Mining Adaptively Frequent Closed Unlabeled Rooted Trees in Data Streams

Albert Bifet Ricard Gavaldà
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Barcelona, Spain
{abifet,gavalda}@lsi.upc.edu

ABSTRACT

Closed patterns are powerful representatives of frequent patterns, since they eliminate redundant information. We propose a new approach for mining closed unlabeled rooted trees adaptively from data streams that change over time. Our approach is based on an efficient representation of trees and a low complexity notion of relaxed closed trees, and leads to an on-line strategy and an adaptive sliding window technique for dealing with changes over time. More precisely, we first present a general methodology to identify closed patterns in a data stream, using Galois Lattice Theory. Using this methodology, we then develop three closed tree mining algorithms: an incremental one IncTreeNat, a sliding-window based one, WinTreeNat, and finally one that mines closed trees adaptively from data streams, AdaTreeNat. To the best of our knowledge this is the first work on mining frequent closed trees in streaming data varying with time. We give a first experimental evaluation of the proposed algorithms.

Categories and Subject Descriptors

H.2.8 [**Database applications**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

Data streams, closed mining, concept drift, patterns, trees

1. INTRODUCTION

Tree-structured representations are a main key idea pervading all of Computer Science; many link-based structures may be studied formally by means of trees. From the parsing structures in Compiler Design and Natural Language Processing, to the B+ indices that make our commercial Database Management Systems useful, through search-tree or heap data structures, tree automata, the decision tree structures in Artificial Intelligence and Decision Theory, or the now-ubiquitous XML, they often represent an optimal compromise between the conceptual simplicity and processing efficiency of strings and the harder but much richer knowledge representation formalisms based on graphs. Accordingly, a wealth of slight variations of the basic notions, both of the structures themselves (binary, bounded-rank, unranked, ordered, unordered) or of their relationships (induced or embedded, top-down or bottom-up subtree relations) have been proposed for study and motivated applications. In particular, mining frequent trees is becoming an important task, with broad applications including chemical informatics, computer vision, text retrieval, bioinformatics, and web analysis. We focus on finding navigation patterns in web sites and web logs, so we are interested on unlabeled induced rooted trees, thus our relevant information is the root and the link structure. Unlabeled trees are also a previous step to mine labeled trees, a more powerful pattern in applications.

Closure-based mining on purely relational data, that is, itemset mining, is, by now, well-established, and there are interesting algorithmic developments. Sharing some of the attractive features of frequency-based summarization of subsets, it offers an alternative view with advantages; first, by imposing closure, the number of frequent sets is heavily reduced and, second, the possibility appears of developing a mathematical foundation that connects closure-based mining with lattice-theoretic approaches like Formal Concept Analysis.

Data streams are defined as data sequences that arrive at high speed. They are so large that we may not be able to store all of what we see, and we do not have too much time to process each item. Several applications naturally generate data streams, a prime example being log records or click-streams in web tracking and personalization. The unlabeled rooted tree is an interesting pattern to obtain from this data. The most frequent way to deal with continuous data streams evolving on time, is to keep in memory a window of examples and refresh its model every time change is detected.

We propose a general methodology to identify closed patterns in a data stream, using Galois Lattice Theory. Using this methodology, we develop three closed tree mining algorithms: IncTreeNat, an incremental closed tree mining algorithm; WinTreeNat, a sliding window closed tree mining algorithm; and finally AdaTreeNat, an adaptive closed tree mining algorithm.

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

AdaTreeNat is a new algorithm that can adaptively mine from data streams that change over time, with no need for the user to enter parameters describing the speed or nature of the change. We take a recently proposed algorithm (ADWIN) [4] for detecting change and keeping updated statistics from a data stream, and use it as a black-box in place or counters or accumulators. Since ADWIN has rigorous performance guarantees, this opens the possibility of extending such guarantees to the new algorithm.

The rest of the paper is organized as follows. We discuss related work in Section 2. Sections 3 and 4 give background and introduce our closure operator and its properties needed for our algorithms. Section 5 introduces the general mining framework and Section 6 shows how to adapt this framework to deal with concept drift. Section 7 shows its application to tree structures. Experimental results are given in Section 8, and some conclusions in Section 9.

2. RELATED WORK

There is a large body of work done on itemset mining. An important part of the most recent work is related to data streams; see the survey [12] and the references there. We can divide these data stream methods in two different classes depending on whether they use a landmark window or a sliding window. Only a small part of these methods deal with frequent closed mining. Moment [5], CFI-Stream [13] and IncMine [11] are the state-of-art algorithms for mining frequent closed itemsets over a sliding window. CFI-Stream stores only closed itemsets in memory, but it must maintain all closed itemsets as it does not implement a minimum support threshold. Moment stores much more information besides the current frequent closed itemsets, but it has a minimum support threshold to reduce the quantity of patterns found. IncMine proposes a notion of semi-FCIs that consists in increasing the minimum support threshold for an itemset as it is retained longer in the window.

There have been subsequent efforts in moving towards closure-based mining on structured data, particularly sequences, trees and graphs. One of the differences with closed itemset mining stems from the fact that uniqueness of set-theoretic intersection no longer holds: whereas the intersection of two sets is a set, the intersection of two sequences or two trees is not one sequence or one tree. This makes it nontrivial to justify the word "closed" in terms of a standard closure operator. Many papers resort to a support-based notion of closeness of a tree or sequence [6]; others (like [1]) choose a variant of trees where a closure operator between trees can be actually defined (via least general generalization). In some cases, the trees are labeled, and strong conditions are imposed on the label patterns (such as non-repeated labels in tree siblings [16] or nonrepeated labels at all in sequences [9]). Chi et al. proposed CMTreeMiner [6], 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 [18].

A lot of research work exist on XML pattern mining. Asai et al. [2] present StreamT, a tree online mining algorithm that uses a forgetting model and is able to maintain a sliding window, but it extracts only frequent trees, not closed ones. Hsieh et al. [10] propose STMer, an alternative to StreamT to deal with frequent trees over data streams, but without using a sliding window. In [8], Feng et al. present SOLARIA*, a frequent closed XML query pattern mining algorithm, but it is not an incremental method. Li. et al [14] present Incre-FXQPMiner, an incremental mining algorithm of frequent XML query patterns, but it does not obtain the closed XML queries, neither it uses a sliding window.

As we are interested in web link structure we focus on unlabeled trees. Labeled trees are trees in which each vertex is given a unique label. Unlabeled trees are trees in which each vertex has no label, or there is a unique label for all vertices. A comprehensive introduction to the algorithms on unlabeled trees can be found in [17].

To the best of our knowledge this is the first approach defined for mining frequent closed trees in streaming data that evolve with time.

3. PRELIMINARIES

Patterns are graphs, composed by a labeled set of nodes (vertices) and a labeled set of edges. The number of nodes in a pattern is called its *size*. Examples of patterns are itemsets, sequences and trees [20].

Given two patterns t and t', we say that t is a *subpattern* of t', or t' is a *super-pattern* of t, denoted by $t \preceq t'$ if there exists a 1-1 mapping from the nodes in t to a subset of the nodes in t' that preserves node and edge labeling. As there may be many mappings with this property, we will define for each type of pattern a more specific definition of subpattern. Two patterns t, t' are said to be *comparable* if $t \preceq t'$ or $t' \preceq t$. Otherwise, they are incomparable. Also $t \prec t'$ if t is a proper subpattern of t' (that is, $t \preceq t'$ and $t \neq t'$).

The (infinite) set of all patterns will be denoted with $\mathcal{T}$, but actually all our developments will proceed in some finite subset of $\mathcal{T}$ which will act as our universe of discourse.

The input to our data mining process, now is a given finite dataset $\mathcal{D}$ of transactions, where each transaction $s \in \mathcal{D}$ consists of a transaction identifier, *tid*, and a pattern. Tids are supposed to run sequentially from 1 to the size of $\mathcal{D}$. From that dataset, our universe of discourse $\mathcal{U}$ is the set of all patterns that appear as subpattern of some pattern in $\mathcal{D}$.

Following standard usage, we say that a transaction s *supports* a pattern t if t is a subpattern of the pattern in transaction s. The number of transactions in the dataset $\mathcal{D}$ that support t is called the *support* of the pattern t. A subpattern t is called *frequent* if its support is greater than or equal to a given threshold *min_sup*. The frequent subpattern mining problem is to find all frequent subpatterns in a given dataset. Any subpattern of a frequent pattern is also frequent and, therefore, any superpattern of a nonfrequent pattern is also nonfrequent (the *antimonotonicity* property).

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

Itemsets are subsets of a set of items. Let $I = \{i_1, \cdots, i_n\}$ be a fixed set of items. All possible subsets $I' \subseteq I$ are itemsets. We can consider itemsets as patterns without edges, and without two nodes having the same label. In itemsets the notions of subpattern and super-pattern correspond to the notions of subset and superset.

Sequences are ordered list of itemsets. Let $I = \{i_1, \cdots, i_n\}$

be a fixed set of items. Sequences can be represented as $\langle(I_1)(I_2)...(I_n)\rangle$, where each I_i is a subset of I, and I_i comes before I_j if $i \leq j$. Without loss of generality we can assume that the items in each itemset are sorted in a certain order (such as alphabetic order). In sequences we are interested in a notion of subsequence defined as following: a sequence $s = \langle(I_1)(I_2)...(I_n)\rangle$ is a subsequence of $s' = \langle(I'_1)(I'_2)...(I'_n)\rangle$ i.e. $s \preceq s'$, if there exist integers $1 \leq j_1 < j_2 \ldots < j_n \leq m$ such that $I_1 \subseteq I'_{j_1}, \ldots, I_n \subseteq I'_{j_n}$.

Trees, viewed as patterns, are discussed in more detail in Section 7.

3.1 Relaxed support

Song et al.[15] introduced the concept of relaxed frequent itemset and we adapt it to pattern mining. The support space of all subpatterns can be divided into $n = \lceil 1/\epsilon_r \rceil$ intervals, where ϵ_r is a user-specified relaxed factor, and each interval can be denoted by $\mathcal{I}_i = [l_i, u_i)$, where $l_i = (n-i) * \epsilon_r \geq 0$, $u_i = (n-i+1) * \epsilon_r \leq 1$ and $i \leq n$. Then a subpattern t is called a *relaxed closed subpattern* if and only if there exists no proper superpattern t' of t such that their suports belong to the same interval $\mathcal{I}_i$.

Relaxed closed mining is a powerful notion that reduces the number of closed subpatterns in data streams where approximation is acceptable.

We can define *Relaxed support* as a mapping from all possible dataset supports to the set of relaxed intervals. We can apply it to our mining algorithms, replacing the calls to support values, to relaxed support values.

We introduce the concept of logarithmic relaxed frequent pattern, by defining $l_i = \lceil c^i \rceil$, $u_i = \lceil c^{i+1} - 1 \rceil$ for the value of c generating n intervals. Depending on the closed pattern distribution on the dataset, and the scale of supports of interest, the notion of logarithmic support may be more appropiate than the linear one.

4. CLOSURE OPERATOR ON PATTERNS

In this section we develop our approach for closed pattern mining based on the use of closure operators. Most previous approaches defined "closed" patterns in terms of support. This essentially leaves antimonotonicity as the only mathematical property to be exploited. Our approach relies on much richer mathematics, which, as usual, leads to more interesting algorithmics.

The following concept is standard in mathematics. If X is a set with a partial order $\leq$, a *closure operator* on X is a function $C : X \to X$ such that $x \leq C(X)$, $x \leq y$ implies $C(x) \leq C(y)$, and $C(X) = C(C(X))$. A Galois connection is defined by two functions, relating two lattices in a certain way. Here our lattices are plain power sets of the transactions, on the one hand, and of the corresponding subpatterns, in the other. On the basis of the binary relation $t \preceq t'$, the following definition and proposition are rather standard.

DEFINITION 1. *The Galois connection pair:*

- *For finite $A \subseteq \mathcal{D}$, $\sigma(A) = \{t \in \mathcal{T} \mid t$ maximally contained in t' for all $t' \in A\}$*
- *For finite $B \subset \mathcal{T}$, not necessarily in $\mathcal{D}$, $\tau_{\mathcal{D}}(B) = \{t' \in \mathcal{D} \mid \forall t \in B\,(t \preceq t')\}$*

PROPOSITION 1. *The composition $\Delta_{\mathcal{D}} = \sigma \circ \tau_{\mathcal{D}}$ is a closure operator on the subsets of $\mathcal{D}$.*

We point out the following easy-to-check properties:

1. $t \in \Delta_{\mathcal{D}}(\{t\})$
2. $\Delta_{\mathcal{D}_1 \cup \mathcal{D}_2}(\{t\}) = \{t_1 \cap t_2 \mid t_1 \in \Delta_{\mathcal{D}1}(\{t\}), t_2 \in \Delta_{\mathcal{D}2}(\{t\})\}$

We can relate the closure operator to the notion of closure based on support, as previously defined, as follows: t is closed for $\mathcal{D}$ if and only if: $\Delta_{\mathcal{D}}(\{t\}) = \{t\}$.

PROPOSITION 2. *Adding a pattern transaction to a dataset of patterns $\mathcal{D}$ does not decrease the number of closed patterns for $\mathcal{D}$.*

PROOF. All previously closed patterns remain closed. A closed pattern will become unclosed if one of its superpatterns reach the same support, but that is not possible because every time the support of a pattern increases, the support of all its subpatterns also increases. □

PROPOSITION 3. *Adding a transaction with a closed pattern to a dataset of patterns $\mathcal{D}$ does not modify the number of closed patterns for $\mathcal{D}$.*

PROOF. Suppose s is a subpattern of a closed pattern t. If s is closed then $\Delta_{\mathcal{D}}(\{s\}) = \{s\}$. If s is not closed, then $\Delta_{\mathcal{D}}(\{s\}) \subset \Delta_{\mathcal{D}}(\{t\}) = \{t\}$. Increasing the support of the closed pattern t will increase the support of all its subpatterns. The subpatterns that are closed will remain closed, and the ones that are non-closed, will remain non-closed because the support of its closure will increase also. □

PROPOSITION 4. *Deleting a pattern transaction from a dataset of patterns $\mathcal{D}$ does not increase the number of closed patterns for $\mathcal{D}$.*

PROOF. All the previous unclosed patterns remain unclosed. A condition for an unclosed pattern to become closed is that its superpatterns with the same support modifies their support, but this is not possible because every time we decrease the support of a superpattern we decrease also the support of this pattern. □

PROPOSITION 5. *Deleting a pattern transaction that is repeated in a dataset of patterns $\mathcal{D}$ does not modify the number of closed patterns for $\mathcal{D}$.*

PROOF. Adding a transaction with a previously closed pattern to a dataset of patterns $\mathcal{D}$ does not modify the number of closed patterns for $\mathcal{D}$. So deleting it does not change the number of closed patterns. □

PROPOSITION 6. *Let $\mathcal{D}1$ and $\mathcal{D}2$ be two datasets of patterns. A pattern t is closed for $\mathcal{D}1 \cup \mathcal{D}2$ if and only if $\Delta_{\mathcal{D}_1 \cup \mathcal{D}_2}(\{t\}) = \{t_1 \cap t_2 \mid t_1 \in \Delta_{\mathcal{D}1}(\{t\}), t_2 \in \Delta_{\mathcal{D}2}(\{t\})\} = \{t\}$.*

Proposition 6 follows from the definition of closed pattern. We use it as a closure checking condition when adding a set of transactions to a dataset of patterns.

COROLLARY 1. *Let $\mathcal{D}1$ and $\mathcal{D}2$ be two datasets of patterns. A pattern t is closed for $\mathcal{D}1 \cup \mathcal{D}2$ if and only if*

- *t is a closed pattern for $\mathcal{D}1$, or*
- *t is a closed pattern for $\mathcal{D}2$, or*

```
CLOSED_SUBPATTERN_MINING_ADD(T_1, T_2, min_sup, T)
    Input: Frequent closed pattern sets T_1 and T_2, and min_sup.
    Output: The frequent closed pattern set T.
 1  T ← T_1
 2  for every t in T_2 in size-ascending order
 3       do if t is closed in T_1
 4            do support_T(t) += support_{T_2}(t)
 5               for every t' that is a subpattern of t
 6                    do if t' is in T_1
 7                         then if t' is not updated
 8                              then insert t' into T
 9                                   support_T(t') += support_{T_2}(t')
10                              else
11                                   skip processing t' and all its subpatterns
12       do if t is not closed in T_1
13            do insert t into T
14               for every t' that is a subpattern of t
15                    do if t' is not updated
16                         then if t' is in T_1
17                              then support_T(t') += support_{T_2}(t')
18                              if {s ∩ t | s ∈ Δ_{T1}({t'})} = {t'}
19                              then insert t' into T
20                                   support_T(t') += support_{T_2}(t')
21                         else
22                              skip processing t' and all its subpatterns
23         return T
```

Figure 1: The Closed Subpattern Mining adding algorithm

- *t is a subpattern of a closed pattern in $\mathcal{D}1$ and a closed pattern in $\mathcal{D}2$ and $\Delta_{\mathcal{D}_1 \cup \mathcal{D}_2}(\{t\}) = \{t\}$*

PROPOSITION 7. *Let $\mathcal{D}$ be a dataset of patterns. A pattern t is closed for $\mathcal{D}$ if and only if the intersection of all its closed superpatterns is t.*

PROOF. Suppose that the intersection of all its closed superpatterns is t' and that $t' \neq t$, then t is not closed because it exists a superpattern t' with the same support. Also, suppose the intersection of all its closed superpatterns is t and that t is not closed. Then $t' \in \Delta(\{t\})$ has the same support as t, and it must be in the intersection of all the closed superpatterns of t. □

We use Proposition 8 as a closure checking condition when deleting a set of transactions from a pattern set.

5. CLOSED PATTERN MINING

5.1 Incremental Closed Pattern Mining

In this subsection we propose a new method to do incremental closed pattern mining. Every time a new batch of patterns $\mathcal{D}_{T2}$ arrives we compute the closed pattern set of the batch $\mathcal{D}_{T2}$, and then we update the closed pattern set T using CLOSED_SUBPATTERN_MINING_ADD as is shown in Figure 1.

In words, let T be the existing set of closed patterns, and T_2 those coming from the new batch. For each closed pattern in $\mathcal{D}_{T2}$, we check whether the pattern is closed in T. If it is closed, we update its support and the support of all its subpatterns, as justified by Proposition 3. If it is not closed, as it is closed for $T2$, we add it to the closed pattern set, as justified by Corollary 2, and we check for each of its subpatterns whether it is closed or not. In line 18, we use Proposition 6 to do the closure-check $\Delta_{T1 \cup T2}(\{t'\}) = \{t_1 \cap t_2 \mid t_1 \in \Delta_{T1}(\{t'\}), t_2 \in \Delta_{T2}(\{t'\})\} = \{t'\}$ using the fact that $\Delta_{T2}(\{t'\}) = \{t\}$. Here $\Delta_{T2}(\{t'\})$ is a closed pattern in $T2$. As we check all the subpatterns of $T2$ in size-ascending order, we know that all closed subpatterns of t have been checked before, and therefore we can suppose that $\Delta_{T2}(\{t'\}) = \{t\}$.

The best (most efficient) data structure to do this task will depend on the pattern. In general, a lattice is the default option, where each lattice node is a pattern with its support, and a list of its closed superpatterns and a list of its closed subpatterns: We can use the lattice structure to speed up the closure check $\Delta_{T1 \cup T2}(\{t'\}) = \{t'\}$.

5.2 Closed pattern mining over a sliding window

Adding a method to delete a set of transactions, we can adapt our method to use a sliding window of pattern transactions.

Figure 2 shows the CLOSED_SUBPATTERN_MINING_DELETE pseudocode. We check for every t pattern in T_2 in ascending order if its subpatterns are still closed or not after deleting some transactions. We can look for a closed superpattern with the same support or use the closure checking condition given by Proposition 8: a pattern t is closed if the intersection of all its closed superpatterns is t. The lattice structure supports this operation well. We can delete a transaction one by one, or delete a batch of transactions of the slid-

```
CLOSED_SUBPATTERN_MINING_DELETE(T1, T2, min_sup, T)
    Input: Frequent Closed pattern sets T1 and T2, and min_sup.
    Output: The frequent closed pattern set T.
 1  T ← T1
 2  for every t in T2 in size-ascending order
 3        do for every t' that can be reduced from t
 4              do if t' is not updated
 5                    then if t' is in T1
 6                          then if t' is not closed
 7                                then delete t' from T
 8                                else support_T(t') −= support_T2(t')
 9                    else
10                         skip processing t' and all its subpatterns
11  return T
```

Figure 2: The Closed Subpattern Mining delete algorithm

ing window. We delete transactions one by one to avoid recomputing the frequent closed patterns of each batch of transactions.

6. ADDING CONCEPT DRIFT

In this section we present a new method for dealing with concept drift in pattern mining, using ADWIN [4], an algorithm for detecting change and dynamically adjusting the length of a data window. First we briefly review the ADWIN algorithm and then we describe our method combining the previous sliding window pattern mining algorithms and ADWIN.

6.1 The ADWIN algorithm

Recently, we proposed an algorithm termed ADWIN (for Adaptive Windowing) that solves in a well-specified way the problem of tracking the average of a stream of bits or real-valued items. ADWIN keeps a variable-length window of recently seen items, with the property that the window has at all times the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window".

More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: one, that change reliably declared whenever the window shrinks; and two, that at any time the average over the existing window can be reliably taken as an estimation of the current average in the stream (barring a very small or very recent change that is still not statistically visible). A formal and quantitative statement of these two points (a theorem) appears in [4].

ADWIN is parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, inherent to all algorithms dealing with random processes.

Also important for our purposes, ADWIN does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique in [7]. In particular, it keeps a window of length W using only $O(\log W)$ memory rather than the $O(W)$ one expects from a naïve implementation. The processing time per item is also $O(\log W)$.

6.2 Concept drift closed pattern mining

We propose two strategies to deal with concept drift:

1. Using a sliding window, with an ADWIN estimator deciding the size of the window
2. Maintaining an ADWIN estimator for each closed set in the lattice structure.

In both strategies we use CLOSED_SUBPATTERN_MINING_ADD to add transactions. In the first strategy we use CLOSED_SUBPATTERN_MINING_DELETE to delete transactions as we maintain a sliding window of transactions.

In the second strategy, we do not delete transactions. Instead, each ADWIN monitors its support and when a change is detected, then the support may

- increase: the number of closed patterns is increasing and it is maintained by CLOSED_SUBPATTERN_MINING_ADD
- decrease: the number of closed patterns may decrease and we have to delete the non-closed patterns from the lattice. We do this in the following way:
  - If the support is lower than min_supp, we delete the closed pattern from the lattice.
  - If the support is higher than min_supp, we check whether it and all its subpatterns are still closed finding a superpattern with the same support, or, alternatively, we can use the closure checking of Proposition 8: a pattern t is closed if the intersection of all its closed superpatterns is t.

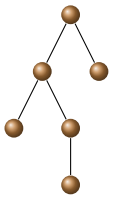
7. CLOSED TREE MINING

In this section we apply the general framework above specifically by considering the tree pattern. *Trees* are connected acyclic graphs, *rooted trees* are trees with a vertex singled out as the root, and *unranked* trees are trees with unbounded arity. We say that $t_1, \ldots, t_k$ are the *components* of tree t if t is made of a node (the root) joined to the roots of all the t_i's. We can distinguish betweeen the cases where the components at each node form a sequence (ordered trees) or

just a set (*unordered trees*). We will deal with rooted, unranked trees. We do not assume the presence of labels on the nodes.

An *induced subtree* of a tree t is any connected subgraph rooted at some node v of t that its vertices and edges are subsets of those of t. An *embedded subtree* of a tree t is any connected subgraph rooted at some node v of t that does not break the ancestor-descendant relationship among the vertices of t. We are interested in induced subtrees. Formally, let s be a rooted tree with vertex set V' and edge set E', and t a rooted tree t with vertex set V and edge set E. Tree s is an *induced subtree* (or simply a *subtree*) of t (written $t' \preceq t$) if and only if 1) $V' \subseteq V$, 2) $E' \subseteq E$, and 3) the labeling of V' is preserved in t. This notation can be extended to sets of trees $A \preceq B$: for all $t \in A$, there is some $t' \in B$ for which $t \preceq t'$.

We represent each tree using natural representations [3]. The *natural representation* of a tree is a sequence over a countably infinite alphabet, namely, the set of natural numbers. This encoding basically corresponds to a preorder 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



is the natural sequence $(0, 1, 2, 2, 3, 1)$. Note that, for example, the subsequence $(1, 2, 2, 3)$ corresponds to the bottom-up subtree rooted at the left son of the root.

The input to our data mining process is a given finite dataset $\mathcal{D}$ of transactions, where each transaction $s \in \mathcal{D}$ consists of a transaction identifier, *tid*, and an unlabeled rooted tree. Figure 3 shows a finite dataset example.

The closure operator defined for trees uses the following Galois connection pair:

- For finite $A \subseteq \mathcal{D}$, $\sigma(A) = \{t \in \mathcal{T} \mid t \text{ maximally contained in } t' \text{ for all } t' \in A\}$
- For finite $B \subset \mathcal{T}$, not necessarily in $\mathcal{D}$, $\tau_{\mathcal{D}}(B) = \{t' \in \mathcal{D} \mid \forall t \in B\,(t \preceq t')\}$.

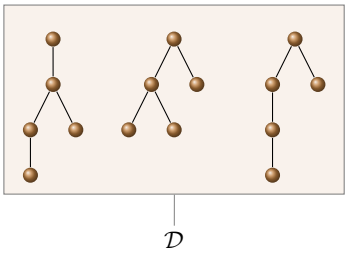
The main results of Section 4 may be established for unlabeled trees as:

COROLLARY 2. *Let $\mathcal{D}1$ and $\mathcal{D}2$ be two datasets of trees. A tree t is closed for $\mathcal{D}1 \cup \mathcal{D}2$ if and only if*

- *t is a closed tree for $\mathcal{D}1$, or*
- *t is a closed tree for $\mathcal{D}2$, or*
- *t is a subtree of a closed tree in $\mathcal{D}1$ and a closed tree in $\mathcal{D}2$ and $\Delta_{\mathcal{D}_1 \cup \mathcal{D}_2}(\{t\}) = \{t\}$.*

PROPOSITION 8. *Let $\mathcal{D}$ be a dataset of trees. A tree t is closed for $\mathcal{D}$ if and only if the intersection of all its closed supertrees is t.*

The closed trees for the dataset of Figure 3 are shown in the Galois lattice of Figure 4.



Tid	Trees
1	$(0, 1, 2, 3, 2)$
2	$(0, 1, 2, 2, 1)$
3	$(0, 1, 2, 3, 1)$

Figure 3: A dataset example

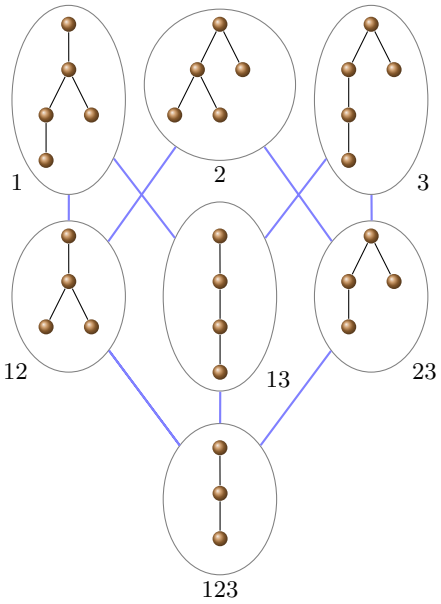


Figure 4: Example of Galois Lattice of Closed trees

7.1 Non Incremental Closed Tree Mining

In [3] the authors presented an algorithm for computing frequent and closed trees from a dataset of trees, in a non-incremental way. They represent the potential subtrees to be checked as frequent and closed on the dataset in such a way that extending them by one single node, in all possible ways, corresponds to a clear and simple operation on the representation. The completeness of the procedure is assured, that is, all trees can be obtained in this way. This allows them to avoid extending trees that are found to be already nonfrequent.

The pseudocode of this method, CLOSED_SUBTREES MINING, is presented in Figures 5 and 6. Note that the first line of the algorithm is a canonical representative checking, a check that is used frequently in tree mining literature. In [3] the authors selected one of the ordered trees corresponding to a given unordered tree to act as a canonical representative: by convention, this canonical representative has larger trees always to the left of smaller ones.

7.2 Incremental Closed Tree Mining

We propose three tree mining algorithms adapting the

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

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

Figure 5: The Closed Subtree Mining algorithm

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

1   t ← •
2   T ← ∅
3   T ← CLOSED_SUBTREE_MINING(t, D, min_sup, T)
4   return T
```

Figure 6: The Closed Unordered Mining algorithm

general framework for patterns presented in Section 5:

- IncTreeNat, an incremental closed tree mining algorithm,
- WinTreeNat, a sliding window closed tree mining algorithm
- AdaTreeNat an adaptive closed tree mining algorithm

The batches are processed using the non-incremental algorithm explained in Subsection 7.1. We use the relaxed closed tree notion to speed up the mining process.

8. EXPERIMENTAL EVALUATION

We tested our algorithms on synthetic and real data, comparing the results with CMTreeMiner [6].

All experiments were performed on a 2.0 GHz Intel Core Duo PC machine with 2 Gigabyte main memory, running Ubuntu 7.10. As far as we know, CMTreeMiner is the state-of-art algorithm for mining induced frequent closed trees in databases of rooted trees. CMTreeMiner and our algorithms are implemented in C++. The main difference with our approach is that CMTreeMiner is not incremental and works with labeled nodes, and we deal with unlabeled trees.

On synthetic data, we use the same dataset as in [6] and [19] for rooted ordered trees restricting the number of distinct node labels to one. We call this dataset TN1, and is generated by the tree generation program of Zaki [19] available from his web page. This program generates a mother

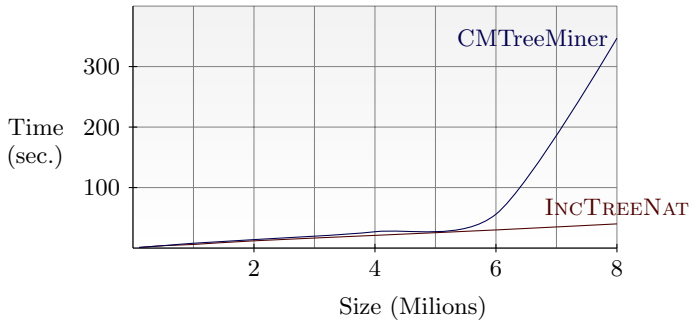


Figure 7: Data experimental time results on ordered trees on TN1 dataset

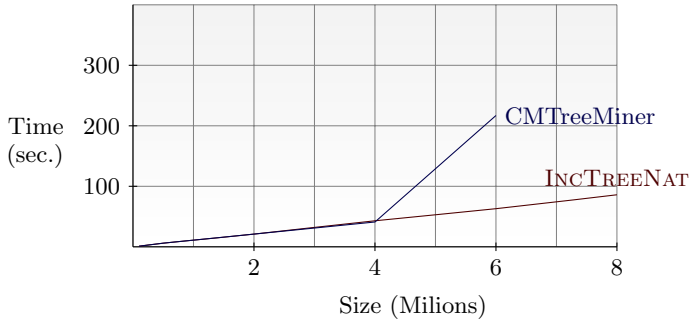


Figure 8: Time used on unordered trees, TN1 dataset

tree that simulates a master website browsing tree. Then it assigns probabilities of following its children nodes, including the option of backtracking to its parent, such that the sum of all the probabilities is 1. Using the master tree, the dataset is generated creating subtrees by randomly picking subtrees according to these probabilities.

In the TN1 dataset, the parameters are the following: the number of distinct node labels is $N = 1$, the total number of nodes in the tree is $M = 10,000$, the maximal depth of the tree is $D = 10$, the maximum fanout is $F = 10$. The average number of nodes is 3.

The results of our experiments on synthetic data are shown in Figures 7, 8, 9 and 10. We changed the dataset size from $100,000$ to 8 milion, and we observe that as the dataset size increases, IncTreeNat time increases linearly, and CMTreeMiner does much worse than IncTreeNat. After 6 milion samples, in the unordered case, CMTreeMiner runs out of main memory and it ends before outputing the closed trees.

Figure 11 shows the result of the second following experiment: we take a TN1 dataset of 2 milion trees, and we introduce artifical concept drift changing the dataset trees from sample 500,000 to 1,000,000 and from 1,500,000 to 2,000,000, in order to have a small number of closed trees. We compare IncTreeNat , WinTreeNat with a sliding window of $500,000$ and $1,000,000$, and with AdaTreeNat. We observe that AdaTreeNat detects change faster, and it quickly revises the number of closed trees in its output. On the other hand, the other methods have to retain all the

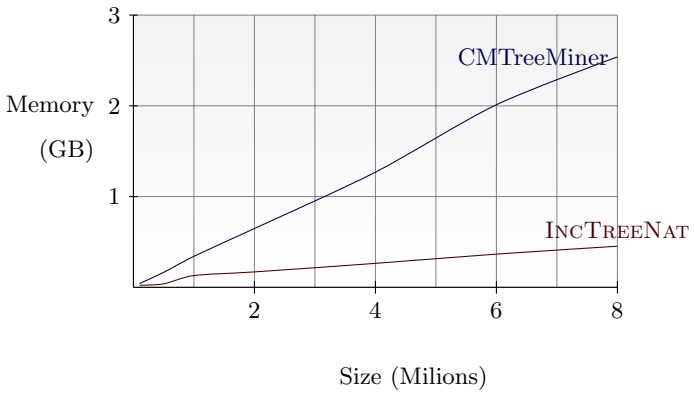


Figure 9: Data experimental memory results on ordered trees on TN1 dataset

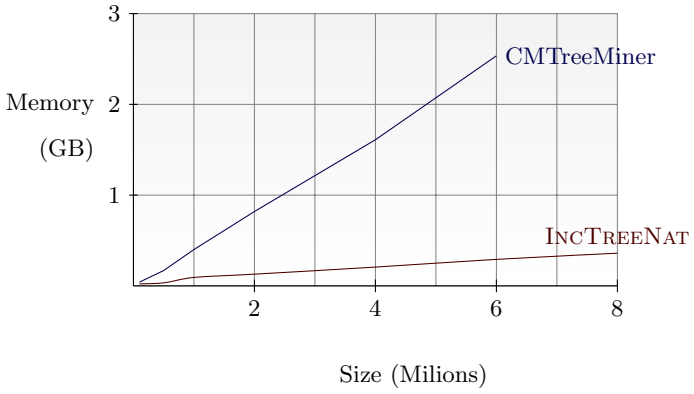


Figure 10: Memory used on unordered trees, TN1 dataset

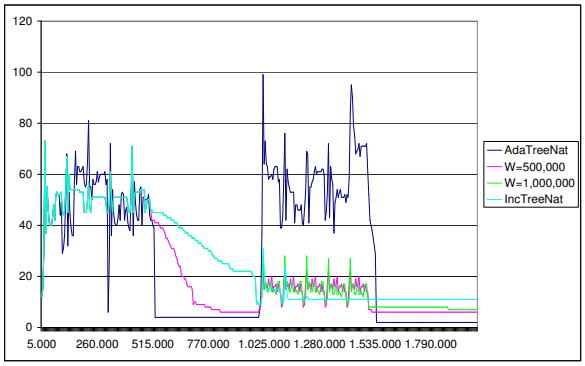


Figure 11: Number of closed trees detected with artificial concept drift introduced

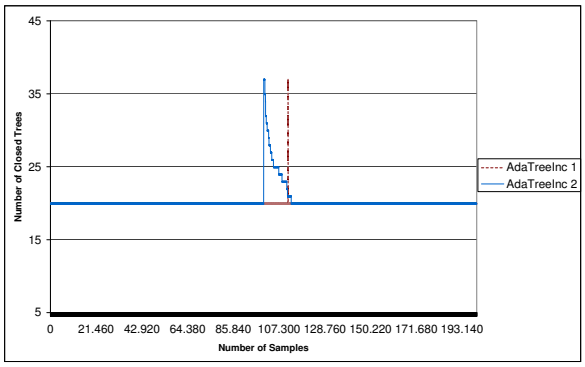


Figure 12: Number of closed trees maintaining the same number of closed datasets on input data

data stored in its window, and they need more samples to change its output number of closed trees.

To compare the two adaptive methods, we perform a third experiment. We use a data stream of 200,000 trees, with a static distribution of 20 closed trees on the first 100,000 trees and 20 different closed trees on the last 100,000 trees. The number of closed trees remains the same. Figure 12 shows the difference between the two methods. The first one, which monitors the number of closed trees, detects change at sample 111,480 and then it reduces the window size immediately. In the second method there are `ADWIN`s monitoring each tree support; they notice the appearance of new closed trees quicker, but overall the number of closed trees decreases more slowly than in the first method.

Finally, we tested our algorithms on the CSLOGS Dataset, available from Zaki's web page [19]. It consists of web logs files collected over one month at the Department of Computer Science of Rensselaer Polytechnic Institute. The logs touched 13,361 unique web pages, and the CSLOGS dataset contains 59,691 trees. The average tree size is 12.

Figure 13 shows the number of closed trees detected on the CSLOGS dataset, varying the number of relaxed intervals. We see that on this dataset support values are distributed in such a way that the number of closed trees using logarithmic relaxed support is greater than using linear relaxed support. When the number of intervals is greater than 1,000 the number of closed intervals is 249, the number obtained using the classical notion of support.

9. CONCLUSIONS

We have presented efficient algorithms for mining ordered and unordered frequent unlabeled closed trees on evolving data streams. If the distribution of the tree dataset is stationary, the best method to use is IncTreeNat, as we do not need to delete any past transaction. If the distribution may evolve, then a sliding window method is more appropiate. If we know which is the right size of the sliding window, then we can use WinTreeNat, otherwise AdaTreeNat would be a better choice, since it does not need the window size parameter.

Future work will be to do more experiments varying other tree parameters, and comparing it to other incremental meth-

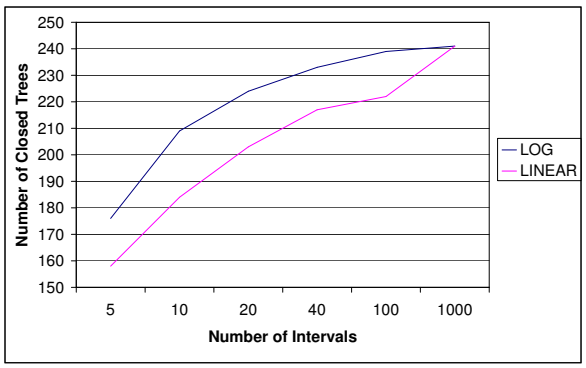


Figure 13: Number of closed trees detected on CSLOGS dataset varying Number of relaxed intervals

ods as StreamT, if they are available. Most importantly, we want to apply the methodology explained in this paper to labeled trees, a pattern that has more applications than unlabeled trees, and in particula compare our methodology with CMTreeMiner using the same type of trees.

10. ACKNOWLEDGMENTS

Partially supported by the EU PASCAL2 Network of Excellence, and by the DGICYT MOISES-BAR project, TIN2005-08832-C03-03. Albert Bifet is supported by a Formació d'Investigadors (FI) grant through the Grups de Recerca Consolidats (SGR) program of Generalitat de Catalunya.

The authors would like to thank José L. Balcázar for his unconditional support and inspiring ideas and discussions.

11. REFERENCES

[1] H. Arimura and T. Uno. An output-polynomial time algorithm for mining frequent closed attribute trees. In *ILP*, pages 1–19, 2005.

[2] T. Asai, H. Arimura, K. Abe, S. Kawasoe, and S. Arikawa. Online algorithms for mining semi-structured data stream. In *IEEE International Conference on Data Mining (ICDM'02)*, page 27, 2002.

[3] J. L. Balcázar, A. Bifet, and A. Lozano. Mining frequent closed unordered trees through natural representations. In *ICCS 2007, 15th International Conference on Conceptual Structures*, pages 347–359, 2007.

[4] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.

[5] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM'04)*, November 2004.

[6] Y. Chi, Y. Xia, Y. Yang, and R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *Fundamenta Informaticae*, XXI:1001–1038, 2001.

[7] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 14(1):27–45, 2002.

[8] J. Feng, Q. Qian, J. Wang, and L.-Z. Zhou. Efficient mining of frequent closed xml query pattern. *J. Comput. Sci. Technol.*, 22(5):725–735, 2007.

[9] G. C. Garriga and J. L. Balcázar. Coproduct transformations on lattices of closed partial orders. In *ICGT*, pages 336–352, 2004.

[10] M. C.-E. Hsieh, Y.-H. Wu, and A. L. P. Chen. Discovering frequent tree patterns over data streams. In *SDM*, 2006.

[11] Y. K. James Cheng and W. Ng. Maintaining frequent closed itemsets over a sliding window. *Journal of Intelligent Information Systems*, 2007.

[12] Y. K. James Cheng and W. Ng. A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems*, 2007.

[13] N. Jiang and L. Gruenwald. CFI-Stream: mining closed frequent itemsets in data streams. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 592–597, 2006.

[14] H.-F. Li, M.-K. Shan, and S.-Y. Lee. Online mining of frequent query trees over xml data streams. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 959–960, 2006.

[15] G. Song, D. Yang, B. Cui, B. Zheng, Y. Liu, and K. Xie. CLAIM: An efficient method for relaxed frequent closed itemsets mining over stream data. In *DASFAA*, pages 664–675, 2007.

[16] A. Termier, M.-C. Rousset, and M. Sebag. DRYADE: a new approach for discovering closed frequent trees in heterogeneous tree databases. In *ICDM*, pages 543–546, 2004.

[17] G. Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.

[18] X. Yan and J. 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.

[19] M. J. Zaki. Efficiently mining frequent trees in a forest. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[20] M. J. Zaki, N. Parimi, N. De, F. Gao, B. Phoophakdee, J. Urban, V. Chaoji, M. A. Hasan, and S. Salem. Towards generic pattern mining. In *ICFCA*, pages 1–20, 2005.