DOI: https://doi.org/10.53555/nncse.v6i9.790 **Publication URL: http://nnpub.org/index.php/CSE/article/view/790**

"A NARRATIVE APPROACH FOR REMOVAL EMBEDDED PROTOTYPE FROM BIG TREE DATA"

K. Valarmathi[1]*, R. Gowdhami[2]

*[*1]Asst. Professor and Head, Department of Computer science, Trinity College for Women (Arts and Science) Namakkal*

[2]Research Scholar, Department of Computer science, Trinity College for Women (Arts and Science), Periyar University, Salem

***Corresponding Author**: -*

Abstract: -

Many modern functions and systems represent and exchange data in treestructured form and process and produce large tree datasets. Discovering informative patterns in large tree datasets is an important research area that has many practical applications. We propose a novel approach that exploits efficient homomorphic pattern matching algorithms to compute pattern support incrementally and avoids the costly enumeration of all patterns matching required by previous approaches. To reduce space consumption, matching information of already computed patterns is materialized as bitmaps. We further optimize our basic support computation method by designing an algorithm which incrementally generates the bitmaps of the embeddings of a new candidate pattern without first explicitly computing the embeddings of this pattern. Our extensive experimental results on real and synthetic large-tree datasets show that our approach displays orders of magnitude performance improvements over a state-of-the-art tree mining algorithm and a recent graph mining algorithm

Keywords: - ***Data*** *mining, Algorithm design and analysis, Informatics, Computational modeling, Databases, Upper bound, Data models.*

I. INTRODUCTION

Extracting frequent tree patterns which are hidden in data trees is central for analyzing data and is a base step for other data mining processes including association rule mining, clustering and classification. Trees have emerged in recent years as the standard format for representing, exporting, exchanging and integrating data on the
web (e.g., XML and JSON). Tree data are adopted in various application areas and systems such as business process management, NoSQL databases, key-value stores, scientific workflows, computational biology and genome analysis. Because of its practical importance, tree mining has been extensively studied. The approaches to tree mining can be basically characterized by two parameters: (a) the type of morphism used to map the tree patterns to the data structure and (b) the type of mined tree data.

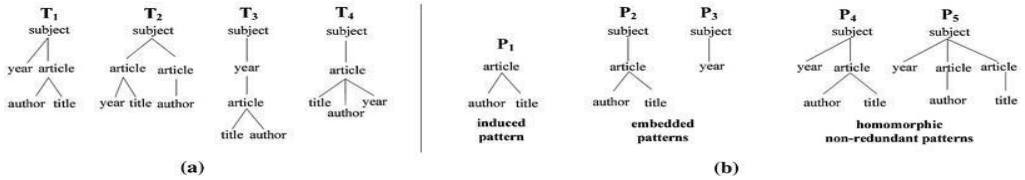

Fig. 1 Different types of mined tree patterns occurring in three of the four data trees. a Data trees, b mined tree patterns

Mining homomorphic tree patterns The morphism determines how a pattern is mapped to the data tree. The morphism definition depends also on the type of pattern considered. In the literature, two types of tree patterns have been studied: patterns whose edges represent parent-child relationships (child edges) and patterns whose edges represent ancestor-descendant relationships (descendantedges). Over the years, research has evolved from considering isomorphisms for mining patterns with child edges (induced patterns) to considering embeddings for mining patterns with descendant edges (embedded patterns). Because of the descendant edges, embeddings are able to extract patterns "hidden" (or embedded) deep within large trees which might be missed by the induced pattern definition. Nevertheless, embeddings are still restricted because: (a) They are injective (one-to-one), and (b) they cannot map two sibling nodes in a pattern to two nodes on the same path in the data tree. On the other hand, homomorphisms are powerful morphisms that do not have those two restrictions of embeddings. We term patterns with descendant edges, mined through homomorphisms, homomorphic patterns. As homomorphisms are more relaxed than embeddings, the mined homomorphic patterns are a superset of the mined embedded patterns.

Figure 1a shows four data trees corresponding to different schemas to be integrated through the mining of large tree patterns. The frequency threshold is set to three. Figure 1b shows induced mined tree patterns, embedded patterns and non-redundant homomorphic patterns. Figure 1b includes the largest patterns that can be mined in each category. As one can see, the shown embedded patterns are not induced patterns, and the shown homomorphic patterns are neither embedded nor induced patterns.
Further, the homomorphic patterns are larger than all the other patterns.

Large patterns are more useful in describing data. Mining tasks usually attach much greater importance to patterns that are larger in size, e.g., longer sequences are usually of more significant meaning than shorter ones in bioinformatics. As mentioned in, large patterns have become increasingly important in many modern applications.

Therefore, homomorphisms and homomorphic patterns display a number of advantages. First, they allow the extraction of patterns that cannot be extracted by embedded patterns. Second, extracted homomorphic patterns can be larger than embedded patterns. Finally, homomorphisms can be computed more efficiently than embeddings. Indeed, the problem of checking the existence of a homomorphism of an unordered tree pattern to a data tree is polynomial, while the corresponding problem for an embedding is NP-complete.

Mining patterns from a large data tree The type of mined data can be a collection of small trees or a single large tree. Surprisingly, the problem of mining tree patterns from a single large tree has only very recently been touched even though a plethora of interesting datasets from different areas are in the form of a single large tree. Examples include encyclopedia databases like Wikipedia, bibliographic databases like PubMed, scientific and experimental result databases like UniprotKB, and biological datasets like phylogenetic trees. These datasets grow constantly with the addition of new data. Big data applications seek to extract information from large datasets. However, mining a single large data tree is more complex than mining a set of small data trees. In fact, the former setting is more general than the latter, since a collection of small trees can be modeled as a single large tree rooted at a virtual unlabeled node. Existing algorithms for mining embedded patterns from a collection of small trees cannot scale well when the size of the data tree increases. Our experiments show that these algorithms cannot scale beyond some hundreds of nodes in a data tree with lowfrequency thresholds.

The problem Unfortunately, previous work has focused almost exclusively on mininginduced and embedded patterns from a set of small trees. The issue of mining homomorphic patterns from a single large data tree has been neglected.

The challenges Mining homomorphic tree patterns is a challenging task. Homomorphic tree patterns are difficult to handle as they may contain redundant nodes. If their structure is not appropriately constrained, the number of frequent patterns (and therefore the number of candidate patterns that need to be generated) can be infinite.

The support of patterns in the single large data tree setting cannot be anymore the number of trees that contain the pattern as is the case in the multiple small trees setting. A new way to define pattern support in the new setting is needed which enjoys useful monotonic characteristics. Typically, one can deal with a large number of frequent patterns, by computing only maximal frequent patterns. In the context of induced tree patterns, a pattern is maximal if there is no frequent superpattern. A nonmaximal pattern is not returned to the user as there is a larger, more specific pattern, which is frequent. However, in the context of homomorphic patterns, which involve descendant edges, the concept of superpattern is not sufficient for capturing the specificity of a pattern.

II. PROBLEM DEFINITION

Trees and inverted lists We consider rooted labeled trees, where each tree has a distinguished root node and a labeling function lb mapping nodes to labels. A tree is called ordered if it has a predefined left-to-right ordering among the children of each node. Otherwise, it is unordered. The size of a tree is defined as the number of its nodes. In this paper, unless otherwise specified, a tree pattern is a rooted, labeled, unordered tree. For every label a in an input data tree T, we construct an inverted list La of the data nodes with label a ordered by their pre-order appearance in T. Figure 2a, b shows a data tree and inverted lists of its labels.

Tree morphisms There are two types of tree patterns: patterns whose edges represent child relationships (child edges) and patterns whose edges represent descendant relationships (descendant edges). In the literature of tree pattern mining, different types of morphisms are employed to determine whether a tree pattern is included in a tree.

III. PROPOSED APPROACH

Our approach for mining homomorphic tree patterns from a large tree iterates between the candidate generation phase and the support counting phase. In the first phase, we use a systematic way to generate candidate patterns that are potentially frequent. In the second phase, we develop an efficient method to compute the support of candidate patterns.

Candidate Generation To generate candidate patterns, we adapt in this section the equivalence class-based pattern generation method proposed in so that it can address pattern redundancy and maximality. A candidate pattern may have multiple alternative isomorphic representations. To minimize the redundant generation o the isomorphic representations of the same pattern, we employ a canonical form for tree patterns

3.1.1 Equivalence Class-Based Pattern Generation

Let P be a pattern of size k-1. Each node of P is identified by its depth-first position in the tree, determined through a depth-first traversal of P, by sequentially assigning numbers to the first visit of the node. The rightmost leaf of P, denoted rml, is the node with the highest depth-first position. The immediate prefix of P is the subpattern of P obtained by deleting the rml from P. The equivalence class of P is the set of all the patterns of size k that have P as their immediate prefix. We denote the equivalence class of P as [P]. Any two members of [P] differ only in their rmls.

3.1.2 Checking Pattern Redundancy

The pattern generation process may produce candidates which are redundant. We discuss below how to efficiently check pattern redundancy by identifying redundant nodes. We exploit a result of which states that: A node X of a pattern P is redundant iff there exists a homomorphism h from P to itself such that h(x) != x. A brute-force method for checking whether a pattern is redundant computes all the possible homomorphisms from P to itself.

The number of expandable patterns enumerated by the equivalence class expansion process can still be very large, particularly when the frequent patterns to find have both a high depth and a high branching factor. In order to further reduce the number of generated patterns, we present below a pattern refining method which exploits properties of the equivalence class-based pattern expansion. We observe that the specificity relation _ induces a linear order on patterns in a given equivalence class whose rightmost leaf nodes have the same labels.

We present now our homomorphic tree pattern mining algorithm called HomTreeMiner. The first part of the algorithm computes the sets containing all frequent 1-patterns F1 (i.e., nodes) and 2-patterns F2 (lines 1–2). F1can be easily obtained by finding inverted lists of T whose size (in terms of number of nodes) is no less than minsup.The total time for this step is O(|T|). F2 is computed by the following procedure: Let X / Y denote a 2-pattern formed by two elements X and Y of F1. The support of X / Y is computed via algorithm TwigStack on the inverted listsLlbðXÞ and LlbðYÞ that are associated with labels lb(X) and lb(Y), respectively. The total time for each 2-pattern candidate is O(|T|).

Experimental Evaluation

We implemented our algorithm HomTreeMiner and we conducted experiments to: (a) compare the features of the extracted (maximal) homomorphic patterns with those of (maximal) embedded patterns and (b) study the performance of HomTreeMiner in terms of execution time, memory consumption and scalability. To evaluate the effect of the pattern refining technique described in consider also a basic version of HomTreeMiner that does not employ that optimization in its mining process. That basic version was introduced in and is called HomTMBasic in the following paragraphs.

To the best of our knowledge, there is no previous algorithm computing homomorphic patterns from data trees. Therefore, we compared the performance of our algorithm with state-of-the-art algorithms that compute embedded patterns on the same dataset.

Datasets We have ran experiments on three real and benchmark datasets with different structural properties. Their main characteristics are summarized in Table 1. Treebank1 is a real XML dataset derived from computation linguistics. It models the syntactic structure of English text and provides a hierarchical representation of the sentences in the text by breaking them into syntactic units based on part of speech. The dataset is deep and comprises highly recursive and irregular structures. XMark2 is an XML benchmark dataset generated using the data generator with factor = 0.05. It is deep and has many regular structural patterns. It includes very few recursive elements.

Table 1 Dataset Statistics

Dataset	Tot. #nodex	#lables	Max/avg depth	#paths
Treebank	2,437.6666	250	36/8.4	1,392,231
XMark	83,533	74	12/5.6	60,853
CSlogs	772,188	13,355	86/4.4	59,691(#trees)

We compare the performance of HomTreeMiner with two unordered embedded tree mining algorithms Sleuth and EmbTreeMiner. Sleuth was designed to mine embedded patterns from a set of small trees. In order to allow the comparison in the single large tree setting, we adapted Sleuth by having it return as support of a pattern the number of its root occurrences in the data tree. EmbTreeMiner is a newer embedded tree mining algorithm which, as HomTreeMiner, exploits the twig-join approach and bitmaps to compute pattern support. To the best of our knowledge, direct mining of maximal embedded patterns has not been studied in the literature. We therefore use post-processing pruning which eliminates non-maximal patterns after computing all frequent embedded patterns. For this task, we implemented the unordered tree inclusion algorithm described in. As our experiments show, the cost of this postprocessing step is in general not significant compared to the frequent pattern mining cost.

Some patterns within a reasonable amount of time, we used a fraction of the Treebank dataset which consists of 35% of the nodes of the original tree. We measured execution times over the entire Treebank dataset in the scalability experiment. We computed different statistics on frequent and maximal frequent patterns mined by HomTreeMiner and Emb-TreeMiner from the three datasets varying the support; the results are summarized in Table 2. For the comparison, we considered only maximal embedded patterns that contain no redundant nodes. We show the total number of maximal embedded patterns in parenthesis in Column 5. We can make the following observations.

Table 2 Statistics for maximal frequent patterns mined from the three datasets

Dataset	Morphism	# freq. patterns	# loc.max patterns	# max. non. red.patterns	%max. over freq. patterns	Average #nodes	Average height	Average fanout	maximum #nodes	#common max.patterns
Treebank	Emb	78	n/a	2 (8)	2.6	0.63	0.375	0.25	3	1
(minsup = 35k)	Hom	521	158	9	1.7	5	2.11	2.11	8	
Treebank	Emb	175	n/a	13 (32)	7.4	1.47	0.66	0.78	5	5
(minsup = 30k)	Hom	2937	915	35	1.2	6.14	2.23	2.57	9	
XMark	Emb	934	n/a	14 (19)	1.5	2.63	1.05	1.58	5	6
(minsup = 800)	Hom	853	26	15	1.76	4.67	1.93	2.6	10	
XMark	Emb	43,441	n/a	27 (54)	0.06	3.33	1	2.09	15	14
(minsup = 550)	Hom	56,160	302	35	0.06	8.74	2.29	5.71	15	
CSlogs	Emb	638	n/a	133 (164)	20.8	2	0.896	1.1	5	119
(minsup = 400)	Hom	816	307	152	18.6	2.53	1.11	1.41	6	
CSlogs	Emb	2192	n/a	250 (375)	11.4	1.68	0.728	0.95	6	192
(minsup = 280)	Hom	1625	676	312	19.2	2.8	1.22	1.57	6	

First, HomTreeMiner is able to discover larger patterns than EmbTreeMiner for the same support level. As one can see in Table 2, the maximum size of frequent homomorphic patterns and the maximum size and average number of nodes, height and fanout of maximum frequent homomorphic patterns is never smaller (substantially larger in many cases) than that of the embedded patterns for the same support level.

Second, the number of homomorphic and embedded frequent patterns is substantially reduced if only maximal patterns are selected (Column 6 of Table 2). However, the effect is larger on homomorphic patterns as the number of frequent homomorphic patterns is usually larger than that of embedded patterns for the same support level (Column 3 of Table 2).

Third, by further looking at the mined maximal patterns, we find that the embedded maximal patterns at a certain support level can be partitioned into sets which correspond one-to-one to the maximal homomorphic patterns at the same support level so that all the embedded patterns in a set are less specific than the corresponding homomorphic pattern. Figure 14 shows, for each of the three datasets, examples of embedded maximal patterns each from the same set in the partition and the corresponding maximal homomorphic pattern. Therefore, for a number of applications, maximal homomorphic patterns can offer more information in a more compact way.

IV. CONCLUSION

We have provided a novel definition of maximal homomorphic patterns which takes into account homomorphisms, pattern specificity and the single tree setting. We have designed an efficient algorithm that discovers all frequent non-redundant maximal homomorphic tree patterns. Our approach employs an incremental stack-based frequency computation method that avoids the costly enumeration of all pattern occurrences required by previous approaches. An originality of our method is that matching information of already computed patterns is materialized as bitmaps, which greatly reduces both memory consumption and computation costs. An optimization technique further prunes the search space of candidate patterns. We have conducted extensive experiments to compare our approach with tree mining algorithms that mine embedded patterns when applied to a large data tree. Our results show that maximal homomorphic patterns are fewer and larger than maximal embedded tree patterns.

Our algorithm is as fast as the state-of the art algorithm mining embedded trees from a single tree while outperforming it in terms of memory consumption and scalability. Several applications are interested in extracting not all the frequent patterns, but only those that comply with a number of restrictions. We are currently working on incorporating user-specified constraints to the proposed approach to enable constraintbased homomorphic pattern mining.

REFERENCES:

[1].Asai T, Arimura H, Uno T, Nakano S-I (2003) Discovering frequent substructures in large unordered trees. In: Discovery, Science, pp 47–61

[2].Bruno N, Koudas N, and Srivastava D (2002) Holistic twig joins: optimal XML pattern matching. In: SIGMOD, pp 310–321.

[3].Chi Y, Xia Y, Yang Y, Muntz RR (2005) Mining closed and maximal frequent subtrees from databases of labeled rooted trees. IEEE Trans Knowl Data Eng 17(2):190–202

[4].Chi Y, Yang Y, and Muntz RR (2004) Hybridtreeminer: an efficient algorithm for mining frequent rooted trees and free trees using canonical form. In: SSDBM, pp 11– 20

[5].Chi Y, Yang Y, Muntz RR (2005) Canonical forms for labeled trees and their applications in frequent subtree mining. Knowl Inf Syst 8(2):203–234.

[6].Dries A, Nijssen S (2012) Mining patterns in networks using homomorphism. In: SDM, pp 260–271

[7].Feng Z, Hsu W, and Lee M-L (2005) Efficient pattern discovery for semistructured data. In: ICTAI, pp 294–301

[8].Goethals B, Hoekx E, and den Bussche JV (2005) Mining tree queries in a graph. In: KDD, pp 61–69

[9].Kibriya AM, Ramon J (2013) Nearly exact mining of frequent trees in large networks. Data Min Knowl Discov 27(3):478–504

[10]. Kilpela¨inen P, Mannila H (1995) Ordered and unordered tree inclusion. SIAM J Comput 24(2):340–356

[11]. Miklau G, Suciu D (2004) Containment and equivalence for a fragment of xpath. J ACM 51(1):2–45

[12]. Nijssen S, Kok JN (2004) A quickstart in frequent structure mining can make a difference. In: KDD, pp 647652

[13]. Tan H, Hadzic F, Dillon TS, Chang E, Feng L (2008) Tree model guided candidate generation for miningfrequentsubtrees from xml documents. TKDD 2(2):1–43

[14]. Tatikonda S, Parthasarathy S, Kurc¸ TM (2006) Trips and tides: new algorithms for tree mining. In: CIKM, pp 455–464

[15]. Termier A, Rousset M-C, Sebag M (2002) Treefinder: a first step towards xml data mining. In: ICDM, pp 450–457.