International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 12 – No. 14, July 2018 – www.ijais.org

# Efficient Mining of Frequent Itemsets using Improved FP-Growth Algorithm

Abdulkader M. Al-Badani
Faculty of Computer Science & Information Systems, Thamar University, Yemen

Basheer M. Al-Maqaleh
Faculty of Computer Science & Information Systems, Thamar University, Yemen

## ABSTRACT

Frequent itemsets are itemsets that appear frequently in a dataset. Finding frequent itemsets plays an important role in association rules mining, correlations, and many other interesting relationships among data. Frequent itemset mining has been an active research area and a large number of algorithms have been developed. FP- Growth algorithm is currently one of the best approaches to frequent itemsets mining. It constructs a tree structure from transaction dataset and recursively traverse this tree to extract frequent itemsets in a depth first search manner. Also, it takes time to build an FP-tree, suffers from the increasing size of FP-tree and generating large number of frequent itemsets. In this paper, an improved frequent itemsets mining algorithm based on FP-Growth algorithm is proposed. The proposed algorithm uses a two dimensional array structure called Ordered Frequent Itemsets Matrix (OFIM) to construct a highly compact FP-tree. It greatly circumvents repeated scanning of datasets and it reduces the computational time, and reduces the number of frequent items that are generated obtaining significantly improved performance for FP-tree based algorithms.

## General Terms

Data Mining, Association Rule, Frequent Itemsets Mining.

## Keywords

FP-Growth Algorithm, Aprioiri Algorithm, FP-tree, Support Count, Ordered Frequent Itemset Matrix

## 1. INTRODUCTION

Data mining is the process of extracting useful information from huge amount of data stored in the databases [1]. Frequent itemset mining is one of the classical problems in the most of the data mining applications [2]. Frequent itemsets are common in real-life data, such as sets of items bought together in a super store. For example, a set of items, such as milk and coffee, which appear frequently together in a transaction dataset, is a frequent itemset . The frequent itemset can be defined as follows[1]:-

Let I={I1, I2, I3,...,In} be a collection of items. D is the set of transactions in the database, where each transaction T is a set of items, so I includes T. For any transaction A which is included in I, if and only if the transaction T concludes A, A can be called the item set. The support count of item set A is the number of the transaction which includes A in the database D. If the support count of itemset A is greater than or equal to a given support count, A can be called the frequent itemset, and the given support count is the minimum support count (*minsup*).

There are a large number of algorithms to extract frequent itemsets. Two main algorithms are Apriori [3], and FP-Growth [4]. Apriori is a typical algorithm for frequent itemset mining and association rule discovery over transactional databases. The frequent itemsets find out by Apriori can be used to find out association rules, which highlight common trends in the databases. It suffers from the large I/O cost caused by multiple database scans [1].

FP-Growth is the first successful tree-based algorithm of mining the frequent itemsets [4]. It works in a divide and conquers way that considerately reduces the size of the subsequent conditional FP-tree . It requires two scans of the datasets. The FP-tree is a compressed representation of the transactions. However, a compact representation does not reduce the potential combinatorial number of candidate itemsets, which is the bottleneck of FP-Growth [5]. Also, the large databases structure does not fit into main memory due to the very large size tree that might be generated [6]. So, in the proposed algorithm a new two-dimensional array structure, called Ordered Frequent Itemsets Matrix (OFIM), based on FP-Growth algorithm is used. This new structure compacts a transactional database to facilitate a suitable environment for efficient frequent itemsets mining.

The rest of paper is organized as follows: Section 2 presents related work. The original FP-Growth algorithm is presented in Section 3. The OFIM is elaborated in detail in Section 4. The proposed algorithm is presented in Section 5. The experimental results and discussions are described in Section 6. Conclusion and future work are given in Section 7.

## 2. RELATED WORK

This section presents some of the existing algorithms related to frequent itemsets mining. Many algorithms related to frequent itemsets mining are presented in [2,3,4,7].
An improved of FP-Growth algorithm for mining description-oriented rules is introduced in [8]. They have proposed new modification for description of gene groups using Gene Ontology (GO) based FP-Growth algorithm and the results show that the new algorithm allows generating rules faster. An association rule mining using new FP-Linked list algorithm is presented in [9]. It has proposed a new frequent pattern mining algorithm based on FP- Growth idea which is using a bit matric and a linked list structure to extract frequent patterns. A combined approach of frequent pattern growth and decision tree of Infrequent Weighted Itemset (IWI) mining are suggested in [10]. In their paper, two novel quality measures are proposed to test the IWI mining process. Efficient algorithms to find frequent itemsets using data mining are proposed in [11]. These algorithms are proposed to achieve privacy, utility and efficiency frequent itemsets mining, which is based on the frequent pattern growth algorithm. An improved algorithm of frequent itemsets mining is developed in [12]. It has proposed a non-recursive more efficient FPNR-growth algorithm which improves the time and space performance. A new hybrid frequent pattern-

Apriori (FP-AP) algorithm of high utility item set mining is developed in [13]. It has presented FP-AP algorithm, which is the combination of frequent pattern and Apriori algorithm. FP is proposed to split the longer transaction rather than truncate it and also to find the high profitable item with privacy to that item set without redundancy. A survey on FP-Growth tree using association rule mining is presented in [14]. The researchers introduced a new technique which extracts all the frequent itemsets without the generation of the conditional FP-trees. A new algorithm CT-PRO which uses the Compressed FP-tree is developed in [15]. The experimental results of this algorithm are much more efficient in terms of performance. An improved FP-tree algorithm with relationship technique for refined result of association rule mining is proposed in [16]. It has adapted the same idea for identifying frequent item set with large database. An efficient and updatable item-item frequency matrix for frequent itemset generation is introduced in [17]. This matrix maps the data between the items in the form of square matrix.
The most contribution of this work is a novel algorithm that uses OFIM structure to greatly compact structure of FP-tree, and then improve the performance of the algorithms operating on FP-trees.

## 3. FP-GROWTH ALGORITHM

FP-Growth algorithm [4] mines the complete set of frequent itemsets without generating the candidate. It retains the itemsets association by compressing the database representing frequent items into a frequent pattern tree, or FP-tree. It only needs two dataset scans when mining all frequent itemsets [2]. The first scan counts the number of occurrences of each item. The second scan constructs the initial FP-tree which contains all frequency information of the original dataset. Mining the dataset then becomes mining the FP-tree. The pseudo-code of the FP-Growth algorithm of a transaction database is given below [4,6].
**Input**: A transaction database DB, and a *minsup* threshold ξ.
**Output**: FP-tree.
**Procedure:-**
Step-1: Scan the transactional database and find support count for each item.

Step-2: If support(item) < *minsup*, discard the item.

Step-3: Construct a header table called I-list to store the sorted of frequent item-sets in descending order based on its support and node link.

Step-4: Initially, construct FP-Growth tree. In the first step, it creates the root of an FP-Growth tree and labels it as “null”. And read the item in each transaction and create branch for each transaction. If each node has shared a common prefix, increment by 1 otherwise create a new node.

Step-5: In header table, each item points to its corresponding occurrences in the tree through a single link list, which is represented by dotted lines.

Step-6: Construct the mine FP-tree is which called FP-Growth tree.

The FP-tree has a header table associated with it. Single items and their counts are stored in the header table in decreasing order of their frequency[4]. Table 1 shows an example of a transactional dataset and Figure 1 shows the FP-tree, which generated by FP-Growth algorithm from this dataset.

**Table 1: A dataset with nine transactions.**

| TID | List of items |
|---|---|
| T1 | I1,I2,I5 |
| T2 | I2,I4 |
| T3 | I2,I3 |
| T4 | I1,I2,I4 |
| T5 | I1,I3 |
| T6 | I2,I3 |
| T7 | I1,I3 |
| T8 | I1,I2,I3,I5 |
| T9 | I1,I2,I3 |

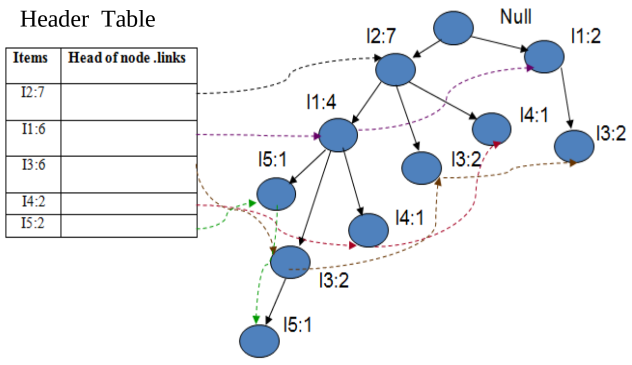

**Figure 1: An Example of FP-tree(*minsup*=50%).**

The generated frequent itemsets are shown in Table 2.

**Table 2: The discovered frequent itemsets by FP-Growth algorithm**

| TID | Conditional FP-tree | Frequent itemsets |
|---|---|---|
| I5 | <I2:2,I1:2> | {I2,I5:2}, {I1,I5:2}, {I2,I1,I5:2} |
| I4 | <I2:2> | {I2,I4:2} |
| I3 | <I2:4,I1:2>,<I1:2> | {I2,I3:4},{I1,I3:4}, {I2,I1,I3:2} |
| I1 | <I2:4> | {I2,I1:4} |

## 4. ORDERED FREQUENT ITEMSET MATRIX

OFIM is a two-dimensional array used to summarize the transactional database and contains all frequent itemsets, which they are sorted in support descending order. The OFIM is N×M, where N is the number of transactions and M is the longest number of ordered frequent items. The proposed algorithm scans the transactional dataset to drive a list of frequent items, in which items ordered in frequency descending order. This ordering is important since the construction of OFIM will follow this order. For any transaction, the candidate itemsets which have occurrence frequencies greater than or equal to the *minsup* threshold are added to the list of frequent itemsets, these lists are called Ordered Frequent Itemsets Lists (OFILs). Rest of the non-frequent candidate itemsets are discarded. The frequent items in each transaction are listed in this ordering in the rightmost column of Table 3.

**Table 3: Transactional dataset with OFILs.**

| TID | List of items | OFILs. |
|---|---|---|
| T1 | I1,I2,I5 | I2,I1,I5 |
| T2 | I2,I4 | I2,I4 |
| T3 | I2,I3 | I2,I3 |
| T4 | I1,I2,I4 | I2,I1,I4 |
| T5 | I1,I3 | I1.I3 |
| T6 | I2,I3 | I2,I3 |
| T7 | I1,I3 | I1,I3 |
| T8 | I1,I2,I3,I5 | I2,I1,I3,I5 |
| T9 | I1,I2,I3 | I2,I1,I3 |

Notice that the frequent items in the transaction is ordered according to their order in the list of frequent items. Referring to the Table 3, the OFIL of transaction I1,I2,I5 is I2,I1,I5. An empty OFIM having N×M is initialized with "0" values. The matrix generation process reads the OFILs list by list. The process extracts items from each list. It then adds the items to the rows and to the corresponding columns of the matrix one by one. The process repeats for each list in the OFILs. Table 4 describes the complete OFIM after reading all the OFILs, which are given in Table 3.

**Table 4. The OFIM.**

| | | | | |
|---|---|---|---|---|
| T1 | I2 | I1 | I5 | 0 |
| T2 | I2 | I4 | 0 | 0 |
| T3 | I2 | I3 | 0 | 0 |
| T4 | I2 | I1 | I4 | 0 |
| T5 | I1 | I3 | 0 | 0 |
| T6 | I2 | I3 | 0 | 0 |
| T7 | I1 | I3 | 0 | 0 |
| T8 | I2 | I1 | I3 | I5 |
| T9 | I2 | I1 | I3 | 0 |

## 5. THE PROPOSED ALGORITHM

Traditional FP-Growth algorithm takes time to construct FP-tree, discovers large number of frequent itemsets, and suffers from the increasing size of FP-tree, which may not fit in the main memory [4,5,6]. The process of discovering frequent itemsets takes the OFIM and a *minsup* threshold as input. The proposed algorithm scans every column in OFIM to compute the support of each different items, and the other (previous) columns are used to distinguish the node's parent node of current column. By this way, we can insert one level of nodes into FP-tree at a time, we cannot compute frequent items one by one in order to save the information of the current nodes and their parent nodes. Also, if any infrequent, say x item is found in any column then there is no any other item after x in the same row is frequent. So, the proposed algorithm deletes this row which greatly reduces the search space and then saves execution time. By using OFIM into the process of tree construction, the expensive frequent items scans in the proposed algorithm are saved. Also, more frequently occurring items are arranged at the top of the FP-tree and thus are more likely to be shared. This indicates that FP-tree structure is usually highly compact. This shortens the time of tree construction and decreasing the size of FP-tree, therefore the performance is much more better than the FP-Growth algorithm. The detailed descriptions of the proposed algorithm are as follows :-

**Input** : A transaction dataset and a *minsup* threshold.
**Output:** FP-tree.

1- Scan the transaction database once. Collect F, the set of frequent items F, and their supports. Sort F in support descending order as OFIL, the list of ordered frequent items. All infrequent itemsets are deleted in this step.

2- Create OFIM. For each row corresponds to the OFIL, all ordered frequent items in OFIL are entered item-by-item into the corresponding columns.

3- Create the root of an FP-tree, T, and label it as "null". Let column number in OFIM be j.

```
4- For (j=1; j<=M; j++)
{
If j=1 Then Do
{
 Collect the set of frequent items and their supports, and sort items according to descending support count. Let the result be [f: n | OFIL] , where f is the first frequent item in OFIL, and n is the count. Insert these nodes as the root's child nodes into the FP-tree. Frequent items, f , are processed according to their order.
}
Else Do
{
Contrast both the current column (j) and the previous columns precede it, compare the set of frequent items and collect their supports. Let the result be [p, f: n | OFIL] where p is the parent frequent items of the previous columns, and f is the current frequent item of column (j). Link the nodes with the same item-name via the node-link structure. This means inserting [f: n] as the child nodes of p into the FP-tree and letting their node-link be linked to the nodes with the same item-name via the node-link structure.
}
}
```

The proposed algorithm mined the FP-tree as follows : start from each frequent length 1 pattern as an initial suffix pattern, construct its conditional pattern base which consists of the set of prefix path in the FP-tree co-occurring with the suffix pattern, then construct its conditional FP-tree and perform mining recursively on such a tree. The pattern growth is a achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree [4]. The FP-tree generated by the proposed algorithm of transactional dataset in Table 1 is shown in Figure 2.

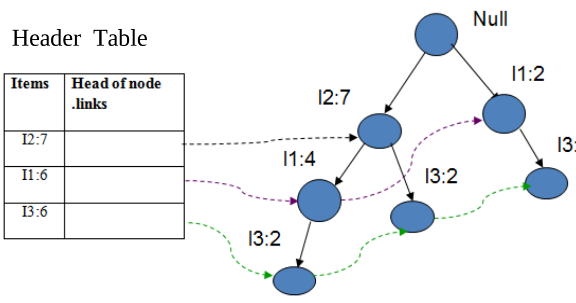

**Figure 2: Construction of FP-tree using the proposed algorithm (*minsup*=50%)**

The generated frequent itemsets are shown in Table 5.

**Table 5: The generated frequent itemsets by the proposed algorithm**

| TID | Conditional FP-tree | Frequent itemsets |
|---|---|---|
| I3 | <I2:4,I1:2>,<I1:2> | {I2,I3:4}, {I1,I3:4}, {I2,I1,I3:2} |
| I1 | <I2:4> | {I2,I1:4} |

Figures 1 and 2 show that the proposed algorithm constructs small size of FP-tree than FP-Growth algorithm. Also, Tables 1 and 2 indicate that the FP-Growth algorithm generates larger number of frequent itemsets than the proposed algorithm.

## 6. EXPERIMENTAL RESULTS AND DISCUSSIONS

The performance of the proposed algorithm is validated on real-world datasets obtained from UCI Machine Learning Repository which is a collection of widely used benchmark and real-world datasets for data mining and KDD community [18]. The performance of the proposed algorithm is evaluated and compared with well-known FP-Growth algorithm in terms of time required to find frequent itemsets and the number of discovered frequent itemsets from the specified datasets. All experiments are performed on a laptop with C++, 32 bit Windows 7, 4GB RAM and 2.2 GHZ Intel core (TM) Duo CPU. Table 6 shows some statistical information about the datasets used in this comparative study.

**Table 6: Characteristics of the test datasets.**

| Datasets | Size | #Transactions |
|---|---|---|
| Grocery | 0.56MB | 10800 |
| Mushroom | 0.12MB | 8124 |
| Car | 0.055MB | 1728 |

The comparison performance of the proposed algorithm and FP-Growth algorithm on these three datasets are demonstrated below :-

### 6.1 Experiment One

This experiment was carried out on the Grocery dataset. It contains one month of real-world point-of- sale transactions data from a typical local grocery outlet. In order to efficiently evaluate the performance of the proposed algorithm over original FP-Growth algorithm, experiments have been conducted and compared several times with different values of *minsup*. The obtained results based on the execution time required to find the frequent itemsets and the number of frequent itemsets of various *minsup* values such as 10%, 20% , 30%, and 50% are shown in Table 7.

**Table 7: Comparison results for the Grocery dataset with various *minsup* thresholds.**

| No. | *minsup* | Execution time per milliseconds (ms) | | # Discovered Frequent itemsets | |
|---|---|---|---|---|---|
| | | FP-Growth | Proposed algorithm | FP-Growth | Proposed algorithm |
| 1 | 10% | 800 | 600 | 13 | 12 |
| 2 | 20% | 700 | 400 | 11 | 6 |
| 3 | 30% | 600 | 300 | 6 | 3 |
| 4 | 50% | 400 | 100 | 3 | 1 |

In general, when the values of *minsup* increase, the execution time and the number of discovered frequent itemsets decrease in both algorithms.

It is observed that the execution time of the proposed algorithm consumes less time compared to the FP-Growth algorithm, even though it varies the *minsup* threshold value. Figure 3 shows the performance of two algorithms according to the execution time for 4 different *minsup* thresholds. It clearly illustrates that the proposed algorithm outperforms than FP-Growth algorithm. The reason behind that the FP-Growth needs to construct a large number of conditional sub-trees and then generates a large number of frequent itemsets, it is not only time consuming but also high memory cost.

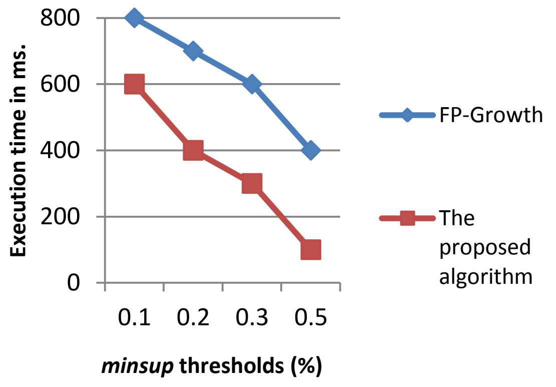

**Figure 3: Comparing the results of the execution time and the *minsup* thresholds for the Grocery dataset.**

### 6.2 Experiment Two

Mushroom dataset was used for this experiment. This dataset has 8124 records, with 23 attributes that contain 119 item. Table 8 shows the execution time and the number of discovered frequent itemsets of the FP-Growth and the proposed algorithm of various *minsup* thresholds such as 10% , 20% , 30%, and 50%.

**Table 8: Comparison results for the Mushroom dataset with various *minsup* thresholds.**

| No. | *minsup* | Execution time per milliseconds (ms) | | # Discovered Frequent itemsets | |
|---|---|---|---|---|---|
| | | FP-Growth | Proposed algorithm | FP-Growth | Proposed algorithm |
| 1 | 10% | 590 | 300 | 20 | 19 |
| 2 | 20% | 560 | 260 | 18 | 7 |
| 3 | 30% | 550 | 250 | 17 | 5 |
| 4 | 50% | 530 | 240 | 13 | 3 |

Table 8 indicates that the execution time of the proposed algorithm are approximately equal even though it varies the *minsup* threshold of the Mushroom dataset and it gives less mining time and a small number of the generated frequent itemsets than the FP-Growth algorithm. The proposed algorithm scales much better than the FP-Growth. This is because as the *minsup* threshold goes down, the number of the frequent itemsets increase dramatically. The candidate sets that the FP-Growth must handle become large, and the pattern matching with a lot of candidates by searching through the FP-tree becomes very expensive. Figure 4 shows the comparison results between the two algorithms according to the execution time with 4 different *minsup* thresholds**.**

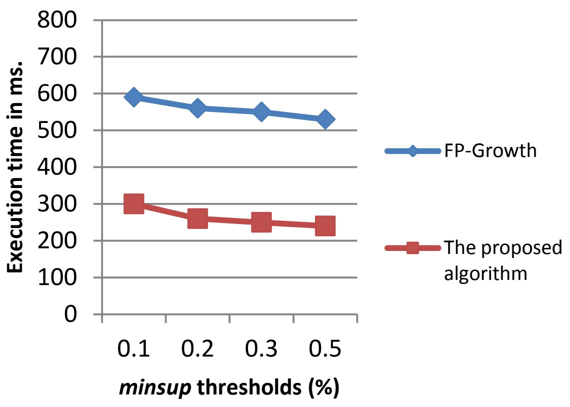

**Figure 4: Comparing the results of the execution time and the *minsup* thresholds for the Mushroom dataset**

## 6.3 Experiment Three

This experiments was carried out on the Car dataset. This dataset contains 1728 instances and 7 attributes. Table 9 shows the execution time and the number of discovered frequent itemsets of the FP-Growth and the proposed algorithm of the various *minsup* thresholds such as 10%, 20% , 30%, and 50%.

**Table 9: Comparison results for the Car dataset with various *minsup* thresholds.**

| No. | *Minsup* | Execution time per milliseconds (ms) | | # Discovered Frequent itemsets | |
|---|---|---|---|---|---|
| | | FP-Growth | Proposed algorithm | FP-Growth | Proposed algorithm |
| 1 | 10% | 900 | 500 | 11 | 8 |
| 2 | 20% | 600 | 400 | 10 | 6 |
| 3 | 30% | 300 | 200 | 9 | 3 |
| 4 | 50% | 200 | 100 | 6 | 2 |

Table 9 indicates that the execution time and the number of discovered frequent itemsets of the proposed algorithm are less than FP-Growth algorithm. The comparison results between the execution time and the *minsup* thresholds of the Car dataset for both algorithms are shown in Figure 5.

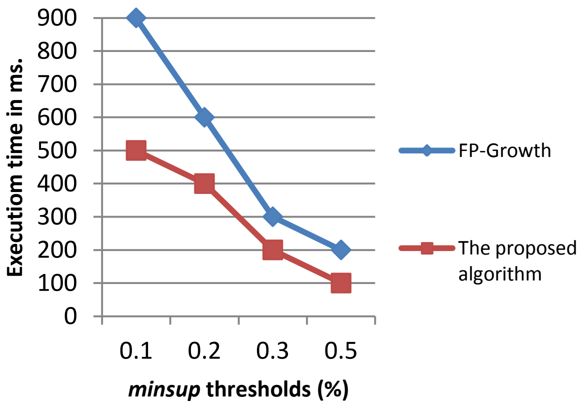

**Figure 5: Comparing the results of the execution time and the *minsup* thresholds for the Car dataset**

This figure indicates as the *minsup* threshold goes up, the execution time and the number of generated frequent itemsets in both algorithms decrease dramatically.

## 7. CONCLUSION AND FUTURE WORK

Efficient algorithms of mining frequent itemsets are crucial for mining association rules. In this paper, an improved FP-Growth algorithm for efficient mining of the frequent itemsets is proposed. The proposed algorithm uses OFILs to construct the OFIM. Consequently, the proposed algorithm uses OFIM to construct a highly compact FP-tree, and then saves the costly dataset scans in the subsequent mining processes which reduces the time of FP-tree construction. It applies a pattern growth method which avoids costly candidate generation. OFIM saves traversal time for all items and the next level of the FP-tree can be initialized directly. The proposed algorithm deletes infrequent items appropriately, thereby the subsequent processes are completing their respective tasks more efficiently without unnecessarily wasting their efforts in processing the irrelevant data. The performance gain achieved by the proposed algorithm is due in most part to the highly compact structure of FP-tree, which stores only the frequent items in a frequency-descending order using OFIM. The experimental results show that the proposed algorithm is superior to the original FP-Growth algorithm in mining time and the number of discovered frequent itemsets.

Future research can improve the proposed algorithm to incrementally update an FP-tree, such as adding daily new transactions into a database containing records accumulated for months.

## 8. REFERENCES

[1] Han, J., Pei, J., and Kamber, M. 2011. Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco, California, USA.

[2] Shridhar, M., and Parmar, M. 2017. Survey on association rule mining and its approaches. International Journal of Computer Sciences and Engineering (IJCSE), 5(3), pp.129-135.

[3] Agrawal, R., and Srikant, R. 1994. Fast algorithms for mining association rules. In Proceeding of 20th International Conference on Very Large Databases (VLDB), pp. 487-499.

[4] Han, J., Pei, J., and Yin, Y. 2000. Mining frequent patterns without candidate generation. ACM. pp, 1-12.

[5] Wei, F., and Xiang, L. 2015. Improved frequent pattern mining algorithm based on FP-Tree. In Proceedings of The Fourth International Conference on Information Science and Cloud Computing (ISCC2015), pp.18-19.

[6] Krupali, R., Garg, D., and Kotecha, K. 2017. An improved approach of FP-Growth tree for frequent itemset mining using partition projection and parallel projection techniques. International Recent and Innovation Trends in Computing and Communication, 5(5), pp. 929-934.

[7] Khanali, H., and Vaziri, B. (2017). A survey on improved algorithms for mining association rules. International Journal of Computer Applications(IJCA), 165(9), pp. 6-11.

[8] Gruca, A. 2014. Improvement of FP-Growth algorithm for mining description-oriented rules. In Man-Machine Interactions, Part of Advances in Intelligent Systems and Computing, (AISC), Springer, vol. 242, pp. 183-192.

[9] Sohrabi, M. K., and Marzooni, H. H. 2016. Association rule mining using new FP-Linked list algorithm. Journal of Advances in Computer Research (JACR), 7(1), pp. 23-34.

[10] Dange, A. S., and Patil, S. J. 2016. A combined approach of frequent pattern growth and decision tree for infrequent weighted itemset mining. International Research Journal of Engineering and Technology ( IRJET), 3(7), pp. 2070- 2075.

[11] Sagar, B. P., and Kale, S. 2017. Efficient algorithms to find frequent itemsets using data mining. International Research Journal of Engineering and Technology ( IRJET), 4(6), pp. 2645- 2648.

[12] Hao, J., and Xu, H. 2017. An improved algorithm for frequent itemsets mining. In 5th International Conference on Advanced Cloud and Big Data (CBD), IEEE Computer Society , pp. 314-317.

[13] Devi, R. S., and Shanthi, D. 2016. A new hybrid frequent Pattern-Apriori (FP-AP) algorithm for high utility item set mining. Middle East Journal of Scientific Research (MEJSR), 24(3), pp. 986-991.

[14] Princy. S, Ankita, H., Babita, P., and Shiv, K. 2017. A survey on FP (Growth) tree using association rule mining. International Research Journal of Engineering and Technology( IRJET), vol. 4, Issue 7, pp. 1637-1640.

[15] Jiten, G., Ashish, P., Swapnit, M., and Christi, L. 2017. Compressed frequent pattern tree. International Journal of Engineering Sciences and Research Technology ( IJESRT), 6(4), pp. 652-657.

[16] Saxena, P. and Jain, R. 2016. An improved FP-Tree algorithm with relationship technique for refined result of association rule mining. International Journal of Scientific Research in Science, Engineering and Technology(IJSRSET), vol. 2, pp. 525-529.

[17] Usman, A., Zhang, P., and Theel, O. 2017. An efficient and updatable item-to-item frequency matrix for frequent itemset generation. ICC'17, Cambridge, United Kingdom, ACM, pp. 978 -983.

[18] Blake, C. L., and Merz., M. J, UCI Repository of Machine Learning Databases [http://www. ics. uci. edu/~ mlearn/ MLRepository. html]. Irvine, CA: University of California‖, Department of Information and Computer Science.