International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) e-ISSN: 2395-5325
Volume 5, Issue 3 (September '2019)

ANALYSIS OF PROBABILITY-BASED FREQUENT SUBTREE PATTERN MINING ALGORITHM IN COMPLEX DATA

S.Ramya,
M.Phil Research Scholar,
PG &Research Department of Computer Science,
PGP College of Arts & Science,
Namakkal,Tamilnadu,India.

S.Gandhimathi,
Head & Assistant professor,
PG &Research Department of Computer Science,
PGP College of Arts & Science,
Namakkal,Tamilnadu,India.

Dr V.Baby Deepa,
Assistant professor,
PG &Research Department of Computer Science,
Government Arts College,
Karur,Tamilnadu,India.

Abstract: Pattern mining plays an essential role in many data mining tasks that attempt to find valuable patterns in massive databases. These patterns can include associations, correlations, sequences, episodes, classifiers and/or clusters. However, discovering such patterns is time-consuming because the extracted patterns are often too numerous and thus difficult to analyze by end users, especially when complex data structures are taken into consideration. Furthermore, existing work mainly. Concentrates on discovering common knowledge in a single data set; and not much attention has been given to identifying significant differences among several data sets. This thesis mainly targets four pattern mining topics on complex data structures to provide solutions to the above problems.

Keywords: Pattern mining, Association Rules, Probability-Based Frequent Subtree, BFS-based Candidate Subtree Generation

I. INTRODUCTION

Pattern mining is a mining process for extracting these valuable data patterns from large amounts of data [1]. With the fast development of computing technology, purely manual data analysis has been replaced by an automatic or semi-automatic data mining process [2]. Various state-of-the-art pattern mining algorithms have been reported in the literature, however why do we still need to explore this topic at a deeper level? To discover patterns is not a difficult task, but to discover interesting patterns from large-scale databases is not easy.

A pattern is interesting if it can provide useful and beneficial knowledge to end users for solving their practical application problems. This is where the complexity of the work comes from. In addition, the large size of the pattern set often causes confusion to the end users so that insignificant knowledge is finally returned to target the expected findings of the data analysis. Thus, the new challenge in the research field of pattern mining is to find the most targeted data patterns that are highly interesting and useful to the end users to meet the requirements of their specific applications. Due to the variety of the forms representing numerous data in the existing research domains, it is infeasible to develop one single pattern mining algorithm that can meet all requirements. The work reported in this thesis focuses on mining data patterns from two forms of complex data: tree structures and relational data, which are popular in many emerging research domains, such as Web mining, Chemistry, Biology, social networks, business and marketing analysis [3].

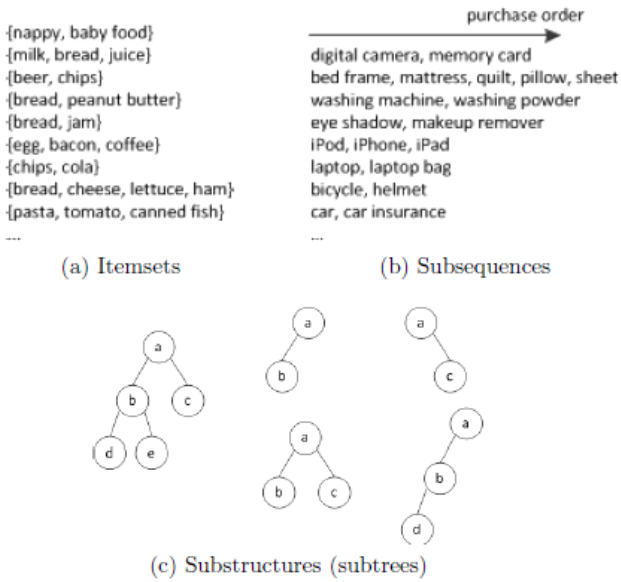


Figure 1: Patterns in Various Data Forms

Pattern mining is quite similar to ore mining [4] which removes soil (noisy data) and extracts valuable minerals (useful patterns) from underground ore bodies. No matter

what advanced techniques miners use or how well they have been trained, the most important precondition of any successful mining activity is that miners known what minerals they are looking for. For example, metallic aluminium is mainly produced from a special type of ore, called Bauxite. If miners have no idea that bauxite is a red-brown rock, they might waste time and energy on finding the shinning silvery gray blocks that rarely exist in the earth's solid surface.

Therefore, before any pattern mining task begins it is important to ask: **What is an interesting pattern?** In data mining tasks, a pattern can be an itemset, a subsequence or substructures appearing with a certain frequency in a database [5]. This pattern is believed to carry some kind of useful information, which can be used to represent particular characteristics of data instances within the database.
A pattern in the form of an itemset was originally discovered from market-basket data analysis using the association rule mining algorithm [2]. Each data instance consists of a number of product items that have been purchased by a customer. The purpose of the analysis task is to find the itemsets that frequently appear in the database. Each discovered itemset shows a set of product items that have been frequently purchased together by the customers in a certain store or supermarket (as shown in Fig. 1a). A sequence can be considered as a set of ordered events, elements or items with or without a concrete notion of time . For example, a new car is purchased before buying car insurance as shown in Fig. 1b. Many sequential pattern mining algorithms have been developed to extract a set of subsequences that frequently appear in a special order. In addition, if the database records the data elements of objects together with their structural information, the potential patterns to be discovered are often in the form of substructures, such as subgraphs in a graph database [3] or subtrees in a tree database [8] (Fig. 1c).

The complexity of mining algorithms increases from mining itemsets, and subsequences to substructures. When the order of items is considered, the traditional itemset pattern mining becomes subsequential pattern mining. Substructure pattern mining can also be regarded as mining patterns from a database representing structural information, where each data structure is a representation of two or more sequences that merge together at the common items.

The rest of this chapter reviews the most influential data mining techniques that have been developed to help discover interesting patterns for various data analysis and mining applications. This chapter begins with an introduction of data patterns that attract the attention of researchers. The relevant pattern mining algorithms and applications are surveyed separately in individual sections with respect to the certain types of data patterns they are interested in. Finally, we discuss current difficulties and problems that still exist in pattern mining, with focus on the challenges of such research.

II. LITERATURE REVIEW

Renáta Iváncsy et al [9] In his paper **"Frequent Pattern Mining in Web Log Data"** says that, Frequent pattern mining is a heavily researched area in the field of data mining with wide range of applications. One of them is to use frequent pattern discovery methods in Web log data. Discovering hidden information from Web log data is called Web usage mining. The aim of discovering frequent patterns in Web log data is to obtain information about the navigational behavior of the users. This can be used for advertising purposes, for creating dynamic user profiles etc. In this paper three pattern mining approaches are investigated from the Web usage mining point of view. The different patterns in Web log mining are page sets, page sequences and page graphs.

Discovery,C.I et al [10] In this paper **"Data Mining and Knowledge Discovery"** process of applying data mining techniques to a sequential database for the purposes of discovering the correlation relationships that exist among an ordered list of events. An important application of sequential mining techniques is web usage mining, for mining web log accesses, where the sequences of web page accesses made by different web users over a period of time, through a server, are recorded. Web access pattern tree (WAP-tree) mining is a sequential pattern mining technique for web log access sequences, which first stores the original web access sequence database on a prefix tree, similar to the frequent pattern tree (FP-tree) for storing non-sequential data. WAP-tree algorithm then, mines the frequent sequences from the WAP-tree by recursively re-constructing intermediate trees, starting with suffix sequences and ending with prefix sequences. This paper proposes a more efficient approach for using the WAP-tree to mine frequent sequences, which totally eliminates the need to engage in numerous re-construction of intermediate WAP-trees during mining. The proposed algorithm builds the frequent header node links of the original WAP-tree in a pre-order fashion and uses the position code of each node to identify the ancestor/descendant relationships between nodes of the tree. It then, finds each frequent sequential pattern, through progressive prefix sequence search, starting with its first prefix subsequence event. Experiments show huge performance gain over the WAP-tree technique.

Kamber Jiawei Han Jenny Y. Chiang [11] In their paper **"Mining of Multi-Dimensional Association Rules Using Data Cubes"** employ a novel approach to metarule-guided, multi-dimensional association rule mining which explores a data cube structure. We propose algorithms for metarule-guided mining: given a metarule containing p predicates, we compare mining on an n-dimensional (n-D) cube structure (where $p < n$) with mining on smaller multiple pdimensional cubes. In addition, we propose an efficient method for precomputing the cube, which takes into account the constraints imposed by the given metarule.

Ashok Savasere Edward Omiecinski Shamkant Navathe [12] in their paper **"An Efficient Algorithm for Mining Association Rules in Large Databases"** says that Mining

for association rules between items in a large database of sales transactions has been described as an important database mining problem. In this paper we present an efficient algorithm for mining association rules that is fundamentally different from known algorithms. Compared to previous algorithms, our algorithm not only reduces the I/O overhead significantly but also has lower CPU overhead for most cases. We have performed extensive experiments and compared the performance of our algorithm with one of the best existing algorithms. It was found that for large databases, the CPU overhead was reduced by as much as a factor of four and I/O was reduced by almost an order of magnitude. Hence this algorithm is especially suitable for very large size databases.

Fedja Hadzic and Henry Tan[13] In their paper **"Efficient algorithm for mining unordered induced subtrees using TMG candidate generation"** says that Semi-structured data sources are increasingly in use today because of their capability of representing information through more complex structures where semantics and relationships of data objects are more easily expressed. Extraction of frequent sub-structures from such data has found important applications in areas such as Bioinformatics, XML mining, Web mining, scientific data management etc. This paper is concerned with the task of mining frequent unordered induced subtrees from a database of rooted ordered labeled subtrees. Our previous work in the area of frequent subtree mining is characterized by the efficient Tree Model Guided (TMG) candidate enumeration, where candidate subtrees conform to the data's underlying tree structure. We apply the same approach to the unordered case, motivated by the fact that in many applications of frequent subtree mining the order among siblings is not considered important. The proposed UNI3 algorithm considers both transaction based and occurrence match support. Synthetic and real world data are used to evaluate the time performance of our approach in comparison to the well known algorithms developed for the same problem.

III.PROBABILITY-BASED FREQUENT SUBTREE PATTERN MINING ALGORITHM

In response to the problems listed for the general frequent subtree pattern mining, we propose a novel Probability-based Frequent Subtree Pattern Mining (ProTreeMiner) algorithm, which generates candidate subtrees in a breath-first searching order and measures the frequency of a candidate based on its occurrence probability in the tree database. In ProTreeMiner, the quality of an output subtree pattern is identified by its similarity to the tree instances using the parse-tree kernel technique instead of direct subtree matching. A framework provided to give an overview of our proposed ProTreeMiner algorithm, followed by detailed mining processes presented in the subsections.

3.1 The Framework

The proposed ProTreeMiner algorithm follows the general mining steps of the traditional Apriori-based approach which generates a set of candidate subtree patterns and then tests them in-turns to select the most frequent and interesting ones. However, it is specially designed to reduce the computational costs from both processes.

Apriori-based algorithms suffer from generating a large number of candidates with no standard principle to identify the best tree node to be added for the subtree extension. The worst case is that all tree nodes that exist in the database are used to generate candidate subtrees. If there are n individual tree nodes in the database, then in one iteration, n new candidate subtrees are generated and their frequencies are also counted. There is a large computational cost but only a few interesting patterns are selected as the final output. To address this problem, we design a novel candidate generation method, in which the candidate subtrees are directly generated based on the probability of their occurrence in the entire tree database.

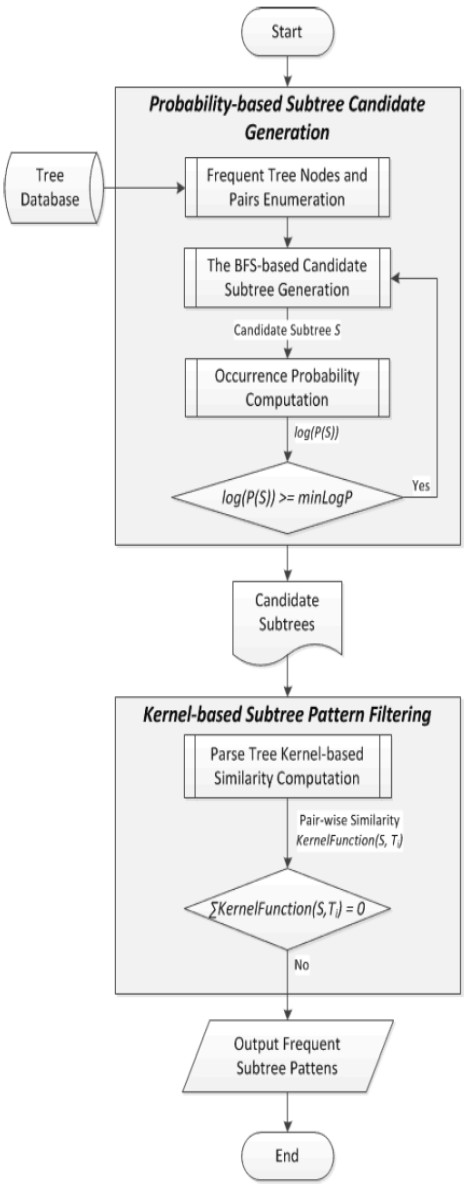


Figure 2: Probability-based Frequent Subtree Pattern Mining (ProTreeMiner) Algorithm

After the first scan of the tree database, we can obtain the following information: a list of all tree nodes that exist in the database; a set of all tree node pairs that present the parent-child relationship between two linked tree nodes; and the supports for both tree nodes and pairs. We generate a new candidate subtree based on these parent-child tree node pairs. For a child node, if none of its parent nodes exist in a generated subtree, then it will not be chosen to extend that subtree. Therefore, we only generate candidate subtrees that can possibly exist in the given tree database, and that the occurrence probability of a candidate subtree can be estimated by calculating the joint probability of all the tree nodes in it. Accordingly, no time is wasted on generating candidate subtrees that do not exist in the database. In addition, by setting a minimum threshold for the occurrence probability, we can easily prune out the candidate subtrees that have a rare chance to appear in the database.

In order to avoid the computational complexity brought by subtree matching for frequency counting, we utilize a parse tree kernel counting function to measure the similarity between a candidate subtree and a tree instance in the second scanning of the database. Parse tree kernel counts the shared sub-structures in two tree structures. This is a recursive process that continuously divides the shared sub-structure until reading the single tree nodes. This means no subtree isomorphism problem needs to be considered in the entire mining process. The filtering process removes those candidate subtrees that do not share any of the same sub-structure with all tree instances in the database. Moreover, the obtained similarity scores can indicate not only the subtrees patterns that were a "partial match" with the tree instances, but also the degree of similarity. We can naturally group tree instances based on their similarity scores calculated from the same set of subtree patterns.

The mining procedure of the proposed ProTreeMiner algorithm is shown in Fig. 2. The rest of this section presents details for theoretical techniques used in both the probability-based subtree candidate generation process and the kernel-based subtree pattern filtering process.

3.2. Probability-based Candidate Subtree Pattern Generation

The main parts of the proposed candidate subtree pattern generation model are: (1) the enumeration of frequent tree nodes and node pairs with their occurrence-match supports; (2) the generation of candidate subtrees via the breath-first searching (BFS) order; and (3) the computation of the occurrence probability to keep the most likely candidate subtrees.

Frequent Tree Nodes and Node Pairs Enumeration

To obtain the full list of frequent tree nodes, we need to extract each tree node with a unique label, and extract all the node pairs that are linked directly in some tree instance. The frequency for each tree node or node pair can be measured by the occurrence match support (OS) [12], which takes the repetition of items in a transaction into account and counts the subtree occurrences in the database as a whole. Hence, the support supp(x) (or supp(x; y)) of a tree node x (or a tree pair (x; y)) in the database D_{db} is equal to the total number of occurrences the node (or the tree pair) in all tree instances in D_{db}. Let OS(x; T) denote the total number of occurrences of node x in a tree T, and OS((x; y); T) is the total number of occurrences of node pair (x; y) in T. Suppose there are M trees in D_{db}, the occurrence-match support of a node x in D_{db} can be defined as:

$$\text{Supp}(x) = \sum_{i=1}^{m} \text{OS}(x, Ti)$$

together, the *occurrence-match support* of a node pair (*x; y*) in *Ddb* can be defined as:

$$\text{Supp}(x,y) = \sum_{i=1}^{m} \text{OS}((x, y), Ti)$$

The obtained support of each tree node, supp(x), is compared with the pre-defined frequency threshold δ_{freq}: if and only if supp(x) $\geq \delta_{freq}$, the tree node x is kept; otherwise, it is discarded from the mining process. The same pruning method is used for all tree pairs as well. The output of this process is two lists; one containing the frequent tree nodes and the other containing the frequent node pairs. The tree nodes and the pairs are sorted in descending order based on their occurrence-match support values.

The *BFS*-based Candidate Subtree Generation

As the candidate subtrees are generated via *breath-first searching* (BFS), a set of candidate subtrees with two levels is generated first, denoted as the candidate 2- subtrees. Then, the candidate subtrees grow level by level, meaning candidate *k*- subtrees will not be generated unless all the frequent candidate ($k-1$)-subtrees were identified.

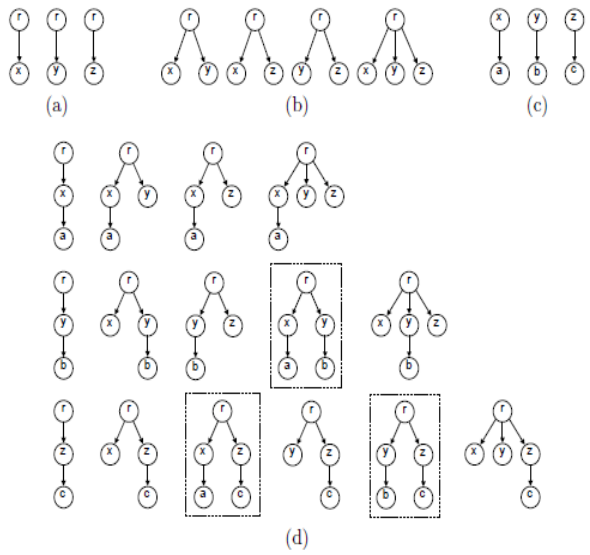


Figure 3: An Example of Candidate Subtree Generation. (a) Frequent Node Pairs with *r* as parent node; (b) Candidate 2-subtrees except the initial node pairs; (c) Candidate More Frequent Node Pairs with *x*, *y* and *z* as parent nodes; and (d) Candidate 3-subtrees, the ones with the border will be removed in further processes due to the low occurrence probabilities.

The detailed generation process is as follows. The candidate 2-subtree is generated based on the node pairs where the parent node is picked from the list of frequent tree nodes in the previous enumeration process. Any node pairs with the same parent node are joined together to generate the

candidate 2-subtrees. For example, to join two node pairs, the parent node becomes the root node and the two child nodes from two node pairs now are the two child nodes of the root node on the second level of the new generated subtree. These two child nodes are also the two leaf nodes in the tree. Following the same approach, a new candidate 2-subtree joined by *n* node pairs with the same parent node has one root node and *n* leaf nodes on the second level. Each new candidate subtree is tested for its frequency which is calculated by its occurrence probability. Only the one with a greater occurrence probability than a given minimum threshold can be kept and join the generation of the candidate subtrees in more levels. Hence, the candidate 3-subtrees are generated based on those frequent candidate 2-subtrees. The only difference is this time the parent nodes of the node pair to be picked for joining are the same as the leaf nodes, not the root node, in existing subtrees. Therefore, the frequent 2-subtrees are extended on one level more to become the candidate 3-subtrees. The same generation process is repeated for the candidate 4-subtrees and so on, until no more frequent *k*-subtrees can be found.

An example is given in Fig.3a. Assume that (*r; x*), (*r; y*) and (*r; z*) are frequent node pairs extracted from the given database D_{db}. As they all have node *r* as the parent, the candidate 1-subtrees can be obtained by enumerating the possible combinations of *r* (as the root, at level 0) and its child nodes *x, y* and *z* (level 1). Fig. 3b shows four candidate 1-subtrees, whose probabilities will be estimated and compared with a pre-defined threshold. Those with greater occurrence probabilities over the given threshold are kept to generate the candidate 2-subtrees. Based on the fact that if $p(x) > p(y)$ then $p(x) \cdot p(z) > p(y) \cdot p(z)$, we can have the following heuristic: if the *k*-subtree *Si* has a higher probability than *k*-subtree *Sj* , then *Si*'s extension of (*k*+1)-subtree has a higher probability than the same extension applied on *Sj* . Therefore, the occurrence probability estimation for each candidate subtree is another issue to be addressed.

IV. EXPERIMENTAL RESULTS

The Probability-based Frequent Subtree Pattern Mining (ProTreeMiner) algorithm is evaluated using both synthetic data sets and the data sets collected from the real applications.

Synthetic Data Sets We used the synthetic data set F5 from Zaki and Aggarwal in [13]. F5 is generated from a master tree W using user-specified parameters, which include the maximum fan-out F of a node, the maximum depth D of the tree, the total number of nodes M in the tree, and the number of node labels N. It is also allowed to have multiple nodes in the master tree to have the same label. We used the following default parameters values for the synthetic tree data set generation: the number of labels N = 100, the number of nodes in the master tree M = 10; 000, the maximum depth D = 10, the maximum fan-out F = 5, and the total number of subtrees T = 100; 000.

Real Data Set This data set consists of Web logs collected over one month at the Computer Science Department, PGP College. It contains 59; 691 user browsing trees, and touched 13; 361 unique Web pages within the department's Web site. The average number of nodes in a user browsing tree was 18.

Data Set	No of Trees	Max(Node)	Avg(Node)	Max(Depth)	Avg(Depth)
Synthetic Data Set F5	100,000	46	2.63	10	2.23
Real Data Set	59,691	428	12.94	86	4.43
CSLOGS	8,074	313	8.02	124	4.40
CSLOGS1	7,409	171	8.05	108	4.46
CSLOGS2	7,628	192	7.98	121	4.42
CSLOGS3					

Table 1: Experimental Data Sets

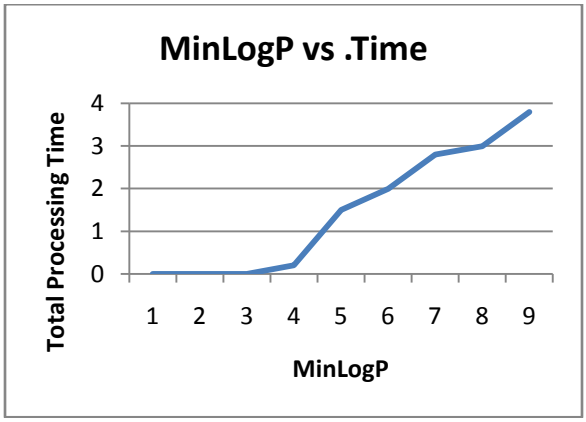
Three other real world data sets were also collected in the same way as *CSLOGS* [14], but they spanned three-weeks of records for XML user-sessions. Each week's log was separated into a different data set, and named by its week number. *CSLGOS1*, *CSLOGS2* and *CSLOGS3* are logs for the first, the second and the third week respectively. Each data set involved 7*;* 500 web logs, which can construct tree structures with various sizes and depths. The largest tree has 313 nodes in 12 levels; while the smallest tree only has two nodes. On average, a single tree contains around 8 nodes. These three data sets together with *CSLOGS* are used to validate the proposed algorithm in a practical setting.

4.1 Candidate Subtree Pattern Generation

We evaluate the performance of the candidate subtree pattern generation process using both F5 and CSLOGS data sets. When the list of the tree nodes were extracted from the data sets, we sorted them based on their occurrence frequency in an ascending order. This operation helps to quickly obtain candidate subtree patterns with the root nodes having a high occurrence frequency. The average occurrence frequency of a single tree node is 0:011 in F5, and a lower score of $7{:}4878e^{-5}$ is obtained from CSLOGS. We set the value range of MinLogP as [−7;−1] to test the generation capability of the proposed algorithm.

Performance Analysis on Synthetic Data Set

Set of extensive experiments were implemented to evaluate the performance of the probability-based candidate subtree pattern generation. We measured the processing time and the number of generated candidate subtree patterns versus different *MinLogP* threshold values. The reported processing time covers the whole working period of the subtree pattern generation, and includes the extraction of tree nodes and node pairs, the generation of candidate subtrees and the estimation of the occurrence probability. The number of candidate patterns counted the subtrees with occurrence probability greater than the specified *MinLogP*. For each run, we increased the *MinLogP* by 0:5 to track the performance. As shown in Fig. 4, the candidate subtree patterns were quickly generated using *ProTreeMiner*. 124*;* 132 candidate patterns were obtained in 865 seconds, and on average each candidate pattern only needed 0:00697 second.



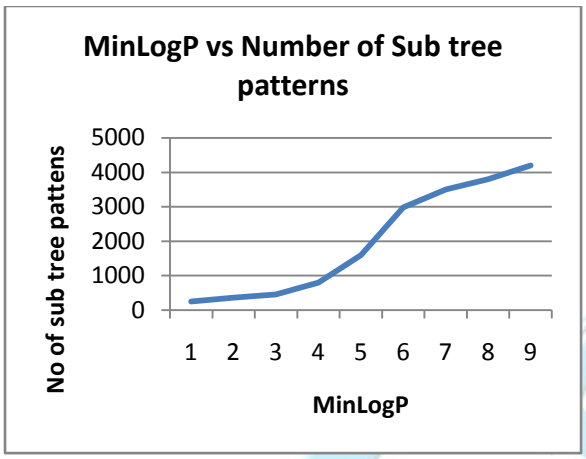


Figure 4: Performance of the Proposed *ProTreeMiner* Algorithm on *F5* Data Set

V.CONCLUSION

The proposed Probability-based Frequent Subtree Pattern Mining (ProTreeMiner) algorithm overcomes these mining costs by only generating a reasonable number of candidates that have a high probability of appearing in the database and employing a kernel-based filtering method to simplify the pruning process. When compared to the well-known UNI3 algorithm and the TreeMiner algorithm on both synthetic and real data sets, our proposed ProTreeMiner algorithm demonstrates outstanding performance to efficiently discover interesting frequent subtree patterns. This includs "partially matched" frequent subtree patterns scanning the database twice and with much lower memory costs. Unlike related algorithms, most of the candidate subtrees generated by ProTreeMiner are identified as interesting and only a few of them are discarded in the later process because they are generated based on the existing parent-child tree pairs that are directly extracted from the given tree database.

VI.REFERENCES

[1]. J. Vreeken. Making Pattern Mining Useful. PhD thesis, the Dutch Research School for Information and Knowledge Systems, Netherlands, 2009.

[2]. I.H. Witten, E. Frank, and A.M. Hall. Data Mining: Practical Machine Learning Tools and Techniques. Elsevier, 3rd edition edition, 2011.

[3]. R. Kosala and H. Blockeel. Web mining research: A survey. SIGKDD Explorations, 2(1):1–15, 2000.

[4]. Wikipedia. Mining, 2012. [Online, Accessed on 12-October-2011] http://en.wikipedia.org/wiki/Mining.

[5]. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future direction. Data Mining and Knowledge Discovery, 15:55–86,2007.

[6]. R. Agrawal, T. Imieinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, Proceedings of the ACM SIGMOD International Conference on the Management of Data, pages 207–216, Washington DC, 1993. ACM Press.

[7]. X. Yan. Mining, Indexing and Similarity Search in Large Graph Data Sets. Phd dissertation, University of Illionis, Urbana-Champaign, 2006.

[8]. R. Agrawal and R. Srikant. Mining sequential patterns. In Proceedings of the 11th International Conference on Data Engineering (ICDE'95), pages 3–14, Taipei, Taiwan, 1995.

[9]. Frequent Pattern Mining in Web Log Data Renáta Iváncsy, István Vajk Vol. 3, No. 1, 2006

[10]. Data Mining and Knowledge Discovery,C.I. EZEIFE of Windsor, Windsor, Fayyad, 10, 5–38, 2005

[11]. Metarule-Guided Mining of Multi-Dimensional Association Rules Using Data Cubes ,Micheline Kamber Jiawei Han Jenny Y. Chiang, From: KDD-97 Proceedings. Copyright © 1997

[12]. An Efficient Algorithm for Mining Association Rules in Large Databases Ashok Savasere Edward Omiecinski Shamkant Navathe

[13]. Efficient algorithm for mining unordered induced subtrees using TMG candidate generation Fedja Hadzic1 , Henry Tan1 and Tharam S. Dillon1

[14]. H. Tan, T.S. Dillon, F. Hadzic, L. Feng, and E. Chang. MB3-Miner: Mining eMBedded subTREEs using tree model guided candidate gerneration. In Proceedings of the 1st International Workshop on Mining Complex Data 2005 in conjunction with ICDM 2005, pages 103–110, Houston, Texas, USA, 2005.

[15]. M.J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. IEEE Transactions on Knowledge and Data Engineering, 17(8):1021–1035, 2005.