Abstract— Data mining represents the process of extracting interesting and previously unknown knowledge (patterns) from data. Frequent pattern mining has become an important data mining technique and has been a focused area in research field. Frequent patterns are patterns that appear in a data set most frequently. Various methods have been proposed to improve the performance of frequent pattern mining algorithms. An association rule expresses the dependence of a set of attribute-value pairs, called items, upon another set of items (item set). The association rule mining algorithms can be classified into two main groups: the level-wise algorithms and the tree-based algorithms. The level-wise algorithms scan the entire database multiple time but they have moderate memory requirement. The two phase algorithms scan the database only twice but they can have extremely large memory requirement. In this research, Performance study has been done which shows the advantages and disadvantage of algorithms used in association rules mining FP growth, Prepost+ and FIN. The main goal of this research is to explore the overview of the current research being carried out using the data mining techniques.

Keywords— Frequent Itemset Mining, FP Tree, FIN, Prepost.

I. INTRODUCTION

Pattern mining algorithms can be applied on various types of data such as transaction databases, sequence databases, streams, strings, spatial data, graphs, etc.

Pattern mining algorithms can be designed to discover various types of patterns: subgraphs, associations, frequent Itemset mining(FIM), indirect associations, trends, periodic patterns, sequential rules, lattices, sequential patterns, high-utility patterns, etc. There are two stages in association rules mining. 1) To find all frequent itemsets. 2) To generate reliable association rules from all frequent itemsets.

There are following types/ways to perform FIM 1) algorithms for discovering frequent itemsets from a transaction database. 2) algorithms for discovering frequent closed itemsets from a transaction database. 3) algorithms for discovering frequent maximal itemsets from a transaction database. 4) algorithms for mining frequent itemsets with multiple minimum supports 5) algorithms for mining generator itemsets from a transaction database 6) algorithms for mining rare itemsets and/or correlated itemsets from a transaction database 7) algorithms for performing targeted and dynamic queries about association rules and frequent itemsets. 8) algorithms to discover frequent itemsets from a stream 9) the U-Apriori algorithm for mining frequent itemsets from uncertain data 10) the VME algorithm for mining erasable itemsets. 11) Unique Constraint Frequent Itemset Mining.

II. FIM

Let I = {I1, I2, …., In} be a set of items. Let D, the task relevant data, be a set of transactions in a supermarket, where each transaction T is a set of items, such that T ⊆ I. Each transaction is assigned an identifier called TID. Let A be a set of items, a transaction T is said to contain A if and only if A ⊆ T. An association rule is an implication of the form A⇒B, where A⊂I, B⊂I, and A∩B=∅. The rule A⇒B holds in the transaction set D with support s, where s is the percentage of transactions in D that contain A∪B (i.e., both A and B). This is taken to be the probability P(A∪B). The rule A⇒B has confidence c in the transaction set D if c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P(B|A). That is, Support (A⇒B) = P(A∪B) = s, Confidence (A⇒B) = P(B|A) =Support (A⇒B)/Support (A) =c. Thus association rules is composed of the following two steps: 1) Find the large item sets that have transaction support above a minimum support and 2) From the discovered large item sets generate the desired association rules.

<table>
<thead>
<tr>
<th>TID</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, f, g</td>
</tr>
<tr>
<td>2</td>
<td>a, b, c, e</td>
</tr>
<tr>
<td>3</td>
<td>b, c, e, i</td>
</tr>
<tr>
<td>4</td>
<td>b, c, e, h</td>
</tr>
<tr>
<td>5</td>
<td>b, c, d, e, f</td>
</tr>
</tbody>
</table>

TABLE 1- Transaction database
TABLE 2- Ordered frequent itemsets of table-1

<table>
<thead>
<tr>
<th>TID</th>
<th>FLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, f</td>
</tr>
<tr>
<td>2</td>
<td>b, c, e, a</td>
</tr>
<tr>
<td>3</td>
<td>b, c, e</td>
</tr>
<tr>
<td>4</td>
<td>b, c, e</td>
</tr>
<tr>
<td>5</td>
<td>b, c, e, f</td>
</tr>
</tbody>
</table>

There are two phases to deal with association rule mining. First one is about the algorithm efficiency. The research on developing an algorithm with less computation complexity is one of the most interesting topics related to association rule mining. The mining efficiency is so important because association rule mining always works on large database. The number of rules grows exponentially with the number of items. The related work mainly focus on efficient pruning on large data sets and reducing the times of scanning data. Second phase is to find the effective ways needed to select interesting rules from discovered rules.

A. The FP-Growth algorithm

Frequent pattern growth (FP-Growth) was introduced by Han, Pei, and Yin in 2000 to forego candidate generation. It inserts sorted items by frequency into a pattern tree or FP-tree.

Definition - A frequent-pattern tree is a tree structure defined below.

1. It consists of one root labeled as “null”, a set of item-prefix subtrees as the children of the root, and a frequent-item-header table.
2. Each node in the item-prefix subtree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.
3. Each entry in the frequent-item-header table consists of two fields, (1) item-name and (2) head of node-link (a pointer pointing to the first node in the FP-tree carrying the item-name).

Based on this definition, we have the following FP-tree construction algorithm.

Algorithm (FP-tree construction)

Input: A transaction database DB and a minimum support threshold s.
Output: The frequent-pattern tree of DB.
Method: The FP-tree is constructed as follows.

1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as FList, the list of frequent items.
2. Create the root of an FP-tree, T, and label it as “null”.

For each transaction Trans in DB do the following:

Select the frequent items in Trans and sort them according to the order of FList. Let the sorted frequent-item list in Trans be [p | P], where p is the first element and P is the remaining list. Call insert tree([p | P], T ). The function insert tree([p | P], T ) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N’s count by 1; else create a new node N, with its count initialized to 1, its parent link linked to T, and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P, N) recursively.

Analysis - The FP-tree construction takes exactly two scans of the transaction database: The first scan collects the set of frequent items, and the second scan constructs the FP-tree. The cost of inserting a transaction Trans into the FP-tree is O(|freq(Trans)|), where freq(Trans) is the set of frequent items in Trans.

Example - Let the transaction database, DB, be represented by the information from Table 1 and s = 0.4. The frequent 1-itemsets set F1 = {a, b, c, e, f}. Fig. 1 shows the FP tree resulting from Example.
Figure 2 shows the PPC-tree resulting from Example 1. The node with (4, 8) means that its pre-order is 4, post-order is 8, the item-name is b, and count is 4. Note that the PPC-tree is constructed using Table 2.

![Fig. 2 - PPC tree (Preorder Postorder Tree)](image)

The set-enumeration tree for finding frequent itemsets is depicted in Fig. 3. For example, the node in the bottom left of Fig. 3 represents itemset \{bceaf\} and registers item b.

**FIN algorithm**

FIN (Fast mining frequent itemsets using Nodesets) introduced by Zhi-Hong Deng, Sheng-Long Lv in 2014 are based on a POC-tree Pre-Order Coding tree. Definition POC-tree is a tree structure: (1) It consists of one root labeled as ‘null’, and a set of item pre-fix subtrees as the children of the root. (2) Each node in the item prefix subtree consists of five fields: item-name, count, children-list, pre-order, item-name registers which item this node represents, count registers the number of transactions presented by the portion of the path reaching this node, children-list registers all children of the node, preorder is the pre-order rank of the node. The framework of FIN consists of: (1) Construct the POC-tree and identify all frequent 1-itemsets; (2) scan the POC-tree to find all frequent 2-itemsets and their Nodesets; (3) mine all frequent k(>2)-itemsets. For enhancing the efficiency of mining frequent itemsets, FIN adopts promotion, which is based on superset equivalence property, as pruning strategy.

![Fig. 3 - A set-enumeration tree](image)

**IV. COMPARISON TABLE**

The following Comparison Table shows comparison between all these algorithms in terms of scalability, runtime, memory consumption, advantages and Disadvantages.

<table>
<thead>
<tr>
<th>Author’s Name and Paper Title</th>
<th>Technique</th>
<th>Characteristics</th>
<th>Limitations</th>
<th>Comparison of Efficiency in terms of scalability,</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **FP-Growth algorithm**  
**Han, Pei, and Yin** | 1. a data structure, called frequent-pattern tree, or FP-tree is constructed.  
2. an FP-tree-based pattern-fragment growth mining method is developed, which starts from a frequent length-1 pattern.  
3. constructs its conditional FP-tree, and performs mining recursively with such a tree.  
4. A partition based search technique is employed. | The major operations of mining are count accumulation and prefix path count adjustment. | FP-growth method becomes inefficient when datasets are sparse because FP-trees become very complex and larger.  
If Datasets contain abundant mixture of long and short frequent patterns, FP-tree is compact most of the time.  
When support is very low, FP-tree becomes bushy.  
The advantages of FP-growth over Apriori becomes obvious when the dataset contains an abundant number of mixtures of short and long frequent patterns.  
FP-growth is faster than Apriori. | runtime, memory consumption | 1. FP-growth method is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm and also faster than some recently reported new frequent-pattern mining methods.  
2. Best one in terms of memory consumption. |
| **PrePost and PrePost+ algorithms**  
**Zhi-Hong Deng, Sheng-Long Lv** | Here PPC-tree structure is used.  
PrePost+ employs a more efficient data structure, namely N-list.  
PrePost+ adopts superset equivalence as | PrePost+ is effective and performs better than PrePost on all datasets in terms of runtime and memory consumption.  
PrePost+ consumes a bit more memory than FP-growth.  
PrePost+ is the fastest one among all algorithms for each minimum support. FP-growth is slower than PrePost and FIN for high minimum support. | runtime, memory consumption | 1. PrePost+ always performs best compared to FP-growth no matter which dataset is used and what the minimum threshold is.  
2. PrePost+ avoids generating lots of redundant N-lists.  
3. In most cases, the memory consumed... |
<table>
<thead>
<tr>
<th>FIN algorithm</th>
<th>Works on the principle of Pre-Order Coding (POC) tree.</th>
<th>encodes each node of a POC-tree with only pre-order (or post-order)</th>
<th>when the minimum support becomes small, the runtime of FIN is lesser compared to Prepost.</th>
<th>scalability of FIN is better than FPgrowth, Efficiency same as Prepost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhi-Hong Deng, Sheng-Long Lv</td>
<td>(1) Construct the POC-tree and identify all subsequent frequent itemsets (2) scan the POC-tree to find all frequent itemsets and their Nodesets (3) mine all frequent itemsets</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CONCLUSIONS

There are several advantages of FP-growth over other approaches: (1) It constructs a highly compact FP-tree, which is usually substantially smaller than the original database and thus saves the costly database scans in the subsequent mining processes. (2) It applies a pattern growth method which avoids costly candidate generation and test by successively concatenating frequent 1-itemset found in the (conditional) FP-trees.

However, for low minimum support, FP-growth becomes more efficient and is faster than PrePost and FIN. PrePost is always faster than FIN. For low minimum support, such as 50%. PrePost+ is faster than FIN, Prepost+ outperforms PrePost by a factor of 3 and FP-growth by a factor of 2. by PrePost+ is less than 1.3 times the memory consumed by FP-growth.

4. When the minimum support becomes small, the runtime of PrePost increase faster than that of FIN and FP-growth.

1. FIN run faster than PrePost and FP-growth on the whole.

2. FIN consumes much less memory than PrePost on dense datasets.

3. It is scalable.

4. Number of candidates that needs to be stored in FIN is much less than that in PrePost+.
Besides adopting N-lists to represent itemsets, PrePost+ employs an efficient pruning strategy named Children–Parent Equivalence pruning to greatly reduce the search space. Although PrePost+ runs fastest, it consumes more memory than FP-growth. Based on Nodesets, an efficient algorithm called FIN is proposed to mine frequent itemsets in databases. The advantage of Nodeset lies in that it encodes each node of a POC-tree with only pre-order (or post-order). This causes that Nodesets consume less memory and are easy to be constructed. The extensive experiments show that the Nodeset structure is efficient and FIN run faster than PrePost and FPgrowth on the whole. Especially, FIN consumes much less memory than PrePost on dense datasets.

REFERENCES

[5] Muhammad Asif, Jamil Ahmed : Analysis of Effectiveness of Apriori and Frequent Pattern Tree Algorithm in Software Engineering Data Mining