Performance Evaluation of the PML Algorithm based on Memory Usage and Execution Time

Surati Sandipkumar B.
Vivekanand College for Advanced Computer and Information Science
Near Saroli Bridge,
Jahangirpura, Surat

Desai Apurva A.
Department of Computer Science
Veer Narmad South Gujarat University, Surat

ABSTRACT
Frequent pattern generation from transaction datasets is an important issue. This can be achieved through frequent pattern mining. There are many popular algorithms, such as Apriori, FP-Growth, and ECLAT, that are widely used for frequent pattern generation. In this paper, we discuss an innovative frequent pattern mining algorithm using a Linked List (PML). This paper also analyzes the performance of PML algorithm in comparison with other popular algorithms in terms of memory usage and time consumption for generating frequent patterns.

General Terms
Frequent Pattern Mining Methods

Keywords
Performance evaluation of PML, PML, Pattern Mining using Linked List, Frequent Pattern Mining Techniques, Frequent Pattern Mining Algorithms, Association Rule Mining Algorithms.

1. INTRODUCTION
Data Mining is the process of extracting useful information from the databases. Association Rule Mining is used to find associations between items, identify frequent patterns, and generate association rules from the discovered frequent patterns [1]. In the last decade, frequent pattern mining has been a major focused in the field of data mining [2]. Mining frequent patterns from databases is a challenging task and has gained significant popularity in recent trends. Frequent patterns are patterns (items or set of items) that occur frequently in transaction datasets. Numerous algorithms have been developed using different techniques and data structures for mining frequent patterns. Pattern Mining using Linked List (PML) is one such pattern mining algorithm used to mine frequent patterns from databases [3]. It uses the linked list data structure to store itemsets, through which nodes can be easily inserted and removed. The PML pruning process is improved to achieve efficient memory usage and reduce execution time. The algorithm uses both horizontal and vertical data layout. PML directly generates frequent patterns for 2-itemsets and high-order itemsets. In this paper, the process of PML, including the creation of linked lists, insertion of items, pruning of items, and the algorithm's memory and time usage is discussed.

This paper is organized into six sections. Section 2 discusses related work in the area of frequent pattern mining. Section 3 describes the process of PML and an example of PML. Section 4 presents the results, describing the memory usage and runtime of PML and alternate algorithms. Section 5 provides the conclusion.

2. RELATED WORK
Frequent itemset generation algorithms are categorized based on the different techniques they use. Many algorithms have been developed over the last few decades, some of which are discussed here. The popular Apriori algorithm was introduced by R. Aggrawal et al. in 1993 [4]. The algorithm generates frequent patterns using candidate generations and reads the database iteratively. It also generates a large number of candidate sets. The FP-Growth algorithm was proposed in 2004 by J. Han et al [5]. FP-Growth generates frequent patterns without candidate generation. It stores transactions in compressed form in an FP-Tree and mines frequent patterns from the tree. The ECLAT algorithm was presented in 2000 by M. J. Zaki [6]. ECLAT uses a vertical layout to find frequent patterns. The support of every itemset can be easily counted by intersecting the covers of two of its subsets that together generate the set itself.

The Improved Apriori algorithm was proposed by S. Chaudhari et al. in 2016 [7]. This algorithm requires fewer database scans because it removes unnecessary transactions and avoids redundant generation of sub-items while pruning candidate sets. It introduces an attribute named Size Of Transaction (SOT), which contains the number of items in each transaction in the database. It also uses memory more efficiently compared to FP-Growth.

The Improved DC_apriori algorithm was developed by J. Du et al in 2016 [8]. The algorithm uses dynamic array vectors and map sets for storage. The k-itemsets are generated by joining frequent 1-itemsets with frequent (k-1) itemsets. The algorithm reduces the number of connections among frequent itemsets and directly obtains frequent itemsets using only one pruning operation. It also requires fewer database scans.

The CT-PRO algorithm was proposed by J. Gudkha et al. in 2017 [9]. The algorithm is based on the Improved Apriori algorithm and the FP-Tree structure. CT-PRO uses a more compact data structure called the Compressed FP-Tree (CFP-Tree). The traversal of the CFP-Tree is performed in a bottom-up manner. It generates frequent patterns by following a non-recursive pattern-growth approach.

The PFP-growth algorithm was developed by S. Bhise et al. in 2017 [10]. The Private Frequent Pattern Mining algorithm achieves high time efficiency, a high degree of data utility and privacy preservation. The algorithm works in two phases. The first phase is the preprocessing phase, which improves utility and privacy and includes a smart splitting method to transform the database. This phase is performed only once. The second phase is the mining phase, which offsets the information lost during transaction splitting and calculates a runtime estimation method to determine the actual support of the itemsets in the

given database. The dynamic reduction method guarantees privacy during the itemset mining process.

The new Bi-Eclat algorithm was proposed by X. Yu et al in 2014 [11]. The algorithm is based on ECLAT algorithm. The itemsets are sorted in descending order according to their frequency in the transaction cache, while support counting uses the ascending order of support values.

ECLAT_Growth was presented by Z. Ma. Et al. in 2016 [12]. It is an improvement over ECLAT. Initially, the algorithm scans the database and stores it in a table in the form of a vertical dataset layout. Next, it generates an increased two-dimensional pattern tree. From the vertical dataset table, it adds itemsets with their TID sets into the pattern tree row by row. In the tree, the newly added itemsets are combined with existing frequent itemsets to generate new frequent itemsets. Finally, all frequent itemsets are obtained by extracting all the nodes from the pattern tree.

The PM algorithm was proposed by S. Surati and A. Desai [13]. In this algorithm, frequent itemsets are generated using simple functions. It performs its operations without using complicated data structures or prefix trees. The concept of an associative array is used to store frequent itemsets. The processing technique used in the PM algorithm is very simple.

This brief survey shows that different data structures have been used by various researchers for frequent pattern generation. However, no attempt using linked list could be found. This paper describes the process of the Pattern Mining using Linked list (PML) algorithm, which generates frequent patterns using Linked list. The algorithm uses both horizontal and vertical data layouts. The advantage of using a vertical layout is that it counts frequency efficiently through intersection operations on transaction IDs (TIDs) and also provides automatic pruning of irrelevant data [14].

3. PROCESS OF PML

The PML algorithm uses both horizontal and vertical dataset layout simultaneously. To generate 1-itemsets, it uses the horizontal dataset layout. For 2-itemsets and higher-order itemsets, it uses the vertical dataset layout.

The PML algorithm works in two phases: 1) Insertion Phase 2) Pruning Phase. To store itemsets, it uses a linked list data structure. The linked list is a core data structure of PML and plays an important role in generating frequent itemsets. The following shows the structure of a node in the linked list.

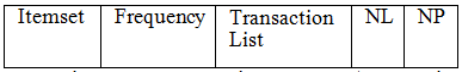

Fig 1: Structure of Node

The first member is the itemset, which stores the item or combination of items that occur in the transactions of the transaction dataset. The second member is frequency, which represents the number of occurrences of the itemset in the transaction dataset.

The Transaction List is the list of transaction IDs(TIDs) in which the item or itemset occurs. NP is the next pointer, which points to the node at the same level. NL is the next level pointer, which points to the node at the next level.

3.1 The PML Algorithm

Step 1: Set the minimum support threshold for the item/itemset.

Step 2: For each transaction do

Step 3: Find the number of items (single item / 1-itemset) from the transaction and generate a node for each new item in the Linked list at 1-level. Set its frequency to 1.

Step 4: If an item already exists, then increment the frequency by 1.

Step 5: End For

Step 6: Remove the nodes whose frequency is less than minimum support threshold.

Step 7: Sort the (item list) Linked list in ascending order as per the items.

Step 8: For each item, in the Linked list

Step 9: Read the transactions from the database and generate a transaction list for the item in which it occurs.

Step 10: Store transaction list of the items in its respected node in the Linked list.

Step 11: End For

Step 12: While SUCCESS is TRUE, do

Step 13: Set SUCCESS=FALSE.

Step 14: For each K-itemset at K-level

Step 15: If only one itemset node at K-level then exit.

Step 16: Generate K+1 itemset with its frequency using the intersection of the transaction list of K-itemset with every other K-itemset's transaction list. If the frequency of generating K+1 itemset satisfies the MIN_SUP, generate itemset node at K+1 level with its frequency and transaction list.

Step 17: If at least one K+1 frequent itemset is generated, set success =TRUE.

Step 18: Remove every K-itemset node at K-level from the Linked list if it is no more needed. In root level, only unnecessary node (the node without next level) and transaction list of every node are removed)

Step 19: Continue this process for all possible combinations of two K-itemset at K-level.

Step 20: End For

Step 21: continue this process for the next level.

Step 22: End while

Set the Minimum support threshold (MIN_SUP). The first step is the insertion of items into the Linked List. During the first scan of the dataset, the PML generates nodes in the linked list for each item and sets the frequency of each item. The items are generated at 1st level of the linked list. These items are called 1-itemsets. All the items at the 1st level are connected by the NP pointer. After the first scan of the transaction dataset, the linked list contains all the 1-itemsets along with their frequencies.

The next step is pruning. If an item is infrequent, then the subsets (items of the next level) generated from that item will also be infrequent, so they must be removed. Remove all the nodes from the 1st level of the linked list that do not satisfy MIN_SUP. The advantage of pruning is that it reduces unnecessary traversal of the linked list and decreases memory consumption.

Now sort the nodes of the linked list item-wise. Output the items and frequencies of the 1st level. The second and final scan of the transaction dataset is performed to store the transaction list of every frequent item in the respective node.

The 2nd level of items is called 2-itemsets. The 2-itemsets are generated from every possible combination of items from the 1st level. All 2-itemsets are generated and stored at the 2nd level if the intersection of the transaction lists for the two items gives a frequency that satisfies MIN_SUP.

The advantage of PML is that it generates direct frequent patterns from the 2nd level onward. Once the 2nd level is generated from the 1st level, the transaction list of the 1st level is removed because the 3rd level will be generated from 2nd level. This helps improve memory utilization. Once the 3rd level is generated, the 2nd level is removed.

This process stops when there are no more frequent itemsets.

3.2 An Example of PML

Table 1. Transaction Dataset

TID	Item list
T1	4 6 7
T2	1 2 3 4 5
T3	1 2 3 4 6 7
T4	1 2 3 5 6
T5	1 2 3 5 7
T6	1 2 3 5 7

Table 1 represents the transaction dataset. It has two columns, namely TID and Item list. The TID column contains a unique transaction ID for each transaction. Each transaction contains a set of the items, which is given in the Item List column. The transaction dataset is in a horizontal data layout. The goal is to generate frequent patterns from the transaction dataset. Let us consider that MIN_SUP=60%. The process of PML algorithm is as follows.

3.2.1 Level – 1 Generation (Root level)

Initially the linked list is empty. The first scan of transaction dataset reads the transactions and generates a node in the linked list for each item in the transaction. If the item is already present in the linked list, then the support count is incremented. Once all the items are generated, the non-frequent nodes are removed. The nodes are sorted item-wise in ascending order and output to a file. The transaction list of each item is stored in the respective node.

3.2.1.1 Insertion

Read the transaction dataset once. Generate the first node in the linked list. Store the item in the itemset member and set its frequency to 1. For each new item in the transaction dataset, a new node is generated, the item is stored in the itemset member of the node and the frequency is set to 1. Every itemset at the 1st level is connected with the NP pointer. If the item already exists, then only its frequency is incremented. Figure 2 shows the 1-itemsets of the 1st level.

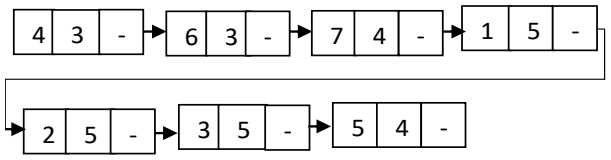

Fig 2: Level 1 - Insertion

3.2.1.2 Pruning

Remove all the 1-itemset nodes from the linked list that do not satisfy MIN_SUP. In the linked list (Fig. 3), itemset 4 and 6 are removed because their frequencies are less than MIN_SUP. Now, we have the frequent 1-itemset list (Fig. 4).

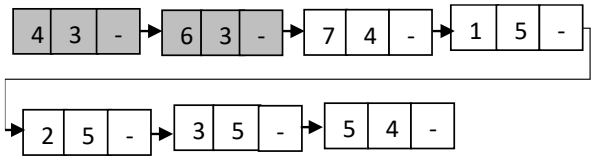

Fig 3: Level 1 - Pruning

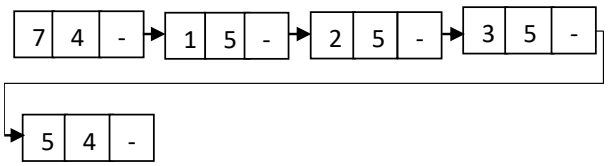

Fig 4: Level 1 – Frequent 1-itemsets

3.2.1.3 Sorting

Now, sort the nodes of the linked list item-wise in ascending order (Fig. 5). This increases the traversal speed. Figure 5 shows the nodes sorted item-wise.

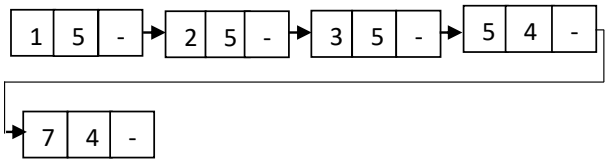

Fig 5: Level 1 – Sorting (Item-wise)

3.2.1.4 Store the transaction list

Read the transaction dataset second time to store the transaction list of each item in the respective 1-itemset node. This is very useful for generating the next levels. The process of frequency counting becomes faster by using intersection operations on transaction IDs (TIDs). It is also useful for generating direct frequent patterns.

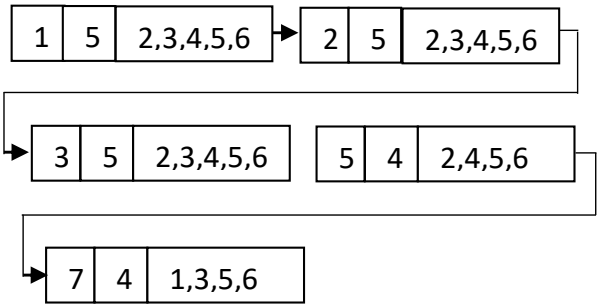

Fig 6: Level 1 – Store the transaction list

3.2.2 Level – 2 Generation

Level -2 is generated from Level -1. There is no need to read the transaction dataset again. From Level-2 onwards, direct frequent itemsets are generated by the intersection of transaction IDs stored in the transaction list of each node. Also, remove the nodes from the transaction list at the 1st level whenever they are no longer needed.

3.2.2.1 Insertion

Combine every possible pair of 1-itemsets from Level-1, intersect their transaction lists and generate a 2-itemset node if the frequency of the intersection satisfies MIN_SUP. Figure 7 shows that item 1 and item 2 of Level-1 generate itemset 1,2 at Level-2 because the intersection of their transaction lists gives a frequency (here, 5) that satisfies MIN_SUP (given as 4). Figure 8 shows the generation of 2-itemsets by combining item 1 with every other item at Level-1.

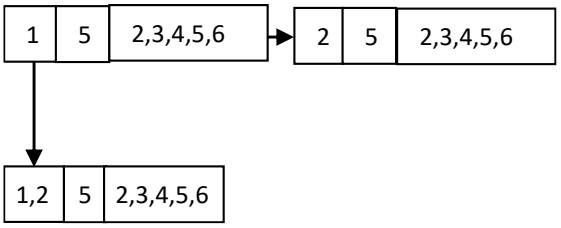

Fig 7: Level 2 – Insertion (Combine item 1 with item 2)

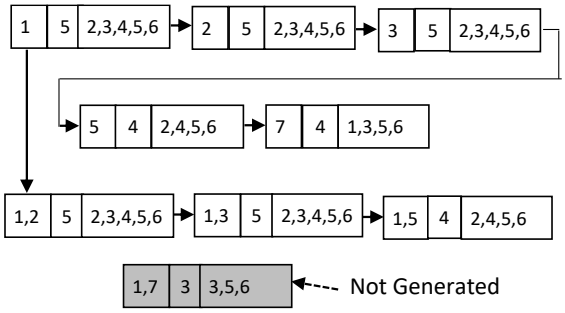

Fig 8: Level 2 – Insertion (Combine item 1 with other 1-itemset)

Figure 9 shows the complete linked list with frequent 2-itemsets at Level-2.

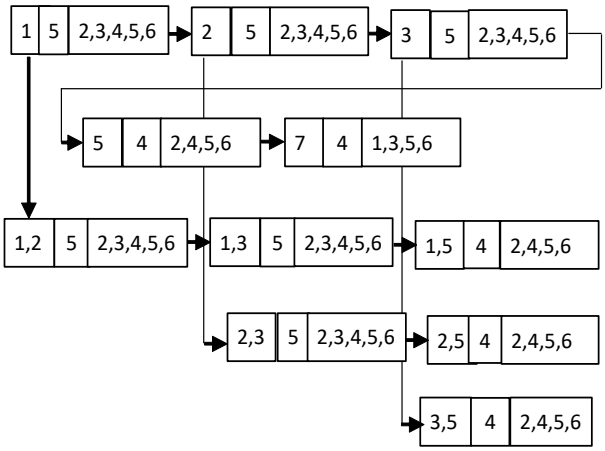

Fig 9: Level 2 – Frequent 2-itemsets

3.2.2.2 Remove transaction lists and unnecessary nodes from Level-1

Once the 2-itemsets are generated, remove transaction lists from the nodes of Level-1. Also, remove those 1-itemset nodes that do not have a next level. Figure 10 shows the pruning of transaction lists and unwanted items from Level-1. Here, item 5 and 7 nodes are removed because they do not have a next level. This process is useful for the efficient use of memory.

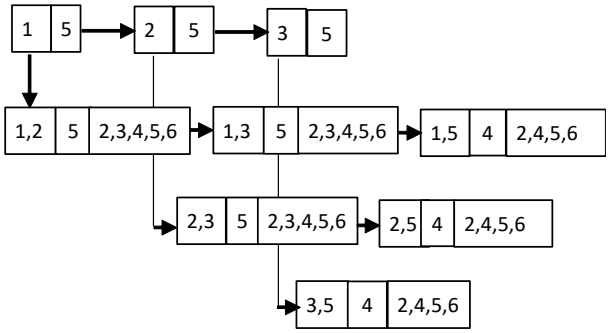

Fig 10: Level 2 – Pruning of transaction list and unnecessary nodes at Level - 1

3.2.3 Level – 3 Generation

Now, 3-itemsets are generated at Level-3 from the 2-itemsets of Level-2, and the 2-itemset nodes are removed whenever they are no longer needed.

3.2.3.1 Insertion

Start with the first node, 1,2. Combine it with every other possible 2-itemset node at Level-2 and intersect their transaction lists. if the frequency of the intersection satisfies MIN_SUP, then generate the node at Level-3. Figure 11 shows the 3-itemsets generated by combining itemset 1,2 with every other possible itemset at Level–2.

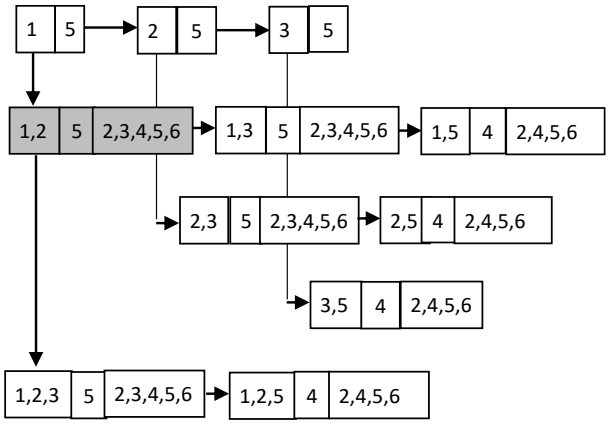

Fig 11: Level 3 - Insertion

3.2.3.2 Pruning

All the possible 3-itemsets are generated from the 1,2 itemset, so the 1,2 itemset is no longer needed. Remove the node 1,2 from the linked list. Figure 12 shows which node has to be removed.

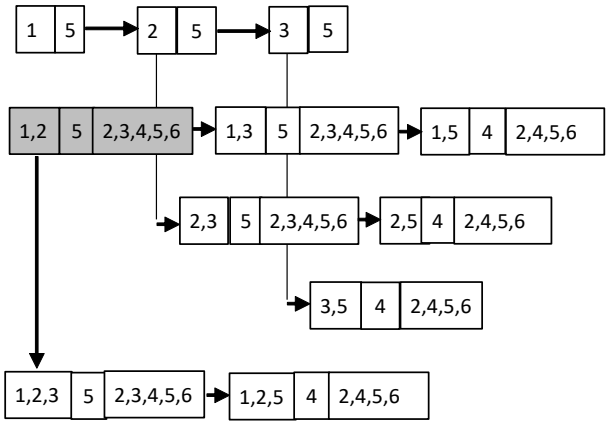

Fig 12: Level 3 - Pruning

Figure 13 shows the linked list after the removal of node 1,2. It also shows how the nodes are relinked after the removal of the specific node. The next level of node 1,2 is linked to the next level of node 1,3, after which node 1,2 is removed.

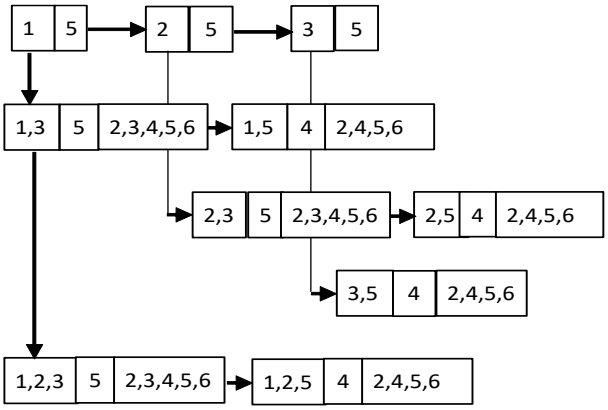

Fig 13: Level 3 - Pruning

The insertion and pruning phases work for the generation of 3-itemsets in the above manner. Figure 14 shows the complete linked list of 3-itemsets at Level-3.

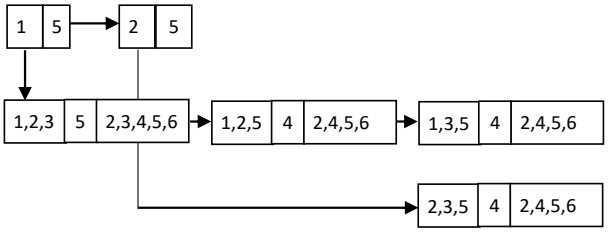

Fig 14: Level 3 – Frequent 3-itemsets

3.2.4 Level – 4 Generation

4-itemsets are generated at Level-4 using every possible combination of 3-itemsets from Level-3. Remove the 3-itemset nodes of Level-3 whenever they are no longer needed.

3.2.4.1 Insertion

Start with node 1,2,3 and combine it with every other possible node at the 3rd level. Intersect their transaction lists, and if their frequency satisfies MIN_SUP, then generate the node at the 4th level. The 2,3,5 itemset will not combine with any node because item 2 is the last node. Only one node is generated at Level -4, so stop. Figure 4 shows the Level -4 generation.

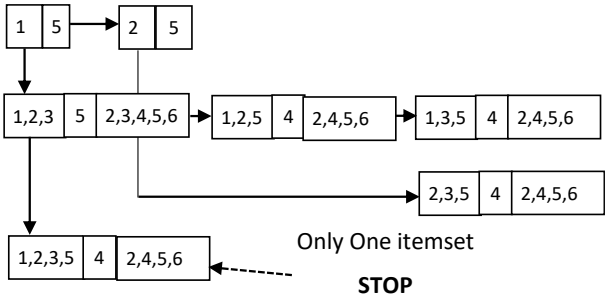

Fig 15: Level 4 – Insertion

4. RESULT

The experiment was conducted using the PML algorithm and alternative algorithms (Apriori, FPGrowth, and ECLAT) on various datasets. The experiment consists of analyzing the time complexity and memory usage of the PML and alternative algorithms. Table 2 provides the details of the Chess, Mushroom and VNSGU datasets.

Table 2. Dataset Details

Dataset	Total No. of items	Total No. of Transactions	Maximum Transaction Length
chess.dat	119	3196	37
mushroom.dat	75	8124	23
VNSGU.txt	11090	94079	295

4.1 Memory Usage

The experiment was conducted on the Chess, Mushroom and VNSGU datasets using the PML and alternate algorithms to determine memory usage at different MIN_SUP values. The following figures show the experiments performed on various transaction datasets. The horizontal axis represents the MIN_SUP, while the vertical axis represents memory usage in megabytes (MBs) at different MIN_SUP values. The solid line refers to the PML algorithm. The round dot line represents the Apriori algorithm. The dash line represents the ECLAT algorithm. The long dash dot dot line represents FP-growth algorithm.

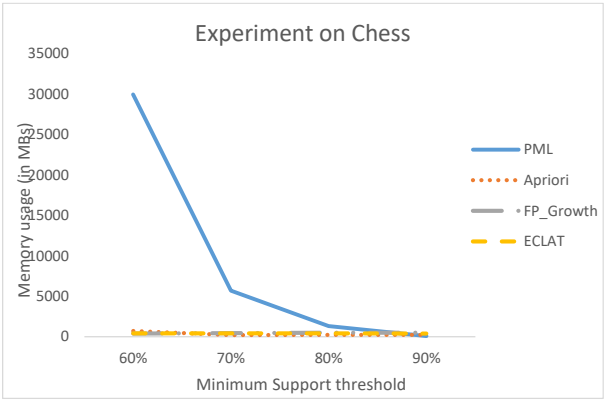

Fig 16: Experiment on Chess dataset (Memory usage)

Figure 16 shows the results for the Chess dataset. The results show that when the MIN_SUP is low (e.g., 60%), the memory usage of PML is higher than that of the alternate algorithms. However, when the MIN_SUP is high (e.g., 90%), the memory usage of PML is lower than that of the alternate algorithms.

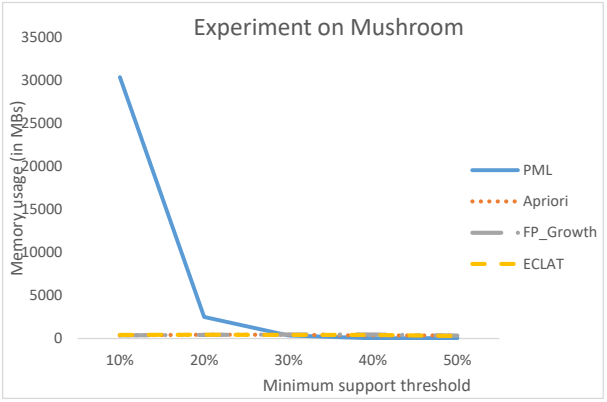

Fig 17: Experiment on Mushroom dataset (Memory usage)

Figure 17 represents the results for the Mushroom dataset. The results for the mushroom dataset show that when the MIN_SUP is low (e.g., 10%), the memory usage is higher than that of the alternate algorithms. However, when the MIN_SUP is high (e.g., 50%), the memory usage is lower compared to the alternate algorithms.

Figure 18 shows the results for the VNSGU dataset. When the MIN_SUP is low (e.g., 2%) PML uses less memory compared to the alternate algorithms. Similarly, when the MIN_SUP is high (e.g., 6%) PML also uses less memory compared to the alternate algorithms (Apriori, ECLAT, and FP-Growth). The reason for the lower memory usage of PML at high MIN_SUP values is that it generates direct frequent patterns from the 2-itemset onwards. Another important feature is the pruning technique, which helps improve memory efficiency.

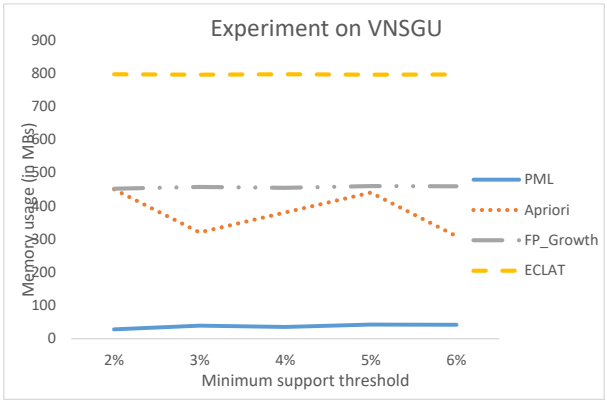

Fig 18: Experiment on VNSGU dataset (Memory usage)

4.2 Time Complexity Analysis

The experiment was conducted on the Chess, Mushroom, and VNSGU datasets using the PML and alternate algorithms to analyze time complexity at different MIN_SUP values. The following figures show the experiments performed on various transaction datasets. The horizontal axis represents MIN_SUP, while the vertical axis represents the runtime in seconds at different MIN_SUP values. The solid line refers to the PML algorithm. The round dot line represents the Apriori algorithm. The dash line represents the ECLAT algorithm. The long dash dot dot line represents the FP-growth algorithm.

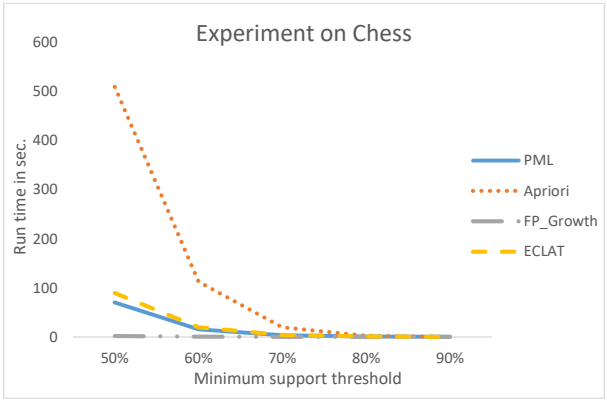

Fig 19: Experiment on Chess dataset (Run time)

The experiment was conducted on the Chess dataset using PML and alternate algorithms at different minimum support thresholds. PML produces the same results as the alternate algorithms; that is, the frequent itemsets generated at different minimum support thresholds are identical to those generated by the alternate algorithms. When the MIN_SUP is 90%, PML is slower compared to the alternate algorithms. However, when MIN_SUP is 80%, PML is faster than Apriori and ECLAT. Similarly, when the MIN_SUP is 70%, 60% and 50%, PML is faster than Apriori and ECLAT. Overall, for the chess dataset, the PML algorithm performs faster than Apriori and ECLAT at most MIN_SUP thresholds, but it is slower than the FP-Growth algorithm.

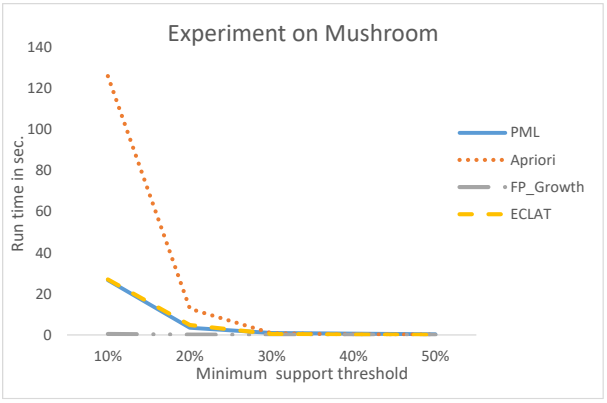

Fig 20: Experiment on Mushroom dataset (Run time)

The experiment was conducted on the Mushroom dataset using PML and alternate algorithms at different minimum support thresholds. PML produces the same results as the alternate algorithms; that is, the frequent itemsets generated at different minimum support thresholds are identical to those generated by the alternate algorithms. When the MIN_SUP is 50%, PML is slower compared to the alternate algorithms. Similarly, when the MIN_SUP is 40% and 30%, PML is also slower than the alternate algorithms. However, when the MIN_SUP is 20% and 10%, PML is faster than Apriori and ECLAT. Overall, for the Mushroom dataset, the PML algorithm performs faster than Apriori and ECLAT when the MIN_SUP threshold is low.

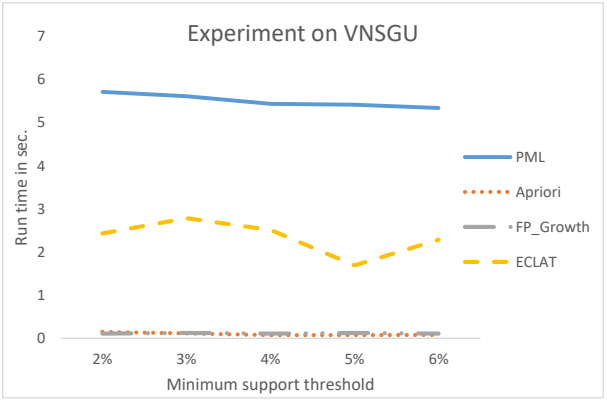

Fig 21: Experiment on VNSGU dataset (Run time)

The experiment was conducted on the VNSGU dataset using PML and alternate algorithms at different minimum support thresholds. PML produces the same results as the alternate algorithms; that is, the frequent itemsets generated at different minimum support thresholds are identical to those generated by the alternate algorithms. When, the MIN_SUP is 5%, PML is slower compared to the alternate algorithms. Similarly, when the MIN_SUP is 2%, 3%, and 4%, PML is also slower than the alternate algorithms. Overall, for the VNSGU dataset, the PML algorithm performs slower than the alternate algorithms at different MIN_SUP thresholds. The results show that when the number of frequent patterns is low, PML performs slower compared to alternate algorithms.

5. CONCLUSION

The PML algorithm uses both horizontal and vertical data layouts to generate frequent patterns. It reads the database only twice. The algorithm uses a linked list data structure that stores information such as itemsets, frequency, and transaction lists in each node. The linked list is used for fast traversal. It generates direct frequent patterns from the 2-itemset level onwards.

The PML and alternate algorithms were executed on various datasets to evaluate the memory usage and time complexity of the algorithms. The results show that when the MIN_SUP is low, PML uses more memory than the alternate algorithms. However, when the MIN_SUP is high, PML uses less memory compared to the alternate algorithms. PML uses a pruning technique that improves memory efficiency and generates direct frequent patterns from the 2-itemset level onwards. When the number of frequent itemsets is large, PML runs faster compared to the alternate algorithms. However, when the number of frequent itemsets is small, PML performs slower than the alternate algorithms.

6. REFERENCES

[1] R. Agrawal, T.Imielinski and A. Swami, 1993. Mining Association Rules between set of items in large database.Proceeding of the 1993 ACM SIGMOD Conference Washington DC, USA, May 1993.

[2] J. Han, H. Cheng, D. Xin, and X. Yan, 2007. Frequent Frequent pattern mining: current status and future directions, Data Mining Knowledge Dis(2007), Springer Science+Business Media, LLC 2007.

[3] S. Surati, A. Desai, 2017, Pattern Mining Using Linked List (PML), 8th Internation Conference on Computing, Communication and Networking Technologies (ICCCNT), Delhi, India, 2017.

[4] R. Agrawal and R. Srikant, 1994. Fast Algorithms for Mining Association Rules. Proc. 20th Int. Conf. on very Large Databases (VLDB 1994, Santiago de Chile), 487–499. Morgan Kaufmann, San Mateo, CA, USA 1994

[5] J. Han, J. Pei, Y. Yin and R. Mao, 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining Knowledge Discovery, 8: 53-87.

[6] M.J. Zaki, 2000. Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng., 12: 372-390.

[7] S. Chaudhari, M. Borkhatariya, A. Churi and M. Bhonsle, 2016. Implementation and Analysis of Improved Apriori Algorithm, International Journal of Engineering Science and Innovative Technology (IJESIT) Vol. 5, Issue 2, March 2016.

[8] J. Du, X. Zhang and H. Zhang, 2016. Research and improvement of Apriori algorithm, 6th Internation conference on information science and technology (ICIST), May 2016.

[9] J. Gudkha, A. Patel and S. More, 2017. Compressed Frequent Pattern Tree, International Journal of Engineering Sciences & Research Technology, April, 2017.

[10] S. Bhise, Prof. S. Kale, 2017. Efficient Algorithms to find Frequent Itemset Using Data Mining, International Journal of Engineering and Technology (IRJET), Volume: 04 Issue: 06, June 2017.

[11] X.Yu and H. Wang, 2014. Improvement of ECLAT algorithm based on Support in Frequent Itemset Mining, Journal of Computers, Vol. 9, No. 9, September 2014.

[12] Z. Ma, J. Yang, T. Zhang and F. Liu, 2016. An Improved ECLAT algorithm for Mining Associaton Rules based on Increased Search Strategy, International Journal of Database Theory and Application, Vol. 9, No. 5 (2016).

[13] S. Surati., A. Desai, Pattern Mining (PM) Algorithm, 2012. VNSGU Journal of Science & Technology, Vol. 3, Issue 2, March 2012.

[14] D. Gunopulos, H. Mannila, and S. Saluja,1997. Discovering all the most specific sentences by randomized algorithms. In Intl. Conf. on Database Theory, January 1997.