International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 17 (2015) pp 38445-38452
© Research India Publications. http://www.ripublication.com

A Single Scan Prefix-Tree Construction Approach For Maximal Pattern Mining

R .Vishnu Priya

School Computing Science and Engineering
VIT University Chennai Campus
{vissaru@yahoo.co.in}

ABSTRACT

Mining maximal patterns from databases is useful for knowledge discovery. In this paper, we propose a novel tree structure called Single Scan FP-tree (SFP-tree), which scan entire transactions only once and the tree is constructed along with its corresponding item list. The items in the item list are sorted by respective items frequency. Then, constructed tree is dynamically reorganized for those items satisfy the minimum support based on the sorted items in the item list. FPmax approach is used for constructing Maximal Single Scan FP-tree from SFP-tree. We have evaluated the performance of the MSFP-tree on benchmark databases such as CHESS, MUSHROOM CONNECT_4, PUMSB_STAR, T10I4D100K and T40I0D100K. It is observed that the time taken for extracting maximal patterns from the MSFP-tree is encouraging compared to the conventional MFI-tree.

Keywords: Maximal Pattern Mining; Association Rule Mining; SFP-tree; MSFP-tree; MFI-tree

INTRODUCTION

The frequent item sets present in the database have been used for finding and extracting interesting patterns for knowledge discovery. Among various data mining techniques, the association rule mining is considered as one of the popular problems. One of the primary steps in association rule mining is to identify the frequent patterns. Agrawal et. al. have introduced the frequent item set problem and later the field has received attention from researchers both in academia and industries. In Apriori algorithm, mining frequent item set requires a large memory space that scans the database multiple times. Han et al. (2004), proposed the FP-tree, which avoid the multiple scans by constructing highly compact frequent descending tree structure and it is built with two scans. It is observed that the number of times for scanning the database increases the mining time considerably. Meanwhile, mining subsets for a large frequent item set of length '*l*', then almost '2^l 'item sets are generated and leads to increase in mining time. This can be avoided from the facts that "any subset of the frequent item set must be frequent" and it is sufficient to find only the maximal frequent item. In this paper, we modify the MFI-tree and proposed a novel tree structure, which scans the transactional database only once. The proposed tree is constructed only for those items satisfies minimum support by reorganizing the nodes in each branch from already constructed tree based on item sorted list. The maximal item sets are mined from the proposed tree using FPmax technique. By this approach, modified tree takes less time for maximal pattern mining compared to the conventional MFI-tree.

We organize the rest of the paper as follows. We discuss the related work in the next section. In Section 3, the procedure for construction and reorganizing of the SFP-tree is presented. We summarize the maximal pattern mining with suitable example in Section 4. We present the experimental result in Section 5 and conclude the paper in the last section of the paper.

RELATED WORKS

Various algorithms have been proposed for improving the performance of Apriori-based algorithms such as hashing technique proposed by Park et al. (1995), partitioning technique presented by Savasere et al. (1995), Toivonen (1996) given a sampling approach, dynamic itemset counting by Brin et al. (1997), Cheung et al. (1996) designed the incremental. The performances of these approaches are found to be better compared to Apriori based to mine the frequent patterns. The issues of apriori based algorithms have been handled using tree based approaches and the prefix tree structure algorithms, namely, FP-tree, Light Partial Support-FP-Forest (Chen et al., 2008) algorithms mines the frequent pattern with two scans using pattern growth approach without generating the candidate sets. In case of LPS-FP-Trees algorithm, it uses two data structures, namely LPS-tree and LPS-forest for mining frequent patterns and in the second scan, the LPS-FP-Forest is built. Since, LPS-FP-Forest is unidirectional, the pointer of the each node in the LPS-FP-Forest is linked to its parent. Even though, these algorithms mines frequent patterns with two scans from larger size database, efficiency of mining is not encouraging. This is due to the fact that, the number of scan increases the time to mine patterns. To improve the efficiency, Yong Qiu et al. (2006) have designed the Quick FP-tree Constructing algorithm, and the tree is built with single scan. The given transactional database is divided into *'n'* parts. During the first scan, the transactions in each part is scanned to find local item lists, that consists of items arranged in descending order based on its local frequency of occurrence. For each part, the local FP-tree is built based on the corresponding local item list. The support of each item is count from all the local FP-tree and items are arranged in descending order based on its frequency. Each branch of the local FP-tree is removed and sorted based on descending order of items. The sorted items are inserted as a branch into the new tree. This process continues till entire local FP-trees are merged. This algorithm uses pattern growth approach to mine the patterns. Even though, the tree is build with single scan, this algorithm requires space to hold local FP-trees and consumes time for merging the trees. The frequent patterns can also be retrieved using different data structure and Pei et al. (2001) has proposed H-mine (Hyper-Structure Mining) algorithm, which is the combination of array and hyper links. To mine the frequent patterns, the new conditional databases such as link adjustment are constructed and the construction cost is low compared to FP-tree. However, H-mine algorithm has a constraint such that the items can be ordered only by a fixed order, since the hyper structure will not change except the hyper link. In addition, the hyper structure is not found to be efficient for dense database and the transaction cost is additional for unfiltered items. In some cases, it is also necessary to process and mine the data format. Zake et al. (1997) has proposed the Equivalent Class Transformation algorithm, which use vertical data format for mining. This is in contrast to the working principle of both Apriori based and FP-tree based algorithm. Both of these algorithms mine the frequent patterns from the horizontal formatted data. It uses lattice theory to represent the database items. However, it requires an additional conversion step. In addition, it also uses a Boolean power set lattice, which uses large space to store the labels and lists of tid. While this algorithm is found to be competent for large item sets, the performance is not encouraging for small item set.

It is noticed from the above discussion that most of the algorithms require larger time for mining the frequent patterns from transactional database and to decrease the mining time, it is enough to find the maximal patterns. Thus, it is imperative that an algorithm is required for mining maximal patterns swiftly. In this paper, we proposed a tree, which scan the transactional database only once and the time for maximal pattern mining is found to be low compared to some of the proposed algorithms.

THE SINGLE SCAN FREQUENT PATTERN – TREE

In our approach, the tree is constructed in three stages such as construction phase, finding local sorted item list phase and reorganizing phase. We explain each phase in following sub-sections.

Construction Phase

The transactions are inserted as a branch into the construction tree one by one based on the order of the items in each transaction and its corresponding item list is generated. This is being done to mine maximal patterns by performing only one scan of the database. Since, the numbers of scans decrease, the time to mine maximal patterns are also decreases. In Table 1, we have presented the sample transactions considered for mining the maximal pattern. The TID is the transaction id and Ii is the item in a particular transaction.

Table 1. Sample Transactions

TID	Transactions
1	I_2 I_1 I_6 I_7 I_8
2	I_4 I_5 I_1
3	I_2 I_1 I_6 I_7 I_4
4	I_1 I_2 I_3
5	I_1 I_4 I_7
6	I_2 I_4 I_7 I_3 I_9

The constructed tree will have the item list along with its occurrence. The items with lower count value than the support threshold are pruned. The remaining items are sorted in descending order of their frequency of occurence and stored in the list called Item sorted list (I_{sort}). Then, we dynamically reorganize the constructed tree based on I_{sort}. Below, we illustrate the process of tree construction with a suitable example. Initially, an empty node is created with null as the label. The first transaction in Table 1, which is <I_2 I_1 I_6 I_7 I_8>, scanned once. Each item with its frequency of occurrence are inserted as a branch into the tree. Let the inserted branch in the tree is <I_2:1 I_1:1 I_6:1 I_7:1 I_8:1>. After inserting the first transaction, the next transaction < I_4 I_5 I_1> is scan once. The first item in this transaction is checked with children of the root node. In this case, the first item of transaction is I_4 and child of root node is I_2. While, if both the items are equal, the count of the node is incremented in that branch by one and the current item is not inserted as a node and the process continued to check the next item. Otherwise, the items in the transaction are inserted as a branch or path with its support value as 1. In this case $I_4 \neq I_2$, hence it is inserted as a new branch. Let the second inserted branch in the tree is <I_4:1 I_5:1 I_1:1>. In the same way, all the remaining transactions are inserted into the tree and its item list consists of the items along with its total number of occurrence in the database is generated. The items with the occurrences less than the support threshold are pruned from the list and remaining items are sorted in descending order of their frequency and stored into the list called I_{sort}. In Figure 1, we show the constructed tree and the corresponding I_{sort} with items satisfy the minimum support.

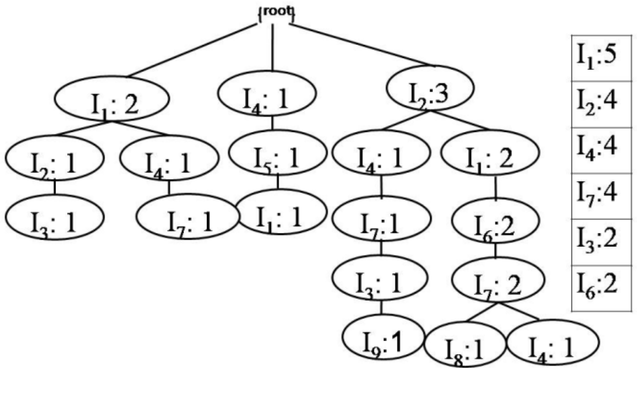


Figure 1: The Constructed Tree with Item Sorted List

However, at this stage, items are not inserted in the descending order of its frequency and thus the constructed tree is similar to Frequent-Independent tree with its I_{sort}.
Below, we present the algorithm for generating the construction tree along with its I_{sort}.

Algorithm for tree construction

Input: Transaction database DB, Total number of transactions N.
Output: Constructed Tree, Item Sorted List I_{sort}.

Method:

```
1.  Read DB, N and set j=1.
2.  Create an empty node for the tree "T" with null as the label.
3.  while (j<=N)
4.  {
5.  Examine the transaction t_j from DB
6.  Let the transaction be [i/I], where i is the foremost item and I is the remaining items in the transaction,then Call construct_tree([i/I],T).
7.  Function construct_tree([i/I],T)
8.  {
9.  if T has no child then
10. insert transaction [i/I] as the branch of T
11. else
12. {
13. if T has a child C such that C.item_name=I.item_name then
    increase C's count by 1 and then child of C is check(equal or not equal) with subsequently item in I.
14. else
15. {
16. Create a new_node I;
17. I's count=1; I's predecessor be connected to I;
18. I's node_link be connected to rest of the items in the transaction
19. }
20. }
21. End Function
22. } Increment j variable.
23. }
24. Find occurrences of each item from the constructed tree and pruned those items not satisfied minimum support. Then items are arranged based on occurrences are stored into I_sort.
```

Once, the tree is constructed by using all the transaction, it is necessary to reorganize the constructed tree in such a way that the nodes with frequency of occurrence will be on the top and the one with less frequency of occurrence will be in the bottom of the tree. This is being carried out for effective maximal pattern mining. The reorganizing phase of the SFP-tree is explained below.

Reorganizing Phase

The constructed tree depicted in Fig. 1, consists of many branches and many paths. In reorganizing phase, the nodes in each branch of the tree will be reorganized based on the order of items in I_{sort}. The reorganizing process can be performed using various sorting techniques, and in our approach the merge sorting technique is used. Each path of the branch is removed from the tree sorted based on I_{sort} and inserted back as a branch into the tree. This reorganizing process continues till the entire branches in the constructed tree are reorganized based on I_{sort}. Now we illustrate the reorganizing phase with a suitable example. The unsorted paths are reorganized into sorted paths. The first path of the second branch of I_2 from the constructed tree <I_2:1 I_1:1 I_6:1 I_7:1 I_8:1> is removed. The removed path is unsorted and is stored in a temporary array. The items stored in the temporary array are sorted based on the items arranged in I_{sort} using merge sort technique. The sorted path along with its corresponding frequency < I_1:1 I_2:1 I_7:1 I_6:1> is inserted back as a branch into the reorganized tree and is shown in Figure 2. We can observe that an item I_8 is pruned from the sorted path, since its frequency of occurrence is less than the specified support threshold.

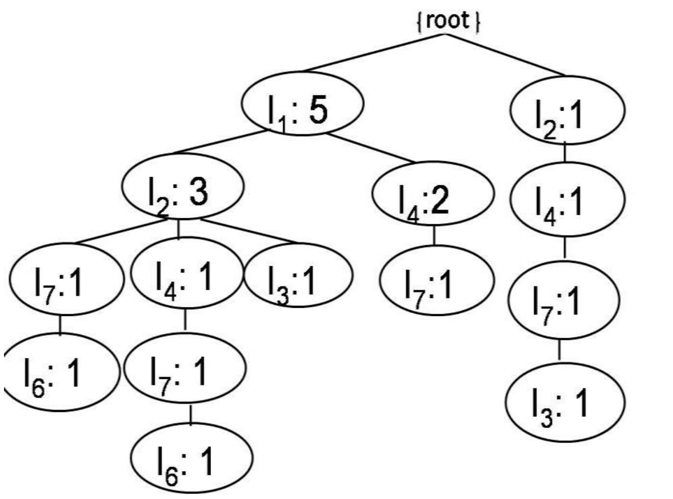


Figure 2: The Reorganized Tree (SFP-tree)

After insertion, the next unsorted branch is removed from the tree, which is < I_4:1 I_5:1 I_1:1> (Figure 1) and stored in temporary array. The items in temporary array are sorted based on I_{sort} and the first item <I_1:1> in I_{sort} is checked with children of the root node, which is <I_1:1> and both are equal. Thus, the count of the item in the branch is incremented by count of the item in temporary array. After checking the first item, the second item <I_4:1> is checked with second node of the branch <I_2:1>. In this case, both are not equal and <I_4:1> is linked as a child of <I_1:5>. The rest of the items in temp array is linked as the child of <I_4:1> as shown in Figure 2.

This process is continued till all the branches in constructed tree are inserted into reorganized tree. While constructing the reorganized tree, the tree is constructed only for all items that satisfy the minimum support. In this case, the minimum support given by the user is "2". Below, we present the algorithm for reorganizing phase.

Algorithm for tree reorganizing

Input: I_{sort} and Constructed Tree T.
Output: Reorganize Tree.

Method:

```
1.  Read $I_{sort}$ and Constructed Tree T
2.  for each branch $B_i$ in T
3.  for each unprocessed path $P_j$ in $B_i$
4.  Remove the unprocessed path $P_j$ from T
5.  If each node in $P_j$ is sorted based on $I_{sort}$
6.  {for each node $n_k$ in $P_j$ from the $leaf_p$ node
7.  for each sub_path S from $n_k$ to $leaf_c$ with $leaf_c$ not equal to $leaf_p$
8.  If frequency of each node in S from $n_k$ to $leaf_c$ are less than frequency of node $n_k$
9.  P= path from the root to $leaf_c$
10. Execute the step from 15 - 18
11. else
12. P= S from $n_k$ to $leaf_c$
13. If P is sorted path
14. Execute the step from 5 - 13 }
15. else
16. {Reduce the frequency of each node in $P_j$ to the frequency of leaf node
17. Stored each node in P into the temp array
18. Insert sorted nodes as a branch into reorganized tree R}
19. Terminate while the entire path in tree T is inserted into R based on $I_{sort}$.
```

In the above algorithm, we retrieves constructed tree and I_{sort} for reorganizing the constructed tree based on I_{sort}. Each branch of the constructed tree is checked, if the branch is unsorted then the branch is sorted using merge sort technique. Otherwise, we skip the sort operation for that path and also spread the path information to all branching nodes in the same path of the whole branch. Hence, for the remaining sub-paths in the same branch, it is sufficient to check only the node from leaf to the branching node of the sub-path. This process get terminate, while all the branches in the constructed tree are reorganized based on I_{sort}.

MINING MAXIMAL PATTERNS USING SFP-TREE

One of the important tasks in the association rule mining is maximal pattern mining. In this section, we illustrate maximal patterns mining from the SFP-tree using FPmax approach. At first, the conditional pattern base is found for all the items in the I_{sort}. Each and every item from the bottom in I_{sort} is considered for finding the conditional pattern base. While finding the conditional pattern base of the current item, we retrieve the set of prefix paths from the root of the reorganized tree till the current item. Each item present in the prefix path carries the occurrence of the corresponding current item in that path. From the conditional pattern base of the current item, a small $Ilist_{cur}$ is generated. $Ilist_{cur}$ consists of the items that satisfy the support threshold. After $Ilist_{cur}$ is created, a new conditional tree is constructed from the conditional pattern base of the current item by eliminating all infrequent items, which are not found in $Ilist_{cur}$. Similarly the conditional trees are found for all the items in the I_{sort}. The procedure for maximal patterns mining is explained with a suitable example. For mining maximal patterns of each item, we start travelling from bottom to top in I_{sort}, which consist of items I_1:5 I_2:4 I_4:4 I_7:4 I_3:2 I_6:2. Since the last item in I_{sort} is I_6:2, the set of prefix paths occurring along with I_6 is retrieved for mining. Hence, we found the two prefix paths namely I_1:5 I_2:3 I_7:1 I_6:1 and I_1:5 I_2:3 I_4:1 I_7:1 I_6:1 that are co-occurring with I_6. I_1, I_2 and I_7 are the items in the first prefix path with the occurrence of 5, 3 and 1 respectively. We already stated that "each items present in the prefix path carries the occurrence of the corresponding item I_6 in that path". Hence, we can find that the occurrence of I_6 in the first path is "1". Thus, the occurrence of each item in the first prefix path became I_1 I_2 I_7:1. Similarly, the second prefix path becomes I_1 I_2 I_4 I_7:1. Now, the conditional pattern base for I_6 consists of two prefix paths such as {(I_1 I_2 I_7:1), (I_1 I_2 I_4 I_7:1)}. After finding the conditional pattern base for I_6, the $Ilist_{I6}$ is created from the conditional pattern base of I_6. Let the items in $Ilist_{I6}$ are I_1:2 I_2:2 I_4:1 I_7:2. In the $Ilist_{I6}$, item I_4 is infrequent having a count less than the minimum support value (i.e.) 2 given by the user. Hence, the item I_4 is eliminated while constructing the conditional tree for I_6. After mining all the patterns suffixing I_6 is completed, we move to the next item in I_{sort} which is "I_3". To mine the maximal patterns for I_3, the same procedure is followed. While finding the conditional pattern base for I_7, we get four prefix paths (Figure 2). One among the four paths is <I_1 I_2 I_7 I_6>, which consists of item I_6 and appears together with the item I_7. However, while finding the conditional pattern base for I_7, we have not included item I_6 in the prefix path. This is due to the fact that we have already analyzed the patterns involving I_6. The same procedure is followed to find the conditional tree for all items in I_{sort}. In Table 2, we have presented the conditional pattern base and its corresponding conditional SFP-tree for all frequent items in I_{sort}.

Table 2. Mined Patterns

item	conditional pattern base	Conditional SFP-tree
I_6:2	<I_1 I_2 I_4 I_7:1> <I_1 I_2 I_7:1>	<I_1:2 I_2:2 I_7:2>
I_3:2	<I_1 I_2:1> <I_2 I_4 I_7:1>	<I_2:2>
I_7:4	<I_1 I_2 I_4:1> <I_1 I_2:1> <I_1 I_4:1> <I_2 I_4:1>	<I_1:3 I_2:3 I_4:3>
I_4:4	<I_1 I_2:1> <I_1:2> <I_2:1>	<I_1:3 I_2:2>
I_2:4	<I_1:3>	<I_1:3>
I_1:5	NULL	NULL

The Maximal SFP-tree is constructed using conditional SFP-tree, which is presented in Table. 2. From the conditional tree of 'I_6' {I_1:2, I_2:2, I_7:2}, we acquire {I_1, I_2, I_7} as frequent item set. Since, {I_1, I_2, I_7} is a maximal item set, it is inserted as a branch into the MSFP-tree and is shown in Fig. 3a. The next frequent pattern from the conditional tree of 'I_3' is {I_2}. The item set of 'I_3' is not inserted into the MSFP-tree, since the one of the subsets of {I_1, I_2, I_7} is {I_2}. For the item 'I_7', the conditional item sets is {I_1 I_2 I_4}. Since, {I_1, I_2} is one of the subset of {I_1 I_2 I_7}, these item sets are inserted directly as a path into the tree and is depicted in Fig. 3b. The conditional item sets from the conditional tree for items'I_4', 'I_2' and 'I_1' are {I_1, I_2}, {I_1}, {Null} respectively. We can determine that these item sets are already inserted into MSFP-tree as the branches and these item set are not inserted into the MSFP-tree

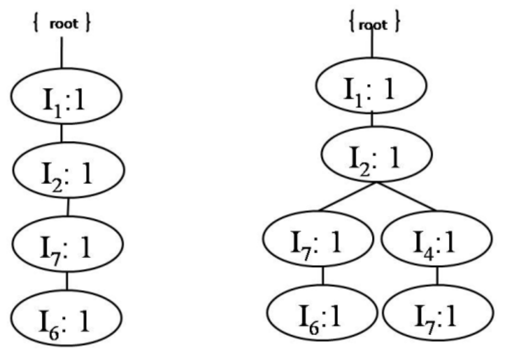


Figure 3: After insertion of the maximal patterns (a) {I_1, I_2, I_7, I_6} (b) {I_1, I_2, I_4, I_7} into the MSFP-tree

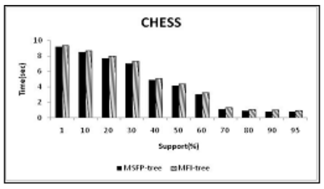
EXPERIMENTAL RESULTS

In this section, we have shown the runtime of the MSFP-tree for mining maximal frequent patterns from the benchmark databases. The synthetic databases developed by IBM Almaden Quest research group and real databases from the UCI Machine Learning Repository have been used for our experimental purpose. The experiments have been carried out using three dense databases such as CHESS, MUSHROOM and CONNECT-4 and three sparse databases such as T10I4D100K, T40I10D100K and PUSMB_STAR. In general, the minimum support threshold values for mining maximal patterns from the dense database is high compared to sparse databases, since, each transaction of dense database contains many items per transaction and only a few distinct items. In contrast, the sparse database represents with few items and contains many distinct items. The real databases used for our experiment are CHESS, MUSHROOM and CONNECT-4, while synthetic database are T10I4D100K, T40I10D100K. The experimental results of the proposed tree are presented in two subsections. In the first subsection, we have presented the performance evaluation of the MSFP-tree in terms of runtime. The runtime includes the tree construction time, rearranging time and time for mining the maximal pattern. The scalability of the MSFP-tree is also presented in the next subsection. We have compared the performance of the MSFP-tree with the MFI-tree.

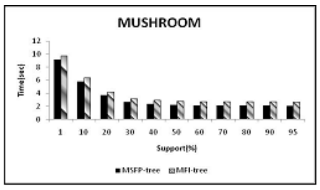
Performance analysis of Runtime

The runtime for mining maximal patterns from dense and sparse databases using both the MSFP-tree and MFI-tree is presented. In Figures 4(a-e), x-axis shows the support threshold values (%) and y-axis shows the runtime in seconds. The runtime of the MSFP-tree and MFI-tree is shown with different minimum support values for various databases. In Figure 4a, we show the time for extracting maximal patterns from the MSFP-tree and MFI-tree using CHESS database. The CHESS database contains 3196 transactions and 75 items with 0.34MB as size. The maximum transaction size and average transaction length are 29 and 10.10 respectively. It is observed from Figure 4a that the MFI-tree takes slightly more time compared to the MSFP-tree for mining maximal patterns. For example, the MSFP-tree takes 9.165 and 0.781 seconds for the support thresholds of 1% and 95% respectively. However, in case of MFI-tree, the time taken is 9.369 and 0.984 seconds. Similarly, the total number of transactions and items in the MUSHROOM database are 8124 and 119 respectively with 0.83MB as size. The maximum length and average size of each transaction in MUSHROOM database is 23 and 23.00 respectively. While mining with MUSHROOM database, the MSFP-tree and the MFI-tree take larger time for lower support values and lesser time for higher support values. The runtime for the support values of 1% and 95% using MSFP-tree is 9.153 and 2.109 seconds respectively. The MFI-tree has taken 9.748 and 2.687 seconds respectively. This is due to the fact that for lower support values the maximal patterns to mine are large in number and it requires higher mining time, which increases the rate of change in overall runtime. In contrary, for higher minimum support, the maximal patterns to mine are low in number and it requires low mining time. The last dense database we have used for the experiment is CONNECT-4, which contains 67,557 transactions and 129 items with 8.82 sizes. Each transaction in CONNECT-4 is with 43 items and average transaction length is 43.00. The runtime for mining using MSFP-tree and MFI-tree for the minimum support of 30% is 63.274 and 71.21 seconds and for 70% it is 37.681 and 45.305 seconds respectively. It is noticed from Figures 4(a-c), that the run time taken by the MSFP-tree is low compared to MFI-tree. Since, MFI-tree has constructed the frequent-descending prefix tree structure with two scans, the runtime is found to be higher.

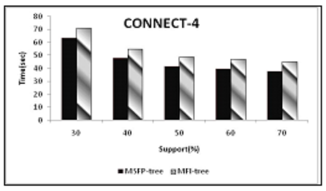
Further, the performance comparisons of the proposed method using sparse databases are shown in Figures 4(d -f). The T10I4D100K is the first sparse database used in our experiment with 3.93MB size. It contains 1, 00,000 transactions and 870 items. The maximum length of the transaction consists of 29 items and average length is 10.10 items respectively. To mine maximal patterns from T10I4D100K database, for the lowest minimum support value of 0.01% is 385.179 and for the highest minimum support value of 3% is 110.342 seconds. However, MFI-tree takes 414.196 and 140.505 seconds respectively. The second sparse database we have used is T40I10D100K with 1, 00,000 transactions and average transaction size is 39.61. The proposed algorithm mines pattern from T40I10D100K in 3062.629 and 2084.923 seconds and from the MFI-tree in 3161.699 and 2193.497 seconds for the support values of 0.1% and 3% respectively. The last sparse database used is PUMSB_STAR with 10.70MB. It consists of 49, 046 transactions and 2088 items. The maximum length of each transaction in PUMSB_STAR is 63 and average length of 50.48 respectively. While, in PUMSB_STAR database, the runtime of the MSFP-tree for the supports 10% and 60% is 712.823 and 748.331 seconds and MFI-tree takes 895.254 and 709.434 seconds respectively. From the experimental result, it is observed that the performance of the proposed approach is encouraging. This performance enhance is due to the fact that the proposed MSFP-tree is built with single scan.



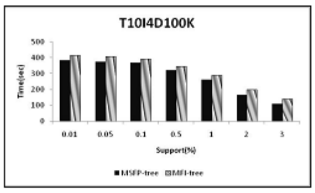
(a)



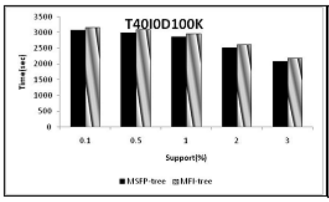
(b)



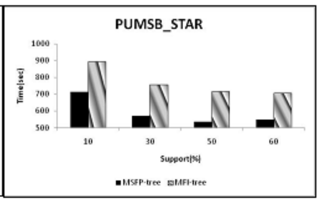
(c)



(d)



(e)



(f)

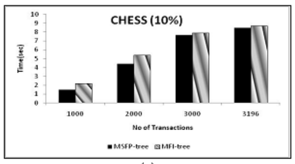
Figure 4: Runtime for different databases. Extracting maximal patterns from (a) CHESS database (b) MUSHROOM database (c) CONNECT-4 database (d) T10I4D100K database (e) T40I10DI00K database (f) PUMSB_STAR data bases

Scalability of the MSFP-tree

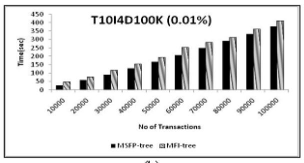
In this subsection, we have shown that scalability of the proposed method by varying the number of transactions as well as support threshold values and mined maximal patterns from both dense and sparse databases. The experimental results for scalability are discussed below.

Scalability with the number of transactions

To measure the scalability by varying the number of transactions, we have selected the CHESS and T10I4D100K databases. We split the CHESS database into four equal parts with 1000 transactions in each part. The SFP-tree is built for 1000 transactions by following the same procedure presented in the construction phase and the maximal patterns are mined. While, the database is updated with next 1000 transactions, the old tree is not used and the entire 2000 transactions are rescanned to construct the SFP-tree by again performing the construction and reorganizing process. In Figure 5a, we have presented the experimental result for scalability with various transaction sizes for CHESS database. The x-axis shows the number of transactions in each parts and y-axis shows the runtime, which includes time for tree construction and reorganizing of the SFP-tree. We have fixed the minimum support value=10% for mining maximal patterns. From the graph, we can observe that as the number of transactions increases, the time to mine maximal patterns increases. This is due to the fact that while the number of transactions increases constantly, the maximal patterns to mine are more and thus the runtime increases linearly. In Figure 5b, we have presented the experimental results for T10I4D100K. We split the T10I4D100K database into ten equal parts consisting 10,000 transactions and the support value is fixed to 0.01%. In Figure 5a, for various transactions, the MSFP-tree takes less time compared to MFI-tree and the similar case for T10I4D100K database also. This performance enhancement is due to the fact that the SFP-tree construction time is less compared to the FP-tree construction time.



(a)



(b)

Figure 5: Scalability for various transaction size (a) CHESS database (b) T10I4D100K database

Scalability with minimum support values

Similar to the above experiment, we have conducted experiment for different minimum support threshold. We have presented experimental results in Figure 6 for CONNECT-4 and PUMSB_STAR databases. We mine the maximal patterns by changing support values from 30% to 70% for CONNECT-4 database and from 10% to 60% for PUMSB_STAR database. We can observe for the higher support values, the runtime for mining maximal patterns decreases, since the number of maximal patterns present in the database is low for higher support values. From the result it is noticed that the runtime of the MSFP-tree for mining maximal patterns is low compared to the runtime of the MFI-tree with varying minimum supports. This is due to the effect of novelty in tree construction phase of the MSFP-tree.

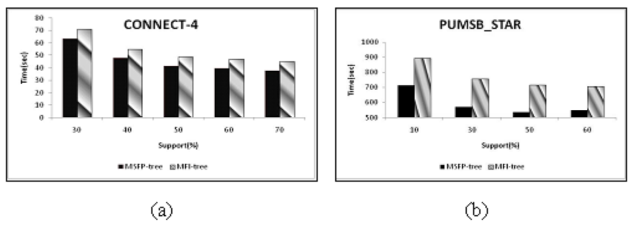


Figure 6: Scalability with various in minimum support values (a) CONNECT-4 database (b) PUMSB_STAR database

CONCLUSIONS

Various algorithms and approaches have been proposed in related works for mining patterns. However, most of these approaches take more time for mining especially in very large voluminous transactional databases. Hence we proposed the algorithm, which captures the entire transactional database with single scan to construct the prefix tree for mining maximal patterns. SFP-tree is constructed using already constructed tree, which is frequent-independent order. Hence, we dynamically reorganize the frequent-independent tree into frequent-descending order using merge sort technique. We pictorially illustrated these procedures for sample transactional database. It is noticed that the database scan time increases the mining time considerably, since, MFI-tree is constructed with two scan, it is imperative that the number of scans of the database required to be reduced for mining maximal patterns efficiently. Hence, the efficiency is developed in our algorithm using single scan of the database. Due to the novel modification in construction process, the MSFP- tree takes lesser time compare to MFI-tree. FPmax approach is used in our algorithm to mine maximal patterns from the MSFP-tree. The performance of the proposed tree is evaluated in benchmark databases with huge number of transactions with varying length. We have also used both sparse and dense databases for mining the maximal patterns. Our future research work to modify this algorithm for sequential patterns mining.

REFERENCES

1. Agarwal, R., Aggarwal & C.C., Prasad, V.V.V. (2001). A tree projection algorithm for generation of frequent itemsets. *J Parallel Distribution Computation* 61:350–371.
2. Agrawal, R., & Shafer, J.C. (1996). Parallel mining of association rules: design, implementation, and experience. *IEEE Trans Knowledge Data Engineering* 8:962–969.
3. Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In: Proceedings of the *international conference on very large data bases (VLDB'94)*, Santiago, Chile, pp 487–499.
4. Bayardo, R.J. (1998). Efficiently mining long patterns from databases. In: Haas LM, Tiwary A (eds) *Proceedings of the SIGMOD 1998, ACM Press*, New York, pp 85–93.
5. Blake, C.L., & Merz, C.J.(1998). UCI Repository of Machine Learning Databases, University of California – Irvine, Irvine, CA.
6. Brin, S et al., (1997). Dynamic itemset counting and implication rules for market basket analysis. In: Proceeding of the *international conference on management of data (SIGMOD'97)*, Tucson, AZ, pp 255–264.
7. Burdick, D., Calimlim, M., & Gehrke, J. (2003). MAFIA: A Performance Study of Mining Maximal Frequent Itemsets. In: Proceedings of the *IEEE ICDM Workshop on Frequent Itemset Mining Implementations Melbourne*, Florida, *USA*.
8. Cheung, D.W., Han, J., Ng, V., & Wong, C.Y. (1996). Maintenance of discovered association rules in large an incremental updating technique. In: Proceeding of the *international conference on data engineering (ICDE'96),* New Orleans, LA, pp 106–114.
9. Cheung, D.W. et al., (1996). A fast distributed algorithm for mining association rules. In: Proceeding of the *international conference on parallel and distributed information systems*, Miami Beach, FL, pp 31–44.
10. Geerts, F., Goethals, B., & Bussche, J. (2001). A tight upper bound on the number of candidate patterns. In: Proceeding of the *international conference on data mining (ICDM'01)*, San Jose, CA, pp 155–162.
11. Grahne, G., & Zhu, J.F. (2003). High Performance Mining of Maximal Frequent Itemsets. In: Proceeding of the *6th SIAM International Workshop on High Performace Data Mining*, San Francisco, CA, pp:135-143.
12. Gouda, K., & Zaki, M. J. (2001). Efficiently Mining Maximal Frequent Itemsets. In: Proceedings of the *IEEE International Conference on Data Mining,* San Jose.
13. Han, J., Pei, J., & Yin, Y. (2004). Mining frequent patterns without candidate generation: A Frequent-Pattern Tree approach. In: *Data Mining and Knowledge Discovery*, 8, 53–87.
14. Holsheimer, M. et al. (1995). A perspective on databases and data mining. In: Proceeding of the international *conference on knowledge discovery and data mining (KDD'95),* Montreal, Canada, pp 150–155.
15. IBM, QUEST Data Mining Project, <http://www.almaden.ibm.com/cs/quest>.
16. Liu, J., Pan, Y., Wang, K., & Han, J. (2002). Mining frequent item sets by opportunistic projection. In: Proceeding of the *ACM SIGKDD international conference on knowledge discovery in databases (KDD'02)*, Edmonton, Canada, pp 239–248.
17. Liu, G. et al. (2003). On computing, storing and querying frequent patterns. In: Proceeding of the

international conference on knowledge discovery and data mining (KDD'03), Washington, DC, pp 607–612.

18. Mannila, H., Toivonen, H., & Verkamo, A.I. (1994). Efficient algorithms for discovering association rules. In: Proceeding of the *AAAI'94 workshop knowledge discovery in databases (KDD'94),* Seattle, WA, pp 181–192.
19. Park, J.S., Chen, M.S., & Yu, P.S. (1995). An effective hash-based algorithm for mining association rules. In: Proceeding of the *international conference on management of data (SIGMOD'95),* San Jose, CA, pp 175–186.
20. Park, J.S., Chen, M.S., & Yu, P.S. (1995). Efficient parallel mining for association rules. In: Proceeding of the *4th international conference on information and knowledge management,* Baltimore, MD, pp 31–36.
21. Pei, J. et al. (2001). PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceeding of the *international conference on data engineering (ICDE'01),* Heidelberg, Germany, pp 215–224.
22. Pei, J. et al. (2001). Hmine: Hyper-structure mining of frequent patterns in large databases. In: Proceeding of the *IEEE International Conference on Data Mining,* pp. 441–448.
23. Rigoutsos, L., & Floratos, A.(1998). Combinatorial pattern discovery in biological sequences:The Teiresias algorithm. *Bioinformatics 14,* 1,pp. 55-67.
24. Sarawagi, S., Thomas, S., & Agrawal, R. (1998). Integrating association rule mining with relational database systems: alternatives and implications. In: Proceeding of the *international conference on management of data (SIGMOD'98),* Seattle, WA, pp 343–354.
25. Savasere, A., Omiecinski, E., & Navathe, S. (1995). An efficient algorithm for mining association rules in large databases. In: Proceeding of the *international conference on very large data bases (VLDB'95),* Zurich, Switzerland, pp 432–443.
26. Shiguang Ju., & Chen Chen. (2008). MMFI: an Effective Algorithm for Mining Maximal Frequent Itemsets. In: Proceedings of the *International Symposiums on Information Processing.*
27. Toivonen, H. (1996). Sampling large databases for association rules. In: Proceeding of the international *conference on very large data bases (VLDB'96)*, Bombay, India, pp 134–145.
28. Xiaoyun Chen et al. (2008). A High Performance Algorithm for Mining Frequent Patterns: LPS-Miner. In: Proceeding of the *International Symposium on Information Science and Engineering, IEEE Computer Society*, Washington, DC, USA.
29. Yong, Qiu., & Yong-Jie., Lan. (2006). Efficient Improvement of FT-Tree Based Frequent Itemsets Mining Algorithms. In: Proceedings of the *First International Conference on Innovative Computing, Information and Control - Volume III (ICICIC'06)*, vol. 3, pp.374-377.
30. Yuejin Yan., Zhoujun Li., & Huowang Chen. (2004). Fast Mining Maximal Frequent ItemSets Based on FP-Tree. P. Atzeni et al. (Eds.)*: ER 2004, © Springer-Verlag Berlin Heidelberg,* LNCS 3288, pp. 348.361.
31. Zaki, M.J., Parthasarathy, S., Ogihara, M., & Li, W. (1997). Parallel algorithm for discovery of association rules. *data mining knowledge discovery,* 1:343–374.
32. Zaki, MJ. (2000). Scalable algorithms for association mining. IEEETransaction Knowledge Data Engineering 12:372–390.