Mining XML Data with Frequent Trees

Alexandre Termier, Marie-Christine Rousset, Michèle Sebag
{termier, mcr, sebag}@lri.fr

LRI - CNRS UMR 8623, Université Paris-Sud, 91405 Orsay

Abstract. The paper aims at extracting common tree structures in heterogeneous XML data. This approach extends Frequent Item Set techniques to tree structure representation. We provide an algorithm which extracts all subtrees embedded in the data above a given frequency threshold. As a result, this technique provides both a clustering of the data and a symbolic characterization of each cluster. This characterisation is made of the maximally specific subtrees embedded in all the data trees of each cluster. The scalability of the algorithm is investigated on artificial medium-sized data.

1 Introduction

The rapid growth of structured documents (e.g., XML pages) available online raises the need for building integration systems that provide a uniform access to multiple and heterogeneous sources of tree-structured data. A data integration system enables users to pose queries through a *mediated schema*, thus freeing them from having to interrogate each source separately, and to deal with the heterogeneity of their schema. For example, Xyleme [Xyl,Xyl01,ACV$^+$00] is a huge warehouse integrating XML data of the Web. In Xyleme, the mediated schema is a set of labelled trees (called *abstract trees*). Each abstract tree is related to a given domain (e.g., culture, tourism), and is an abstract merger of the *concrete trees* modeling the tree-structure of the actual XML documents relative to that domain. The abstract trees are the support of a visual query interface tool, based on forms, intended to be used by end-users. Today, the abstract trees in the mediated schema in Xyleme are built manually. To scale up to the Web, the challenge is to build them as automatically as possible.

In this paper, we present a method that automatically extracts from a collection of labelled trees a set of *frequent trees* occurring as common (exact or approximate) subtrees embedded in a sufficient number of trees of the collection. By construction, this method provides (i) a clustering of the input trees, and (ii) a characterization of each cluster by a set of frequent trees. The important point is that we are not looking for exact and ordered subtrees but for subtrees that may be approximately embedded in several input trees. An *approximate tree inclusion* preserves the ancestor relation but not necessarily the parent relation. This point distinguishes our work from existing work on DTD inference [PV00]. This choice is motivated by the need for robustness regarding the possible variations in the label nesting of XML documents which we still want to be recognized as having a similar tree structure.

The problem of determining frequent trees in a collection of input trees generalizes the problem of finding *frequent item sets* in a collection of flat items. This problem has been extensively studied because it is a core issue in many data mining applications (e.g., customized marketing, store layout, customer segmentation). The databases involved in these applications are very large and an intensive work has been done for designing fast algorithms for discovering frequent item sets [AS94,Toi96,HGN00]. Those algorithms handle boolean data. In contrast, we have to deal with structured data (i.e., labelled trees).

The paper is organized as follows. In Section 2, we start with a motivating example. In section 3, we describe the formal background of our approach. In Section 4, we describe our two-step method for discovering frequent trees, which has been implemented in the $TreeFinder$ system. In Section 5, we report preliminary experimental results on medium-size artificial data. Finally, in Section 6 we compare our approach with related work and we draw some conclusions and perspectives.

2 Motivating example

Figure 1 is an illustration of heterogeneous XML data structures with various nesting of the labels. For example, the nesting of the *Title* node differs in trees D_3 and D_4. Similarly, the node labelled with *Model* is the direct son of the node labelled with *UsedCar* in D_1, while there is a node in between the corresponding nodes in D_2.

Despite the variation in the structure, we want to group together:

- D_1 and D_2, because the same tree P_1 (Figure 2 (a)) is (exactly or approximately) included in D_1 and D_2,

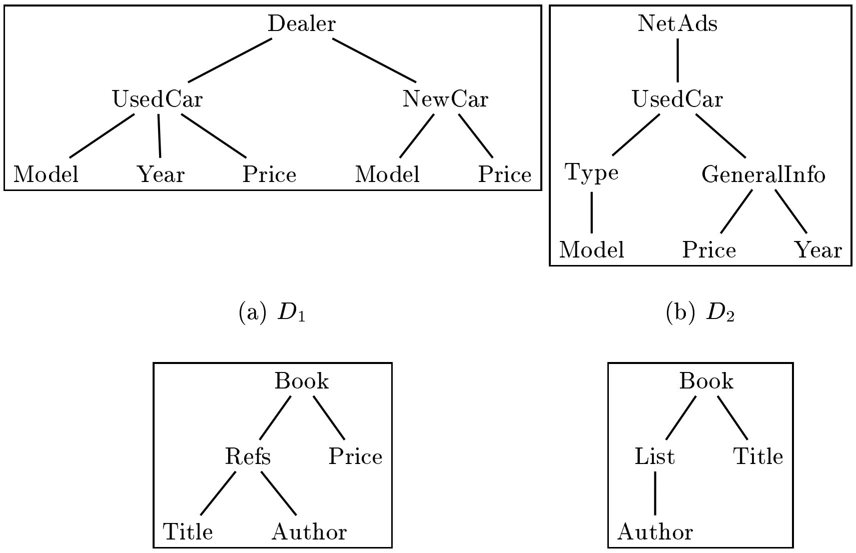
- D_3 and D_4, because they have in common the tree P_2 (Figure 2 (b)), even if the embedding in D_3 and D_4 is approximate.

The two trees P_1 and P_2 of Figure 2 are the 2-frequent trees corresponding to the input trees of Figure 1: they are common to at least 2 trees in the input collection.

3 Formal background

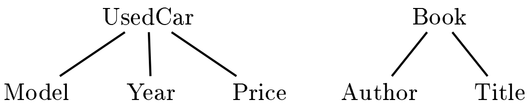
In this section, we introduce an elementary tree model for XML data structure. We further introduce the formal definitions of approximate tree inclusion, (maximal) common subtrees and frequent subtrees.

Modeling XML Data Structure. We model an XML data structure (e.g., a DTD) as a labelled tree. We ignore ID-references and hyperlinks. Each node d has a label, written as $label(d)$. We do not distinguish between elements and attributes. They are all mapped to nodes using the element or the attribute name as label. We do not consider semantic heterogeneity of attribute and element names. We assume that all element or attribute names modeling the same real world concept also carry the same label. This can be achieved by a preprocessing



(a) D_1 (b) D_2 (c) D_3 (d) D_4

Fig. 1. Various tree-structures



(a) P_1 (b) P_2

Fig. 2. 2-Frequent trees

function mapping element names that are synonymous to a common label. We also ignore the possible problems of polysemy (a same label denoting different concepts).

We consider that we are given a collection of labelled trees that constitute the input set of trees to which our method applies for discovering frequent trees.

Approximate unordered tree inclusion. We have to define the tree inclusion that we consider precisely, which further determines the definitions of (maximal) common subtrees and frequent trees. As pointed out in introduction, we are interested in finding subtrees that are included in a sufficient number of input trees (according to a given threshold) even if they are (i) approximately embedded, (ii) the embedding is not ordered.

The *exact ordered tree embedding* problem has been extensively studied (e.g., [RR92]) and is solvable in polynomial time. The *approximate unordered tree embedding* problem has been introduced in [Kil92] where it is shown as beeing NP-complete. Its definition is the following. It is based on the existence of a mapping function (called an *embedding*) preserving labels and ancestorship.

Definition 1. *Let t and t' be two labelled trees. We say that t is* ***included*** *in t', denoted by $t \triangleright t'$ if there exists an injective mapping function f from the nodes of t into the set of nodes of t' such that for all nodes u and v of t :*

- $label(u) = label(f(u))$
- *u is the ancestor of v in t iff $f(u)$ is an ancestor of $f(v)$ in t'*

For example, the tree P_1 of Figure 2 is included in the tree D_2 of Figure 1 (and also in D_1).

Common subtrees and frequent trees. The following definition formally defines the notion of maximal subtree common to several trees.

Definition 2. *Let $t, t_1, ..., t_n$ be labelled trees. We say that t is a* ***maximal common subtree*** *of $t_1, ..., t_n$ iff :*

- $\forall i \in [1..n] \;\; t \triangleright t_i$
- *t is maximal for the previous property, i.e. if there is a labelled tree t' such as $t \triangleright t'$ and $\forall i \in [1..n] \;\; t' \triangleright t_i$ then $t' \triangleright t$*

For example, the tree P_1 of Figure 2 is a maximal subtree common to the two trees D_1 and D_2 of Figure 1.

We can now define the notion of *frequent tree.*

Definition 3. *Let t be a labelled tree. Let F be a set of labelled trees. Let k be an integer: t is a k-**frequent tree** of F iff there exists $\{t_1, ..., t_l\} \in F$ such that $l \geq k$ and t is a maximal common subtree of $\{t_1, ..., t_l\}$.*

Let t be a k-frequent tree of F. We call **support set** of t in F a maximal set (according to set inclusion) of trees $\{t_1, ..., t_m\}$ such that t is a maximal common tree of $\{t_1, ..., t_m\}$.

4 Overview of the TreeFinder system

Our method for discovering frequent trees in a set of input trees is a two-step algorithm.

Step1: clustering step. The first step splits the input trees into sets of flat items, where each item denotes the presence of a label l_2 in the scope of a label l_1 within a tree, denoted $l_1..l_2$ and called a *label pair*, for every label l_1 and l_2 in the label vocabulary. More formally, a label pair $l_1..l_2$ denotes the existence of a node with label l_1 within the ancestors of a node with label l_2. Such an information (e.g., $UsedCar..Year$) is considered as semantically significant if it occurs frequently in the input data. This splitting of the trees in label pairs breaks the tree structure but makes possible the use of a standard frequent item sets algorithm for discovering frequent co-occurrence of label pairs in the input trees. The result of this step is a set of clusters grouping together trees which all contain a given set of label pairs. For example, we may obtain a cluster supported by the two items $UsedCar..Year$ and $UsedCar..Model$. Many algorithms have been developped to compute frequent item sets. So far, our implementation uses the simplest one: Apriori [AS94].

Step2: Computation of maximal subtrees included in all the trees of each cluster. This step is based on an appropriate relational encoding of the labelled trees of each cluster, and the use of a standard algorithm that computes the *least general generalization* [Plo70] of the resulting set of relational conjunctive formulas.

The relational representation that we consider for a labelled tree T consists in a conjunction of atoms $Rel(T)$ encoding the parent relation between the nodes of T: for each node u parent of v in T, we create the atom $p(X_u, X_v, l_u, l_v)$, where X_u and X_v are variables and l_u and l_v are constants such that $l_u = label(u)$ and $l_v = label(v)$. Conversely, to each relational formula R made of a conjunction of atoms of the form $p(X, X', l, l')$ we can associate a unique labelled graph $Rel^{-1}(R)$, whose nodes are the set of variables appearing in R.

Checking the approximate unordered tree inclusion (Definition 1) between t and t' corresponds to checking the θ-subsumption [Plo70,MS01] of $Rel^*(t)$ by $Rel^*(t')$, where $Rel^*(t)$ is obtained from $Rel(t)$ by replacing the relation p by its transitive closure p^*. In other words, $Rel^*(t)$ is a relational encoding of the ancestor relationship between the nodes in t. Figure 3 illustrates the two encoding functions Rel and Rel^* for two labelled trees.

Let $t_1, ...t_n$ be the set of trees of a cluster. We compute the least general generalizer[1] of the relational formulas encoding the ancestor relationship of t_1, ..., t_n: $lgg(Rel^*(t_1), ..., Rel^*(t_n))$. In the example of Figure 3 :
$lgg(Rel^*(T_1), Rel^*(T_2)) =$

[1] For any set $\{c_1, ..., c_n\}$ of conjunctions of atoms without function symbols, there exists a unique least common generalizer (up to variables renaming), i.e., a unique conjunction of atom c such that c θ-subsumes every c_i and c is minimal for that property (every c' which θ-subsumes every c_i θ-subsumes c).

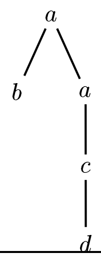
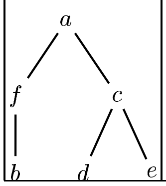
T_i	$Rel(T_i)$	$Rel^*(T_i)$
	$p(N_1, N_2, a, b)$ $p(N_1, N_3, a, a)$ $p(N_3, N_4, a, c)$ $p(N_4, N_5, c, d)$	$p^*(N_1, N_2, a, b)$ $p^*(N_1, N_3, a, a)$ $p^*(N_1, N_4, a, c)$ $p^*(N_1, N_5, a, d)$ $p^*(N_3, N_4, a, c)$ $p^*(N_3, N_5, a, d)$ $p^*(N_4, N_5, c, d)$
	$p(N'_1, N'_2, a, f)$ $p(N'_1, N'_3, a, c)$ $p(N'_2, N'_4, f, b)$ $p(N'_3, N'_5, c, d)$ $p(N'_3, N'_6, c, e)$	$p^*(N'_1, N'_2, a, f)$ $p^*(N'_1, N'_3, a, c)$ $p^*(N'_1, N'_4, a, b)$ $p^*(N'_1, N'_5, a, d)$ $p^*(N'_1, N'_5, a, e)$ $p^*(N'_2, N'_4, f, b)$ $p^*(N'_3, N'_5, c, d)$ $p^*(N'_3, N'_6, c, e)$

Fig. 3. Two example trees T_1 and T_2, and their relational encoding

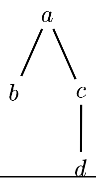
lgg	$star^{-1}$
$p^*(V_1, V_2, a, b)$ $p^*(V_1, V_3, a, c)$ $p^*(V_1, V_4, a, d)$ $p^*(V_3, V_4, c, d)$	$p(V_1, V_2, a, b)$ $p(V_1, V_3, a, c)$ $p(V_3, V_4, c, d)$
$Rel^{-1}(star^{-1}(lgg(Rel^*(T_1), Rel^*(T_2))))$	
	

Fig. 4. Relational lgg and its tree decoding

$p^*(V_1, V_2, a, b) \wedge p^*(V_1, V_3, a, c) \wedge p^*(V_1, V_4, a, d) \wedge p^*(V_3, V_4, c, d)$.

The subtle point is that from $lgg(Rel^*(t_1), ..., Rel^*(t_n))$, which is a conjunction of atoms of the form $p^*(X_u, X_v, l_u, l_v)$ that can be considered as defining the relation p^*, we can reconstruct the atoms defining the underlying relation p whose p^* is the transitive closure. This results from the fact that, because of the tree structure of the inputs t_1, ..., t_n, the implicit relation p whose transitive closure leads to $lgg(Rel^*(t_1), ..., Rel^*(t_n))$ has necessarily a forest structure, and the corresponding explicit p^* has a dag structure. It is the application of a traversal of that dag structure in a topological order that makes possible to reconstruct p from p^* in $lgg(Rel^*(t_1), ..., Rel^*(t_n))$. Let $star^{-1}(lgg(Rel^*(t_1), ..., Rel^*(t_n)))$ be the resulting conjunction of atoms of the form $p(X, X', l, l')$. In the example of Figure 3 :
$star^{-1}(lgg(Rel^*(T_1), Rel^*(T_2))) = p(V_1, V_2, a, b) \wedge p(V_1, V_3, a, c) \wedge p(V_3, V_4, c, d)$.

Finally, it can be proved that $Rel^{-1}(star^{-1}(lgg(Rel^*(t_1), ..., Rel^*(t_n))))$ is a forest made of all the maximal trees that are included in t_1, ..., t_n.

Figure 4 illustrates the relational least general generalizer corresponding to our example and its tree decoding.

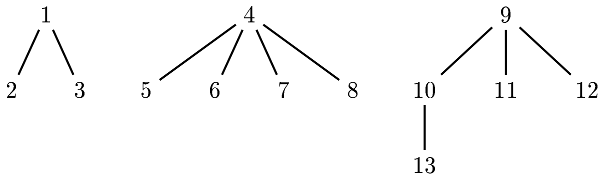
5 Experimental results

In this section, we report the first results of an experimental protocol that we have set up for analysing the execution time of our system, in order to analyse its scalability with respect to its input size.

Experimental protocol. We consider artificial inputs that are randomly generated from a fixed set F of target frequent trees and a fixed set L of labels. In our first experiment, the set of labels is made of the first hundred integers, and the target frequent trees are given in Figure 5. The varying parameters are the number and the size of the generated trees that will serve as input. The size of those trees is constrained by a maximal branching factor B_{max} (5 in our experiment), and a maximal depth D_{max} (3 in our experiment). The tree generation is a recursive process [2]. For each tree, it is decided with a probability p (set to 0,2 in our experiment) whether it is obtained by perturbating a target tree randomly chosen in F, or whether it is created as a new tree. In this case, its root is labelled with a label randomly chosen in L, the number of its subtrees is randomly chosen between 0 and B_{max}, and each subtree is created by iteratively applying the generation process. The perturbation of a target tree t consists in selecting a random number of nodes in $[|t|..|t|^2]$ to add to t, and in incrementally adding those nodes either (with a probability p_e, fixed to 0.5 in our experiment) between two successive nodes of t, or as a new son of an existing node of t.

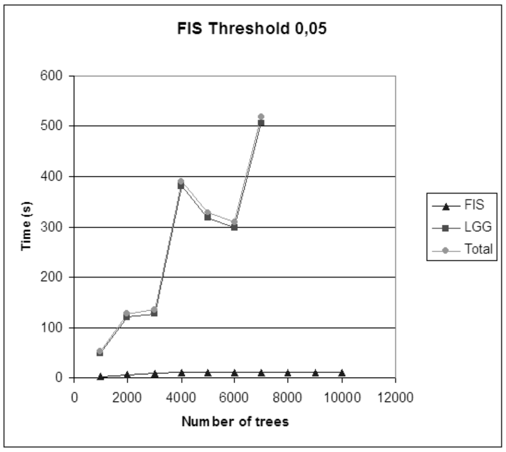
Experimental results. Our first results are presented in Figure 6, with a frequent item set algorithm threshold set to 0,05. In Figure 6 a), we can observe that *in practice*, the time complexity is linear with respect to the number of

[2] The generator implementation is available at `http://www.lri.fr/~termier/generator.tgz`

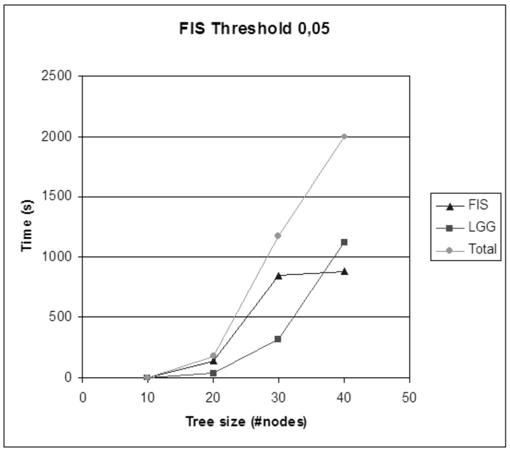


(a) P_1 (b) P_2 (c) P_3

Fig. 5. Target frequent trees



a) Varying tree number, random tree size



b) Varying tree size, 1000 trees

Fig. 6. Test results

input trees. Note that the least general generalisation (LGG) times for 8000, 9000 and 10000 trees are not displayed due to memory saturation. Figure 6 b) is a test of worst case complexity, where the number of trees is fixed to 1000, and the probability that a node of a random tree becomes the root of a target frequent tree is set to 1. Hence all random trees are perturbated target frequent trees. With a small change in the experimental protocol, we can control exactly the number of nodes added to a target frequent tree, and the results are obtained by varying this number. This case represents hard instances because the number of items handled by the Apriori algorithm is large, and each produced frequent item set is associated to a big cluster of documents. Thus the least general generalisation of each cluster is computed on many relational formulas, further degrading performances.

6 Conclusion

To our knowledge, this paper is the first work on the problem of discovering frequent trees among a set of labelled trees. It provides a first step towards mining tree-structured data, and in particular XML data. The preliminary experimental results that we have obtained are encouraging and indicate several directions of investigation for improving the performance of our *TreeFinder* system and its scalability.

Several existing works are related to the problems or the techniques we have presented in this paper.

The work on DTD inference (e.g., [PV00]) studies the automatic inference of a unique DTD from a set of XML data. It is also based on modeling XML data structures as labelled trees, and DTDs as tree grammars. Viewed as a problem of infering trees from input trees, the trees expected as output must satisfy an exact and ordered tree embedding. In contrast, to face with irregularities in the structure, we deal with approximate and unordered tree inclusion.

Several works related to querying XML data are based on approximate matching of tree queries against data trees (e.g.,[SN00,DR01,AYCS02]). Among them, the work by [SN00] is explicitly based on the same approximate tree embedding as in this paper for defining approximate tree matching. The others are based on relaxation operators that guarantee that the set of answers that will be returned by a relaxed query is a superset of the set of answers of the initial query. Some of those operators correspond to the approximate tree embedding considered in this paper (e.g., deleting a node), some others do not (e.g., node unfolding like in [DR01]).

WARMR[DT01] is a powerful tool based on inductive logic programming for discovering frequent relational patterns from relational databases. An appropriate relational encoding of trees (in the spirit of the one we consider in the second step of our method) would make possible to use a tool like WARMR to discover frequent trees from tree databases. The main problem of such a general tool is its scalability. In the near future, we intend to compare our two-step approach implemented in our *TreeFinder* system with an approach based on a relational encoding of labelled trees and the use of Warmr. The main difficulty is that the exact encoding of trees into relational formulas requires inequality statements between the variables encoding the nodes. Dealing with conjunctive relational formulas with inequalities is an additional source of complexity in inductive logic programming.

Acknowledgements : The authors wish to thank Jérôme Azé for the Apriori implementation and Jérôme Maloberti for the least general generalisation implementation based on his work on θ-subsumption ([MS01]).

References

[ACV+00] V. Aguiléra, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez. Querying the XML Documents on the Web. In *Proc. of the ACMSIGIR Workshop on XML and I. R., Athens*, July 28, 2000.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[AYCS02] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *Proc. of the EDBT 2002 Conference*, 2002.

[DR01] C. Delobel and M-C. Rousset. A uniform approach for querying large tree-structured data through a mediated schema. In *Proc. of the 2001 Int. Workshop on Foundations of Models for Information Integration*, September 2001.

[DT01] L. Dehaspe and H. TT Toivonen. *Relational Data Mining*, chapter Discovery of Relational Association Rules, pages 189–212. Springer-Verlag, 2001.

[HGN00] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining – a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.

[Kil92] P. Kilpeläinen. *Tree Matching Problemas with Applications to Structured Text Databases*. PhD thesis, University of Helsinki, 1992.

[MS01] J. Maloberti and M. Sebag. Theta-subsumption in a constraint satisfaction perspective. In *Proc. of the ILP'01 Conference*, pages 164–178, 2001.

[Plo70] G. Plotkin. A note on inductive generalisation. *Machine Intelligence*, 5:153–163, 1970.

[PV00] Yannis Papakonstantinou and Victor Vianu. DTD inference for views of XML data. In *Symposium on Principles of Database Systems*, pages 35–46, 2000.

[RR92] R. Ramesh and L.V. Ramakrishnan. Nonlinear pattern matching in trees. *Journal of the ACM*, 39(2):295–316, April 1992.

[SN00] T. Schlieder and F. Naumann. Approximate tree embedding for querying XML data. In *ACM Sigir Workshop on Information Retrieval*, July 2000.

[Toi96] Hannu Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *In Proc. 1996 Int. Conf. Very Large Data Bases*, pages 134–145. Morgan Kaufman, 09 1996.

[Xyl] Xyleme.
`http://www.xyleme.com`.

[Xyl01] Lucie Xyleme. A dynamic warehouse for XML data of the web. *IEEE Data Engineering Bulletin*, 2001.