Mining Frequent Closed Unordered Trees Through Natural Representations *

José L. Balcázar, Albert Bifet and Antoni Lozano

Universitat Politècnica de Catalunya,
{balqui,abifet,antoni}@lsi.upc.edu

Abstract. Many knowledge representation mechanisms consist of link-based structures; they may be studied formally by means of unordered trees. Here we consider the case where labels on the nodes are nonexistent or unreliable, and propose data mining processes focusing on just the link structure. We propose a representation of ordered trees, describe a combinatorial characterization and some properties, and use them to propose an efficient algorithm for mining frequent closed subtrees from a set of input trees. Then we focus on unordered trees, and show that intrinsic characterizations of our representation provide for a way of avoiding the repeated exploration of unordered trees, and then we give an efficient algorithm for mining frequent closed unordered trees.

1 Introduction

Trees, in a number of variants, are basically connected acyclic undirected graphs, with some additional structural notions like a distinguished vertex (root) or labelings on the vertices. They are frequently a great compromise between graphs, which offer richer expressivity, and strings, which offer very efficient algorithmics. From AI to Compilers, through XML dialects, trees are now ubiquitous in Informatics.

One form of data analysis contemplates the search of frequent (or the so-called "closed") substructures in a dataset of structures. In the case of trees, there are two broad kinds of subtrees considered in the literature: subtrees which are just induced subgraphs, called *induced subtrees*, and subtrees where contraction of edges is allowed, called *embedded subtrees*. In these contexts, the process of "mining" usually refers, nowadays, to a process of identifying which common substructures appear particularly often, or particularly correlated with other substructures, with the purpose of inferring new information implicit in a (large) dataset. In our case, the dataset would consist of a large set (more precisely, bag) of trees; algorithms for mining embedded labeled frequent trees include TreeMiner [22], which finds all embedded ordered subtrees that appear with a

* Partially supported by the 6th Framework Program of EU through the integrated project DELIS (#001907), by the EU PASCAL Network of Excellence, IST-2002-506778, by the MEC TIN2005-08832-C03-03 (MOISES-BAR), MCYT TIN2004-07925-C03-02 (TRANGRAM), and CICYT TIN2004-04343 (iDEAS) projects.

frequency above a certain threshold, and similarly SLEUTH [23], for unordered trees. Algorithms for mining induced labeled frequent trees include FreqT [2], which mines ordered trees, and uFreqT [15], uNot [3], HybridTreeMiner [8], and PathJoin [18] for unordered trees. FreeTreeMiner [11] mines induced unordered free trees (i.e. there is no distinct root). A comprehensive introduction to the algorithms on unlabeled trees can be found in [17] and a survey of works on frequent subtree mining can be found in [9].

Closure-based mining refers to mining closed substructures, in a sense akin to the closure systems of Formal Concept Analysis; although the formal connections are not always explicit. For trees, a closed subtree is one that, if extended in any manner, leads to reducing the set of data trees where it appears as a subtree; and similarly for graphs. Frequent closed trees (or sets, or graphs) give the same information about the dataset as the set of all frequent trees (or sets, or graphs) in less space. Yan and Han [19, 20] proposed two algorithms for mining frequent and closed graphs. The first one is called gSpan (graph-based Substructure pattern mining) and discovers frequent graph substructures without candidate generation. gSpan builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the depth-first search strategy to mine frequent connected subgraphs. The second one is called CloseGraph and discovers closed graph patterns. CloseGraph is based on gSpan, and is based on the development of two pruning methods: equivalent occurrence and early termination. The early termination method is similar to the early termination by equivalence of projected databases method in CloSpan [21], an algorithm of the same authors for mining closed sequential patterns in large datasets. However, in graphs there are some cases where early termination may fail and miss some patterns. By detecting and eliminating these cases, CloseGraph guarantees the completeness and soundness of the closed graph patterns discovered.

Chin et al. proposed CMTreeMiner [10], the first algorithm to discover all closed and maximal frequent labeled induced subtrees without first discovering all frequent subtrees. CMTreeMiner shares many features with CloseGraph, and uses two pruning techniques: the *left-blanket* and *right-blanket* pruning. The *blanket* of a tree is defined as the set of immediate supertrees that are frequent, where an *immediate supertree* of a tree t is a supertree of t that has one more vertex than t. The *left-blanket* of a tree t is the blanket where the vertex added is not in the right-most path of t (the path from the root to the rightmost vertex of t). The *right-blanket* of a tree t is the blanket where the vertex added is in the right-most path of t. Their method is as follows: it computes, for each candidate tree, the set of trees that are occurrence-matched with its blanket's trees. If this set is not empty, they apply two pruning techniques using the left-blanket and right-blanket. If it is empty, then they check if the set of trees that are transaction-matched but not occurrence matched with its blanket's trees is also empty. If this is the case, there is no supertree with the same support and then the tree is closed.

Arimura and Uno [1] considered closed mining in attribute trees, which is a subclass of labeled ordered trees and can also be regarded as a fragment of description logic with functional roles only. These attribute trees are defined using a relaxed tree inclusion. Termier et al. [16] considered the frequent closed tree discovery problem for a class of trees with the same constraint as attribute trees.

We propose a representation of ordered trees in Section 2.1 and describe a combinatorial characterization and some properties. In Section 3 we present the new algorithm for mining ordered frequent trees. In Section 4 we extend our method to unordered trees and show that intrinsic characterizations of our representation provide for a way of avoiding the repeated exploration of unordered trees. We propose in Section 5 a new efficient algorithm for mining rooted induced closed subtrees taking advantage of some combinatorial properties of our new representation of trees.

These mining processes can be used for a variety of tasks. Let us describe some possibilities. Consider web search engines. Already the high polysemy of many terms makes sometimes difficult to find information through them; for instance, a researcher of soil science may have a very literal interpretation in mind when running a web search for "rolling stones", but it is unlikely that the results are very satisfactory; or a computer scientist interested in parallel models of computation has a different expectation from that of parents-to-be when a search for "prams" is launched. A way for distributed, adaptive search engines to proceed may be to distinguish navigation on unsuccessful search results, where the user follows a highly branching, shallow exploration, from successful results, which give rise to deeper, little-branching navigation subtrees. Indeed, through an artificially generated bimodal dataset, consisting of shallow trees (modeling unsuccessful searches, with average fanout 8, depth 3) and deeper, slender trees (modeling successful searches, fanout 2, depth 8), the closures found by our algorithms distinguish quite clearly about 2/3 of the shallow trees; they fail to distinguish small shallow trees from small deep trees, which is not surprising since when the size is small, trees are both slender and shallow. However, a large majority of the shallow trees got clearly separated from the others.

As another example, consider the KDD Cup 2000 data [14]. This dataset is a web log file of a real internet shopping mall (gazelle.com). This dataset, of size 1.2GB, contains 216 attributes. We used the attribute 'Session ID' to associate to each user session a unique tree. The trees record the sequence of web pages that have been visited in a user session. Each node tree represents a content, assortment and product path. Trees are not built using the structure of the web site, instead they are built following the user streaming. Each time a user visits a page, if he has not visited it before, we take this page as a new deeper node, otherwise, we backtrack to the node this page corresponds to, if it is the last node visited on a concrete depth. The resulting dataset consists of $225,558$ trees. On them, an improved variation of our algorithms was considerably faster than the only alternative algorithm now available for the task, CMTreeMiner. Further analysis and improvements will be reported detailedly in [6].

2 Preliminaries

Our *trees* will be finite rooted trees with nodes of unbounded arity, and we will consider two kinds of trees: *ordered trees*, in which the children of any node form a sequence of siblings, and *unordered trees*, in which they form a set of siblings. A *bottom-up subtree* of a tree t is any connected subgraph rooted at some node v of t which contains all the descendants of v in t and no more. The *depth* of a node is the length of the path from the root to that node (the root has depth 0). A bottom-up subtree of a tree t is at depth d if its root is at depth d in t.

In order to compare link-based structures, we will also be interested in a notion of subtree where the root is preserved. A tree t' is a *top-down subtree* (or simply a *subtree*) of a tree t if t' is a connected subgraph of t which contains the root of t.

Given a finite dataset D of unlabeled rooted trees, we say that one of them s_i *supports* a tree t if t is a subtree of s_i. The number of indices i whose s_i in the dataset D supports t is called the *support* of the tree t. A subtree t is called *frequent* if its support is greater than or equal to a given threshold min_sup specified. The frequent subtree mining problem is to find all frequent subtrees in a given dataset. Any subtree of a frequent tree is also frequent and any supertree of an nonfrequent tree is also nonfrequent.

We define a frequent tree t to be *closed* if none of its proper supertrees has the same support as it has. Generally, there are much fewer closed sets than frequent ones. In fact, we can obtain all frequent subtrees with their support from the set of closed frequent subtrees with their supports. So, the set of closed frequent subtrees maintains the same information as the set of all frequent subtrees.

2.1 Natural Representations

Sequences of natural numbers will play a technical role in our development in order to represent both ordered and unordered trees.

Definition 1. *A* natural sequence *is a finite sequence of natural numbers, starting and ending by 0, and where each difference of consecutive numbers is either +1 or -1.*

In order to describe our representation of ordered trees, we will consider the following operations of general sequences of natural numbers (not necessarily natural sequences).

Definition 2. *Given two sequences of natural numbers x, y, we represent by*

- $x \cdot y$ *the sequence obtained as concatenation of x and y*
- $x_{i:j}$ *the subsequence of numbers starting at position i up to position j in x (where positions range from 1 to $|x|$, the length of x)*
- $x + i$ *($x - i$) the sequence obtained adding (subtracting) i to each component of x*

We represent by x^+ the sequence $x+1$ and consider that $+$ has precedence over concatenation.

For example, if $x = (0) \cdot (0,1,0)^+ \cdot (0)$, then $x = (0,1,2,1,0)$, and $x_{3:5} = (2,1,0)$. Now, we can represent trees by means of natural sequences.

Definition 3. *We define a function $\langle \cdot \rangle$ from the set of ordered trees to the set of natural sequences as follows. Let t be an ordered tree. If t is a single node, then $\langle t \rangle = (0)$. Otherwise, if t is composed of the trees $t_1, \ldots, t_k$ joined to a common root r (where the ordering $t_1, \ldots, t_k$ is the same of the children of r), then*

$$\langle t \rangle = (0) \cdot \langle t_1 \rangle^+ \cdot (0) \cdot \langle t_2 \rangle^+ \cdot \ldots \cdot (0) \cdot \langle t_k \rangle^+ \cdot (0).$$

We say that $\langle t \rangle$ is the natural representation of t.

Note that, first of all, the above function is well defined: the recursive definition of $\langle t \rangle$ ensures that it is a natural sequence. It is also easy to see that it is a bijection between the ordered trees and the natural sequences and that $\langle t \rangle$ basically corresponds to a pre-post-order traversal of t where each number of the sequence represents the depth of the current node in the traversal. As an example, the natural representation of the tree in the Figure 1 is the natu-

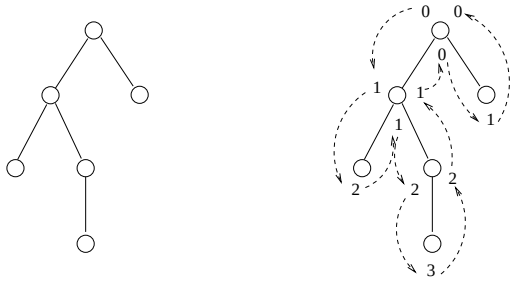

Fig. 1. A tree on the left and its pre-post-order traversal on the right.

ral sequence $(0,1,2,1,2,3,2,1,0,1,0)$. Note that, for example, the subsequence $(1,2,1,2,3,2,1)$ corresponds to the bottom-up subtree rooted at the left son of the root. We can state this fact in general:

Proposition 1. *Let $x = \langle t \rangle$, where t is an ordered tree. Then, t has a bottom-up subtree r at depth $d > 0$ if and only if $(d-1) \cdot (\langle r \rangle + d) \cdot (d-1)$ is a subsequence of x.*

Proof. We prove it by induction on d. If $d = 1$, then since

$$(d-1) \cdot (\langle r \rangle + d) \cdot (d-1) = (0) \cdot \langle r \rangle^+ \cdot (0)$$

the property holds by the recursive definition of natural representation. For the induction step, let $d > 1$. To show one direction, suppose that r is a bottom-up subtree of t at depth d. Then, r must be a bottom-up subtree of one of the bottom-up subtrees corresponding to the children of the root of t. Let t' be the bottom-up subtree at depth 1 that contains r. Since r is at depth $d-1$ in t', the induction hypothesis states that $(d-2)\cdot(\langle r\rangle + d - 1)\cdot(d-2)$ is a subsequence of $\langle t'\rangle$. But $(0)\cdot\langle t'\rangle^+\cdot(0)$ is also, by definition, a subsequence of x. Combining both facts, we get that $(d-1)\cdot(\langle r\rangle + d)\cdot(d-1)$ is a subsequence of x, as desired. The argument also works in the contrary direction, and we get the equivalence. □

3 Mining frequent subtrees in the ordered case

Our approach here is similar to gSpan: we represent the potential subtrees to be checked for frequent on the dataset in such a way that extending them by one single node corresponds to a clear and simple operation on the representation. The completeness of the procedure is mathematically proved, that is, we argue that all trees can be obtained in this way with our notion of extension, defined below. This allows us to avoid extending trees that are found to be already nonfrequent.

We show now that our representation allows us to traverse the whole subtree space by an operation of extension by a single node, in a simple way. The possible places where the new node can be added in our natural representation belong to the right-most branch of the tree, which we call the *tail* —our natural representations are divided into a head and a tail—.

Definition 4. *Given a natural sequence x, the* tail *of x, denoted by $\mathsf{tail}(x)$, is the longest decreasing subsequence which is a suffix of x. The* head *of x, denoted by $\mathsf{head}(x)$, is the prefix y such that $x = y\cdot\mathsf{tail}(x)$.*

Now we can specify how to make one or more steps in the generation of (a representation of) a tree.

Definition 5. *Given two natural sequences x, y, we say that x* yields *y* in one step *(in symbols, $x \vdash^1 y$) if y is obtained from x by replacing some element e in $\mathsf{tail}(x)$ by the subsequence $(e, e+1, e)$.*

Definition 6. *A succession of natural sequences $x_1, x_2, \ldots, x_n$ such that $x_1 \vdash^1 x_2 \vdash^1 \cdots \vdash^1 x_n$ is called a* generating path *from x_1 to x_n. Given two natural sequences x, y, if there is a generating path from x to y, we write $x \vdash^* y$ and we say that x* yields *y.*

Our terminology with trees will be the following: if t and t' are two trees and $\langle t\rangle$ yields $\langle t'\rangle$, then we say that t' *is extended from* t (*in one step* if $\langle t\rangle$ yields $\langle t'\rangle$ in one step). All the trees extended from t are called the *extensions of* t. Now we can state the main result that validates the algorithm above.

Theorem 1. *For every natural sequence x, there is a unique generating path from (0) to x.*

Proof. The existence of a generating path for a natural sequence x is straightforward by induction on $n = (|x| + 1)/2$. Note that all natural sequences are of odd length. If $n = 1$, then $|x| = 1$, and the only possibility is that $x = (0)$; therefore, there is a generating path (of length 0). Now suppose that $n > 1$, and let $k = |\mathsf{tail}(x)|$. Note that $k \geq 2$. By eliminating the first two elements of $\mathsf{tail}(x)$ in x, we get the natural sequence

$$x' = \mathsf{head}(x) \cdot \mathsf{tail}(x)_{3:k}$$

where $\mathsf{tail}(x)_{3:k}$ is the empty sequence if $|\mathsf{tail}(x)| = 2$. Note that the last number e in $\mathsf{head}(x)$ belongs to the tail of x', so we can replace it by $(e, e+1, e)$, obtaining x. Therefore $x' \vdash^1 x$. Since $(|x'| + 1)/2 = n - 1$, by induction hypothesis there is a generating path from (0) to x' and, therefore, to x.

To show uniqueness, suppose that y, with $|y| = n$, is a natural representation obtained by two different generating paths from (0). Let x be the natural representation in both paths from which the two different paths split. Then, there are two natural sequences x' and x'' such that $x' \neq x''$ and

$$x \vdash^1 x' \vdash^* y \quad \text{and} \quad x \vdash^1 x'' \vdash^* y.$$

Since x yields x' and x'' in one step, $|x'| = |x''|$ and $\mathsf{tail}(x') \neq \mathsf{tail}(x'')$ (different replacements in $\mathsf{tail}(x)$ produce different new tails), the lengths of the tails $\mathsf{tail}(x')$ and $\mathsf{tail}(x'')$ must be different. W.l.o.g., suppose that $|\mathsf{tail}(x')| < |\mathsf{tail}(x'')| = k$. Now we claim that

$$x'_{n-k+1} < x''_{n-k+1}, \tag{1}$$

the reason being that x''_{n-k+1} belongs to the tail of x'' (contrarily to x'_{n-k+1}), and so it has the maximum possible value at position $n - k + 1$. Consider the following properties, which are straightforward and we state without proof.

Property 1. If $u \vdash^* v$ for two natural sequences u and v, then $\mathsf{head}(u)$ is a prefix of v.

Property 2. If $u \vdash^* v$ for two natural sequences u and v, then $u_i \leq v_i$ for every $i \leq |u|$.

Now, we have the following. The element x'_{n-k+1} of x' does not belong to $\mathsf{tail}(x')$ (because its length is less than k). Therefore, by Property 1 and the fact that $x' \vdash^* y$ we have that $y_{n-k+1} = x'_{n-k+1}$. On the other hand, since $x'' \vdash^* y$, we have by Property 2 that $x''_{n-k+1} \leq y_{n-k+1}$. Finally, by inequality 1, we get the contradiction

$$y_{n-k+1} = x'_{n-k+1} < x''_{n-k+1} \leq y_{n-k+1}.$$

□

For this section we could directly use gSpan, since our structures can be handled by that algorithm. However, our goal is the improved algorithm described in the next section, to be applied when the ordering in the subtrees is irrelevant for the application.

Indeed, we have designed our representation in such a way that it will allow us to check only canonical representatives for the unordered case, thus saving the computation of support for all (except one) of the ordered variations of the same unordered tree. Figures 2 and 3 show the gSpan-based algorithm, which is as follows: beginning with a tree of a single node, it calls recursively the FREQUENT_ORDERED_SUBTREE_MINING algorithm doing one-step extensions and checking that they are still frequent.

FREQUENT_ORDERED_MINING(D, min_sup)

Input: A tree dataset D, and min_sup.
Output: The frequent tree set T.

1 $t \leftarrow$ one node tree
2 $T \leftarrow \emptyset$
3 $T \leftarrow$ FREQUENT_ORDERED_SUBTREE_MINING(t, D, min_sup, T)
4 **return** T

Fig. 2. The Frequent Ordered Mining algorithm

FREQUENT_ORDERED_SUBTREE_MINING(t, D, min_sup, T)

Input: A tree t, a tree dataset D, and min_sup.
Output: The frequent tree set T.

1 insert t into T
2 **for** every t' that can be extended from t in one step
3 **do if** support(t') $\geq min\_sup$
4 **then** $T \leftarrow$ FREQUENT_ORDERED_SUBTREE_MINING(t', D, min_sup, T)
5 **return** T

Fig. 3. The Frequent Ordered Subtree Mining algorithm

4 Mining frequent subtrees in the unordered case

The main result of this section is a precise mathematical characterization of the natural representations that correspond to canonical variants of unordered trees.

In unordered trees, the children of a given node form sets of siblings instead of sequences of siblings. Therefore, ordered trees that only differ in permutations of the ordering of siblings are to be considered the same unordered tree. We select one of them to act as canonical representative of all the ordered trees corresponding to the same unordered tree: by convention, this canonical representative has larger trees always to the left of smaller ones. More precisely,

Definition 7. *Let t be an unordered tree, and let $t_1, \ldots, t_n$ be all the ordered trees obtained from t by ordering in all possible ways all the sets of siblings of t. The* canonical representative *of t is the ordered tree t_0 whose natural representation is maximal (according to lexicographic ordering) among the natural representations of the trees t_i, that is, such that*

$$\langle t_0 \rangle = \max\{\langle t_i \rangle \mid 1 \leq i \leq n\}.$$

We can use, actually, the same algorithm as in the previous section to mine unordered trees; however, much work is unnecessarily spent in checking repeatedly ordered trees that correspond to the same unordered tree as one already checked. A naive solution is to compare each tree to be checked with the ones already checked, but in fact this is an inefficient process, since all ways of mapping siblings among them must be tested.

A far superior solution would be obtained if we could count frequency only for canonical representatives. We explain next one way to do this: the use of natural representations allows us to decide whether a given (natural representation of a) tree is canonical, by using an intrinsic characterization which is not stated in terms of the representations of all other ordered trees; but only in terms of the natural representation itself.

For the characterization, it is convenient to use the notion of splitting triplet:

Definition 8. *Given a natural sequence $(x_1, \ldots, x_n)$, a* splitting triplet *(i, j, k) represents three different positions $i < j < k$ of the sequence having the same values ($x_i = x_j = x_k$) and with all intermediate values strictly higher.*

The notion of splitting triplet is the natural-representation analogue of "consecutive" bottom-up subtrees.

Lemma 1. *Let t be an ordered tree and $x = \langle t \rangle$. Then, x contains a splitting triplet (i, j, k) if and only if there is a node in t at depth x_i which is the parent of two bottom-up subtrees t_1, t_2 such that $\langle t_1 \rangle = x_{i:j} - x_i$ and $\langle t_2 \rangle = x_{j:k} - x_i$, and t_1 appears immediately before t_2.*

Proof. Suppose that x contains the splitting triplet (i, j, k), and let $d = x_i + 1$. Consider the subsequence $x_{i:j}$, which can be expressed as

$$(x_i) \cdot x_{i+1:j-1} \cdot (x_j) = (d-1) \cdot y \cdot (d-1)$$

where all the values in y are strictly higher than $d-1$ (because of the conditions of splitting triplets). Since y is a subsequence of x, $y-d$ must be a natural sequence

which, by Theorem 1, has a generating path and, therefore, corresponds to a tree t_1. So, $y = \langle t_1 \rangle + d$ and we get that

$$x' = (d-1) \cdot (\langle t_1 \rangle + d) \cdot (d-1)$$

is a subsequence of x. Similarly, it can be argued that

$$x'' = (d-1) \cdot (\langle t_2 \rangle + d) \cdot (d-1)$$

is a subsequence of x, for some tree t_2. By Proposition 1, both x' and x'' correspond to two bottom-up subtrees of t at depth d. Since they share the position j (as the end of x' and beginning of x''), the roots of t_1 and t_2 must be siblings, and t_1 must appear immediately before t_2.

The other direction can be shown by a similar argument: the existence of the two subtrees t_1 and t_2 as described in the statement implies, by Proposition 1, that their natural representations must appear in x (modulo an increase of d in their values) as is said above. □

Theorem 2. *A natural sequence x corresponds to a canonical representative if and only if, for every splitting triplet (i, j, k), $x_{i:j} \geq x_{j:k}$ in lexicographic order.*

Proof. If x corresponds to a canonical representative, it is clear that the condition about the splitting triplets must hold. Otherwise, the existence of a triplet (i, j, k) in x where $x_{i:j} < x_{j:k}$, would imply, by Lemma 1, the possibility of a better ordering of the two corresponding bottom-up subtrees, resulting in a tree t' with $\langle t' \rangle > x$, which is a contradiction.

To show the other direction, suppose that x does not correspond to a canonical representative. Therefore, a reordering of two bottom-up subtrees whose roots are consecutive siblings must give a larger natural representation. Again by Lemma 1, this corresponds to the existence of a triplet (i, j, k) in x which does not satisfy the above condition, that is, such that $x_{i:j} < x_{j:k}$. □

We must point out here that a further, later refinement of the notion of natural representation described here, however, has given us improved performance for the mining algorithms. Hence, the major contribution of this work lies in these algorithms, and not in the present section, and we suggest the interested reader to consult [6] (soon to be available at the authors web pages) for the most attractive results on how to check for unordered tree representatives in natural representations.

Figure 4 shows the gSpan-based algorithm for unordered trees (for which we use a similar calling procedure as for the ordered case shown in Figure 2). The main difference with the algorithm for the case of ordered trees is that FREQUENT_UNORDERED_SUBTREE_ MINING checks at the beginning that the input tree is its canonical representative using the method implicit in Theorem 2.

```
FREQUENT_UNORDERED_SUBTREE_MINING(t, D, min_sup, T)
    Input: A tree t, a tree dataset D, and min_sup.
    Output: The frequent tree set T.

 1  if not CANONICAL_REPRESENTATIVE(t)
 2     then return T
 3  insert t into T
 4  C ← ∅
 5  for every t' that can be extended from t in one step
 6        do if support(t') ≥ min_sup
 7              then insert t' into C
 8  for each t' in C
 9        do T ← FREQUENT_UNORDERED_SUBTREE_MINING(t', D, min_sup, T)
10  return T
```

Fig. 4. The Frequent Unordered Subtree Mining algorithm

5 Closure-based mining

In [5] we aim at clarifying the properties of closed trees, providing a more detailed justification of the term "closed" through a closure operator obtained from a Galois connection, along the lines of [12], [7], [13], or [4] for unstructured or otherwise structured datasets. Also, we designed two algorithms for finding the intersection of two subtrees: the first one in a recursive way, and the second one using dynamic programming.

In this section, we propose a new algorithm to mine unordered frequent closed trees. Figure 5 illustrates the framework.

Figure 6 shows the pseudocode of CLOSED_UNORDERED_SUBTREE_ MINING. It is similar to FREQUENT_UNORDERED_SUBTREE_MINING, adding a checking of closure in lines 9-10.

```
CLOSED_UNORDERED_MINING(D, min_sup)
    Input: A tree dataset D, and min_sup.
    Output: The closed tree set T.

1  t ← one node tree
2  T ← ∅
3  T ← CLOSED_UNORDERED_SUBTREE_MINING(t, D, min_sup, T)
4  return T
```

Fig. 5. The Closed Unordered Mining algorithm

```
CLOSED_UNORDERED_SUBTREE_MINING(t, D, min_sup, T)
    Input: A tree t, a tree dataset D, and min_sup.
    Output: The closed frequent tree set T.

 1  if not CANONICAL_REPRESENTATIVE(t)
 2     then return T
 3  C ← ∅
 4  for every t' that can be extended from t in one step
 5        do if support(t') ≥ min_sup
 6              then insert t' into C
 7        do if support(t') = support(t)
 8              then t is not closed
 9  if t is closed
10     then insert t into T
11  for each t' in C
12        do T ← CLOSED_UNORDERED_SUBTREE_MINING(t', D, min_sup, T)
13  return T
```

Fig. 6. The Closed Unordered Subtree Mining algorithm

References

1. Hiroki Arimura and Takeaki Uno. An output-polyunomial time algorithm for mining frequent closed attribute trees. In *ILP*, pages 1–19, 2005.
2. Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *SDM*, 2002.
3. Tatsuya Asai, Hiroki Arimura, Takeaki Uno, and Shin-Ichi Nakano. Discovering frequent substructures in large unordered trees. In *Discovery Science*, pages 47–61, 2003.
4. Jaume Baixeries and José L. Balcázar. Discrete deterministic data mining as knowledge compilation. In *Workshop on Discrete Math. and Data Mining at SIAM DM Conference*, 2003.
5. José L. Balcázar, Albert Bifet, and Antoni Lozano. Intersection algorithms and a closure operator on unordered trees. In *MLG 2006, 4th International Workshop on Mining and Learning with Graphs*, 2006.
6. José L. Balcázar, Albert Bifet, and Antoni Lozano. Mining frequent closed rooted trees. Submitted, 2007.
7. José L. Balcázar and Gemma C. Garriga. On Horn axiomatizations for sequential data. In *ICDT*, pages 215–229 (extended version to appear in Theoretical Computer Science), 2005.
8. Y. Chi, Y. Yang, and R. R. Muntz. HybridTreeMiner: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms. In *SSDBM '04: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, page 11, Washington, DC, USA, 2004. IEEE Computer Society.

9. Yun Chi, Richard Muntz, Siegfried Nijssen, and Joost Kok. Frequent subtree mining – an overview. *Fundamenta Informaticae*, XXI:1001–1038, 2001.
10. Yun Chi, Yi Xia, Yirong Yang, and Richard Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *Fundamenta Informaticae*, XXI:1001–1038, 2001.
11. Yun Chi, Yirong Yang, and Richard R. Muntz. Indexing and mining free trees. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 509, Washington, DC, USA, 2003. IEEE Computer Society.
12. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer-Verlag, 1999.
13. Gemma C. Garriga. Formal methods for mining structured objects. PhD Thesis, 2006.
14. Ron Kohavi, Carla Brodley, Brian Frasca, Llew Mason, and Zijian Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000.
15. Siegfried Nijssen and Joost N. Kok. Efficient discovery of frequent unordered trees. In *First International Workshop on Mining Graphs, Trees and Sequences*, pages 55–64, 2003.
16. Alexandre Termier, Marie-Christine Rousset, and Michele Sebag. DRYADE: a new approach for discovering closed frequent trees in heterogeneous tree databases. In *ICDM*, pages 543–546, 2004.
17. Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.
18. Yongqiao Xiao, Jenq-Foung Yao, Zhigang Li, and Margaret H. Dunham. Efficient data mining for maximal frequent subtrees. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 379, Washington, DC, USA, 2003. IEEE Computer Society.
19. Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 721, Washington, DC, USA, 2002. IEEE Computer Society.
20. Xifeng Yan and Jiawei Han. CloseGraph: mining closed frequent graph patterns. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295, New York, NY, USA, 2003. ACM Press.
21. Xifeng Yan, Jiawei Han, and Ramin Afshar. CloSpan: Mining closed sequential patterns in large databases. In *SDM*, 2003.
22. Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
23. Mohammed J. Zaki. Efficiently mining frequent embedded unordered trees. *Fundam. Inform.*, 66(1-2):33–52, 2005.