Optimized Substructure Discovery for Semi-structured Data

Kenji Abe[1], Shinji Kawasoe[1], Tatsuya Asai[1], Hiroki Arimura[1,2], and Setsuo Arikawa[1]

[1] Department of Informatics, Kyushu University
6–10–1 Hakozaki Higashi-ku, Fukuoka 812–8581, Japan
{k-abe,s-kawa,t-asai,arim,arikawa}@i.kyushu-u.ac.jp
[2] PRESTO, JST, Japan

Abstract. In this paper, we consider the problem of discovering interesting substructures from a large collection of semi-structured data in the framework of optimized pattern discovery. We model semi-structured data and patterns with labeled ordered trees, and present an efficient algorithm that discovers the best labeled ordered trees that optimize a given statistical measure, such as the information entropy and the classification accuracy, in a collection of semi-structured data. We give theoretical analyses of the computational complexity of the algorithm for patterns with bounded and unbounded size. Experiments show that the algorithm performs well and discovered interesting patterns on real datasets.

1 Introduction

Recent progress of network and storage technologies have increased the species and the amount of electronic data, called *semi-structured data* [2], such as Web pages and XML data [26]. Since such semi-structured data are heterogeneous and huge collections of weakly structured data that have no rigid structures, it is difficult to directly apply traditional data mining techniques to these semi-structured data. Thus, there are increasing demands for efficient methods for extracting information from semi-structured data [10,18,19,27].

In this paper, we consider a data mining problem of discovering characteristic substructure from a large collection of semi-structured data. We model semi-structured data and patterns with *labeled ordered trees*, where each node has a constant label and has arbitrary many children ordered from left to right. For example, Fig. 1 shows a semi-structured data encoded as XML data, where the data is nested by pairs <*tag*> and </*tag*> of balanced parentheses.

Our framework of data mining is *optimized pattern discovery* [20], which has its origin in the statistical decision theory in 1970's [11] and extensively studied in the fields of machine learning, computational learning theory, and data mining for the last decade [5,13,14,16,17,20,23]. In optimized pattern discovery, the input data is a collection of semi-structured data with binary labels indicating if a user is interested in the data. Then, the goal of a mining algorithm is to discover

T. Elomaa et al. (Eds.): PKDD, LNAI 2431, pp. 1–14, 2002.

```
<people>
   <person age="40">
      <name> Alan </name> <tel> 7786 </tel> <tel> 2133 </tel> </person>
   <person height="155">
      <name> <first> Sara </first> <last> Green </last> </name> </person>
   <person age="33" height="187"> <name> Fred </name> </person>
</people>
```

Fig. 1. An Example of semi-structured data

such patterns that optimize a given statistical measure, such as the classification error [11] and the information entropy [22] over all possible patterns in the input collection. In other words, the goal is not to find *frequent* patterns but to find *optimal* patterns.

Intuitively speaking, the purpose of optimized pattern discovery is to find the patterns that characterize a given subset of data and separate them from the rest of the database [6]. For instance, suppose that we are given a collection of movie information entries from an online movie database[1]. To find a characteristic patterns to its subcollection consisting only of *action movies*, a simplest approach is to find those patterns frequently appearing in action movies. However, if a characteristic pattern has small frequency, then its occurrences may be hidden by many trivial but frequent patterns.

Another approach is to find those patterns that appear more frequently in action movies but less in the other movies. By this, we can expect to find slight but interesting patterns that characterize the specified sub-collection. The precise description of optimized pattern discovery will be given in Section 2.1.

1.1 Main Results

We present an efficient algorithm OPTT for discovering optimized labeled ordered trees from a large collection of labeled ordered trees based on an efficient frequent tree miner FREQT devised in our previous paper [7]. Unlike previous tree miners equipped with a straightforward generate-and-test strategy [19] or Apriori-like subset-lattice search [15,27], FREQT is an efficient incremental tree miner that simultaneously constructs the set of frequent patterns and their occurrences level by level.

In particular, since we cannot use the standard frequency thresholding as in Apriori-like algorithms [3] in optimized pattern discovery, the potential search space will be quite large. To overcome this difficulty, we employs the following techniques to implement an efficient tree minor:

– Based on the rightmost expansion technique of [7,28], which is a generalization of the item set-enumeration tree technique of Bayardo [8], we can efficiently generate all labeled ordered trees without duplicates.

[1] E.g., *Internet Movie database*, `http://www.imdb.com/`

– Using the rightmost leaf occurrence representation [7], we can store and update the occurrences of patterns compactly.
– Using the convexity of the impurity function ψ, we can efficiently prune unpromising branch in a search process by the method of [21].

Then, we present theoretical results on the performance and the limitation of our tree miner OPTT. For patterns of bounded size k, we show a non-trivial $O(k^{k+1}b^k N)$ time upperbound of the running time of the algorithm OPTT, where N and b is the total size the maximum branching of an input database $\mathcal{D}$. This says that if k and b are small constants as in many applications, then the algorithm runs linear time in N, while a generate-and-test algorithm may have super-linear time complexity when the number of unique labels grows.

In contrast, for patterns of unbounded size, we also show that the optimal pattern discovery problem for labeled ordered trees is hard to approximate. Precisely, the maximum agreement problem, which is a dual problem of the classification error minimization, is not polynomial time approximable with approximation ratio strictly less than $770/767$ if $P \neq NP$.

Finally, we run some experiments on real datasets and show that the algorithm is scalable and useful in Web and XML mining. In particular, we observe that the pruning with convexity is effective in a tree miner and that the depth-first search strategy is an attractive choice from the view of space complexity.

1.2 Organization

The rest of this paper is organized as follows. In Section 2, we prepare basic notions and definitions. In Section 3, we present our algorithm OPTT for solving the optimized pattern discovery problem for labeled ordered trees. In Section 4, we give theoretical analysis on the computational complexity of the algorithm and the problem. In Section 5, we run experiments on real datasets to evaluate the proposed mining algorithm. In Section 6, we conclude.

1.3 Related Works

There are many studies on semi-structured databases [2,26]. In contrast, there have not been many studies on *semi-structured data mining* [7,10,15,18,19,27,28]. Among them, most of the previous studies [7,10,19,27,28] consider frequent pattern discovery but not optimized pattern discovery. We also note that most of these works other than [7,28] are based on a straightforward generate-and-test search or Apriori-like levelwise search and does not use the notion of the rightmost expansion.

On the other hand, the algorithm by Matsuda and Motoda *et al.* [18] finds near optimal tree-like patterns using a greedy search method called the graph-based induction. Inokuchi *et al.* [15] presented an Apriori-style algorithm for finding frequent subgraphs and generalized it for optimized pattern discovery.

Most related work would be a tree miner for labeled ordered tree with gaps by Zaki [28], recently proposed independently to our previous work [7]. The

algorithm uses the essentially same enumeration technique to ours, and equipped with a number of interesting ideas that speed-up the search.

2 Preliminaries

2.1 Optimized Pattern Discovery

We give a problem description of optimized pattern discovery according to [11,20]. A *sample* is a pair $(\mathcal{D}, \xi)$ of a collection $\mathcal{D} = \{D_1, \ldots, D_m\}$, called the *database*, of *document trees* and an *objective attribute* $\xi : \mathcal{D} \to \{0, 1\}$ that indicates if a user is interested in a document tree. A tree $D \in \mathcal{D}$ is *positive* if $\xi(D) = 1$ and *negative* otherwise. We are also given a class $\mathcal{P}$ of patterns, i.e., the class of labeled ordered trees. For a fixed database $\mathcal{D}$, each pattern $T \in \mathcal{P}$ can be identified as a binary attribute $T : \mathcal{D} \to \{0, 1\}$ through tree matching, and splits the database D into disjoint sets D_1 and D_0 of *matched* and *unmatched* documents, where $\mathcal{D}_\alpha = \{ D \in \mathcal{D} \mid T(D) = \alpha \}$ for every $\alpha = 0, 1$.

A natural question here is what patterns are better to characterize the subset D_1 relative to D_0. We measure the goodness of a pattern $T : \mathcal{D} \to \{0, 1\}$ by using an *impurity function* $\psi : [0, 1] \to \mathbf{R}$ that is a convex function having the maximum value at $1/2$ and the minimum value at 0 and 1, and represents the ambiguity of the split [11]. For example, the classification error $\psi_1(x) = \min(x, 1 - x)$ [16], the information entropy $\psi_2(x) = -x \log x - (1 - x) \log(1 - x)$ [22], and the Gini index functions $\psi_3(x) = 2x(1-x)$ [11] are instances of impurity functions ψ. Now, we state the *Optimized Pattern Discovery Problem* for a class $\mathcal{P}$ of patterns and with impurity function ψ as follows:

Optimized Pattern Discovery Problem. The goal of the optimized pattern discovery is to discover a pattern $T \in \mathcal{P}$ that minimizes the following cost function induced from ψ:

$$\Psi_{S,\xi}(T) = (N_1^T + N_1^F) \cdot \psi\left(\frac{N_1^T}{N_1^T + N_1^F}\right) + (N_0^T + N_0^F) \cdot \psi\left(\frac{N_0^T}{N_0^T + N_0^F}\right) \quad (1)$$

where N_α^β is the number of the trees $D \in \mathcal{D}$ that has $T(D) = \alpha$ and $\xi(D) = \beta$ for every $\alpha \in \{1, 0\}$ and $\beta \in \{T, F\}$.

Then, such a pattern T is called *optimal* w.r.t. ψ. We note that the function $\Psi_{S,\xi}(T)$ above is directly optimized in our framework, while $\Psi_{S,\xi}(T)$ is used only as a guide of greedy search in many empirical learning algorithms such as *C4.5* [22].

Furthermore, it is shown that any algorithm that efficiently solves the optimized pattern discovery problem can approximate an arbitrary unknown distribution of labeled data well within a given class of patterns [16]. Thus, the optimized pattern discovery has been extensively studied and applied to the discovery of geometric patterns or numeric association rules [13,14,17], association rule [20,23], and string patterns [5,24].

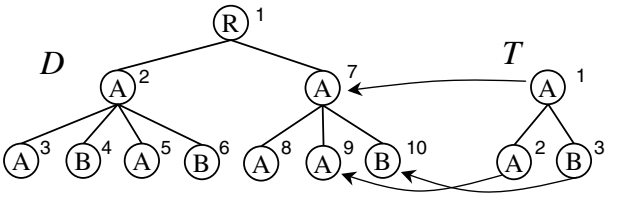


Fig. 2. A data tree D and a pattern tree T on the set $\mathcal{L} = \{A, B\}$ of labels

2.2 Labeled Ordered Trees

We define the class of labeled ordered trees as a formal model of semi-structured data and patterns [2] according to [7]. For the definitions of basic terminologies on sets, trees, and graphs, we refer to a textbook by, e.g. [4]. For a binary relation B, the *transitive closure* of B is denoted by B^+.

First, we fix a possibly infinite alphabet $\mathcal{L} = \{\ell, \ell_0, \ell_1, \ldots\}$ of *labels*. Then, *labeled ordered tree on* $\mathcal{L}$ is a rooted, connected directed acyclic graph T such that each node is labeled by an element of $\mathcal{L}$ and all node but the root have the unique parent and their children are ordered from left to right [4]. Note that the term *ordered* means the order *not* on labels but on children. More precisely, a *labeled ordered tree* of size $k \geq 0$ is represented to be a 6-tuple $T = (V, E, B, \mathcal{L}, L, v_0)$, where V is a set of nodes, $E \subseteq V^2$ is the set of edges (or the direct child relation), $B \subseteq V^2$ is the direct sibling relation, $L : V \to \mathcal{L}$ is the *labeling function*, and $v_0 \in V$ is the root of the tree. We denote the rightmost leaf of T by $rml(T)$. Whenever $T = (V, E, B, \mathcal{L}, L, v_0)$ is understood, we refer to $V, E, B, L,$, respectively, as V_T, E_T, B_T and L_T throughout this paper.

A *pattern tree* on $\mathcal{L}$ (a *pattern*, for short) is a labeled ordered tree T on $\mathcal{L}$ whose node set is $V_T = \{1, \ldots, k\}$ $(k \geq 0)$ and all nodes are numbered consecutively by the preorder traversal [4] on T. Obviously, the root and the rightmost leaf of T are 1 and k, respectively. A *k-pattern* is a pattern of size exactly k. We assume the *empty tree* $\perp$ of size zero. For every $k \geq 0$, we denote by $\mathcal{T}$, $\mathcal{T}_k$, and $\mathcal{T}^k = \cup_{i \leq k} \mathcal{T}_i$ the classes of all patterns, all patterns of size exactly k, and all pattern of size at most k on $\mathcal{L}$, respectively.

Let $(\mathcal{D}, \xi)$ be a sample consisting of a database $\mathcal{D} = \{D_1, \ldots, D_m\}$ of ordered trees on $\mathcal{L}$ and an objective attribute $\xi : \mathcal{D} \to \{0, 1\}$. Without loss of generality, we assume that V_{D_i} and V_{D_j} are disjoint if $i \neq j$. Then, a pattern tree $T \in \mathcal{P}$ *matches* a data tree $D \in \mathcal{D}$ if there exists some *order-preserving embedding* or a *matching function of* T *into* D, that is, any function $\varphi : V_T \to V_D$ that satisfies the following conditions (i)–(iv) for any $v, v_1, v_2 \in V_T$:

(i) φ is a one-to-one mapping .
(ii) φ preserves the parent relation, i.e., $(v_1, v_2) \in E_T$ iff $(\varphi(v_1), \varphi(v_2)) \in E_D$.
(iii) φ preserves the (transitive closure of) the sibling relation, i.e., $(v_1, v_2) \in (B_T)^+$ iff $(\varphi(v_1), \varphi(v_2)) \in (B_D)^+$.
(iv) φ preserves the labels, i.e., $L_T(v) = L_D(\varphi(v))$.

Algorithm OPTT
Input: An integer $k \geq 0$, a sample $(\mathcal{D}, \xi)$, and an impurity function ψ.
Output: All ψ-optimal patterns T of size at most k on $(\mathcal{D}, \xi)$.
Variable: A collection $BD \subseteq (\mathcal{T} \times (V_{\mathcal{D}})^*)$ of pairs of a pattern and its rightmost occurrences in $\mathcal{D}$, called *boundary set*, and a priority queue $R \subseteq \mathcal{T} \times \mathbf{R}$ of patterns with real weight.
1. $BD := \{ \langle \bot, RMO(\bot) \rangle \}$, where $RMO(\bot)$ is the preorder traversal of D.
2. While $BD \neq \emptyset$, do:
   (a) $\langle T, RMO(T) \rangle := Pop(BD)$;
   (b) Compute $eval := \Psi_{\mathcal{D},\xi}(T)$ using $RMO(T)$ and ξ; $R := R \cup \{ \langle T, eval \rangle \}$;
   (c) Let (x, y) be the stamp point of T and $eval_{opt}$ be the smallest $eval$ value in R. Then, if $\min(\Phi(x, 0), \Phi(0, y)) > eval_{opt}$ then the next step and go to the beginning of the while-loop.
   (d) For each $\langle S, RMO(S) \rangle \in$ Expand-A-Tree$(T, RMO(T))$, do:
      – $Push(\langle S, RMO(S) \rangle, BD)$;
3. Return all optimal patterns $\langle T, eval \rangle$ in the priority queue R.

Fig. 3. An efficient algorithm for discovering the optimal pattern of bounded size, where search strategy is either breadth-first or depth-first depending on the choice of the boundary set BD

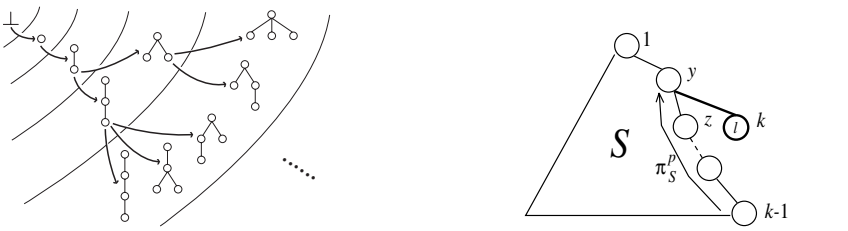
Then, we also say that T *occurs in* D. We assume that the empty tree $\bot$ matches to any tree at any node. Suppose that there exists some matching function φ of k-pattern T into a data tree $D \in \mathcal{D}$. Then, we define the *root occurrence* and the *rightmost leaf occurrence* of T in D w.r.t. φ by the node $Root(\varphi) = \varphi(1)$ and the node $Rmo(\varphi) = \varphi(k)$, respectively. We denote by $RMO_{\mathcal{D}}(T)$ the set of all rightmost leaf occurrences of T in trees of $\mathcal{D}$.

Example 1. In Fig. 2, we show examples of labeled ordered trees D and T on $\mathcal{L} = \{A, B\}$, where the node name is attached to the right corner of and a label is contained in a circle. A set of three arrows from T to D illustrates a matching function φ_1 of T to D. Then, there are two root-occurrences of T in D, namely 2 and 7, while there are three rightmost leaf occurrences $4, 6$ and 10.

In a labeled ordered tree T, the *depth* of node v, denoted by $depth(v)$, is the length of the path, the number of nodes in it, from the root to v. For every $p \geq 0$, the *p-th parent* of node v, denoted by $\pi_T^p(v)$, is the unique ancestor u of v such that the length of the path from u to v is $p+1$. Clearly, $\pi_T^0(v) = v$ itself.

3 Mining Algorithms

In this section, we present an efficient algorithm for solving the optimal pattern discovery problem for labeled ordered trees.



(a) A search graph for ordered trees (b) The (p, ℓ)-expansion of tree S

Fig. 4. The rightmost expansion for ordered trees

3.1 Overview of the Algorithm

Let us fix an impurity function ψ, and $k \geq 0$ be the maximum size of patterns. In Fig. 3, we present a mining algorithms OPTT for discovering all optimal patterns T of size at most k that minimize the cost function $\Psi_{\mathcal{D},\xi}(T)$ in the sample $(\mathcal{D}, \xi)$ for the class of labeled ordered trees of bounded size k.

In Fig. 3, a *boundary set* is a collection BD of labeled ordered trees with the push operation $Push(BD, x)$ and the pop operation $Pop(BD)$. The algorithm OPTT maintains candidate patterns in the boundary set BD to search those labeled ordered trees appearing in database $\mathcal{D}$. The algorithm and its subprocedures Expand-A-Tree (Fig. 5), Update-RMO (Fig. 6) also maintain for each candidate tree T, the list $RMO(T)$ of its rightmost occurrences in $\mathcal{D}$.

Starting with the boundary set BD containing only the empty pattern $\perp$, the algorithm OPTT searches the hypothesis space $\mathcal{T}^k = \cup_{0 \leq i \leq k} \mathcal{T}_i$ with growing candidate patterns in BD by attaching a new node one by one (Sec. 3.2). Whenever a successor $S \in \mathcal{T}$ is generated from a pattern T using the rightmost expansion, the algorithm incrementally computes the new occurrence list $RMO(S)$ of S from the old rightmost occurrence list $RMO(T)$ of T (Sec. 3.3). Repeating this process, the algorithm finally exits from the while loop and reports all optimal patterns with the smallest *eval* values in R.

3.2 Efficient Enumeration of Ordered Trees

In this subsection, we present an enumeration technique for generating all ordered trees of normal form without duplicates by incrementally expanding them from smaller to larger. This is a generalization of the itemset enumeration technique of [8], called the *set-enumeration tree*.

A *rightmost expansion* of a $(k-1)$-pattern T is any k-pattern S obtained from T by attaching a new leaf x, namely $x = k$, with a label $\ell \in \mathcal{L}$ to a node y on the rightmost branch so that k is the rightmost child of y. Then, we say S is a *successor* of T and write $T \to S$. In the case that the attached node y is the p-th parent $\pi_T{}^p(x)$ of the rightmost leaf x of T and the label of y is $\ell \in \mathcal{L}$, then S is called the (p, ℓ)-*expansion* of T ((a) of Fig. 4). An *enumeration graph* on $\mathcal{T}$ is the graph $G = (\mathcal{T}, \to)$ with the node set $\mathcal{T}$ and the node set $\to$, the corresponding successor relation over $\mathcal{T}$.

Algorithm Expand-A-Tree$(T, RMO(T))$
$\Gamma := \emptyset$;
For each pairs $(p, \ell) \in \{0, \ldots, depth(rml(T)) - 1\} \times \mathcal{L}$, do:
– S :=the (p, ℓ)-expansion of T; $RMO(S) :=$ Update-RMO$(RMO(T), p, \ell)$;
– $\Gamma := \Gamma \cup \{S, RMO(S)\}$;
Return Γ;

Fig. 5. The algorithm for computing all successors of a pattern

Theorem 1 ([7]). *The enumeration graph $(\mathcal{T}, \rightarrow)$ is a tree with the unique root $\perp$, that is, a connected acyclic graph such that all nodes but the unique root $\perp$ have exactly one parent. This is true even if we restrict nodes to $\mathcal{T}^{(k)}$.*

Using the rightmost expansion technique, Expand-A-Tree of Fig. 5 enumerates all members of $\mathcal{T}$ without duplicates using an appropriate tree traversal method.

3.3 Updating Occurrence Lists

A key of our algorithm is how to efficiently store and update the information of a matching φ of each pattern T in $\mathcal{D}$. Instead of recording the full information $\varphi = \langle \varphi(1), \ldots, \varphi(k) \rangle$, we record only the rightmost occurrences $Rmo(\varphi) = \varphi(k)$ as the partial information on φ. Based on this idea, our algorithm maintains the rightmost occurrence list $RMO(T)$ for each candidate pattern $T \in BD$.

Fig. 6 shows the algorithm Update-RMO that, given the (p, ℓ)-expansion T of a pattern S and the corresponding occurrence list $RMO(S)$, computes the occurrence list $RMO(T)$ without duplicates. This algorithm is base on the following observation: For every node y, y is in $RMO(T)$ iff there is a node x in $RMO(S)$ such that y is the strict younger sibling of the $(p-1)$-th parent of x. Although a straightforward implementation of this idea still results duplicates, the *Duplicate-Detection* technique [7] at Step 2(b) ensures the uniqueness of the elements of $RMO(T)$ (See [7], for detail).

Lemma 1 (Asai *et al.* [7]). *For a pattern S, the algorithm* Update-RMO *exactly computes all the elements in $RMO(T)$ from $RMO(S)$ without duplicates, where T is a rightmost expansion of S.*

3.4 Pruning by Convexity

Let N^T and N^F be the total numbers of positive and negative data trees in $\mathcal{D}$ and $N = N^T + N^F$. For a pattern $T \in \mathcal{T}$, a *stamp point* corresponding to T is a pair $(x, y) \in [0, N^T] \times [0, N^F]$ of integers, where $x = N_1^T$ and $y = N_1^F$ are the numbers of matched positive and negative data trees in $\mathcal{D}$. Recall that the goal is to minimize the cost function $\Psi_{S,\xi}(T)$ of Eq. 1 in Section 2.1. Since N^T and

Algorithm Update-RMO(RMO, p, ℓ)

1. Set RMO_{new} to be the empty list ε and $check := null$.
2. For each element $x \in RMO$, do:
   (a) If $p = 0$, let y be the leftmost child of x.
   (b) Otherwise, $p \geq 1$. Then, do:
      – If $check = \pi_D{}^p(x)$ then skip x and go to the beginning of Step 2 (Duplicate-Detection).
      – Else, let y be the next sibling of $\pi_D{}^{p-1}(x)$ (the $(p-1)$st parent of x in D) and set $check := \pi_D{}^p(x)$.
   (c) While $y \neq null$, do the following:
      – If $L_D(y) = \ell$, then $RMO_{\text{new}} := RMO_{\text{new}} \cdot (y)$; /* Append */
      – $y := next(y)$; /* the next sibling */
3. Return RMO_{new}.

Fig. 6. The incremental algorithm for updating the rightmost occurrence list of the (p, ℓ)-expansion of a given pattern T from that of T

N^F are constants for a fixed sample $(\mathcal{D}, \xi)$, we can regard $\Psi_{S,\xi}(T)$ as a function of a stamp point (x, y) and written as follows:

$$\Psi_{S,\xi}(T) = (x+y) \cdot \psi(\frac{x}{x+y}) + (N - (x+y)) \cdot \psi(\frac{N^T - x}{N - (x+y)}). \tag{2}$$

To emphasize this fact, we write $\Phi(x, y) \stackrel{\text{def}}{=} \Psi_{S,\xi}(T)$ as a function of (x, y). Then, Morishita [21] showed that if $\psi(\theta)$ is an impurity function, then $\Phi(x, y)$ is *convex*, i.e., for every stamp points $\mathbf{x}_1, \mathbf{x}_2 \in [0, N^T] \times [0, N^F]$, $\Phi(\alpha\,\mathbf{x}_1 + (1-\alpha)\,\mathbf{x}_1) \geq \alpha\,\Phi(\mathbf{x}_1) + (1-\alpha)\,\Phi(\mathbf{x}_2))$ for any $0 \leq \alpha \leq 1$. This means that the stamp points with optimal values locates the edges of the 2-dimensional plain $[0, N^T] \times [0, N^F]$. Thus, we have the following theorem:

Theorem 2 (Morishita and Sese [21]). *Let T be any pattern and S be any pattern obtained from T by finite application of the rightmost expansion. Let (x, y) and (x', y') be the stamp points corresponding to T and S, respectively, w.r.t. $(\mathcal{D}, \xi)$. Then,*

$$\Phi(x', y') \geq \min(\Phi(x, 0), \Phi(0, y)) \tag{3}$$

From the above theorem, we incorporate the following pruning rule in the algorithm OPTT of Fig. 3 at Step 2(c).

Convexity Pruning Rule. During the computation of OPTT, for any pattern $T \in \mathcal{T}$ with the stamp point (x, y), if $\min(\Phi(x, 0), \Phi(0, y))$ is strictly larger than the present optimal value of the patterns examined so far, then prune T and all of its successors.

4 Theoretical Analysis

4.1 The Case of Bounded Pattern Size

For a sample database $(\mathcal{D}, \xi)$, we introduce the parameters N, l, and b as the total number of nodes, the number of distinct labels, and the maximum branching factor of data trees in $\mathcal{D}$, respectively. In real databases such as collections of Web pages or XML data, we can often observe that l is not a constant but a slowly growing function $l(N)$ in N, while b is a constant.

In this setting, we can analyze the running time $T(N)$ of a straightforward generate-and-test algorithm for the optimized pattern discovery. Let $\mathcal{L}(\mathcal{D})$ be the set of labels in $\mathcal{D}$. Since there exists $\Theta(2^{ck}l(N)^k)$ distinct labeled ordered trees on $\mathcal{L}(\mathcal{D})$ for some c, if we assume $l(N) = O(N^\alpha)$ is a polynomial with degree $0 < \alpha < 1$ then the estimation of the running time is $T(N) = \Theta(2^{ck}N^{1+k\alpha})$, and thus not linear in N even if k and b are constants. In contrast, we show the following theorem on the time complexity of our algorithm OPTT, which is linear for constants k and b.

Theorem 3. *Under the above assumptions, the running time of* OPTT *on a sample* $(\mathcal{D}, \xi)$ *is bounded by* $O(k^{k+1}b^kN)$.

Proof. For the maximum pattern size K and every $0 \leq k \leq K$, let $\mathcal{C}_k$ be the set of all k-patterns and $R(k)$ be the total length of the rightmost occurrences (rmo) of the patterns in $\mathcal{C}_k$. We will estimate the upper bound of $R(k)$. First, we partition the patterns in $\mathcal{C}_k = \cup_p \mathcal{C}_{k,p}$ by the value of $0 \leq p < k$ when the pattern is generated by (p, ℓ)-expansion. Let $R(k, p)$ be the total length of the rmo of the patterns in $\mathcal{C}_{k,p}$. Then, we can show that $R(0, p) \leq N$ and $R(k, p) \leq bR(k-1)$ for any p. Since $R(k) \leq \sum_{p=0}^{k-1} bR(k-1, p)$, we have the recurrence $R(k) \leq kbR(k-1)$ for every $k \geq 0$. Solving this, we have $R(k) = O(k!\, b^k) = O(k^{k-1}b^kN)$. Since the running time of OPTT is bounded by $R = \sum_{k=1}^{K} kR(k) = O(K^{K+1}b^KN)$, the result immediately follows. □

4.2 The Case of Unbounded Pattern Size

The maximum agreement problem is a dual problem of the classification error minimization problem and defined as follows: Given a pair (D, ξ), find a pattern T that maximizes the *agreement* of T, i.e., the ratio of documents in S that is correctly classified by T.

Recently, Ben-David *et al.* [9] showed that for any $\varepsilon > 0$, there is no polynomial time $(770/767 - \varepsilon)$-approximation algorithm for the maximum agreement problem for Boolean conjunctions if $P \neq NP$. When we can use arbitrary many labels, we can show the following theorem by using the approximation factor preserving reduction [25]. For the proof of Theorem 4, please consult the full paper [1]. The proof is not difficult, but we present the theorem here for it indicates the necessity of the bound on the maximum pattern size for efficient mining.

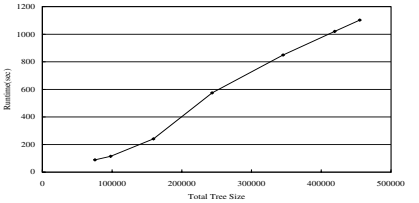


Fig. 7. The scalability: The running time with varying the input data size

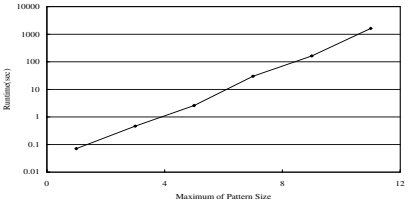


Fig. 8. The running time with varying the maximum pattern size

Table 1. Comparison of tree mining algorithms in running time and space

Algorithm	OPTT+DF	OPTT+DF+C	OPTT+BF	OPTT+BF+C	FREQT(0.1%)+BF
Time	29.7 (sec)	21.5 (sec)	20.2 (sec)	20.0 (sec)	10.4 (sec)
Space	8.0 (MB)	8.0 (MB)	96.4 (MB)	96.4 (MB)	20.7 (MB)

Theorem 4. *For any $\varepsilon > 0$, there exists no polynomial time $(770/767 - \varepsilon)$-approximation algorithm for the maximum agreement problem for labeled ordered trees of unbounded size on an unbounded label alphabet if $P \neq NP$. This is true even when either the maximum depth of trees is at most three or the maximum branching of trees is at most two.*

5 Experimental Results

We run experiments on the following two data sets *Citeseers* and *Imdb*. *Citeseers* consists of CGI generated HTML pages from an Web site[2], and *Imdb* is a collection of movie entries in XML obtained and hand-transformed from an online movie database [3]. Both data contains several hundred thousands of nodes and several thousands of unique tags. We implemented several versions of the optimized tree miner OPTT in Java (SUN JDK1.3.1 JIT) using a DOM library (OpenXML). In the experiments, the suffix BF, DF, and C following OPTT designate the versions with the breadth-first, the depth-first, and the convex pruning. All experiments were run on PC (Pentium III 600MHz, 512 MB, Linux 2.2.14).

Scalability and Running Time

Fig. 7 shows the running time with a constant maximum pattern size $k = 5$ with varying the size of the data tree from 316 KB (22,847 nodes) to 5.61 MB (402,740 nodes) on *Citeseers*. The running time seems to linearly scale on this data set for fixed k and this fits to the theoretical bound of Theorem 3.

[2] Research Index, `http://citeseer.nj.nec.com/`
[3] Internet Movie Database, `http://www.imdb.com/`

% Optimal: All action movies and some family movie have genre "action"
No. 2, Size 3, Gini 0.125, X 15/15 (Action), Y 1/15 (Family):

```
<MOVIE> <GENRE> ACTION </GENRE> </MOVIE>
```

% Optimal: Most action movie has been rated as no-one-under-15 at a country
No. 4, Size 4, Gini 0.333, X 12/15 (Action), Y 0/15 (Family):

```
<MOVIE> <CERTIFICATION> <CERTIF> 15 </CERTIF> </CERTIFICATION> </MOVIE>
```

% Frequent: Any movie is directed by someone
No. 4, Size 3, Freq 1.00, X 15/15 (Action), Y 15/15 (Family):

```
<MOVIE> <DIRECTED_BY> <PERSON> </PERSON> </DIRECTED_BY> </MOVIE>
```

Fig. 9. Examples of discovered optimal patterns

Fig. 8 shows the running time on a fixed dataset a subset of *Imdb* of size 40 KB (5835 nodes) with varying the maximum pattern tree size k from 1 to 11. Since the y-axis is log-scaled, this plot indicates that when the data size is fixed, the running time is exponential in the maximum pattern size k.

Search Strategies and Pruning Techniques

Table. 1 shows the running time of optimized tree miners OPTT+DF, OPTT+DF +C, OPTT+BF, OPTT+BF+C, and a frequent tree miner FREQT on *Imdb* data of size 40 KB. This experiment shows that on this data set, OPTT+DF saves the main memory size more than ten times than OPTT+BF, while the difference in the running time between them is not significant. Also, the use of pruning with convexity (denoted by C) in Section 3.4 is effective in the depth-first search; OPTT+DF+C is 1.5 times faster than OPTT+DF.

Examples of Discovered Patterns. In Fig. 9, we show examples of optimal patterns in XML format discovered by the OPTT algorithm by optimizing the Gini index ψ on a collection of XML entries for 15 action movies and 15 family movies from the *Imdb* dataset. Total size is 1 MB and it contains over two hundred thousands nodes. At the header, the line "No. 4, Size 4, Gini 0.333, X 12/15 (Action), Y 0/15 (Family)" means that the pattern is the 4th best pattern with Gini index 0.333 and that appears in 12/15 of action movies and 0/15 of family movies. The first optimal pattern is rather trivial and says that "*All action movies and some family movie have genre* action" and the second optimal pattern says that "*Most action movie has been rated as no-one-under-15 in at least one country.*" For comparison, we also show a frequent but trivial pattern saying that "*Any movie is directed by someone.*"

6 Conclusion

In the context of semi-structured data mining, we presented an efficient mining algorithm that discovers all labeled ordered trees that optimize a given statistical objective function on a large collection of labeled ordered trees. Theoretical analyses show that the algorithm works efficiently for patterns of bounded size. Experimental results also confirmed the scalability of the algorithm and the effectiveness of the search strategy and the pruning technique with convexity.

Acknowledgments

The authors would like to thank Akihiro Yamamoto, Masayuki Takeda, Ayumi Shinohara, Daisuke Ikeda and Akira Ishino for fruitful discussion on Web and text mining. We are also grateful to Shinichi Morishita, Masaru Kitsuregawa, Takeshi Tokuyama, and Mohammed Zaki for their valuable comments.

References

1. K. Abe, S. Kawasoe, T. Asai, H. Arimura, S. Arikawa, Optimized substructure discovery for semi-structured data, DOI, Kyushu Univ., DOI-TR-206, Mar. 2002. ftp://ftp.i.kyushu-u.ac.jp/pub/tr/trcs206.ps.gz 10
2. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kaufmann, 2000. 1, 3, 5
3. R. Agrawal, R. Srikant, Fast algorithms for mining association rules, In *Proc. VLDB'94/*, 487–499, 1994. 2
4. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983. 5
5. H. Arimura, A. Wataki, R. Fujino, S. Arikawa, An efficient algorithm for text data mining with optimal string patterns, In *Proc. ALT'98*, LNAI 1501, 247–261, 1998. 1, 4
6. H. Arimura, J. Abe, R. Fujino, H. Sakamoto, S. Shimozono, S. Arikawa, Text Data Mining: Discovery of Important Keywords in the Cyberspace, In *Proc. IEEE Kyoto Int'l Conf. on Digital Libraries*, 2000. 2
7. 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. 2, 3, 5, 8
8. R. J. Bayardo Jr. Efficiently mining long patterns from databases. In *Proc. SIGMOD98*, 85–93, 1998. 2, 7
9. S. Ben-David, N. Eiron, and P. M. Long, On the difficulty of Approximately Maximizing Agreements, In *Proc. COLT 2000*, 266–274, 2000. 10
10. L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proc. KDD-98*, 30–36, 1998. 1, 3
11. L. Devroye, L. Gyorfi, G. Lugosi, *A Probablistic Theory of Pattern Recognition*, Springer-Verlag, 1996. 1, 2, 4
12. R. Fujino, H. Arimura, S. Arikawa, Discovering unordered and ordered phrase association patterns for text mining. In *Proc. PAKDD2000*, LNAI 1805, 2000.

13. T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules. In *Proc. SIGMOD'96*, 13–23, 1996. 1, 4
14. R. C. Holte, Very simple classification rules perform well on most commonly used datasets, *Machine Learning*, 11, 63–91, 1993. 1, 4
15. A. Inokuchi, T. Washio and H. Motoda An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, In *Proc. PKDD 2000*, 13–23, 2000. 2, 3
16. M. J. Kearns, R. E. Shapire, L. M. Sellie, Toward efficient agnostic learning. *Machine Learning*, 17(2–3), 115–141, 1994. 1, 4
17. W. Maass, Efficient agnostic PAC-learning with simple hypothesis, In *Proc. COLT94*, 67–75, 1994. 1, 4
18. T. Matsuda, T. Horiuchi, H. Motoda, T. Washio, *et al.*, Graph-based induction for general graph structured data. In *Proc. DS'99*, 340–342, 1999. 1, 3
19. T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Discovery of frequent tree structured patterns in semistructured web documents. In *Proc. PAKDD-2001*, 47–52, 2001. 1, 2, 3
20. S. Morishita, On classification and regression, In *Proc. Discovery Science '98*, LNAI 1532, 49–59, 1998. 1, 4
21. S. Morishita and J. Sese, Traversing Itemset Lattices with Statistical Metric Pruning, In *Proc. PODS'00*, 226–236, 2000. 3, 9
22. J. R. Quinlan, *C4.5: Program for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, CA, 1993. 2, 4
23. R. Rastogi, K. Shim, Mining Optimized Association Rules with Categorical and Numeric Attributes, In *Proc. ICDE'98*, 503–512, 1998. 1, 4
24. H. Arimura, S. Arikawa, S. Shimozono, Efficient discovery of optimal word-association patterns in large text databases *New Gener. Comput.*, 18, 49–60, 2000. 4
25. V. V. Vazirani, *Approximaiton Algorithms*, Springer, Berlin, 1998. 10
26. W3C Recommendation. *Extensibe Markup Language (XML) 1.0*, second edition, 06 October 2000. `http://www.w3.org/TR/REC-xml`. 1, 3
27. K. Wang and H. Q. Liu. Discovering structual association of semistructured data. *IEEE Trans. Knowledge and Data Engineering (TKDE2000)*, 12(3):353–371, 2000. 1, 2, 3
28. M. J. Zaki. Efficiently mining frequent trees in a forest. Computer Science Department, Rensselaer Ploytechnic Institute, *PRI-TR01-7-2001*, 2001. `http://www.cs.rpi.edu/~zaki/PS/TR01-7.ps.gz` 2, 3