Efficient Learning of Semi-structured Data from Queries

Arimura, Hiroki
PRESTO, Japan Science and Technology Co. | Dept. of Informatics, Kyushu University

Sakamoto, Hiroshi
Dept. of Informatics, Kyushu University

Arikawa, Setsuo
Dept. of Informatics, Kyushu University

出版情報：DOI Technical Report. 191, 2001-05. Department of Informatics, Kyushu University
バージョン：
権利関係：

DOI-TR-191 Department of Informatics, Kyushu Univeristy, May 2001

Efficient Learning of Semi-structured Data from Queries

Hiroki Arimura [1,2] Hiroshi Sakamoto [1] Setsuo Arikawa [1]

[1] Dept. of Informatics, Kyushu University, Fukuoka 812-8581, Japan
[2] PRESTO, Japan Science and Technology Co., Japan
{arim, hiroshi, arikawa}@i.kyushu-u.ac.jp

Abstract

This paper studies the learning complexity of classes of structured patterns for HTML/ XML-trees in the query learning framework of Angluin. We present polynomial time learning algorithms for *ordered gapped tree patterns*, OGT, and *ordered gapped forests*, OGF, under the into-matching semantics using equivalence queries and subset queries. As a corollary, the learnability with equivalence and membership queries is also presented. This work extends the recent results on the query learning for ordered forests under onto-matching semantics by Arimura *et al.* (1997), for unordered forests under into-matching semantics by Amoth, Cull, Tadepalli (1999), and for regular string patterns by Matsumoto and Shinohara (1997).

Keywords:
Exact learning, Query learning, polynomial time complexity,
Pattern languages, tree patterns, information extraction

Correspondence:
Hiroki Arimura
Department of Informatics, Kyushu University
6-10-1 Hakozaki, Fukuoka 812-8581, Japan
EMAIL: arim@i.kyushu-u.ac.jp
TEL: +81-92-642-2688, FAX: +81-92-642-2698

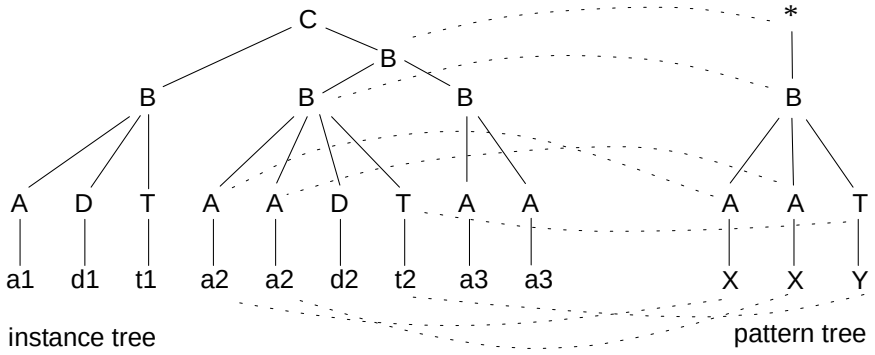

Figure 1: An ordered tree and an ordered gapped tree pattern

1 Introduction

Huge amount of electronic data have been available on the Web. They are generated automatically from databases, exchanged through networks, inspected by human users, or processed by application programs for further use. These heterogeneous and huge collections of electronic data and called *semi-structured data* and attract remarkable attention in network and database communities. XML [19] is a simple data format for semi-structured data modeled as ordered node-labeled trees associated with a set of user-defined tags.

In this paper, we introduced the class *OGT* of *ordered gapped tree patterns with the into-match semantics* (Fig. 1) by generalizing the class of ordered tree patterns with the into-match semantics of Amoth, Cull, and Tadepalli [3] and the class of regular pattern languages [5, 9, 14]. OGT are simplifications of tree pattern languages used in recent query languages for semi-structured data and XML data, such as Lorel [1], UnQL [10], and XML-QL [20]. We also introduced unordered forests with the into-match semantics as unions (sets) of patterns in OGT.

Learning of tree patterns dates back to Plotkin [17], where OT with the onto-semantics considered and shown to be polynomial time learnable from examples or equivalence query (EQ), alone. Arimura *et al.* [7] extended Plotkin's algorithm for bounded number of ordered forests. Page and Frisch [16] showed that a class of OT with background theory is polynomial time learnable by a similar algorithm. Arimura *et al.* [8] and Amoth *et al.* [4] showed that ordered forests OF with the onto semantics is learnable using EQ and subset queries (SQs). Frazier and Pitt [11] introduced the notion of learning from entailment, and presented that a class of description logic called CLASSIC, a class of labeled DAG, is learnable with EQ and membership queries (MQ), (or entailment membership queries, EntMQ). Amoth, Cull, and Tadepalli [3] introduced the into-matching semantics and unordered trees (UT) and unordered forests (UF) are polynomial time learnable with EQ and SQ in this semantics.

In this paper, we study polynomial time learning of ordered gapped tree patterns (OGT) and ordered gapped forests (OGF) in the framework of exact learning with equivalence and subset/membership queries. Based on the learning techniques developed by Amoth *et al.* [3] and Matsumoto and Shinohara [14], we presented a polynomial time learning that exactly learns the class *OGT* with the into-matching semantics using equivalence queries EQ and subset queries SQ when the alphabet size is infinite. We also show that the subset queries are replaced with standard membership queries since the alphabet is infinite in the above results.

As summary, our results generalize the polynomial time learnability of Amoth *et al.* [3] and Matsumoto and Shinohara [14]. It seems that even the polynomial time learnability of OT, a subclass of OGT, has not been shown so far though our techniques are mostly from [3, 14]. Our results also provide a theoretical model of information extraction from

Web and XML [18], and indicates possibility and limitations of such task.

This paper is organized as follows. In Section 3, we briefly review basic definitions and results on ordered tree patterns and related concept classes. In Section 3, we present the polynomial time learning algorithm for ordered gapped trees OGT with EQ and SQ. In Section 4, we extends our algorithm for ordered gapped forests OGF with EQ and SQ. In Section 5, we conclude.

2 Preliminaries

Let Σ, Γ, and $\mathbf{X}$ be mutually disjoint alphabets. We assume that $\mathbf{X}, \Gamma$ are countable and Σ is possibly countable. We refer to elements of Σ as *constant labels*, which will be denoted by $a, b, f, g, a_1, a_2, \ldots$, to elements of Γ as *gaps* (gap-variables or path-variables), denoted by $*, \gamma, \gamma_1, \gamma_2, \ldots$, and to elements of $\mathbf{X}$ as *variables* (or tree-variables), denoted by $x, y, x_1, x_2, \ldots$. Intuitively, gaps are wildcards on a path or sequence of tags that matches a sequence of nodes or labels, and variables matches subtrees.

For a set A, we denote by A^* and A^+ the sets of all possibly empty strings and all nonempty strings over A. $\#A$ denotes the cardinality of A.

2.1 Ordered Tree Patterns In this subsection, we introduce ordered gapped tree patterns, OGT for short. Throughout this paper, we consider *ordered trees* (*trees*, for short) [2] with nodes labeled by elements of some alphabet. We assume that trees are nonempty. An ordered trees t is not *ranked*, i.e., each internal node of t may have arbitrary many children independent of its label.

For an ordered tree t, we denote by V_t the set of all nodes of t, and by $|t|$ the *size* of t, the number of all nodes in t. For a tree pattern t and a node $v \in V_t$, $\ell_t(v) \in \Sigma \cup \Gamma \cup \mathbf{X}$ denotes the label of v in t, and t/v denotes the subtree of t rooted at v. A node is a *chain node* if it has exactly one child.

An *ordered gapped tree pattern* (*tree pattern* or OGT, for short) is an ordered tree t with internal node labeled by elements of $\Sigma \cup \Gamma$ and with leaves labeled by labels in Σ or tree-variables in $\mathbf{X}$. We assume that all gap-variables appearing in t are mutually distinct, and thus OGT is repetition-free in gap-variables. A variable x in t is *repeated* if x occurs in t more than once, and a *solitary* otherwise. An OGT is a *ordered tree pattern* (OT or a tree pattern, for short) [3] if it contains no gap-variables and a *constant tree* (a tree, for short) if it contains no tree-variables and no gap-variables. In Fig. 1, we show examples of a constant tree and an ordered gapped tree pattern, where the dotted lines indicate a matching from the tree pattern to the constant tree.

We denote by T, OT, OGT the classes of all ordered constant trees over Σ, all ordered tree patterns over $\Sigma \cup \mathbf{X}$, and all ordered gapped tree patterns over $\Sigma \cup \Gamma \cup \mathbf{X}$.

For a tree pattern t, $var(t)$ and $gap(t)$ denote the sets of all tree-variables and all gap-variables, respectively, appearing in t. For a node v in t, v is a *constant node* if its label $\ell_t(v)$ is constant in t, a *variable node* if its label is a tree-variable in t, and a *gap node* if its label is a gap-variable.

We next introduce basic operations on trees. Let t be an ordered tree and v be a node. The *delete* operation removes v from its parent and the resulting tree is denoted by $delete(t, v)$. The *change* operation changes the label $\ell_t(v)$ of v to a new label ℓ and is denoted by $change(t, v, \ell)$. The change operation also changes the whole subtree rooted at v to a new tree s and is denoted by $change(t, v, s)$. The *contract* operation contracts the tree t by eliminating a chain node v and directly connects the parent of t and the unique child of v and is denoted by $contr(t, v)$.

2.2 Matching semantics We introduce two matching semantics for ordered trees according to Amoth *et al.* [3].

Definition 1 An ordered tree s *matches* another ordered tree t in the *into-matching semantics*, denoted by $s \sqsupseteq_i t$, if there exists a one-to-one mapping $\phi : V_s \to V_t$ from the nodes of s into the nodes of t such that (1) The root of s maps to the root of t, (2) If u maps to v and u is an internal node with k children, then there exists some $1 \leq j_1 < \cdots < j_k \leq k$ such that the i-th child of u maps to the j_i-th child of v for every $1 \leq i \leq k$. (3) If u maps to v and u have a constant label $\ell_s(u) \in \Sigma$, then u and v have the same labels. (4) If u_1 and u_2 are labeled with the same variable $\ell_t(u_1) = \ell_t(u_2) \in \mathbf{X}$ and u_1 and u_2 map v_1 and v_2, respectively, then v_1 and v_2 have the identical subtrees. (5) If u has a gap-variable $\ell_s(u) \in \Gamma$ and its parent $p(u)$ maps to v, then u maps to a *proper* descendant of v.

The one-to-one mapping ϕ above is called a *matching* from s to t. If $s \sqsupseteq_i t$ then t is called an *instance* of s, or s is a *generalization* of t. If $s \sqsupseteq_i t$ and $t \sqsupseteq_i s$ then we define $s \equiv t$ and say s is equivalent to t. If $s \sqsupseteq_i t$ but $t \not\sqsupseteq_i s$ then we define $s \sqsupset_i t$ and say t is a proper instance of s, or s is a proper generalization of t.

Lemma 1 *For any $s, t \in OGT$, $s \equiv t$ if and only if s and t are identical modulo renaming of variables on $\Gamma \cup \mathbf{X}$.*

Note that when restricted to OT, the matching semantics defined above with conditions (1)–(4) coincides to the the into-match semantics of [3]. When restricted to OT, if the range of the mapping ϕ is the set of all nodes of t then we refer to the resulting semantics as the *onto-matching semantics* and write $s \sqsupseteq_o t$, which is a standard matching semantics in first-order logic and term rewriting systems. [1] Intuitively speaking, condition (5) says that a nonempty path can be substituted for a gap-variable, and this definition relates to the *non-erasing substitution semantics* for pattern languages [5, 14]. As an alternative definition, we may allow an empty path to be substituted for a gap-variable, and then this definition yeilds the *erasing substitution semantics* [15]. However, we do not consider the latter case. If the order among matching positions $j_1, \ldots, j_k$ for children is arbitrary in condition (3) of the above definition, then we refer to the resulting tree patterns as the *unordered tree patterns* (UT, for short) with the into-matching semantics. See Amoth *et al.* [3] for UT.

For an ordered tree pattern t, we define the *language* of t as the set $L_i(t)$ consisting of all ordered constant trees $w \in T$ that is an instance of t with the into-match semantics: $L_i(t) = \{ w \in T \mid t \text{ into-matches } w \}$.

Note that even if t is a constant tree, the set $L_i(t)$ may be infinite with the into-semantics. We denote by OT and UT the classes of ordered tree patterns and unordered tree patterns over $\Sigma \cup \mathbf{X}$, respectively. Since we consider only the ordered tree patterns with the into-semantics, we may omit the subscript i in $\sqsupseteq_i$ and $L_i(t)$ if it is clear from the context.

Lemma 2 *For any ordered tree patterns $s, t \in OGT$ and constant trees $u, w \in T$, the following properties hold.*

1. *The into-match relation $\sqsupseteq$ is reflexive and transitive.*

[1] We can extend the onto-match semantics for OGT by changing the definition as that the range of the mapping ϕ is the set of all but the mapped-gap nodes, where a node is a *mapped-gap node* if for a gap-node u and its parent p, it appears on the path between the children of $\phi(p)$ to $\phi(u)$.

2. *If* $s \sqsupseteq t$, *then* $|s| \leq |t|$
3. *If* $w \in L(s)$, *then* $|s| \leq |w|$
4. *If* $s \sqsupseteq t$, *then* $L(s) \supseteq L(t)$.
5. *If* Σ *is infinite, then* $s \sqsupseteq t$ *if and only if* $L(s) \supseteq L(t)$.
6. *If* $|\Sigma| \geq 2$ *then,* $s \equiv t$ *if and only if* $L(s) = L(t)$.

The *into-matching problem* for OGT is the problem to decide if a pattern P into-matches a tree T. The onto-matching problem for OT is polynomial time solvable [13, 17]. The into or onto-matching problems for the class UT of *unordered* tree patterns are NP-complete [3].

Proposition 3 *The into-matching problems for OT and thus OGT are NP-complete.*

Proof: It is easy to see that the problem belongs to NP. Conversely, we give a log-space reduction from the one-in-three SAT problem into the into-matching problem for OGT as follows. Let $V = \{x_1, \ldots, x_n\}$ be the set of Boolean variables and $F = \{C_1, \ldots, C_m\}$ be an instance CNF. We use alphabets $\Sigma = \{F, V, C_1, \ldots, C_m, A, B, 0, 1\}$ and $\mathbf{X} = \{X_1, \ldots, X_n\}$. For every $1 \leq i \leq n$, define a pair of an ordered tree and an ordered pattern by $T_v = V(B_1(0,1), \ldots, B_n(0,1))$ and $P_v = V(B_1(X_1), \ldots, B_n(X_n))$. Then, we know that if P_v into-matches T_v then the value of $(X_1, \ldots, X_n)$ corresponds with an assignment in $\{0,1\}^n$. For every $1 \leq j \leq m$, let $C_j = (L_{i_1} \wedge L_{i_2} \wedge L_{i_3})$ be the j-th clause and, for every $k = 1, 2, 3$ define 1_{i_k} and $0_{i_k} \in \{0,1\}$ be the assignments of 0 or 1 to variable X_{i_k} satisfying and falsifying L_{i_k}, resp. Then, we define $T_{cj} = C_j(A(1_{i_1}, 0_{i_2}, 0_{i_3}), A(0_{i_1}, 1_{i_2}, 0_{i_3}), A(0_{i_1}, 0_{i_2}, 1_{i_3}))$ and $P_{cj} = C_j(A(X_{i_1}, X_{i_2}, X_{i_3}))$. Let (T, P) be the instance of the into-matching problem as the pair of the tree $T = F(T_v, T_{c1}, \ldots, T_{cm})$ and the pattern $P = F(P_v, P_{c1}, \ldots, P_{cm})$. It is not hard to see that $(b_1, \ldots, b_n) \in \{0,1\}^n$ is an yes-instance of F in one-in-three SAT if and only if P into-matches T by a tree-substitution $\theta = \{ (X_i := b_i) \mid i = 1, \ldots, n \}$. This completes the proof. ■

2.3 Learning model As a learning model, we use the exact learning model of Angluin [6], where a learning algorithm accesses the information on the target concept t_* by using the following queries. Let t_* be a target hypothesis. An *equivalence query* for t_* (EQ) is to propose any tree pattern $t \in OGT$ and denoted by $EQ(t)$. The answer is *yes* if $L(t) = L(t_*)$. Otherwise, a *counterexample* $w \in (L(t_*) - L(t)) \cup (L(t) - L(t_*))$ is returned. A counterexample w is *positive* if $w \in L(t_*)$ and *negative* otherwise. A *subset query* (SQ), denoted by $SQ(t)$, is to propose any tree pattern $t \in OFT$, and receives as the answer *yes* if $L(t) \subseteq L(t_*)$ and *no* otherwise. A *membership query* (MQ), denoted by $MQ(w)$, is to propose any constant tree $w \in \mathcal{T}$, and receives as the answer *yes* if $w \in L(t_*)$ and *no* otherwise. An *entailment membership query* (EntMQ), denoted by $EntMQ(t)$, is to propose any tree pattern $t \in OGT$, and receives as the answer *yes* if $t \sqsubseteq t_*$ and *no* otherwise.

The goal of an *exact learning algorithm* $\mathbf{A}$ is exact identification of the target hypothesis t_* using making equivalence and membership queries for t_*. $\mathbf{A}$ must halt and output a hypothesis $t \in \mathcal{H}$ that is equivalent to t_*, i.e., $L(t) = L(t_*)$, and, at any stage in learning, the running time of $\mathbf{A}$ must be bounded by a polynomial in the size of t_* and of the longest counterexample returned by equivalence queries so far. Unfortunately, MQ is NP-complete for our class OGT with the into-match semantics as seen in the previous section, while EQ is easily decidable.

Algorithm LEARN-INTO-OGT
Given: Oracles for equivalence queries EQ and subset queries SQ with target t_*.
Output: An ordered gapped tree pattern t equivalent to t_*.
if $(EQ(\bot) = yes)$ then return $\bot$;
else let t be a counterexample returned by EQ;
repeat
  $t := Prune(t)$;
  $t := Shrink(t)$;
  $t := Partition(t)$;
until t does not change in the loop;
return t;

Figure 2: A learning algorithm for OGT with the into-match semantics using EQ and SQ

3 Learning Ordered Gapped Tree Patterns

In this section, we present an efficient algorithm for learning ordered gapped tree patterns and using equivalence and subset queries. Then in the next section, we extend it to ordered gapped forests.

3.1 Outline of the algorithm In Fig. 2, we show a learning algorithm LEARN-INTO-OGT for the class OGT with the into-semantics using EQ and SQ. Let Σ be an infinite alphabet and $t_* \in OGT$ be a target ordered tree pattern. We assume a special tree $\bot$, the *bottom*, such that $t \sqsupset \bot$ for every $t \in OGT$. For every $n \geq 0$, we denote by t_n the hypothesis produced in the n-th stage (the n-the execution of the repeat loop).

The algorithm LEARN-INTO-OGT first receives a positive instance t by making the equivalence query $EQ(\bot)$. Then, the algorithm successively generalizes this example $t_0 = t$, called *instance tree*, of the target by applying a set of generalization operations: pruning a constant node, changing a constant node to a variable node or a gap node, contracting an edge, simultaneously changing the subtrees, and splitting a set of identical variables into distinct variables, which are originally developed by Amoth *et al.* [3] for unordered patterns and by Matsumoto and Shinohara [14] for repetition-free pattern languages.

These operations are designed to produce a pattern properly general than the original. Thus, we use subset queries to test if the resulting pattern is still an instance of t_*. Hence, the algorithm produces strictly increasing sequence $t_0 = t \sqsubset t_1 \sqsubset \cdots t_{n-1} \sqsubset t_n \sqsubset \cdots$ $(n \geq 0)$ of the hypotheses limiting to the target tree pattern t_*.

3.2 Generalization of ordered tree patterns In the algorithm LEARN-INTO-OGT, all subprocedures Prune, Shrink, and Partition apply a generalization operation to the current hypothesis t_n and update the current hypothesis with the modified tree only if it is still an instance of the target t_*. Thus, it is ensured that $t_n \sqsubseteq t_*$ for ever $n \geq 0$.

Suppose that $t \sqsubseteq t_*$ holds with a matching ϕ from t_* to t. (1) A node v in t is *constrained by constant* w.r.t ϕ if a constant node u maps to v. (2) A node v in t is *constrained by variable* w.r.t ϕ if a variable-node u maps to an ancestor of v. (3) A node v in t is *constrained by gap* w.r.t ϕ if a gap-node u maps to a (possibly non-proper) descendant of v and the parent p of v, if exists, maps to a proper ancestor of v. That is, v is on the path in t from the children of $\phi(p)$ to $\phi(v)$. (4) A node v is *constrained w.r.t.* ϕ if it is constrained by either a constant, a variable, or a gap. A node v is *free* if it is not

constrained.

The next lemma say that if the matching is strict, i.e., $t_n \sqsubset t_*$, then there is an evidence or an excess in t that can be eliminated or changed by a learning algorithm to generalize t. We say that a node v in t *can be generalized* by some operation on t if the resulting tree t' satisfies $t \sqsubset t' \sqsubseteq t_*$.

Lemma 4 *If $t_* \sqsupset t$ then for a matching ϕ from t_* to t, one of (1) – (5) below holds:*

(1) There is a free node in t w.r.t. ϕ. In this case, any free node v in t w.r.t. ϕ can be eliminated to generalize t.

(2) There is a constant node v in t such that there is a solitary variable node in t_* that maps v w.r.t. ϕ. In this case, all but root nodes in the subtree t/v can be eliminated to generalize t. The root node can be changed to a new variable $x \notin t$.

(3) There are constant nodes with the identical subtrees in t such that there are the occurrences of a repeated variable in t that map into these constants w.r.t. ϕ. That is, the set of these constants includes $\phi(O_{t*}(x))$ for some repeated variable $x \in var(t_*)$. A subset (possibly containing all) of these subtrees can be simultaneously changed to the identical copies of a new variable to generalize t.

(4) There are variable nodes labeled with the identical variables in t such that there are the occurrences of more than one repeated variables $x_1, \ldots, x_k$ $(k \geq 2)$ in t that map into these variables w.r.t. ϕ. That is, the set of these variables includes $\phi(\cup_i O_{t*}(x_i))$. A proper *subset of these variables can be simultaneously changed to the identical copies of a new variable to generalize t.*

(5) There is a node v in t constrained by a gap node u in t_*. If the gap node maps to the node v then the node v must be a constant node. In this case, any gap-constrained node v other than the image $\phi(u)$ of the gap node itself can be eliminated to generalize t, and the node $\phi(u)$ can be generalized by changing to a new gap-variable.

Proof: Let ϕ be any matching from t_* to t. Suppose to contradict that none of (1) – (4) above does not hold for ϕ. From the condition (1) – (3), we know that ϕ is a matching onto V_t, and thus V_t and V_{t*} have the same number of nodes and are isomorphic. Thus, the inverse ϕ^{-1} is also total mapping on V_{t*}. Again by condition (2), any solitary node in t_* maps to a variable node in t. By condition (3), there is no repeated variable in t_* that maps to non-variable nodes in t. From condition (4), distinct variables in t_* map to distinct variables in t. Combining these observations, the inverse ϕ^{-1} is a matching from t to t_*, and thus, we have $t \equiv t_*$. Since this contradicts the assumption, we have one of (1) – (4) above holds for ϕ. The proofs on the correctness of the generalization operations are straightforward. ■

3.3 The pruning algorithm The procedure Prune in Fig. 3 handles case (1) and case (2) of Lemma 4 by locally modifying a leaf of t. This pruning algorithm is a descendant of the procedure Prune in Frazier and Pitt [11] to learn description logic and extensively used by Amoth *et al.* [3]. By property (4) of Lemma 2, we can replace the test for the matching relation $t \sqsubseteq t_*$ with the test for the containment $L(t) \subseteq L(t_*)$ using SQ due to the assumption of an infinite alphabet Σ.

Lemma 5 *If an instance t of target t_* satisfies either case (1) or case (2) of Lemma 4, then the algorithm Prune(t) computes a tree pattern s such that $t \sqsubset s \sqsubseteq t_*$ in $O(n)$ time using $O(n)$ SQ, where n is the size of the initial counterexample given by EQ.*

Procedure Prune(t)
  repeat
    for each leaf v of t with any label do
      if $(SQ(delete(t, v)) = yes)$ then
        $t := delete(t, v)$;
        continue the loop;
    for each leaf v of t with constant label do
      let $x \notin var(t)$ be a new variable;
      if $(SQ(change(t, v, x)) = yes)$ then
        $t := change(t, v, x)$;
        continue the loop;

Procedure Shrink(t)
  repeat
    for each chain node v of t with any label do
      if $(SQ(contr(t, v)) = yes)$ then
        $t := contr(t, v)$;
        continue the loop;
    for each chain node v of t with constant label do
      let $* \notin gap(t)$ be a new gap variable;
      if $(SQ(change(t, v, *)) = yes)$ then
        $t := change(t, v, *)$;
        continue the loop;

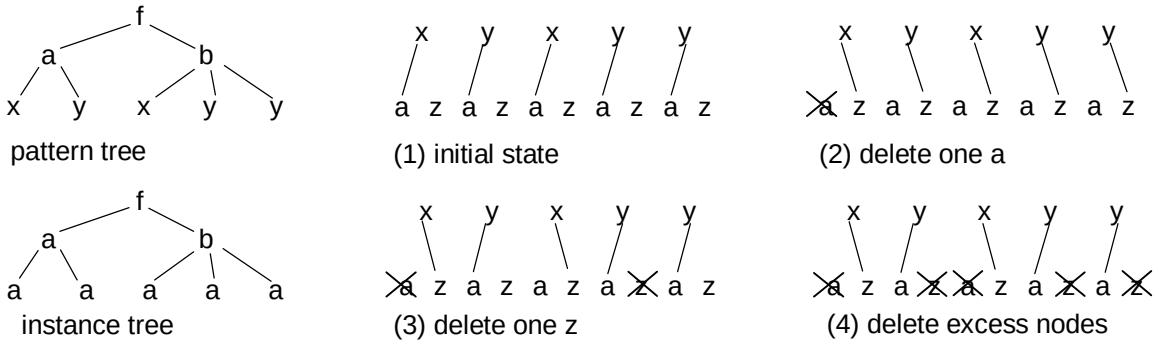

Figure 5: An example computation of the partition algorithm

3.4 The shrinking algorithm The shrinking algorithm in Fig. 4 handles case (5) of Lemma 4 by locally modifying a label of an internal node of t. This shrinking algorithm is introduced by Matsumoto and Shinohara [14] to learn the class of regular (or repetition-free) pattern languages.

Suppose that there is a sequence of nodes in t that are all constrained by a gap node in t_* w.r.t. some matching ϕ. Then, these nodes form a path in t where the gap node maps to the bottom of the path. Unfortunately, some nodes on this path may have side branch and thus cannot be eliminated by the shrinking algorithm. However, we can remove these side branches by applying the pruning algorithm first since any gap node is a chain node. Hence, applying case (5) of Lemma 4, we know that we can eliminate all but bottom nodes of this path. For bottom node, if it is instantiated to a constant then we generalize this node by changing to a new gap variable.

Lemma 6 *If an instance t of target t_* satisfies case (5) of Lemma 4, then the algorithm Shrink(t) computes a tree pattern s such that $t \sqsubset s \sqsubseteq t_*$ in $O(n)$ time using $O(n)$ SQ, where n is the size of the initial counterexample given by EQ.*

3.5 The partitioning algorithm Handling case (3) and case (4) is more difficult than handling case (1) and case (2) because we have to simultaneously generalize a set of subtrees or variables generated by repeated variables. It is not sufficient to locally modify one by one.

To overcome this problem, Amoth, Cull, and Tadepalli [3] developed an elegant par-

Procedure Partition(t)
/* Simultaneously changing a copies of the same subtrees same new variables. */
for each distinct subtree s of t do begin
$S_s := O_t(s)$ and $k := |S_s|$; let $x \notin var(t)$ be a new variable;
for each $v \in S_s$ do
create a new node u labeled with a copy of x;
attach u to the parent of v as the adjacent right sibling of v;
$S_x := O_t(x)$ and $S := S_s \cup S_x$;
while ($|S| > k$) do begin
if there is some $v \in S \cap S_s$ such that $SQ(delete(t, v) = yes)$ then
$t := delete(t, v)$; $S := S \backslash \{v\}$;
else if there is some $v \in S \cap S_x$ such that $SQ(delete(t, v) = yes)$ then
$t := delete(t, v)$; $S := S \backslash \{v\}$;
if ($\emptyset \subset (S \cap S_x) \subset S$) then break the for-loop; (*1)
if ($S = S_x$ and s is not a variable) then break the for-loop; (*2)
end for;
return t;

Figure 6: The partitioning algorithm

tition technique to generalize repeated variables in unordered trees and forests. In Fig. 6, we show the subprocedure *Partition* based on the partition algorithm of Amoth *et al.*. A small difference between our procedure and theirs is that we cannot do the greedy search in the elimination steps due to the order restriction of OGT.

Let us explain the algorithm *Partition* with an example in Fig. 5. In the left of the figure, we have a pattern tree $t_* = f(a(x, y), b(x, y, y))$ with variables x and y and an instance tree $t = f(a(a, a), b(a, a, a))$. Suppose that the pruning procedure is already applied to the instance tree so that $|t| = |t_*|$. Let ϕ_0 be the original matching that maps all five variables in t_* to five constants a.

First, the partition algorithm inserts the copies of new variable, say z, at the right to the occurrences of a constant a. At this point, the original matching ϕ_0 correctly maps (1). Next, the algorithm first tries to delete one of the old leaves labeled with a (2), and then tries to delete one of the new leaves labeled with z (3). If the set of the variables contains at least two distinct variables, both of the above deletion steps succeed. At this point of (3), we have four a's and four z's, while we have five variables (two x's and three y's). Therefore, the maching in (3) splits the variables into two groups. Finally, the algorithm eliminates all the excess leaves to which no variable nodes map by deleting them one by one. This can be executed until exactly five nodes survive (4).

In this way, Partition either changes constants to variables (case (*1)) or splits the copies of the same variables into two groups (case (*2)). These conditions are checked at line (*1) and (*2) of Fig. 6. If none of (*1) and (*2) is succeeded then the algorithm continues the search. Hence, Partition strictly generalizes the hypothesis tree t if it meets conditions (3) or (4).

Lemma 7 *If an instance t of target t_* satisfies either case (3) or case (4) of Lemma 4, then the algorithm Partition(t) computes a tree pattern s such that $t \sqsubset s \sqsubseteq t_*$ in $O(n^2)$ time using $O(n)$ SQ, where n is the size of the initial counterexample given by EQ.*

3.6 Analysis In Fig. 2, let $t_0 \in T$ be the initial positive instance returned by EQ, and for every $n \geq 0$, let t_n be the hypotheses generated in the n-th execution of the repeat

loop.

Lemma 8 *For every $n \geq 0$, both of $t_{n-1} \sqsubset t_n$ and $t_n \sqsubseteq t_*$ hold.*

Proof: The lemma immediately follows from Lemma 5, Lemma 6 and Lemma 7. ■

We introduce a measure *size* of the complexity of a tree pattern. For a tree pattern t, we define the size complexity by $size(t) = 2 \times |t| - (\#var(t) + \#gap(t))$. [2] Obviously, $0 \leq size(t) \leq |t|$ holds.

Lemma 9 *For any OGT s, t, if $s \sqsubset t$ then $size(s) > size(t)$ holds.*

Theorem 10 *Let Σ be an infinite alphabet. The algorithm LEARN-INTO-OGT exactly learns any ordered gapped tree pattern in OGT with the into-match semantics in time $O(n^3)$ using $O(1)$ EQ and $O(n^3)$ SQ, where n is the size of the initial counterexample given by EQ.*

Proof: By construction, the algorithm LEARN-INTO-OGT correctly identifies the target tree pattern t_* when it terminates. Thus, it suffices to show the termination of the algorithm. Combining Lemma 8 and Lemma 9, we know that , $size(t_{n-1}) > size(t_n)$ holds for every $n \geq 0$. This implies that the repeat loop of the algorithm can be executed at most $O(n)$ times. Thus, the running time and the number of queries made can be derived from Lemma 5, Lemma 6 and Lemma 7.

Corollary 11 *The class OGT is polynomial time learnable in the following models.*

- *Exact learning using EQ and MQ when Σ with infinite alphabet.*
- *Exact learning using EQ and EntMQ with finite alphabet.*
- *Exact learning using one positive example and MQ with infinite alphabet.*

The proof is by substitution of new constants and dovetailing. It is almost same as the corresponding proofs in Amoth, Cull, and Tadepalli [3] and omitted here.

4 Extension for ordered gapped forests

An *ordered gapped forest* (OGF, for short) is a finite set $H = \{t_1, \ldots, t_k\}$ $(k \geq 0)$ of ordered gapped tree patterns in OGT. Forests of ordered tree patterns with the onto-semantics are called unions of tree patterns in [8]. For an ordered gapped forest H, we define the *language* of H with the into-semantics as the union $L(H) = \cup_{t \in H} L(t)$ of the languages defined by its members.

Lemma 12 *Let Σ be an infinite alphabet. For any ordered gapped forests $P, Q \in OGF$, $L(P) \supseteq L(Q)$ if and only if for every $q \in Q$ there exists some $p \in P$ such that $p \sqsupseteq q$.*

Proof: We give a sketch of the proof. Suppose we are given P and let $\Sigma(P)$ be the set of all labels appearing in P. We construct an instance s of some tree gapped pattern $q \in Q$ by *substituting* mutually distinct constants from $\Sigma \backslash \Sigma(P)$ for the tree variables and gap variables in q. Clearly $s \in L(P)$. Since every substituted symbols do not appear in P, if there is any matching ϕ from some pattern $p \in P$ to s then every constant node maps to the nodes with original labels in q. Hence, the result follows. ■

[2] For OT with the into-match semantics, the factor 2 of the tree size in this definition is necessary to handle the case where a node with a solitary variable is deleted; For OT with the onto-match semantics, this case cannot occur, and thus the factor can be 1 as in Plotkin [17].

Algorithm LEARN-INTO-OGF
Given: Oracles for EQ and SQ for the target forest T_*.
Output: An ordered gapped forest t equivalent to t_*.
$H := \emptyset$;
while $(EQ(H) = no)$ do
  let t be a counterexample returned by EQ;
  run the algorithm LEARN-INTO-OGT of Fig. 2
    with t as the initial positive counterexample and with SQ for T_*.
  let s be the tree pattern returned by LEARN-INTO-OGT.
  $H := H \cup \{s\}$;
end while;
return H;

Figure 7: A learning algorithm for OGF with the into-match semantics using EQ and SQ

Theorem 13 *Let Σ be an infinite alphabet. The algorithm LEARN-INTO-OGF of Fig. 7 exactly learns any ordered gapped forests T_* in OGF with the into-match semantics in time $O(mn^3)$ using $O(m)$ EQ and $O(mn^3)$ SQ, where m is the cardinality of T_* and n is the size of the initial counterexample given by EQ.*

Proof: In the initial stage, the algorithm received a positive instance of t_* since $L(\emptyset) = \emptyset$. By induction on the number of stages, we can show that the example given to the subroutine LEARN-INTO-OGT is always positive. Let t_0 and t be the initial example to and hypothesis maintained by the subroutine. We will show that $h \not\sqsupseteq t$ for any $h \in H$ and thus the subroutine can correctly simulate the subset query for $L(T_*)\backslash L(H)$ using SQ for $L(T_*)$, where H is the current forest in the main algorithm. Suppose contrary that $h \sqsupseteq t$ for some $h \in H$. Since $t \sqsupseteq t_0$ by Lemma 8, this means $h \sqsupseteq t_0$ and thus $t_0 \in L(H)$. This contradicts the assumption, and we showed the claim. For any counterexample t_0 to $SQ(H)$, $t_0 \in L(T_*)\backslash L(H)$, and thus there is some $t_* \in T_*$ such that $t_* \sqsupseteq t_0$ with some matching ϕ. Based on the existence of ϕ, the subroutine eventually identifies one of the t_* such that $t_* \sqsupseteq t_0$. Since the main algorithm identify at least one member of the target forest T_* in each execution of its while loop, the while loop of the main algorithm can be executed at most m times. This completes the proof. ■

5 Conclusion

In this paper, we have presented efficient learning algorithms for the class OGT of ordered gapped tree patterns and the class OGF of ordered forests, sets of trees, with the same semantics using EQ and SQ. The results of this paper depend on the assumption of infinite alphabet. Hence, it is a future problem to investigate the learnability of the class of bounded forests, sets of at most k OGTs, with a finite alphabet. Connection to the learnability of pattern languages [5, 15] or first-order logic [12] is another future problem.

References

[1] S. Abiteboul, Quass, McHugh, J. Widom, J. L. Wiener, The Lorel query language for semistructured data, *International Journal on Digital Libraries*, 1(1), 68-88, 1997.

[2] A. V. Aho, J. E. Hopcroft and U. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

[3] T. R. Amoth, P. Cull, and P. Tadepalli, Exact learning of unordered tree patterns from queries, In *Proc. COLT'99*, ACM Press, 323–332, 1999.
[4] T. R. Amoth, P. Cull, and P. Tadepalli, Exact learning of tree patterns from queries and counterexamples, In *Proc. COLT'98*, ACM Press, 175-186, 1988.
[5] D. Angluin, Finding patterns common to a set of strings, *JCSS*, 21, 46–62, 1980.
[6] D. Angluin, Queries and concept learning, *Machine Learning*, 2(4), 319–342, 1988.
[7] H. Arimura, H. Ishizaka, T. Shinohara, S. Otsuki, A generalization of the least general generalization, *Machine Intelligence*, 13, 59–85, 1994.
[8] H. Arimura, H. Ishizaka, T. Shinohara, Learning unions of tree patterns using queries, *Theoretical Computer Science*, 185(1), 47–62, 1997.
[9] H. Arimura, T. Shinohara, S. Otsuki, Finding minimal generalizations for unions of pattern languages and its application to inductive inference from positive data, In *Proc. STACS'94*, LNCS 775, Springer-Verlag, 649–660, 1994.
[10] P. Buneman, M. F. Fernandez, D. Suciu, UnQL: A query language and algebra for semistructured data based on structural recursion, *VLDB Journal*, 9(1), 76–110, 2000.
[11] M. Frazier, L. Pitt, CLASSIC learning, *Machine Learning*, 25 (2-3), 151–193, 1996.
[12] R. Khardon, Learning function-free Horn expressions, *Mach. Learn.*, 35(1), 241–275, 1999.
[13] K-I. Ko, A. Marron, Tzeng, Learning string patterns and tree patterns from examples, In *Proc. 7th Internat. Conference on Machine Learning*, 384–391, 1990.
[14] S. Matsumoto and A. Shinohara, Learning Pattern Languages Using Queries, *Proc. Euro COLT'97*, LNAI, Springer-Verlag, 185–197, 1997.
[15] J. Nessel and S. Lange, Learning erasing pattern languages with queries, Proc. ALT2000, LNAI 1968, Springer-Verlag, 86–100, 2000.
[16] C. D. Page and A. M. Frisch, Generalization and learnability: a study of constrained atoms, In *Inductive Logic Programming*, Academic Press (1992) 29 – 61.
[17] G. D. Plotkin, A note on inductive generalization, In *Machine Intell.*, 5, Edinburgh Univ. Press, 153–163, 1970.
[18] H. Sakamoto, Y. Murakami, H. Arimura, S. Arikawa, Extracting Partial Structures from HTML Documents, In *Proc. FLAIRS 2001*, AAAI Press, Key West, May 2001. (To appear)
[19] Extensible Markup Language (XML) Version 1.0. *W3C Recommendation 1998.*
[20] XML-QL: A Query Language for XML W3C Note, Aug. 1998.