International Journal of Computer Science and Business Informatics

IJCSBI.ORG

Present a Way to Find Frequent Tree Patterns using Inverted Index

Saeid Tajedi
Department of Computer Engineering
Lorestan Science and Research Branch, Islamic Azad University
Lorestan, Iran

Hasan Naderi
Department of Computer Engineering
Iran University of Science and Technology
Tehran, Iran

ABSTRACT
Among all patterns occurring in tree database, mining frequent tree is of great importance. The frequent tree is the one that occur frequently in the tree database. Frequent subtrees not only are important themselves but are applicable in other tasks, such as tree clustering, classification, bioinformatics, etc. In this paper, after reviewing different methods of searching for frequent subtrees, a new method based on inverted index is proposed to explore the frequent tree patterns. This procedure is done in two phases: passive and active. In the passive phase, we find subtrees on the dataset, and then they are converted to strings and will be stored in the inverted index. In the active phase easily, we derive the desired frequent subtrees by the inverted index. The proposed approach is trying to take advantage of times when the CPU is idle so that the CPU utilization is at its highest in in evaluation results. In the active phase, frequent subtrees mining is performed using inverted index rather than be done directly onto dataset, as a result, the desired frequent subtrees are found in the fastest possible time. One of the other features of the proposed method is that, unlike previous methods by adding a tree to the dataset is not necessary to repeat the previous steps again. In other words, this method has a high performance on dynamic trees. In addition, the proposed method is capable of interacting with the user.

Keywords: Tree Mining, Inverted Index, Frequent pattern mining, tree patterns.

1. INTRODUCTION
Data mining or knowledge discovery deals with finding interesting patterns or information that is hidden in large datasets. Recently, researchers have started proposing techniques for analyzing structured and semi-structured datasets. Such datasets can often be represented as graphs or trees. This has led to the development of numerous graph mining and tree mining algorithms in the literature. In this article we present an efficient algorithm for mining trees.

Data mining has evolved from association rule mining, sequence mining, to tree mining and graph mining. Association rule mining and sequence mining are one-dimensional structure mining, and tree mining and graph mining are two-dimensional or higher structure mining. The applications of tree mining arise from Web usage mining, mining semi-structured data, and bioinformatics, etc.

Basic and fundamental ideas of tree mining, roughly since the early '90s were seriously discussed and during this decade were completed. These can be stated that the origin and beginning of these ideas is their application especially on the web. First, some essential and basic concepts are described, and then describe the proposed method and finally the results will be evaluated.

2. Related Works

2.1 Pre Order Tree Traversal

There are several ways to navigate through the ordered trees; pre order traversal is one of the most important and most widely used of them. In this way, we are acting like Depth First Search algorithm. This means that on the tree like T starting from the root, then the left child and finally the right child is navigating; this method is done recursively on all nodes of the tree.

2.2 Post Order Tree Traversal

It is also among the most important and widely used methods of ordered trees traversal. In this method, we first on the tree like T starting from the left child, then right child and finally the root is navigating, the operation is performed recursively on all nodes of the tree.

Using either method, the above display, we can assign a number to each of the nodes that in fact, it is represents a time to meet each node. If we use the Post Order Traversal, that number is called PON.

2.3 RMP, LMP

LMP is the acronym Left Most Path represents a path from the root to the leftmost leaf and the RMP is the acronym Right Most Path represents a path from the root to the rightmost leaf.

2.4 Prüfer Sequence [23]

This algorithm was introduced in 1918 and used to convert the tree to string. The algorithm works as follows in the tree like T, in every step the node with the smallest label has been removed and label the parent node of this tree is added to the Prüfer Sequence. This process is repeated n-2 times to 2 nodes remain.

2.5 Label Sequence

The next concept is Label Sequence. This sequence is produced according to the Post Order Traversal. In other words, in the Post Order Traversal, label of each node that will be scanned, add to the sequence.

2.6 Support

Simply this implies that the S pattern has been repeated several times in the tree T.

$$Support(S) = \frac{\#\ of\ trees\ where\ S\ is\ embedded}{\#\ of\ trees\ in\ D} \quad (1)$$

Where S is a tree pattern and D is a database of trees. This concept for determining the number of occurrences of each subtree in a set of trees is being used.

2.7 Inverted Index [24]

Inverted Index is a structure that used to indexing frequent string elements in set of documents and is consists of two main parts: Dictionary and Posting List. Frequent string elements uniquely stored in Dictionary and the number of occurrences of each of these elements in total documents is determined. Informations about the frequent elements such as the document name, number of occurrences in each document are determined in Posting List.

3. An overview of research history

In recent years, much research about the frequent subtrees mining has been done. Yongqiao Xiao et al in 2003 used the Path Join Algorithm and a compact data structure, called FST-Forest to find frequent subtrees [25]. In this way, we first find frequent root path in all directions and then with integrate these paths, frequent subtrees are reached. Shirish Tatikonda et al published an article in 2006 on the basis of the pattern growth[26]; In this way that all trees in the database tree are converted to strings, that is done with the two different methods: Prüfer Sequence and DFS algorithms; then scroll all strings in which there is a subtree or pattern such as S, we are seeking a new edge can be added to S. Then, concurrently with the previous step, as production of the candidate subtrees, the threshold values are evaluated for be frequent. In 2009, Federico Del Razo Lopez et al presented an idea to make flexible the tightly constrained tree mining In non-fuzzy[27]. This paper used the principle of Partial Inclusion; that to say that there is a pattern S in a tree T, it is no need to exist all the pattern nodes in the tree. The proposed algorithm uses Apriori property for pruning undesirable patterns.

4. The proposed approach

This procedure is done in two phases: Passive and Active. In the Passive phase, first we need to find all subtrees available in all trees and then must be store in Inverted Index. In the Active phase, simply use it and will extract frequent tree patterns.

4.1 Passive Phase

This phase is done in two stages, in first stage; we must find all subtrees of every tree in dataset then they will be converted to a string called that tree and in second stage; produced strings in the first stage should be stored in the Inverted Index.

4.1.1 First stage of Passive phase

The first important point is that in each tree, each node label in the tree can be repeated many times, but every node in every tree has a unique label; to solve this problem, we use the method of Prüfer Sequence. This means that each tree can be traced to Post Order and In fact, Prüfer Sequence Algorithm works based on the PON. As a result, each node label of a tree will be marked with a unique number.

The next issue is that the Prüfer Sequence able to cover all the nodes, therefore, the algorithm implementation process rather than n-2 steps process will continue until n steps and rather than the parent label of the last node, put the number 0. In Figure 1 you can see an example of this method is that the purpose of the NPS is Prüfer Sequence that has been achieved using Post Order.

The next thing is that every subtree should be displayed uniquely; to this end, must obtain CPS for each node. In fact, CPS will merge Prüfer Sequence and Label Sequence. In other words, CPS(T) = (NPS,LS)(T). CPS can uniquely display a rooted and labeled tree. As you can see in Figure 1, the T1 tree can be displayed uniquely using two strands that are complementary.

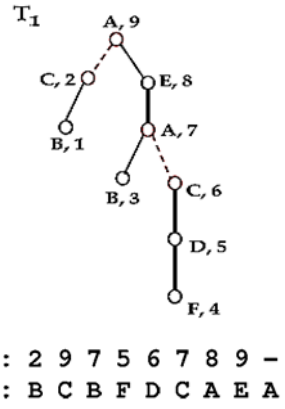

Figure 1. An example of the Prüfer Sequence and Label Sequence for T1 tree

The next thing that we need to ensure that in each tree can produce all subtrees and each subtree is created only once, for this purpose, we use the LMP to generate the subtree. This means that if we show the T tree using Prüfer sequence and n is the subtree, A node such as v that is to be added to the n should be included in the LMP of the T tree and since the PON is built on the Prüfer sequence, just the v node should be after the last node of the n and attached to that in the Prüfer sequence of the T tree. So is guaranteed to be generated only once for each subtree and if it is done for all the nodes, entire subtree of each tree will produce.

Now we will introduce the algorithm. The proposed algorithms for generating subtrees and convert them into a string can be seen in Figure 2.

```
Insert CPS(T) in Array  A
For i=n downto 1 do
{
  Subtree=A[n]
  Insert CPS(A[n]) in Treestringi
  Sub(subtree,i,A, stack1,stack2)
}

Sub(subtree,index,A[],stack1,stack2)
c=0
t=0
For j= 1 to index-1 do
If index in A[j] then
{
   stack3=stack1
   stack4=stack2
   subtree2=subree
   while stack3 not empty
   {
        t++
        Pop x from stack1
        Pull y from stack2
        Subtree2=subtree2+x
        if t>0 then
        {
        Insert CPS(subtree2) in treestringi
        Sub(subtree2,y, A[],stack3,stack4)
        }
   }
    If c>0 then
    {
   Temptree push in stack1
   TempIndex push in stack2
   }
   Temptree= a[j]
   TempIndex=j
   c++
   Subtree=subtree+a[j]
```


```
Insert CPS(subtree) in treestringi
Sub(subtree,j,A[],stack1,stack2)
while stack1 not empty
{
    c--
    Pull x from stack1
    Pull y from stack2
    Insert CPS(subtree+x) in treestringi
    Sub(subtree+x,y, A[],stack1,stack2)
}
}
```

Figure 2. The algorithm of the subtrees generation and convert them to a string

In the following we examined the algorithm works with an example. In the beginning starting from the first tree and CPS (T) are stored in the array A. As a result, the array will be completed for T1 according to Figure 3.

۱	۲	۳	۴	۵	۶	۷	۸	۹
B2	C9	B7	F5	D6	C7	A8	E9	A0

Figure 3. Production of the array using CPS (T)

In this step we identify all existing subtrees and store them in a string. To do this, we start from the root node of T1, therefore the last element of the array namely A0 and respectively, the branching subtrees from this node should be stored in the string. As a result, at first A0 is stored in the string according to the algorithm, next we run sub function. Considering that the index of previous node is equal to 9 to find the subtrees with two nodes, respectively start from the first element of the array and review to the element with the pre Index of the previous node namely 8, If the value contains the index of the previous node namely 9, It has added to the previous tree namely A and CPS of the found subtree can be inserted in the string of this tree that here A0C2 and A0E2 are stored in the string and recursively repeat the same steps for new generated subtrees. Given that both produced subtrees branched from a node, adding node with smaller index from Stack1 and its index from stack2 are extracted and added to the subtree with a larger index and its CPS is stored in the string,therefore in this step is also added A0E3C3 and also this is repeated for the whole produced subtrees with larger index in the next step. Similarly, the work continues recursively until all subtrees branching from the first node of the array to be stored in string. Then do the same procedure for the next elements of the array, until complete the string of the subtrees of the tree and then we proceed next trees until for each tree, the string is created for all subtrees of the tree.

4.1.2 First stage of Passive phase

In the second stage of this phase we use the Inverted Index. Thus, the strings created in the previous stage are inserted into the Inverted Index. CPS and the number of occurrences of each subtree in the all trees are stored in the Dictionary and the name of the trees that are containing the subtree will be stored in the corresponding Posting List.

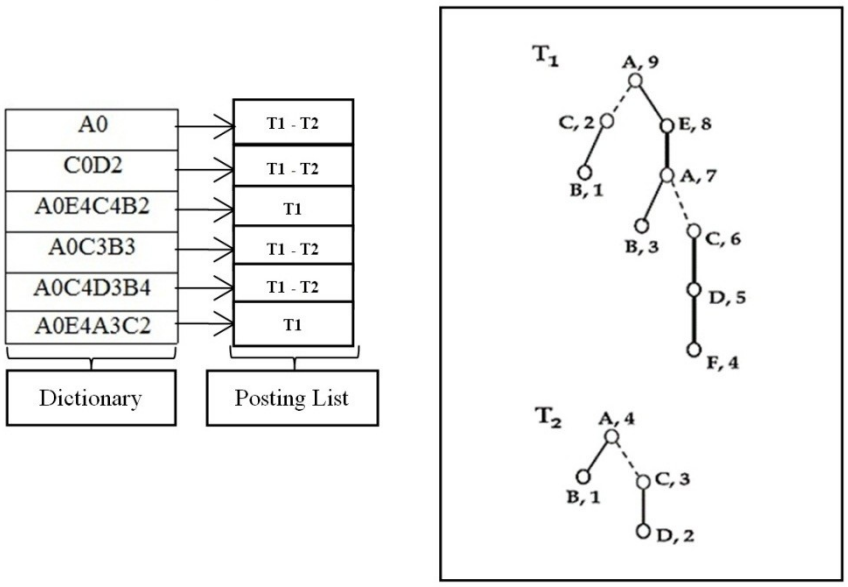

Figure 4. Part of the Inverted Index made for the collection of trees T1, T2

As can be seen in the subtrees are stored in the Dictionary and the parent trees of the corresponding subtrees are stored in the Posting List.

4.2 Active Phase

In this phase, simply use inverted index made in the previous phase and will extract frequent tree patterns. Simply types of queries about frequent subtree mining to be answered quickly by using inverted index made in the previous phase. Then we will examine several types of different queries.

4.2.1 Find the occurrence of the desired pattern in tree set

First, we are achieved CPS of the desired pattern and then search it into Dictionary of the inverted index and easily extract the number of the occurrence and name of trees that contain the desired pattern from the Posting List of the inverted index. For example, to find the number of occurrences of the S pattern on the collection of trees T1, T2 in Figure 5, should search CPS (S) ie A0C3B3 into Inverted Index that T1 and T2 will be the result.

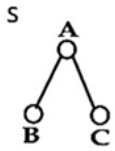

Figure 5. Part of the Inverted Index made for the collection of trees T1, T2

4.2.2 Find frequent subtrees in considering the Support

If we want to find some subtrees that them Support are greater than a threshold, must find the subtrees with their occurrence compared to the total trees is greater than the Support. So we can search in the inverted index and easily find subtrees that the length of them Posting List compared to the total trees is at least equal to Support.

4.2.3 Find frequent subtrees in considering the Support and minimum nodes

In this case, in addition to Support, the number of nodes is also the criterion, so easily search in Inverted Index and only show the subtrees with the following conditions. First, in Dictionary Length the subtree is greater than the minimum number of nodes and Second, length of corresponding Posting List compared to the total trees is at least equal to Support.

5. Evaluation

In this section, the proposed method will be evaluated from various aspects. We present the experimental evaluation of the proposed approach on synthetic datasets. In the following discussion, dataset sizes are expressed in terms of number of trees. In the graphs is used from symbolizes Algorithm to display proposed method. Name and details of synthetic datasets are shown in Table 1.

Table 1. Name and details of synthetic datasets

Name	Description
DS1	-T 10 -V 100
DS2	-T 10 -V 50

As shown in Table 1, the synthetic datasets DS1 and DS2 are generated using the PAFI[28] toolkit developed by Kuramochi and Karypis (PafiGen). Since PafiGen can create only graphs we have extracted spanning trees from these graphs and used in our analysis. We also used minsup to analyze the various factors. This means if the number of replicated subtree is less than minsup value, the tree won't be indexed in Inverted Index. Minsup value is from 1 to infinity, which is the default value is equal to 1 in the proposed algorithm. In addition, we also use from maxnode in evaluations. Maxnode is the symbol to specify the maximum number of nodes in each subtree in Inverted Index. This means if the number of subtree nodes reach maxnode amount in the proposed algorithm, production of its subtree will halt.

Maxnode value is from 1 to infinity, and the default value is equal to infinity.

5.1 Evaluating the performance of the proposed method

At the beginning, we evaluated our proposed algorithm on two synthetic datasets DS1 and DS2. The performance of the proposed algorithm for frequent tree minig on synthetic datasets is shown in Diagram 1. In this experiment, the minsup equal to one and the maxnode is equal to infinity. Given that the Subtrees are indexed in passive phase at times when the system is idle, mining time in Inverted Index rises with a gentle slope By increasing the number of trees. So clearly spelled out the introduced algorithm is scalable.

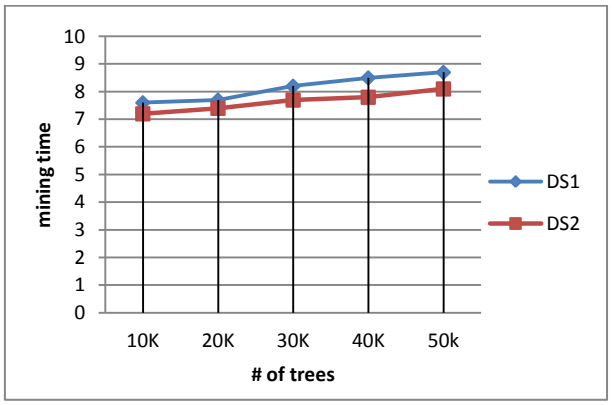

Diagram 1: The performance of the algorithm on synthetic datasets

5.2 Evaluating effect of minsup on the number of indexed patterns

We examine effect of minsup on the number of indexed patterns in Diagram 2. This experiment has been done on synthetic datasets DS1 and DS2 generated by Pafi and with size 50K. In this experiment the maxnode is the default value ie infinity. As can be seen in the diagram, the number of indexed patterns is increasing exponentially by decreasing minsup.

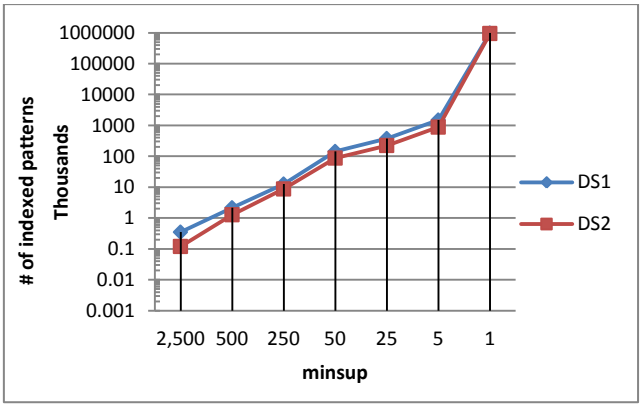

Diagram 2: Effect of minsup on the number of indexed patterns

5.3 Evaluating effect of maxnode on usage memory

We examine the effect the maximum number of nodes in the indexed subtrees on usage memory in passive phase. This experiment has been done on synthetic datasets DS1 and DS2 generated by Pafi and with size 50K. In this experiment the minsup is the default value ie 1. As can be seen, the usage memory of the algorithm is increasing by increasing the number of indexed nodes in each subtree.

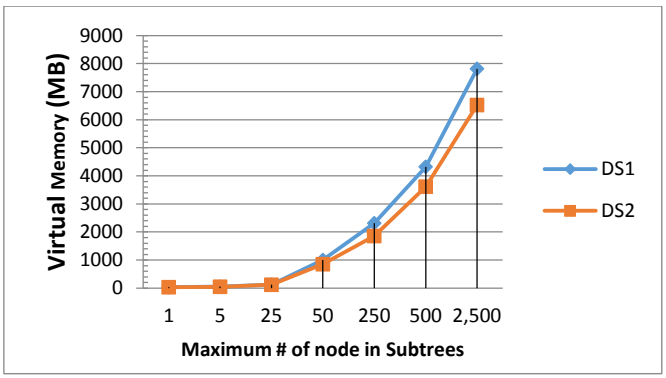

Diagram 3: Effect of maxnode on the usage memory

5.4 Evaluation of CPU utilization compared with the Tree Miner

In diagram 4, the comparison is performed between the proposed algorithm with Tree Miner that was introduced by Zaki and is one of the best algorithms for tree mining[29]. This experiment has been done on synthetic dataset DS1 generated by Pafi and with size 50K. Given that in passive phase the proposed algorithm is searching for subtrees and adding them to inverted index, consequently, as can be seen in the diagram, CPU utilization is close to 100 percent in most situations while the average CPU utilization on TreeMining algorithm is approximately 90%.

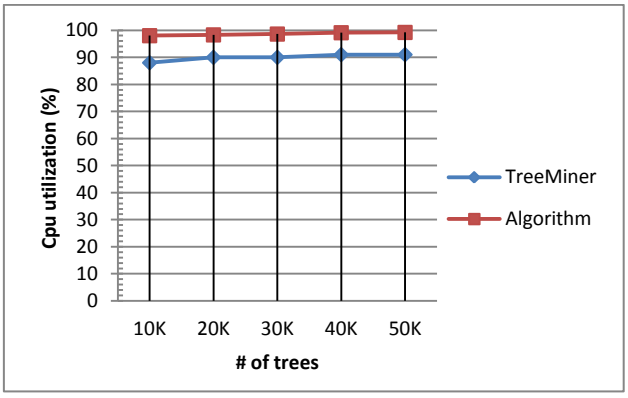

Diagram 4: Comparison CPU utilization between TreeMiner and the algorithm

6. Conclusions and Recommendations

In this paper, a new method based on the Inverted Index in order to frequent pattern mining was introduced to overcome many of the disadvantages of previous methods. One problem with existing approaches is that mainly act as a static on the set of trees and if a new tree is added to the set of trees, all mining operations must be done from scratch again. This problem has been overcome by the Inverted Index in the proposed approach. This means that all the trees are indexing in the Passive phase and if a new tree is added to the treeset at any stage, just the tree is indexed and there is no need to repeat the previous operations. This algorithm will result in a high performance on a collection of dynamic trees. Another advantage of this method compared to other methods is that it is scalable. As listed in Section 5.1, the performance of this algorithm is not slowed by increasing the treeset. As listed in Section 5.4, one of the most striking features of this algorithm is efficient use of CPU. In this method, the user interaction is also present.

As Listed in Section 5.2, the number of indexed patterns increases exponentially by decreasing minsup, while the generally patterns with low occurrences doesn't matter to us. As a result, we can speed up indexing in passive phase with determining the appropriate amount of the minsup. As Listed in Section 5.3, the usage memory increases by increasing the maximum number of nodes in the indexed subtrees, while the usually subtrees with very large number of nodes doesn't matter to us. As a result, we can manage the usage memory with determining the appropriate amount of the maxnode.

REFERENCES

[1] B. Vo, F. Coenen, and B. Le, "A new method for mining Frequent Weighted Itemsets based on WIT-trees," *International Journal of Advanced Computer Research,* p. 9, 2013.

[2] L. A. Deshpande and R. S. Prasad, "Efficient Frequent Pattern Mining Techniques of Semi Structured data: a Survey," *International Journal of Advanced Computer Research,* p. 5, 2013.

[3] A. M. Kibriya and J. Ramon, "Nearly exact mining of frequent trees in large networks," *Data Mining and Knowledge Discovery (DMKD),* p. 27, 2013.

[4] G. Pyun, U. Yun, and K. H. Ryu, "Efficient frequent pattern mining based on Linear Prefix tree," *International Journal of Advanced Computer Research,* p. 15, 2014.

[5] C. K.-S. Leung and S. K. Tanbeer, "PUF-Tree: A Compact Tree Structure for Frequent Pattern Mining of Uncertain Data," *Advances in Knowledge Discovery and Data Mining,* p. 13, 2013.

[6] A. Fariha, C. F. Ahmed, C. K.-S. Leung, S. M. Abdullah, and L. Cao, "Mining Frequent Patterns from Human Interactions in Meetings Using Directed Acyclic Graphs," *Advances in Knowledge Discovery and Data Mining,* p. 12, 2013.

[7] J. J. Cameron, A. Cuzzocrea, F. Jiang, and C. K. Leung, "Mining Frequent Itemsets from Sparse Data," *Web-Age Information Management,* p. 7, 2013.

[8] G. Lee, U. Yun, and K. H. Ryu, "Sliding window based weighted maximal frequent pattern mining over data streams," *Advances in Knowledge Discovery and Data Mining,* p. 15, 2014.

[9] H. He, Z. Yu, B. Guo, X. Lu, and J. Tian, "Tree-Based Mining for Discovering Patterns of Reposting Behavior in Microblog," *Advanced Data Mining and Applications,* p. 13, 2013.

[10] U. Yun, G. Lee, and K. H. Ryu, "Mining maximal frequent patterns by considering weight conditions over data streams", *Advances in Knowledge Discovery and Data Mining,* 2014.

[11] B. Kimelfeld and P. G. Kolaitis" ,The complexity of mining maximal frequent subgraphs," *Proceedings of the 32nd symposium on Principles of database systems,* p. 12, 2013.

[12] B. Vo, F. Coenen, and B. Le, "A new method for mining Frequent Weighted Itemsets based on WIT-trees," *International Journal of Advanced Computer Research,* p. 9, 2013.

[13] L. A. Deshpande and R. S. Prasad, "Efficient Frequent Pattern Mining Techniques of Semi Structured data: a Survey," *International Journal of Advanced Computer Research,* p. 5, 2013.

[14] A. M. Kibriya and J. Ramon, "Nearly exact mining of frequent trees in large networks," *Data Mining and Knowledge Discovery (DMKD),* p. 27, 2013.

[15] G. Pyun, U. Yun, and K. H. Ryu, "Efficient frequent pattern mining based on Linear Prefix tree" *International Journal of Advanced Computer Research,* p. 15, 2014.

[16] C. K.-S. Leung and S. K. Tanbeer, "PUF-Tree: A Compact Tree Structure for Frequent Pattern Mining of Uncertain Data," *Advances in Knowledge Discovery and Data Mining,* p. 13, 2013.

[17] A. Fariha, C. F. Ahmed, C. K.-S. Leung, S. M. Abdullah, and L. Cao, "Mining Frequent Patterns from Human Interactions in Meetings Using Directed Acyclic Graphs," *Advances in Knowledge Discovery and Data Mining,* p. 12, 2013.

[18] J. J. Cameron, A. Cuzzocrea, F. Jiang, and C. K. Leung, "Mining Frequent Itemsets from Sparse Data," *Web-Age Information Management,* p. 7, 2013.

[19] G. Lee, U. Yun, and K. H. Ryu, "Sliding window based weighted maximal frequent pattern mining over data streams," *International Journal of Advanced Computer Research,* p. 15, 2014.

[20] H. He, Z. Yu, B. Guo, X. Lu, and J. Tian, "Tree-Based Mining for Discovering Patterns of Reposting Behavior in Microblog," *Advanced Data Mining and Applications,* p. 13, 2013.

[21] U. Yun, G. Lee, and K. H. Ryu, "Mining maximal frequent patterns by considering weight conditions over data streams," *International Journal of Advanced Computer Research,* 2014.

[22] B. Kimelfeld, and P. G. Kolaitis, "The complexity of mining maximal frequent subgraphs," *Proceedings of the 32nd symposium on Principles of database systems,* p. 12, 2013.

[23] H. Prüfer. *Prüfer sequence*. Available: http://en.wikipedia.org/wiki/Pr%C3%BCfer_sequence

[24] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press, 2008.

[25] Y. Xiao, J.-F. Yao, Z. Li, and M. H. Dunham, "Efficient data mining for maximal frequent subtrees," *Proceedings of 3rd IEEE International Conference on Data Mining,* p. 8, 2003.

[26] S. Tatikonda, S. Parthasarathy, and T. Kurc, "TRIPS and TIDES: New Algorithms for Tree Mining," *Proceedings of 15th ACM International Conference on Information and Knowledge Management (CIKM),* p. 12, 2006.

[27] F. D. R. Lopez, A.Laurent, P.Poncelet, and M.Teisseire, "FTMnodes: Fuzzy tree mining based on partial inclusion," *Advanced Data Mining and Applications,* pp. 2224–2240, 2009.

[28] Kuramochi and Karypis. Available: http://glaros.dtc.umn.edu/gkhome/pafi/overview/

[29] M. J. Zaki, "Efficiently Mining Frequent Trees in a Forest," *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD), Edmonton, Canada,* p. 10, 2002.

This paper may be cited as:

Tajedi, S. and Naderi, H., 2014. Present a Way to Find Frequent Tree Patterns using Inverted Index. *International Journal of Computer Science and Business Informatics, Vol. 14, No. 1, pp. 66-78*.