Co-occurrence of frequent itemsets in Association Rule Mining

Shubha Chaturvedi, Asha Khatri, Swati Kabra

Abstract-In this paper, we have proposed an algorithm for Association rule mining to find frequent itemsets. This approach is based on Frequent Pattern Tree to find co-occurrence of frequent itemset which will ultimately reduce the scanning of the database resulting in lesser CPU time and is more efficient than the existing FP growth approach. In this approach a relatively small tree is created for each frequent item based on the user defined minimum support so the memory required by data structure is comparatively very low as huge memory is required for storing itemset in data structure in previous approach and also a very simple and non recursive mining process is done.

Index Terms—frequent itemset, FP Growth, Non-recursive, Support.

I. INTRODUCTION

Data Mining is the extraction of the hidden predictive information from large databases. It is the powerful technology to analyze important information stored in large database for large volume of data and is one of the process of knowledge discovery in databases. It is one of the research field that makes problem easy by extracting significant and useful patterns from large collections of data. One of the major task in Data Mining is discovering Association rules and is considered as one of the important tasks, and is one of the research topic from the last few years. Several solutions and algorithms have been proposed for finding sequential and parallel patterns. However, in the existing algorithms for finding certain patterns a huge amount of memory is required for database scanning and the solutions doesn’t exist for large datasets.

II. PROBLEM STATEMENT

The problem of mining association rules over market basket Analysis was introduced in [1]. The problem consists of finding association between items or itemsets in transactional database. The data could be in the form of

Customer care, text documents, and images. Association rules have been shown to be useful for other applications in decision support, recommender system and even supervised classification. Formally, as defined in [2], the problem is stated as follows: Let $I=\{i_1,i_2\ldots i_m\}$ be a set of literals called items and $m$ is considered the dimensionality of the problem.

Let $D$ be a set of transactions, where each transaction $T$ is a set of items such that $TCI$. A unique identifier $TID$ is given to each transaction.

A transaction $T$ is said to contain $X$, a set if items in $I$, if $x \in CT$. An association rule is an implication of the form “$X=>Y$”, where $X \subseteq I, Y \subseteq I$ and $X\cap Y = \emptyset$. An itemset $X$ is said to be frequent if its support $s$ is greater than or equal to a given minimum support threshold. Discovering association rules, however is nothing more than an application for frequent itemset mining

2.1 Previous Work

Mining for frequent itemsets is a canonical task, fundamental for many data mining applications and is an intrinsic part of many other data mining tasks. Mining for frequent itemsets is the major initial phase for discovering association rules. Associative classifiers rely on frequent itemsets. These frequent pattern are also used in some clustering algorithms. Finding frequent items is also an inherent part of many data analysis processes. Many frequent itemset mining algorithms have been reported in the last decade in the literature. The most important, and at the basis of many other approaches, is Apriori [1]. The property that is at the heart of Apriori and forms the foundation of most algorithms simply states that for an itemset to be frequent all its subsets have to be frequent.

This monotone property reduces the candidate itemset space drastically. However, the generation of candidate sets, especially when very long frequent patterns exist, is still very expensive. Moreover, Apriori is heavily I/O bound. Another approach that avoids generating and testing itemsets is FP-Growth [11]. FP-Growth generates, after only two I/O scans, a compact prefix tree representing all sub-transactions with only frequent items. A clever and elegant recursive method mines the tree by creating projections called conditional trees and discovers patterns of all lengths without directly generating candidates the way Apriori does.

The idea of COFI, which we adopt in this paper, is to build projections from the FP-tree each corresponding to sub-transactions of items co-occurring with a given frequent

Manuscript received Aug 15, 2011.

Shubha Chaturvedi, M.Tech Scholar, Department of Computer Science and Engg, Medicaps Institute of Technology, Indore. M.P., India. (e-mail: chaturvedishubha@rediffmail.com).

Asha Khatri, Sr. Lecturer, Department of Computer Science and Engg, Medicaps Institute of Technology, Indore. M.P., India.

Swati Kabra, Lecturer, Department of Computer Science and Engg, Medicaps Institute of Technology, Indore. M.P., India.
item. These trees are built and efficiently mined one at a time making the footprint in memory significantly small.

The COFI algorithm generates candidates using a top-down approach, where its performance shows to be severely affected while mining databases that has potentially long candidate patterns that turns to be not frequent, as COFI needs to generate candidate sub-patterns for all its candidates patterns. We build upon the COFI approach to find the set of frequent patterns but after avoiding generating useless candidates.

2.2 Preliminaries, Motivations and Contribution

The Co-Occurrence Frequent Item Tree algorithm that we are presenting in this paper is based on the core idea of the FP-Growth algorithm proposed by Han et al. in [6]. A compacted tree structure, FP-Tree, is built based on an ordered list of the frequent 1-itemsets present in the transactional database. However, rather than using FP-Growth which recursively builds a large number of relatively large trees called conditional trees [6] from the built FP-tree, we successively build one small tree (called COFI-tree) for each frequent 1-itemset and mine the trees with simple non-recursive traversals. We keep only one such COFI-tree in main memory at a time. The COFI-tree approach is a divide and conquers approach, in which we do not seek to and all frequent patterns at once, but we independently all frequent patterns related to each frequent item in the frequent 1-itemset.

The main differences between our approach and the FP-growth approach are the followings: (1) It build one COFI-tree for each frequent item A. This COFI-tree is non-recursively traversed to generate all frequent patterns related to item A. (2) Only one COFI-tree resides in memory at one time and it is discarded as soon as it is mined to make room for the next COFI-tree. Algorithms like FP-Tree-based depend heavily on the memory size as the memory size plays an important in defining the size of the problem. Memory is not only needed to store the data structure itself, but also to generate recursively in the mining process the set of conditional trees. This phenomenon is often overlooked. As argued by the authors of the algorithm, this is a serious constraint [8]. Other approaches such as in [7], build yet another data structure from which the FP-Tree is generated, thus doubling the need for main memory. The current association rule mining algorithms handle only relatively small sizes with low dimensions. Most of them scale up to only a couple of millions of transactions and a few thousands of dimensions [8, 5].

III. FREQUENT PATTERN TREE

3.1 Frequent Pattern Tree: Design and Construction

The COFI-tree approach we propose consists of two main stages. Stage one is the construction of the Frequent Pattern tree and stage two is the actual mining for these data structures, much like the FP-growth algorithm.

3.2 Construction of the Frequent Pattern Tree

The goal of this stage is to build the compact data structures called Frequent Pattern Tree [6]. This construction is done in two phases, where each phase requires a full I/O scan of the dataset. A first initial scan of the database identifies the frequent 1-itemsets. The goal is to generate an ordered list of frequent items that would be used when building the tree in the second phase. This phase starts by enumerating the items appearing in the transactions. After enumeration these items (i.e. after reading the whole dataset), infrequent items with a support less than the support threshold are weeded out and the remaining frequent items are sorted by their frequency. This list is organized in a table, called header table, where the items and their respective support are stored along with pointers to the first occurrence of the item in the frequent pattern tree. Phase 2 would construct a frequent pattern tree. This phase requires a second complete I/O scan from the dataset. For each transaction read only the set of frequent items present in the header table is collected and sorted in descending order according to their frequency. These sorted transaction items are used in constructing the FP-Trees as follows: for the first item on the sorted transactional dataset, check if it exists as one of the children of the root. If it exists then increment the support for this node. Otherwise, add a new node for this item as a child for the root node with 1 as support. Then, consider the current item node as the newly temporary root and repeat the same procedure with the next item on the sorted transaction. During the process of adding any new item-node to the FP-Tree, a link is maintained between this item-node in the tree and its entry in the header table. The header table holds as one pointer per item that points to the first occurrences of this item in the FP-Tree structure.

For illustration, we use an example with the transactions shown in Table 1. Let the minimum support threshold set to 4. Phase 1 starts by accumulating the support for all items that occur in the transactions. Step 2 of phase 1 removes all non-frequent items, in our example (G, H, I, J, K, L, M, N, O, P, Q and R), leaving only the frequent items (A, B, C, D, E, and F).

Finally all frequent items are sorted according to their support to generate the sorted frequent 1-itemset. This last step ends phase 1 of the COFI-tree algorithm and starts the second phase. In phase 2 the first transaction (A, G, D, C, B) read is filtered to consider only the frequent items that occur in the header table (i.e. A, D, C and B). This frequent list is sorted according to the items' supports (A, B, C and D). This ordered transaction generates the first path of the FP-Tree with all item-node support initially equal to 1. A link is established between each item-node in the tree and its corresponding item entry in the header table.

The same procedure is executed for the second transaction (B, C, H, E, and D), which yields a sorted frequent item list (B, C, D, E) that forms the second path of the FP-Tree.
### Table 1: Transactional Database

<table>
<thead>
<tr>
<th>T No.</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A</td>
</tr>
<tr>
<td>T2</td>
<td>B</td>
</tr>
<tr>
<td>T3</td>
<td>B</td>
</tr>
<tr>
<td>T4</td>
<td>C</td>
</tr>
<tr>
<td>T5</td>
<td>A</td>
</tr>
<tr>
<td>T6</td>
<td>A</td>
</tr>
<tr>
<td>T7</td>
<td>A</td>
</tr>
<tr>
<td>T8</td>
<td>L</td>
</tr>
<tr>
<td>T9</td>
<td>A</td>
</tr>
<tr>
<td>T10</td>
<td>C</td>
</tr>
<tr>
<td>T11</td>
<td>A</td>
</tr>
<tr>
<td>T12</td>
<td>D</td>
</tr>
<tr>
<td>T13</td>
<td>M</td>
</tr>
<tr>
<td>T14</td>
<td>C</td>
</tr>
<tr>
<td>T15</td>
<td>B</td>
</tr>
<tr>
<td>T16</td>
<td>J</td>
</tr>
<tr>
<td>T17</td>
<td>A</td>
</tr>
<tr>
<td>T18</td>
<td>C</td>
</tr>
</tbody>
</table>

Transaction 3 (B, D, E, A, and M) yields the sorted frequent item list (A, B, D, E) that shares the same prefix (A, B) with an existing path on the tree. Item-nodes (A and B) support is incremented by 1 making the support of (A) and (B) equal to 2 and a new sub-path is created with the remaining items on the list (D, E) all with support equal to 1. The same process occurs for all transactions until we build the FP-Tree for the transactions.

**2.3 Co-Occurrence Frequent-Item-trees: Construction and Mining**

Our approach for computing frequencies relies first on building independent relatively small trees for each frequent item in the the header table of the FP-Tree called COFI-trees. Then we mine separately each of the trees as soon as they are built, minimizing the candidacy generation and without building conditional sub-trees recursively. The trees are discarded as soon as mined. At any given time, only one COFI-tree is present in main memory.

**IV. CONSTRUCTION OF THE CO-OCCURRENCE FREQUENT-ITEM-TREES**

The small COFI-trees we build are similar to the conditional FP-trees in general in the sense that they have a header with frequent items and horizontal pointers pointing to succession of nodes containing the same frequent item, and the prefix tree per-se with paths representing sub-transactions.
However, the COFI-trees have bidirectional links in the tree allowing bottom-up scanning as well, and the nodes contain not only the item label and a frequency counter, but also a participation counter as explained later in this section. The COFI-tree for a given frequent item \( x \) contains only nodes labeled with items that are more frequent or as frequent as \( x \).

To illustrate the idea of the COFI-trees, we will explain step by step the process of creating COFI-trees for the FP-Tree of Figure 2. With our example, the first Co-Occurrence Frequent Item tree is built for item \( F \) as it is the least frequent item in the header table. In this tree for \( F \), all frequent items which are more frequent than \( F \) and share transactions with \( F \) participate in building the tree. This can be found by following the chain of item \( F \) in the FP-Tree structure. The \( F \)-COFI-tree starts with the root node containing the item in question, \( F \). For each sub-transaction or branch in the FP-Tree containing item \( F \) with other frequent items that are more frequent than \( F \) which are parent nodes of \( F \), a branch is formed starting from the root node \( F \). The support of this branch is equal to the support of the \( F \) node in its corresponding branch in FP-Tree. If multiple frequent items share the same prefix, they are merged into one branch and a counter for each node of the tree is adjusted accordingly. Figure 3 illustrates all COFI-trees for frequent items of Figure 2. In Figure 3, the rectangle nodes are nodes from the tree with an item label and two counters.

The first counter is a support-count for that node while the second counter, called participation-count, is initialized to 0 and is used by the mining algorithm discussed later, a horizontal link which points to the next node that has the same item-name in the tree, and a bi-directional vertical link that links a child node with its parent, parent with its child. The bi-directional pointers facilitate the mining process by making the traversal of the tree easier. The squares are actually cells from the header table as with the FP-Tree. This is a list made of all frequent items that participate in building the tree structure sorted in ascending order of their global support. Each entry in this list contains the item-name, item-counter, and a pointer to the first node in the tree that has the same item-name.

To explain the COFI-tree building process, we will highlight the building steps for the \( F \)-COFI-tree in Figure 3. Frequent item \( F \) is read from the header table and its first location in the FP-Tree is located using the pointer in the header table. The first location of item \( F \) indicate that it shares a branch with item \( A \), with support = 1 for this branch (following the upper links for this item). To illustrate the idea of the COFI-trees, we will explain step by step the process of creating COFI-trees for the FP-Tree of Figure 2. With our example, the first Co-Occurrence Frequent Item tree is built for item \( F \) as it is the least frequent item in the header table. In this tree for \( F \), all frequent items which are more frequent than \( F \) and share transactions with \( F \) participate in building the tree. This can be found by following the chain of item \( F \) in the FP-Tree structure. The \( F \)-COFI-tree starts with the root node containing the item in question, \( F \). For each sub-transaction or branch in the FP-Tree containing item \( F \) with other frequent items that are more frequent than \( F \) which are parent nodes of \( F \), a branch is formed starting from the root node \( F \). The support of this branch is equal to the support of the \( F \) node in its corresponding branch in FP-Tree. If multiple frequent items share the same prefix, they are merged into one branch and a counter for each node of the tree is adjusted accordingly. Figure 3 illustrates all COFI-trees for frequent items of Figure 2. In Figure 3, the rectangle nodes are nodes from the tree with an item label and two counters.

The first counter is a support-count for that node while the second counter, called participation-count, is initialized to 0 and is used by the mining algorithm discussed later, a horizontal link which points to the next node that has the same item-name in the tree, and a bi-directional vertical link that links a child node with its parent, parent with its child. The bi-directional pointers facilitate the mining process by making the traversal of the tree easier. The squares are actually cells from the header table as with the FP-Tree. This is a list made of all frequent items that participate in building the tree structure sorted in ascending order of their global support. Each entry in this list contains the item-name, item-counter, and a pointer to the first node in the tree that has the same item-name.

To explain the COFI-tree building process, we will highlight the building steps for the \( F \)-COFI-tree in Figure 3. Frequent item \( F \) is read from the header table and its first location in the FP-Tree is located using the pointer in the header table. The first location of item \( F \) indicate that it shares a branch with item \( A \), with support = 1 for this branch (following the upper links for this item). To explain the COFI-tree building process, we will highlight the building steps for the \( F \)-COFI-tree in Figure 3. Frequent item \( F \) is read from the header table and its first location in the FP-Tree is located using the pointer in the header table. The first location of item \( F \) indicate that it shares a branch with item \( A \), with support = 1 for this branch (following the upper links for this item). To explain the COFI-tree building process, we will highlight the building steps for the \( F \)-COFI-tree in Figure 3. Frequent item \( F \) is read from the header table and its first location in the FP-Tree is located using the pointer in the header table. The first location of item \( F \) indicate that it shares a branch with item \( A \), with support = 1 for this branch (following the upper links for this item).
of first entry for each item in the COFI-tree. A link is also made for each node in the tree that points to the next location of the same item in the tree if it exists. The mining process is the last step done on the F-COFI-tree before removing it and creating the next COFI-tree for the next item in the header table.

V. MINING THE COFI-TREES

The COFI-trees of all frequent items are not constructed together. Each tree is built, mined, then discarded before the next COFI-tree is built. This is done for each tree independently with the purpose of finding all frequent k-itemset patterns in which the item on the root of the tree participates. Steps to produce frequent patterns related to the E item for example, are illustrated in Figure 4. From each branch of the tree, using the support-count and the participation-count, candidate frequent patterns are identified and stored temporarily in a list. The non-frequent ones are discarded at the end when all branches are processed. The mining process for the E-COFI-tree starts from the most locally frequent item in the header table of the tree, which is item B. Item B exists in three branches in the E-COFI-tree which are (B:1, C:1, D:5 and E:8), (B:4, D:5, and E:8) and (B:1, and E:8). The frequency of each branch is the frequency of the first item in the branch minus the participation value of the same node. Item B in the first branch has a frequency value of 1 and participation value of 0 which makes the first pattern EDB frequency equals to 1. The participation values for all nodes in this branch are incremented by 1, which is the frequency of this pattern. In the first pattern EDB: 1, we need to generate all sub-patterns that item E participates in which are ED:1 EB:1 and EDB:1. The second branch that has B generates the pattern EDB: 4 as the frequency of B on this branch is 4 and its participation value is equal to 0. All participation values on these nodes are incremented by 4. All patterns already exist with support value equals to 1, and only updating their support value is needed to make it equal to 5. The last branch EB:1 will generate only one pattern which is EB:1, and consequently its value will be updated to become 6.

The second locally frequent item in this tree, D* exists in one branch (D: 5 and E: 8) with participation value of 5 for the D node. Since the participation value for this node equals to its support value, then no patterns can be generated from this node. Finally all non-frequent patterns are omitted leaving us with only frequent patterns that item E participates in which are ED:5, EB:6 and EDB:5. The COFI-tree of Item E can be removed at this time and another tree can be generated and tested to produce all the frequent patterns related to the root node. The same process is executed to generate the frequent patterns. The D-COFI-tree is created after the E-COFI-tree. Mining this tree generates the following frequent patterns: DB:8 ,DA:5, and DBA:5. C-COFI-tree generates one frequent pattern which is CA: 6. Finally, the B-COFI-tree is created and the frequent pattern BA: 6 is generated.

The Co-occurrence trees of all frequent items are not constructed together. Each tree is built, mined and then discarded before the next tree is built. The mining process is done for each tree independently with the purpose of finding all frequent k-itemset Patterns in which the item on the root of the tree participates. Steps to produce frequent patterns related to the E item for example as the F- co-occurrence tree will not be mined based on the pruning results we found on the previous step are illustrated Form each branch of the tree using the support –count and participation –count ,candidate frequent patterns are identified and stored temporarily. The non-frequent ones are discarded at the end when all branches are processed. The mining process for the E- co-occurrence tree starts from the most locally frequent item of the tree ,which is item B .Item B exists in two branches in tree which are (B:5,D:5 and E:8) and (B:1 and E:8).The frequency of each branch is the frequency of the first item in the branch minus the participation value of the same node. Item B in the first branch has the frequency value of 5 and participation value of 0 which makes the first pattern EDB frequency equals to 5. The participation values for all nodes in this branch are incremented by 5 ,which is the frequency of this pattern .In the first pattern EDB:5 we need to generate all sub patterns that item E participates in , which are ED:5 ,EB:5 and EDB:5 .The second branch that has B generates the pattern EB:1. EB already exists and its counter is adjusted to become 6. The COFI tree of Item E can be removed at this time and another tree can be generated and tested to produce all the frequent patterns related to the root node. The same process is executed to generate the frequent patterns. The D co-occurrence tree is created after the E co-occurrence tree. Mining this tree generates the following frequent patterns :DBA:5 ,DA:5 and DB:8 .The same process occurs for the remaining trees that would produce AC:6 for the C-occurrence tree and BA:6 for the B- co-occurrence tree.

5.1 Architecture of Co-occurrence Frequent Itemsets

The approach we have used for finding frequent itemsets are more vulnerable then existing FP Growth algorithm because the database scanning is only done once in the co-occurrence frequent itemset tree and trees are generated for each and every frequent itemsets.

The dataset which was used for the analysis is shown in the figure in which the various elements are shown in which finding of frequent itemsets are done by support count through which selected items are taken into consideration and rest of the items which was not used frequently are removed from the co-occurrence frequent item tree. The dataset which was used for the analysis is shown in which all the frequent items are counted according to which the frequent pattern tree is created and the co-occurrence tree for each frequent itemset is created in which a support count is used which is the value of the items that are found to the minimum level and accordingly are set and compared to obtain all the frequent items which are having the value equals to them or greater.
than that and for each frequent items co-occurrence tree are generated and after its creation the co-occurrence tree is removed from the memory and room is made for the next co-occurrence tree.

![Data Set Used](image)

**Fig 5.1** Data used for the analysis

### 5.2 Algorithm for the co-occurrence of frequent itemsets

1. Creating and pruning the itemsets.
2. Only put NULL on the header of the FP tree.
3. Put maximum occurrence value in header of FP tree.(If there are still frequent item)
4. Count the frequency of all items and arrange according to their occurrence.
5. Remove all non-locally frequent itemsets
6. It will create the FP tree
7. Next create a root node for tree with frequency count =0.
8. If the frequency count = frequency node and participation count =0 for all nodes in the Path else.
9. Create a new frequency count of the path.
10. While there are still nodes do
   a. Set remaining nodes to the root.
   b. Node A frequency is count node.
11. Generate all candidate items from 10a.
12. Arrange all items to the accordance of occurrence.
13. Repeat steps 10-13 until all the frequent items are not found.
14. Based on support threshold remove all non- frequent items from item set.
15. End

### VI COMPARISON OF FP TREE AND CO-OCCURRENCE TREE

The algorithm we have proposed in this work is based on the core idea of FP- Growth algorithm which was proposed by Hanental. A compacted tree structure FP tree is created by an ordered list of frequent 1 –item set present on the transaction database. By using FP-Growth which builds large number of relatively large trees called conditional trees from the FP tree we successively build on e small tree i.e. co –occurrence tree for each frequent 1 –item set and mine tree with non recursive traversals. There is only need of one tree to be kept in main memory.

In this approach divide and conquer approach, in which do not need to find all frequent patterns at once, but independently find all frequent patterns related to each frequent item in frequent item set. One of the major differentiating features are:

1. We build only one co-occurrence tree for each frequent item A and this tree is no –recursively traversed to generate all frequent patterns related to item A.
2. Only on e tree resides in memory at one time and is discard as soon as it is mined to make space for the next tree.

The memory size plays the major role where the memory is itself required to store the data structure itself, but also to generate recursively in the mining process the set of conditional trees other approach build yet another data structure from which the the FP tree is generated thus doubling the need for main memory.

Generally, in association rule mining algorithms handle only relatively small sizes with low dimensions. Most of them scale up to only a couple of millions of transactions and a few thousands of dimension. None of the existing algorithms scales to millions of transactions with thousands of dimensions, in which each transaction has at an average of at least a couple of dozen items.

In this approach the goal for the first database scan is to find frequency of each item in the transactional database and frequencies are stored in data structure called candidate items. Each entry of this candidate item is a structure made of two integers representing the item and frequency then all frequent items are stored in special file called F1 items. The data structure is sorted in descending order based on the frequency of each item.

The second scan starts by eliminating all non frequent items from each transaction read and then sort this transaction based on the frequency of each frequent item. This process occurred in the sort –transaction method. The FP tree is built based on the sub transaction made of the frequent items. The FP tree data structure is a tree of n children which has been used to create each node of this tree is created between each node and its first child and the other link is created between each node and other link is maintained to create linked list of all children of the same node. This linked list is built ordered base don the frequency of each item. The header list is maintained using the structure. After building the FP tree we start building the first co-occurrence tree by selecting the item with least frequency from the frequent list. A scan is made of the FP tree starting from the linked list of this item to find frequency of other items with respect to this item. After that, the co-occurrence tree is created based on only locally frequent items. Finally the frequent patterns are generated and stored in the frequent pattern tree structure. All nodes that have support greater or equal than the given support present a frequent pattern. The co-occurrence tree and Frequent pattern tree are
removed from memory and the next co–occurrence tree is created until we mine all frequent trees.

One of the improvement is the fact that the participation counter was also added to the header table of the co-occurrence tree cumulates the participation of the item in all patterns already discovered, in the current co-occurrence tree. The difference between the participation in the node and the participation in the header is that the counter in the node counts and the participation of the node item in all paths where the node appears, while the new counter in the co–occurrence tree header counts the participation of the node item in all the paths where the node appears. Its major advantage is that counter looks ahead to see if all nodes of a specific item have already been traversed or not to reduce the unneeded scans.

VI. EXPERIMENTAL RESULT

To find a scalable algorithm for association rule mining in extremely large database is the main goal for the research. To reach at this goal, a new algorithm is proposed which is FP tree based which identifies the main problem of the FP growth in which recursive creation and mining of many conditional pattern trees which are equal in number to the distinct frequent patterns generated we have replaced this step by creating one Co-occurrence tree for each frequent item. A simple non recursive mining process is applied to generate all frequent patterns related to the tested co–occurrence tree.

To study the co-occurrence tree mining strategies we have conducted various experiments on a variety of data sets comparing with the well known approach FP-Growth in which our algorithm is very efficient.

The algorithm is implemented and used for discovering frequent itemsets based on the minimum support for datasets which are containing the transactions of 100, 500, 1000, 5000, attributes or items. The time required to discover all frequent item sets is shown Figures.

The proposed algorithm is superior to the FP tree algorithm in following ways:
1. Scans database only once.
2. No sorting of each item of the transaction.
3. No repeatedly searching the header table for maintaining link

VII. COMPARISON RESULT

Following figure shows the comparision result analysis

Fig 7.1 Comparative chart for FP tree and Co-occurrence tree

REFERENCES