International Journal of Information and Communication Technology Research

http://www.esjournals.org

An Efficient Prefix Tree for Incremental Frequent Pattern Mining

Mohadeseh Hamedanian, Mohammad Nadimi, Mohammad Naderi
Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Isfahan, Iran

ABSTRACT

In many applications, databases are frequently changed by insertions, deletions, and/or modifications of transactions. Consequently, the frequent patterns extracted from them must be updated. Researchers propose incremental mining to update frequent patterns efficiently instead of mining all frequent patterns from scratch. Although FP-tree is one of the most efficient algorithms for frequent pattern mining, it is not easily adoptable with incremental updating. Accordingly, the CP-tree and restructuring method of Branch-Sorting have been proposed for incremental mining of frequent pattern. This method consists of two main phases of insertion and restructuring. Since during construction of CP-tree items are sorted in descending order of previous insertion phase, then its restructuring can be very costly. To solve this weakness, in this paper a new efficient prefix tree structure has been proposed to reduce the time of restructuring. The proposed tree is created based on the frequency of last items and it requires just one database scan. The experimental results show that using the proposed tree and Branch-Sorting method can enhance the efficiency of incremental mining of frequent patterns from both dense and sparse datasets.

Keywords: *Frequent pattern mining, incremental updating, FP-tree, Branch-Sorting method*

1- INTRODUCTION

Since the introduction of the Apriori algorithm [1], frequent pattern mining plays an essential role in data mining research. Apriori algorithm applies an iterative approach in which k-itemsets are used to produce k+1-itemsets. In order to improving efficiency of this approach, an important attribute has been proposed, reducing searching space. On basis of this attribute, all of non-vacant subsets from a frequent itemsets should be frequent. This algorithm will need several database scans so that at worst, it equals to maximum frequent patterns length. In addition, much more memory volume is required in this algorithm because of generating a lot of candidates. To obviate two drawbacks existed within Apriori algorithm, a extended tree structure called FP-tree and an efficient algorithm for frequent patterns extraction known as FP-growth were developed by Han et al 2000 [2]. The goal of designing FP-tree was to capturing and compacting database content in a prefix tree structure. The FP-tree reduces the number of database scans from k to only two. In the first scan, the frequency of items existed within database transactions are calculated and frequent items in length of one are extracted. In the second scan, the frequent items existed within each transactions are sorted in descending order of frequency and inserted into FP-tree. Eventually, the FP-growth finds all frequent patterns from FP-tree without candidate generation. Although, since the introduction of FP-tree, many studies have been proposed to improve functionality and performance, there have been addressed a few contributions to solve incremental updating in FP-tree. In [3] a tree structure called CP-tree and a restructuring method known as Branch-Sorting was proposed to solve above problem. Their method consists of two main steps. In the first step or insertion step, CP-tree is constructed based on a standard order (alphabetic for example) and then items are inserted in a list called 1-list in the same order. Simultaneously to item insertion in tree, the number of their frequency is registered in 1-list and in case, deviation rate of items order in 1-list is much more than deviation threshold, tree was restructured by Branch-Sorting method. In the second step or restructuring step, CP-tree was restructured branch by branch. Since during construction of CP-tree items are sorted in descending order of previous insertion phase, this tree requires large restructuring. In this paper an efficient prefix tree structure has been proposed reduce the time of restructuring. According to this approach, items frequencies in previous transactions are used to insert each transaction in proposed tree. Then, this tree is restructured using method Branch-Sorting. Applying proposed tree will have led to increased efficiency of Branch-Sorting restructuring method.

Total frame of present paper is organized as following: in section 2 basic concepts and literature review are presented. The proposed tree is postulated in section 3. Then experimental results are illustrated in section 4. Finally concluding remarks will be stated in section 5.

2- PROBLEM DESCRIPTION AND RELATED WORK

In this section, key concepts and terms widely used in frequent patterns mining are explained. Let $I = \{i_1,\ldots, i_n\}$ be a set of N distinct items and DB be a database consist of M transactions $\{t_1, \ldots, t_m\}$ such that each transaction t_i is a subset of I ($t_i \subseteq I$). An itemset or pattern x is a subset of I which if $|x|=k$, it is called a k-itemset. One of the properties of x is its support count or Sup(x) which is the number of transactions in DB that contain the itemset x. If Sup(x) is no less than a user specified threshold, called Minsup, it is called a frequent pattern. The objective of frequent pattern mining is to discover all frequent patterns satisfying Minsup from a given database DB.

The first known proposed method for extracting frequent patterns is Apriori algorithm proposed by Agrawal et al [1]. There have been developed enormous modified versions to

improve it [4-10]. Since the main drawback for Apriori-based algorithms was involving multiple database scans and generation of a large number of candidates, it was not appropriate where scalability and efficiency are important. Then in 2000 an efficient method consisted of a well-defined tree structure and a frequent pattern extraction algorithm known as FP-growth put forwarded by Han et al [2]. In this approach the number of database scans was reduced from K to just two times without candidate generation. So far there have been conducted enormous studies on improving this algorithm efficiency.

In [11] a tree structure called CATS is proposed to mine frequent patterns in an incremental manner. The proposed tree structure improves FP-tree on data compression and allows extracting frequent patterns without the need to generate a set of candidate items. According to this method, the first transaction in database is added to the tree's root. For subsequent transactions, the items within the transaction are compared with the items in the tree for identify shared items. If there is any item in common between tree nodes and the transaction, the transaction is merged with the node that has the highest frequency level. Then, the remainder of the transaction is added to the merged nodes. This process is recursively repeated until all common items are discovered.

In [12] authors proposed the AFPIM algorithm for incremental mining. Similar to FP-tree, it only keeps frequent items. In this algorithm, a threshold called PreMinsup is considered whose values are set less than the Minsup. Since, items are ordered based on the number of events, the insertion, deletion or modification of transactions may affect the frequency and order of the items. More specifically, items in the tree are adjusted when the order of the items changes. The AFPIM algorithm swaps such items by applying bubble sort algorithm that involves huge calculation.

In [13] authors put forward a new tree structure called Can-Tree. Can-tree algorithm is used for incremental mining and needs only one database scan. According to Can-Tree algorithm, items are ordered on the basis of a canonical standard (e.g. alphabetical) which can be determined by the user. Therefore, any changes in frequency, which is caused by incremental updates (such as insert, delete, or modify transactions), will not affect the order of items in the Can-tree. Therefore, new transactions are inserted into the tree without swapping any tree nodes.

In [3] a new tree structure called CP-tree is put forward. CP-tree is a dynamic tree which can be used for interactive as well as incremental mining. In this method, all the transactions are inserted into the tree in accordance with a predefined item order. The item order of a CP-tree is maintained by a list, called I-list. After inserting some of the transactions, if the item order of the I-list differs from the current frequency-descending item order to a predefined degree, the CP-tree is restructured through a method called the branch sorting. Then, the item order is updated with the current list.

As reviewed the above, the most of efficient algorithms proposed for mining frequent patterns are based on FP-growth. However, some methods have proposed for mining frequent patterns in different fashion. In [14] a new method is proposed consist of a novel tree structure called PC-tree and a mining algorithm called PC-Miner. This method transfers each transaction into an integer number by using prime number characteristics. This transformation technique reduces the size of transaction database by which the content can be captured by PC-tree and kept in memory by only one database scan. Since content is represented by integer values, mining process can be done by using basic mathematical operations such as multiplication and division which enhance the performance of the mining process. Moreover, exploring PC-tree is based on prime factorization and an efficient set called Head set. A comprehensive experimental analysis conducted by several sparse and dense datasets verifies the efficiency of this method.

3- PROPOSED METHOD

As discussed in the previous sections, in order to solve incremental updating in FP-tree, in [3] a new tree structure known as CP-tree has been introduced by which database content are inserted in tree with one database scan. The construction of CP-tree is occurred in two main phases of insertion and restructuring. During insertion phase, items are sorted in descending order of frequencies associated to previous insertion phase and subsequently they are added to tree. This approach reduces the number of ordered paths present in CP-tree. In restructuring phase, the Branch-Sorting method removes all of unordered paths in tree, sort them into a temporary array and again insert sorted paths into the tree. Accordingly the efficiency of Branch-Sorting is directly related to tree orderings. For this, an effective technique has been presented in this paper to increase the number of ordered paths. Following, first that how proposed tree and CP-tree are constructed will be deal with and then we shed light on both tree restructuring based on Branch-Sorting method.

The rest of this section is devoted to describe proposed tree. Firstly, proposed tree and CP-tree are constructed for a transaction database DB shown in Table 1. Then, these trees are restructured by Branch-Sorting method.

3-1- Proposed Tree Construction

To increase the number of ordered present paths in tree and minimize restructuring time intervals, in this paper a new tree structure has been introduced. In this method in order to insert every transaction in proposed tree, items frequencies in all of previous transaction are applied. It leads to considering exact number of items events while inserting a transaction. For that reason, a list so called "LastSup" is applied to create a proposed tree. LastSup list involves the last items

frequencies present in transactions. At the first time of tree construction, all items frequency equals to zero. This list is updated by inserting every transaction.

Table 1: Transaction database DB

TID	Items
T_1	d, e, c
T_2	a, d
T_3	c, d, b
T_4	e, d, f
T_5	e, a, b
T_6	d, a, e

To illustrate proposed tree construction, let us to consider database incorporated in Table 1 which includes six transaction and every transaction involves subset of I={a, b, c, d, e, f}. First root node is created and labeled to Null value. LastSup is created as well, then all items get valued to zero.

Following root node and LastSup list construction, transaction insertion within trees begins. Before inserting each transaction, first items frequencies within which are added into LastSup list so that transaction of interest is ordered based on this list to be inserted in tree. In this approach items having the same frequency are ordered alphabetically. The details about proposed tree construction related to Table 1 are shown in Figure 1.

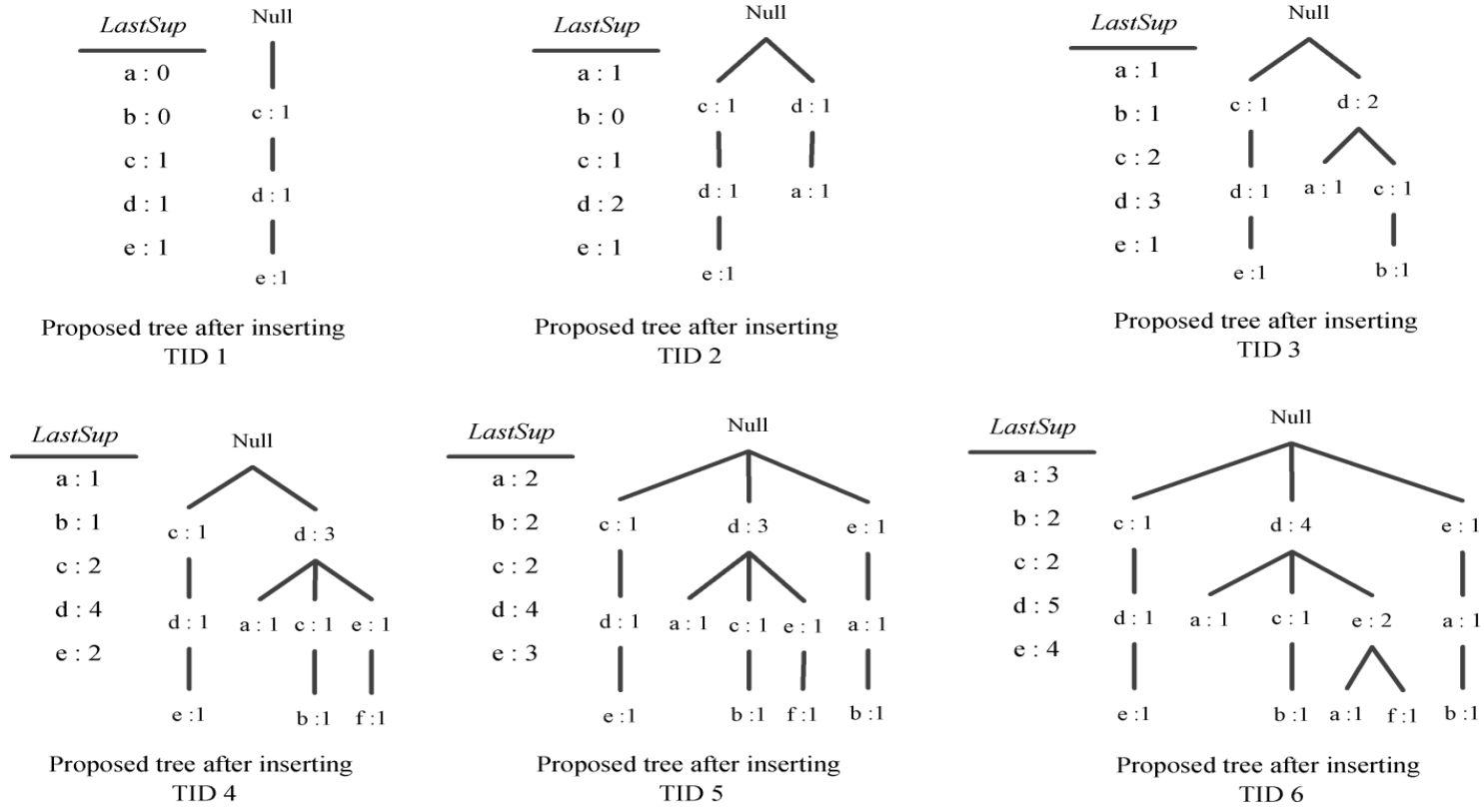
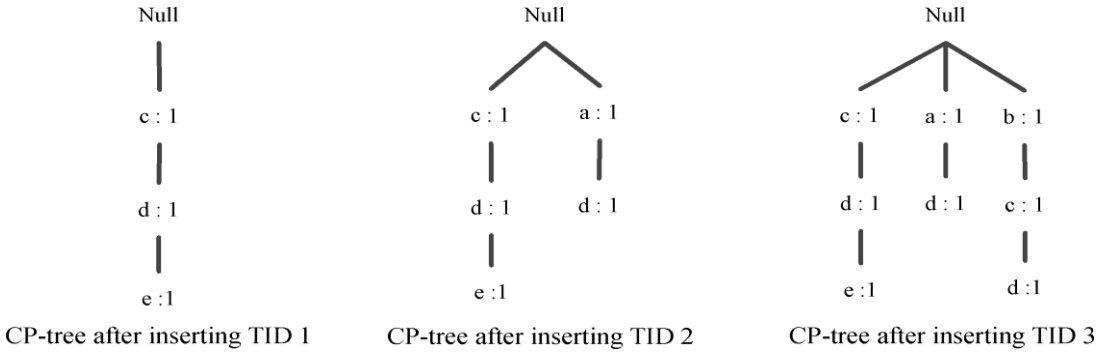


Figure 1: Step by step construction of the proposed tree

As in this paper, proposed tree structure is compared to CP-tree, construction this tree is discussed. In early phase of CP-tree construction (the first insertion phase), items are ordered in terms of standard orders (here, alphabetically) within 1-list. Simultaneously to items insertion in tree, their frequencies are captured in 1-list as well. In next phases (the second and more), items are sorted according to the order of previous phase and added to tree. Although using previous 1-list in part increases trees ordering degree, however after database updating, items orders subjects to many changes so many paths must be restructured.

Following CP-tree construction with two restructuring phases (having the three first and second transaction inserted) are discussed. First three first transactions are inserted into tree and at the same time items frequencies are added into 1-list. Phases related to insertion of these three transactions are shown in Figure 2. Then in restructuring phase, 1-list is

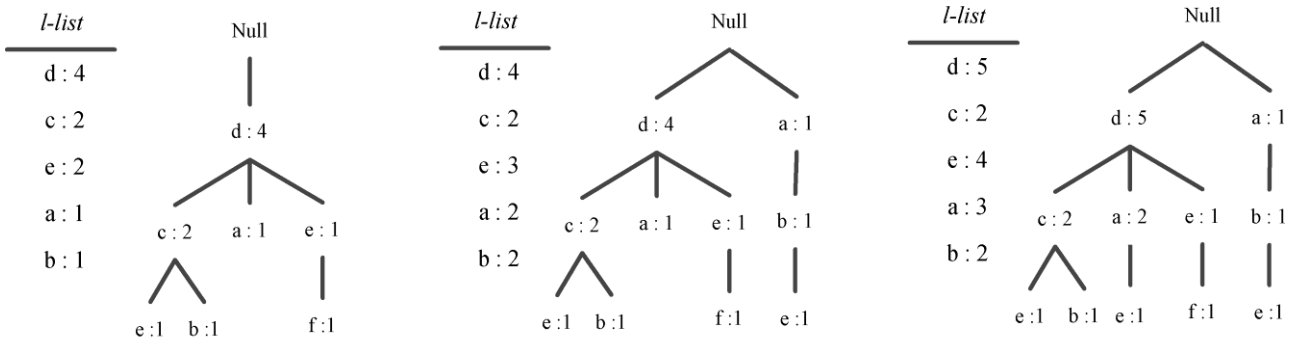
ordered in terms of items descending frequency and CP-tree is restructured based on this ordered list.



CP-tree after inserting TID 1 CP-tree after inserting TID 2 CP-tree after inserting TID 3

Figure 2: Step by step insertion TIDs 1-3 into CP-tree (first insertion phase)

In the second insertion phase, the forth to sixth transactions are added into CP-tree. In this phase, transactions are ordered based on previous ordered1-list to be inserted in tree. Results for present phase are shown in Figure 3.



CP-tree after inserting TID 4 CP-tree after inserting TID 5 CP-tree after inserting TID 6

Figure 3: Step by step insertion TIDs 4-6 into CP-tree (second insertion phase)

3-2- PROPOSED TREE RESTRUCTURING

As it was pointed out previously, if deviation rate of items order in 1-list is much more than deviation threshold, tree of interest is restructured. In Branch-Sorting method, all paths present in tree are evaluated and in case items would not ordered in terms of their descending frequencies, path of interest is removed from tree and having been ordered, it is inserted in tree again. So it is worthy to note that one of the effective factors in efficiency of Branch-Sorting method is ordering degree of associated tree. To evaluate this method, both proposed tree and CP-tree created in previous section are restructured based on Branch-Sorting restructuring method.

Table 2: Frequency of items for TID 1-3 of DB

Items	Frequency
d	3
c	2
a	1
b	1
e	1

Restructuring of proposed tree is shown in Figure 4. According to Branch-Sorting method, first paths present in this tree are identified to comparison of items orders in every path to Table 2. As it is illustrated in Figure 4, out of six path

existed in this tree, just there are two unordered ones and remnant are in ordered manner.

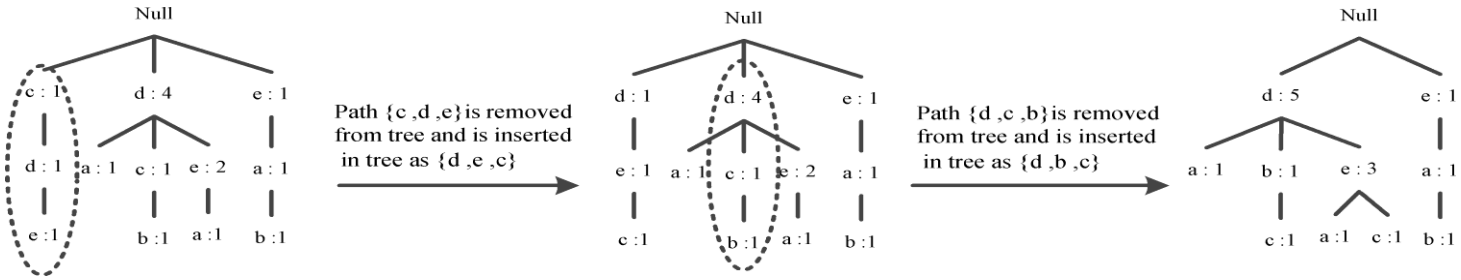


Figure 4: Step by step restructuring of the proposed tree

Consider CP-tree which constructed in previous section in two insertion phases. The first restructuring phase of this tree including first to third transactions is shown in Figure 5. In this phase, all paths existed in tree were unordered and restructured.

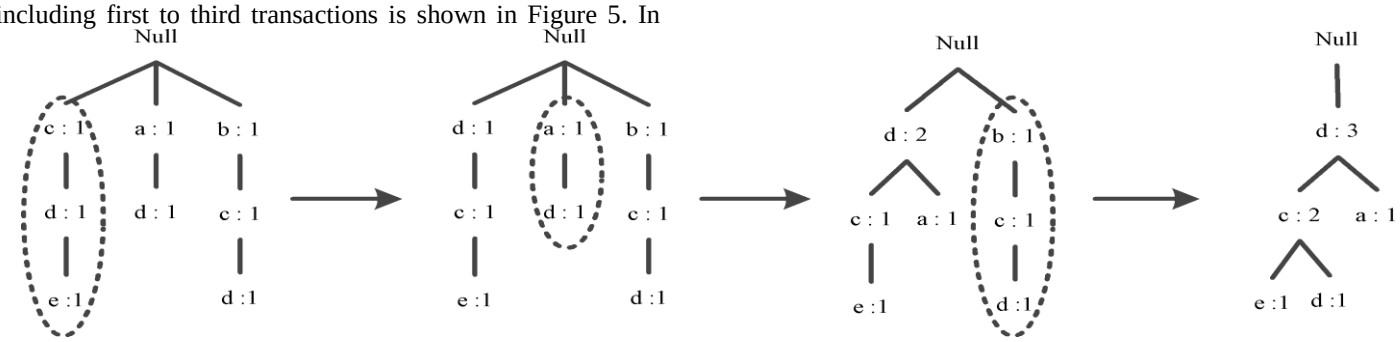


Figure 5: Step by step first phase restructuring the CP-tree

The second restructuring phase for CP-tree (from forth to sixth transactions) are observed in Figure 6. In spite of restructuring of all paths present in first phase, in the second phase out of six existed paths, four ones were unordered and restructured.

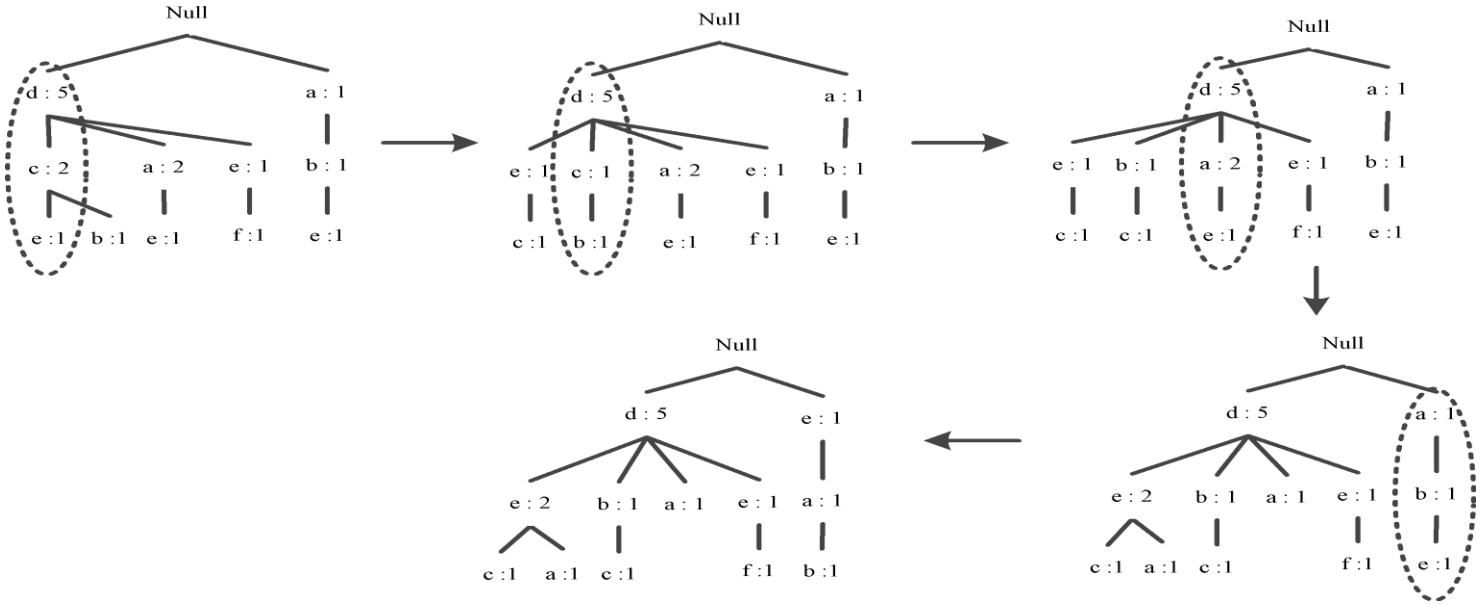


Figure 6: Step by step second phase restructuring the CP-tree

4- THE EXPERIMENTAL RESULTS

In this section, efficiency of proposed tree is evaluated. All of experiments are conduction windows XP with 2.7 GHz CPU and 3 GB memory. In order to evaluate efficiency of proposed tree, several real and synthetic datasets have been applied. Information related to these datasets is shown in Table 3.

Table 3: Datasets Properties

Dataset	# Item	# Trans	Dense/ Sparse	Trans Len	
				Ave	Max
Mushroom	119	8124	Dense	23	23
Chess	75	3916	Dense	37	37
T10I4D100K	870	100000	Sparse	10	29
Connect-4	129	67557	Dense	43	43
Pumsb*	2088	49046	Sparse	50	63
T40I10D100K	942	100000	Sparse	39	77

In the first experiment, time required to construct both proposed tree and CP-tree is compared. These experiments have been carried out on above datasets whose results are shown in Figure 7. Although proposed tree is a semi-ordered tree, however, time needed to its construction is approximately equals to that needed to CP-tree construction.

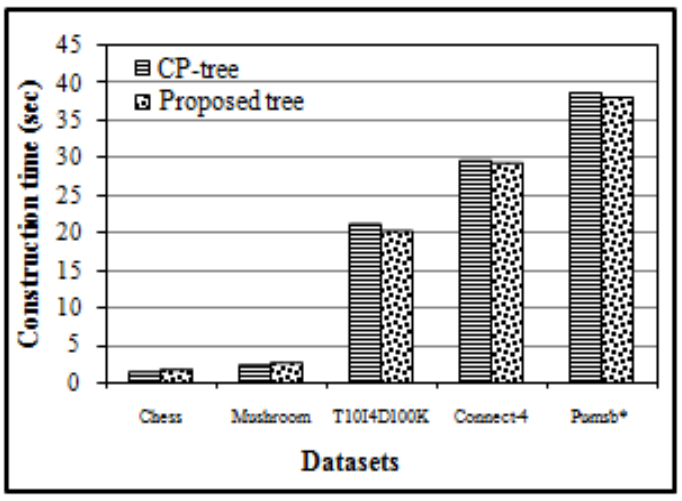


Figure 7: Construction time vs. dataset

In the second experiment, the numbers of ordered paths in created trees are evaluated in the first experiment. Results for these experiments are observed in Figure 8. According to these results, out of four sets of above five sets, 100% of paths are unordered. Subsequently all paths must be removed from tree and having been ordered, they are inserted in tree again.

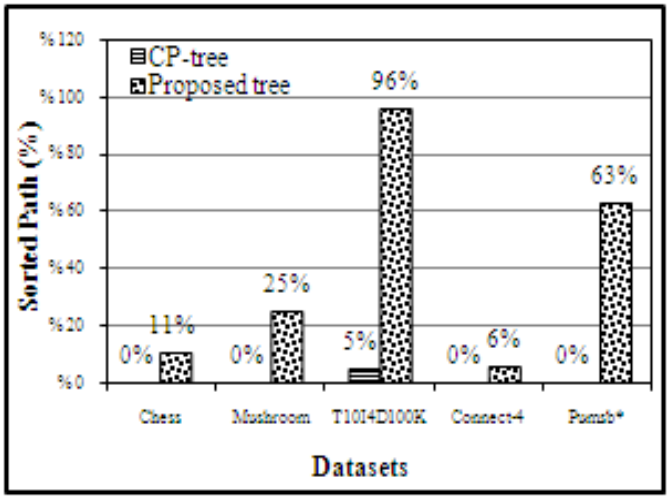


Figure 8: Percent of sorted path vs. dataset

In the third experiment, time needed for above tree restructuring by method Branch-Sorting is calculated. Results of these experiments (Figure 9) indicate that in every five dataset used in these experiments, proposed tree had higher efficiency than that of CP-tree.

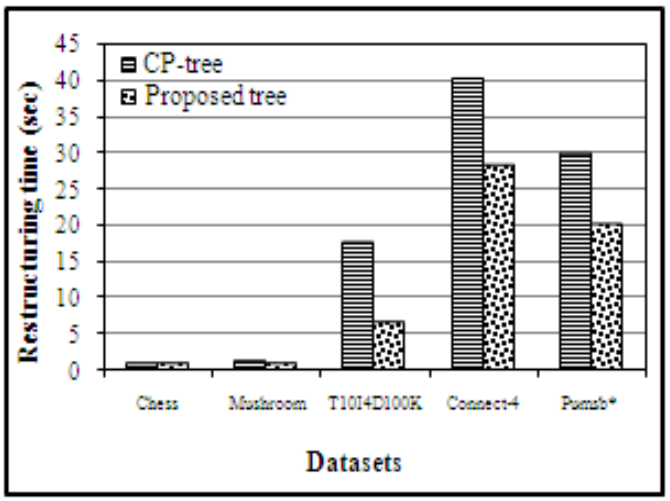


Figure 9: Restructuring time vs. dataset

In last experiment, datasets of T40I10D100K are used to evaluate proposed tree potential for incremental updating. For this, dataset were divided into several unequal parts and efficiency of suggested approach to updating by different parts of this dataset has been investigated. The experimental result is shown in Figure 10. Based on obtained results, proposed tree is more efficient than CP-tree in the incremental mining.

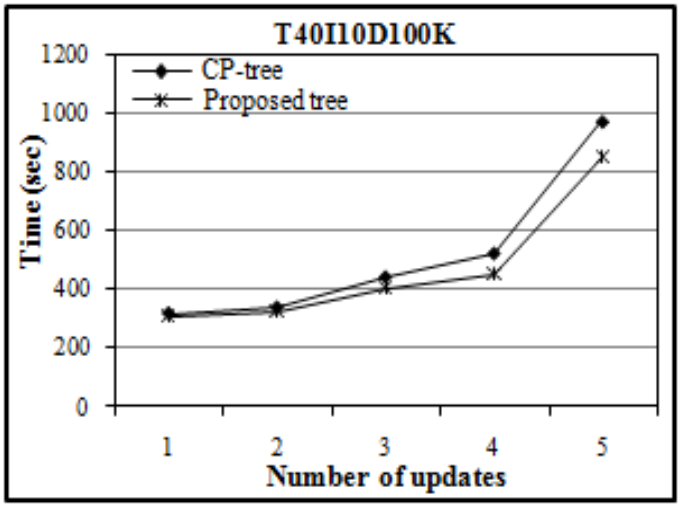


Figure 10: Incremental mining of the proposed tree (Minsup=1)

5- CONCLUSION

FP-growth algorithm using FP-tree structure serves as one of the famous algorithms in field of frequent patterns mining. As FP-tree structure has no updating potential, in case of insertion of new transactions or changing present transactions, FP-tree will be invalidated. In order to obviate and address incremental update of FP-tree, so far, several tree structures have been developed. CP-tree using Branch-Sorting restructuring method is one of the efficient proposed algorithms to solving incremental mining problem in FP-tree. Using previous 1-list to ordering transactions in process of CP-tree construction, increases time needed to its restructuring by Branch-Sorting method. In this paper a efficient tree structure have been proposed in which transactions are ordered based on the last items frequencies. The experimental Results indicate that using of proposed tree enhances efficiency of Branch-Sorting method.

REFERENCES

[1] Agrawal, R. and R. Srikant. *Fast algorithms for mining association rules*. in *International conference on very large data bases(VLDB'94)*. 1994.

[2] Han, J., J. Pei, and Y. Yin, *Mining frequent patterns without candidate generation.* ACM SIGMOD Record, 2000. 29(2): p. 1-12.

[3] Tanbeer, S.K., et al., *Efficient single-pass frequent pattern mining using a prefix-tree.* Information Sciences, 2009. 179(5): p. 559-583.

[4] Savasere, A., E.R. Omiecinski, and S.B. Navathe. *An efficient algorithm for mining association rules in large databases*. in *International conference on very large data bases(VLDB'95)*. 1995.

[5] Toivonen, H. *Sampling large databases for association rules*. in *International conference on very large data bases(VLDB'96)* 1996.

[6] Brin, S., et al. *Dynamic itemset counting and implication rules for market basket data*. 1997: ACM.

[7] Tiwari, A., R.K. Gupta, and D.P. Agrawal. *A novel algorithm for mining frequent itemsets from large database*. in *International Journal of Information Technology*. 2009.

[8] Park, J.S., M.S. Chen, and P.S. Yu, *An effective hash-based algorithm for mining association rules.* ACM SIGMOD Record, 1995. 24(2).

[9] Cheung, D.W., et al. *Maintenance of discovered association rules in large databases: An incremental updating technique*. 1996: IEEE.

[10] Gunaseelan, D. and P. Uma, *An Improved Frequent Pattern Algorithm for Mining Association Rules.* International Journal of Information and Communication Technology Research, 2012. 2(5): p. 436 - 441.

[11] Cheung, W. and O.R. Zaiane. *Incremental mining of frequent patterns without candidate generation or support constraint*. in *Seventh IEEE international database engineering and applications symposium(IDEAS'03)*. 2003.

[12] Koh, J.L. and S.F. Shieh, *An efficient approach for maintaining association rules based on adjusting FP-tree structures1*. Lecture Notes in Computer Science, 2004: p. 417-424.

[13] Leung, C.K.S., et al., *CanTree: a canonical-order tree for incremental frequent-pattern mining.* Knowledge and Information Systems, 2007. 11(3): p. 287-311.

[14] Nadimi-Shahraki, M.H., et al., *Efficient prime-based method for interactive mining of frequent patterns.* Expert Systems with Applications, 2011. 38(10): p. 12654-12670.