TreeFinder: a First Step towards XML Data Mining

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

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

Abstract. In this paper, we consider the problem for searching *frequent trees* from a collection of tree-structured data modeling XML data. The $TreeFinder$ algorithm aims at finding trees, such that their exact or perturbed copies are frequent in a collection of labelled trees. To cope with the complexity issue, $TreeFinder$ combines a pre-processing step extracting candidate subsets of data with a propositional Frequent Item Set algorithm, on one hand, and a generalization algorithm borrowed to Inductive Logic Programming on the other hand. This way, the additional complexity due to the tree-structured formalism only occurs in the final step, during the generation of the tree solutions.
As a counterpart, $TreeFinder$ is correct but not complete: it finds a subset of the actually frequent trees. The default of completeness is experimentally investigated on artificial medium size datasets; it is shown that $TreeFinder$ reaches completeness or falls short to it for a range of experimental settings.

1 Introduction

Discovery of frequent patterns in large data collections has been investigated in a variety of settings. The simplest setting, which has been extensively studied, comes from the market basket analysis. It consists in discovering frequent item sets from databases storing customers transactions. 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. This work has been extended to the search of frequent sequential patterns from series of transactions [AS95,SA96,MHV97], and of frequent relational queries from relational databases [DT01].

In this paper, we consider the problem for searching *frequent trees* from a collection of tree-structured data modeling XML data. We present a method that automatically extracts from a collection of labelled trees a set of frequent trees occurring as common (exact or approximate) trees 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 an exact embedding but for trees 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 main motivating application of this work is the construction of a tree-based *mediated schema* for integrating multiple and heterogeneous sources of XML 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.

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 in Section 7.

2 Motivating example

Fig. 1 is an illustration of heterogeneous XML data structures with various nesting of the labels.

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, D_2 and D'_2, because the same tree P_1 (Fig. 2 (a)) is (exactly or approximately) included in D_1, D_2 and D'_2,

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

The two trees P_1 and P_2 of Fig. 2 are the frequent trees corresponding to the input trees of Fig. 1: they are common to several different trees in the input collection.

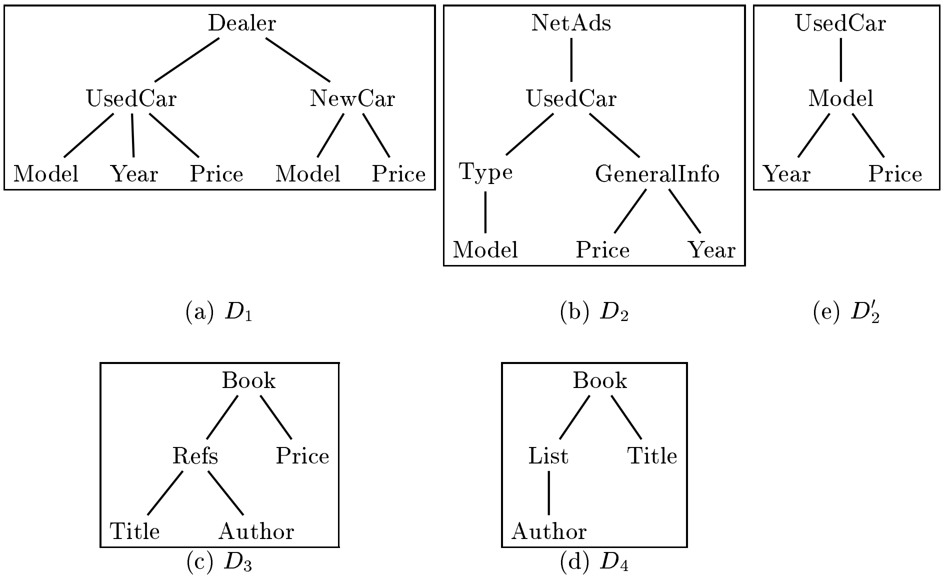


Fig. 1. Various tree-structures

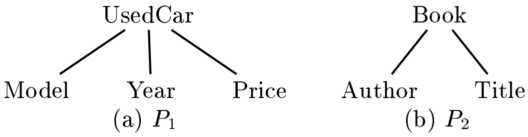


Fig. 2. Frequent trees

3 Formal background

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

3.1 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 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).

Definition 1 (Labelled trees). *A* **labelled tree** *is a pair $\langle t, label\rangle$ where (i) t is a finite* **tree** *whose nodes are in $\mathcal{N}$, (ii) label is a* **labeling function** *that assigns a label to each node in t.*

The labels play the role of the begin-end tag in XML data models. We use the current terminology about trees: root, children, descendant, leaf, subtree, etc. We also use some functional notations such as: $root(t)$ returns the root node of the tree t, $children(u)$ (resp. $desc(u)$, $anc(u)$) returns the set of nodes that are children (resp. descendants, ancestors) of node u, and $label(t)$ returns the set of labels of the nodes of the tree t.

Note that within a given tree, two different nodes may have the same label.

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.

3.2 Tree inclusion

We consider three variations of the notion of tree inclusion. The most restricted definition is that of *subtree inclusion.*

Definition 2 (Subtree inclusion).
Let t and t' be two labelled trees. We say that t is ***included as a subtree*** *in t' iff there exists a subtree of t' which is identical with t.*

The subtree inclusion problem has been extensively studied (e.g., [RR92]) and is solvable in linear time. For instance, tree P_1 (Fig. 2 (a)) is included as a subtree in tree D_1 (Fig. 1 (a)), but it is not in tree D_2 (Fig. 1 (b)).

However, as pointed out in the introduction, our aim is to find frequent tree patterns in a set of trees. The above definition is too restricted since one would typically like tree P_1 to be considered as present in D_2.

Therefore a relaxed inclusion definition, first proposed by [Kil92], based on an injective mapping termed *tree embedding* and preserving the labels and ancestor relation in the trees, is considered.

Definition 3 (Inclusion by tree embedding).
Let t and t' be two labelled trees. We say that t is ***included by embedding*** *in t' if there exists a mapping f from the nodes of t into the set of nodes of t' such that f is injective and f strictly preserves the ancestor relation:*

$$\forall\ u \ in\ t,\ label(u) = label(f(u))\ and\ \ \forall\ u, v\ in\ t\ anc(u, v) \Longleftrightarrow anc(f(u), f(v))$$

Example. According to that definition, tree P_1 (Fig. 2) is included in D_2 and D_1 (Fig. 1), but is not included in D'_2 (Fig. 1): there exists no ancestor relation between nodes *Model* and *Year* or *Price* in P_1, while there exists one in D'_2.

A weaker order relation termed *tree subsumption*, is then defined on trees by relaxing the inclusion definition above.

Definition 4 (Inclusion by tree subsumption).
Let t and t' be two labelled trees. We say that t is ***included by tree subsumption*** *in t' if there exists a mapping f from the nodes of t into the set of nodes of t' such that f preserves the ancestor relation:*

$$\forall\ u\ in\ t,\ label(u) = label(f(u))\ and\ \ \forall\ u,v\ in\ t, anc(u,v) \Longrightarrow anc(f(u), f(v))$$

The advantage of the above definition is the following: if we choose to represent labelled trees as relational formulas ([Llo87]), then tree subsumption is equivalent to the θ-subsumption relation defined by ([Plo70]). Let us first describe how labelled trees can be put in relational form.

3.3 Relational representation of labelled trees.

The relational representation that we consider for a labelled tree is a conjunction of atoms noted $Rel(t)$, describing the parent relation over the set of nodes of t. We also introduce another relational representation noted $Rel^+(t)$, describing the transitive closure of the parent relation, i.e. the ancestor relation.

Definition 5 (Relational description of labelled trees). *Let t be a labelled tree. $Rel(t)$ is the conjunction of all atoms $ab(u,v)$, such that u and v are nodes in t, with $label(u) = a$, $label(v) = b$ and u is the parent node of v.*
$Rel^+(t)$ is the conjunction of atoms $a{*}b(u,v)$, such that u and v are nodes in t, with $label(u) = a$, $label(v) = b$ and u is an ancestor node of v.

Fig. 3 illustrates the two encoding functions Rel and Rel^+ for two labelled trees.

Let us recall the definition of the θ-subsumption.

Definition 6 (θ-subsumption). *Let $\mathcal{C}$ and $\mathcal{C}'$ be two first order logic formulas. We say that $\mathcal{C}$ θ-subsumes $\mathcal{C}'$ if there exists a mapping θ from the variables in $\mathcal{C}$ onto the variables and constants in $\mathcal{C}'$ such that every atom in $\mathcal{C}\theta$ appears in $\mathcal{C}'$.*

The following proposition states the equivalence between inclusion by tree subsumption (Definition 4) and θ-subsumption.

Proposition 1. *Let t and t' be two labelled trees. Then:*
t is included by tree subsumption in t' iff $Rel^*(t)$ θ-subsumes $Rel^*(t')$

The advantage of this relational representation is that, though θ-subsumption test is NP complete, efficient implementations have been proposed [MS01].

3.4 Common trees and frequent trees.

The following definition formally defines the notion of maximal (according to tree inclusion) tree common to several trees. It relies on, and thus depends on, the definition of tree inclusion that is considered.

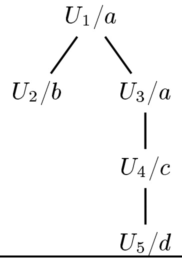
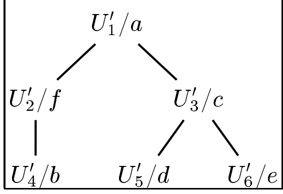
T_i	$Rel(T_i)$	$Rel^+(T_i)$
	$ab(U_1, U_2)$ $aa(U_1, U_3)$ $ac(U_3, U_4)$ $cd(U_4, U_5)$	$a^*b(U_1, U_2)$ $a^*a(U_1, U_3)$ $a^*c(U_1, U_4)$ $a^*d(U_1, U_5)$ $a^*c(U_3, U_4)$ $a^*d(U_3, U_5)$ $c^*d(U_4, U_5)$
	$af(U'_1, U'_2)$ $ac(U'_1, U'_3)$ $fb(U'_2, U'_4)$ $cd(U'_3, U'_5)$ $ce(U'_3, U'_6)$	$a^*f(U'_1, U'_2)$ $a^*c(U'_1, U'_3)$ $a^*b(U'_1, U'_4)$ $a^*d(U'_1, U'_5)$ $a^*e(U'_1, U'_5)$ $f^*b(U'_2, U'_4)$ $c^*d(U'_3, U'_5)$ $c^*e(U'_3, U'_6)$

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

Definition 7 (Maximal common tree). *Let $t, t_1, ..., t_n$ be labelled trees. We say that t is a* ***maximal common tree*** *of $t_1, ..., t_n$ iff :*

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

For instance, tree P_1 (Fig. 2) is a maximal common tree of D_1, D_2 and D_2' (Fig. 1) w.r.t. tree subsumption (Definition 4), but not w.r.t. subtree inclusion and inclusion by embedding (Definitions 2 and 3).

Let us now define the notion of *frequent tree.*

Definition 8 (Frequent tree). *Let T be a set of labelled trees, and let t be a labelled tree. Let ε be a real number in $[0, 1]$.*
We say that t is a ***ε-frequent tree*** *of T iff:*

- there exists l subtrees $\{t_1, ..., t_l\}$ in T such that t is maximal common tree of $\{t_1, ..., t_l\}$.

- l is greater or equal to $\varepsilon.|T|$.

The ***support set*** *of t in T is the set of all trees t_i such that t is included in t_i.*

For instance according to Definition 4 (but not to Definitions 3 and 2), the two trees P_1 and P_2 (Fig. 2) are the 0.4-frequent trees w.r.t. the set of the trees D_1, D_2, D_2', D_3 and D_4 (Fig. 1).

Let F_ε denote the set of all ε-frequent trees w.r.t. a set of trees T; due to monotonicity,

$$\varepsilon > \varepsilon' \Longrightarrow F_\varepsilon \subseteq F_{\varepsilon'}$$

Maximal ε-frequent trees are defined as the maximal elements (for tree inclusion) in F_ε.

4 Overview of the TreeFinder system

Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a set of labelled trees, let ε be a frequency threshold. The $TreeFinder$ method for discovering ε-frequent trees in $\mathcal{T}$ is a two-step algorithm. The first step is described in Section 4.1. It is a clustering step that groups input trees in which same pairs of labels occur together frequently enough in the ancestor relation. This is realized by applying a standard algorithm computing frequent item sets (e.g., Apriori [AS94]) to an appropriate abstraction of the input trees. The second step is described in Section 4.2. It is a tree construction step based on the computation of maximal trees that are common to all the trees of each cluster. In Section 4.3, the trees that are returned as output of this two-step algorithm are formally characterized w.r.t. to its input.

4.1 Clustering guided by co-occurence of labels pairs

Input: An abstraction of the input $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, where each t_i is viewed as a transaction made of all the items l*m such that l is the label of an ancestor of a node labelled by m in t_i. Each item l*m has a unique identifier. Let $\mathcal{I}$ be the set of those identifiers.
For example, the input transactions corresponding to trees of Fig. 1 are:
D_1 = { Dealer*UsedCar, Dealer*NewCar, UsedCar*Model, UsedCar*Year, UsedCar*Price, NewCar*Model, NewCar*Price, Dealer*Model, Dealer*Year, Dealer*Price}
D_2 = { NetAds*UsedCar, UsedCar*Type, UsedCar*Model, UsedCar*GeneralInfo, GeneralInfo*Price, GeneralInfo*Year, UsedCar*Price, UsedCar*Year, NetAds*Model, NetAds*Price, NetAds*Year }
D'_2 = { UsedCar*Model, Model*Year, Model*Price, UsedCar*Year, UsedCar*Price }
D_3 = { Book*Refs, Book*Price, Refs*Title, Refs*Author, Book*Title, Book*Author}
D_4 = {Book*List, Book*Title, List*Author, Book*Author }

This splitting of the trees in separate items corresponding to pairs of labels breaks the tree structure but makes possible the use of a standard frequent item sets algorithm for discovering frequent label pairs in the input trees. The co-occurrence of same pairs of labels is considered as semantically significant if it occurs frequently in the input data.

Clustering method: Many algorithms have been developped to compute frequent item sets. So far, our implementation uses the simplest one: *Apriori* [AS94]. We apply the *Apriori* algorithm to the set of transactions $\mathcal{T}$ over the items $\mathcal{I}$ identifying the pairs of labels, with the frequency threshold (a.k.a minimum support) set to ε.

Output: the *support sets* of the *largest frequent item sets* returned by the *Apriori* algorithm. The support set of an item set $s \subseteq \mathcal{I}$, denoted $support(s)$, is the subset of $\mathcal{T}$ made of all the transactions including s. A frequent item set is a subset of $\mathcal{I}$ the support set of which has a size greater than $\varepsilon.|\mathcal{T}|$.

For instance, the largest frequent item sets returned by the *Apriori* algorithm applied with the frequency threshold 0.4 to the set of transactions $\{D_1, D_2, D'_2, D_3, D_4\}$, given previously and abstracting the trees of Fig. 1 are:
s_1 = {UsedCar*Model, UsedCar*Year, UsedCar*Price}
s_2 = {Book*Price, Book*Title, Book*Author}
The corresponding support sets, that are thus returned as output clusters, are:
$support(s_1) = \{D_1, D_2, D'_2\}$
$support(s_2) = \{D_3, D_4\}$.

4.2 Computation of maximal common trees

Input: This step takes as input the output of the previous clustering step, and computes for each cluster the maximal trees that are common to (i.e. included in) *all* the trees of the cluster From now on, the tree inclusion we consider is the subsumption-inclusion (Definition 4).

Method: For each cluster $\{t_{i_1}, t_{i_2}, \ldots, t_{i_n}\}$, we compute the *least general generalization* [Plo70] $LGG(Rel^+(t_1), ..., Rel^+(t_n))$ of the relational formulas encoding the trees.

The *least general generalization* of two relational formulas $Rel(f_1)$ and $Rel(f_2)$ is the most specific formula which θ-subsumes $Rel(f_1)$ and $Rel(f_2)$. From [Plo70] it is shown that the least general generalization of two conjunctive formulas with no function symbol is unique (up to variable renaming).

In the example of Fig. 3 :
$LGG(Rel^+(T_1), Rel^+(T_2)) = a^*b(U_1, U_2) \wedge a^*c(U_1, U_3) \wedge a^*d(U_1, U_4) \wedge c^*d(U_3, U_4)$.

Output: the set of trees resulting from the tree decoding of the least general generalizers of the relational formulas encoding the trees for each cluster.

The subtle point is that from $LGG(Rel^+(t_1), ..., Rel^+(t_n))$, which is a conjunction of atoms of the form $l^*m(U_i, U_j)$, modeling the ancestor relation, we can reconstruct the atoms defining the underlying parent relation. This results from the fact that, because of the tree structure of the inputs t_1, ..., t_n, the implicit parent relation p whose transitive closure leads to $LGG(Rel^+(t_1), ..., Rel^+(t_n))$ has necessarily a forest structure, and the corresponding explicit ancestor relation has a dag structure. It is the application of a traversal of that dag structure in a topological order that makes possible to reconstruct the parent relation from the ancestor relation which is stated 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 $lm(U_i, U_j)$.

In the example of Fig. 3 :
$star^{-1}(LGG(Rel^*(T_1), Rel^*(T_2))) = ab(U_1, U_2) \wedge ac(U_1, U_3) \wedge cd(U_3, U_4)$.

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

LGG	$star^{-1}$
$a^*b(U_1, U_2)$ $a^*c(U_1, U_3)$ $a^*d(U_1, U_4)$ $c^*d(U_3, U_4)$	$ab(U_1, U_2)$ $ac(U_1, U_3)$ $cd(U_3, U_4)$
$Rel^{-1}(star^{-1}(LGG(Rel^+(T_1), Rel^+(T_2))))$	

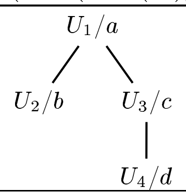


Fig. 4. Relational LGG and its tree decoding

4.3 Formal characterization of the TreeFinder results

The following proposition states that the results produced by $TreeFinder$ are correct. We also provide a sufficient condition for these results to be complete, i.e. to include *all* the maximal frequent trees present in the input trees.

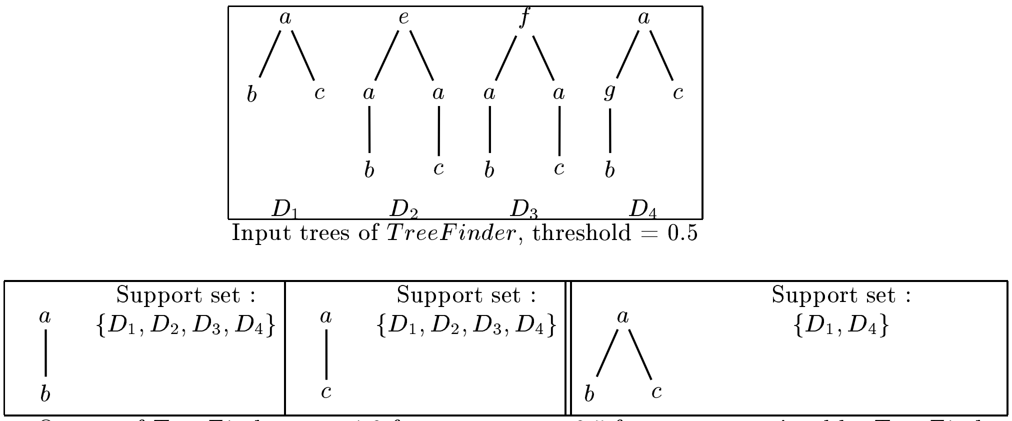
Proposition 2. *Let $\mathcal{T}$ be a set of labelled trees, let ε be a frequency threshold, let $ft_1, ft_2, \ldots, ft_n$ be the trees obtained as output of the two-step $TreeFinder$ method applied to $\mathcal{T}$:*

*- **correctness:** $ft_1, ft_2, \ldots, ft_n$ are ε-frequent trees for $\mathcal{T}$.*

*- **sufficient condition for completeness:** If $\mathcal{T}$ is such that the trees in the support of each maximal ε-frequent trees have no label pair in common with trees out of the support, then $ft_1, ft_2, \ldots, ft_n$ are exactly the maximal ε-frequent trees of $\mathcal{T}$.*

The following example shows that $TreeFinder$ is not guaranteed to find the maximal ε-frequent trees in the general case.

Fig. 5 illustrates a case where $TreeFinder$ applied with a frequency threshold of 0.5 would find a single big cluster made of all the input trees (corresponding to the maximal item set $\{a{*}b,\ a{*}c\}$, which appears in fact in all the transactions). The result returned by $TreeFinder$ would then be the two trees ft_1, ft_2 reduced to one edge each, which are common to all the input trees: they are 1.0-frequent trees and thus a fortiori 0.5-frequent trees. However, $TreeFinder$ does not find the unique 0.5-frequent tree whose root is labelled by a and has two children labelled by b and c respectively.



Input trees of $TreeFinder$, threshold = 0.5

Output of $TreeFinder$: two 1.0-frequent trees   0.5-frequent tree missed by $TreeFinder$

Fig. 5.

This example is illustrative of the fact that $TreeFinder$ can fail to find all the ε-frequent trees in the data. We estimate this loss of completeness by comparison with the ideal algorithm, which we will refer to as *AprioriTree*. *AprioriTree* is an extension of the levelwise algorithm *Apriori* to tree structures:

- Find the trees of size 1 (trees with one edge) that are common to more than $\varepsilon \times$ (number of input trees) input trees.
- $i := 1$
- While there exists frequent trees of size i:
  - **generate** candidate frequent trees of size $i + 1$ from the frequent trees of size i and those of size 1,
  - **test** the inclusion of those candidate frequent trees in the input trees in order to select those candidate frequent trees that are actually frequent.
  - $i := i + 1$

AprioriTree is a naive method which is not tractable in practice because clearly, it hardly scales up: first of all, the test step intensively relies on tree subsumption, which is NP complete. Furthermore, the number of candidate trees constructed from a frequent i-edge tree grows like $i \times e$, where e is the number of frequent edges; in contrast, the number of candidates constructed from a frequent i-item set is $e - i$, where e is the number of frequent items.

We have implemented *TreeFinder* in C++, and its experimental validation is detailed in the next section.

5 Experimental results

This section first details the experiment goals and the experimental setting used for the empirical validation of *TreeFinder*. The results obtained are then reported and discussed with regard to computational cost (section 5.3) and frequent tree quality (section 5.4).

5.1 Experiment goals

A key criterion for data mining algorithms concerns their robustness and scalability w.r.t. the characteristics of input data [AS94]. Therefore, the computational cost of *TreeFinder* w.r.t. the number and size of input trees will be investigated in section 5.3.

Another criterion regards the quality of the results provided by *TreeFinder*. Let us consider as baseline algorithm *AprioriTree*.

Proposition 2 has given sufficient conditions for *TreeFinder* to find exactly the same frequent trees as *AprioriTree*. In the general case however, *TreeFinder* will miss some frequent trees as illustrated in Fig. 5.

The second experiment goal will thus be to estimate the percentage of frequent frequent trees missed by *TreeFinder*, compared to the baseline algorithm. This percentage, termed loss factor, will be investigated experimentally in section 5.4.

All experiments consider artificial medium-size datasets, according to the following experimental setting.

5.2 Experimental setting

As mentioned in the introduction, the mining of XML data raises two major difficulties. The first one, not considered in the present paper, regards the potential synonymy and polysemy of tags. The second difficulty, tackled in the paper, concerns the tree-structure of the XML data.

As a preliminary investigation, we thus consider artificial XML data, generated from known target frequent trees using a randomized tree-structure generator. This generator[1] proceeds as follows:

- A set of target frequent trees $\mathcal{P} = \{P_1, \ldots P_K\}$ is given by the user.
- A set of external labels $\mathcal{L}$ is defined, disjoint (unless otherwise specified) from the labels involved in the target frequent trees. The size L of $\mathcal{L}$ is a user-supplied parameter of the generator.
- Each tree-document is recursively generated. Let u be the top node in the node list (initialized at the root node). Two options are considered: with probability p, a (perturbed) copy of one target frequent tree is inserted at node u (see below the target frequent tree insertion). Otherwise (with probability $1-p$), the number of son nodes for the current node u is uniformly selected in $[0..B]$, where B is the maximum branching factor; the label for the current node u is selected with uniform probability in $\mathcal{L}$; u is removed from the node list, and its son nodes are added to the node list.
- The insertion of a target frequent tree P, uniformly selected in $\mathcal{P}$, proceeds by copying at node u a perturbed copy of P. A number m of additional nodes is randomly selected in $[0, \delta]$, where δ is the perturbation parameter. These nodes are randomly inserted in the target frequent tree, and their labels are randomly selected in $\mathcal{L}$.

The parameters of the artificial tree generator are summarized in Table 1.

K	number of target frequent trees
L	number of external labels
p	probability of including a target frequent tree in each node
δ	maximal number of nodes added to perturbate a target frequent tree
B	maximal branching factor
D	maximal tree depth

Table 1. Parameters of the artificial tree-structure generator

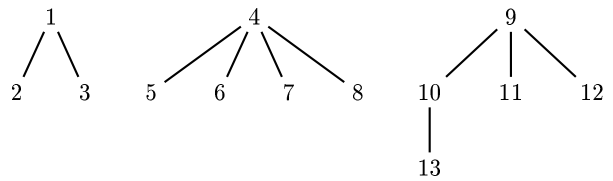
Due to the stochastic generation of the datasets, the reported results are obtained by averaging the results obtained for ten independent runs (launched with same parameter values and distinct random seeds).

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

5.3 Empirical study of computational cost

Various experiments consider up to 10,000 artificial tree-structured documents, generated with the following parameters:

- The set of target frequent trees is displayed in Fig. 6.
- The total number of labels is set to 100. With no loss of generality, labels 1 to 13 denote the labels involved in the target frequent trees, and $\mathcal{L}$ is made of all labels $14 \ldots 100$. In these tests, the labels of the target frequent trees are not disjoint from the external labels, so $L = 100$.
- The probability p of including a target frequent tree at each node is set to 0.2
- The maximal number δ of perturbations is set to 25.
- The maximal branching factor is set to 5.
- The maximal tree depth is set to 3



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

Fig. 6. Target frequent trees

A first series of results, obtained with the frequency threshold 0.05 is displayed on Fig. 7.

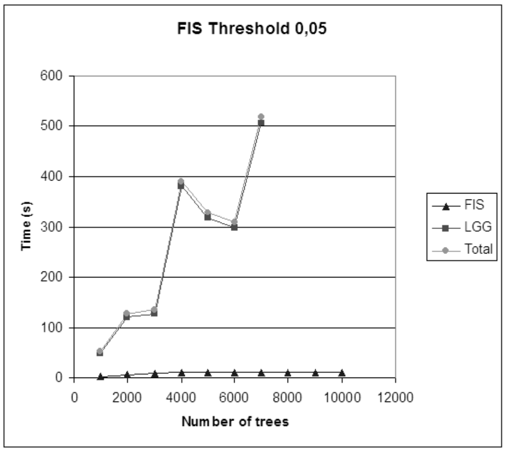
The average computational complexity (Fig. 7 a) is dominated by the least general generalization (LGG) computational complexity. The LGG thus is the limiting factor; no results could be obtained for a number of trees greater than 8,000 due to memory saturation.

Similar results are obtained by varying the average tree size. For these tests parameter p was set to 1.0, so trees were just perturbated target frequent trees, and we varied the value of δ. Actually, the FIS step only depends on the number of edges in the target frequent trees. But the supports of frequent edge sets are large, which implies that LGG considers many trees (Fig. 7 b).

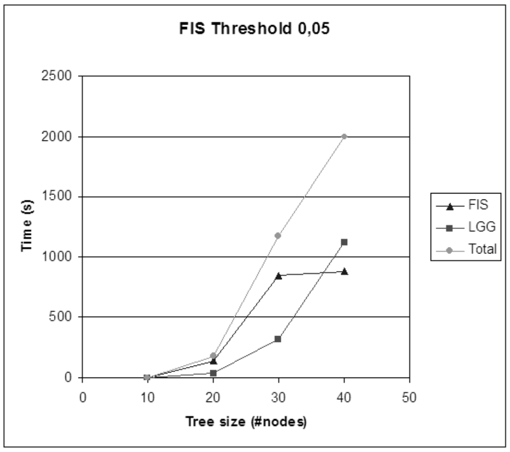
Again, it can be seen that the limiting complexity factor is the LGG step.

5.4 Empirical study of *TreeFinder* loss factor

The empirical study investigates the loss factor, measuring the percentage of "true" frequent trees missed by *TreeFinder*.



a) Varying tree number, random tree size



b) Varying tree size, 1000 trees

Fig. 7. Scalability tests results

As noted earlier (Proposition 2), the loss factor is zero when the target frequent trees do not overlap. The empirical study therefore considers five sets of target frequent trees, with increasing overlapping edges. Practically, five series of experiments are launched, corresponding to the five sets of four target frequent trees displayed in Fig. 8.

- $overlap_0$ corresponds to the baseline case, with no overlap.
- $overlap_1$ involves a small overlap (P_1 and P_2 share edge $1-3$); the overlapping rate, defined as the average percentage of edges that a target frequent tree shares with another target frequent tree, is 25%.
- $overlap_2$ involves a medium overlap (P_1 and P_2 share edge $1-3$; P_2 and P_3 share edge $1-4$); similarly the overlapping rate is 50%,
- $overlap_3$ involves a moderate overlap (P_1 and P_2 share edge $1-3$; P_2 and P_3 share edge $1-4$; P_3 and P_4 share edge $1-5$); the overlapping rate is 75%.
- $overlap_4$ involves a large overlap (P_1 and P_2 share edge $1-3$; P_2 and P_3 share edge $1-4$; P_3 and P_4 share edge $1-5$; P_4 and P_1 share edge $1-2$); the overlapping rate is 100%.

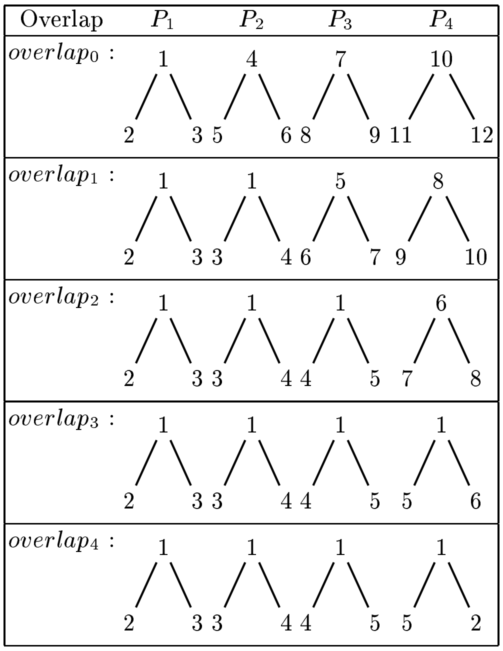


Fig. 8. Set of target trees with increasing overlap

The other generator parameters are set as follows:

- The number L of external labels is set to 50.

– The probability p of including a target frequent tree at each node is set to 0.4.

TreeFinder is run with frequency threshold ε varying in $0.2,\ 0.1,\ 0.05$.

Fig. 9 displays the loss factor for each frequency threshold ε and for each collection of target frequent trees.

The loss factor has been computed as follows. Ideally, it measures the percentage of frequent trees that would return *AprioriTree*, which are missed by *TreeFinder*. However, because of the inherent complexity of *AprioriTree*, the frequent trees occuring in the actual tree datasets cannot be extracted with reasonable complexity. The number of such frequent trees is therefore approximated during the generation step.

Practically, for a tree t, let $\mathcal{Q}(t)$ be the set of target frequent trees P_i such that t includes (at least) one occurence of P_i; $\mathcal{Q}(t)$ can easily be determined since no destructive perturbations of target frequent trees are considered in the generator.

For each such set of target frequent trees $\mathcal{Q}$ (with $\mathcal{Q}$ included in the set of target frequent trees $overlap_i$ at hand), one might thus determine its support in the generated input trees; if its support includes more than $100.\varepsilon\%$ of the input data, $\mathcal{Q}$ is a frequent tree.

The number of such frequent trees is taken as a lower bound on the number of actually frequent trees. For each frequency threshold ε and set of frequent trees $overlap_i$, the loss factor is defined as:

$$Loss(\varepsilon, overlap_i) = 1 - \frac{TF(\varepsilon, overlap_i)}{AT(\varepsilon, overlap_i)}$$

where $TF(\varepsilon, overlap_i)$ denotes the (median) number of frequent trees actually produced by *TreeFinder* over 10 datasets independently generated according to the target frequent tree set $overlap_i$, and $AT(\varepsilon, overlap_i)$ likewise is the lower bound on the number of actually frequent trees, defined as above. The loss factor so-computed thus corresponds to a pessimistic estimate.

As expected from Proposition 2, the loss factor is zero for non-overlapping target frequent trees (Fig. 9). Two types of behavior are observed depending on the frequency threshold.

For a small frequency threshold ($\varepsilon = 0.05$), the loss factor degrades gracefully as the overlapping rate increases from small to moderate; the loss is around 20% for an overlapping rate of 75%.

For a larger frequency threshold ($\varepsilon = 0.1$), the loss factor remains zero until the overlapping rate becomes actually heavy (all target frequent trees share one edge with another target frequent tree); then the loss factor abruptly rises up to almost 80%. One interpretation for this fact is the following. As the overlapping rate increases from $overlap_3$ to $overlap_4$, a discontinuity occurs: all edges become equally frequent; this entails that the edge sets can no longer be filtered out based on the frequency threshold ($TF(\varepsilon = 0.1, overlap_4) \approx TF(\varepsilon = 0.05, overlap_3)$). In the meanwhile, the number of frequent trees decreases as ε increases due to monotonicity $(AT(\varepsilon = 0.1, overlap_4) < AT(\varepsilon = 0.05, overlap_4)$, which explains why the loss factor is worse for $\varepsilon = 0.1$ than for $\varepsilon = 0.05$ in the case of $overlap_4$.

This effect can be likened to the example case illustrated on Fig. 5: the more overlap between the target frequent trees, the larger the support of any set of frequent edges, the more general the frequent trees extracted through LGG from their support, and the more likely specific trees are missed by $TreeFinder$.

It must be noted that for an even larger frequency threshold ($\varepsilon = 0.2$) the loss factor remains 0 in all the experiments.

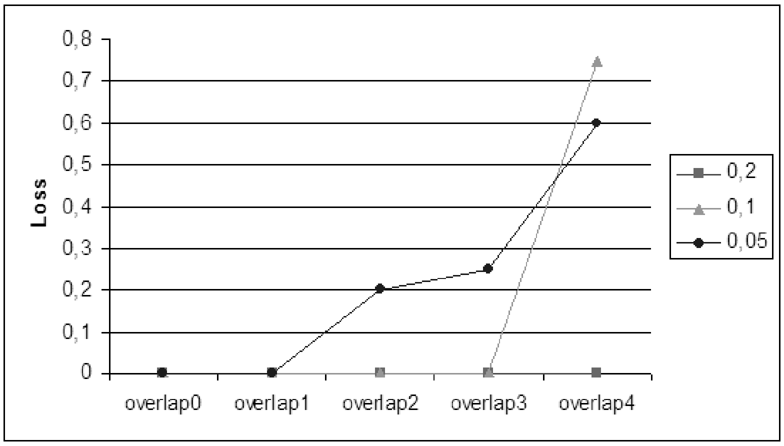


Fig. 9. Sensitivity analysis: $TreeFinder$ Loss factor for $\varepsilon = 0.05$, 0.1 and 0.2 depending on the overlapping rate between target frequent trees

6 Related Work

The extraction of frequent patterns from structured or semi-structured data has attracted much interest recently, as more and more application domains involve structured (e.g. bio-informatics, chemistry [DT01]) or semi-structured (text, Web pages [MSU$^+$01]) data. The earliest work aiming at complex frequent patterns to our best knowledge is [CH94], which gradually abstracts frequent substructures from graphs using an MDL criterion. The major difference with our approach is twofold. On the one hand, $TreeFinder$ takes advantage of the tree structure being simpler than that of graphs; typically, $TreeFinder$ would not extend to handle graph data, for the transitive closure of graphs will only deliver coarse information (connex components) in the general case. On the other hand, [CH94] can only consider frequent substructures in the graph which are made of edges; if it were applied on trees (trees being a particular case of graphs) it could only discover frequent trees in the sense of subtree inclusion (Definition

2). Same remarks apply for more recent works concerned with graph mining, AGM (Apriori-based Graph Mining) [IWM00] and FREQT (Frequent Trees) [AAK+02].

Another work aiming at frequent structured patterns is WARMR [DT01], tackling First Order Logic expressions. Again the main difference is that WARMR is designed for full-fleshed First Order Logic data, while *TreeFinder* takes advantage of the comparative simplicity of trees. Admittedly, *TreeFinder* as well as WARMR cannot scale up beyond certain limits; but the computational explosion should be delayed in *TreeFinder* compared to WARMR, for the test step (the θ-subsumption cost) is less expensive.

The closest work to ours is presented by [Zak01], which proposes two algorithms, *TreeMinerH* and *TreeMinerV*, for mining frequent trees in a forest (set of trees). Interestingly, these two algorithms also rely on subsumption inclusion. They propose a smart tree representation based on string encoding, to facilitate the candidate checking (subsumption test) step. The main difference with *TreeFinder* is that *TreeFinder* computes an approximation of the result to ensure better scalability. The approximated result is a set of frequent trees guaranteed to subsume the actual frequent trees of the input.

Note that the discovery of common tree structures has been tackled from another perspective, that of DTD inference. For instance, [PV00] studies the automatic inference of a unique DTD (tree grammar) from a set of XML data (labelled trees). The problem considered in this work is different from ours since the expected output tree structures must satisfy a subtree inclusion and no approximation is allowed.

Likewise, approximate tree matching has been considered in the perspective of answering XML queries (e.g.,[SN00,DR01,AYCS02]). [SN00] is based on tree embedding, the others use relaxation operators, some of which (e.g. node deletion) correspond to tree embedding, some others (e.g. node unfolding [DR01]) do not.

7 Discussion and Perspectives

Extending a previous work [TRS02], this paper presents the *TreeFinder* algorithm, concerned with the discovery of frequent trees w.r.t. a set of trees. Contrasting with [AAK+02], *TreeFinder* uses a relaxed inclusion test which allows for detecting trees which are not present *verbatim* in the data. Therefore, *TreeFinder* achieves a more flexible and robust tree-mining and can detect trees that could not be discovered using a strict subtree inclusion.

The main limitation of *TreeFinder* is to be an approximate miner; in the general case, it is only guaranteed to find a subset of the actual frequent trees. Its performances were empirically validated on artificial medium-size data, demonstrating that it reaches completeness (or falls short to it) in a range of problems.

The distinctive feature of *TreeFinder* is to involve a preliminary process based on a flat boolean representation of the trees, coding the presence or absence of all possible label pairs (ancestor relations) in each tree. This way, standard FIS algorithms can be applied to determine the frequent label pairs sets (FLPS). Each FLPS is thereafter exploited in order to actually construct frequent trees, through a least general generalization of the FLPS support trees. Indeed, the LGG step is computationally expensive; however, it is done only once for each FLPS. As a counterpart, this saves the subsumption tests undergone in the candidate test step, the number and complexity of which are exponential in the size of the frequent trees.

Ongoing work is concerned with investigating in more depth the loss factor of *TreeFinder*, using WARMR to emulate *AprioriTree*. Further research will investigate how the postprocessing of the frequent label pairs sets can be used to refine their support sets, and further decrease the chance of missing frequent trees.

Acknowledgements

The authors wish to thank Jérôme Azé and Jérôme Maloberti, LRI, who kindly gave respectively their implementation of Apriori, and of least general generalisation.

References

[AAK+02] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *Proc. Second SIAM International Conference on Data Mining*, pages 158–174, 2002.

[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.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of Eleventh International Conference on Data Engineering*, pages 3–14, 1995.

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

[CH94] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[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.

[IWM00] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000.

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

[Llo87] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 2nd edition, 1987.

[MHV97] H. Mannila, H.Toivonen, and A.I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

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

[MSU$^+$01] T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Discovery of frequent tree structured patterns in semistructured web documents. In Springer-Verlag, editor, *Proc. 5th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 47–52, 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.

[SA96] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of 5th International Conference on Extending Database Technology*, pages 3–17, 1996.

[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.

[TRS02] Alexandre Termier, Marie-Christine Rousset, and Michèle Sebag. Mining XML data with frequent trees. In *DBFusion'02 Workshop*, 2002.

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

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

[Zak01] Mohammed Zaki. Efficiently mining frequent trees in a forest. Technical Report 01-7, Renselaer Polytechnic Institute, 2001.