An Output-Polynomial Time Algorithm for Mining Frequent Closed Attribute Trees

Hiroki Arimura[1,⋆] and Takeaki Uno[2]

[1] Hokkaido University, Kita 14-jo, Nishi 9-chome, Sapporo 060-0814, Japan
arim@i.kyushu-u.ac.jp
[2] National Institute of Informatics, Tokyo 101–8430, Japan
uno@nii.jp

Abstract. Frequent closed pattern discovery is one of the most important topics in the studies of the compact representation for data mining. In this paper, we consider the *frequent closed pattern discovery problem* for a class of structured data, called *attribute trees* (AT), which is a subclass of labeled ordered trees and can be also regarded as a fragment of description logic with functional roles only. We present an efficient algorithm for discovering all frequent closed patterns appearing in a given collection of attribute trees. By using a new enumeration method, called the *prefix-preserving closure extension*, which enable efficient depth-first search over all closed patterns without duplicates, we show that this algorithm works in polynomial time both in the total size of the input database and the number of output trees generated by the algorithm. To our knowledge, this is one of the *first result for output-sensitive algorithms* for frequent closed substructure disocvery from trees and graphs.

Keywords: frequent closed pattern mining, tree mining, attribute tree, description logic, semi-structured data, the least general generalization, closure operation, output-sensitive algorithm.

1 Introduction

Frequent closed pattern discovery [19] is the problem of finding all the frequent closed patterns in a given data set, where *closed patterns* are the maximal patterns among each equivalent class that consists of all frequent patterns with the same occurrence sets in a tree database. It is known that the number of frequent closed patterns is much smaller than that of frequent patterns on most realworld datasets, while the frequent closed patterns still contain the complete information of the frequency of all frequent patterns. Closed pattern discovery is useful to increase the performance and the comprehensivity in data mining.

On the other hand, rapid growth of semi-structured data [1] such as HTML and XML data enabled us to accumulate a massive amount of weakly structured data on the networks. There is a potential demand for efficient methods

⋆ Present address: LIRIS, University Claude-Bernard Lyon 1, France.

S. Kramer and B. Pfahringer (Eds.): ILP 2005, LNAI 3625, pp. 1–19, 2005.

for extracting useful patterns from these semi-structured data, so called *semi-structured data mining*. For the last years, a number of researches on efficient algorithms for semi-structured data mining have been done for ordered trees [3,27], unordered trees [4,11,15,22] , and general graphs [14,26]. Presently, one of the major topics in semi-structured data mining is so-called *closed tree mining*, an extension of closed pattern mining framework to semi-structured data [11,23,26].

In this paper, we consider the frequent closed pattern discovery problem for a class of structured data, called *attribute trees* ($\mathcal{AT}$), which is a subclass of labeled ordered trees and can be also regarded as a fragment of description logic [9] with functional roles only. We present an efficient algorithm for discovering all frequent closed patterns appearing in a given collection of attribute trees.

Most of the present closed tree mining algorithms adopted an approach that combines fast enumeration of frequent patterns and explicit checking of its maximality [11,23,26]. Unfortunately, this approach does not yield any efficient algorithms with theoretical performance guarantee, in terms of output-sensitive algorithms or enumeration algorithm. To overcome this problem, we developed a new enumeration technique, called the *prefix-preserving closure expansion*, which is originally introduced to frequent closed itemset discovery by Uno *et al.* [24], with combining the notions of the rightmost expansion [3,18,27] and the least general generalization [20] for trees.

Based on these techniques, we present an efficient algorithm CLOATT (Closed Attribute Tree Miner) that enumerates all frequent closed attribute trees in a given collection of attribute trees without duplicates in polynomial time per closed tree in the total size n of the database using a small amount of memory space that only depends on n. The key of the algorithm is a tree-shaped search space generated by the prefix-preserving closure expansion, that enables us to make efficient enumeration using depth-first search of closed patterns, without storing any of the previously discovered patterns for maximality check.

To the best of our knowledge, this is one of the first results on output-polynomial time closed pattern miners for structured objects. Hence, this is a first step towards efficient closed pattern discovery for general structured objects including trees and graphs.

Related Works: Termier *et al.* [23] recently considered the frequent closed tree discovery problem for a class of trees with same constraint as attribute trees in $\mathcal{AT}$. Though they presented an efficient algorithm using an interesting idea of *hooking*, its output-sensitive complexity is not yet analyzed. Cumby and Roth [13] presented a framework for learning and inference with relation data using a fragment of description logic, called *feature description logic*, which is similar to the class $\mathcal{AT}$ of attribute trees considered in this paper. However, the focus is on the knowledge representation issues in complex structural data domains, and closed pattern discovery is not considered [13]. Wang and Liu [25] studied the frequent tree discovery problem for the class of sets of paths from a given collection of labeled trees, which is closely related to frequent discoverry problem for the class AT.

Organization of This Paper: The rest of this paper is organized as follows. In Section 2, we give basic notion and definitions on attribute trees and closed patterns. In Section 3, we give a characterization of closed trees in terms of least general generalization. In Section 4, we develop the ppc-expansion (prefix-preserving expansion) and then present an output-polynomial time algorithm for frequent closed attribute trees. In Section 4, we show an experimental result, and in Section 5, we conclude.

2 Preliminaries

In this section, we introduce basic definitions on the class of ranked trees and closed tree discovery.

For a set A, $|A|$ denotes the cardinality of A and $\varepsilon \in A^*$ denotes the *empty sequence* of length zero. We denote by A^* and $A^+ = A^* \backslash \{\varepsilon\}$, respectively, the sets of all finite sequences and all non-empty finite sequences over A. For sequences $\alpha, \beta \in A^*$, we denote by $\alpha\beta$ the concatenation of α, and β and by $|\alpha|$ the length of α. If $\alpha\gamma = \beta$ holds for some possibly empty sequence $\gamma \in A^*$ then we say that α is a *prefix* of β. Furthermore, if γ is not the empty sequence then the prefix α is said to be *proper*. For a binary relation $R \subseteq R^2$ over a set X, R^+ denotes the *transitive closure* relation of R.

2.1 Attribute Trees

In this subsection, we model semi-structured data by a special type of labeled rooted trees, called attribute trees.

Let $\mathcal{A} = \{a_0, a_1, a_2, \ldots\}$ be a countable set of *labels* associated with a total order $\leq$ over $\mathcal{A}$. Sometimes, we call the elements of $\mathcal{A}$ *attributes* or *value*, too. For simplicity, we use a single alphabet A, and think of the labels at internal nodes and leaves as the encodings of attributes and values, respectively, as in [10]. Throughout this paper, we assume without loss of generality that $\mathcal{A}$ is the set of all nonnegative integers $\mathcal{A} = \{0, 1, 2, \ldots\}$ and $\leq$ is the partial order over integers.

Definition 1. *Let $\mathcal{A}$ is an alphabet of labels. An* attribute tree on $\mathcal{A}$ *(*tree*, for short) is a rooted, node labeled, directed acyclic graph $T = (V, E, r, label)$, where*

1. *The set $V = \{v_1, \ldots, v_n\}$ $(n \geq 0)$ is a finite set of* nodes.
2. *The set $E \subseteq V \times V$ is a finite set of* edges. *If $(u, v) \in E$ then we say that either u is the* parent *of v or v is a* child *of u.*
3. *The node $r \in V$ is a distinguished node, called the* root. *Any node v except r has exactly one parent.*
4. *The function label : $V \to \mathcal{A}$ is a* labeling function *for assigning a label label(v) to each node v of T.*
5. *For every label $a \in \mathcal{A}$, each node $v \in V$ has at most one child w labeled by a. Then, the unique node w is called the a-*child *of v.*

We assume that $V_T = \{1, \ldots, n\}$ and identify the isomorphic patterns. The *size* of T, denoted by $|T|$, is defined by the number $|V|$ of the nodes in T. Let $u, v \in V$. If $(u, v) \in (E)^+$ then we say that either u is an *ancestor* of v or v

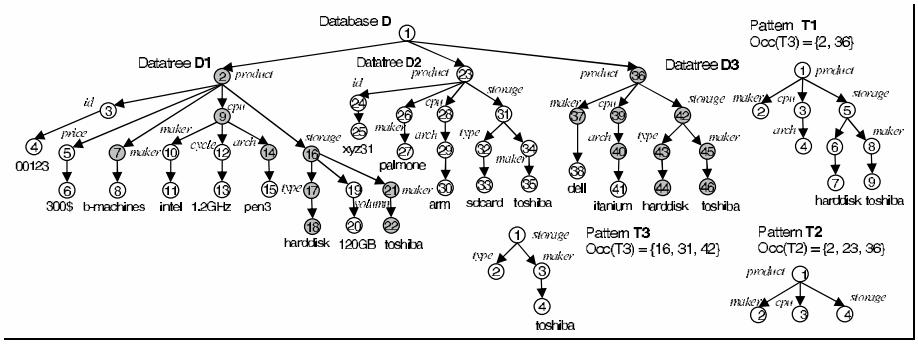

Fig. 1. An example of a tree database and patterns in attribute trees, where each circle indicates a node, each number in a circle indicates the node number, and each name in italic or bold face next to a circle indicates a node label. For instance, the pattern T_1 occurs in the database at positions 2 and 36.

is a *descendant* of u, where (E^+) is the transitive closure of E. A *path* in T is a sequence of nodes $\pi = (v_1, \ldots, v_d)$, $d \geq 0$, such that $(v_i, v_{i+1}) \in E$ for every $i = 1, \ldots, d-1$ and its *length* is the number of its nodes $|\pi| = d$. The *depth* of node v is the length of the unique path from the root to v. Other notions on trees such as height can be found in a standard textbook, e.g., [2]. We denote by $\mathcal{AT}$ the *class of all attribute trees* over $\mathcal{A}$. In what follows, for an attribute tree $T = (V, E, r, \mathit{label})$, we refer to $V, E, \leq, r, \mathit{label}$ as V_T, E_T, r_T, and label_T, respectively if it is clear from context.

In Fig. 1, we show an example of attribute trees, where nodes are numbered in the preorder (as ordered trees), and each label is used to represents either an *attribute* (in italic face) or a *value* (in block face). These labels can be used to represent edge labels, too, since we deal with trees only.

2.2 Tree Matching Relation

The semantics of attribute trees is given by the *matching functions* as follows [3,8,16,17].

Definition 2. Let S and $T \in \mathcal{AT}$ be attribute trees over $\mathcal{A}$. Then, S *matches* T, denoted by $S \sqsubseteq T$, if there exists some function $\varphi : V_S \to V_T$ that satisfies the following conditions (i)–(iv) for any $v, v_1, v_2 \in V_S$.

(i) φ is a *one-to-one mapping*: $v_1 \neq v_2$ implies $\varphi(v_1) \neq \varphi(v_2)$.
(ii) φ *preserves the parent-child relation*: $(v_1, v_2) \in E_S$ iff $(\varphi(v_1), \varphi(v_2)) \in E_T$.
(iii) φ *preserves the node labels*: $\mathit{label}_S(v) = \mathit{label}_T(\varphi(v))$.

The function φ is called a *matching function* from S to T.[1] We denote by $\Phi(S, T)$ the set of all matching function from S to T.

[1] In Kilpelainen and Mannila [16], φ is called a path inclusion since it preserves the parent-child relationship. The fuction φ is called an *embedding*.

Example 1. In the example of Fig. 1, the tree T_2 with node set $V_{T_2} = \{1, 2, 3, 4\}$ occurs in the data tree $\mathcal{D}$ with matching functions $\varphi_1 = (2, 7, 9, 16)$, $\varphi_2 = (23, 26, 28, 31)$, and $\varphi_3 = (36, 37, 39, 42)$, where each φ is represented by tuple of its images $(\varphi(1), \varphi(2), \varphi(3), \varphi(4))$.

If $S \sqsubseteq T$ holds, then we also say that S *occurs* in T, S *is included* by T, or S *subsumes* T. If $S \sqsubseteq T$ and $T \not\sqsubseteq S$ then we define $S \sqsubset T$ and say that S is *properly included* in T or S *properly subsumes* T. For convention, we assume a special tree $\perp$ of size 0, called the *empty tree*, such that $\perp \sqsubseteq T$ for every $T \in \mathcal{AT}$.

Lemma 1. *The subsumption relation $\sqsubseteq$ is a partial order over $\mathcal{AT}$.*

The *matching problem* w.r.t. $\sqsubseteq$ is the problem to decide if a pattern tree P matches a data tree D, i.e., $P \sqsubseteq D$ holds.

Lemma 2. *The matching problem w.r.t. $\sqsubseteq$ is computable in $O(mn)$ time for attribute trees, where m and n are the sizes of a pattern tree and a data trees.*

2.3 Databases, Patterns, Denotations, and Closed Patterns

Let $\mathcal{D} = \{D_1, \ldots, D_m\}$ be a *tree database* (*database*, for short), where each $D_i \in \mathcal{AT}$ is an attribute tree, called a *data tree*, and the node sets $V_{D_1}, \ldots, V_{D_m}$ are mutually disjoint. In the later sections, we often identify $\mathcal{D}$ as a single database tree with a virtual master root v_0 labeled by a null label.[2] We define the domain and the size of $\mathcal{D}$ by $V_{\mathcal{D}} = \bigcup_i V_{D_i}$ and $||\mathcal{D}|| = |V_{\mathcal{D}}|$, respectively.

A *pattern* or *tree* in $\mathcal{D}$ is any attribute tree $T \in \mathcal{AT}$ that occurs in $\mathcal{D}$. A *position* in $\mathcal{D}$ is any node $v \in V_{\mathcal{D}}$. If there exists some matching function $\varphi \in \Phi(T, \mathcal{D})$ such that $p = \varphi(r_T)$, then we say that either T *occurs at* position p or p is an *occurrence* of T. For attribute trees, each occurrence $p = \varphi(r_T)$ of tree T determines the matching function φ in a unique way. The *occurrence set* of T in $\mathcal{D}$, denoted by $Occ_{\mathcal{D}}(T)$, is the set of all occurrences of T in $\mathcal{D}$, that is, $Occ_{\mathcal{D}}(T) = \{\, \varphi(r_T) \,|\, \varphi \in \Phi(T, \mathcal{D}) \,\}$. Trees S and T are *equivalent* if $Occ_{\mathcal{D}}(S) = Occ_{\mathcal{D}}(T)$. The equivalent class for T on $\mathcal{D}$ is denoted by $EQ(T) = \{\, T' \in \mathcal{AT} \,|\, Occ_{\mathcal{D}}(T') = Occ_{\mathcal{D}}(T) \,\}$. From now on, we fix a database $\mathcal{D}$, and we may omit the subscript $\mathcal{D}$ if no confusion arises in the future sections.

Let $0 \leq \sigma \leq ||\mathcal{D}||$ be a nonnegative integer, called a *minimum frequency threshold* or a *min-freq*. Then, a tree $T \in \mathcal{AT}$ is *frequent* in $\mathcal{D}$ if $|Occ_{\mathcal{D}}(T)| \geq \sigma$ holds. Both of $Occ_{\mathcal{D}}(T)$ and $|Occ_{\mathcal{D}}(T)|$ are computable in $O(|T| \cdot ||\mathcal{D}||)$ time.

Definition 3 (Closed trees). *A frequent tree T is* closed *in $\mathcal{D}$ if there exists no equivalent tree to T within $\mathcal{AT}$ that properly includes T, that is, there exists no such $T' \in \mathcal{AT}$ that (i) $T \sqsubset T'$ and (ii) $Occ_{\mathcal{D}}(T') = Occ_{\mathcal{D}}(T)$.*

Example 2. In the database of Fig. 1, patterns T_1 and T_2 have the occurrence sets $Occ_{\mathcal{D}}(T_1) = \{2, 36\}$ and $Occ_{\mathcal{D}}(T_2) = \{2, 23, 36\}$, respectively. We also see

[2] Note that the whole database tree $\mathcal{D}$ with the master root v_0 is not an attribute tree and represents a forest of data trees, since v_0 may have children with possibly same labels. However, it is justified whenever no pattern is allowed to occur at v_0.

that $T_2 \sqsubseteq T_1$ holds. Let $\sigma = 2$ be a minimum frequency threshold. T_1, T_2, and T_3 are frequent patterns in $\mathcal{D}$ since. T_1 is closed in $\mathcal{D}$.

In other words, a closed tree T is a maximal element of $EQ(T)$. For threshold σ, $\mathcal{F}_\sigma$ and $\mathcal{C}_\sigma$ denotes the classes of all frequent trees and all frequent closed trees, respectively, in $\mathcal{D}$. We write $\mathcal{F}$ and $\mathcal{C}$ for the threshold $\sigma = 1$. Now, we state our data mining problem as follows.

CLOSED PATTERN MINING PROBLEM FOR ATTRIBUTE TREES
Given a database $\mathcal{D} = \{D_1, \ldots, D_m\}$ $(m \geq 0)$ of attribute trees and a minimum frequency threshold $0 \leq \sigma \leq ||\mathcal{D}||$, find all frequent closed trees $T \in \mathcal{AT}$ in $\mathcal{D}$ without duplicates.

Our goal in this paper is to design an output-polynomial time algorithm for the frequent closed pattern problem for the class $\mathcal{AT}$ using as small memory footprint as possible. An algorithm $\mathcal{M}$ solves an enumeration problem Π in *output-polynomial time* [5] if the running time of $\mathcal{M}$ is bounded by a polynomial time in m and n, where $m = |\mathcal{C}_\sigma|$ and $n = ||\mathcal{D}||$.

Since a transaction database with attributes $\mathcal{A}$ can be encoded by a forest of depth three over alphabet $\mathcal{A} \cup \{\ell_{\mathrm{db}}, \ell_{\mathrm{record}}\}$, the following lemmas for attribute trees follows from the corresponding lemmas for transaction databases.

Lemma 3. *There exist some database $\mathcal{D} \subseteq \mathcal{AT}$ and $\sigma \geq 0$ such that $|\mathcal{C}|$ is exponentially larger than the input size $||\mathcal{D}||$.*

Lemma 4 (Uno *et al.* [24]). *There exists some database $\mathcal{D} \subseteq \mathcal{AT}$ and $\sigma \geq 0$ such that $|\mathcal{F}_\sigma|$ is exponentially larger than $|\mathcal{C}_\sigma|$.*

From Lemma 3 and Lemma 4, we see that a naïve generate-and-test algorithm with enumeration of all frequent patterns cannot be output-sensitive.

2.4 Relationship to Other Models of Semi-structured Data

The class $\mathcal{AT}$ of attribute trees can be related in several ways to the existing models of structured and semi-structured data as follows.

- $\mathcal{AT}$ is a slight modification of *ranked trees* in the studies of tree automata and formal logic. In ranked trees, the domain of indices is restricted to non-negative integers rather than arbitrary countable set $\mathcal{A}$. Also the number of children of each node, called *rank* is determined by the symbol attached to the node.
- $\mathcal{AT}$ is a special case of *labeled ordered trees* [3,18,27] and *labeled unordered trees* [4,15], which are extensively studied in semi-structured data mining. Trees in $\mathcal{AT}$ have the constraint that the labels of the children of each node are mutually distinct.
- $\mathcal{AT}$ is corresponds to the class of complex objects with the tuple constructor only [6,10] where a *complex object* over an attribute alphabet A is either an empty object $O = \emptyset$ or a hierarchical tuple $O = \{a_1 : O_1, \ldots, a_n : O_n\}$ for attributes $a_1, \ldots, a_n \in A$ and complex objects $O_1, \ldots, O_n$.

- $\mathcal{AT}$ can be considered as a fragment of *description logic* [9] where only functional roles/attributes are allowed and equivalence constraints and complex logical constructs are not allowed. The relation $\sqsubseteq$ corresponds to the subsumption relation of such logic. Furthermore, roughly speaking, if we regard a tree database $\mathcal{D}$ as a model $I_{\mathcal{D}}$ and a tree pattern T as a formula ϕ_T in this version of description logic, then the occurrence set $Occ(T)$ of T corresponds to the extension of ϕ_T in $I_{\mathcal{D}}$, where a matching function is not necessarily one-to-one.
- $\mathcal{AT}$ corresponds to a simple subclass of *conjunctive queries* in deductive databases and first-order logic programs. The database has monadic predicates $Q_1(\cdot), \ldots, Q_m(\cdot)$ for labels and a binary predicate $R(\cdot, \cdot)$ for edges. A database has tree structure in the edge predicate, and a pattern is a definite clause of the form

  $$P(X) \leftarrow Q_1(X_1), \ldots, Q_m(X_m), E_1(Y_1, Z1), \ldots, E_n(Y_n, Z_n)$$

  with an underlying variable dependency structure of tree-shape, with some constraint on the appearance of monadic predicate corresponding to the definition of attribute trees.

A natural question is how useful the class of attribute trees is. Clearly, not all XML databases are attribute trees. For a non-attribute labeled tree T, there are two possible ways to derive an attribute tree version of T as follows. The first way is to simply remove all but first nodes with the same label in siblings. The second way is recursively merge the siblings with the same labels starting from the root node of T. In Section 5, we give an example of such an attribute tree derived from a real world dataset.

3 Characterization of Closed Attribute Trees

In this section, we give a characterization for closed attribute trees, which plays a central role in our output-polynomial time algorithm for frequent closed tree mining. This characterization used the notion of least general generalization for trees, and is a natural generalization of properties of closed itemsets to attribute trees.

In this and the next sections, we identify a tree in $\mathcal{AT}$ and its address set representation if no confusion arises.

3.1 A Representation for Attribute Trees

In this subsection, we introduce the address set representation of attribute trees, which is a combination of sequence representation for frequent itemsets [7] with tree domains for ranked trees.

For an attribute tree $T \in \mathcal{AT}$, each node v of T has the unique path π from the root to v. Then, the *address* of v, denoted by $dom(v)$, is the sequence $\alpha = (a_1, \ldots, a_m) \in \mathcal{A}^*$ of node labels spelled out by the path π. We also call any element of $\mathcal{A}^*$ an *address* on $\mathcal{A}$. The *address set* (or *domain*) of a tree $T \in \mathcal{AT}$ is defined by the set $dom(T) = \{\, dom(v) \in \mathcal{A}^* \mid v \in V_T \,\} \cup \{\varepsilon\}$, the set of all

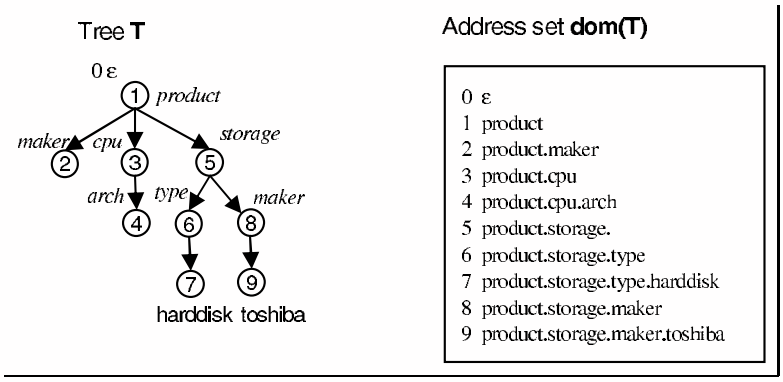

Fig. 2. A attribute tree T and its address set $dom(T)$

addresses for the nodes of T. For the empty tree $\bot$, we define $dom(\bot) = \{\varepsilon\}$.[3] Intuitively, an address and an address set over $\mathcal{A}^*$ correspond to a node and a tree in $\mathcal{A}$, respectively.

For an address $\alpha = (a_1, \ldots, a_{d-1}, a_d)$ of length $d \geq 1$, the *parent address* of α is the address $pa(\alpha) = a_1, \ldots, a_{d-1}$ of length $d-1$. A set $A \subseteq \mathcal{A}^*$ is *prefix-closed* if $\alpha \in A$ implies $pa(\alpha) \in A$ for any address $\alpha \in \mathcal{A}^*$. The following lemmas are well known saying that the address set precisely encodes an attribute tree.

Lemma 5. *Let $A \subseteq \mathcal{A}^*$ be any set of addresses. Then, $dom(T) = A$ for some tree $T \in \mathcal{AT}$ iff A is prefix-closed.*

Lemma 6. *Trees T_1 and $T_2 \in \mathcal{AT}$ are isomorphic iff $dom(T_1) = dom(T_2)$ holds.*

The conversion between T and $dom(T)$ can be done in each direction in linear time of the input size.

Consider the set $\mathcal{A}^*$ of all addresses and the lexicographic order $<_{\text{lex}}$ over $\mathcal{A}^*$. It is often convenient to regard an address set $A = \{\alpha_1, \ldots, \alpha_n\}$ as an ordered sequence $(\alpha_{i_1}, \ldots, \alpha_{i_n})$, where $\alpha_{i_1} <_{\text{lex}} \cdots <_{\text{lex}} \alpha_{i_n}$ for some permulation $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$. With this sequence notation, we have the following definition. Let $\gamma \in \mathcal{A}^*$ be any address. The *γ-prefix* and the *strict γ-prefix* of A are the elements of A that are less than or equal to γ and strictly less than γ, that is, $A(\gamma) = \{\, \alpha \in A \,|\, \alpha \leq_{\text{lex}} \gamma \,\}$, and $A(\gamma - 1) = \{\, \alpha \in A \,|\, \alpha <_{\text{lex}} \gamma \,\}$, respectively.[4] The *head* and the *tail* of A is the the minimum and the maximal elements $hd(A) = \min(A)$ and $tl(A) = \max(A)$, respectively (They are equivalent to α_{i_1} and α_{i_n} in the sequence notation above).

For a tree A, an address $\alpha \in \mathcal{A}^*$ is *open* for an address set A if $pa(\alpha) \in A$ and $\alpha \notin A$ hold. We denote by $Open(A)$ the set of all open addresses for A.

[3] Here, we assume that every tree T contains an invisible *grand root* with the address ε. This treatment is just necessary to ensure a tree domain to be the prefix-closed.

[4] If $\mathcal{A}$ is finite then actually the address $\gamma - 1$ exists as the predecessor of address γ. It is not the case when $\mathcal{A}$ is countably infinite and γ ends with the smallest letter in $\mathcal{A}$. However, we can still use this notation safely if A is finite as in our case.

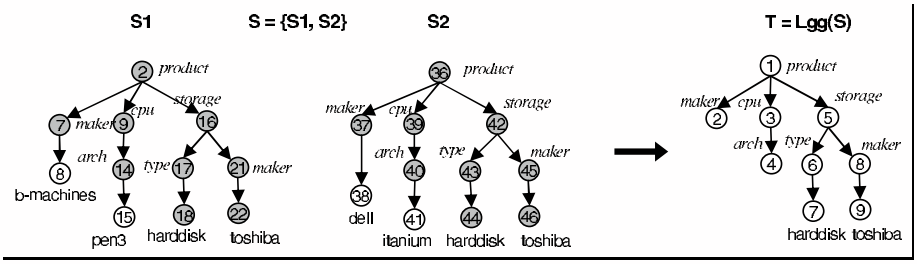

Fig. 3. The least general generalization $T = Lgg(S)$ of set of trees $\{S_1, S_2\}$. The tree T is the unique maximal tree that is more general than both of S_1 and S_2.

3.2 The Least General Generalization and Closure Operation

In this subsection, we introduce the least general generalization for attribute trees by extending the original definition for atomic formulas by Plotkin [20] and Reynolds [21]. Then, we give the closure operation for trees in $\mathcal{AT}$.

We define a binary relation $\preceq$ over $\mathcal{AT}$, called the *generalization relation*, as follows. For any trees $S, T \in \mathcal{AT}$, if there exists some $\varphi \in \Phi(S,T)$ such that $\varphi(r_S) = r_T$ then we define $S \preceq T$ and say that S is *more general than* T or T is *more specific than* S. If $S \preceq T$ but $T \not\preceq S$ then S is *properly more general than* T or T is *properly more specific than* S. Clearly, $S \preceq T$ implies $S \sqsubseteq T$, and thus, $\preceq$ is a partial order. However, the converse does not hold in general since φ have to map the root of S into the root of T in the case for $\preceq$.

Lemma 7. *For any $S, T \in \mathcal{AT}$, $S \preceq T$ iff $dom(S) \subseteq dom(T)$.*

The generalization relation satisfies the following anti-monotonicity.

Lemma 8. *Let $S, T \in \mathcal{AT}$ be any trees.*

1. *If $S \preceq T$ then $Occ(S) \supseteq Occ(T)$.*
2. *If $S \preceq T$ then $|Occ(S)| \geq |Occ(T)|$.*

Then, the least general generalization of a set of trees is defined as follows. Let $\mathcal{S} \subseteq \mathcal{AT}$ be a finite set of trees. A tree $T \in \mathcal{AT}$ is a *common generalization* for $\mathcal{S}$ if $T \preceq S$ for every $S \in \mathcal{S}$. A common generalization T of $\mathcal{S}$ is the *least general generalization* (*lgg*) of $\mathcal{S}$ if T is more specific than any common generalization for $\mathcal{S}$, i.e., $T' \preceq T$ for any common generalization T' for $\mathcal{S}$. We denote the lgg of $\mathcal{S}$ by $Lgg(\mathcal{S})$. The following theorem says that $Lgg(\mathcal{S})$ always exists and unique.

Lemma 9. *For any set $\mathcal{S} \subseteq \mathcal{AT}$, $Lgg(\mathcal{S})$ is the unique tree $T_\cap$ such that $dom(T_\cap) = \bigcap_{S \in \mathcal{S}} dom(S)$.*

Proof: Let $\mathcal{S} = \{S_1, \ldots, S_n\} \subseteq \mathcal{AT}$ be a finite set of trees, where $m \geq 0$. Then, we can show that $Lgg(\mathcal{S})$ is the unique tree $T_\cap \in \mathcal{AT}$ whose address set is given by the intersection of all address sets $A_1, \ldots, A_n$, that is, $A_\cap = \bigcap_i A_i$, where $A_i = ad(S_i)$ for every $i = 1, \ldots, m$. Now, we give the proof for the above

claim. Let $T_\cap \in \mathcal{AT}$ be the tree defined in the above statement. If both sets A_1 and A_2 are prefix-closed then so is $A_1 \cap A_2$. Thus, the intersection $A_\cap$ is also prefix-closed. From Lemma 6 and Lemma 5, such a tree $T_\cap$ always exists and is unique. On the other hand, assume that we have a common p-generalization T' of $\mathcal{S}$. If $T' \preceq S_i$ then $A' = ad(T')$ is included in $A_i = ad(S_i)$ for every i. Thus, $A' \subseteq \bigcap_i ad(S_i) = A_\cap$ holds. From Lemma 7, this implies that $T' \preceq T$ for any common generalization T'. Hence, we know that T is the unique least general generalization of $\mathcal{S}$ w.r.t. $\preceq$. □

Theorem 1. *The least general generalization $Lgg(\mathcal{S})$ for a finite set $\mathcal{S} \subseteq \mathcal{AT}$ of attribute trees is unique, of polynomial size, and polynomial time computable in the total size of $\mathcal{S}$.*

From the proof of the above theorem, we present an $O(mn)$ time algorithm for $Lgg(\mathcal{S})$ as in [20]. Cohen *et al.* [12] studied the least general generalization for a more general fragment of description logic, called CLASSIC.

Now, we give the closure operation for trees.

For a position $v \in V_\mathcal{D}$, the *subtree (or half-tree) rooted at position v* is the tree S whose domain is given by $dom(S) = \{ \beta \in \mathcal{A}^* \mid \alpha\beta \in dom(\mathcal{D}) \}$ for the address α of v. For a set $P \subseteq V_\mathcal{D}$ of positions, the *tree set* of tree T for P, denoted by $Tree(T)$, is the set of all subtrees of $\mathcal{D}$ rooted at some positions in P.

Definition 4 (The closure operation). *Let $\mathcal{D}$ be a database. The* closure *of a tree $T \in \mathcal{AT}$ is the tree $Clo_\mathcal{D}(T) = Lgg(Tree_\mathcal{D}(Occ_\mathcal{D}(T)))$.*

Lemma 10. *$Clo(T)$ is computable in $O(mn)$ time in the size $m = |T|$ of T and the total size $n = ||\mathcal{D}||$ of $\mathcal{D}$.*

Theorem 2. *$Clo(T)$ is the unique maximal tree in the equivalence class $EQ(T)$*

Proof: Let $\mathcal{S}$ be the set of all half-trees in $\mathcal{D}$ rooted at the occurrences of T in the database. Then, the closure of T is $L = Lgg(\mathcal{S})$. Now, we show that if P is any member of $EQ(T)$ then the pattern P is also more general than the closure $Lgg(\mathcal{S})$. Let P be any member of $EQ(T)$. Then, P occurs at all occurrences of T in the database, i.e., $Occ(T) \subseteq Occ(P)$. By the definition of half-trees, this implies that P is more general than all half-trees in $\mathcal{S}$. From the definition, $Lgg(\mathcal{S})$ is the unique greatest tree that is more general than all half-trees in $\mathcal{S}$. Thus, it immediately follows that P is more general than $Lgg(\mathcal{S})$. Since this is valid for all $P \in EQ(T)$, we see that $Lgg(\mathcal{S})$ is the greatest member of $EQ(T)$ in terms of $\preceq$. □

Theorem 3. *A tree $T \in \mathcal{AT}$ is a closed tree in $\mathcal{D}$ iff $Clo(T) = T$.*

From Theorem 3 and Lemma 10, we can test if T is closed or not in polynomial time in $|T|$ and $||\mathcal{D}||$. We listed below some properties of closed trees, which are useful in show in the above theorems and also for the discussion in the later sections.

Lemma 11. *For any trees $T, T_1, T_2 \in \mathcal{AT}$, the following properties hold:*

1. $T \preceq Clo(T)$.
2. *If* $T_1 \preceq T_2$ *then* $Clo(T_1) \preceq Clo(T_2)$.
3. *If* $Occ(T_1) \subseteq Occ(T_2)$ *then* $Clo(T_1) \preceq Clo(T_2)$.
4. $Clo(Clo(T)) = Clo(T)$.
5. $Clo(T)$ *is the unique smallest closed tree including* T.
6. *For closed trees* $T_1, T_2 \in \mathcal{C}$, $T_1 \preceq T_2$ *iff* $Occ(T_1) \preceq Occ(T_2)$.

4 Output-Polynomial Time Algorithm for Closed Trees

In this section, we present an efficient algorithm for enumerating all frequent closed trees in polynomial time per tree without duplicates in the total size of the input database.

4.1 Possible Approaches

In this subsection, we consider and briefly summarize the possible approaches for computing frequent closed trees and point out some problems in them.

The first approach is to use a frequent tree mining algorithm. In mining of labeled ordered trees, an efficient enumeration technique, called *rightmost expansion* in [3], is used for generating all frequent labeled ordered trees with depth-first search. In our representation for attribute trees with address set, the definition is given as follows.

Definition 5 (Rightmost expansion). *Let $k \geq 1$ and $S \in \mathcal{AT}$ be a tree of size $k-1$. Then, a tree T of size k is said to be a* rightmost expansion *of S if $T = S \cup \{\beta\}$ for some open address $\beta \in Open(S)$ such that $\beta >_{\mathrm{lex}} td(S)$.*

Using rightmost expansion, we can implement an algorithm that enumerates all frequent trees in $\mathcal{AT}$ without duplicate, which starts from the empty tree, and searches all frequent trees from smaller to larger by the rightmost expansion as Asai *et al.* showed for the computation of frequent ordered trees [3].

We can modify this algorithm to compute all frequent closed trees by first enumerating each tree, and then testing if it is closed. This algorithm requires at least time proportional to $||\mathcal{F}_\sigma|| > ||\mathcal{C}_\sigma||$. Thus, it cannot be an output-polynomial time algorithm at all.

The second approach is to use the closure operation to generate closed trees. T is an *expansion* for S if $T = S \cup \{\beta\}$ for some open address $\beta \in Open(S)$.

Definition 6 (Closure expansion). *Let $k \geq 1$ and $S \in \mathcal{AT}$ be a tree of size $k-1$. Then, a tree T of size k is said to be a* closure expansion *of S if T is the closure of an expansion for S, that is, $T = Clo_{\mathcal{D}}(S \cup \{\beta\})$ for some $\beta \in Open(S)$.*

It is not hard to see that any closed tree T is a closure expansion of some closed tree S. Then, we can implement an algorithm for computing all frequent closed trees working with level by level using breadth-first search or level-wise search as Uno *et al.* showed for the computation of frequent closed itemsets [24].

This algorithm starts from the set of frequent closed trees of size one, and for every level $k = 1, 2, \ldots$ then iteratively computes from the set $\mathcal{C}_k$ of trees of size k the set $\mathcal{C}_{k+1}$ by using closure expansion. Since the same tree can be generated more than one parent tree by closure expansion, we have to check if each generated closed tree is not repeated, using the current set of closed trees in a breadth-first manner.

We can prove that the computation time of this approach can be output-polynomial in $||\mathcal{D}||$, using the results to be shown in the following sections.

Corollary 4. *There exists an output-polynomial time algorithm in $||\mathcal{D}||$ for the frequent closed pattern problem using the space proportional to the output size $||\mathcal{C}_\sigma||$.*

However, this algorithm with closure expansion alone requires at least the memory space proportional to the total size $||\mathcal{C}_\sigma||$ of outputs due to its breadth-first search scheme.

Overall, neither of the approaches with rightmost expansion alone and with closure expansion alone are not satisfactory yet. To overcome these problems, we combine both approaches in the following sections to achieve efficient enumeration with small amount of space proportional to $||\mathcal{D}||$ rather than $||\mathcal{F}_\sigma||$ and $||\mathcal{D}||$.

4.2 Tree-Shaped Search Space for Closed Trees

In this subsection, we introduce a tree-shaped search structure over $\mathcal{C}$, which is based on a search technique, called reverse search [5].

We first give a parent function over closed trees. Let $\mathcal{D}$ be a database. Then, the *root closed pattern* is the smallest tree $root_{\mathcal{C}} = Clo(\bot)$ equivalent to the empty pattern $\bot$ and always exists. Let $A \subseteq \mathcal{A}^*$ be an address set of a tree. Recall that we introduced the notations $A(\gamma)$ and $A(\gamma-1)$ for an address $\gamma \in \mathcal{A}^*$ in Section 3.1, where $A(\gamma)$ is the set of addresses in A less than or equal to γ and $A(\gamma-1)$ is the set of addresses strictly less than γ. We define the *core index* of A by

$$core\_i(A) = \min\{\, \gamma \in A \,|\, Occ(A) = Occ(A(\gamma)) \,\},$$

that is, the minimum address $\gamma \in A$ such that $Occ(A) = Occ(A(\gamma))$. For $root_{\mathcal{C}}$, we define $core\_i(root_{\mathcal{C}}) = -1$.

Definition 7 (The parent tree). *Let $T \in \mathcal{C}\backslash\{root_{\mathcal{C}}\}$ be any non-root closed tree. Then, the* parent *of T, denoted by $\mathcal{P}(T) \in \mathcal{AT}$, is defined by*

$$\mathcal{P}(T) = Clo(T(core\_i(T) - 1)).$$

Lemma 12. *For any non-root closed tree $T \in \mathcal{C}\backslash\{root_{\mathcal{C}}\}$, the parent tree $\mathcal{P}(T)$ always exists, is unique, and is also a member of $\mathcal{C}$.*

Proof: Since $T \in \mathcal{C}\backslash\{root_{\mathcal{C}}\}$, T is not equivalent to the empty tree $\bot$ in its occurrence set. Thus, its core index $\gamma = core\_i(T)$ must be greater than zero, and thus the prefix $T(\gamma-1)$ is defined. For any tree T, its closure $Clo(T)$ always exists. Hence, the result follows. □

Lemma 13. *For any non-root closed tree $T \in \mathcal{C}\backslash\{root_{\mathcal{C}}\}$, the following properties hold:*

1. $|\mathcal{P}(T)| < |T|$ *holds.*
2. $\mathcal{P}(T) \prec T$ *holds.*

Proof: The proof follows from Lemma 11. □

Let us consider a directed graph $\mathcal{T} = (\mathcal{C}, \mathcal{P}, root_{\mathcal{C}})$, called a *search graph* for $\mathcal{C}$, where each node is a closed tree T and there exists an edge (a reverse edge) from a tree T to tree S if $\mathcal{P}(T) = S$. From Lemma 12 and Lemma 13, we have the following lemma.

Lemma 14 (Existence of tree-shaped search space for $\mathcal{C}$). *The search graph $\mathcal{T} = (\mathcal{C}, \mathcal{P}, root_{\mathcal{C}})$ for $\mathcal{C}$ is a spanning tree over all closed trees in $\mathcal{C}$ with the unique root $root_{\mathcal{C}}$.*

Lemma 14 is also valid for frequent closed trees in $\mathcal{C}_\sigma$ with min-freq threshold σ since the parent edge satisfies the anti-monotonicity in frequency (Lemma 8).

4.3 Prefix-Preserving Closure Expansion

In this subsection, we give the prefix-preserving closure expansion. Let $S, T \in \mathcal{AT}$ be trees. $T \in \mathcal{AT}$ is said to be a *prefix-preserving closure expansion* (ppc-expansion) of S if

(i) $T = Clo(S \cup \{\beta\})$ for some $\beta \in Open(S)$, that is, T is obtained by first adding a new node to S, and then taking its closure.
(ii) the address β satisfies $\beta > core\_i(S)$.
(iii) $S(\beta - 1) = T(\beta - 1)$, that is, the strict β-prefix of S is preserved.

In the search, starting from the root tree $root_{\mathcal{C}}$, we search the search tree $\mathcal{T}$ by growing the present tree by taking its ppc-expansions. The next lemma says that the core index can be recursively computed.

Lemma 15. *Let S be a closed tree and $T = Clo(S \cup \{\beta\})$ be a ppc-expansion of S. Then, β is the core index of T.*

Proof: Since $Clo(T(\beta)) = T$ by assumption, we at least know that $core\_i(T) \leq \beta$. Assume to contradict that $core\_i(T) < \beta$ and that $Clo(T(\delta)) = T$ holds for some $\delta < \beta$ such that $\delta \notin S$. Then, we can show that $Clo(S \cup \{\delta\}) = T$ holds. However, this implies the contradiction that $S(\beta) \neq T(\beta)$ since $\delta \notin S$ but $\delta \in T$. Thus, we conclude that $core\_i(T) = \beta$. □

Lemma 16. *Let S be a closed tree. Then, all ppc-expansions of S can be generated in polynomial time per ppc-expansions in $|S|$ and $||\mathcal{D}||$.*

We show that any non-root tree T can be generated from its parent $\mathcal{P}(T)$ by ppc-expansion. In the following proofs, we assume that $\mathcal{A}$ is finite for simplicity. However, these lemmas also hold for infinite $\mathcal{A}$.

Lemma 17. *Let T be a non-root closed tree, and $S = \mathcal{P}(T)$ be the parent tree of T. Then, T is a ppc-expansion of S.*

Proof: Let $\gamma = core\_i(T)$. We will show that $T = Clo(S \cup \{\gamma\})$. By assumption, $S = Clo(T(\gamma - 1))$. Thus, the core index of S is at least strictly smaller than γ and this satisfies condition (*ii*) of ppc-expansion. Since $T(\gamma) = T(\gamma - 1) \cup \{\gamma\}$. By Lemma 11, we have $T(\gamma) \preceq S \cup \{\gamma\} \preceq T$. Since $Clo(T(\gamma)) = T$ for the core index γ, it follows from Lemma 11 that condition (i) $Clo(S \cup \{\gamma\}) = T$ of ppc-expansion. Since $S = Clo(T(\gamma - 1))$, $S(\gamma - 1)$ already includes $T(\gamma - 1)$. The converse is also true T is a closure of $S \cup \{\gamma\}$. Thus, we have condition (iii) $S(\gamma - 1) = T(\gamma - 1)$ of ppc-expansion. □

Lemma 18. *Let S be a closed tree, and T be a ppc-expansion of S. Then, S is the parent tree of T, i.e., $S = \mathcal{P}(T)$.*

Proof: By assumption, (i) $T = Clo(S \cup \{\beta\})$ for some $\beta > core\_i(S)$. Then, by condition (iii) of ppc-expansion $S(\beta - 1) = T(\beta - 1)$, we know that $\gamma = core\_i(T)$ is at least larger than β, and thus strictly larger than $core\_i(S)$. This implies that $\gamma - 1$ is larger than or equal to $core\_i(S)$. Since $\gamma - 1 \geq \beta - 1$, we have $S(\gamma - 1) \preceq T(\gamma - 1)$. On the other hand, since $\gamma - 1 \geq core\_i(S)$ as above, we have $Clo(S(\gamma - 1)) = S$. This implies that $Clo(T(\gamma - 1))$ is at least as general as S, and thus equivalent to S. This shows that $\mathcal{P}(T(core\_i(T) - 1)) = S$. □

Combining Lemma 17 and Lemma 18, we show that the ppc-expansion correctly generates the children of a closed pattern in the search graph.

Theorem 5. *Let S and T be a closed tree such that $T \neq root_{\mathcal{C}}$. Then, S is the parent tree of T iff T is a ppc-expansion of S.*

4.4 Algorithm

In Fig. 4, we show our algorithm CLOATT (Closed Attribute Tree Miner) for discovering all frequent closed trees in a given database. This algorithm uses ppc-expansion introduced in the previous section. Starting from the smallest closed tree $root_{\mathcal{C}}$, the algorithm performs depth-first search for closed trees by finding the children of the present closed tree using ppc-expansion.

Theorem 6. *Let $\mathcal{D} \subseteq \mathcal{AT}$ be a database and $\sigma \geq 1$ be a minimum frequency threshold. Then, the algorithm* CLOATT *of Fig. 4 finds all frequent closed trees $T \in \mathcal{AT}$ appearing in $\mathcal{D}$ in $O(bm^2n)$ amortized time per tree without duplicates using $O(n)$ memory space, where b is the maximal branching of each data trees in $\mathcal{D}$, $m = |T|$ is the size of the tree found, and $n = ||\mathcal{D}||$ is the total size of the database $||\mathcal{D}||$.*

Proof: From Lemma 14 and Theorem 5, the algorithm CLOATT correctly searches all closed trees on the spanning tree $\mathcal{T}$ for $\mathcal{F}$. Since this search space forms tree, each closed tree is generated exactly once. For each closed tree T, we can compute $Occ(T)$ in $O(mn)$ time. From Lemma 16, we can also compute all ppc-expansions of T in polynomial time, more exactly in $O(bm^2n)$ time

```
1  Algorithm CLOATT
2    input: a database 𝒟 and a min-frequency threshold 1 ≤ σ ≤ ||𝒟||.
3    output: all frequent closed patterns in 𝒟 with min-freq σ;
4    T_0 := Clo(⊥).              // Most general closed pattern root_C
5    γ_0 := core_i(T_0).         // Core index of T_0
6    PPC-Expand(T_0, γ_0, Occ(T_0), 𝒟, σ).

7  Proc. PPC-Expand(S, γ, Occ(S), 𝒟, σ)
8    If |Occ(S)| < σ then return                  // Not frequent
9    Else if Clo(S) ≠ S then return               // Not closed
10   Else                                         // Closed pattern
11     Output S.
12     For each address β ∈ Open(S) such that β >_lex γ do:
         // PPC-Expansion
13       T := Clo(S ∪ {β});
14       If S(γ − 1) ≠ T(γ − 1) then return.
15       PPC-Expand(T, β, Occ(T), 𝒟, σ).

16   Prefix(γ) := { α ∈ 𝒜* | α <_lex γ }.
```

Fig. 4. A frequent closed pattern miner using prefix-preserving closure (PPC) expansion. This algorithm runs in output-polynomial time also with a small amount of memory due to the pure depth-first search.

since there are at most bm ppc-extensions of T. From the recursive computation scheme of the algorithm, if all ppc-expansions of T are not closed then this branch of computation is terminated and the algorithm backtracks. Therefore, the amortized computation time per generated tree is again $O(bm^2n)$ time. Since the recursive call for the subprocedure *PPC-Expand* can be implemented by using a stack of length at most $m = |T|$ where each entry contains a pair of an ancestor tree S of T and its occurrence set $Occ(S)$, the memory space used is $O(\ell) = O(mn)$, where ℓ is the sum of $|Occ(S)| = O(||\mathcal{D}||)$ for all ancestors of the current tree T. Furthermore, this can be reduced to $O(n)$ by recording only the differences of these occurrence lists. This completes the proof. □

Corollary 7. *There exists an output-polynomial time algorithm for the frequent closed pattern problem using the space proportional to the total database size.*

5 Experiments

In this section, we present some experimental results on a qualitative assessment at the utility of closed attribute tree mining. In particular, we examine how much reduction is possible by closed pattern discovery on a real world dataset.

Table 1. The size of original and pruned datasets

Dataset	# documents	# nodes	Is AT?
dblp1830.xml	986	37,276	No
dblp1830at.xml	986	33,468	Yes

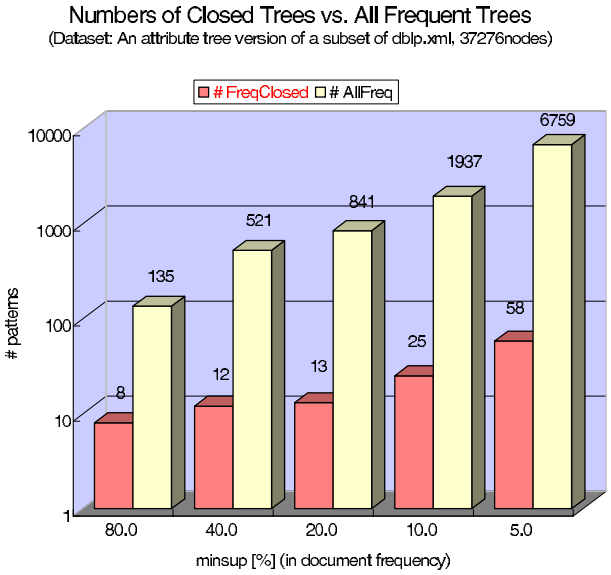

Fig. 5. The number of frequent closed trees and all frequent trees against the minimum frequency thresholds

We first build a pruned dataset consisting of attribute trees derived from a real world dataset as follows. The orignal dataset is a subset of an XML dataset `dblp1830.xml` consisting of 986 XML documents extracted from a bibliographic database DBLP (`dblp.xml`).[5] Since this dataset contains repeated occurrences of the same label, such as `author` and `ee` in siblings, we prune the dataset by removing all but first occurrences of the repeated attributes and its subtrees in siblings. The resulting dataset, called `dblp1830at.xml`, consists only of attribute trees. In Table 1, we show statistics of the original and pruned datasets. From the table, we can see that the dataset retains a large part of the structure in the original datasets.

Then, we compute the sets of all frequent closed trees and frequent trees in the pruned dataset `dblp1830at.xml`. To compute all frequent attribute trees, we used an implementation of a frequent unordered tree miner UNOT [4]. Since

[5] `http://www.informatik.uni-trier.de/~ley/db/`

we have not implemented the algorithm CLOATT in the previous section, we compute all closed patterns by explicitly checking if the condition of Definition 3 holds for each frequent trees computed by UNOT. In Figure 5, we show the number of the frequent closed trees and the number of all frequent trees when we vary the minimum frequency threshold from 80.0% to 5.0% in document frequency. From this figure, we can observe that the number of the frequent closed trees is order of magnitude smaller than the number of all frequent trees for most minimum frequency threshold values. Some of the discovered closed trees corresponded to a schema structure inherent to the DBLP database.

6 Conclusion

In this paper, we presented an output-polynomial time algorithm for mining all frequent closed patterns for the class of attribute trees.

This algorithm computes all frequent closed trees in polynomial time per closed tree without duplicates in the total size of the input database using a small amount of memory with depth-first search. For the purpose, we gave a characterization of closed trees in terms of the least generalization for attribute trees, and an efficient enumeration method, called pcc-expansion, for realizing direct enumeration of closed trees only using the depth-first search.

The class $\mathcal{AT}$ of attribute trees can be related in several ways to the existing models of structured and semi-structured data. In particular, $\mathcal{AT}$ has a close relationship to a fragment of description logic with functional roles only. Thus, it is an interesting future work to generalize the result of this paper to richer fragment of description logic. This may include the introduction of equivalence constraints and non-functional roles.

In this paper, we are working only with theoretical framework for efficient closed tree miners using ppc-extension. The implementation of the proposed algorithm CLOATT and the estimation of its efficiency on realworld datasets will be future works.

Acknowledgment

The authors would like to thank Ken Satoh, Shinichi Nakano, Ryutaro Ichise, Hideaki Takeda, Akihiro Yamamoto, Hiroshi Sakamoto, and Shinichi Shimozono for their valuable discussions and comments. The first author also would like to thank Fabien de Marchi and Salima Benbernou of UCBL1 and Makoto Haraguchi of Hokkaido University for their introduction to and discussions on description logic and to also thank Mohand-Said Hacid of UCBL1 and Yuzuru Tanaka of Hokkaido University for giving the opportunity for the research. The authors also thank anonymou referees for their valuable comments that greatly improve the quality of this paper. This research is partly supported by Grant-in-Aid for Scientific Researchon Priority Areas on "Informatics," Ministry of Education, Culture, Sports, Science and Technology of Japan, and Cooperative Fund by National Institute of Informatics, Japan.

References

1. S. Abiteboul, P. Buneman, D. Suciu, Data on the Web, Morgan Kaufmann, 2000.
2. Aho, A. V., Hopcroft, J. E., Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, 1983.
3. T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *Proc. the 2nd SIAM Int'l Conf. on Data Mining (SDM2002)*, 158–174, 2002.
4. T. Asai, H. Arimura, T. Uno, S. Nakano, Discovering frequent substructures in large unordered trees, In *Proc. the 6th Int'l Conf. on Discovery Science (DS'03)*, LNAI 2843, Springer-Verlag, 47-61, 2003.
5. D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Applied Mathematics, 65(1–3), 21–46, 1996.
6. Bancilhon, Khoshafian, A calculus for complex objects, In *PODS'86*, 53–59, 1986.
7. R. J. Bayardo Jr., *Efficiently Mining Long Patterns from Databases*, In Proc. SIGMOD98, 1998, pp. 85–93.
8. B. Bringmann, Matching in Frequent Tree Discovery, In *'Proc. IEEE ICDM 2004*, 335–338, 2004.
9. A. Borgida, R. J. Brachman, D. L. McGuinness, L. A. Resnick, CLASSIC: A Structural Data Model for Objects, In *Proc. SIGMOD'89*, ACM, 58–67, 1989.
10. P. Buneman, S. B. Davidson, G. G. Hillebrand, D. Suciu, A query language and optimization techniques for unstructured data, In *Proc. SIGMOD'96*, ACM, 505–516, 1996.
11. Y. Chi, Y. Yang, Y. Xia, and R. R. Muntz, Cmtreeminer: Mining both closed and maximal frequent subtrees, In *Proc. PAKDD'04*, 2004.
12. W. W. Cohen, A. Borgida, H. Hirsh, Computing Least Common Subsumers in Description Logics, In *Proc. AAAI'92*, 754-760, 1992.
13. C. M. Cumby and D. Roth, Learning with feature description logic, In *Proc. ILP 2002*, LNAI 2583, 23–47, 2003.
14. A. Inokuchi, T. Washio, H. Motoda, An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, In *Proc. PKDD 2000*, 13–23, LNAI 1910, Springer-Verlag, 2000.
15. S. Nijssen, J. N. Kok, Effcient Discovery of Frequent Unordered Trees In *Proc. the First International Workshop on Mining Graphs, Trees and Sequences (MGTS'03)*, Sep. 2003.
16. P. Kilpelainen, H. Mannila, Ordered and Unordered Tree Inclusion, SIAM J. Computing, 24(2), 340–356, 1995.
17. Kosaraju, S. R., Efficient tree pattern matching, In *Proc. 30th FOCS*, 178–183, 1989.
18. S. Nakano, Efficient generation of plane trees, Information Processing Letters, 84, 167–172, Elsevier,2002.
19. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering Frequent Closed Itemsets for Association Rules, In Proc. ICDT'99, 398–416, 1999.
20. G. D. Plotkin, A note on inductive generalization. *Machine Intelligence*, 5, 153–163, Edinburgh University Press.
21. J. C. Reynolds, Transformational systems and the algebraic structure of atomic formulas, Machine Intelligence 5, 135–151, Edinburgh University Press.
22. A. Termier, M.-C. Rousset, M. Sebag, Treefinder: a first step towards xml data mining, In *Proc. ICMD'02*, 2002.
23. A. Termier, M.-C. Rousset, M. Sebag, DRYADE: a new approach for discovering closed frequent trees in heterogeneous tree databases, In *Proc. ICMD'04*, 2004.

24. T. Uno, T. Asai, Y. Uchida, H. Arimura, An efficient algorithm for enumerating closed patterns in transaction databases, In *Proc. DS'04*, LNAI 3245, Springer-Verlag, 16-30, 2004.
25. K. Wang, H. Liu, Schema Discovery for Semistructured Data, In *Proc. KDD'97*, 271–274, 1997.
26. X. Yan, J. Han, CloseGraph: Mining Closed Frequent Graph Patterns In *Proc. SIGKDD'03*, ACM, 2003.
27. M. J. Zaki. Efficiently mining frequent trees in a forest, In *Proc. SIGKDD'02*, ACM, 2002.