FTMnodes: Fuzzy tree mining based on partial inclusion

F. Del Razo Lopez[c], A. Laurent[a,*], P. Poncelet[b], M. Teisseire[a]

[a] *LIRMM, CNRS UMR 5506, Univ. Montpellier 2, France*
[b] *LGI2P, EMA, France*
[c] *Instituto Tecnológico de Toluca, Mexico*

Available online 26 February 2009

Abstract

Mining frequent patterns from huge databases have been addressed for many years and results have been applied to many fields, including banking, marketing, biology, health, etc. Fuzzy approaches have been proposed in order to soften the constraints on the patterns found by the algorithms. However, when dealing with complex databases such as tree databases (as it is for instance the case for XML databases), only a few methods have been proposed in order to handle soft constraints in discovering the frequent subtrees from a forest of trees. Such algorithms can hardly deal with real data in a soft manner. Indeed, they consider a subtree as *fully included* in the super-tree, meaning 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*. We propose the FTMnodes method together with the associated definitions, and we report the experiments lead on synthetical and real databases, showing the interest of our approach.

Keywords: Semi-structured data; Tree mining; Tree inclusion; Soft inclusion

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. Also known as *structured data mining*, this research area has several applications, including for example phylogeny, or the automatic discovery of mediator schemas from XML databases.

Indeed, trees are now one of the major medias for hierarchical data exchanges, using the semi-structured XML format. For instance, DBLP[1] references [1] are stored using the XML format. Each entry of the database is a tree describing a paper published in a DBLP linked conference or journal. Fig. 1 shows the typical form of such trees.

Mining such databases allow us to get information about how the data are structured, thus providing clues to query them.

The background in this research is mainly constituted by the work by Asai et al. and Zaki et al. [2–6]. These 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

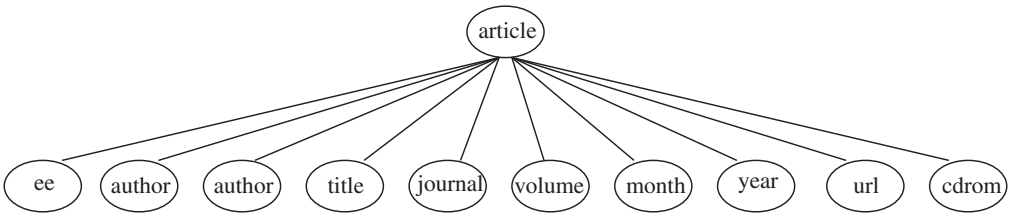

Fig. 1. Form of the DBLP trees.

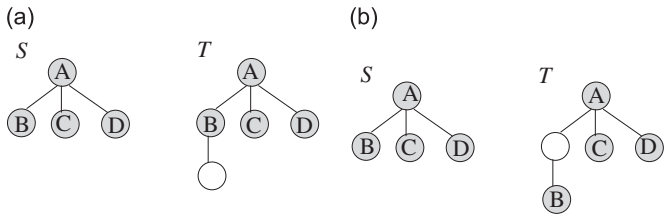

Fig. 2. Traditional tree inclusion: (a) with connex nodes and (b) with intermediate nodes.

distinguish between approaches where (i) either all the pairs of connex nodes in the tree S must be found in T with no intermediate node, (ii) or some intermediate nodes are accepted. Fig. 2 illustrates this difference.

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

- the representation of the trees and
- the extraction of frequent subtrees.

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. Usually, the process of extracting frequent subtrees is based on the a priori approach [7], which is a recursive process. It can be divided into the following two steps: for each size of trees (i) generation of candidates and (ii) validation of candidates within the database. Roughly speaking, a candidate is a tree that is considered as being potentially frequent. Basically, the well-known method applied for tree mining performs by building the candidates of size k from the frequent subtrees of size $k-1$. However, all the existing methods consider that a tree *is* or *is not* included within another one, which is definitively too restrictive to be efficient and relevant. For example, when building a mediator schemata we must extract schemas from several databases and with a too strict inclusion, the resulting mediator schema will be empty and useless. The problem would be the same when dealing with XML queries, it is obvious that if we do not relax some constraints we will not be able to extract frequent substructures. So, we first propose the concept of fuzzy tree mining, which has been introduced in [8]. This concept has been detailed in [9], 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 (*e.g. 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, Fig. 3 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 Fig. 3 S has 75% of its nodes included in T [10]. Furthermore, an other 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. Conducted experiments both on synthetical and real databases are reported in Section 5. Finally, Section 6 concludes this work and presents our future working directions.

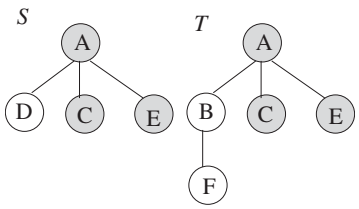

Fig. 3. Partial inclusion.

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 subtrees. 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 $\mathcal{L} = \{a, b, c, \ldots\}$ be a set of labels. A *labelled ordered tree* is a tree $T = (r, N, B, L, \preccurlyeq)$ 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 $\preccurlyeq$ is an ordered relation between brother nodes. For instance, in Fig. 3, S and T are two labelled ordered trees. In S, the root node is A and all its children are ordered: $D \preccurlyeq C \preccurlyeq E$. In T, children of A are such that $B \preccurlyeq C \preccurlyeq E$.

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 $Support(S) \geqslant \sigma$ where σ is a user-defined minimal support threshold.

Depending on the way ancestors and siblings are considered, several kinds of tree inclusion can be defined [11]. For instance, [6] 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 $n, m \in N_S$:

- ϕ keeps the labels: $L_S(n) = L_T(\phi(n))$;
- ϕ keeps the relations *ancestor-descendant*: $(n, m) \Longleftrightarrow (\phi(n), \phi(m))$; and
- ϕ keeps order relations: $(n \preccurlyeq_S m) \Longleftrightarrow (\phi(n) \preccurlyeq_T \phi(m))$.

As highlighted in [8], 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. We propose, in our approach, 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. We propose to soften this rule by considering the degree to which the nodes are embedded in the tree.

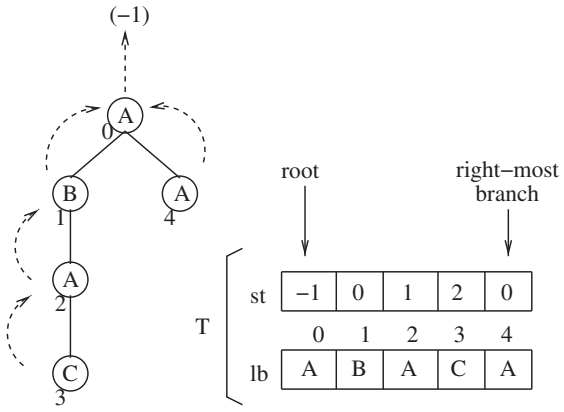

Fig. 4. Representation of a tree.

- *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 [9]. In the rest of this paper, we focus on the partial inclusion.

2.1.1. Algorithms

Several algorithms have been designed to address the problem of tree mining. The main ones are reported in *TreeMiner* [6], *FreqT* [2], *Chopper* [12], *Free-Tree-Miner* [13], and *CMTreeMiner* [14]. 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 extensively 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 incompatibility 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 parent node (a node has at most one parent). 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 $st[i],\ i = 1, 2, \ldots, k-1$ correspond to all other predecessor positions, as shown in Fig. 4.

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.

The second vector is denoted by lb. It is used to store all the tree labels. $lb[i],\ i = 0, 1, \ldots, k-1$ 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.

2.3. Fuzzy tree mining

Fuzzy data mining has been addressed for the last years, aiming at providing methods for discovering both trends and exceptions using the fuzzy logic framework to provide more comprehensible and valuable results, but remaining scalable.

When dealing with fuzzy constraints in data mining, the challenge is often to maintain several solutions during the mining process so as to finally come up with the best solution of counting to which extent an object can fit a constraint. The counting methods have to be adapted to soft constraints, as for instance done for fuzzy sequential pattern mining in [16].

When dealing with fuzzy tree mining, the counting method has also to be revised, based on the definition of how to consider the fuzzy inclusion of a tree within another one. As it has been pinpointed before, four fuzzy inclusions have been presented in [8] to consider fuzzy inclusion of a tree within another one:

- The first way aims at considering the vertical paths of trees. Contrary to induced one, embedded inclusion allows us to soften the ancestor-descendant relationship by computing to which extent a tree is included within another one with respect with a fuzzy membership function defining the approximate acceptable number of nodes that can occur between the ancestor and its descendant. Sanchez [9] proposes to control this degree of relationship.
- In the second way, we address the horizontal paths of trees. While classical approaches consider that sibling nodes are ordered or not, there is no way to consider the proportion of sibling nodes included.
- The third way (addressed in this paper) generalizes the previous one by considering the proportion of nodes.
- The last one considers similarities between nodes.

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 $\phi : N_S \to N_T \cup \perp$ such that for all $n, m \in N$:

- ϕ keeps the labels: $L_S(n) = L_T(\phi(n))$ or $\phi(n) = \perp$;
- ϕ keeps the relations *ancestor-descendant*: $(n, m) \iff (\phi(n), \phi(m))$ or $(\phi(n), \phi(m)) = \perp$;
- ϕ keeps the order relations:
  $(n \preccurlyeq_S m) \iff (\phi(n) \preccurlyeq_T \phi(m))$ or $(\phi(n) \preccurlyeq_T \phi(m)) = \perp$;
- $\delta(S, T) = |\{n \in S : \phi(n) \neq \perp\}|/\#$ *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 (OWA) [17]. An OWA operator of dimension n is a mapping

$$F : R^n \to R$$

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 jth 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 Σ-count so that:

- a tree cannot be considered as being embedded within another one if the number of embedded nodes is too low and
- 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 τ 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

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

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 τ. 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 (e.g. the a priori algorithm). We have the following property:

Property. *Considering that the first n nodes of tree S matched to nodes from T, and that π% 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 π.*

This property comes from the fact that if it has not been possible to match v 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 $v + 1$).

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

Note that it may be the case that a subtree is included within another one in different manners, as illustrated by Fig. 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 Section 2.3.

The following process is thus considered in our approach:

- 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 and
  - if the number of mismatched nodes is greater than the threshold τ or if T has been fully scanned, then discard this anchor.
- compute the best inclusion from non-discarded anchoring paths.

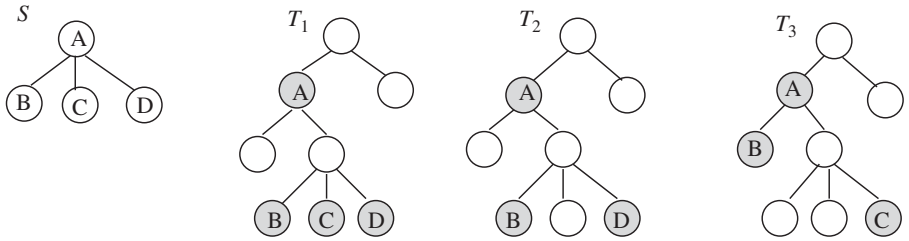

Fig. 5. Several ways of including S in $\{T_1, T_2, T_3\}$ with $\tau = 0.75$ (at least 3 nodes out of 4).

Algorithms 1 and 2 formally describe the process:

Algorithm 1. ANCHORING.
Data: S //subtree to validate,
T //tree from database
Result: *true* //if S is embedded within T

$\mathcal{M} \leftarrow \emptyset$; // 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 2. PARTIALINCLUSIONDEGREE.
Data: S //the subtree to validate,
T //a tree from the database,
n, m //anchoring points, from $n \in S$ to $m \in T$,
M //occurrence of S

$M[n] \leftarrow m$;
$n \leftarrow n + 1$;
if $n <= |S|$ **then**
  $P \leftarrow \{w : w \in T$ such that $L(w) = L(n)$ and $m \preccurlyeq w$ 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;

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 [18] (see experiments below).

Table 1
Parameters chosen for the generation of the tree database.

Parameter	Value
Number of trees to be generated	10,000
Maximal depth of a tree	5
Maximal number of edges for a node	5
Maximal number of labels	50
Probability for a node to be a parent	0.4

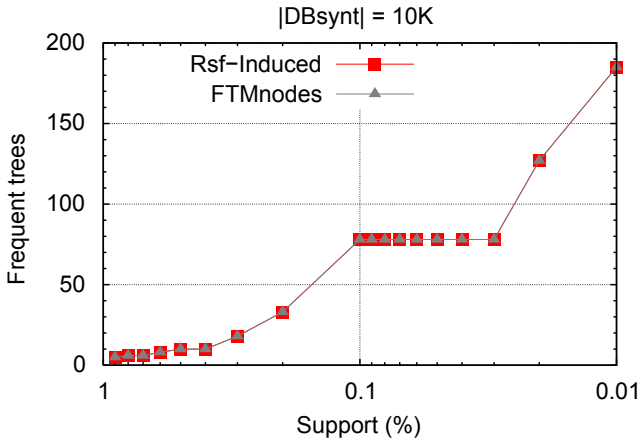

Fig. 6. Number of frequent trees w.r.t. support with $\tau = 1$.

5. Experiments

In this section, we report experiments on synthetical and real databases. We aim at showing that by considering soft inclusion instead of crisp induced inclusion, it is easier to find out frequent subtrees (more frequent patterns are discovered) but that we also remain scalable as we can run our program on large databases. We thus use RSF with the induced inclusion and we compare the results to the ones obtained by considering the partial inclusion defined in this paper.

We detail below the results we have obtained using a core 2 duo/2 Gb computer, running Mac OS X (leopard 10.5). The algorithms have been developed using C + + (STL library) and compiled with gcc 4.0.1.

5.1. Synthetical databases

Synthetical databases are generated using the XML tree generator developed by Alexandre Termier [4]. This application allows to adapt the tree parameters (e.g. number of trees, depth of trees, number of distinct labels) to the behaviour we want to test (e.g. scalability with respect to the depth of the tree).

Table 1 details the parameters chosen for our experiments.

Note the memory used for these experiments is not reported here as it has been regular and non-explosive, whatever the database type (synthetical of real). This result is due to the RSF representation, which is a non-memory consuming one.

As shown in Figs. 6 and 7, and as expected, the results obtained when considering $\tau = 1$ are exactly the same as the ones obtained with the crisp algorithm. Note that the runtime, displayed by Fig. 8, is also the same, showing that our approach is fully consistent.

When considering τ values lower than 1, more frequent subtrees are discovered, as the inclusion constraint has been relaxed. Figs. 9–12 report these results.

The number of candidates has of course been also increased, as shown in Figs. 13–16.

The runtime remains reasonable, as shown in Figs. 17–20.

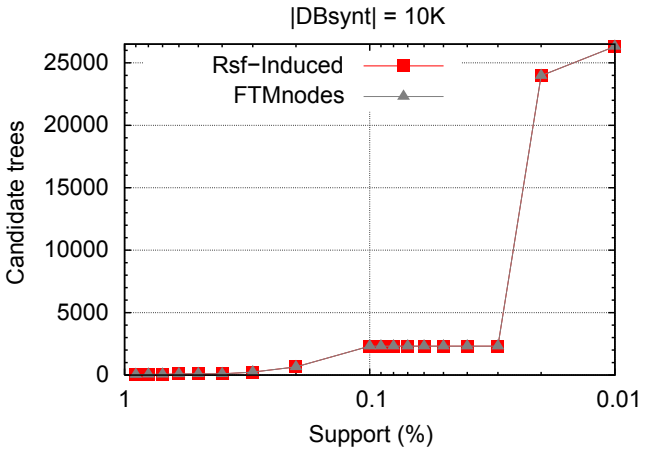

Fig. 7. Number of candidates generated w.r.t. support with $\tau = 1$.

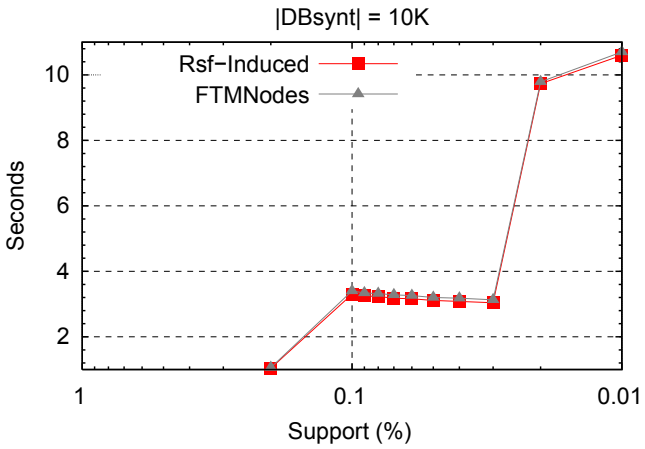

Fig. 8. Runtime w.r.t. support with $\tau = 1$.

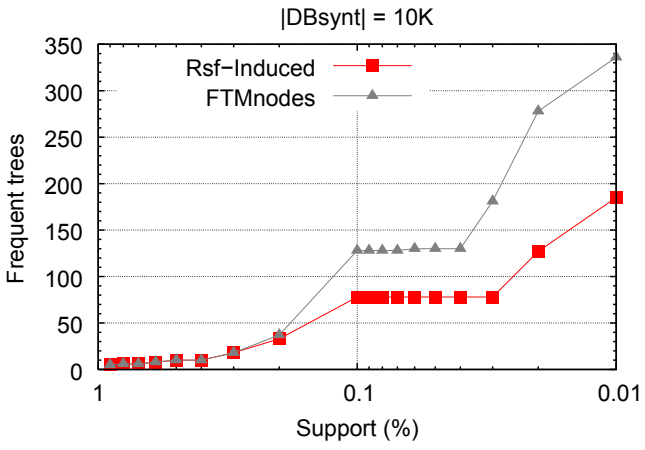

Fig. 9. Number of frequent trees w.r.t. support with $\tau = 0.9$.

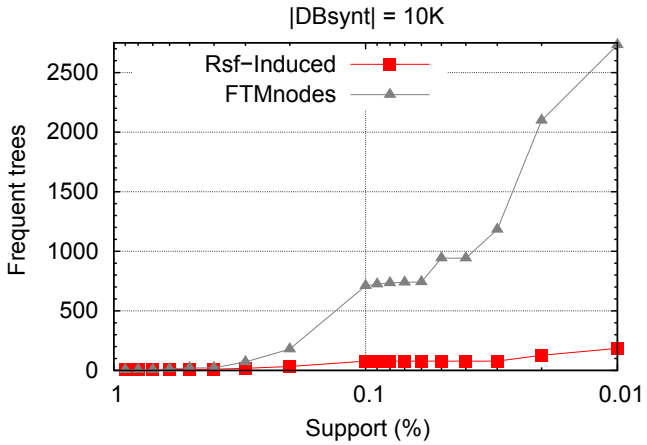

Fig. 10. Number of frequent trees w.r.t. support with $\tau = 0.8$.

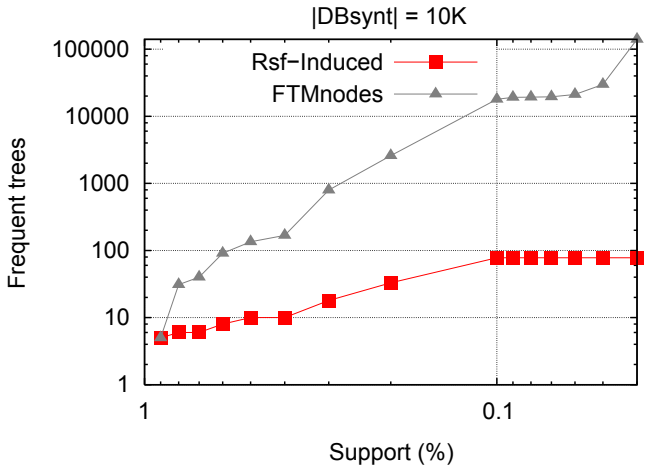

Fig. 11. Number of frequent trees w.r.t. support with $\tau = 0.7$.

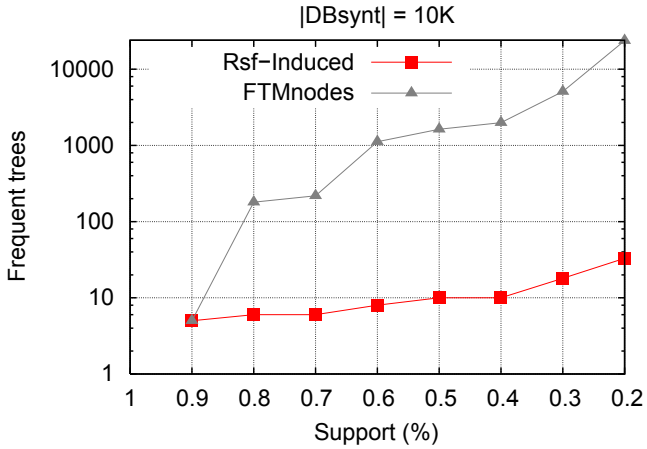

Fig. 12. Number of frequent trees w.r.t. support with $\tau = 0.6$.

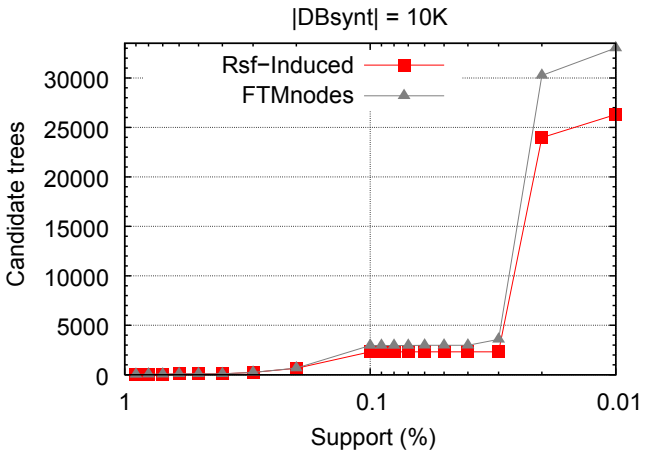

Fig. 13. Number of candidates generated w.r.t. support with $\tau = 0.9$.

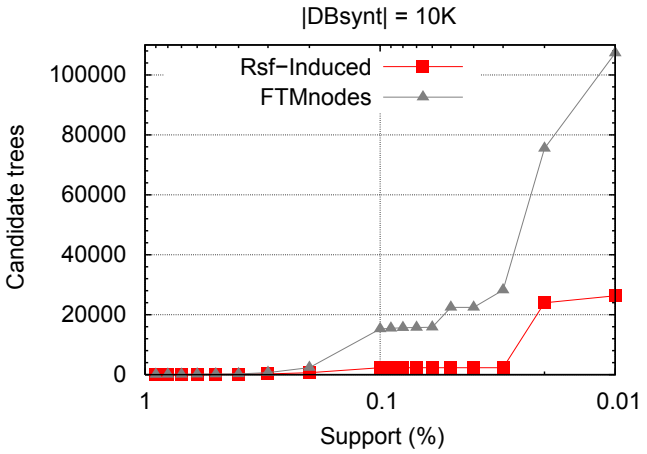

Fig. 14. Number of candidates generated w.r.t. support with $\tau = 0.8$.

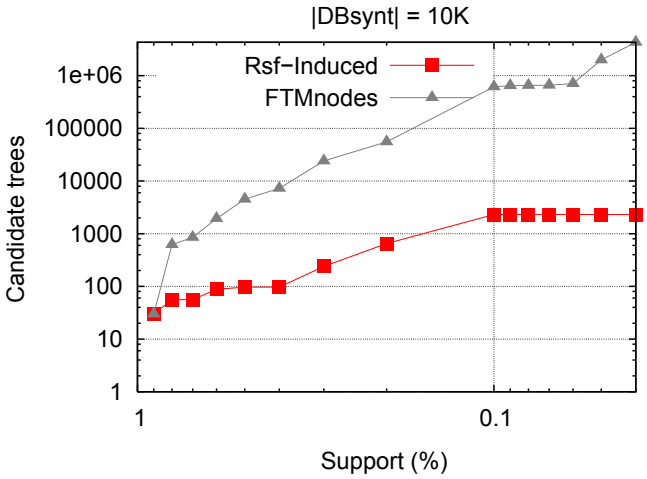

Fig. 15. Number of candidates generated w.r.t. support with $\tau = 0.7$.

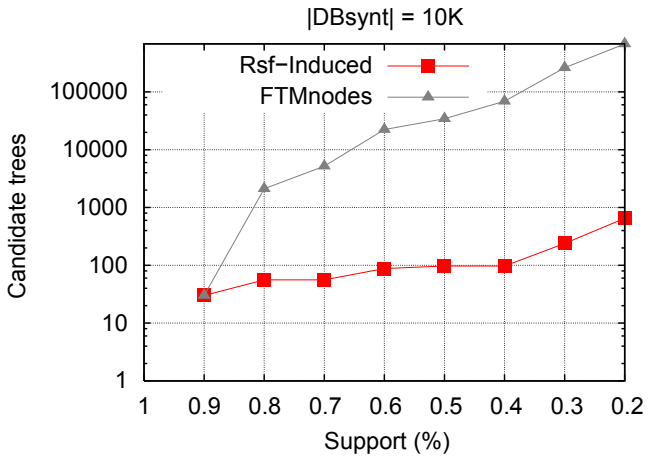

Fig. 16. Number of candidates generated w.r.t. support with $\tau = 0.6$.

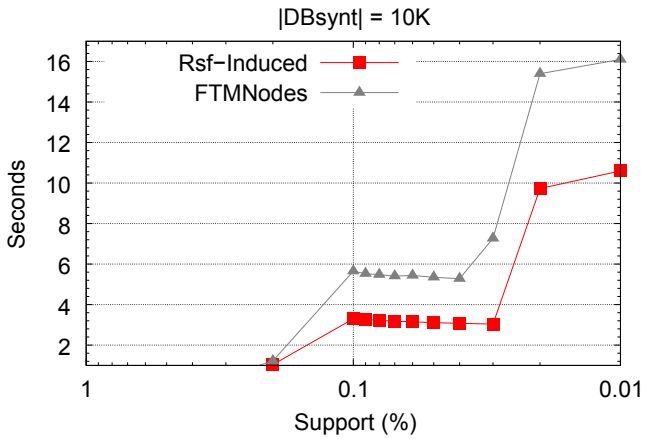

Fig. 17. Runtime w.r.t. support with $\tau = 0.9$.

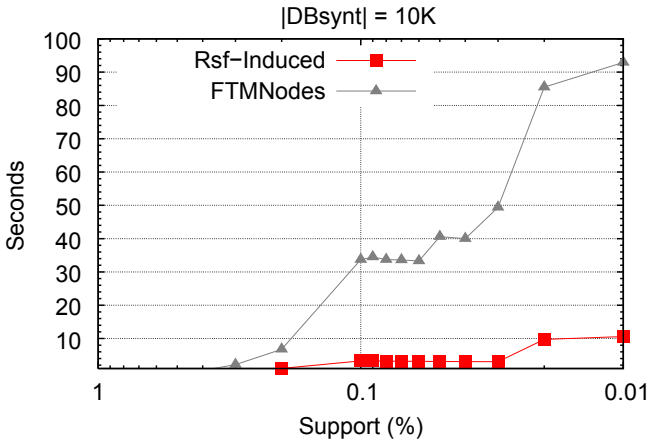

Fig. 18. Runtime w.r.t. support with $\tau = 0.8$.

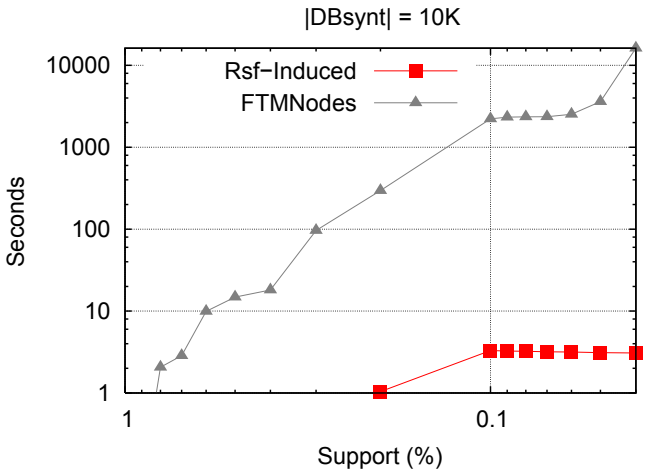

Fig. 19. Runtime w.r.t. support with $\tau = 0.7$.

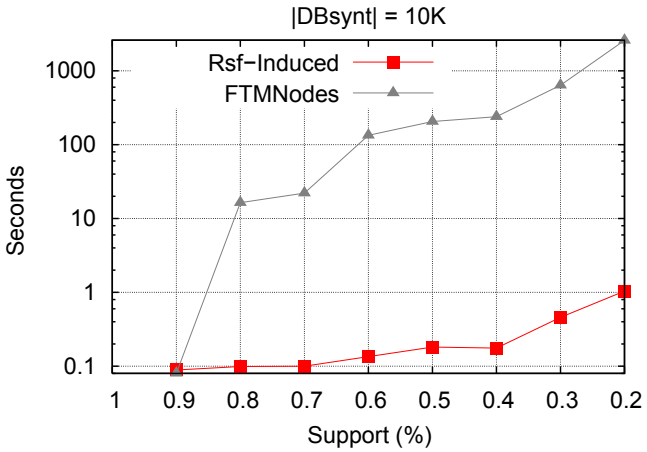

Fig. 20. Runtime w.r.t. support with $\tau = 0.6$.

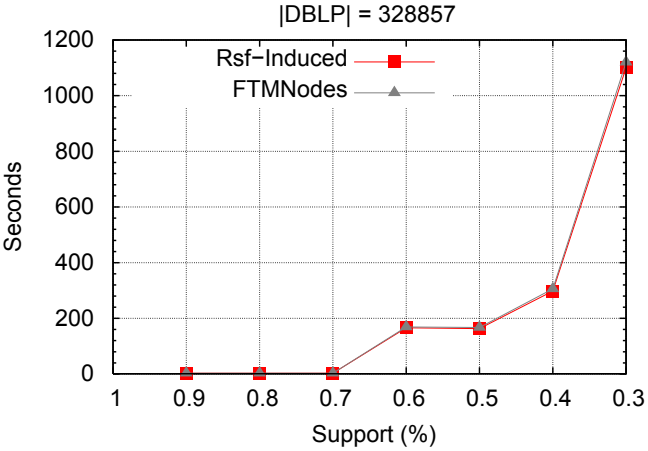

Fig. 21. Runtime w.r.t. support with $\tau = 1$.

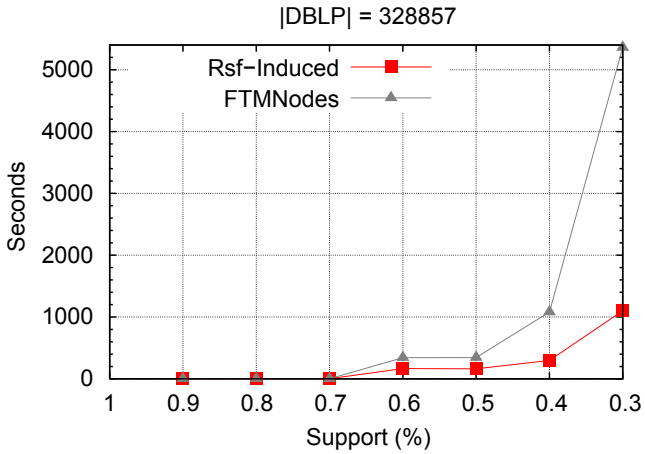

Fig. 22. Runtime w.r.t. support with $\tau = 0.9$.

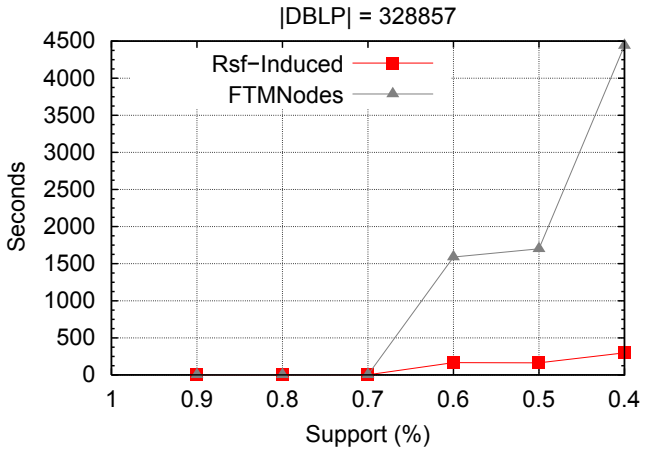

Fig. 23. Runtime w.r.t. support with $\tau = 0.8$.

5.2. Real databases

The real database we have experimented our methods on is the DBLP database [1]. This database stores bibliographic information about the articles and conference papers published in the field of computer science. It contains 328,458 objects.

Figs. 21–23 report the runtime with respect to the support value. Figs. 24–26 report the number of candidates generated with respect to the support value. Figs. 27–29 report the number of frequent subtrees extracted with respect to the support value.

These results confirm the ones obtained on synthetical databases, as we remained quite efficient and extracted more frequent subtrees are discovered.

6. 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)

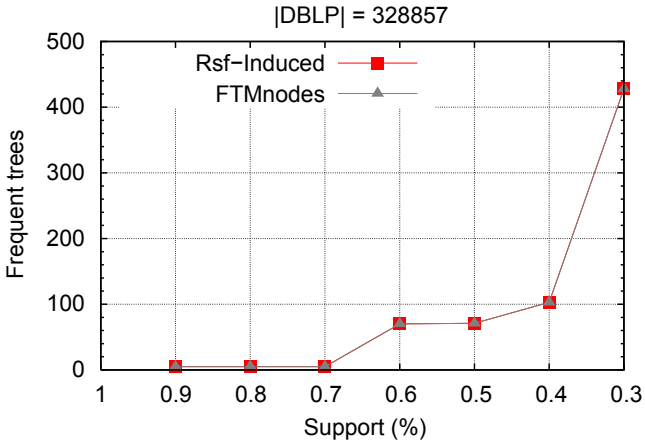

Fig. 24. Number of candidates w.r.t. support with $\tau = 1$.

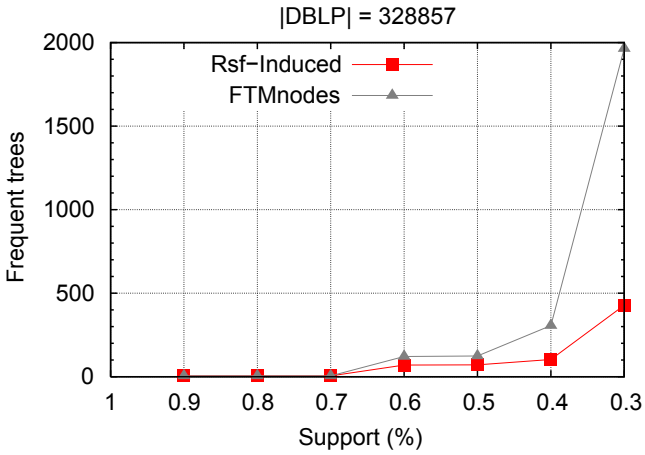

Fig. 25. Number of candidates w.r.t. support with $\tau = 0.9$.

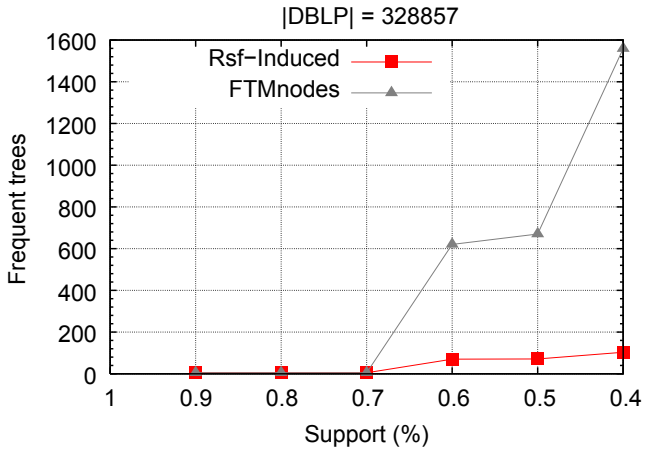

Fig. 26. Number of candidates w.r.t. support with $\tau = 0.8$.

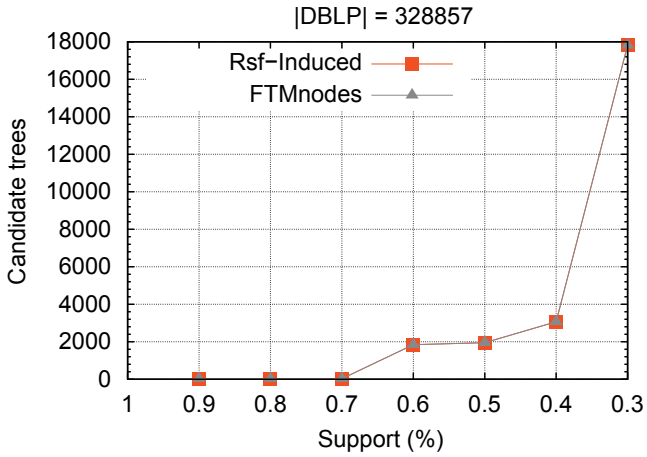

Fig. 27. Number of frequent trees w.r.t. support with $\tau = 1$.

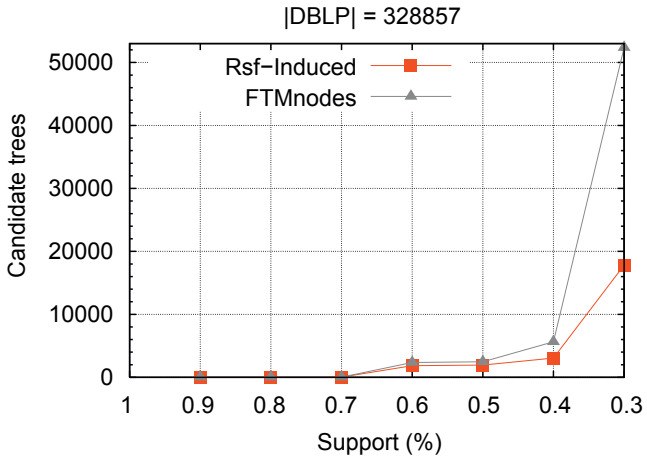

Fig. 28. Number of frequent trees w.r.t. support with $\tau = 0.9$.

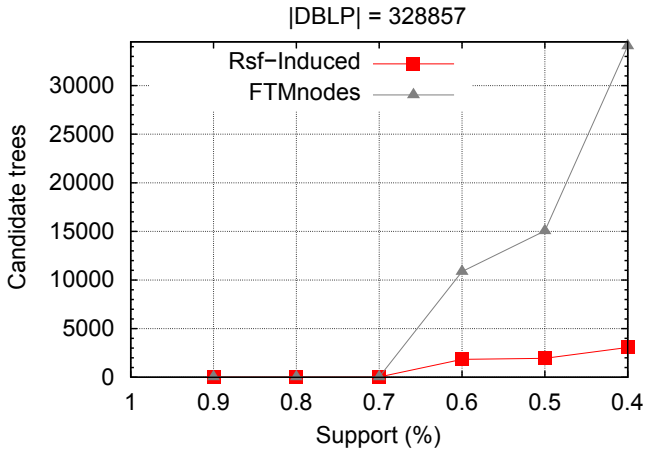

Fig. 29. Number of frequent trees w.r.t. support with $\tau = 0.8$.

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] Department of Computer Science & Engineering, University of Washington, Xml data repository, 2002, in ⟨http://www.cs.washington.edu/research/xmldatasets⟩.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, Efficient substructure discovery from large semi-structure data, in: Second Annual SIAM Symp. on Data Mining, SDM2002, Springer, Arlington, VA, USA, 2002.

[3] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: IEEE Internat. Conf. on Data Mining (ICDM), 2001.

[4] A. Termier, M.-C. Rousset, M. Sebag, Treefinder, a first step towards XML data mining, in: IEEE Conf. on Data Mining (ICDM), 2002, pp. 450–457.

[5] X. Yan, J. Han, gspan: graph-based substructure pattern mining, in: Proc. IEEE Conf. on Data Mining (ICDM), 2002.

[6] M.J. Zaki, Efficiently mining frequent trees in a forest, in: KDD'02, ACM, Edmonton, Alberta, Canada, 2002.

[7] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proc. 20th VLDB Conf., Santiago, Chile, 2002.

[8] A. Laurent, P. Poncelet, M. Teisseire, Fuzzy data mining for the semantic web: building XML mediator schemas, in: E. Sanchez (Ed.), Fuzzy Logic and the Semantic Web, Elsevier, Amsterdam, 2006, pp. 249–265.

[9] S. Sanchez, A. Laurent, P. Poncelet, M. Teisseire, Fuzbt: a binary approach for fuzzy tree mining, in: Proc. 11th IPMU Internat. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2006), 2006.

[10] F.D.R. López, A. Laurent, P. Poncelet, M. Teisseire, Fuzzy tree mining: go soft on your nodes, in: Proc. Internat. Fuzzy Systems Association World Congress (IFSA 07), Lecture Notes in Computer Science, Vol. 4529, Springer, Berlin, Heidelberg, 2007, pp. 145–154.

[11] Y. Chi, R.R. Muntz, S. Nijssen, J.N. Kok, Frequent subtree mining—an overview, Fundamenta Informaticae XXI (2005) 1001–1038.

[12] C. Wang, Q. Yuan, H. Zhou, W. Wang, B. Shi, Chopper: an efficient algorithm for tree mining, Journal of Computer Science and Technology 19 (2004) 309–319.

[13] Y. Chi, Y. Yang, R.R. Muntz, Indexing and mining free trees, in: Internat. Conf. on Data Mining 2003 (ICDM2003), 2003.

[14] Y. Chi, Y. Yang, R. Muntz, Cmtreeminer: mining both closed and maximal frequent subtrees, in: The Eighth Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'04), 2004.

[15] M.A. Weiss, Data Structures and Algorithm Analysis in C, Addison-Wesley, Reading, MA, 1998.

[16] C. Fiot, A. Laurent, M. Teisseire, From crispness to fuzziness: three algorithms for soft sequential pattern mining, IEEE Transactions on Fuzzy Systems 15 (6) (2007) 1263–1277.

[17] R. Yager, Families of owa operators, Fuzzy Sets and Systems 57 (3) (1993) 125–148.

[18] F. Del Razo, A. Laurent, P. Poncelet, M. Teisseire, Rsf—a new tree mining approach with an efficient data structure, in: Proc. Joint Conf.: Fourth Conf. of the European Society for Fuzzy Logic and Technology (EUSFLAT 2005), 2005, pp. 1088–1093.