EfficientTreeMiner: Mining Frequent Induced Substructures from XML Documents without Candidate Generation

#P.Santhi Thilagam [1], Ananthanarayana V.S [2]
[1]*Dept of Computer Engineering,*
santhi_soci@yahoo.co.in
[2]*Dept of Information Technology,*
NITK-Surathkal, Karnataka- 575 025, India
anvs@nitk.ac.in

Abstract

Tree structures are used extensively in domains such as XML databases, computational biology, pattern recognition, computer networks, web mining, multi-relational data mining and so on. In this paper, we present an EfficientTreeMiner, a computationally efficient algorithm that discovers all frequently occurring induced subtrees in a database of labeled rooted unordered trees. The proposed algorithm mines frequent subtrees without generating any candidate subtrees. Efficiency is achieved by compressing the large database into a condensed data structure, namely prefix string representation, which reduces space complexity and by adopting a Frequent Immediate Descendents method that avoids the costly generation of candidate sets. Experimental results show that our algorithm has less time complexity when compared to existing approaches and is also scalable for mining both long and short frequent subtrees.

1. INTRODUCTION

For the last decades, data mining is extensively studied in theory and practice, and applied to various fields such as business, industry, and natural sciences [6]. On the other hand, by rapid progress of network and storage technologies, a huge amount of weakly structured data, called *semi-structured data* [4], [13], have been available on Internet and intranets. Hence, there have been increasing demands for efficient methods that extract rules and patterns from semi-
structured data [4], [14], namely *semi-structured data mining*. We can efficiently transform these semi-structured data into tree structures. So the frequent substructure mining problem [1] is reduced into the problem of frequent subtree mining. Thus, we cannot directly apply traditional data mining methods [8] for relational databases to semi-structured data. The previous studies adopt an Apriori-like approach [9], whose essential idea is to iteratively generate the set of candidate subtrees of size (k+1) from the set of frequent subtrees of size k (for $k \geq 1$), and check their corresponding occurrence frequencies in the database. An important heuristic adopted in these methods, called Apriori heuristic [2], [3], which may greatly reduce the size of candidate subtree set [11]. However, in situations with prolific frequent subtrees, long subtrees, or quite low minimum support thresholds, an Apriori-like algorithm may still suffer from the following two nontrivial costs:

i) It is costly to handle a huge number of candidate subtrees.
ii) It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long subtrees.

This is the motivation of this study, especially for mining databases containing a mixture of large numbers of long and short patterns. If one can avoid generating a huge set of candidate subtrees, the performance of frequent subtree mining can be substantially improved. This problem is solved as described in section 3.

2. RESEARCH PROBLEM STATEMENT

Given a collection of labeled unordered rooted Trees[12] and *minsupport* value, the problem is to find

1-4244-0716-8/06/$20.00 ©2006 IEEE.

out the frequent induced subtrees from the given collection of trees.

Given a threshold *minfreq*, a class of trees C, a transitive subtree relation $P \leq T$ between trees $P, T \in C$, a finite data set of trees $D \subseteq C$, the frequent tree mining problem is the problem of finding all trees $\mathcal{P} \subseteq C$ such that no two trees in $\mathcal{P}$ are isomorphic and for all $P \in \mathcal{P}$: $freq(P,D) = \Sigma_{T \in D} d(P,T) \geq minfreq$, where d is an anti-monotone function such that $\forall T \in C : d(P',T) \geq d(P,T)$ if $P' \leq P$.

We will always denote a *pattern tree* [15] – a tree which is part of the output $\mathcal{P}$ – with a P, and a *text tree* – which is a member of the data set D – with a T. The subtree relation $P \leq T$ defines whether a tree P occurs in a tree T. The simplest choice for function d is given by the indicator function:

$$d(P,T) = \begin{cases} 1 & \text{if } P \leq T \\ 0 & \text{otherwise.} \end{cases}$$

In this simple case the frequency of a pattern tree is defined by the number of trees in the data set that contains the pattern tree. We call this frequency definition, which closely matches that of item set frequency, a *transaction based frequency*. The indicator function is anti-monotone due to the transitivity of the subtree relation. From the transaction based frequency, one can also easily define a transaction based *support*: $sup(P, D) = freq(P, D)/|D|$.

3. FRAMEWORK FOR MINING FREQUENT INDUCED SUBTREES

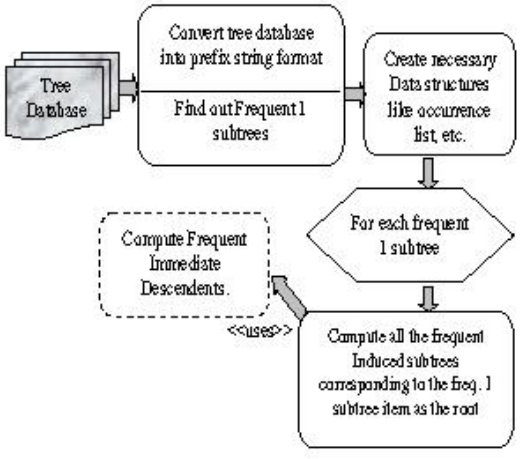

Fig. 1: Flow diagram for EfficientTreeMiner

The framework for the overall mining procedure is described in this section. There are two main steps for mining. The first step is preprocessing step which reads the tree database and converts it into its prefix string representation. Next step is to mine frequent induced subtrees. These steps are explained with examples in the next subsections. The overall workflow is shown diagrammatically in Fig.1.

The main issues to be considered while solving the problem of *mining frequent subtrees* are: i) Representation of large semi-structured data in the memory, ii) Enumeration of subtrees, iii) Frequency counting (for support pruning). The efficiency of any frequent subtree mining algorithm [1] lies in these three issues.

A. Representation of tree database

The representation of trees should be compact and unique for the given tree. A single tree should not be represented with more than one form. One should

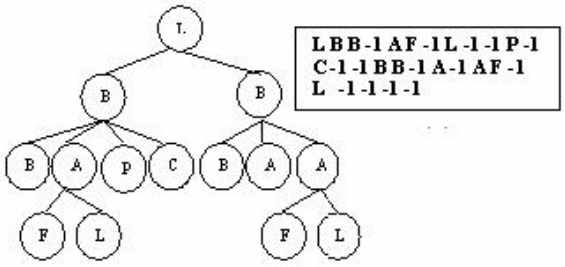

Fig. 2: Prefix string representation

choose the canonical representation and also the chosen representation should be useful and should not increase the complexity in the later stages of mining process.

The proposed approach uses the *prefix string representation*, which is a string obtained by preorder (depth first order) traversal of the tree. This representation was chosen because of its compact representation which takes O(N) space where N is the size of the tree (number of nodes in the tree). Fig.2. illustrates how a sample tree database is preprocessed, and its *prefix string* form is also given.

B. Mining induced subtrees

The proposed approach eliminates the generation of unnecessary candidate sets and also reduces the unnecessary frequency counting. The frequent sub trees are enumerated using the prefix string representation of the database of trees and using the Frequent Immediate Descendents (FID) a novel

method that we propose to efficiently find out the possible subtrees that might be frequent. In our approach, FID method computes the frequent immediate descendents for each occurrence of particular node in the tree database. The frequency counting process is improved with an ***occurrence list based approach***. Each frequent subtree is associated with an occurrence list which stores the information about the occurrences of that particular subtree in each tree in the tree database.

The size of the occurrence list is therefore bounded by the product of the size of the database and the size of the pattern. The time complexity can be reduced when compared to the naïve approach, as the frequency of occurrence of an enumerated subtree is simply obtained from the cardinality of the occurrence list. This proposed method for mining substructures is explained with an illustrative example in the next section

4. MINING ALGORITHM AND ANALYSIS

The high level structure of EfficientTreeMiner is shown in Fig.3. The main steps include preprocessing, finding frequent-1 subtrees and computing all other frequent induced subtrees using the frequent-1 subtrees and the *procedure* ComputeFISubtreesHavingRoot which is described in Fig.6. We will now explain the EfficientTreeMiner how it mines the frequent induced subtrees with an illustrative example.

Algorithm 1 : EfficientTreeMiner.
Input : *tree database, minsupp.*
Output : set of *induced frequent subtrees.*

Method :

1. Preprocessing tree database.
   a. Read the tree database and build the prefix string representation.

2. Find out all frequent-1 subtrees and build the occurrence lists.

3. **for each** *frequent-1 subtree <f>*
   a. *Freqst[]* = **ComputeFISubtreesHavingRoot(*f,OL(f)*)**
   b. **for each** *subtree* in *Freqst[]*
   i. Output the *subtree.*

Fig. 3: Algorithm for EfficientTreeMiner

Consider a tree database as shown in Fig.4 and the minimum support value, *minsupp*, is given as 2. Here the number of trees is three. We will first find out the frequent-1 subtrees. Frequent-1 subtrees are nothing but the subtrees having single node. During the first scan of tree database itself we can compute these. In this example, the nodes 1,2,3,4,5,8,9 are identified to be frequent-1 subtrees. Also construct the occurrence lists for these subtrees. This is shown in Fig.5(a). As given in *Algorithm 1*, we will compute the complete sets of frequent subtrees as a collection of equivalence classes where in each class represents the collection of trees whose root nodes are same. That means all the trees whose roots are same, belongs to the same class. Each class of subtrees is computed in the similar way.

Consider the class corresponding to node 1, (the trees having root as node 1). Construct a transaction database with the descendents of the node 1 at all its occurrences. The transaction database corresponding to node 1 in this example is {{2, 3, 8}, {3, 5, 8}}. By computing the Frequent Immediate Descendents from these we get the frequent item sets are {{3}, {8}, {3, 8}}. The frequent subtrees are constructed using these sets along with its occurrence lists as shown in Fig. 5(b). Consider tree {1 3 -1 -1}, now the transaction database is {{4, 1}, {4}}. The frequent subsets from FID are {{4}}. The identified frequent subtrees are shown in Fig.5(c). On similar lines, the whole procedure is repeated for all the remaining subtrees identified in the previous level and finally all the equivalence classes are computed.

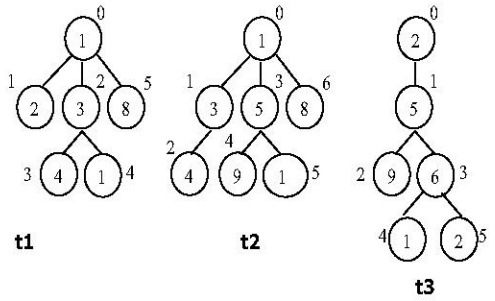

Fig. 4: The example Tree Database

We analyze the time and space complexity of the proposed algorithm. For representing tree database in the memory, the prefix string representation [10] is used. As the size of the tree increases, the corresponding prefix string size also increases. In the worst case FID method will take $O(|V_D|+2^k)$ time complexity which was given in [3] where k is the largest number of nodes that a node can have as immediate children which is usually very much less than all other factors of tree. So the overall worst case time complexity for EfficientTreeMiner algorithm is $O(l.m.((|V_D|)+2^k))$ where l is the number of distinct labels in the database and m is the number of nodes in the largest tree of the database.

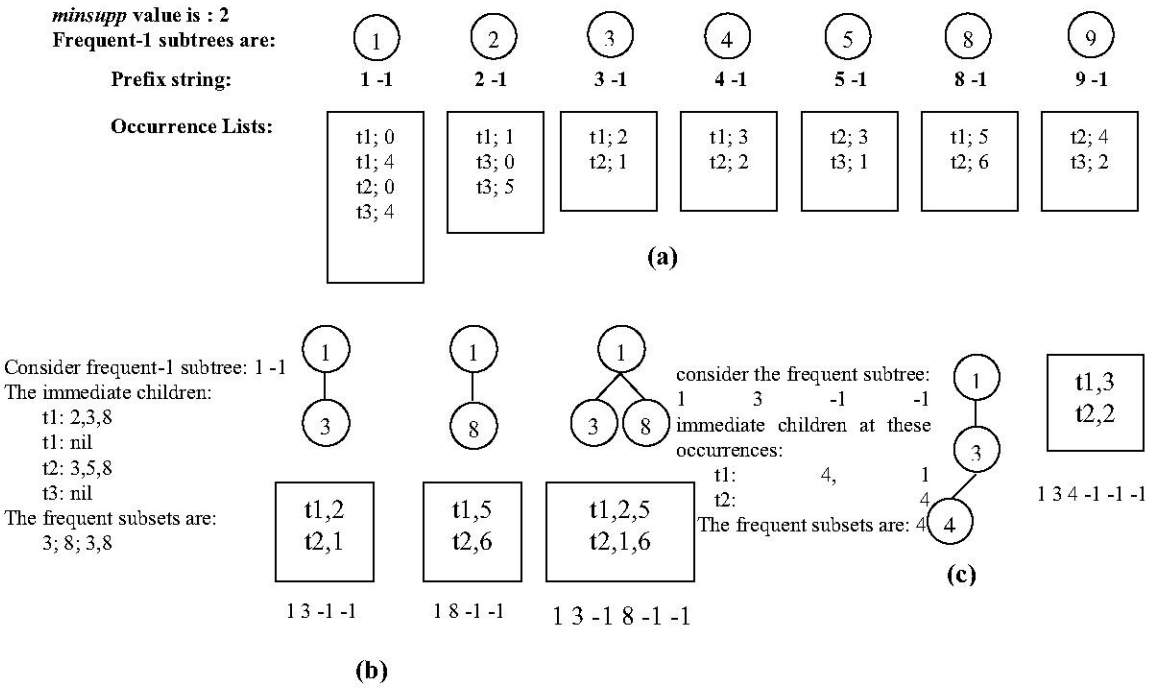

Fig. 5: Example for enumeration of frequent subtrees

Algorithm 2 : ComputeFISubtreesHavingRoot
Input : *f* : node, *OL* : OccurrenceList
Output : Class of frequent induced subtrees that are having the root as *f*.
Method :

1. According to *OL* traverse the tree database and get immediate descendents and construct Transaction database with those.
2. Find frequent subsets for the constructed transaction database using *FID method*.
3. **if** no frequent subsets are there then *return*.
   else construct the frequent subtrees using *f* as the root and the frequent subsets as its children. Also compute the occurrence lists for the computed frequent subtrees.
4. **for all** frequent subsets *fss*
   a. **if** *fss* is singleton then call
      ComputeFISubtreesHavingRoot (*fss, newocclist*)
   b. **else** call the recursive function for all the elements from the *fss* and get the combinations of those returned frequent subtrees
      Construct new frequent subtrees along with the occurrence lists.
5. Return the constructed frequent subtrees along with the occurrence lists.

Fig. 6: Algorithm for computing a class of frequent subtrees

5. EXPERIMENTAL EVALUATION

In this section, we will evaluate the performance of EfficientTreeMiner. The Proposed algorithm is compared with Unot [5]. The algorithm is also tested for the scalability with respect to the data size as well as minimum support. All the experiments are performed on a Pentium-IV processor, 2.40 GHz CPU, 256 MB RAM, 40 GB Hard disk. The operating system environment is Fedora Core 2. *C++* and *perl* are used for developing the proposed approach. The compiler for compiling the EfficientTreeMiner programs is gcc 3.2.2 and the *perl* interpreter is *perl* V*5.8.3*. We used *perl* for implementing the preprocessing. The tree mining algorithm is implemented in C++.

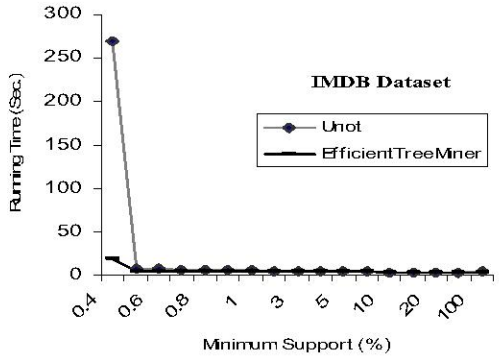

Fig 7(b): Minimum Support Vs. Running Time for IMDB

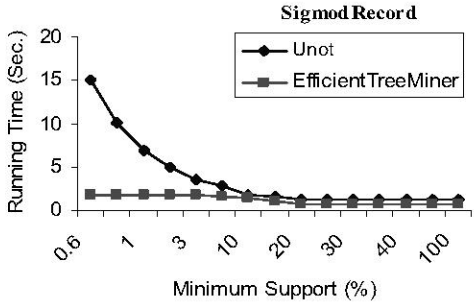

Fig. 7(a): Minimum Support Vs. Running Time for Sigmod record.

There are different ways of defining the support threshold. We implemented the algorithm both for root occurrence and document occurrence.

We considered the XML documents for evaluation purpose. Experiments were conducted using real datasets. Two popular sets among those are Sigmod records from UW repository [16] and IMDB (Internet Movie DataBase) [17]. We also tested EfficientTreeMiner for its scalability with respect to both the size of the dataset (number of trees) and the minimum support value which considers document occurrence of the frequent patterns. Fig. 7(a) shows the performances of two algorithms EfficientTreeMiner and Unot [5] with *sigmod dataset*. Fig. 7(b) shows the performance graph obtained for IMDB dataset [17]. From these graphs, we notice that the performance of EfficientTreeMiner is more stable than that of Unot. Fig. 7(c) shows the scalability graph with respect to the data size (number of trees). The scalability in terms of minimum support value for the EfficientTreeMiner is shown in Fig 7(d). From the experiments we conducted, we can conclude that the EfficientXMLMiner is an efficient way for Frequent induced subtree (or frequent induced substructure) mining for Labeled Rooted Unordered trees.

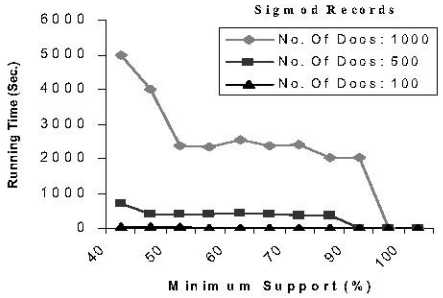

Fig. 7(c): Scalability graph for sigmod records with respect to the size of the data set

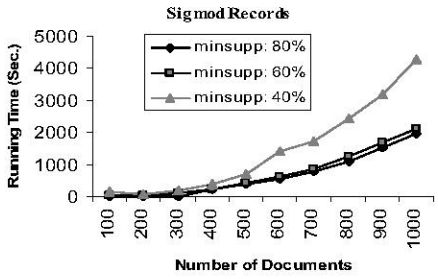

Figure 7(d). Scalability graph for sigmod records with respect to the minimum support value

6. CONCLUSION

In this paper, we have looked at the issue of mining frequent induced substructures from database of XML documents which are modeled as *labeled rooted unordered trees*. In the proposed approach, we reduce the space as well as the database scan complexities by compressing the huge tree database into compact data structure, prefix string representation. The proposed algorithm mines frequent induced subtrees without

generating any candidate subtrees which is the main bottleneck in the Apriori method. Overall, the mining procedure takes a single database scan. One way to measure the data mining algorithms is in terms of its speed. So, considering speed as the measurement, the proposed EfficientXMLMiner algorithm performs better than the other existing algorithms for mining frequent induced subtrees. The time and space complexity analysis and the experimental results show that our algorithm is more efficient than the existing algorithms and is also scalable for varying data size and minimum support values. Our algorithm ca be extended to employ user-defined constraints and the use of condensed representations. This leads *to constraint-based mining of maximal patterns and closed patterns* [7] for trees.

REFERENCES

[1]Abe. K., Kawasoe. S., Asai. T., Arimura. H., Sakamoto. H., Arikawa. S., "Mining Frequent Substructures from Web", *Active Mining –New Directions of Data Mining–*, IOS Press, 2002, pp. 83-94.

[2]Agrawal. R., and Srikant. R. "Fast algorithms for mining association rules". In Proc. 1994 Int. Conf. Very Large Data Bases, Santiago, Chile, September 1994, pp. 487–499.

[3]Agrawal, R., Manilla, H., Srikant, R., Toivonen, H., Verkamo, A.: Fast Discovery of Association Rules. In: U.M. Fayyad et. Al. (cds). Advances in Knowledge Discovery and Datamining. AAAI/MIT Press. (1996).

[4]Asai, T., Abe, K., Kawasoe, S., Arimura, H., Satamoto, H., Arikawa, S.: "Efficient Substructure Discovery from Large Semi-Structured Data", *2nd SIAM Int. Conf. on Data Mining,* April 2002.

[5]Asai, T., Arimura, H., Uno, T., Nakano, S.: "Discovering Frequent Substructures in Large Unordered Trees", *The 6th International Conference on Discovery Science,* October 2003.

[6]Chi, Y., Nijssen, S., Muntz, R., Kok, J. N.,: "Frequent Subtree Mining – An overview", *Fundamental Informaticae Special Issue on Graph and Tree Mining*, 2005, Pages:1001-1037, IOS Press.

[7]Chi, Y., Yang, Y., Xia, Y., Muntz, R. R.:"CMTreeMiner: Mining Both Closed and Maximal Frequent Subtrees", *The Eighth Paci_c Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, May 2004.

[8] Han, J., Pei, J., and Yin, Y. 2000. "Mining frequent patterns without candidate generation", *In Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00),* Dallas, TX, pp. 1-12.

[9]Inokuchi. A., Washio. T., Motoda. H., "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data", In *Proc. the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00)*, LNAI 1910, Springer-Verlag, 2000, pp. 13-23.

[10]Kiselyov. O., "A better XML parser through functional programming -- *Fourth International Symposium on Practical Aspects of Declarative Languages", (PADL '02). January 2002*

[11]Lakshmanan. R. Ng, L. V. S., Han. J., and Pang. A., Exploratory mining and pruning optimizations of constrained associations rules. In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data, Seattle, Washington, June 1998, pp.13-24.

[12]Nijssen, S., Kok, J. N., "Efficient Discovery of Frequent Unordered Trees", *First International Workshop on Mining Graphs, Trees and Sequences,* 2003.

[13]Nijssen, S., Kok, J. N., "A Quickstart in Frequent Structure Mining Can Make a Difference", *Proc. of the 2004 Int. Conf. Knowledge Discovery and Data Mining (SIGKDD'04)*, August 2004.

[14]Termier. A., Rousset. M., Sebag. M., "TreeFinder: a First Step towards XML Data Mining", In *Proc. the 2002 IEEE International Conference on Data Mining (ICDM'02)*, IEEE Computer Society, 2002, pp. 450-457.

[15]Zaki, J.: "Efficiently Mining Frequent Trees in a Forest", In proceedings of the SIGKDD'02, Edmonton, Canada (2002).

[16]XMLDataSets *www .cs . washington. edu/research/XMLdasets*

[17]The Internet Movie Database (IMDb) http://us.imdb.com/