# Fuzzy Tree Mining: Go Soft on Your Nodes

Federico Del Razo Lopez[1], Anne Laurent[1], Pascal Poncelet[2], and Maguelonne Teisseire[1]

[1] LIRMM-CNRS UMR5506, Université Montpellier 2, 161 rue Ada, 34392 Montpellier, France
{delrazo,laurent,teisseire}@lirmm.fr
[2] LGI2P - EMA, France
pascal.poncelet@ema.fr

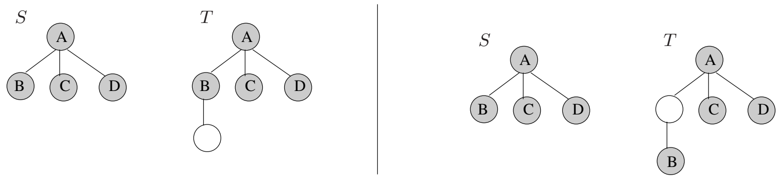
**Abstract.** Tree mining consists in discovering the frequent subtrees from a forest of trees. This problem has many application areas. For instance, a huge volume of data available from the Internet is now described by trees (e.g. XML). Still, for several documents dealing with the same topic, this description is not always the same. It is thus necessary to mine a common structure in order to query these documents. Biology is another field where data may be described by means of trees. The problem of mining trees has now been addressed for several years, leading to well-known algorithms. However, these algorithms can hardly deal with real data in a soft manner. Indeed, they consider a subtree as *fully included* in the super-tree. This means that all the nodes must appear. In this paper, we extend this definition to fuzzy inclusion based on the idea that a tree is included to a certain degree within another one, this fuzzy degree being correlated to the number of *matching nodes*.

## 1 Introduction

Tree mining is a subfield of data mining aiming at discovering automatically all the subtrees that appear frequently in a database of trees. This research area has several applications, including the discovery of mediator schemas. The background in this research is mainly constituted by the work by Asai et al. and Zaki et al.[2,10,13,17,18]. This work address the problem of tree mining considering several ways to define when a tree $S$ is included within another one $T$. Inclusion is then decided depending on the way ancestry and brotherhood are considered. In this respect, the authors distinguish between approaches where (i) either all the pair of connex nodes in the tree $S$ must be found in $T$ with no intermediate node, (ii) or some intermediate nodes are accepted. Figure 1 illustrates this difference.

The work from the literature is then twofolded, considering both:

– the representation of the trees,
– the extraction of frequent subtrees.



(a) with connex nodes (b) with intermediate nodes

**Fig. 1.** Traditional tree inclusion

It should be noted that designing efficient algorithms to tackle the problem of extracting frequent subtrees is highly correlated to the representation of the trees, as this representation may help scanning the trees. The process of extracting frequent subtrees is based on the Apriori process[1], which is a recursive process. It can be divided into the following steps: for each size of trees (i) generation of candidates and (ii) validation of candidates. A candidate is a tree that is considered as being potentially frequent. The candidates of size $k$ are built based on the frequent subtrees of size $k-1$. This family of methods is well-known and has been applied for tree mining. However, all the existing methods consider that a tree **is** or **is not** included within another one, which is too restrictive to be efficient and relevant. We first propose the concept of fuzzy tree mining, which has been introduced in [11]. This concept has been detailed in [12], where we have defined a fuzzy ancestor-descendant relation (fuzzy vertical path). In this paper, we consider another way to softenize the tree inclusion definition by considering that some nodes may be discarded (partial inclusion). In classical approaches, *all* the nodes of a subtree $S$ must be included in a tree $T$ if $S$ is included in $T$. For instance, Figure 2 shows a tree $S$ that will not be considered as being included in $T$. However, we argue that this is too restrictive when mining data from the real world where imperfections are often present. For instance, in Figure 2 $S$ has 75% of its nodes included in $T$.

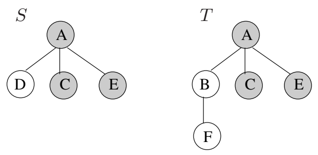


**Fig. 2.** Partial Inclusion

The challenging part of our work is that we want to remain efficient, in the framework of fuzzy data mining. The paper is organized as follows: Section 2 recalls the existing work on tree mining and our previous work on dealing with fuzzy tree mining. Section 3 introduces the necessary definitions for dealing with partial inclusion. Section 4 introduces the algorithms we design for extracting frequent subtrees from a tree database in a soft manner by considering partial inclusion. Finally, Section 5 concludes this work and presents our future working directions.

## 2 Background

In this section, we recall from the literature and from previous work the basic definitions of tree mining and the ways trees can be represented.

### 2.1 Tree Mining

A *tree* is a connected graph containing no cycle. A tree is composed by nodes, which are linked by edges such that their exists a particular node called *root* and such that all the nodes but the root are composed by sub-trees. A tree is said to be an *ordered tree* if the children from a node are ordered. A tree is said to be an *unordered tree* otherwise.

Let $\mathcal{L} = \{a, b, c, ...\}$ be a set of labels. A *labeled ordered tree* is a tree $T = (r, N, B, L, \preceq)$ where: $r$ is the root, $N$ is the set of nodes, $B$ is the set of edges such that $B \subseteq N^2$, $(L : N \rightarrow \mathcal{L})$ is a mapping from the set of labels $\mathcal{L}$ to the set of nodes $N$, and $\preceq$ is an ordered relation between brother nodes.

Tree Mining refers to the process of extracting all the subtrees that appear frequently in a database $D$ of trees. The frequency is computed using the notion of support: Given a database $D$, the *support* of a tree $S$ is the proportion of trees from the database where $S$ is embedded:

$$Support(S) = \frac{\#\ of\ trees\ where\ S\ is\ embedded}{\#\ of\ trees\ in\ D}$$

$S$ is said to be frequent if $Support(S) \geq \sigma$ where $\sigma$ is a user-defined minimal support threshold.

Several kinds of tree inclusion can be defined [3], depending on the way ancestors and siblings are considered. For instance, [18] defines the inclusion as follows:

**Definition 1.** *A tree $S$ is* embedded *into a tree $T$ if there exists an injective and total function $\phi : N_S \rightarrow N_T$ such as for all $n, m \in N_S$:*

- *$\phi$ keeps the labels: $L_S(n) = L_T(\phi(n))$;*
- *$\phi$ keeps the relations* ancestor-descendant*: $(n, m) \iff (\phi(n), \phi(m))$;*
- *$\phi$ keeps order relations: $(n \preceq_S m) \iff (\phi(n) \preceq_T \phi(m))$.*

As highlighted in [11], fuzzy data mining can help when mining frequent subtrees from a tree database. Four ways to soften classical approaches has been proposed:

- *ancestor-descendant degree*: in classical approaches, a node *is* or *is not* an ancestor of another one. In our approach, we propose to indicate by a degree

between 0 and 1 to which extent a node is an ancestor of another one, meaning that if there are too many nodes between them, then this degree will decrease.
- *sibling ordering degree*: in classical approaches, nodes *are* or *are not* searched in the initial order. In our approach, we propose to indicate by a degree the sibling disorder.
- *partial inclusion*: in classical approaches, all the nodes from the candidate must be in the tree. In our approach, we propose to soften this rule by considering the degree to which the nodes are embedded in the tree.
- *Node similarity*: in classical methods, a node label *is* or *is not* the same as another one. In our approach, we propose to soften this by indicating by a degree to which extent two nodes are similar (*e.g.* based on a taxonomy).

The *ancestor-descendant degree* has been studied in [12]. In the rest of this paper, we focus on the partial inclusion.

**Algorithms.** Several algorithms have been designed to address the problem of tree mining: *TreeMiner* in [18], *FreqT* in [2], *Chopper* [14], *FreeTreeMiner* [5] and *CMTreeMiner* [4]. All of them are based on a process consisting of the following two iterative or recursive steps: generation of candidates and validation of candidates. This process starts from the candidates that contain only one node, to discover the frequent 1-node subtrees, which are used to build the 2-node candidates, and so on.

These two steps have been studied. The generation of candidates is either based on methods that build trees containing $n$ nodes by considering one tree containing $n-1$ nodes and adding another node, or is based on methods that mix two subtrees containing $n$ nodes and sharing $n-1$ nodes to build a new candidate subtree containing $n+1$ nodes.

The validation aims at checking whether a tree is embedded within another one. Several approaches have been proposed. In our previous work, we have defined some algorithms that are based on the idea of *anchoring*: we try to anchor the root of the subtree until we find a node that matches. Then the following nodes are tested until (i) it is no more possible to find some remaining nodes for matching, or (ii) an incompability has been detected or (iii) the subtree fully matches.

## 2.2 Tree Representation

Several ways of representing trees have been proposed to support the algorithms cited above. The representation impacts the two steps discussed above (generation and validation of candidates). However, it may be the case that the representation is too rich and requires too much memory (e.g. representing trees as strings). We have thus proposed in previous work a low-memory representation of trees: RSF. This representation is defined below.

When representing a tree $T$, we keep in mind the following property: all the nodes but the root have one and only one predecessor. We propose thus to use two vectors to represent a tree, as proposed in [15]. The first vector is denoted by $st$. It stores the position of each node predecessor. Nodes are numbered considering a depth-first traversal. The root is numbered as being at position 0, with $st[0] = -1$ since it has no predecessor. The values $st[i], i = 1, 2, ..., k-1$ correspond to all other predecessor positions, as shown on Figure 3.

This representation provides a constant-time method to retrieve the predecessor of a node. Moreover, it allows us to find directly the most right leaf when considering an index $k$. Finally, when visiting the tree, it is possible to build all direct links from predecessors to descendants.

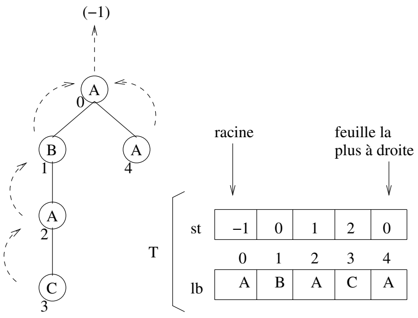


**Fig. 3.** Representation of a Tree

The second vector is denoted by $lb$. It is used to store all the tree labels. $lb[i], i = 0, 1, ..., k-1$ are the labels of each node $n_i \in T$.

The data structure we have chosen needs very low memory since it is reduced to the size of $2|T|$. Moreover, it has good properties when mining frequent subtrees.

As presented in [12], in order to manage trees as efficiently as possible, each tree $T$ is transformed into a binary representation denoted by $T_B$ where each node cannot have more than two children [9]. For this purpose, we propose the following transformation: the first child of a node is put as the left-hand child while the other children are put in the right-hand path, as illustrated in Fig. 4 b).

**Encoding Binary Trees.** Once the tree has been transformed into a binary tree, nodes must be encoded in order to be retrieved. The encoding is then used first in order to identify each node and second in order to determine whether a node is a child or a brother. In order to do so, we consider the Huffman algorithm [8] which we slightly modify in order to fit our needs. The root has address 1. The other node addresses are computed by concatenating the father address with: 1

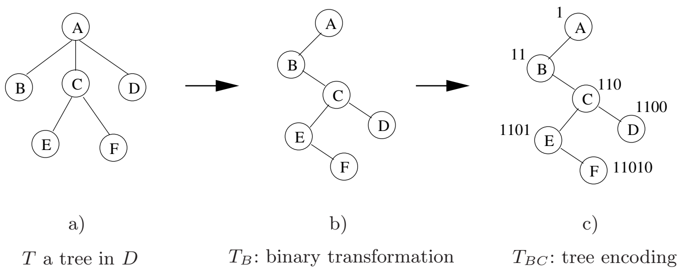


**Fig. 4.** Example of a Binary Tree Transformation and Node Addressing

if it is a child (left-hand path) and 0 otherwise (right-hand path), as shown on Fig. 4 c).

## 3 FTMnodes: Definitions

In this paper, we formally extend the definition of tree inclusion to partial inclusion based on the number of nodes that are matched. Partial inclusion is defined as follows:

**Definition 2.** *Given a null value $\perp$, a tree $S$ is* partially embedded *into a tree $T$ with a degree $\delta(S,T)$ if there exists an injective and total function $\phi : N_S \rightarrow N_T \cup \perp$ such that for all $n, m \in N$:*

- *$\phi$ keeps the labels: $L_S(n) = L_T(\phi(n))$ or $\phi(n) = \perp$;*
- *$\phi$ keeps the relations* ancestor-descendant*: $(n,m) \iff (\phi(n), \phi(m))$ or $(\phi(n), \phi(m)) = \perp$;*
- *$\phi$ keeps the order relations: $(n \preceq_S m) \iff (\phi(n) \preceq_T \phi(m))$ or $(\phi(n) \preceq_T \phi(m)) = \perp$;*
- *$\delta(S,T) = \frac{|\{n \in S:\ \phi(n) \neq \perp\}|}{\#\ of\ nodes\ in\ S}$.*

From this definition, it is possible to define the support of a subtree, as follows:

**Definition 3.** *Given a database $D$ and a tree $S$, the support of $S$ in $D$ is given by:*

$$Support(S) = Agg_{T \in D}(\delta(S,T))$$

*where Agg is a function of aggregation.*

For instance, we may use Ordered Weighted Aggregators (also known as OWA) [16]. An OWA operator of dimension $n$ is a mapping

$$F : R^n \rightarrow R$$

that has an associated $n$ vector $W = (w_1, w_2, \ldots, w_n)^T$ such that $w_i \in [0,1]$ and $\sum_{i=1}^{n} w_i = 1$. We have $F(a_1, a_2, \ldots, a_n) = \sum_{j=1}^{n} w_j \cdot b_j$ where $b_j$ is the $j^{th}$ largest value of the $a_i$.

For instance, the average may be applied:

$$Support(S) = \frac{\sum_{T \in D} \delta(S,T)}{\#\ of\ trees\ in\ D}$$

In fact, we consider a thresholded $\Sigma$-count so that:

- a tree cannot be considered as being embedded within another one if the number of embedded nodes is too low,
- the degree to which a tree is embedded within another one is taken into account.

We thus have:

**Definition 4.** *Given a database $D$, a threshold $\tau$ and a tree $S$, the support of $S$ in $D$ is given by:*

$$Support(S) = \sum_{T \in D} (\alpha_\tau(\delta(S,T)))$$

*where*

$$\alpha_\tau(x) = \begin{cases} 0 \ if\ x > \tau \\ x\ otherwise \end{cases}$$

## 4 FTMnodes: Algorithms

Note that in the classical case, mining totally included trees allows to cut in the database scan since whenever a node cannot be matched, there is not necessary to look for the other ones. In our approach, outliers are accepted, which may be considered as a drawback considering scalability. However, it is still possible to cut off the search when the proportion has been overpassed.

As defined previously, we consider that a tree cannot be considered as being embedded within another one if the number of matching nodes is not greater than a user-defined threshold $\tau$. This definition not only guarantees the quality of the research from a semantic point of view, but it also guarantees the scalability of our approach. Indeed, it is then possible to draw the property of anti-monotonicity which is the basis of levelwise algorithms. We have the following properties:

*Considering that the first $n$ nodes of tree $S$ matched to nodes from $T$, and that $\pi$% of the nodes of $S$ have been matched, then the first $n+1$ nodes of $S$ cannot be embedded in $T$ to a proportion greater than $\pi$.*

This property comes from the fact that if it has not been possible to match $\nu$ nodes among the first $n$ nodes of $S$, then the number of nodes being not matched when going ahead in the process to the first $n+1$ nodes will either be equal or will be greater (equal to $\nu+1$).

As a consequence, whenever the threshold $\tau$ is overpassed, the process can be stopped for this path as it will never be considered in the thresholded $\sum$-count.

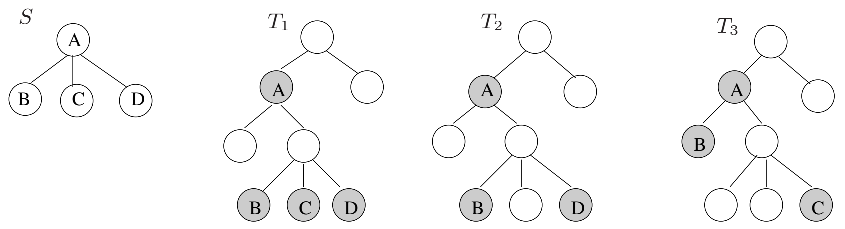


**Fig. 5.** Several ways of including $S$ in $\{T_1, T_2, T_3\}$ with $\tau = 0.75$ (at least 3 nodes out of 4)

Note that it may be the case that a subtree is included within another one in different manners, as illustrated by Figure 5. In this case, the best degree of inclusion will be considered and this best degree is found by maintaining all the possible ways of inclusion until all the solutions have been considered as studied in [7] when considering fuzzy sequential patterns.
The following process is thus considered in our approach (algorithms 1 and 2):

- anchoring
- for each possible anchor, for each node $n$ in $S$ to be matched
  - scan the nodes of $T$ until $n$ is matched, start another way to find the other possible matches,
  - if no match is possible then go to the next node in $n$ and increment the number of mismatched nodes
  - if the number of mismatched nodes is greater than the threshold $\tau$ or if $T$ has been fully scanned, then discard this anchor
- compute the best inclusion from non discarded anchoring paths

```
Data: $S$ //subtree to validate,
      $T$ //tree from database
Result: $true$ //if $S$ is embedded within $T$
$\mathcal{M}$    // mapping set of $S$ within $T$;
foreach $node\ m \in N_T$ do
    $n \leftarrow root(S)$;
    if $L(n) = L(m)$ then
        PartialInclusionDegree($S, T, n, m, M$);
        $\mathcal{M} \leftarrow \bigcup M$;
return the best inclusion $\{M \in \mathcal{M} | MIN\{M.mismatchedNodes\}\}$;
```

**Algorithm 1.** Anchoring

Note that our approach is consistent, meaning that if $\tau = 1$ (i.e. all the nodes must be mapped), then our algorithms are exactly the same as the ones defined in the crisp case [6].

```
Data: $S$ //subtree to validate, $T$ //tree from database,
          $n, m$ //anchoring point, $n \in S$ to $m \in T$,
          $M$ //occurrence of $S$
$M[n] \leftarrow m$;
$n \leftarrow n + 1$;
if $n <= |S|$ then
    $P \leftarrow \{w\colon w \in T$ such that $L(w) = L(n)$ and $m \preceq w$ and $ancestor(w) =$
        $M[ancestor(n)]\}$;
    if $P \neq \emptyset$ then
        foreach node $w \in P$ do
            PartialInclusionDegree($S, T, n, w, M$);
    else
        $M.mismatchedNodes \leftarrow M.mismatchedNodes + 1$;
        if $M.mismatchedNodes >= \tau$ then
            exit;
        else
            PartialInclusionDegree($S, T, n, m, M$);
return;
```

**Algorithm 2.** PartialInclusionDegree

## 5 Conclusion

In this paper, we have detailed our previous work on fuzzy tree mining by giving the necessary definitions and algorithms in order to address the partial inclusion. Partial inclusion is a big deal in tree mining as it is not possible to consider full matches in real applications. However, it is necessary to remain scalable as the volumes of data being considered in real databases is huge. We thus design solutions based on levelwise algorithms, which consider anti-monotonic properties that guarantee the scalability. The algorithms presented here are currently implemented, and it is possible to conclude that this approach allows the extraction of more frequent subtrees (as fuzziness is introduced) while remaining scalable. Future work include the comparison of the results depending on the choices of the aggregation function. This comparison will be lead both on the quality of frequent subtrees and on runtime, as some aggregation functions are easier to compute than other ones.

## References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. *In Proceedings of the 20th* VLDB *Conference,* Santiago, Chile, 2002.
2. T. Asai, K. Abe, S. Kawasoe, H. Arimura, and H. Sakamoto. Effcient substructure discovery from large semi-structure data. In *2nd Annual* SIAM *Symposium on Data Mining, SDM2002,* Arlington, VA, USA, 2002. Springer-Verlag.

3. Y. Chi, R. R.Muntz, S. Nijssen, and J. N. Kok. Frequent subtree mining - an overview. *Fundamenta Informaticae XXI,* pages 1001–1038, 2005.
4. Y. Chi, Y. Yang, and R. Muntz. Cmtreeminer: Mining both closed and maximal frequent subtrees. In *The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04),* 2004.
5. Y. Chi, Y. Yang, and R. R. Muntz. Indexing and mining free trees. In *International Conference on Data Mining 2003 (ICDM2003),* 2003.
6. F. Del Razo, A. Laurent, P. Poncelet, and M. Teisseire. Rsf - a new tree mining approach with an efficient data structure. In *Proceedings of the joint Conference: 4th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2005),* pages 1088–1093, 2005.
7. C. Fiot, A. Laurent, and M. Teisseire. From crispness to fuzziness: Three algorithms for soft sequential pattern mining. *IEEE Transactions on Fuzzy Systems. (to appear),* 2007.
8. D. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the Institute of Radio Engineers,* 1952.
9. D. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms.* Addison-Wesley, 1973.
10. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *IEEE International Conference on Data Mining (ICDM),* 2001.
11. A. Laurent, M. Teisseire, and P. Poncelet. *Fuzzy Data Mining for the Semantic Web: Building XML Mediator Schemas,* chapter 12. Elsevier, E. Sanchez(ed.), To appear, 2006.
12. S. Sanchez, A. Laurent, P. Poncelet, and M. Teisseire. Fuzbt: a binary approach for fuzzy tree mining. In *Proceedings of the 11th IPMU International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2006),* 2006.
13. A. Termier, M.-C. Rousset, and M. Sebag. Treefinder, a first step towards XML data mining. In *IEEE Conference on Data Mining (ICDM),* pages 450–457, 2002.
14. C. Wang, Q. Yuan, H. Zhou, W.Wang, and B. Shi. Chopper: An eficient algorithm for tree mining. *Journal of Computer Science and Technology,* 19:309–319, May 2004.
15. M. A. Weiss. *Data Structures And Algorithm Analysis In* C. AddisonWesley, 1998.
16. R. Yager. Families of owa operators. *Fuzzy Sets and Systems,* 57(3):125–148, 1993.
17. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *IEEE Conference on Data Mining (ICDM),* 2002. 18.
18. M. J. Zaki.*Efficiently Mining Frequent Trees in a Forest.* In KDD'02, Edmonton, Alberta, Canada, 2002. ACM.