Knowledge Discovery from Semistructed Texts

Sakamoto, Hiroshi
Department of Informatics, Kyushu University

Arimura, Hiroki
Department of Informatics, Kyushu University

Arikawa, Setsuo
Department of Informatics, Kyushu University

http://hdl.handle.net/2324/3044
Knowledge Discovery from Semistructured Texts

Hiroshi Sakamoto, Hiroki Arimura, Setsuo Arikawa

Department of Informatics, Kyushu University
Hakozaki 6-10-1, Higashi-ku, Fukuoka-shi 812-8581, Japan
{hiroshi, arim, arikawa}@i.kyushu-u.ac.jp

Abstract. This paper surveys our recent results on the knowledge discovery from the semistructured texts. These texts contain heterogeneous structures which can be represented by labeled trees. The aim of our study is to extract useful information from the Web. First, we obtain the theoretical results on the learning rewriting rules between labeled trees. Second, we apply our method to the learning HTML trees in the framework of the wrapper induction. We also examined our algorithms for real world HTML texts and present the results.

1 Introduction

The present paper summarizes our study on the information extraction from semistructured texts. The HTML documents distributed on the network can be regarded as a very large text database. These markup texts are structured by many tags that have their own special meanings beforehand. Although computers cannot understand the meaning of languages, they can perform a complicated rendering using the structures. Recently the XML have been recommended by the World Wide Web Consortium (W3C), which are expected to realize more intelligent information exchange.

We begin with the investigation of the data exchange model for semistructured texts like markup texts. The aim of this study is to construct a framework of useful rewriting for tree like structures. A markup text is expressed by a rooted ordered tree. The root is the unique node which denotes the whole document, the other internal nodes are labeled by tags, and the leaves are labeled by the contents of the document or the attributes of the tags. Thus, the object considered in this paper is the translation between input and output trees. For this purpose, we produce some classes of appropriate translations and analyse their learning complexity.

For example, an XML document is translated to an HTML for the use of browsing. This document may be translated to another XML document in different format in data exchange. These translations are described by the language XSLT, which is also recommended by W3C in 1999. This language is very powerful because regular expression and recursion are allowed in this language, and thus, it seems that it is hard to learn this language from given examples alone. Thus, we introduce more restricted classes for tree translations.
We introduce two types of data exchange models in this paper. One is called the extraction and the other is called the reconstruction. The extraction is a very simple translation $T \rightarrow t$ such that a small tree $t$ is obtained by only (1) renaming labels of $T$ or (2) deleting nodes of $T$. This model is suitable for the situation that a user takes out specific entries from a very large table as a small table, or renaming a specific tag without changing the structure of the document.

On the other hand, the reconstruction is more complicated. It is characterized by term rewriting $f \rightarrow g$ for term $f$ and $g$ with variables. In this model, we can do more powerful translation of trees so that exchanging any two subtrees of an input tree and renaming labels depending on ancestors or descendants of the current node. For example, it is possible to change the order of title and author in digital books card. This translation cannot be defined by the erasing homomorphism because the order of any two node must be preserved.

The rewriting problem under the extraction is characterized by the decision problem to find the rules which maps the input tree to the output tree. The complexity of this problem is shown as well as several restricted problems. The rewriting problem under the reconstruction is clearly more difficult. Thus, we assume an additional information for this problem. We consider the learning problem such that an algorithm can use the membership query and the equivalence query [2]. We show that the rewriting class introduced is learnable in polynomial-time using the queries.

Next we apply the obtained learning theory to the real world data, like the HTML texts. The information extraction from the Web have been widely studied in the last few years. In case of the Web data, this problem is particularly difficult because we can not represent a rich logical structure by the limited tags of the HTML. The framework of wrapper induction by Kushmerick [16] is a new approach to handle this difficulty, which is a natural extension of the PAC-learning [20]. The result of his study is to show the effectiveness and efficiency of simple wrappers with string delimiters in the information extraction tasks.

In the wrapper induction, an HTML document is called a page and the contents of the page is called the label. The goal of the learning algorithm is, given the sequence of examples $(P_n, L_n)$ of pages and labels, to output the program $W$ such that $L_n = W(P_n)$ for all $n$. Other extracting models, for example, are in [10,12,13,17]. The program $W$ is called Wrapper.

In this model, we assume a special structure in the pages as follows. Every text containing in a page belongs to a class and the name of the class is called the attribute. Then, the label $L$ of a page $P$ is a set of $t_i = \langle ta_1, \ldots, ta_K \rangle$, where $ta_j$ is a set of texts contained by $P$. We call $t_i$ the $i$-th attribute. The number $K$ is a constant depending on the target. The aim of wrapper algorithm is to extract all texts from input page and classifies them into the correct attributes. For example, the strings beginning with mailto: must be the email attribute.

We propose a new wrapper class called the Tree-Wrapper over the tree structures and present the learning algorithm of the Tree-Wrappers. This is an extension of Kushmerick's LR-Wrapper [16]. The aim of the learning algorithm is to find a small tree which is a generalization of input trees.
For each node $n$ of an HTML tree, we define the node label consisting of the node name, the position number, and the set of HTML attributes. Then, the Tree-Wrapper $W$ is the sequence $(E_1, \ldots, E_k)$. The $E_i$, called the extraction path, is of the form $(ENL_{i_1}, \ldots, ENL_{i_k})$, where $ENL_{i_j}$ is called the extraction node label which is a general expression of node label by using the wild card $*$ matching any string.

For a given tree $P_i$ of an HTML page $P$ and a Tree-Wrapper $W$, the semantics for extraction is as follows. Let $EP_i = \langle ENL_{i_1}, \ldots, ENL_{i_k} \rangle$ and $p$ be a path in $P_i$ of length $\ell$. We call that $EP_i$ matches with $p$ if the node label of $j$-th node of $p$ matches with $ENL_{i_j}$ by a substitution for all $*$ of $ENL_{i_j}$. If $EP_i$ matches with $p$, then the attribute of the last node is extracted. These values are considered to be the members of $i$-th attribute in the page.

We experiment the prototype of our learning algorithm for more than 1,000 pages of HTML documents and present the performance of our algorithm. Moreover, we compare the efficiency of our model and Kushmerick's Wrapper models for sufficiently large data.

This paper is organized as follows. In Section 2, the complexity of announced decision problem is considered. We obtain the NP-completeness of this problem with respect to the restrictions either given trees are strings or output tree is labeled by a single alphabet. Moreover we show that a nontrivial subproblem is decidable in polynomial-time.

In this section we also consider the learning problem of linear translation system by query learning model. We present a learning algorithm based on the theory of $[3,4]$ and we show that our algorithm identifies each target using at most $O(m)$ equivalence queries and at most $O(kn^{2k})$ membership queries, where $m$ is the number of rules of the target and $n$ is the number of nodes of counterexamples.

In Section 3, we summarize the results on the Tree-Wrapper. First, we define the HTML trees by ordered labeled trees. Second we give the syntactic definition of the Tree-Wrapper and the semantics of the extraction. Third, we describe the learning algorithm for Tree-Wrapper. Finally, we explain the examinations of our algorithm for some popular Internet sites.

In Section 4, we conclude this study and mention the further work.

2 Tree Translation

In this section, we explain the two models for tree rewriting. First is called the extraction such that a target tree is obtained from a tree by erasing nodes. The complexity of several decision problems are presented. Second is called the reconstruction which is a kind of term rewriting systems. We introduce the class of $k$-variable linear translation and show the query learnability of this class.

2.1 Tree rewriting by erasing

We adopt the following standard definition of the ordered trees. An ordered tree is a rooted tree in which the children of each node are ordered. That is, if a node
has \( k \) children, then we can designate them as the first child, the second child, and so on up to the \( k \)-th child.

Let \( \lambda \) denote the unique null symbol not in \( \Sigma \). We define two operations on tree \( T \). One is renaming, denoted by \( a \rightarrow b \), to replace all labels \( a \) in \( T \) by \( b \). Another is deleting, denoted by \( a \rightarrow \lambda \), to remove any node \( n \) for \( \ell(n) = a \) in \( T \) and make the children of \( n \) become the children of the parent of \( n \).

Let \( S = \{ a \rightarrow b : a \in \Sigma, b \in \Sigma \cup \{ \lambda \} \} \) be a set of operations. Then, we write \( T \rightarrow_S T' \) if \( T' \) is obtained by applying all operations in \( S \) to \( T \) simultaneously.

**Definition 1.** Let \((T, P)\) be a pair of trees over \( \Sigma \). Then, the problem of erasing homomorphism is to decide whether there exists a set \( S \) of operations such that \( T \rightarrow_S P \). The input tree \( T \) is called the target and \( P \) pattern. This problem is denoted by \( \text{EHP}(T, P) \). The problem of erasing isomorphism, denoted by \( \text{EIP}(T, P) \), is to decide whether \( T \rightarrow_S P \) such that if \( a \rightarrow c, b \rightarrow c \in S \), then either \( a = b \) or \( c = \lambda \), that is, any two different symbols are never renamed to a same symbol.

When we consider the restriction that any two nodes of a pattern tree \( P \) are labeled by distinct symbols, this problem is the special case of \( \text{EIP}(T, P) \). Moreover, this problem is equivalent to the tree inclusion problem [15] which is decidable in \( O(|T| \cdot |P|) \) time.

The problem \( \text{EHP}(T, P)_k \) is a restriction of \( \text{EHP}(T, P) \) such that the depth of the tree \( T \) is at most \( k \). The problem \( \text{EIP}(T, P)_k \) is defined similarly. First, we obtain the complexity of \( \text{EIP}(T, P) \) and show that a subclass is in P. Next, we prove the NP-hardness of more general problem \( \text{EHP}(T, P) \). Recall that \( \text{EIP}(T, P)_1 \) is the problem that \( T \) and \( P \) are both strings. The following result tells us that \( \text{EIP}(T, P) \in P \) iff \( \text{EIP}(T, P)_1 \in P \).

**Theorem 1 ([18]).** \( \text{EIP}(T, P) \) is polynomial time reducible to \( \text{EIP}(T, P)_1 \).

By Theorem 1, we can reduce the \( \text{EIP}(T, P) \) to \( \text{EIP}(T, P)_1 \). Thus, it is sufficient to consider only the problem \( \text{EIP}(T, P)_1 \). In the following parts, we write \( \text{EIP}(T, P) \) instead of \( \text{EIP}(T, P)_1 \). Using this result, we derive the result that there is a subclass of \( \text{EIP}(T, P) \) to be in P.

Let \( w, \alpha, \beta \) be strings. There exists an overlap of \( \alpha \) and \( \beta \) on \( w \) if there exist occurrences \( i \) and \( j \) of \( \alpha \) and \( \beta \) on \( w \) such that \( i < j < |\alpha| + i - 1 \) or \( j < i < |\beta| + j - 1 \). If a string is of the form \( \alpha \alpha \alpha \) for some \( \alpha \in \Sigma \) and \( \alpha \) does not occur in \( \alpha \), then we call the string an interval of \( \alpha \). A string \( w \in \Sigma^* \) is called \( k \)-interval free if \( w \) contains an overlap of at most \( (k - 1) \) intervals. A string \( w \in \Sigma^* \) is said to have a split if an \( i \)-th symbol of \( w \) does not contain in any interval. The problem \( \text{EIP}(T, P) \) is denoted \( \text{EIP}(T, P)^k \) if \( T \) and \( P \) are both \( k \)-interval free. For this problem, we obtain the following positive result.

**Example 1.** The string \( \text{ABBAC} \) is an interval of \( \text{A} \) but \( \text{ABAC} \) is not. The string \( \text{ABCADB} \) contains no split because each symbol is contained in an interval of \( \text{A} \) or \( \text{B} \). On the other hand, \( \text{ACDBBB} \) has a split. The followings are example of 3-interval string and 3-interval string.

**Theorem 2 ([18]).** \( \text{EIP}(T, P)^3 \in P. \)
However, we obtain the following negative results for the general problem $EHP(T, P)$.

**Theorem 3** ([18]). The $EHP(T, P)$ is NP-complete even if (1) $P$ is labeled by a single alphabet, or (2) $T$ is a string.

### 2.2 Tree translation systems

In this subsection, we introduce the class of *ranked trees* whose node label is ranked and the out-degree of a node is bounded by the rank of its node label, where we do not allow any operations such as deletion and insertion that may change the out-degree of a node. Let $\Sigma = \cup_{n \geq 0} \Sigma_n$ be a finite ranked alphabet of *function symbols*, where for each $f \in \Sigma$, a nonnegative integer $arity(f) \geq 0$, called *arity*, is associated. We assume that $\Sigma$ contains at least one symbol of arity zero. Let $X$ be a countable set of *variables* disjoint with $\Sigma$, where we assume that each $x \in X$ has arity zero.

We denote by $T(\Sigma, X)$ the set of labeled, rooted, ordered trees $t$ such that

- Each node $v$ of $t$ is labeled with a symbol in $\Sigma \cup X$, denoted by $t(v)$.
- If $t(v)$ is a function symbol $f \in \Sigma$ of arity $k \geq 0$ then $v$ has exactly $k$ children.
- If $t(v)$ is a variable $x \in X$ then $v$ is a leaf.

We call each element $t \in T(\Sigma, X)$ a *pattern tree* (*pattern* for short).

A pattern tree is also called a first-order term in formal logic. We often write $T$ by omitting $\Sigma$ and $X$ if they are clearly understood from context. For pattern $t$, we denote the set of variables appearing in $t$ by $var(t) \subseteq X$ and define the number of the nodes of $t$ by $size(t)$. A pattern $t$ is said to be a *ground pattern* if it contains no variables.

**Definition 2.** A tree translation rule (rule for short) is an ordered pair $(p, q) \in T \times T$ such that $var(p) \supseteq var(q)$. We also write $(p \rightarrow q)$ for rule $(p, q)$. A tree translation system $(TT)$ is a set $H$ of translation rules.
A pattern $t$ is called linear if any variable $x \in X$ appears in $t$ at most once. A pattern $t$ is of $k$-variable if $\text{var}(t) = \{x_1, \ldots, x_k\}$. For $k \geq 0$, we use the notation $t[x_1, \ldots, x_k]$ to indicate that pattern $t$ is a $k$-variable linear pattern with mutually distinct variables $x_1, \ldots, x_k \in X$, where the order of variable in $t$ is arbitrary. For $k$-variable linear pattern $t[x_1, \ldots, x_k]$ and a sequence of patterns $s_1, \ldots, s_k$, we define $t[s_1, \ldots, s_k]$ as the term obtained from $t$ by replacing the occurrence of $x_i$ with patterns $s_i$ for every $1 \leq i \leq k$.

**Definition 3.** A translation rule $C = (p, q)$ is of $k$-variable if $\text{card}(\text{var}(C)) \leq k$, and linear if both of $p$ and $q$ are linear.

For every $k \geq 0$, we denote by $LR(k)$ and $\text{LTT}(k)$ the classes of all $k$-variable linear translation rules, and all $k$-variable linear tree translation systems, respectively. We also denote by $\text{LTT} = \cup_{k \geq 0} \text{LTT}(k)$ all linear tree translation systems.

**Definition 4.** Let $H \in \text{LTT}$ be a linear translation system. The translation relation defined by $H$ with the set $M(H) \subseteq \mathcal{T} \times \mathcal{T}$ is defined recursively as follows.

- Identity: For every pattern $p \in \mathcal{T}$, $(p, p) \in M(H)$.
- Congruence: If $f \in \Sigma$ is a function symbol of arity $k \geq 0$ and $(p_i, q_i) \in M(H)$ for every $i$ then $(f(p_1, \ldots, p_k), f(q_1, \ldots, q_k)) \in M(H)$.
- Application: If $(p[x_1, \ldots, x_i], q[x_1, \ldots, x_i]) \in H$ is a $k$-variable linear rule, and $(p_i, q_i) \in M(H)$ for every $i$ then $(p[p_1, \ldots, p_k], q[q_1, \ldots, q_k]) \in M(H)$, where note that $p$ and $q$ are $k$-variable linear terms.

If $C \in M(H)$ then we say that rule $C$ is derived by $H$. The definition of the meaning $M(H)$ above corresponds to the computation of top-down tree transducer [8] or the a special case of term rewriting relation [7] where only top-down rewriting are allowed.

We show that there exists a polynomial time algorithm that exactly identifies any translation system in $\text{LTT}(k)$ using equivalence and membership queries. Our problem is identifying an unknown tree translation system $H_*$ from examples of ordered pairs $E \in M(H_*)$ that are either derived or not derived by $H_*$. As a formal model, we employ a variant of exact learning model by Angluin [2] called learning from entailment [3, 4, 9, 14], which is tailored for translation systems.

Let $\mathcal{H}$ be a class of translation systems to be learned, called hypothesis space, and $LR$ be the set of all ordered pairs, called the domain of learning. In our learning framework, the meaning or the concept represented by $H \in \mathcal{H}$ is the set $M(H_*)$. If $M(P) = M(Q)$ then we define $P \equiv Q$ and say that $P$ and $Q$ are equivalent.

A learning algorithm $A$ is an algorithm that can collect the information about $H_*$ using the following type of queries. In this paper, we assume that the alphabet $\Sigma$ is given to $A$ in advance and the maximum arity of symbols in $\Sigma$ is constant.
Definition 5. An equivalence query (EQ) is to propose any translation system $H \in \mathcal{H}$. If $H \equiv H_*$ then the answer to the query is “yes”. Otherwise the answer is “no”, and $A$ receives any translation $C \in LR$ as a counterexample such that either $C \in M(H_*) \setminus M(H)$, or $C \in M(H) \setminus M(H_*)$. A counterexample is positive if $C \in M(H_*)$ and negative if $C \not\in M(H_*)$. A membership query (MQ) is to propose any translation $C \in LR$. The answer to the membership query is “yes” if $C \in M(H_*)$, and “no” otherwise.

The goal of $A$ is exact identification in polynomial time. $A$ must halt and output a rewriting system $H \in \mathcal{H}$ such that $H_* \equiv H$, where at any stage in learning, the running time and thus the number of queries must be bounded by a polynomial $\text{poly}(m, n)$ in the size $m$ of $H_*$ and the size $n$ of the longest counterexample returned by equivalence queries so far.

Although this setting first seems to be unnatural, it is known that any exact learnability with equivalence queries implies polynomial time PAC-learnability [20] and polynomial time online learnability [2] under a mild condition on the class of target hypothesis whether additional membership queries are allowed or not [2].

Theorem 4 ([18]). There exists an algorithm which exactly identifies any translation system $H_*$ in LTT $(k)$ using $O(m)$ equivalence queries and $O(kn^{2k})$ membership queries.

3 Wrappper Induction

In this section, we give the define of the HTML tree, the learning algorithm for the Tree-Wrapper, and the experimental result for the real world data.

3.1 Data model

For each tree $T$, the set of all nodes of $T$ is a subset of $IN = \{0, \ldots, n\}$ of natural numbers, where the 0 is the root. A node is called a leaf if it has no child and called an internal node otherwise. If $n, m \in IN$ has the same parent, then $n$ and $m$ are sibling and $n$ is a left sibling of $m$ if $n \leq m$. The sequence $(n_1, \ldots, n_k)$ of nodes of $T$ is called the path if $n_1$ is the root and $n_i$ is the parent of $n_{i+1}$ for all $i = 1, \ldots, k - 1$.

For a node $n$, the node label of $n$ is the triple $NL(n) = \langle N(n), V(n), HAS(n) \rangle$ such that $N(n)$ and $V(n)$ are strings called the node name and node value, respectively, and $HAS(n) = \{HA_1, \ldots, HA_{n_i}\}$ is the set of the HTML attributes of $n$, where each $HA_i$ is of the form $\langle a_i, v_i \rangle$ and $a_i, v_i$ are strings called HTML attribute name, HTML attribute value, respectively.

If $N(n) \in \Sigma^+$ and $V(n) = \varepsilon$, then $n$ is called the element node and the string $N(n)$ is called the tag. If $N(n) = \varepsilon$ for the reserved string $\varepsilon$TEXT and $V(n) \in \Sigma^+$, then $n$ is called the text node and the $V(n)$ called the text value. We assume that every node $n \in IN$ is categorized to the element node or text node.
An HTML document is called a page. A page $P$ is corresponding to an ordered labeled tree. For the simplicity, we assume that the $P$ contains no comment part, that is, any string beginning the `<!` and ending the `>` is removed.

**Definition 6.** For a page $P$, the $P_i$ is the ordered labeled tree defined recursively as follows.

1. Each empty tag $<$tag$>$ in $P$ corresponds to a leaf $n$ in $P_i$ such that $NL(n) = \langle N(n), V(n), HAS(n) \rangle$, $N(n) = \text{tag}$, $V(n) = \varepsilon$, and $HAS(n) = \emptyset$.
2. Each string $w$ in $P$ containing no tag corresponds to a leaf $n$ such that $N(n) = \varepsilon$, $V(n) = w$, and $HAS(n) = \emptyset$.
3. Each string of the form $<$tag $a_1 = v_1$, $\ldots$, $a_t = v_t$ $>$ $<$tag$>$ corresponds to a subtree $t = n(n_1, \ldots, n_k)$ such that $N(n) = \text{tag}$, $V(n) = \varepsilon$, and $HAS(n) = \{(a_1, v_1), \ldots, (a_t, v_t)\}$, where $n_1, \ldots, n_k$ are the roots of the trees $t_1, \ldots, t_k$ obtained recursively from the $w$ by the 1, 2 and 3.

What the HTML Wrapper of this paper extracts is the text values of text nodes. These text nodes are called text attributes. A sequence of text attributes is called tuple. We assume that the contents of a page $P$ is a set of tuple $t_i = \langle t_{a_1}, \ldots, t_{a_K} \rangle$, where the $K$ is a constant for all pages $P$. It means that all text attributes in any page is categorized into at most $K$ types. Let us consider the example of an address list. This list contains three types of attributes, name, address, and phone number. Thus, a tuple is of the form $\langle \text{name, address, phone} \rangle$. However, this tuple can not handle the case that some elements contain more than two values such as some one has two phone numbers. Thus, we expand the notion of tuple to a sequence of a set of text attributes, that is $t = \langle t_{a_1}, \ldots, t_{a_K} \rangle$ and $t_{a_i} \subseteq \mathbb{N}$ for all $1 \leq i \leq K$. The set of tuples of a page $P$ is called the label of $P$.

**Example 2.** The Fig.1 denotes the tree containing the text attributes name, address, and phone. The first tuple is $t_1 = \{(3), \{4\}, \{5, 6\}\}$ and the second tuple is $t_2 = \{(8), \{\}\}, \{9\}\}$. The third attribute of $t_1$ contains two values and the second attribute of $t_2$ contains no values.

![Fig. 2. The tree of the text attributes, name, address, and phone.](image-url)
3.2 Tree-Wrapper

Next we explain the wrapper algorithm and the learning algorithm. The wrapper algorithm extracts the attributes from the page $P_{i}$ using a Tree-Wrapper $W$. On the other hand, the learning algorithm finds the Tree-Wrapper $W$ for the sequence $E = \ldots, \langle P_{n}, L_{n} \rangle, \ldots$ of examples, where $L_{n}$ is the label of the page $P_{n}$.

Definition 7. The extraction node label is a triple $ENL = \langle N, Pos, HAS \rangle$, where $N$ is a node name, $Pos \in N \cup \{\ast\}$, $HAS$ is an HTML attribute set. The extraction path is a sequence $EP = \langle ENL_{1}, \ldots, ENL_{\ell} \rangle$.

The first task of the wrapper algorithm is to find a path in $P_{i}$ which matches with the given $EP$ and to extract the text value of the last node of the path. The matching semantics is defined as follows.

Let $ENL$ be an extraction node label and $n$ be a node of a page $P_{i}$. The $ENL$ matches with the $n$ if $ENL = \langle N, Pos, HAS \rangle$ such that (1) $N = N(n)$, (2) $Pos$ is the number of the left siblings $n'$ of $n$ such that $N(n') = N(n)$ or $Pos = \ast$, and (3) for each $(a_{i}, v_{i}) \in HAS(n)$, either $(a_{i}, v_{i}) \in HAS$ or $(a_{i}, \ast) \in HAS$.

Moreover, let $EP = \langle ENL_{1}, \ldots, ENL_{\ell} \rangle$ be an extraction path and $p = \langle n_{1}, \ldots, n_{\ell} \rangle$ be a path of a page $P_{i}$. The $EP$ matches with the $p$ if the $ENL_{i}$ matches with $n_{i}$ for all $i = 1, \ldots, \ell$.

Intuitively, an $EP$ is a general expression of all paths $p$ such that $p$ is an instance of $EP$ under a substitution for $\ast$ in $EP$.

Definition 8. The Tree-Wrapper is a sequence $W = \langle EP_{1}, \ldots, EP_{K} \rangle$ of extraction paths $EP_{i} = \langle ENL_{i1}, \ldots, ENL_{i\ell_{i}} \rangle$, where each $ENL_{i\ell_{i}}$ is an extraction label.

Then, we briefly explain the wrapper algorithm for given a tree wrapper $W = \langle EP_{1}, \ldots, EP_{K} \rangle$ and a page $P_{i}$. This algorithm outputs the label $L_{i} = \{t_{1}, \ldots, t_{m}\}$ of $P_{i}$ as follows.

1. For each $EP_{i}$, find all paths $p = \langle n_{1}, \ldots, n_{\ell} \rangle$ of $P_{i}$ such that $EP_{i}$ matches with $p$ and add the pair $(i, n_{\ell})$ into the set $Att$.
2. Sort all elements $(i, n_{\ell}) \in Att$ in the increasing order of $n_{\ell}$’s. Let $LIST$ be the list and $j = 1$.
3. If the length of $LIST$ is 0 or $j > m$, then halt. If not, find the longest prefix list of $LIST$ such that all element is in non-decreasing order of $i$ of $(i, n)$ and for all $i = 1, \ldots, K$, compute the set $ta_{i} = \{n | (i, n) \in list\}$. If the list is empty, then let $ta_{i} = \emptyset$.
4. Let $t_{j} = \langle ta_{1}, \ldots, ta_{K} \rangle$, $j = j + 1$, remove the list from $LIST$ and go to 3.

3.3 The learning algorithm

Let $\langle P_{n}, L_{n} \rangle$ be a training example such that $L_{i} = \{t_{1}, \ldots, t_{m}\}$ and $t_{i} = \langle ta_{1}, \ldots, ta_{K} \rangle$. The learning algorithm calls the procedure to find the extraction path $EP_{j}$ for the $j$-th text attribute as follows.
The procedure computes all paths \( p_i \) from the node \( n \in ta_i \) to the root, where \( 1 \leq i \leq m \). For each \( p_i \), set \( EP_i \) be the sequence of node labels of \( p_i \). Next, the procedure computes the composition \( EP \) of all \( EP_i \) and sets \( EP_j = EP \). The definition of the composition of extraction paths is given as follows. Fig. 3 is an example for a composite of two extraction path.

**Definition 9.** Let \( HAS_1 \) and \( HAS_2 \) be sets of HTML attributes. The common HTML attribute set \( CHAS \) of \( HAS_1 \) and \( HAS_2 \) is the set of HTML attributes such that \( \langle a, v \rangle \in CHAS \) iff \( \langle a, v \rangle \in HAS_1 \cap HAS_2 \) and \( \langle a, * \rangle \in CHAS \) iff \( \langle a, v_1 \rangle \in HAS_1 \), \( \langle a, v_2 \rangle \in HAS_2 \), and \( v_1 \neq v_2 \).

**Definition 10.** Let \( ENL_1 \) and \( ENL_2 \) be extraction node labels. The composition of \( ENL_1 \cdot ENL_2 \) is \( ENL = \langle N, Pos, HAS \rangle \) such that (1) \( N = N_1 \) if \( N_1 = N_2 \) and \( ENL \) is undefined otherwise, (2) \( Pos = Pos_1 \) if \( Pos_1 = Pos_2 \), and \( Pos = * \) otherwise, and (3) \( HAS \) is the common HTML attribute set of \( HAS_1 \) and \( HAS_2 \).

**Definition 11.** Let \( EP_1 = \langle ENL_1^1, \ldots, ENL_1^\ell \rangle \) and \( EP_2 = \langle ENL_2^1, \ldots, ENL_2^m \rangle \) be extraction paths. The \( EP = EP_1 \cdot EP_2 \) is the longest sequence \( \langle ENL_i^1 \cdot \ldots \cdot ENL_i^\ell \rangle \) such that all \( ENL_i^1 \cdot ENL_i^2 \) are defined for \( i = 1, \ldots, \ell \), where \( \ell \leq \min\{n, m\} \).

![Fig. 3. The composition of extraction paths.](image)

### 3.4 Experimental results

We equip the learning algorithm by Java language and experiment with this prototype for HTML documents. For parsing HTML documents, we use the OpenXML 1.2 (http://www.openxml.org) which is a validating XML parser written in Java. It can also parse HTML and supports the HTML parts of the DOM (http://www.w3.org/DOM).
The experimental data of HTML pages is collected by the citeseers which is a scientific literature digital library (http://citeseers.nj.nec.com). The data consists of 1,300 HTML pages. We chose the title, the name of authors, and the abstract as the first, the second, and the third attributes. All pages are indexed to be $P_1, \ldots, P_{1300}$ in the order of the file size. The training example is $E = \{(P_i, L_i) \mid i = 1, \ldots, 10\}$, where the $L_i$ is the label made from the $P_i$ in advance. The result is shown in Fig. 4 which is the Tree-Wrapper $W$ found by the learning algorithm.

**Fig. 4. The Tree-Wrapper found by the learning algorithm**

Next, we practice the obtained Tree-Wrapper for the remained pages $P_i$ ($i = 11, \ldots, 1300$) to extract all tuples from $P_i$. The three pages can not be extracted. The $P_{1305}$ is one of the pages. We explain the reason by this page. In Fig. 4, we can find that the first extraction path $EP_1$ contains the extraction node label for the `TABLE` tag. The HTML attribute set $HAS$ of this node contains the attribute “cellpadding” whose value is 0. However, the corresponding node in $P_{1305}$ has the HTML attribute value “cellpadding= 1”. Thus, the $EP_1$ does not match with the path. Any other pages are exactly extracted, thus, this algorithm is effective for this site.

Moreover we examine the performance of the Tree-Wrapper for several Internet sites and compare the expressiveness of Tree-Wrapper and Kushnerick’s LR-Wrapper. One of the results is shown in the Fig. 5. We select 9 popular search engine sites and 1 news site and obtained text data by giving them keywords concerned with computer science. For each site, we made two sets of training data and test data. An entry of the form $n(m)$ of Fig. 5, for example 2(260), means that $n$ tuples of training samples are sufficient to learn the site by the wrapper class and the learning time is in $m$ milli-seconds. The symbol $F$ means that the algorithm could not learn the wrapper for the site even though using
all training samples. This figure shows that almost sites can be expressed by Tree-Wrapper and the learning algorithm learn the Tree-Wrappers within a few samples. The learning time is about 2 or 3 times slower than the LR-Wrapper learning algorithm. Thus, we conclude that the Tree-Wrapper class is efficient compared with the LR-Wrapper.

<table>
<thead>
<tr>
<th>Resource &amp; URL</th>
<th>LR-Wrapper</th>
<th>Tree-Wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ALTA VISTA  (<a href="http://www.altavista.com/">www.altavista.com/</a>)</td>
<td>F</td>
<td>2 (260)</td>
</tr>
<tr>
<td>2. excite  (<a href="http://www.excite.com/">www.excite.com/</a>)</td>
<td>F</td>
<td>3 (236)</td>
</tr>
<tr>
<td>3. LYCOS  (<a href="http://www.lycos.com/">www.lycos.com/</a>)</td>
<td>F</td>
<td>2 (243)</td>
</tr>
<tr>
<td>4. Fast Search  (<a href="http://www.fast.no/">www.fast.no/</a>)</td>
<td>2 (101)</td>
<td>2 (247)</td>
</tr>
<tr>
<td>5. HOT BOT  (hotbot.lycos.com/)</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>6. WEB CRAWLER  (<a href="http://www.webcrawler.com/">www.webcrawler.com/</a>)</td>
<td>F</td>
<td>2 (182)</td>
</tr>
<tr>
<td>8. ARGOS  (<a href="http://www.argos.evansville.edu/">www.argos.evansville.edu/</a>)</td>
<td>2 (45)</td>
<td>2 (313)</td>
</tr>
<tr>
<td>9. Google  (<a href="http://www.google.com/">www.google.com/</a>)</td>
<td>F</td>
<td>2 (225)</td>
</tr>
<tr>
<td>10. Kyodo News  (<a href="http://www.kyodo.co.jp/">www.kyodo.co.jp/</a>)</td>
<td>3 (55)</td>
<td>1 (144)</td>
</tr>
</tbody>
</table>

Fig. 5. The comparison of the number of training samples and the learning time (ms) of LR-Wrapper and Tree-Wrapper. The symbol F means that the learning is failed.

4 Conclusion

We presented the results of our study on information extraction from semistructured texts. First we investigated the theory of rewriting system for labeled trees. The two models for rewriting trees were introduced. One is extraction defined by erasing nodes. The other is reconstruction defined by a restriction of translation system. For the extraction model, the complexity of the decision problem of finding a rewriting rule between two trees was proved to be NP-complete with respect to several restricted conditions. On the other hand, we proved that there exists a sub-problem in P. For the reconstruction model, we presented the polynomial time learning algorithm to learn the class of k-variable linear translation systems using membership and equivalence queries. Second, in order to apply our algorithm to the real world data, we restricted our data model and introduced the Tree-Wrapper class to express the HTML texts. In the framework of Kushmerick's wrapper induction, we constructed the learning algorithm for the Tree-Wrapper and examined the performance of our algorithm. In particular we showed that Tree-Wrapper can express almost data which cannot be expressed by LR-Wrapper.

References

8. F. Drewes, Computation by Tree Transductions, Ph D. Thesis, University of Bremen, Department of Mathematics and Informatics, February 1996.