Knowl Inf Syst
DOI 10.1007/s10115-015-0831-x

REGULAR PAPER

Frequent pattern mining in attributed trees: algorithms and applications

Claude Pasquier[1,2,3] · **Jérémy Sanhes**[3] · **Frédéric Flouvat**[3] · **Nazha Selmaoui-Folcher**[3]

Received: 30 March 2014 / Revised: 29 October 2014 / Accepted: 17 March 2015

Abstract Frequent pattern mining is an important data mining task with a broad range of applications. Initially focused on the discovery of frequent itemsets, studies were extended to mine structural forms like sequences, trees or graphs. In this paper, we introduce a new domain of patterns, attributed trees (atrees), and a method to extract these patterns in a forest of atrees. Attributed trees are trees in which vertices are associated with itemsets. Mining this type of patterns (called asubtrees), which combines tree mining and itemset mining, requires the exploration of a huge search space. To make our approach scalable, we investigate the mining of condensed representations. For attributed trees, the classical concept of closure involves both itemset closure and structural closure. We present three algorithms for mining all patterns, closed patterns w.r.t. itemsets (content) and/or structure in attributed trees. We show that, for low support values, mining content-closed attributed trees is a good compromise between non-redundancy of solutions and execution time.

Keywords Tree mining · Frequent pattern mining · Attributed tree · Condensed representation

1 Introduction

Frequent pattern mining plays an important and essential role in the field of data mining. These patterns allow to describe the data with association rules, correlations or other interesting relationships. In addition, they show their usefulness in classification tasks [13,15], indexing [36], clustering [28] and other data mining tasks. Initially focused on the discovery

✉ Claude Pasquier
claude.pasquier@unice.fr

1 University of Nice Sophia Antipolis, I3S, UMR 7271, 06900 Sophia Antipolis, France

2 CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France

3 Pôle Pluridisciplinaire de la Matière et de l'Environnement (PPME), University of New Caledonia, 98851 Nouméa, New Caledonia

Published online: 28 March 2015

of frequent itemsets [1], studies were extended to mine sequences [2] and structural forms like trees [9] or graphs [34]. While itemset mining seeks frequent combinations of items in a set of transactions, structural mining seeks frequent substructures. Most existing studies focus only on one kind of problem (itemset mining or structural mining). However, in order to represent richer information, it seems natural to consider itemsets that are organized in complex structures. In this paper, we are interested in discovering complex patterns in attributed trees that are tree structures in which each vertex is associated with an itemset. We introduce a new kind of pattern that captures both structural trends and attribute evolution. Combining structural enumeration with itemset enumeration is still an open problem.

For example, an epidemic of dengue fever is characterized by a set of interacting factors, causing the spread of the disease in space and time. In this situation, it is important to know which factors have an effect on dengue epidemiology, and how. Even if the global influence of environmental factors (water points, nearby mangrove, rainfall, humidity etc.) is known, the impact of all the factors together with their interactions is still an open problem. Indeed, the spread of a vector-borne disease in a city (e.g., the Dengue fever) can be represented in an attributed structure in which vertices represent city districts at a given timestamp, vertex attributes are characteristics of the districts and edges represent spatial or temporal neighboring between vertices.

Another illustration of the use of attributed trees can be found in Weblog analysis. In this kind of applications, it is common to represent user browsing in tree-like data where each page is identified with a unique id [39]. However, one can more pertinently characterize browsed pages with lists of keywords related to their content. This approach allows to capture the browsing habits of users even when the web site is reshuffled.

Other applications can be imagined in various areas such as retweet tree mining, spatio-temporal data mining, phylogenetic tree mining and XML document mining.

The key contributions of our work are the following:

1. We present the problem of mining ordered and unordered substructures in a collection of attributed trees.
2. We define canonical forms for attributed trees.
3. We propose a method for attributed tree enumeration that is based on two operations: itemset extension and tree extension.
4. We present an efficient algorithm IMIT for extracting frequent substructures in a set of attributed trees.
5. We perform extensive experiments on several synthetic and real datasets.

The rest of this paper is organized as follows. Section 2 presents basic concepts and defines the problem. Section 3 proposes a brief overview of related works, particularly a few studies that mix itemset mining and structure mining. Section 4 describes our proposed method to explore the search space, to compute frequency and to prune candidates. Section 5 reports several applications of the algorithms to mine both synthetic and real datasets. Finally, Sect. 6 concludes the paper and presents possible extensions of the current work.

2 Basic concepts and problem statement

In this section, we give basic concepts and definitions and then, we introduce the problem of attributed tree mining.

2.1 Preliminaries

Let $\mathcal{I} = \{i_1, i_2, .., i_n\}$ be a set of items. An **itemset** is a subset $\mathcal{P}$ of $\mathcal{I}$ ($\mathcal{P} \subseteq \mathcal{I}$). The size of an itemset is the number of items. The items belonging to an itemset are sorted by lexicographical order and are accessed by their index. The ith item of $\mathcal{P}$ is denoted by $\mathcal{P}[i]$.

A **tree** $S = (V, E)$ is a directed, acyclic and connected graph where V is a set of vertices (nodes) and $E = \{(u, v)|u, v \in V\}$ is a set of edges. A distinguished node $r \in V$ is considered as the root, and for any other node $x \in V$, there is a unique path from r to x. If there is a path from a vertex u to v in $S = (V, E)$, then u is called an *ancestor* of v (v is called a *descendant* of u). If $(u, v) \in E$ (i.e., u is an immediate ancestor of v), then u is the *parent* of v (v is a *child* of u). An ordered tree has a left-to-right ordering among the siblings. In this paper, we consider that all trees are unordered unless otherwise specified.

An **attributed tree**, or (**atree**) is a triple $T = (V, E, \lambda)$ where (V, E) is the underlying tree and $\lambda : V \rightarrow pow(\mathcal{I})$ is a function which associates an itemset $\lambda(v) \subseteq \mathcal{I}$ to each vertex $v \in V$. $pow(\mathcal{I})$ denotes the power set of $\mathcal{I}$. The size of an attributed tree is the number of items associated with its vertices. A k-atree is an atree of size k.

Attributed trees can be seen as itemsets organized in a tree structure. As such, **attributed tree inclusion** can be defined w.r.t. itemset inclusion and structural inclusion. For itemset inclusion, we say that atree T_1 is contained in atree T_2 if both atrees have the same structure and for each vertex of T_1, the associated itemset is contained in the itemset of the corresponding vertex in T_2.

Definition 1 *(Itemset/content inclusion)* $T_1 = (V_1, E_1, \lambda_1)$ is **contained in** $T_2 = (V_2, E_2, \lambda_2)$, and is denoted by $T_1 \sqsubset_I T_2$, *if* $V_1 = V_2$ and $E_1 = E_2$ and $\forall x \in V_1, \lambda_1(x) \subseteq \lambda_2(x)$.

Structural inclusion is represented by the classical concept of subtree [6,9,17,25,31,35,37]. Thus, we generalize the notion of asubtree in the following way to include both content and structural inclusion.

Definition 2 *(Structural and content inclusion)* $T_1 = (V_1, E_1, \lambda_1)$ is an **asubtree** of an atree $T_2 = (V_2, E_2, \lambda_2)$ and is denoted $T_1 \sqsubset T_2$ *if* T_1 is an isomorphic asubtree of T_2, i.e., there exists a mapping φ: $V_1 \rightarrow V_2$ such that (i) $T_1 \neq T_2$, (ii) $\forall(u, v) \in E_1, \exists(\varphi(u), \varphi(v)) \in E_2$ and (iii) $\forall x \in V_1, \lambda_1(x) \subseteq \lambda_2(\varphi(x))$.

If T_1 is an asubtree of T_2, we say that T_2 is an **asupertree** of T_1. The two kinds of inclusion are illustrated in Fig. 1.

Three subtypes of inclusion relationships are usually studied in tree mining: induced, embedded and embedded with gaps.

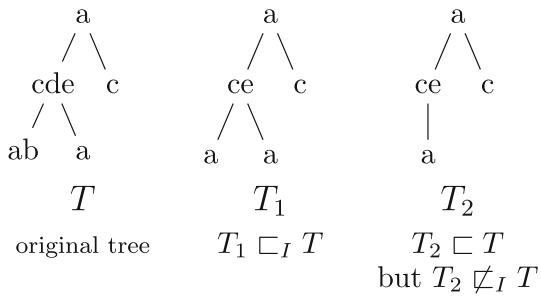


Fig. 1 Illustation of the different types of inclusion

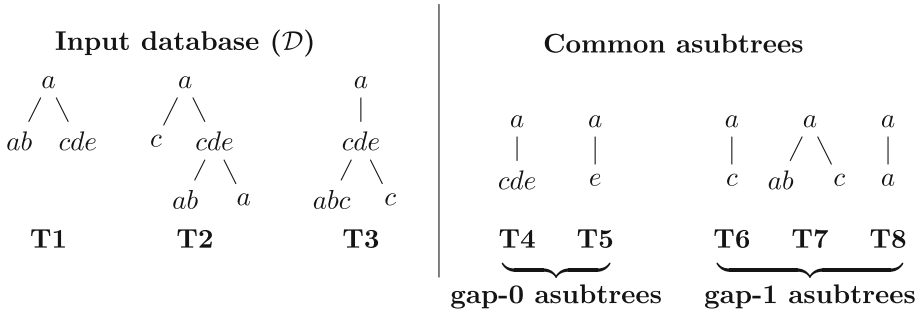


Fig. 2 Example of an atree database with some common asubtrees

Definition 3 *(Induced asubtrees)* T_1 is called an **induced asubtree** of T_2 *iff* T_1 is an isomorphic asubtree of T_2 and φ preserves the parent–child relationships.

Definition 4 *(Embedded asubtrees)* T_1 is called an **embedded asubtree** of T_2 *iff* T_1 is an isomorphic asubtree of T_2 and φ preserves the ancestor-descendant relationships.

Definition 5 *(Gap-i asubtrees)* $T_1 = (V_1, E_1, \lambda_1)$ is called a **gap-i asubtree** of $T_2 = (V_2, E_2, \lambda_2)$ *iff* T_1 is an isomorphic asubtree of T_2 and φ preserves the ancestor-descendant relationships with the following constraint: $\forall(u, v) \in E_1, d(\varphi(u), \varphi(v)) \leq i+1$ in T_2, where $d(x, y)$ represents the number of edges between x and y in the atree.

A gap-0 asubtree is synonym to induced asubtree as it designates an asubtree where each descendant is separated from its ancestor by a distance of 1, i.e., each descendant is a child. A gap-free asubtree is equivalent to an embedded asubtree.

Figure 2 shows an example of an atree database ($\mathcal{D}$) composed of three different atrees with two (incomplete) sets of common asubtrees using maximum gaps of 0 and 1.

2.2 Problem statement

2.2.1 Frequent atree mining

Given a database $\mathcal{B}$ of atrees and an atree T, the **per-tree frequency** of T is defined as the number of asupertrees of T in $\mathcal{B}$. An atree is frequent if its per-tree support is greater than or equal to a minimum threshold value. The problem consists in enumerating all frequent patterns in a given forest of atrees.

2.2.2 Mining closed atrees

The problem with frequent atree mining is that the number of frequent patterns is often huge. In real applications, generating all solutions can be very expensive or even impossible. Moreover, many of those frequent atrees contain redundant information. In Fig. 2, for example, the atree $T5$ is present in all input atrees, but this pattern is already contained in the atree $T4$. The atree $T8$ is also redundant w.r.t. $T7$ since it is an asubtree of this pattern and it occurs in the same input atrees.

Since the proposal of Mannila and Toivonen [21], huge efforts have been made to design condensed representations that are able to summarize solutions in smaller sets. A set of closed

patterns is an example of such a condensed representation [27]. We say that an atree T is a **closed atree** if none of its proper asupertrees has the same support as T.

However, mining frequent closed atrees could also be very expensive. We propose then to relax the closure property on atrees and to mine only atrees which are closed w.r.t. their content (i.e., itemsets associated with vertices). This condensed representation, called the set of **c-closed atrees** (content closed), is a superset of the set of closed atrees (and a subset of the set of atrees).

Definition 6 We say that an atree T is a **c-closed atree** if there is no pattern with the same structure, the same support, and the same supersets in the corresponding vertices ("contained in" relationship previously defined).

3 Related works

Most of the earlier frequent tree mining algorithms are derived from the well-known Apriori strategy [1]: a succession of candidate generation and pruning (in which support computation is done) steps. Two strategies are possible for candidate generation: extension and join. With extension, a new candidate tree is generated by adding a node to a frequent tree [3,25]. With join, a new candidate is created by combining two frequent trees [17,38]. Combination of the two principles has also been studied [11].

The extension principle is a simple method suitable to mine induced trees because the number of nodes that can be used to extend a given subtree is often lower than the number of frequent subtrees.

Other tree mining algorithms are derived from the FP-growth approach [16]. These algorithms adopt the divide-and-conquer pattern-growth principle, which avoids the costly process of candidate generation. However, pattern-growth approach cannot be extended simply to tackle the frequent tree pattern mining problem. Existing implementations are limited w.r.t. the type of trees they can handle: induced unordered trees with no duplicate labels in each node's children [35], ordered trees [33] or embedded ordered trees [40] are some kind of trees that were successfully mined with pattern-growth approaches.

Finding condensed representations of frequent patterns is a natural extension of frequent tree mining. For itemset mining, the notion of closure is a classical property [27]. Several works have explored this topic in the context of tree mining and have proposed mining methods as well as various implementations [12,31,32]. To the best of our knowledge, no method has been proposed for the general case of attributed trees.

Recently, interest has grown in mining itemsets organized in structures. Several studies [14,23,24] deal with attributed graphs. However, they focus on a specific problem dedicated to the finding of dense subgraphs with vertices sharing similar features. In our work, we are not concerned with the density of the subgraphs. We adhere to the domain of frequent subgraph mining (FSM), as defined by [18], whose objective is to extract all the frequent subgraphs in a given data set.

Closest to our work, Miyoshi et al. [22] study the mining of labeled graphs with quantitative attributes associated with vertices. However, the kind of structure they are using allows to solve the problem by combining a "classical" subgraph mining algorithm for the labeled graph with an existing itemset mining algorithm for quantitative itemsets in each vertex.

We have developed recently several algorithms to mine induced and embedded attributed trees [26]. The current paper describes in more details the method with some improvments (such as, for example the processing of gap-i asubtrees) and present various applications of the algorithm.

4 Mining frequent atrees

We are mainly interested in identifying induced ordered and unordered asubtrees. Depending on applications, some patterns including gaps in the ancestor–descendant relationship can also be considered. However, in order to collect only interesting patterns, the gap used should remain small. Otherwise, the relationship between a node and its descendants is not really meaningful. Although we focus on induced asubtrees, we design a general method that can mine asubtrees with any gap value, including embedded asubtrees. However, because of our primary objective, our method works better for induced asubtree mining and performances decrease as gap parameter increases.

4.1 Theoretical framework

4.1.1 String encoding for attributed trees

Choosing an appropriate pattern encoding has an important impact on the performance of the mining process. Many algorithms represent trees with preorder strings [10,11,19,20,39] or equivalent encodings like depth sequences [4,25]. In this paper, we extend the string encoding defined by Zaki [38] for labeled trees to the case of attributed trees.

The string encoding for an atree T is generated by enumerating the nodes of T in a depth-first preorder traversal and adding a special symbol \$ when backtracking from a child to its direct parent. The string representation of a node is generated by listing all the items present in the associated itemset in lexicographical order. For example, the string representation of atree T_2 in Fig. 2 is "$a\ c\ \$\ cde\ ab\ \$\ a$." In the paper, for simplicity, we omit the last backtracks (i.e., \$).

4.1.2 Canonical atrees

All tree mining algorithms dealing with unordered trees have to face the isomorphism problem. To avoid the redundant generation of equivalent solutions, one tree is chosen as the canonical form and other alternative forms are discarded [3,11,25,35,38]. For labeled trees, canonical forms are based on a lexicographical order on node labels. In our work, we define a new order based on node associated itemsets.

Definition 7 Given two itemsets $\mathcal{P}$ and $\mathcal{Q}$, we say that $\mathcal{P} \leq \mathcal{Q}$ iff:

1. $\forall i \in [1, \min(|\mathcal{P}|, |\mathcal{Q}|)]: \mathcal{P}[i] \leq \mathcal{Q}[i]$, and
2. if $\forall i \in [1, \min(|\mathcal{P}|, |\mathcal{Q}|)]: \mathcal{P}[i] = \mathcal{Q}[i]$, then $|\mathcal{P}| \geq |\mathcal{Q}|$.

From the definition above, an order among atrees can be defined recursively.

Definition 8 Let X and Y be two attributed trees, r_x and r_y denote the root nodes of X and Y and let $c_1^{r_x}, \ldots, c_m^{r_x}$ and $c_1^{r_y}, \ldots, c_n^{r_y}$ denote the ordered list of subtrees of r_x and r_y, respectively. Then, $X \leq Y$ iff either:

1. $\lambda(r_x) < \lambda(r_y)$, or
2. $\lambda(r_x) = \lambda(r_y)$ and either:
   (a) $m > n$ and $\forall i \in [1, n]$, $c_i^{r_x} = c_i^{r_y}$ (i.e., Y is a prefix of X), or
   (b) $\exists j \in [1, min(m, n)]$, such that $c_j^{r_x} < c_j^{r_y}$ and $c_i^{r_x} = c_i^{r_y}$ for all $i < j$.

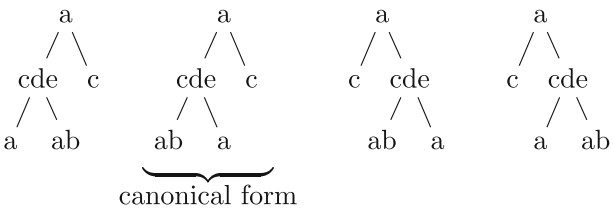


Fig. 3 Canonical form of isomorphic attributed trees

With this order, the smallest attributed tree in a set of isomorphic atrees is choosen as the **canonical form** [9].

Figure 3 shows examples of isomorphic trees. The second attributed tree is choosen as the canonical form because its encoding, namely "*a cde ab* $ *a* $ $ *c*," is the smallest one.

4.2 Atree enumeration

Rymon's generic set-enumeration tree is often used to illustrate how itemsets are (completely) enumerated by itemset mining algorithms [7,29]. An enumeration tree is generated in the following way: The root node of the tree is at the top level and labeled with $\emptyset$ (i.e., empty atree). Then, for each node, children (candidate atrees) are generated either by **tree extension** or by **itemset extension**. Note that frequency computation is done during candidate generation and thus does not require to access the input atree database (see Sect. 4.3).

4.2.1 Tree extension

For tree extension, we use a variation of the well-known rightmost path extension method [3,25]. Let T be an atree of size k. T can be extended to generate new atrees in two different ways. In the first way, a new child node N is added to the rightmost node of T (**right-node extension**). In the second way, a new sibling N is added to a node in the rightmost path of T (**right-path extension**) [8]. In both cases, N becomes the rightmost node of the generated $k+1$-atree.

In the existing approaches, the extension node N represents every valid node from the input database. In our approach, a new extension node N can be created from every item occurring in a valid node Q of the input database. In other words, given a valid node Q from the input database, we can extend the atree T with every node of $\{N_i, i \in [1, |\lambda(Q)|] \mid \lambda(N_i) = \lambda(Q)[i]\}$.

4.2.2 Right-node extension

For right-node extension, we use a classical approach, which has been shown to be complete as well as non-redundant [3,4].

Figure 4 shows an attributed tree with one of its embedded attributed subtree T. T is represented in canonical form: "*a b* $ *c ae*." From the candidate T, right-node extension is used to generate new candidates by adding children to its rightmost node. Two nodes, in the input tree, can be used to perform the extension: node "*ab*" and node "f." Using the method described previously, node "*ab*" is used to extend T with nodes "*a*" and "*b*," while node "f" allows to extend T with node "f." Right-node extension leads to the generation of three new candidates: "*a b* $ *c ae a*," "*a b* $ *c ae b*" and "*a b* $ *c ae* f." Figure 5 shows the attributed trees obtained by performing a right-node extension on pattern T from Fig. 4.

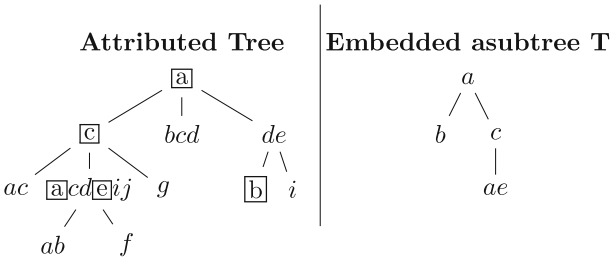


Fig. 4 Example of an attributed tree and one of its embedded attributed subtree T displayed in canonical form. Framed elements highlight the position of T in the attributed tree on the *left*

Fig. 5 Attributed trees generated by right-node extension of pattern T from Fig. 4

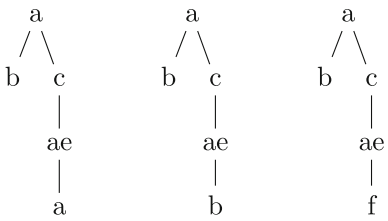
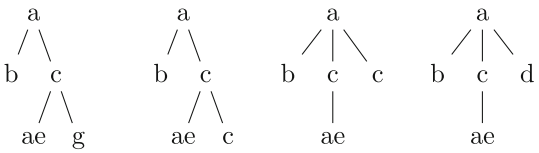


Fig. 6 Attributed trees generated by right-path extension of pattern T from Fig. 4



4.2.3 Right-path extension

Right-path extension in ordered atree mining. For ordered tree mining, candidate nodes are added in the order of appearance in the initial trees. This is a commonly used approach (detailed in [3]) which is complete and non-redundant.

Right-path extension in unordered atree mining. The way to perform right-path extension in unordered mining differs from previous approaches. Nodes that can be used for right path extension of an unordered candidate atrees are those greater or equal than any of their siblings. As candidate atrees subject to extension are in canonical form, the test has only to be done against nodes in the right-hand path.

In Fig. 4, right-path extension is used to add new siblings after nodes belonging to the right-hand path of T, namely node "ae" (by adding a new child to node c) and node "c" (by adding a new child to node a). Nodes that can be used to generate siblings of node "ae" are nodes "ac" and "g." Node "g" is used to right path extend T with a new node "g" and node "ac" allows to extend T with node "c" which is the only node that is greater than "ce" w.r.t. the order previously defined (sect. 2). This operation generates two candidate atrees: "$a\ b\ \$\ c\ ae\ \$\ g$" and "$a\ b\ \$\ c\ ae\ \$\ c$." The only node that can be used to generate siblings of node "c" is node "bcd." Node "de" is not a valid candidate as it is an ancestor of node "b" already present in the candidate atree. Node "bcd" is used to right path extend T with nodes "c" and "d" (adding node "b" is not an option because $b < c$). This generates two new attributed trees: "$a\ b\ \$\ c\ ae\ \$\ \$\ c$" and "$a\ b\ \$\ c\ ae\ \$\ \$\ d$." Figure 6 illustrates the attributed trees generated by performing right-path extensions on pattern T from Fig. 4.

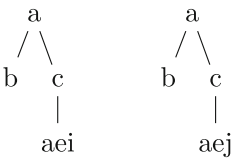


Fig. 7 Attributed trees generated by itemset extension of pattern T from Fig. 4

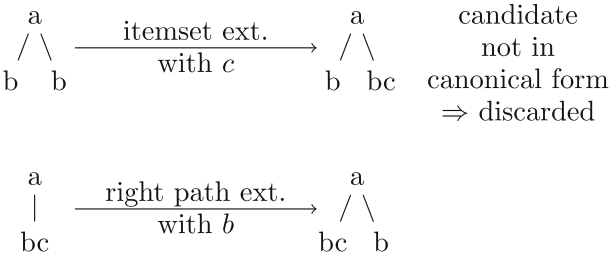


Fig. 8 Illustration of the generation of two isomorphic attributed trees by using different extension paths

This method can generate redundant patterns in the form of isomorphic attributed trees. Duplicate candidates are detected and discarded before the candidate extension process by performing a canonical check. Note that because of the way nodes are selected for tree extension, candidates are mostly in canonical form. Using the node ordering instead of the order of appearance of nodes does not alter the completeness of the atree enumeration process.

4.2.4 Itemset extension

Our itemset extension method is a variation of the method presented by Ayres et al. [5]. In this variation, a new item I is added to the itemset associated with the rightmost node of the candidate atree T. Items used for itemset extension are derived from the itemsets associated with this node in the input database. The constraint is that the new item i must be greater than any item associated with the rightmost node of T. After the extension, i (the last item of the itemset associated with the righmost node of the generated $k+1$-atree) becomes the rightmost element of T.

For example, in Fig. 4, the items that can be used for itemset extension of T are "i'' and "j." This operation generates two new candidates atrees: "*a b $ c aei*" and "*a b $ c aej*." Figure 7 shows the attributed trees obtained by performing itemset extensions on pattern T from Fig. 4.

4.2.5 Filtering non-canonical patterns

In the case of unordered atree mining, the combination of right-path extension and itemset extension may lead to the generation of redundant patterns. For example, two isomorphic attributed trees for "*a bc $ b*" are generated either by adding item "*c*" to the rightmost node of "*a b $ b*" or by adding node "*b*" as a sibbling of node "*bc*" in "*a bc*." As explained before, candidates that are not in canonical forms (in our example, "*a b $ bc*") are discarded (see Fig. 8).

4.2.6 Completeness of atree enumeration

Proposition 1 *If the set of candidates of size $|T|$ is complete, then the set of candidates of size $|T|+1$ is also complete.*

Proof Let $\mathcal{T}$ be the set of candidates obtained by extending a candidate atree T using right-node extension, right-path extension or itemset extension. Every $T' \in \mathcal{T}$, of size $|T|+1$ differs from T only by its last element (w.r.t. the canonical form). T is the prefix of T'. This last element is either a node associated with an item added to the end of the candidate tree, or an item added to the itemset associated with the rightmost node of the candidate tree. As there is no other possibility to extend T, $\mathcal{T}$ represents the complete set of candidates of size $|T|+1$ having T as prefix. We can easily deduce that, from a complete set of candidates of size $|T|$, it is possible, by applying all possible extensions, to generate a set of candidates of size $|T|+1$ which is also complete. □

Candidates at depth 1 of the enumeration tree (representing the search space) are atrees of size 1, i.e., atrees with one node associated with an itemset of size 1 (i.e., an item). The set of size-1 candidates is built by scanning the input database and collecting all items. This set is complete and our previous proposition states that every set of candidates of size i is also complete (by mathematical induction).

For unordered tree mining, we stop the extension of candidates that are not in canonical forms. To prove the completeness of the enumeration, we need to be certain that every candidate in canonical form can be obtained by the extension of a candidate which is also in canonical form.

Proposition 2 *If T is in canonical form, then every prefix of T is also in canonical form.*

Proof The proposition has been proved in the case of labeled trees [25]. Thus, for attributed trees, the proposition is true if the last element of T is a node associated with an itemset of size 1. Suppose now that the last element of T is an itemset. Let T be a candidate in canonical form and x be its last element (the last item in the itemset associated with the rightmost node of T). Removing x from T allows to obtain candidate U which is a prefix of T. We have $\lambda(T) = \lambda(U) \cup \{x\}$, i.e., $\lambda(U)$ is a prefix of $\lambda(T)$. With the order previously defined (see Sect. 4.1.2), $\lambda(U) > \lambda(T)$ (case 2 of the definition). Then, the order of nodes in the prefix tree is not changed and the prefix is also in canonical form. □

4.3 Frequency computation

A data structure is used to store all information needed for the mining process. Our structure is an extension of the vertical representation of trees introduced by Zaki [37,38]. Briefly, each candidate asubtree is associated with its pattern and several data allowing to pinpoint all its occurrences in the database. The first candidates, composed of a unique node associated with one item, are generated by scanning the input database. Using only this unique structure, it is easy to compute the number of occurrences of each pattern. In addition, this same structure is sufficient to generate all possible extensions of a given pattern. When a pattern of size k is processed, all occurrences are extended with tree extension and itemset extension methods described above to generate new $(k+1)$-candidates that are themselves stored in the structure.

4.4 Candidate pruning

Rules as the ones specified by Agrawal et al. [1] 20 years ago, can be applied to the case of atrees: (i) any sub-pattern of a frequent pattern is frequent, and (ii) any super-pattern of

$IMIT(\mathcal{D}, minSup)$
1: $\mathcal{C} \leftarrow \{all\ asubtrees\ of\ size\ 1\ in\ \mathcal{D}\}$
2: **while** $\mathcal{C} \neq \emptyset$ **do**
3: $T \leftarrow getFirst(\mathcal{C})$
4: **if** $isCanonical(T)$ **and** $f_t(T) \geq minSup$ **then**
5: $\mathcal{X} \leftarrow extend(T)$
6: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{X}$
7: $\mathcal{S} \leftarrow \mathcal{S} \cup \{T\}$
8: **end if**
9: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{T\}$
10: **end while**
11: $printSolutions(\mathcal{S})$

Fig. 9 IMIT algorithm

an infrequent pattern is non-frequent. As the frequency count is an anti-monotonic function (extending a pattern cannot lead to a new pattern with a greater frequency), it is possible to stop the exploration of a branch of the search space when the frequency of a candidate is less than the minimum support. For example, in Fig. 2, when we examined pattern "*a c a*" and found that its frequency was lower than the minimum support, we did not generate candidates obtained by extending "*a c a*" (e.g., "*a c ab*," "*a c a* $ *b*," "*a c a* $ $ *c*").

In addition, in the case of unordered tree mining, extension of a candidate is stopped if it is not in canonical form.

4.5 Mining algorithms

We have developed and evaluated three different algorithms. IMIT is used to enumerate all frequent attributed subtrees, while the algorithms IMIT_CLOSED and IMIT_CONTENT_CLOSED are designed to mine closed atrees and c-closed atrees, respectively.

4.5.1 IMIT

Figure 9 shows the high level structure of the IMIT algorithm. The algorithm takes in parameter a database ($\mathcal{D}$) and a minimum support ($minSup$). First (line 1), a set $\mathcal{C}$ containing all asubtrees of size 1 is built by scanning $\mathcal{D}$. Then, a loop allows to process every candidate in the set. The function $GetFirst$ returns the smallest candidate in the set according to the order relation defined in Sect. 4.1.2. In line 4, the algorithm does a canonical test (function $isCanonical$) and a frequency test (f_t is a function that returns the per-tree frequency of an atree). A frequent candidate (which is in canonical form) is added to $\mathcal{S}$, the list of solutions (line 7). The frequent candidate is also extended (line 5) to generate new candidates in the next iteration (line 6). The processing of a candidate finishes by removing it from the candidate list (line 9).

This algorithm is sufficient to enumerate all solutions, but it has a huge search space.

4.5.2 IMIT_CLOSED

Only enumerating closed atrees (i.e., atrees that are not attributed subtrees of another atree with the same support) drastically reduces the search space. It involves the storage of every frequent pattern found with their associated per-tree frequency and their total number of

```
IMIT_CLOSED(D, minSup)
 1: C ← {all asubtrees of size 1 in D}
 2: while C ≠ ∅ do
 3:   T ← getFirst(C)
 4:   if isCanonical(T) and f_t(T) ≥ minSup then
 5:     X ← extend(T)
 6:     if ∄T' ∈ X : T ⊏ T' and f_o(T') = f_o(T) then
 7:       C ← C ∪ X
 8:     end if
 9:     if ∄T' ∈ S ∪ X : T ⊏ T' and f_t(T') = f_t(T) then
10:       S ← S ∪ {T}
11:     end if
12:     for all T' ∈ S such that T' ⊏ T and f_t(T') = f_t(T') do
13:       S ← S \ {T'}
14:     end for
15:   end if
16:   C ← C \ {T}
17: end while
18: printSolutions(S)
```

Fig. 10 IMIT_CLOSED algorithm

occurrences in the database (the **occurrence-match frequency**). We define a new functions: f_o which returns the occurrence-match frequency of an atree.

Definition 9 *(Closure test)* Let T be the candidate atree currently processed, $\mathcal{S}$ be the set of all previously identified frequent atrees and $\mathcal{X}$ be the set of candidates generated by extension of T. We say that T is a closed atree if $\nexists T' \in \mathcal{S}U\mathcal{X}$ such that $T \sqsubset T'$ and $f_t(T') = f_t(T)$.

However, finding an itemset extension of T with the same per-tree frequency as T does not allow to stop the exploration of other candidates in $\mathcal{X}$. The following additional conditions must also be satisfied:

Definition 10 *(Pruning test)* Let T be the candidate atree currently processed, $\mathcal{S}$ be the set of all previously identified frequent atrees and $\mathcal{X}$ be the set of candidates generated by extension of T. If $\exists T' \in \mathcal{X} : T \sqsubset T'$ and $f_o(T') = f_o(T)$, then T can be pruned from $\mathcal{X}$.

In Fig. 2, for example, the first candidate to be examined is "a" with a per-tree frequency of 3. By extension, we generate, among others, candidate "ab" which has a per-tree frequency of 3, therefore, candidate "a" is not c-closed as "a" $\sqsubset$ "ab." However, pattern "a appears seven times in the database, while the total number of occurrences of candidate "ab" is 3. The four times where "a" occurs in an itemset which does not contain "b" may lead to the generation of other closed patterns. This is the case in Fig. 2 where a right-node extension of pattern "a" generates candidate "$a\ e$" with a per-tree frequency of 3.

The IMIT_CLOSED algorithm, which is used to mine closed asubtrees is presented in Fig. 10. The structure of the algorithm is similar to IMIT's structure. The instructions at the beginning (first 5 lines) and at the end (last 4 lines) are identical. The insertion of the set of extensions $\mathcal{X}$ in the set of candidates $\mathcal{C}$ remains (line 7), but in IMIT_CLOSED, this operation is only performed if the test allowing to end the exploration of $\mathcal{X}$ ($\mathcal{X}$ pruning test) fails. The insertion of the current candidate T in the set of solutions is performed (line 10) only if the candidate is canonical (success of canonical test). The new loop in lines 12–14 is added to remove from the list of solutions the attributed trees that are not closed anymore (i.e., all atrees that are asubtrees of T with the same per-tree frequency).

$IMIT\_CONTENT\_CLOSED(\mathcal{D}, minSup)$
1: $\mathcal{C} \leftarrow \{all\ asubtrees\ of\ size\ 1\ in\ \mathcal{D}\}$
2: **while** $\mathcal{C} \neq \emptyset$ **do**
3: $T \leftarrow getFirst(\mathcal{C})$
4: **if** $isCanonical(T)$ **and** $f_t(T) \geq minSup$ **then**
5: $\mathcal{X}_I \leftarrow extend_I(T)$; $\mathcal{X}_S \leftarrow extend_S(T)$
6: **if** $\nexists T' \in \mathcal{X}_\mathcal{I} : T \sqsubset_I T'$ **and** $f_o(T') = f_o(T)$ **then**
7: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{X}_\mathcal{I} \cup \mathcal{X}_\mathcal{S}$
8: **end if**
9: **if** $\nexists T' \in \mathcal{S} \cup \mathcal{X}_I : T \sqsubset_I T'$ **and** $f_t(T') = f_t(T)$ **then**
10: $\mathcal{S} \leftarrow \mathcal{S} \cup \{T\}$
11: **end if**
12: **end if**
13: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{T\}$
14: **end while**
15: $printSolutions(\mathcal{S})$

Fig. 11 IMIT_CONTENT_CLOSED algorithm

These added lines require one to perform several subtree isomorphism checks that are costly operations.

As illustrated in Sect. 5, the algorithm is costly when the set of results is large and it cannot be used to mine large input databases with low support.

4.5.3 IMIT_CONTENT_CLOSED

For this algorithm, we introduce two subsets of $\mathcal{X}$. $\mathcal{X}_I$ is the set of atrees generated by itemset extension of T and $\mathcal{X}_T$ is composed of tree extensions of T. These sets are obtained by applying the functions $extend_I$ and $extend_S$, respectively.

Definition 11 T is a c-closed atree if $\nexists T' \in \mathcal{S}U\mathcal{X}_I$ such that $T \sqsubset_I T'$ and $f_t(T') = f_t(T)$.

The extension of T can be ended if $\exists T' \in \mathcal{X}_\mathcal{I} : T \sqsubset T'$ and $f_o(T') = f_o(T)$.

The IMIT_CONTENT_CLOSED algorithm for mining c-closed asubtrees is presented in Fig. 11. This algorithm differs from IMIT_CLOSED algorithm in the following way:

- In lines 6 and 9, $\sqsubset$ is replaced by $\sqsubset_I$ and $\mathcal{X}$ by $\mathcal{X}_I$. This allows to only perform itemset inclusion tests that are less costly than subtree isomorphism checks.
- The search in the set of solutions previously found (those that are asubtree of the current candidate) is not needed anymore for the extraction of c-closed patterns.

Experiments presented in Sect. 5 show that this third algorithm is a good compromise between non-redundancy of solutions and execution time.

5 Experimental results

Our algorithms were implemented in C++ using the Standard Template Library (STL). Experiments were performed on a computer with an Intel © Core™ i5-2400 @ 3.10 GHz and 16 Gb main memory. Performance evaluation was based on total execution time, including all preprocessing and result output.

Multiple datasets were used to test our algorithms: several synthetic datasets and two real databases.

5.1 Synthetic datasets

We modified the synthetic data generation program proposed by Zaki [37] for labeled trees in order to generate atrees with itemsets of different sizes. We added two new parameters controlling the minimum and maximum itemset sizes.

From the D10 dataset presented in [37], we generated nine datasets by varying the size of the itemsets from two to ten. We named these dataset $D10 - x$, where x represents the itemset size. Thus, in $D10 - 5$, the topologies of attributed trees are similar as those in the D10 dataset, but every node is associated with an itemset of size 5.

The comparison with existing work on labeled tree mining was performed using the D10 dataset (100,000 subtrees), the T1M dataset (1,000,000 subtrees) defined by Zaki [37] and a new dataset that we called T10M with a number of subtrees T sets to 10,000,000.

5.2 Weblog dataset

We built a dataset based on logs given by our university following the method described by Zaki [37]. However, instead of labeling nodes with URLs of the browsed pages, we associated them with itemsets representing keywords of their content. For each visited page, we collected the ten most frequent words (except common words like prepositions) and used them as items characterizing the page. Finally, we obtained a set of attributed trees whose nodes identify visited URLs; their itemsets are keywords associated with pages, and edges represent the ancestor-descendant relationships on the website tree structure. The dataset is composed of 126,396 attributed trees with itemsets of size 10 (10 keywords by page).

5.3 Dengue dataset

Dengue is an infectious tropical disease. The virus is transmitted to humans by the bite of an infected mosquito. The Aedes aegypti mosquito is the most important transmitter or vector of the dengue virus.

Our real dataset represents the weekly evolution of dengue infection in Nouméa and suburbs from February 1998 to September 2004. The city is divided into 32 districts, and each district is characterized by 15 epidemic and environmental attributes (e.g., number of dengue cases, temperature, precipitation, wind speed and Aedes density level).

Attributes that are normally distributed are discretized into three classes:

- One class, labeled as "NORMAL," gathers normal values representing 60 % of all measures centered around the mean,
- One class, labeled as "LOW," includes the smallest values representing 20 % of all measures,
- One class, labeled as "HIGH," includes the largest values representing 20 % of all measures.

The attributes that are discretized by this way are Breteau Indice (number of positive containers per 100 houses inspected), IPA indice (number of A. aegypti in late pre-imaginal stages per inspected house), House (premise) index (percentage of houses infested with A. aegypti larvae and/or pupae), IHRE (number of A. aegypti in late pre-imaginal stages per infested house), precipitation (the level of rainfall in milimeters), average and maximum wind speed, minimum, average and maximum temperature, average humidity, atmospheric pressure and solar irradiance (in J/cm^2 between 2 AM and 2 PM and between 2 PM and 2 AM).

The only exception is the attribute maximum wind direction that is discretized in NORTH, EAST, SOUTH and WEST.

From this dataset, we incrementally constructed a data tree forest, where the vertices of the trees represented the evolution of neighboring areas in space and time. The principle of our approach is detailed in [30].

We were interested in factors that could highlight the beginnings of an epidemic phase. As a consequence, we decided that, for every time point, each district in which four or more cases of dengue had been reported, represented the root of an attributed tree. Each root node was associated with a set of items which represented the characteristics of an "infected" zone at a given time point. These trees composed of only one node were then extended to reflect the evolution of neighboring zones before the report of dengue cases in the "infected" zone (represented by the root of the tree). We recursively generated for each leaf of a tree (that contained the characteristics of a zone at a time point t), several edges that linked to nodes representing the characteristics of neighbor zones at time $t - 1$ (see Table 1).

Our dataset was built using five time points. Therefore, each tree represented the four-week evolution of the characteristics associated with zones that leads to the occurence of dengue cases.

The dataset was composed of 181 attributed trees with each node being associated with 15 attributes. The idea was that frequent patterns in this dataset might give some hints concerning the evolution of data in the few weeks preceeding the begining of multiple dengue cases. To filter out common patterns that were not related to the apparition of dengue cases, we constructed another dataset composed of attributed trees for which roots were associated with zones where no-dengue case had been reported. This second dataset, that we called no-dengue dataset, was composed of 9087 attributed trees. Both datasets were used to search distinguishable features, i.e., patterns that are frequently present before the report of several dengue cases, but rares in the absence of dengue.

5.4 Performance evaluation on labeled trees

To position our algorithm w.r.t. existing works, we compare its execution time to mine labeled trees against HybridTreeMiner [11], the best performing algorithm in the literature for mining unordered trees.

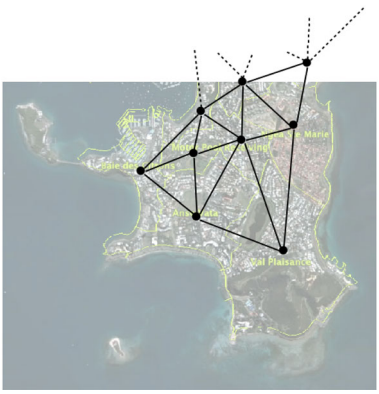
Both algorithms successfully mined all datasets with a minimum support as low as 2. For T10M, the largest dataset, composed of ten millions labeled trees, this represents a relative support of 0.00002 %. With such a low support, more than 35 million frequent patterns were returned. With a support of 2, HybridTreeMiner mined the dataset in 15 min, and it took IMIT four times longer to do it (1 h). Figure 12 shows the performances comparison of IMIT and HybridTreeMiner for mining T10M.

On the other datasets, the performance of the two algorithms is in the same order of magnitude. HybridTreeMiner clearly outperforms IMIT on all datasets and for all support values by a factor varying from 2 to 4. It is perfectly normal that our algorithm, designed to mine attributed tree, is less efficient than algorithms specifically dedicated to mine labeled trees. As such, it is normal to perform worst on mining labeled trees than dedicated implementations.

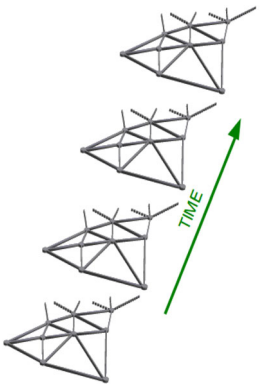
In light of these experiments, we can state that, for both algorithms, the increase in the number of trees has a moderate impact on processing time. For example, for the smallest support of 2, the processing time doubles for each tenfold increase in the dataset size.

Table 1 Creating a database of attributed trees from the dengue dataset

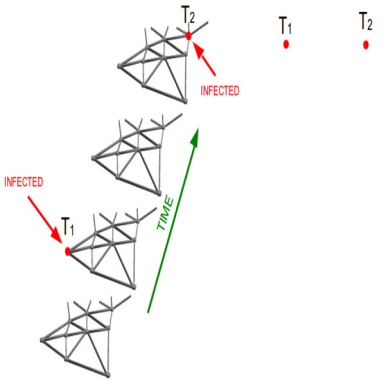
The city is divided in districts characterized by 15 epidemic and environmental attributes.



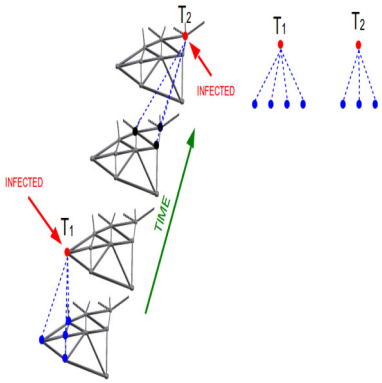
The situation at each time point is represented by an attributed graph.



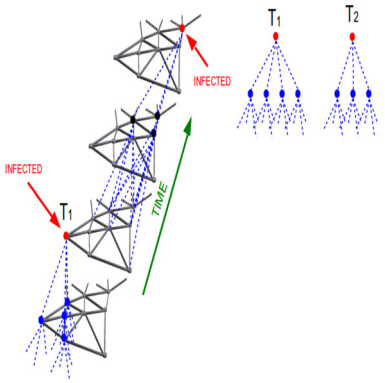
The whode dataset is represented by a succession of attributed graphs.



The roots of attributed trees correspond to infected districts.



Nodes that are just below a root correspond to neighbor zones at time $t-1$.



Nodes at n^{th} level below a root represent neighbor districts at time $t-n$.

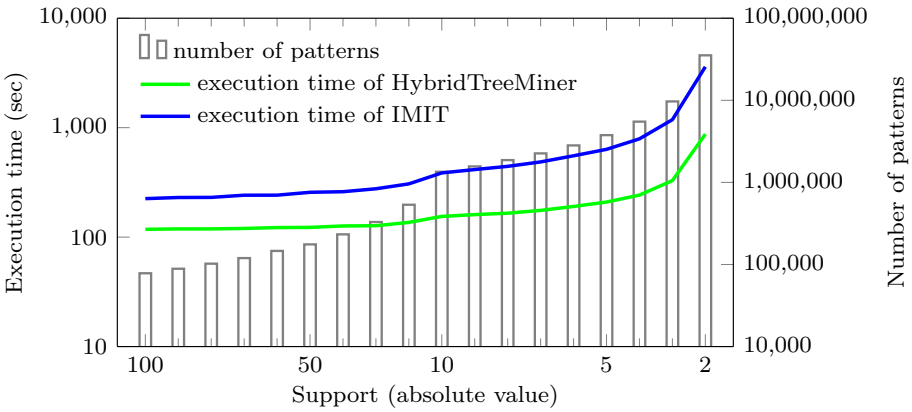


Fig. 12 Performance comparison of IMIT and HybridTreeMiner for mining T10M dataset

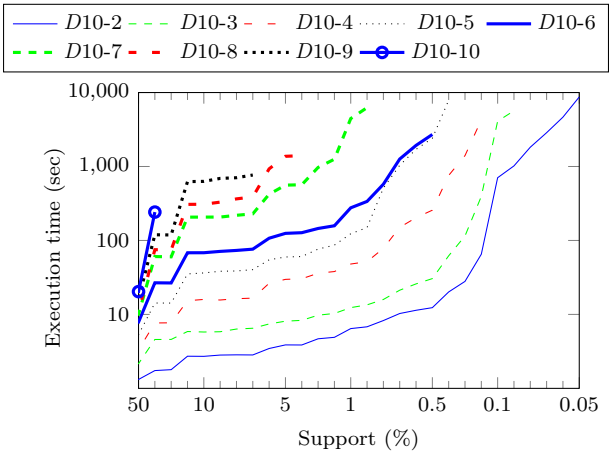


Fig. 13 Execution time of IMIT_CONTENT_CLOSED for mining c-closed sets of induced unordered patterns on 9 synthetic datasets

5.5 Performance evaluation on attributed trees

Figure 13 shows the execution time for mining content-closed patterns. We can see that the increase in the size of itemsets associated with nodes has a huge impact on performance.

For comparison, the mining of D10 dataset in which one item is associated with each node is performed in less than 1.6 s at 1 % support. With two items associated with nodes, the processsing takes 6.43 s. With itemsets of size 4, it takes 48 s and 1 h 15 min with itemsets of size 7. This shows that mining attributed trees is extremely more computing intensive than mining labeled trees and the difference is widely underestimated because only content-closed patterns were mined.

Figure 14 shows the execution time and the number of patterns found by the three versions of IMIT for mining induced unordered patterns on $D10$-3 dataset. As one can see in the figure, mining all patterns generates a huge number of solutions and takes a long time, e.g., mining the $D10$-3 dataset with a minimum support of 5 % outputs half a million patterns in 3 h with the basic IMIT algorithm.

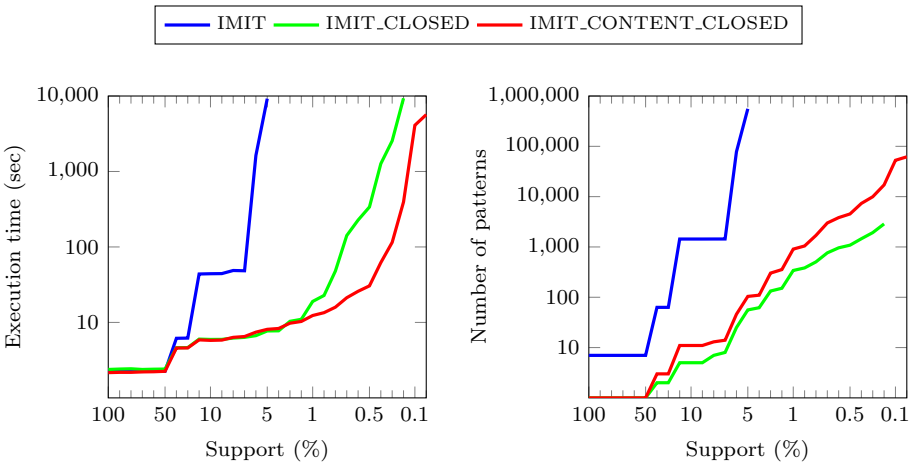


Fig. 14 Execution time (*left plot*) and number of patterns found (*right plot*) by the different versions of IMIT for mining induced unordered patterns on $D10$-3 dataset

Mining content-closed atrees allows to reduce both the number of patterns and the execution time. Thus, with the same dataset, at 5 % minimum support, 104 content-closed patterns are found in 8 s.

As shown in the same figures, the search for closed patterns allows to further reduce the number of patterns. At 5 % minimum support, for example, the number of patterns drops to 56, while the execution time remains the same. However, because of the costly subtree isomorphism checks, in return, performance collapses when patterns become numerous. At 0.5 % minimum support, for example, 4530 content-closed patterns are found in 40 s but it takes 338 s to mine the 1085 closed patterns. The difference in computation time increases as the minimum support decreases. At 0.2 % minimum support, it takes 3 h to mine the 2870 closed patterns while it only takes 8 min to enumerate the 17,097 content-closed patterns. In addition, the mining of content-closed patterns can be pursued further to a minimum support of 0.09 %. Using this minimum support, the algorithm generates 61,844 patterns in 1 h and 45 min.

Figure 15 shows the execution time and the number of patterns found by the 3 versions of IMIT for mining induced unordered patterns on $D10$-5 dataset. The behavior of the three algorithms is very similar to what was found on the $D10$-3 dataset.

5.6 Mining the weblog dataset

The weblog dataset contains fewer attributed trees than the synthetic datasets used before, but as more items are associated with each node, the dataset becomes more difficult to mine.

With a minimum support of 40 %, IMIT extracted 3069 frequent patterns in 41 min. However, at 35 % minimum support, this algorithm did not succeed to extract all frequent patterns in less than 10 h.

In comparison, with a minimum support sets at 40 %, both IMIT_CLOSED and IMIT_CONTENT_CLOSED extracted 18 patterns in 85 and 83 s respectively. By lowering the minimum support to 35 %, the computation took 10 times more time to output 57 patterns.

Detailed results are presented in Fig. 16. On the weblog dataset, the closed patterns and content-closed patterns found were identical for all support values between 6 and 50 %.

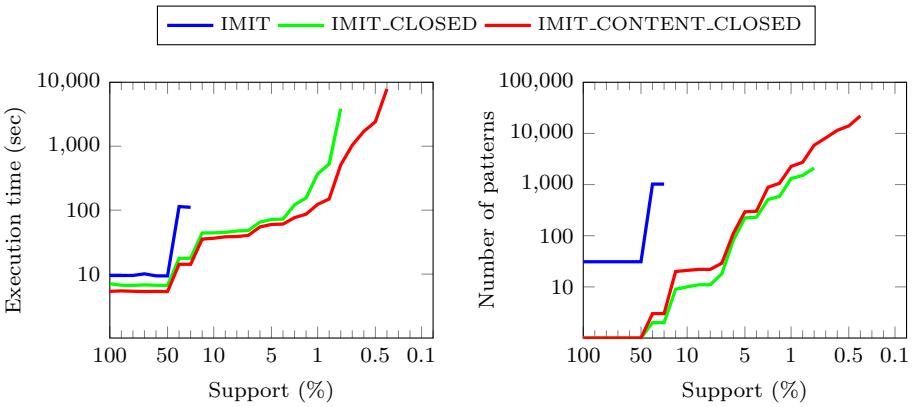


Fig. 15 Execution time (*left plot*) and number of patterns found (*right plot*) by the different versions of IMITfor mining induced unordered patterns on $D10\text{-}5$ dataset

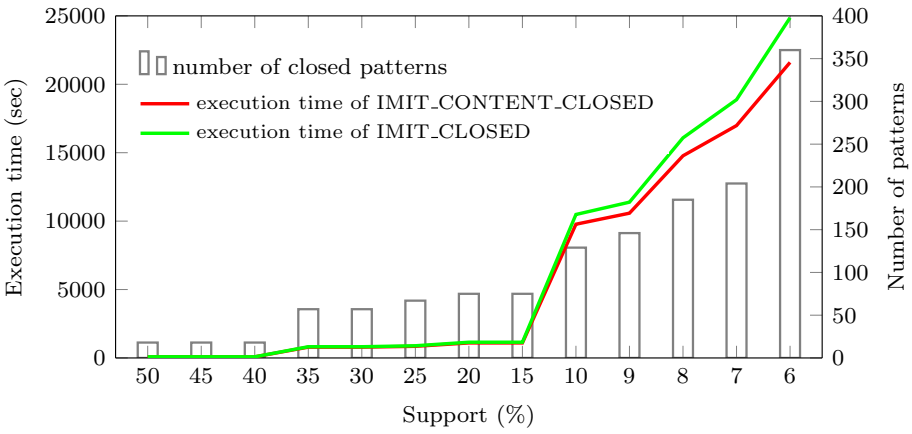


Fig. 16 Performances of IMIT_CLOSED and IMIT_CONTENT_CLOSED for mining induced unordered patterns on the weblog dataset

Mining the weblog dataset with a minimum support of 6 % lasted 6 h and returned 360 patterns. It was not possible to get results with smaller support values.

At 6 % minimum support, the patterns extracted were shared by more than 7000 logs. So, they represented very general features, and it was not possible to extract any useful information from them.

5.7 Mining the dengue dataset

The dengue dataset contains few attributed trees but, as more items are associated with each node, this dataset is intractable for small support values. Fortunately, we were not trying to extract patterns with small support values as we were interested in identifying global trends in the dataset.

We decided to mine the dataset using a minimum support of 70 %. With IMIT, 127 patterns were identified in 19 min and 7 s. The number of extracted patterns was reduced to 112 by using IMIT_CONTENT_CLOSED at the price of a slight increase of the process-

Table 2 Legend of items used in Table 3

Id	Description
1	IHRE = HIGH
2	Precipitation = LOW
3	Maximum wind direction = EAST
4	Average temperature = HIGH
5	Minimum temperature = HIGH
6	Solar irradiance 2AM–2PM = HIGH
7	Solar irradiance 2PM–2AM = LOW
10	Average humidity = NORMAL
14	Average humidity = HIGH
15	Solar irradiance 2AM–2PM = LOW

Table 3 Frequent patterns in the dengue dataset

Patterns	Freq. in dengue dataset	Freq. in no-dengue dataset
1:2:4	79.33	28.93
1:2:4:6	73.74	23.61
1:3:4	79.89	24.87
14 1:4:5 $ 1	75.14	18.55
14 1:4:5 $ 4:5	74.86	18.40
14 1 $ 4	75.46	20.27
14 7:15 $ 7	78.21	26.55
3 1:4 $ 1	69.27	20.59

ing time to 19 min and 34 s. IMIT_CLOSED proved to be the best choice to process this dataset, as it reduced the number of patterns to 98 and the processing time to 18 min and 37 s.

These 98 induced patterns that occurred frequently before the report of several dengue cases could not be considered as indicators of dengue occurrence as they might also have occurred in a situation where no-dengue case was reported. Unfortunately, all frequent patterns in the dengue dataset were also frequent in the no-dengue dataset. The only exception was a pattern that highlighted the fact that before the occurrence of several dengue cases in a district, a high humidity was measured, while a moderate humidity level had been measured the week before in several neighboring districts. Using the legend in Table 2, the pattern was "14 10 $ 10 $ 10." It was present in 80.11 % of the dengue case and in only 27.28 % of the no-dengue dataset.

Mining embedded patterns allow to catch successions of events with time gaps. Unfortunately, the mining of embedded patterns at 70 % minimum support failed with IMIT and IMIT_CLOSED. Only IMIT_CONTENT_CLOSED was capable to process the dataset. The algorithm extracted 9265 patterns in 19 h. As for induced pattern mining, we collected the patterns that were frequent in the dengue dataset but rare in the no-dengue dataset. From the 9,265 closed patterns found in more that 70 % of the trees in the dengue dataset, only eight of them were present in the no-dengue dataset with a support below 30 %.

Table 3 shows the list of these patterns using the legend given in Table 2. The frequency of the patterns is given in the dengue dataset and in the no-dengue dataset.

The patterns listed confirmed some background knowledge in the domain. For example, dengue is preceded by a high temperature associated with a high IHRE index followed by a high humidity; a high IHRE index with a high temperature followed by east wind; or also a low solar irradiance followed by high humidity. A combination of the following factors is also indicative of dengue: High IHRE, low precipitation, high-temperature and high solar irradiance from 2 AM to 2 PM.

6 Conclusion and perspectives

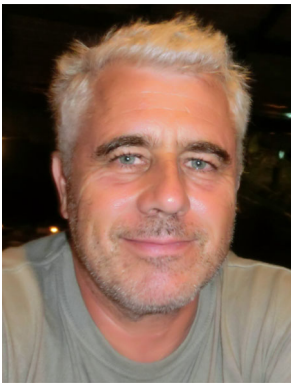
In this paper, we introduced the problem of mining attributed trees. We investigated three types of attributed trees: atrees, closed atrees and c-closed atrees. We proposed three algorithms to mine these patterns. C-closed atrees are a new condensed representation of frequent atrees that is defined w.r.t. itemset inclusion only. As shown by our experiments, mining c-closed atrees provides a good compromise between performances and succinctness. Indeed, mining all frequent atrees returns a huge number of patterns, and mining only closed ones is time consuming due to the cost of subtree isomorphism tests. The efficiency of the proposed algorithm, IMIT_CONTENT_CLOSED, has been demonstrated on large synthetic datasets. It was also used to mine weblog data and a dengue dataset. Concerning the dengue dataset, we show that some patterns, although already known, were characteristics of dengue events.

One of our future direction of work is to extend the proposed algorithm to effectively mine frequent closed patterns. Another direction consists in developing similar methods for mining more complex structures such as attributed graphs.

References

1. Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. SIGMOD Rec 22(2):207–216
2. Agrawal R, Srikant R (1995) Mining sequential patterns. In: ICDE, 95, pp 3–14
3. Asai T, Abe K, Kawasoe S, Arimura H, Sakamoto H, Arikawa S (2002) Efficient substructure discovery from large semi-structured data. In: SDM
4. Asai T, Arimura H, Uno T, Nakano S-I (2003) Discovering frequent substructures in large unordered trees. In: The 6th International Conference on Discovery Science, Springer, pp 47–61
5. Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential pattern mining using a bitmap representation. In: KDD, pp 429–435
6. Balcázar JL, Bifet A, Lozano A (2010) Mining frequent closed rooted trees. Mach Learn 78(1–2):1–33
7. Bayardo RJ (1998) Efficiently mining long patterns from databases. In: ACM SIGMOD International Conference on Management of Data SIGMOD 98, pp 85–93
8. Chehreghani MH (2011) Efficiently mining unordered trees. In: ICDM, pp 111–120
9. Chi Y, Muntz RR, Nijssen S, Kok JN (2004) Frequent subtree mining—an overview. Fundam Inf 66(1–2):161–198
10. Chi Y, Yang Y, Muntz RR (2003) Indexing and mining free trees. In: Proceedings of the 2003 IEEE International Conference on Data Mining (ICDM'03)
11. Chi Y, Yang Y, Muntz RR (2004) Hybridtreeminer: an efficient algorithm for mining frequent rooted trees and free trees using canonical form. In: Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on, pp 11–20
12. Chi Y, Yang Y, Xia Y, Muntz RR (2004) Cmtreeminer: mining both closed and maximal frequent subtrees. In: PAKDD, pp 63–73
13. Deshpande M, Kuramochi M, Karypis G (2003) Frequent sub-structure-based approaches for classifying chemical compounds. In: Third IEEE International Conference on Data Mining, IEEE Comput. Soc, pp 35–42

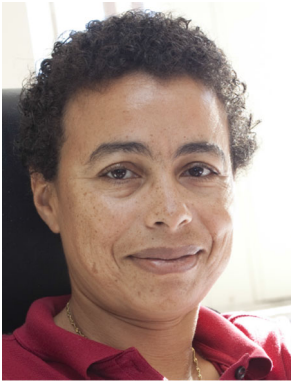
14. Fukuzaki M, Seki M, Kashima H, Sese J (2010) Finding itemset-sharing patterns in a large itemset-associated graph. In: PAKDD, pp 147–159
15. Gay D, Selmaoui-Folcher N, Boulicaut J-F (2010) Application-independent feature construction based on almost-closedness properties. Knowl Inf Syst 30(1):87–111
16. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. SIGMOD Rec 29(2):1–12
17. Hido S, Kawano H (2005) Amiot: induced ordered tree mining in tree-structured databases. In: ICDM, pp 170–177
18. Jiang C, Coenen F, Zito M (2013) A survey of frequent subgraph mining algorithms. Knowl Eng Rev 28:75–105
19. Luccio F, Enriquez AM, Rieumont PO, Pagli L (2001) Exact rooted subtree matching in sublinear time, Universita Di Pisa Technical Report TR-01 14
20. Luccio F, Pagli L (1995) Approximate matching for 2 families of trees. Inf Comput 123(1):111–120
21. Mannila H, Toivonen H (1996) Multiple uses of frequent sets and condensed representations. In: KDD, pp 189–194
22. Miyoshi Y, Ozaki T, Ohkawa T (2009) Frequent pattern discovery from a single graph with quantitative itemsets. In: ICDM Workshops, pp 527–532
23. Moser F, Colak R, Rafiey A, Ester M (2009) Mining cohesive patterns from graphs with feature vectors. In: SDM, pp 593–604
24. Mougel P-N, Rigotti C, Gandrillon O (2012) Finding collections of k-clique percolated components in attributed graphs. In: PAKDD, pp 181–192
25. Nijssen S, Kok JN (2003) Efficient discovery of frequent unordered trees. In: First International Workshop on Mining Graphs, Trees and Sequences (MGTS)
26. Pasquier C, Sanhes J, Flouvat F, Selmaoui-Folcher N (2013) Frequent Pattern Mining in Attributed trees. In: Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'13)., Gold Coast Australia, pp 26–37
27. Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: ICDT, pp 398–416
28. Pensa RG, Boulicaut J-F (2005) From local pattern mining to relevant bi-cluster characterization. In: 6th International Symposium on Intelligent Data Analysis (IDA 2005), pp 293–304
29. Rymon R (1992) Search through systematic set enumeration. In: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92), pp 539–550
30. Selmaoui-Folcher N, Flouvat F (2011) How to use "classical" tree mining algorithms to find complex spatio-temporal patterns?. In: DEXA (2), pp 107–117
31. Termier A, Rousset M-C, Sebag M (2004) Dryade: a new approach for discovering closed frequent trees in heterogeneous tree databases. In: ICDM, pp 543–546
32. Termier A, Rousset M-C, Sebag M, Ohara K, Washio T, Motoda H (2008) Dryadeparent, an efficient and robust closed attribute tree mining algorithm. IEEE Trans Knowl Data Eng 20(3):300–320
33. Wang C, Hong M, Pei J, Zhou H, Wang W, Shi B (2004) Efficient pattern-growth methods for frequent tree pattern mining. In: PAKDD, pp 441–451
34. Washio T, Motoda H (2003) State of the art of graph-based data mining. SIGKDD Explor Newsl 5(1):59–68
35. Xiao Y, Yao J-F, Li Z, Dunham MH (2003) Efficient data mining for maximal frequent subtrees. In: ICDM, pp 379–386
36. Yan X, Yu PS, Han J (2004) Graph indexing: a frequent structure-based approach. In: SIGMOD Conference, pp 335–346
37. Zaki MJ (2002) Efficiently mining frequent trees in a forest. In: KDD, pp 71–80
38. Zaki MJ (2004) Efficiently mining frequent embedded unordered trees. Fundam Inf 66(1–2):33–52
39. Zaki MJ (2005) Efficiently mining frequent trees in a forest: algorithms and applications. IEEE Trans Knowl Data Eng 17(8):1021–1035
40. Zou L, Lu Y, Zhang H, Hu R (2006) Prefixtreespan: a pattern growth algorithm for mining embedded subtrees. In: WISE, pp 499–505



Claude Pasquier received a Ph.D. degree in Computer Science from the University of Nice - Sophia Antipolis, France, in 1994. During his thesis, he explored the use of software engineering paradigms in the field of structured document manipulation. Subsequently, he was a postdoctoral researcher at the Biophysics and Bioinformatics Laboratory of the University of Athens, Greece where he conducted research on protein structure prediction. He held positions at the French National Institute for Research in Computer Science and Control (INRIA) and with Schlumberger, Smart Cards & Terminals division (now Gemalto) where he worked on language-based systems. He is presently a researcher at the french National Center for Scientific Research (CNRS). His current research interests include data mining, bioinformatics and Knowledge-based systems.

Jérémy Sanhes is a Ph.D. student at the University of New Caledonia and INSA Lyon (France). His thesis deals with spatio-temporal data mining issues.

Frédéric Flouvat is an Associate Professor at the University of New Caledonia (Nouméa, New Caledonia), where he is teaching Algorithmic and Databases. He is also a member of a multidisciplinary research team on material and environment. This laboratory brings together geologists, physicists and computer scientists to address both fundamental and applied questions related to the concepts of risk and sustainable development. His research interests are spatio-temporal data mining and its application to environmental sciences.



Nazha Selmaoui-Folcher is an Associate Professor (HDR) at the University of New Caledonia since 1998. She is the leader of a multidisciplinary laboratory on material and environment since 2012 (PPME EA 3325). She is teaching Computer Sciences and Mathematics. She is the coordinator for the FOSTER national projects dedicated to Knowledge Discovery on Spatio-temporal Databases and Application to Soil Erosion. She received her Ph.D. degree in 1992 from the "Institut National des Sciences Appliques" at Lyon (France) and her Habilitation degree in 2012 from the University of Lyon I. Her current research interests are data mining of time series of satellite images analysis and spatio-temporal data and its application to environmental sciences. She is involved in the program committees of many data mining conferences.