. I., & Su, Y. (2002). Mining incremental association rules with generalized FP-tree. Advances in Artificial Intelligence, 2, 147–160. https://doi.org/10.1007/3-540-47922-8_13

[10] Cheung, W., & Zaïane, O. R. (2003). Incremental mining of frequent patterns without candidate generation or support constraint. Proceedings of the International Database Engineering and Applications Symposium, IDEAS, 111–116. https://doi.org/10.1109/IDEAS.2003.1214917

[11] Koh, J. L., & Shieh, S. F. (2004). An Efficient Approach for Maintaining Association Rules Based on Adjusting FP-Tree Structures*. Database Systems for Advanced Applications, (92), 221–227. Retrieved from http://www.springerlink.com/index/GC9V53KY5CBB83TV.pdf

[12] G. Lee, K.L. Lee and A.L.P. Chen. (2001), An efficient Graph-Based Algorithms for Discovering and Maintaining Association Rules in Large Databases. Knowledge and Information Systems, Springer-Verlag, Vol. 3, pages 338-355**.**

[13] Leung, C. K.-S., Khan, Q. I., & Hoque, T. (2005). CanTree: A tree structure for efficient incremental mining of frequent patterns. Proceedings - IEEE International Conference on Data Mining, ICDM, 274–281. https://doi.org/10.1109/ICDM.2005.38

[14] Li, X., Deng, X., & Tang, S. (2006). A fast algorithm for maintenance of association rules in incremental databases: vol. 4093. Advanced Data Mining and Applications (pp. 56–63). Heidelberg: Springer LNCS (LNAI).

[15] Tanbeer, S. K., Ahmed, C. F., Jeong, B. S., & Lee, Y. K. (2009). Efficient single-pass frequent pattern mining using a prefix-tree. Information Sciences, 179(5), 559–583. https://doi.org/10.1016/j.ins.2008.10.027

[16] Hong, T. P., Chen, H. Y., Lin, C. W., & Li, S. T. (2008). Incrementally fast updated sequential pattern trees. Proceedings of the 7th International Conference on Machine Learning and Cybernetics, ICMLC, 7, 3991–3996. https://doi.org/10.1109/ICMLC.2008.4621100

[17] Totad, S. G., R. B., G., & Reddy, P. P. (2010). Batch Processing for Incremental FP-tree Construction. International Journal of Computer Applications, 5(5), 28–32. https://doi.org/10.5120/910-1288

[18] Vishnu Priya, R., Vadivel, A., & Thakur, R. S. (2010). Frequent pattern mining using modified CP-tree for knowledge discovery. In LNCS: 6440 (pp. 254–261). Berlin, Heidelberg: Springer. 2010. doi: 10.1007/978- 3- 642- 17316- 5- 24.

[19] Koh, Y. S., & Dobbie, G. (2011). SPO-tree: Efficient single pass ordered incremental pattern mining. In LNCS: 6862 (pp. 265–276). Berlin, Heidelberg: Springer. 2011. doi: 10.1007/978- 3- 642- 23544- 3- 20.

[20] Totad, S. G., Geeta, R. B., & Prasad Reddy, P. V. G. D. (2012). Batch incremental processing for FP-tree construction using FP-Growth algorithm. Knowledge and Information Systems, 33(2), 475–490. https://doi.org/10.1007/s10115-012-0514-9