DRYADE: a new approach for discovering closed frequent trees in heterogeneous tree databases

Alexandre Termier*, Marie-Christine Rousset & Michèle Sebag
CNRS & Université Paris-Sud (LRI) - INRIA (Futurs)
Building 490, Université Paris-Sud, 91405 Orsay Cedex, France.
{termier, mcr, sebag}@lri.fr

Abstract

In this paper we present a novel algorithm for discovering tree patterns in a tree database. This algorithm uses a relaxed tree inclusion definition, making the problem more complex (checking tree inclusion is NP-complete), but allowing to mine highly heterogeneous databases. To obtain good performances, our DRYADE *algorithm discovers only closed frequent tree patterns.*

1. Introduction

With the rapid growth of structured documents (eg., XML documents) available online, discovering frequent tree structures in huge collections of tree data becomes a crucial issue for information extraction. In this paper, we propose a novel algorithm for discovering frequent trees. It has two main distinguishing features. First, it handles a tree inclusion definition which is more general than all those considered in the existing tree mining literature, thus leading to the discovery of non trivial pattern trees even in highly heterogeneous collections of tree data. Second, it computes *closed* frequent trees, which has the advantage to provide a compact representation of frequent trees without loss of information. The paper is structured as follows: in section 2, we give the formal background for tree mining. The state of the art is briefly reviewed in section 3. In section 4 we describe the DRYADE algorithm, and in section 5 we give some experimental results. The section 6 concludes this paper and provides some research perspectives.

2. Formal Background

Let $L = \{l_1, ..., l_n\}$ be a set of labels. A *labelled tree* $T = (N, A, root(T), \varphi)$ is an acyclic connected graph, where N is the set of nodes, $A \subset N \times N$ is a binary relation over N defining the set of edges, $root(T)$ is a distinguished node called the *root*, and φ is a labelling function $\varphi : N \mapsto L$ assigning a label to each node of the tree.

Let $u \in N$ and $v \in N$. If there exists an edge $(u, v) \in A$, then v is a *child* of u, and u is the *parent* of v. If there exists a path from u to v in the tree ($(u, v) \in A^+$), then v is a *descendant* of u, and u is an *ancestor* of v.

Ancestor tree inclusion : Let $T_1 = (N_1, A_1, root(T_1), \varphi_1)$ and $T_2 = (N_2, A_2, root(T_2), \varphi_2)$ be two trees. T_1 is *included* into T_2 (noted $T_1 \sqsubseteq T_2$) if there exists an injective mapping $\mu : N_1 \mapsto N_2$ such that: **1.** μ preserves the labels : $\forall u \in N_1 \quad \varphi_1(u) = \varphi_2(\mu(u))$ and **2.** μ preserves the ancestor relationship : $\forall u, v \in N_1$ if $(u, v) \in A_1$ then $(\mu(u), \mu(v)) \in A_2^+$.

The set of mappings supporting the ancestor tree inclusion (or tree inclusion when no confusion is possible) is denoted $\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$. The set of *occurrences* of T_1 in T_2, denoted $Locc_{\sqsubseteq}(T_1, T_2)$, is defined as the set of nodes $\mu(root(T_1))$, where μ ranges over $\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$. Similarly, the set of *images* of T_1 in T_2 is the set of trees $\mu(T_1)$ where μ ranges over $\mathcal{EM}_{\sqsubseteq}(T_1, T_2)$.

Frequent trees : Let $TD = \{T_1, ..., T_m\}$ be a tree database. The *datatree* $\mathcal{D}$ is the tree whose root is an unlabelled node, and whose subtrees are the trees $\{T_1, ..., T_m\}$. Our goal is to find *frequent trees* in this datatree. Let ε be an absolute frequency threshold. P is a frequent tree of $\mathcal{D}$ if P has at least ε occurrences in $\mathcal{D}$ i.e. $\mid Locc_{\sqsubseteq}(P, \mathcal{D}) \mid \geq \varepsilon$.

A frequent tree T is *closed* if either i) T it is not included in any other frequent tree, or ii) for any frequent tree T' such that $T \sqsubseteq T'$, there exists at least one node in $Locc_{\sqsubseteq}(T, \mathcal{D})$ which is not contained in the image of T' in $\mathcal{D}$.

Testing the ancestor tree inclusion is a NP-complete problem [9]. In order to reduce the combinatorial explosion, we impose a second order restriction on the trees to be found by our algorithm DRYADE: from now on we restrict the discovery task to trees that do not contain two siblings with the same label. We call such trees *patterns*.

*Present address : ISIR, Osaka University, Japan

3. Related work

The first tree mining algorithms were proposed in 2002 by Asai et al. [1] and Zaki [13]. Both of these algorithms are based on efficient enumeration techniques, but handle simple tree inclusion definitions where the *sibling order* must be preserved by mapping μ. Asai's approach adds another constraint : μ must preserve the parent relationship instead of the ancestor relationship, further restraining the admissible solutions.

In 2003, several attempts were done to extend the above approaches and get rid of the constraint on the sibling order [2, 5], using canonical representations of the unordered trees to perform an efficient candidate enumeration.

In [12], we first introduced the general tree inclusion considered in this paper and presented the (incomplete) TreeFinder algorithm.

The most recent frequent tree algorithm to our best knowledge was proposed by Chi et al. [6]. In this *CMTreeMiner* algorithm, the search is restricted to closed trees, entailing significant savings with respect to both memory and computational resources. However, *CMTreeMiner* is based on a tree inclusion definition where the mapping preserves the parent relationship. Thus, as in [1, 2, 5, 13], the complexity of testing the considered tree inclusion is polynomial.

For truly heterogeneous databases, imposing either constraint (preserving the parent relationship or the sibling order) puts severe restrictions on the target solutions: not handling the variations in the order and nesting of the node labels result in finding many equivalent subtrees, and ultimately missing worthy subtrees.

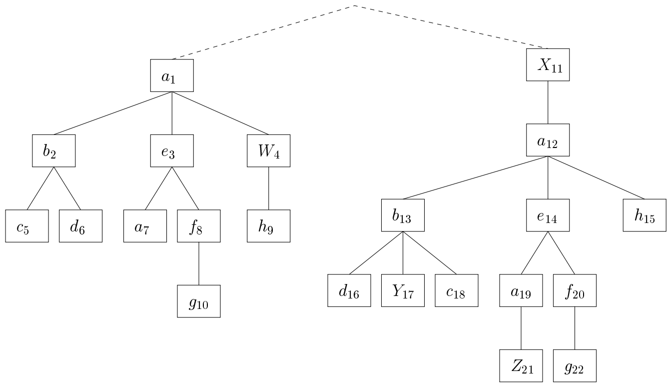

Figure 1. Datatree example *Legend a_i denotes a node with label a and identifier i (unique). The target frequent tree includes all labels with lowercase letters; capital letters corresponding to additional nodes.*

4 The DRYADE algorithm

The DRYADE algorithm presented in this section makes use of the most general tree inclusion proposed in section 2.

The basic principle in DRYADE is to discover the closed frequent patterns of depth 1, and then to hook them together in order to build the higher depth closed frequent patterns in a levelwise fashion. The specificity of the DRYADE approach is to intensively use task decomposition and reformulation in order to perform all the frequency tests *using propositional algorithms*, focusing on the different depth levels of the closed frequent patterns to discover.

DRYADE is presented step-by-step in the following subsections. The results of each step are illustrated on the example datatree of figure 1, with $\varepsilon = 2$.

4.1 Discovering patterns of depth 1

The first step is to compute the closed frequent patterns of depth 1. This task is delegated to a closed Frequent Item Set algorithm, by reformulating the data as follows: for each label $l \in L$, create a matrix M_l with as many lines as nodes of label l and as many columns as existing distinct labels in the datatree such that the boolean sign in the cell corresponding to the node u and the label l' indicate that the node u (of label l) has a descendant of label l'. Then a closed Frequent Item Set algorithm applied to M_l with threshold ε will provide all the closed frequent sets of descendants for nodes of label l in the datatree. The patterns whose roots are labelled by l and whose children are the nodes belonging to the closed frequent itemsets found previously are all the closed frequent patterns of depth 1 whose root has label l. The tid-list for each closed frequent pattern of depth 1 provides the set of occurrences of the corresponding closed frequent patterns of depth 1.

Here is the matrix of the labels of descendants of nodes of label a in our example:

a	a	b	c	d	e	f	g	h	W	X	Y	Z
1	X	X	X	X	X	X	X	X	X			
7												
12	X	X	X	X	X	X	X	X			X	X
19												X

With threshold $\varepsilon = 2$, the closed frequent itemsets are $\{a, b, c, d, e, f, g, h\}$ for occurrences $\{1, 12\}$, and $\{Z\}$ for occurrences $\{12, 19\}$. The closed frequent patterns of depth 1 are shown in figure 2.

We call $\mathcal{F}^1$ the set of closed frequent patterns of depth 1.

4.2 Pattern hooking

Let $\mathcal{F}$ denote the current set of closed frequent patterns, initialized to $\mathcal{F}^1$. DRYADE proceeds by hooking frequent patterns of $\mathcal{F}^1$ onto the patterns in $\mathcal{F}$, to gradually obtain deeper frequent trees.

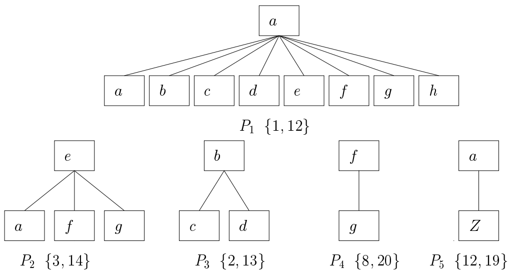

Figure 2. The closed frequent patterns of depth 1 and their occurrences list

The challenge here is to prevent redundant hookings (e.g. resulting in non closed frequent patterns). To this aim, we first define the set of candidate hookable patterns, and use a propositional reformulation to determine which candidate patterns can be hooked simultaneously.

Candidate hooking patterns. A binary relation on the closed patterns is defined as follows. Pattern Q is hookable on pattern P iff i) the root label of Q is one of the leaf labels in P; ii) there exists at least one occurrence u of P and one occurrence v of Q such that u is an ancestor of v.

This binary relation induces a stratification on the current closed patterns (which is updated at each level). As formally proved in [11], the search can be restricted with no loss of information by hooking patterns Q on P, where i) P is maximal wrt the stratification order, we thus call P a *root pattern*; ii) Q is immediately below P (e.g. there is no R such that R is hookable on P and Q is hookable on R). In our example (Fig. 4.1), the root pattern is P_1, and only patterns P_2 and P_3 are candidates to be hooked on P_1.

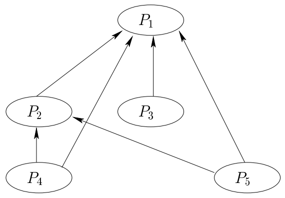

Figure 3. Stratification of $\mathcal{F}^1$

Closed hooking. Let P be a root pattern, and let $\mathcal{Q}_P = \{Q_1, ..Q_n\}$ the candidate patterns to be hooked on pattern P. In order to save redundant computations, we extract all closed subsets of $\mathcal{Q}_P$ which can be simultaneously hooked on P. As in section 4.1, this step is achieved using a propositional reformulation. To pattern P is associated the transaction matrix M_P; to each occurrence u of P in the data is associated a transaction; the items in transaction u are the patterns Q_j such that Q_j admits an occurrence v and u is an ancestor of v. As in section 4.1, a propositional (vertical) Frequent Item Set algorithm can be used to construct all closed subsets of $\mathcal{Q}_P$ which can be simultaneously hooked on P. To each such closed subset $T = \{Q_{i_1}, ..Q_{i_K}\}$ is associated a novel closed pattern, obtained by hooking every Q_{i_j} on P. Due to space limitations, the interested reader is referred to [11] for more detail. In our example, the patterns $\{P_2, P_3\}$ are hooked together on P_1.

Pruning redundant leaves. From our example hooking $\{P_2, P_3\}$ on P_1 results in the pattern of figure 4.

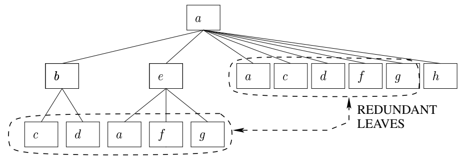

Figure 4. Hooking $\{P_2, P_3\}$ on P_1

The leaves of the root with label a, c, d, f, g represent descendant relations between nodes of label a and nodes of label a, c, d, f, g. This information is actually redundant with the existance of the leaves of depth 2 having the same labels. Hence the leaves of the root are redundant and must be pruned.

To detect the closed frequent combinations of redundant leaves, once again DRYADE uses a propositional closed Frequent Item Set algorithm. The input is a matrix M having as many lines there are occurrences of the pattern P whose redundant leaves we want to discover, and as many columns as potentially redundant leaves. The boolean sign in the cell corresponding to the occurrence o and the leaf lv indicate that the leaf lv is redundant for this occurrence. The closed frequent itemsets of M are the redundant leaves that must be pruned.

In our example after pruning the redundant leaves we obtain the pattern P_6 of figure 5 a). The next iteration of DRYADE will produce the final pattern P_7 shown in figure 5 b).

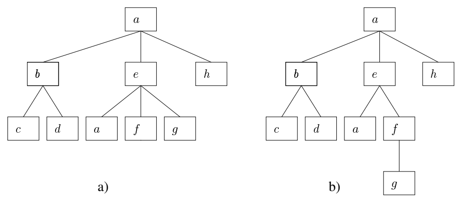

Figure 5. a) P_6, occurrences $\{1, 12\}$ b) P_7, occurrences $\{1, 12\}$

5 Experiments

This section briefly reports on the experimental validation of DRYADE, considering artificial and real-world datasets.

Experimental setting. A stochastic problem generator was implemented to test the scalabitily of DRYADE. More details on this generator can be found in [11]. For each order parameter values, 1,000 problems are independently generated. The frequency threshold ε is 15.

Comparison with WARMR. To our best knowledge, the only frequent pattern algorithm that accommodates a fully relational inclusion definition is WARMR [7]. With courtesy of L. Dehaspe, we could experiment WARMR on the artificial datasets. As could have been expected, these experiments show that WARMR is limited with respect to the size and number of labels of the trees. Comparatively, DRYADE scales up well, with computational cost lower by several orders of magnitude. The performance gain is explained from two specificities of DRYADE. First of all, DRYADE exploits the specific tree-structure of the data, while WARMR aims at the general extraction of relational patterns. Secondly, the closure restriction severely reduces the computational and memory resources needed.

Real world data. A corpus of XML documents from the AFP press agency, kindly given by the Xyleme company, was considered. This corpus includes 3396 documents, with depth in $[2, 3]$, and branching factor in $[4, 5]$, involving 32 different labels. Most of these documents share a complex structure (28 parent/child edges, average depth 2.15). Finding this common structure took 19.3 hours (Athlon 1 GHz, 1 Gb memory). The analysis of the computational time, omitted due to lack of space, shows that over 80% of the effort was spent to computing the closed frequent patterns of depth 1. It must be noted that DRYADE was implemented on the top of the propositional ECLAT algorithm [3]. Indeed, the cost of this first step could be trimmed up to an order of magnitude, using instead the closed CHARM algorithm [14]. This way, DRYADE will hopefully make it feasible to mine a few thousands of real-world XML documents and find worthy patterns in a couple of hours.

6 Conclusion and perspectives

A new algorithm for discovering closed frequent trees in a datatree was presented in this paper. The main motivation for this DRYADE algorithm is to face with highly heterogeneous data, involving many variations in their structures. The challenge was therefore to tackle efficiently a very general, fully relational definition of tree inclusion. This challenge was addressed using two different strategies. The first one is to restrict the search to closed solutions, as in [10, 14, 6]. The second one is based on the reformulation of several search operations in a propositional language. This way, DRYADE can benefit from any progress made in the rapidly evolving field of propositional Frequent Itemt Set algorithms.

This work opens several perspectives for further research. First of all, along the phase transition paradigm [4], ongoing studies are performed to determine the critical values for the order parameters, especially the number of labels. Another promising perspective is to extend the same approach to other types of structured data, e.g. DAGs and graphs, along the same lines as [8].

Acknowledgments

We acknowledge Luc Dehaspe and Chris Borgelt, who kindly made available their WARMR and ECLAT algorithms to us. Thanks also go to Jérôme Azé, Jérôme Maloberti and Mary Felkin, for their help on this work.

References

[1] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *SDM2002, Arlington*, 2002.

[2] T. Asai, H. Arimura, T. Uno, and S. ichi Nakano. Discovering frequent substructures in large unordered trees. In *Discovery Sciences'03, Sapporo*, 2003.

[3] C. Borgelt. Efficient implementations of apriori and eclat. In *FIMI 2003, Melbourne, FL, USA*, 2003.

[4] M. Botta, A. Giordana, L. Saitta, and M. Sebag. Relational learning as search in a critical region. *Journal of Machine Learning Research*, 4:431–463, 2003.

[5] Y. Chi, Y. Yang, and R. R. Muntz. Mining frequent rooted trees and free trees using canonical forms. Technical report, UCLA, 2004.

[6] Y. Chi, Y. Yang, Y. Xia, and R. R. Muntz. Cmtreeminer: Mining both closed and maximal frequent subtrees. In *PAKDD'04, Sidney*, 2004.

[7] L. Dehaspe. *Frequent pattern discovery in first-order logic*. Phd, K.U.Leuven, Leuven, Belgium, dec 1998.

[8] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD'00, Lyon*, 2000.

[9] P. Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, University of Helsinki, Novembre 1992. TR A-1992-6.

[10] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT '99, Jerusalem, Israel*.

[11] A. Termier. Phd. Technical Report 1388, LRI, May 2004. http://www.lri.fr/∼termier/publis/phdTermierEN.ps.gz.

[12] A. Termier, M.-C. Rousset, and M. Sebag. Treefinder: a first step towards xml data mining. In *ICDM'02, Maebashi*.

[13] M. J. Zaki. Efficiently mining frequent trees in a forest. In *8th ACM SIGKDD*, 2002.

[14] M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proc. 2nd SIAM ICDM, Arlington*, Avril 2002.