Available online at www.sciencedirect.com
SCIENCE DIRECT®
Pattern Recognition 38 (2005) 1372–1383
PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/patcog

Learning context-free grammars using tabular representations☆

Yasubumi Sakakibara*

Department of Biosciences and Informatics, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan

Received 17 March 2004; accepted 17 March 2004

Abstract

We present a novel algorithm using new hypothesis representations for learning context-free grammars from a finite set of positive and negative examples. We propose an efficient hypothesis representation method which consists of a table-like data structure similar to the parse table used in efficient parsing algorithms for context-free grammars such as Cocke–Younger–Kasami algorithm. By employing this representation method, the problem of learning context-free grammars from examples can be reduced to the problem of partitioning the set of nonterminals. We use genetic algorithms for solving this partitioning problem. Further, we incorporate partially structured examples to improve the efficiency of our learning algorithm, where a structured example is represented by a string with some parentheses inserted to indicate the shape of the derivation tree of the unknown grammar. We demonstrate some experimental results using these algorithms and theoretically analyse the completeness of the search space using the tabular method for context-free grammars.

Keywords: Grammatical inference; Context-free grammar; Genetic algorithm; Structured example; Dynamic programming

1. Introduction

Inductive learning of formal languages, often called *grammatical inference*, is an active research area in machine learning and computational learning theory [1]. In particular, the problems of learning finite automata from positive and negative examples have been widely and well studied. Inductive learning of formal languages inherently contains computationally hard problems. For example, the problem of finding a deterministic finite automaton with a minimum number of states consistent with a given finite set of positive and negative examples is known as NP-hard [2]. Nevertheless, one advantage in learning finite automata is that once we construct a so-called prefix-tree automaton from the positive examples (that is, a specific deterministic finite automaton which exactly accepts the positive examples), the learning problem can be reduced to the problem of merging states in the prefix-tree automaton to be consistent with the given examples [3,4]. Dupont [5] applied a genetic algorithm to solve the partitioning problem for the set of states in the prefix-tree automaton. Angluin [6] made use of membership queries to efficiently calculate the consistent and minimum partition for the set of states with respect to the unknown finite automaton.

In this paper, we study the problem of inductively learning context-free grammars from positive and negative examples. This is a more difficult learning problem than learning finite automata. The difficulty arises from two specific aspects of the problem of learning context-free grammars from examples: determining the grammatical structure (topology) of the unknown grammar, and identifying the set of nonterminals in the grammar. The first problem is especially hard because the number of all possible grammatical structures for a given positive example becomes exponential in

☆ Preliminary versions of the paper were presented at 16th International Conference on Machine Learning [15] and 5th International Colloquium on Grammatical Inference [16].

* Tel./fax: +81 45 566 1791.
E-mail address: yasu@bio.keio.ac.jp.

0031-3203/$30.00
doi:10.1016/j.patcog.2004.03.021

the length of the positive example. Thus, the hypothesis space of context-free grammars is very large—too large in which to search for a correct context-free grammar consistent with the given examples. Sakakibara [7] has shown that if information on the grammatical structure of the unknown context-free grammar is available for the learning algorithm, there exists an efficient algorithm for learning context-free grammars from only positive examples.

To overcome the hardness of learning context-free grammars from examples without structural information available, we propose a new hypothesis representation method, called *tabular representation*, which consists of a table-like data structure similar to the parse table used in the Cocke–Younger–Kasami parsing algorithm (CYK algorithm, [8]) for context-free grammars of Chomsky normal form. The tabular representations efficiently represent a hypothesis space of context-free grammars. More precisely, the tabular representations use the dynamic programming technique to efficiently store the exponential number of all possible grammatical structures in a table of polynomial size. We also consider that the concept of tabular representations for context-free grammars corresponds to the prefix-tree automata. By employing this representation method, the problem of learning context-free grammars from examples can be reduced to the problem of partitioning the set of nonterminals. We use genetic algorithms for solving this partitioning problem, because the partitioning problems contains NP-hard problems such as finding minimum-state finite automata consistent with given examples. We demonstrate some experimental results using this algorithm. We also theoretically analyse the completeness of the search space using the tabular method for context-free grammars.

Furthermore, we incorporate partially structured examples to improve the efficiency of our learning algorithm. While our new hypothesis representation method efficiently represents an exponential number of possible grammatical structures in a table of polynomial size, the learning algorithm that uses genetic algorithms still takes a large amount of time. On the other hand, it may be reasonable to assume the availability of some partially structured examples while it is impractical and difficult to assume completely structured examples as input. A completely structured string is a string with parentheses inserted to indicate the shape of the derivation tree of the unknown grammar, or equivalently an unlabeled derivation tree of the grammar. A partially structured string is defined to be a completely structured string with lack of some pairs of left and right parentheses. We show that the partially structured examples contribute to significantly improving the efficiency of the learning algorithm. We demonstrate that our learning algorithm from partially structured examples can identify a grammar with the intended structure, that is, structurally equivalent to the unknown grammar, and is more flexible and applicable than Sakakibara's learning algorithm [7] from completely structured examples.

2. Preliminaries

A *context-free grammar* (CFG) is defined by a quadruple $G = (N, \Sigma, P, S)$, where N is an alphabet of *nonterminal symbols*, Σ is an alphabet of *terminal symbols* such that $N \cap \Sigma = \emptyset$, P is a finite set of production rules of the form $A \to \alpha$ for $A \in N$ and $\alpha \in (N \cup \Sigma)^*$, and S is a special nonterminal called the *start symbol*. A *derivation* is a rewriting of a string in $(N \cup \Sigma)^*$ using the production rules of the CFG G. In each step of the derivation, a nonterminal from the current string is chosen and replaced with the right-hand side of a production rule for that nonterminal. This replacement process is repeated until the string consists of terminal symbols only. If a derivation begins with a nonterminal A and derive a string $\alpha \in (N \cup \Sigma)^*$, we denote $A \Rightarrow \alpha$. The *language generated* by a CFG G is denoted $L(G)$, that is, $L(G) = \{w \mid S \Rightarrow w, w \in \Sigma^*\}$. Two CFGs G and G' are said to be *equivalent* if and only if $L(G) = L(G')$. A CFG $G = (N, \Sigma, P, S)$ is in *Chomsky normal form* if each production rule is of the form $A \to BC$ or $A \to a$ where $A, B, C \in N$ and $a \in \Sigma$.

In the following, we fix a terminal alphabet Σ, and without loss of generality, we only consider context-free grammars without any ε-production rules and any useless production rules where an ε-production rule is of the form $A \to \varepsilon$ (where ε means the empty string of length 0) and a production rule $A \to \alpha$ is *useless* if there is no derivation $S \Rightarrow \beta A \gamma \to \beta\alpha\gamma \Rightarrow w$ for any $\beta, \gamma \in (N \cup \Sigma)^*$ and any $w \in \Sigma^*$.

We assume the unknown (target) CFG, denoted G_*, to be learned. A *positive example* of G_* is a string in $L(G_*)$ and a *negative example* of G_* is a string not in $L(G_*)$. A *representative sample* of G_* is defined to be a finite subset of $L(G_*)$ that exercises every production rule in G_*, that is, every production is used at least once to generate the subset.

A string with grammatical structure, called *a structured string* or *a structural description* (*of string*), is a string with some parentheses inserted to indicate the shape of the derivation tree of the unknown grammar G_*, or equivalently an unlabeled derivation tree of G_*, that is, a derivation tree whose internal nodes have no labels.

3. Tabular method for representing hypotheses of CFGs

For learning context-free grammars or context-free languages, we usually use CFGs themselves for representing hypotheses. However, the hypothesis space of CFGs is very large—too large in which to search for a correct CFG consistent with the given examples. This is because the problem of learning CFGs from examples has two specific aspects:

(1) Determining the grammatical structure (topology) of the unknown grammar, and
(2) Identifying the set of nonterminals in the grammar.

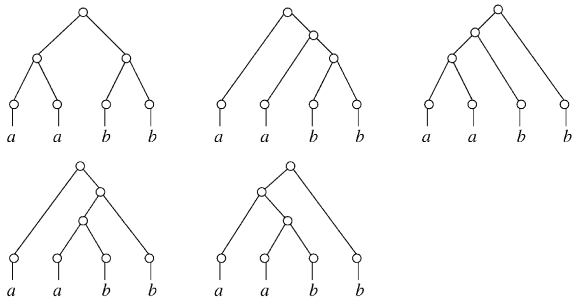

Fig. 1. All possible grammatical structures for the string "*aabb*".

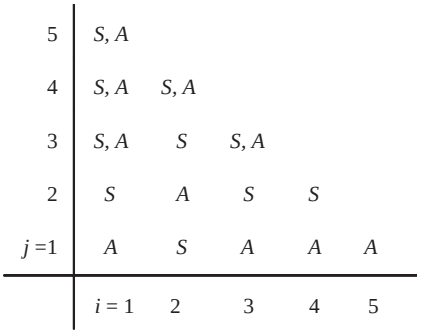

Fig. 2. The parse table T of G for "*abaaa*".

The first problem is especially hard. For example, assume that a given positive example is a string "*aabb*". All possible grammatical structures for this string are shown in Fig. 1 when we consider the Chomsky normal form of CFGs.

The number of all possible grammatical structures for a string of length n becomes "exponential" in n. In order to solve this hardness, we propose a new representation method which consists of a table-like data structure similar to the parse table used in the Cocke–Younger–Kasami parsing algorithm (CYK algorithm, [8]) for CFGs of Chomsky normal form.

First, we explain the tabular parsing method of the CYK algorithm, and next we propose a tabular representation method which efficiently represents an exponential number of possible grammatical structures in a table of polynomial size. The CYK algorithm is a polynomial time algorithm to solve the parsing (membership) problem of CFGs using "dynamic programming". The CYK algorithm assumes Chomsky normal form of CFGs, and the essence of the algorithm is the construction of a triangular *parse table* T. Given a CFG $G=(N, \Sigma, P, S)$ and the input string $w=a_1a_2\cdots a_n$ in Σ^* to be parsed according to G, each element of T, denoted $t_{i,j}$, for $1\leqslant i\leqslant n$ and $1\leqslant j\leqslant n-i+1$, will have a value which is a subset of N. The interpretation of T is that a nonterminal A will be in $t_{i,j}$ if and only if $A \Rightarrow a_ia_{i+1}\cdots a_{i+j-1}$, that is, A derives the substring of w beginning at position i and of length j. To determine whether the string w is in $L(G)$, the algorithm computes the parse table T and look to see whether S is in entry $t_{1,n}$.

In the first step of constructing the parse table, the CYK algorithm sets $t_{i,1}=\{A \mid A \to a_i \text{ is in } P\}$. In the second step, it assumes that $t_{i,j'}$ has been computed for $1\leqslant i\leqslant n$ and $1\leqslant j'<j$, and it computes $t_{i,j}$ by examining the nonterminals in the following pairs of entries:

$$(t_{i,1}, t_{i+1,j-1}), (t_{i,2}, t_{i+2,j-2}), \ldots, (t_{i,j-1}, t_{i+j-1,1})$$

and if B is in $t_{i,k}$ and C is in $t_{i+k,j-k}$ for some k $(1\leqslant k<j)$ and the production $A \to BC$ is in P, adding A to $t_{i,j}$.

For example, we consider a simple CFG $G=(N, \Sigma, P, S)$ of Chomsky normal form where $N=\{S, A\}$, $\Sigma=\{a, b\}$ and

$$P=\{S \to AA, S \to AS, S \to b, A \to SA, A \to a\}.$$

This CFG generates a string "*abaaa*", that is, $S \Rightarrow abaaa$, and the parse table T for *abaaa* is shown in Fig. 2. The parse table can efficiently store all possible parse trees of G for *abaaa*.

Now we propose the tabular representation method employing this parse table to efficiently represent the hypotheses of CFGs. Given a positive example $w=a_1a_2\cdots a_n$ of length n, the *tabular representation* for w is the triangular table $T(w)$ where each element, denoted $t_{i,j}$, for $1\leqslant i\leqslant n$ and $2\leqslant j\leqslant n-i+1$, contains the set $\{X_{i,j,k_1}, \ldots, X_{i,j,k_{j-1}}\}$ of $j-1$ distinct nonterminals. For $j=1$, $t_{i,1}$ is the singleton set $\{X_{i,1,1}\}$ (see Fig. 3). The *primitive* CFG $G(T(w))=(N, \Sigma, P, S)$ derived from the tabular representation $T(w)$ is defined to be as follows, in Chomsky normal form:

$$\begin{aligned}
N &= \{X_{i,j,k} \mid 1\leqslant i\leqslant n, 1\leqslant j\leqslant n-i+1, 1\leqslant k<j\} \\
&\quad \cup \{X_{i,1,1} \mid 1\leqslant i\leqslant n\}, \\
P &= \{X_{i,j,k} \to X_{i,k,l_1}X_{i+k,j-k,l_2} \mid 1\leqslant i\leqslant n, \\
&\quad 1\leqslant j\leqslant n-i+1, 1\leqslant k<j, 1\leqslant l_1<k, 1\leqslant l_2<j-k\} \\
&\quad \cup \{X_{i,1,1} \to a_i \mid 1\leqslant i\leqslant n\} \\
&\quad \cup \{S \to X_{1,n,k} \mid 1\leqslant k\leqslant n-1\}.
\end{aligned}$$

For each nonterminal $X_{i,j,k}$ at entry $t_{i,j}$, $X_{i,j,k}$ derives the substring of w beginning at position i and of length j, that is, $X_{i,j,k} \Rightarrow a_ia_{i+1}\cdots a_{i+j-1}$, and there are the set of production rules of the form $X_{i,j,k} \to X_{i,k,l_1}X_{i+k,j-k,l_2}$ consisting of the nonterminals at entries $t_{i,k}$ and $t_{i+k,j-k}$ in the right-hand side. The number of nonterminals contained in $G(T(w))$ is at most n^3 and the size (the number of production rules) of the primitive CFG $G(T(w))$ is $O(n^5)$, that is, "polynomial" of n.

This primitive CFG $G(T(w))$ only generates the string w but can generate all possible grammatical structures on w. For example, when a positive example "*aabb*" is given, the tabular representation $T(aabb)$ and the primitive CFG $G(T(aabb))$ become as shown in Fig. 4. While the size of $G(T(aabb))$ is polynomial of the length of the given example, $G(T(aabb))$ can generate all possible grammatical structures for *aabb*, as shown in Fig. 5.

n	$\{X_{1,n,1},\cdots,X_{1,n,n-1}\}$			
$n-1$	$\{X_{1,n-1,1},\cdots;X_{1,n-1,n-2}\}$	$\{X_{2,n-1,1},\ldots,X_{2,n-1,n-2}\}$		
$\vdots$	$\vdots$	$\vdots$	$\ddots$	
2	$\{X_{1,2,1}\}$	$\{X_{2,2,1}\}$	$\cdots$	
$j=1$	$\{X_{1,1,1}\}$	$\{X_{2,1,1}\}$	$\cdots$	$\{X_{n,1,1}\}$
	$i=1$	2	$\cdots$	n

Fig. 3. The tabular representation $T(w)$ for w.

4	$X_{1,4,1}, X_{1,4,2}, X_{1,4,3}$			
3	$X_{1,3,1}, X_{1,3,2}$	$X_{2,3,1}, X_{2,3,2}$		
2	$X_{1,2,1}$	$X_{2,2,1}$	$X_{3,2,1}$	
$j=1$	$X_{1,1,1}$	$X_{2,1,1}$	$X_{3,1,1}$	$X_{4,1,1}$
	$i=1$	2	3	4
	a	a	b	b

$$
\begin{aligned}
P = \{\ & S \to X_{1,4,1}, \quad S \to X_{1,4,2}, \quad S \to X_{1,4,3}, \\
& X_{1,4,1} \to X_{1,1,1}X_{2,3,1},\ X_{1,4,1} \to X_{1,1,1}X_{2,3,2},\ X_{1,4,2} \to X_{1,2,1}X_{3,2,1}, \\
& X_{1,4,3} \to X_{1,3,1}X_{4,1,1},\ X_{1,4,3} \to X_{1,3,2}X_{4,1,1},\ X_{1,3,1} \to X_{1,1,1}X_{2,2,1}, \\
& X_{1,3,2} \to X_{1,2,1}X_{3,1,1},\ X_{2,3,1} \to X_{2,1,1}X_{3,2,1},\ X_{2,3,2} \to X_{2,2,1}X_{4,1,1}, \\
& X_{1,2,1} \to X_{1,1,1}X_{2,1,1},\ X_{2,2,1} \to X_{2,1,1}X_{3,1,1},\ X_{3,2,1} \to X_{3,1,1}X_{4,1,1}, \\
& X_{1,1,1} \to a, \quad X_{2,1,1} \to a, \quad X_{3,1,1} \to b, \\
& X_{4,1,1} \to b\ \}
\end{aligned}
$$

Fig. 4. The tabular representation $T(aabb)$ and the derived primitive CFG $G(T(aabb))$.

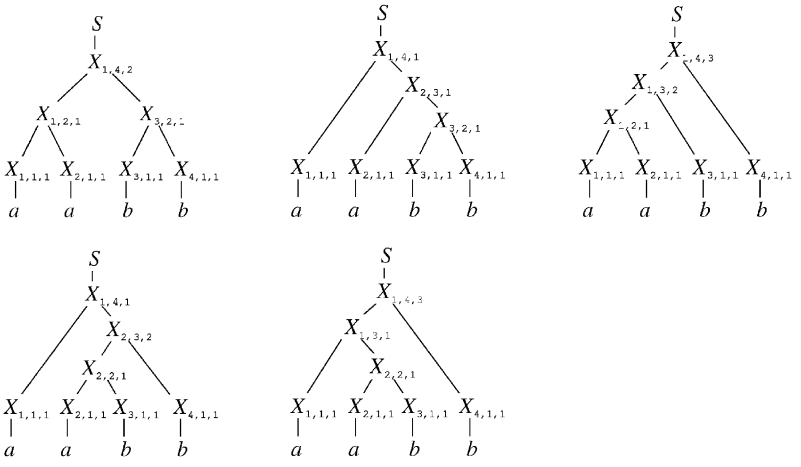

Fig. 5. All possible parse trees of $G(T(aabb))$ for "$aabb$".

Thus, the tabular representations use the dynamic programming technique to efficiently store the exponential number of all possible grammatical structures in a table of polynomial size.

4. Learning algorithm for CFGs using tabular representations

As we have seen in the previous section, the parsing is a process that given a CFG G, constructs a parse table for a string w and sees whether $w \in L(G)$. Learning a CFG is the completely reverse process, which given a sample of positive examples w and negative examples, constructs the tabular representation $T(w)$ for w, merges the nonterminals in $G(T(w))$ to be consistent with the sample and produces a CFG.

4.1. Learning algorithm TBL using tabular representations

A *partition* of some set X is a set of pairwise disjoint nonempty subsets of X whose union is X. If π is a partition of X, then for any element $x \in X$ there is a unique element of π containing x, which we denote $K(x, \pi)$ and call the *block* of π containing x. Let $G = (N, \Sigma, P, S)$ be a CFG and π be a partition of the set N of nonterminals. The CFG $G/\pi = (N', \Sigma, P', S')$ induced by π from G is defined as follows:

$$N' = \pi \text{ (the set of blocks of } \pi),$$

$$P' = \{K(A, \pi) \to K(B, \pi)\ K(C, \pi) \mid A \to BC \in P\} \cup \{K(A, \pi) \to a \mid A \to a \in P\},$$

$$S' = K(S, \pi).$$

Our learning algorithm for CFGs is designed as follows: Given positive examples w, we first construct the tabular representation $T(w)$ and the primitive CFG $G(T(w))$, and second we merge distinct nonterminals in $G(T(w))$ to be consistent with the given positive and negative examples and minimize the number of nonterminals in $G(T(w))$. We use a genetic algorithm to solve the problem of merging distinct nonterminals, that is, partitioning the set of nonterminals $\{X_{i,j,k} \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n - i + 1, 1 \leqslant k < j\} \cup \{X_{i,1,1} \mid 1 \leqslant i \leqslant n\}$. This partitioning problem contains the problem of finding minimum-state finite automata consistent with given examples and hence it is NP-hard. Therefore, it is reasonable to use a genetic algorithm for solving the computationally hard problem of partitioning the set of nonterminals. Fig. 6 illustrates our scenario.

We assume that a finite set U of positive and negative examples which contains a representative sample of the unknown CFG G_* is given. Let U_+ denote the set of positive examples in U and U_- denote the set of negative examples. The learning algorithm TBL for CFGs is shown in Fig. 7. In the learning algorithm TBL, we need to rename the nonterminals of $G(T(w))$ for each $w \in U_+$ before taking the union of CFGs so that every $G(T(w))$ $(w \in U_+)$ has no same nonterminal symbols each other. For a string w, let $G^w(T(w))$ denote the CFG $G(T(w))$ except that every nonterminal X in $G(T(w))$ is renamed to X^w.

4.2. Genetic algorithm to solve the partitioning problem

We employ a genetic algorithm (GA) to solve the partitioning problem for the set of nonterminals in the primitive CFGs. The genetic algorithm is a parallel search technique in which a set, namely the *population*, of potential solutions, called *individuals* is progressively updated through a *selection mechanism* and *genetic operations*, that is, the *crossover* and the *mutation*. Each individual is coded as a fixed length string, called a *chromosome*.

Starting from a randomly generated initial population, each individual is evaluated by the *fitness function* to be optimized. The population of the next generation is obtained in three steps. Firstly, a random selection with repetitions is performed, usually on the whole population. The probability of a given individual to be selected is obtained from the ratio between its fitness value and the average fitness value computed over the whole population. Secondly, at a given *crossover rate* pairs are selected from this temporary population. Each pairs of *parents* give rise to two children which are constructed by taking alternative subparts from their parent chromosomes. After the crossover operation, the parents are replaced by their children in the temporary population. Thirdly, some individuals are selected at a given *mutation rate*. One position in the associated chromosome of each selected individual is randomly chosen and the corresponding value is replaced. The final population of the next generation is made of the individuals obtained after fitness proportionate reproduction, crossover and mutation. This process is iterated until some termination criterion is satisfied.

Genetic algorithms have been well studied for the partitioning problems [9] and we take a successful approach by von Laszewski [10] for partitioning n elements into k categories (blocks). This approach encodes partitions using *group-number encoding*, that is, partitions are represented as strings of integers of length n,

$$(i_1, i_2, \ldots, i_n)$$

where the jth integer $i_j \in \{1, \ldots, k\}$ indicates the block number assigned to element j. Let π_p denote the partition represented by a string p. This representation is supported by specific GA operators called "intelligent structural operators": *structural crossover* and *structural mutation*.

Given a positive example "*abaaa*"

⇓ **construct the tabular representation** $T(abaaa)$

5	$X_{1,5,1}, X_{1,5,2}, X_{1,5,3}, X_{1,5,4}$				
4	$X_{1,4,1}, X_{1,4,2}, X_{1,4,3}$	$X_{2,4,1}, X_{2,4,2}, X_{2,4,3}$			
3	$X_{1,3,1}, X_{1,3,2}$	$X_{2,3,1}, X_{2,3,2}$	$X_{3,3,1}, X_{3,3,2}$		
2	$X_{1,2,1}$	$X_{2,2,1}$	$X_{3,2,1}$	$X_{4,2,1}$	
$j=1$	$X_{1,1,1}$	$X_{2,1,1}$	$X_{3,1,1}$	$X_{4,1,1}$	$X_{5,1,1}$
	$i=1$	2	3	4	5

⇓ **merge nonterminals**

5	S, A				
4	S, A	S, A			
3	S, A	S	S, A		
2	S	A	S	S	
$j=1$	A	S	A	A	A
	$i=1$	2	3	4	5

⇓ **output a CFG**

$$P = \{ S \to AA,\ S \to AS,\ S \to b,\ A \to SA,\ A \to a \}$$

Fig. 6. Illustrating our strategy: upper, constructing the tabular representation $T(abaaa)$ for a given positive example "*abaaa*"; middle, merging nonterminals in the primitive CFG $G(T(abaaa))$; and lower, the resulting CFG.

Learning Algorithm TBL:

(1) Construct the tabular representation $T(w)$ for each positive example w in U_+;
(2) Derive the primitive CFG $G(T(w))$ and $G^w(T(w))$ for each w in U_+;
(3) Take the union of those primitive CFGs, that is,

$$G(T(U_+)) = \bigcup_{w \in U_+} G^w(T(w))$$

(4) Find a smallest partition π_* such that $G(T(U_+))/\pi_*$ is consistent with U, that is, consistent with the positive examples U_+ and the negative examples U_-;
(5) Output the resulting CFG $G(T(U_+))/\pi_*$.

Fig. 7. The learning algorithm TBL for CFGs using tabular representations.

(1) *Structural crossover*: the mechanism is explained by the following example. Assume that there are two selected parents (strings of length 12)

$p_1 = (112341232253)$ and $p_2 = (112423122335)$.

These strings encode the following partitions:

π_{p_1}: {1, 2, 6}, {3, 7, 9, 10}, {4, 8, 12}, {5}, {11},
π_{p_2}: {1, 2, 7}, {3, 5, 8, 9}, {6, 10, 11}, {4}, {12}.

First, a random block is selected, say block #2, from p_1. This block is copied (introduced) from p_1 into p_2, by taking the union of the original corresponding block in p_2 and the block to be moved from p_1:

$p_2' = (112423222235)$,

and the resulting partition is:

$\pi_{p_2'}$: {1, 2}, {3, 5, 7, 8, 9, 10}, {6, 11}, {4}, {12}.

Thus, the structural crossover has an effect to take an union of both parent partitions of a randomly selected block.

(2) *Structural mutation*: it replaces a single component of a string by some random number. That is, a parent

$$p_2' = (112423222235)$$

may produce the following offspring (the number on position 9 is replaced):

$$p_2'' = (112423225235),$$
$$\pi_{p_2''}\colon \{1,2\},\ \{3,5,7,8,10\},\ \{6,11\},\ \{4\},\ \{9,12\}.$$

Thus, the structural mutation consists of a random selection of an element in some block of a given partition and the random assignment of this element to a block. The resulting partition sometimes merges previously different nonterminals and sometimes separates previously merged nonterminals.

In addition to those operators, we introduce a *special mutation operator that deletes a randomly selected element* with small probability. The deletion operator randomly selects a single component (position) of a string and replaces it to the special number (say, -1) indicating that the element at the position is deleted. This deletion operator significantly contributes to deleting unnecessary nonterminals in $G(T(w))$ and reducing the size of the learned CFGs.

We define the fitness function f from individuals to real numbers [0, 1] between 0 and 1. Let p be an individual and π_p be the partition represented by p for the set of nonterminals in $G(T(U_+))$. Let U_+ be the given positive examples and U_- be the given negative examples. The fitness function f is defined as follows:

$$f_1(p) = \frac{|\{w \in U_+ \mid w \in G(T(U_+))/\pi_p\}|}{|U_+|},$$

$$f_2(p) = \frac{1}{|\pi_p|},$$

$$f(p) = \begin{cases} 0 & \text{if some } w \text{ in } U_- \text{ is generated by } G(T(U_+))/\pi_p, \\ \dfrac{C_1 \times f_1(p) + C_2 \times f_2(p)}{C_1 + C_2} & \text{otherwise,} \end{cases}$$

where C_1, C_2 are some constants.

This fitness function f tries to find a CFG consistent with both the given positive and negative examples. In particular, the value of the fitness function f for a CFG which accepts any negative example is set for 0, which implies that the CFG is immediately discarded. Among the CFGs consistent with the positive and negative examples, we follow the "Occam razor" to choose the best ones. That is, we choose a CFG containing the minimal number of nonterminals among the consistent CFGs.

4.3. Theoretical analyses of tabular method based on tree automaton theory

In this section, we theoretically analyze the completeness of the search space using the tabular method for context-free grammars. That is, we show that given an appropriate positive sample U_+ of the unknown context-free grammar G_*, there exists some partition π such that $G(T(U_+))/\pi$ is equivalent to G_*. In order to show the completeness, we first extend a fundamental theorem for the completeness of search space for the inductive inference of finite automata to the one for tree automata, and next apply the theorem to show the completeness of the search space using the tabular method for context-free grammars.

We quote the following theorem [11] that shows the completeness of the search space of the regular inference problem:

Let $M_* = (Q, \Sigma, \delta, q_0, F)$ be the unknown finite automaton. A *structurally complete* sample I_+ with respect to M_* is a set of positive examples such that (i) every transition of M_* is exercised, and (ii) every final state in F is used as accepting state. Let $I_+ = \{w_1, w_2, \ldots, w_m\}$, where $w_i = a_{i,1} \cdots a_{i,|w_i|}$ for $1 \leqslant i \leqslant m$. The *maximal canonical automaton* $\mathrm{MCA}(I_+) = (Q, \Sigma, \delta, q_0, F)$ with respect to I_+ is defined as follows:

$$Q = \{v^{i,j} \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant |w_i|, v = a_{i,1} \cdots a_{i,j}\} \cup \{\varepsilon\},$$
$$\delta(v^{i,j}, a) = \{u^{i,j+1} \mid u = va_{i,j+1} \text{ and } a = a_{i,j+1}\},$$
$$\delta(\varepsilon, a) = \{u^{i,1} \mid u = a \text{ and } a = a_{i,1}\},$$
$$q_0 = \varepsilon,$$
$$F = \{w_i^{i,|w_i|} \mid 1 \leqslant i \leqslant m, w_i \in I_+\}.$$

There exists a partition π of Q such that the finite automaton $\mathrm{MCA}(I_+)/\pi$ induced by π from $\mathrm{MCA}(I_+)$ is isomorphic to M_*.

We introduce the notion of tree automaton which is a finite automaton defined on trees.

A *ranked alphabet* V is a finite set of symbols associated with a *rank*. Intuitively, an associated rank for a symbol f represents an arity of f or the number of children of a node labeled f in a tree. A tree over a ranked alphabet V is a rooted, directed, connected acyclic finite graph in which the direct successors of any node are linearly ordered from left to right. The predecessor of a node is called the *parent*, the successor, a *child*. The size of a tree t, denoted $|t|$, is the number of nodes in t. The nodes in a tree t of size n are numbered from 1 to n according to the preorder where the root node is numbered 1, and each node of t is labeled by a symbol in V. Let λ denote the empty tree.

A *nondeterministic (frontier-to-root) tree automaton* over a ranked alphabet V is a quadruple $M = (Q, V, \delta, F)$ such

that Q is a finite set of states, F is a set of final states, and δ is the state transition function:

$$\delta_k : V_k \times (Q \cup V_0)^k \to 2^Q,$$

where V_k is the set of symbols of rank k in V and V_0 denote the terminal alphabet. Let $L(M)$ denote the set of trees accepted by the tree automaton M.

We can define a *structurally complete* sample I_+ of trees, denoted $I_+ = \{t_1, t_2, \ldots, t_m\}$, with respect to the unknown tree automaton M_* in a same manner as finite automata on strings, and define the related *maximal canonical tree automaton* $\mathrm{MCT}(I_+) = (Q, \Sigma, \delta, F)$ with respect to I_+ as follows:

$$Q = \{v^{i,j} \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant |t_i|, v \text{ is a subtree of root node } j \text{ in } t_i\} \cup \{\lambda\},$$

$$\delta_k((v_1^{i,j_1}, v_2^{i,j_2}, \ldots, v_k^{i,j_k}), f) = \{f(v_1, \ldots, v_k)^{i,j} \mid f(v_1, \ldots, v_k) \text{ is a subtree of root node } j \text{ in } t_i\},$$

$$F = \{t_i^{i,1} \mid 1 \leqslant i \leqslant m, t_i \in I_+\}.$$

We can show the following theorem of the completeness of the search space for the inductive inference of tree automata.

> Let I_+ be a structurally complete sample with respect to the unknown tree automaton M_* and $\mathrm{MCT}(I_+) = (Q, \Sigma, \delta, F)$ be the maximal canonical tree automaton with respect to I_+. There exists a partition π of Q such that the tree automaton $\mathrm{MCT}(I_+)/\pi$ induced by π from $\mathrm{MCT}(I_+)$ is isomorphic to M_*.

In order to apply this result to show the completeness of the search space using the tabular method for CFGs, we first quote the relationship between tree automata and CFGs.

Let σ denote a special symbol used for labeling internal nodes of trees. A *skeleton* is a tree in which all the internal nodes are labeled by σ. The *skeletal description* of a tree t over V, denoted $s(t)$, is a tree t except that all labels of the internal nodes of t are replaced with σ. The skeletal description of a tree describes only the shape and terminal nodes of the tree. Let $D(G)$ denote the set of derivation trees of a CFG G and $s(D(G))$ denote the set of unlabeled derivation trees (structured strings) of G.

It is known that the set of derivation trees of a CFG can be accepted by some tree automaton [12]. Further, the set of unlabeled derivation trees of a CFG can also be accepted by some tree automaton.

Based on these observations, we can show the following argument. Let $M(G_*)$ be a tree automaton which accepts the set $s(D(G_*))$ of unlabeled derivation trees of the unknown CFG G_*. Let I_+ be a structurally complete sample of skeletons with respect to the tree automaton $M(G_*)$ and $\mathrm{MCT}(I_+) = (Q, \Sigma, \delta, F)$ be the maximal canonical tree automaton with respect to I_+. Then, there exists a partition π of Q such that the tree automaton $\mathrm{MCT}(I_+)/\pi$ induced by π from $\mathrm{MCT}(I_+)$ is isomorphic to $M(G_*)$ and hence $L(M(G_*)) = s(D(G_*))$. This theorem implies that given a structurally complete sample of skeletons with respect to the tree automaton $M(G_*)$ for the unknown CFG G_*, we can infer a tree automaton $\mathrm{MCT}(I_+)/\pi$ which accepts $s(D(G_*))$ by finding an appropriate partition π of Q.

Further, we can translate a tree automaton M which accepts a set of skeletons into a CFG $G(M)$ such that $s(D(G)) = L(M)$ by a coding of the recognizing process of a tree automaton in the formalism of a context-free grammar.

> Let $M = (Q, \{\sigma\} \cup \Sigma, \delta, F)$ be a tree automaton over skeletons. The corresponding CFG $G(M) = (N, \Sigma, P, S)$ is defined as follows:
>
> $$N = Q \cup \{S\},$$
>
> $$P = \{p \to q_1 \cdots q_k \mid q_1, \ldots, q_k \in Q \cup \Sigma \text{ and } p \in \delta_k(\sigma, q_1, \ldots, q_k)\} \cup \{S \to q_1 \cdots q_k \mid \delta_k(\sigma, q_1, \ldots, q_k) \cap F \neq \emptyset\}.$$

Combined with these results, we can state the following theorem. Let G_* be the unknown CFG and $M(G_*)$ be a corresponding tree automaton such that $s(D(G_*)) = L(M(G_*))$. Let I_+ be a structurally complete sample of skeletons with respect to the tree automaton $M(G_*)$ and $\mathrm{MCT}(I_+) = (Q, \Sigma, \delta, q_0, F)$ be the maximal canonical tree automaton with respect to I_+. Note that $I_+ \subseteq s(D(G_*))$. Then, there exists a partition π of Q such that the tree automaton $\mathrm{MCT}(I_+)/\pi$ induced by π from $\mathrm{MCT}(I_+)$ is isomorphic to $M(G_*)$ and further the corresponding CFG $G(\mathrm{MCT}(I_+)/\pi)$ is equivalent to the unknown CFG G_*, that is, $L(G(\mathrm{MCT}(I_+)/\pi)) = L(G_*)$. This theorem implies that given a structurally complete sample of skeletons with respect to the tree automaton $M(G_*)$ for the unknown CFG G_*, we can infer a CFG $G(\mathrm{MCT}(I_+)/\pi)$ which is equivalent to the unknown CFG G_* by finding an appropriate partition π of Q.

Now, we show the completeness of the search space using the tabular method for CFGs. We assume the unknown CFG G_* in Chomsky normal form. Let $M(G_*)$ be a corresponding tree automaton such that $s(D(G_*)) = L(M(G_*))$ and I_+ be a structurally complete sample of skeletons with respect to the tree automaton $M(G_*)$. For a skeleton u, let $l(u)$ denote the terminal string on the leaf (terminal) nodes of u. Note that $l(u) \in L(G_*)$ for $u \in s(D(G_*))$. We can observe that for a skeleton u in I_+, the tabular representation $T(l(u))$ contains the skeleton u and the primitive CFG $G(T(l(u)))$ generates a derivation tree t for $l(u)$ which has the same structure as u, that is, $u \in s(D(G(T(l(u)))))$. Further, all nonterminals attached to the nodes in the derivation tree t by $G(T(l(u)))$ are different. On the other hand, the primitive CFG $G(T(l(u)))$ generates other kinds of

derivation trees for $l(u)$ which have different structures from u. All nonterminals and production rules in $G(T(l(u)))$ to be used to generate such different structures are useless and not contained in the original grammar G_*. Hence we freeze those useless nonterminals to avoid being merged in a partitioning process. We denote all nonterminals in $G(T(l(u)))$ used to derive a skeleton u by live$(u, G(T(l(u))))$. Then, we can show the following theorem for the completeness of the search space using the tabular method:

> Let G_* be the unknown CFG in Chomsky normal form. Let I_+ be a structurally complete sample of skeletons with respect to the tree automaton $M(G_*)$ for the unknown CFG G_*, and let $U_+ = \{l(u) \mid u \in I_+\}$ be given as a positive sample of the unknown CFG G_*. Let $G(T(U_+)) = (N, \Sigma, P, S)$ be the derived primitive CFG from U_+. Then, there exists a partition π of $\bigcup_{u \in I_+} \text{live}(u, G(T(l(U_+))))$ such that the CFG $G(T(U_+))/\pi$ induced by π from $G(T(U_+))$ is equivalent to the unknown CFG G_*, that is, $L(G(T(U_+))/\pi) = L(G_*)$.

This result indicates that we need to investigate some adequate fitness function for genetic algorithm to find a partition π such that $L(G(T(U_+))/\pi) = L(G_*)$ and the fitness function may not be the Occam razor but one for freezing useless nonterminals to avoid being merged. We need a further analysis for adequate fitness functions of GAs.

5. First experimental results

We had two preliminary experiments for learning two context-free languages using our learning algorithm TBL.

5.1. Problem 1

The unknown (target) context-free language is $\{a^m b^m c^n \mid m, n \geqslant 1\}$ over $\Sigma = \{a, b, c\}$. This context-free language is especially important to check the performance of the learning algorithm because it contains all typical grammatical structures specific to the CFG: branch structure ($A \to BC$), parenthesis (paired) structure ($A \to aBb$), and iteration structure ($A \to aA$).

We gave all positive examples of length up to 10 and all negative examples of length up to 20. The learning algorithm TBL outputs the following correct CFG $G = (N, \Sigma, P, S)$ as the best individual in the population of size 100 after 1200 iterations:

$$N = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 112\rangle, \langle 116\rangle, \langle 140\rangle\},$$

$$\begin{aligned} P = \{&S \to \langle 140\rangle,\ \langle 140\rangle \to \langle 140\rangle\langle 3\rangle,\ \langle 140\rangle \to \langle 116\rangle\langle 3\rangle,\\ &\langle 116\rangle \to \langle 112\rangle\langle 2\rangle,\quad \langle 116\rangle \to \langle 1\rangle\langle 2\rangle,\\ &\langle 112\rangle \to \langle 1\rangle\langle 116\rangle,\quad \langle 1\rangle \to a,\ \langle 2\rangle \to b,\quad \langle 3\rangle \to c\}. \end{aligned}$$

5.2. Problem 2

The unknown context-free language is $\{ac^m \mid m \geqslant 1\} \cup \{bc^m \mid m \geqslant 1\}$ over $\Sigma = \{a, b, c\}$. This is the union of two languages.

We gave all positive examples of length up to 6 and all negative examples of length up to 12. The learning algorithm TBL outputs the following correct CFG $G = (N, \Sigma, P, S)$ as the best individual in the population of size 20 after 1000 iterations:

$$N = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 112\rangle, \langle 113\rangle\},$$

$$\begin{aligned} P = \{&S \to \langle 113\rangle,\ \langle 113\rangle \to \langle 1\rangle\langle 112\rangle,\ \langle 113\rangle \to \langle 2\rangle\langle 112\rangle,\\ &\langle 113\rangle \to \langle 1\rangle\langle 3\rangle,\ \langle 113\rangle \to \langle 2\rangle\langle 3\rangle,\\ &\langle 112\rangle \to \langle 112\rangle\langle 3\rangle,\ \langle 112\rangle \to \langle 3\rangle\langle 3\rangle,\\ &\langle 1\rangle \to a,\ \langle 2\rangle \to b,\ \langle 3\rangle \to c\}. \end{aligned}$$

6. Learning algorithm to incorporate partially structured examples

While the tabular representation method efficiently represents an exponential number of possible grammatical structures, the learning algorithm using genetic algorithms still takes a large amount of time. On the other hand, while it is impractical and difficult to assume completely structured examples as input, it may be reasonable to assume the availability of some partially structured examples. In this section, we propose an approach to make use of some partial information about the grammatical structure of the given examples and present the learning algorithm to incorporate partially structured examples. We show that partially structured examples contribute to improving the efficiency of the learning algorithm.

The learning algorithm incorporating partially structured examples consists of three steps:

(1) Construct the tabular representation $T(w)$ and the primitive CFG $G(T(w))$ for the given positive examples w,
(2) Eliminate unnecessary nonterminals and production rules from $G(T(w))$ based on the given partially structured examples,
(3) Merge distinct nonterminals to be consistent with the given positive and negative examples using the genetic algorithm for the partitioning problem.

6.1. Partially structured example

We assume the unknown CFG G_*. A (completely) *structured string* is a string with parentheses inserted to indicate the shape of the derivation tree of G_*, or equivalently an unlabeled derivation tree of G_*, that is, a derivation tree whose internal nodes have no labels. A *partially structured string* is defined to be a completely structured string with lack of some pairs of left and right parentheses.

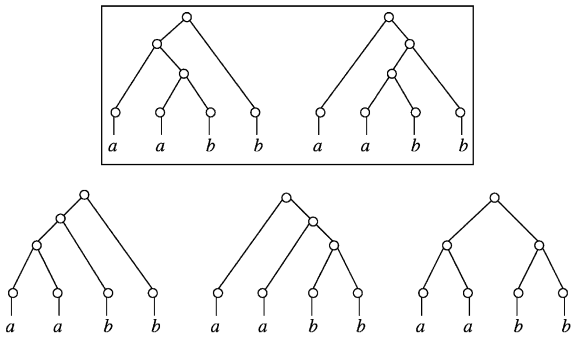

Fig. 8. Upper: two grammatical structures consistent with a partially structured string "$a(ab)b$" and lower, three grammatical structures inconsistent with "$a(ab)b$".

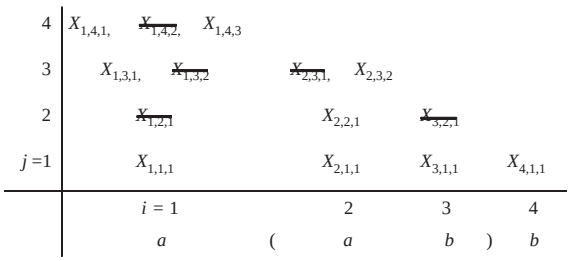

Fig. 9. Unnecessary nonterminals in the tabular representation $T(aabb)$ given a partially structured string "$a(ab)b$".

For example, assume $G_* = (\{S, A, B\}, \{a, b\}, P = \{S \to Ab, A \to aB, B \to ab\}, S)$. For a string "$aabb$", the completely structured string is "$(a(ab))b$", and all partially structured strings are "$a(ab)b$", "$(aab)b$", "$aa(bb)$", and "$(a(ab))b$".

We define the notion of grammatical structures (structured strings) "consistent with" a given partially structured string. If a structured string w' is obtained by inserting some pairs of parentheses into a partially structured string w, we say w' is *consistent with* w. For example, the grammatical structures consistent with a partially structured string "$a(ab)b$" are shown in Fig. 8.

6.2. Eliminating unnecessary nonterminals and production rules

As we have seen, the primitive CFG $G(T(w))$ can generate all possible grammatical structures on w. When a partially structured string w is given, we do not need some nonterminals in $G(T(w))$ which form a grammatical structure "inconsistent" with the partially structured string w. The algorithm to eliminate such unnecessary nonterminals consists of the following two steps:

(1) Eliminate unnecessary nonterminals and production rules based on a given partially structured examples,
(2) Eliminate nonterminals that become useless as a result of step (1).

We have implemented the eliminating algorithm and done some experiments to see the effectiveness of partially structured examples for learning CFGs. For example, given the partially structured string "$a(ab)b$", the eliminating algorithm eliminates 5 nonterminals in the tabular representation $T(aabb)$, as shown in Fig. 9, and eliminate 8 production rules in the primitive CFG $G(T(aabb))$, as shown in Fig. 10.

7. Second experimental results

The second set of experiments investigate how several different partially structured examples reduce the size of the tabular representations and improve the efficiency of the learning algorithm. Let $L_* = \{a^m b^m c^n \mid m, n \geqslant 1\}$ over $\Sigma = \{a, b, c\}$ be the unknown context-free language to be learned, which is the same context-free language tested at Problem 1 in Section 5. We assume the following unknown CFG $G_*=(N, \Sigma, P, S)$ for L_* to give the partially structured examples:

$$N = \{S, A, B, C, X, Y, Z\},$$
$$P = \{S \to XZ,\ S \to XC,$$
$$X \to YB,\ Y \to AX,\ X \to AB,\ Z \to CZ,$$
$$Z \to CC, A \to a,\ B \to b,\ C \to c\}.$$

$$\begin{array}{llll}
P = \{ & S \to X_{1,4,1}, & \overline{S \to X_{1,4,2}}, & S \to X_{1,4,3}, \\
& \overline{X_{1,4,1} \to X_{1,1,1}X_{2,3,1}}, & X_{1,4,1} \to X_{1,1,1}X_{2,3,2}, & \overline{X_{1,4,2} \to X_{1,2,1}X_{3,2,1}}, \\
& X_{1,4,3} \to X_{1,3,1}X_{4,1,1}, & \overline{X_{1,4,3} \to X_{1,3,2}X_{4,1,1}}, & X_{1,3,1} \to X_{1,1,1}X_{2,2,1}, \\
& \overline{X_{1,3,2} \to X_{1,2,1}X_{3,1,1}}, & \overline{X_{2,3,1} \to X_{2,1,1}X_{3,2,1}}, & X_{2,3,2} \to X_{2,2,1}X_{4,1,1}, \\
& \overline{X_{1,2,1} \to X_{1,1,1}X_{2,1,1}}, & X_{2,2,1} \to X_{2,1,1}X_{3,1,1}, & \overline{X_{3,2,1} \to X_{3,1,1}X_{4,1,1}}, \\
& X_{1,1,1} \to \mathrm{a}, & X_{2,1,1} \to \mathrm{a}, & \\
& X_{3,1,1} \to \mathrm{b}, & X_{4,1,1} \to \mathrm{b} & \}
\end{array}$$

Fig. 10. Unnecessary production rules in the primitive CFG $G(T(aabb))$ given a partially structured string "$a(ab)b$".

No.	partially structured example	# N_p	# P_p	GAsteps
1	*aabbccc*	61	139	1200
2	*(aabb)ccc*	27	39	1000
3	*(aabb)(ccc)*	22	29	760
4	*(a(ab)b)ccc*	22	28	440
5	*(a(ab)b)(ccc)*	17	20	400
6	*(a(ab)b)(c(cc))*	15	16	390
7	*((a(ab))b)(c(cc))*	13	13	150

N_p : the number of nonterminals in the primitive CFG $G(T(aabbccc))$,
P_p : the number of production rules in $G(T(aabbccc))$,
GAsteps : the number of generations (iterations) of the genetic algorithm until convergence to a correct CFG.

Fig. 11. Experimental results using partially structured examples.

We have done seven different experiments using seven different partially structured examples for the string "*aabbccc*", which range from an unstructured one to completely structured one according to G_*:

(1) $aabbccc$,
(2) $(aabb)ccc$,
(3) $(aabb)(ccc)$,
(4) $(a(ab)b)ccc$,
(5) $(a(ab)b)(ccc)$,
(6) $(a(ab)b)(c(cc))$,
(7) $((a(ab))b)(c(cc))$.

The summary of our experiments is shown in Fig. 11. We give all positive examples of length up to 10 and all negative examples of length up to 20, and each partially structured example from 1st to 7th. The learning algorithm outputs a correct CFG as the best individual in the population of size 100.

We clearly see that given more structured examples, the number of nonterminals and the number of production rules in the primitive CFG $G(T(aabbccc))$ and the number of iterations of the genetic algorithm until convergence to a correct CFG are all significantly decreasing. Therefore, partially structured examples contribute to improving the efficiency of the learning algorithm. It is found interesting that two CFGs which have different grammatical structures have been learned:

- In the case of the structured examples of 3, 5, 6, and 7 given, the learned CFG is a CFG structurally equivalent to the unknown CFG G_* as follows:

$$N = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 104\rangle, \langle 113\rangle, \langle 121\rangle, \langle 148\rangle\},$$
$$P = \{S \to \langle 104\rangle,\ \langle 104\rangle \to \langle 148\rangle\langle 121\rangle,$$
$$\langle 104\rangle \to \langle 148\rangle\langle 3\rangle,\ \langle 148\rangle \to \langle 113\rangle\langle 2\rangle,$$
$$\langle 148\rangle \to \langle 1\rangle\langle 2\rangle,\ \langle 113\rangle \to \langle 1\rangle\langle 148\rangle,$$
$$\langle 121\rangle \to \langle 3\rangle\langle 121\rangle,\ \langle 121\rangle \to \langle 3\rangle\langle 3\rangle,$$
$$\langle 1\rangle \to a, \langle 2\rangle \to b,\ \langle 3\rangle \to c\}.$$

- In the case of the structured examples of 1, 2 and 4 given, the learned CFG has a smaller number of nonterminals than the unknown CFG G_* as follows:

$$N = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 112\rangle, \langle 116\rangle, \langle 140\rangle\},$$
$$P = \{S \to \langle 140\rangle, \langle 140\rangle \to \langle 140\rangle\langle 3\rangle,$$
$$\langle 140\rangle \to \langle 116\rangle\langle 3\rangle,\ \langle 116\rangle \to \langle 112\rangle\langle 2\rangle,$$
$$\langle 116\rangle \to \langle 1\rangle\langle 2\rangle,\ \langle 112\rangle \to \langle 1\rangle\langle 116\rangle,$$
$$\langle 1\rangle \to a,\ \langle 2\rangle \to b,\ \langle 3\rangle \to c\}.$$

8. Conclusions

We have proposed a new hypothesis representation method, called the tabular representation, based on the parse table for learning CFGs examples. The problem of learning CFGs is reduced to the problem of partitioning the set of nonterminals, and we have used a genetic algorithm for solving the partitioning problem. Furthermore, we have proposed an algorithm for learning context-free grammars from partially structured examples. We have shown via experiments that the partially structured examples contribute to improving the efficiency of the learning algorithm. While it is impractical and difficult to assume completely structured examples as input, it may be reasonable to assume the availability of some partially structured examples. In this sense, our learning algorithm from partially structured examples is more flexible and applicable than the learning algorithm from completely structured examples. Further, our learning algorithm can also identify a grammar having the intended structure, that is, structurally equivalent to the unknown grammar. In practice, there are many partially bracketed sentences available in the databases for natural language processing. Our future work is to apply our learning algorithm to such real databases.

There are some related works to our tabular method for learning CFGs. Dupont [5] presented a grammatical inference method for finite automata by employing the genetic algorithm to search an adequate partition of the set of states in the initial automaton. Our work was motivated by it and extends his method to learning CFGs by introducing the tabular representation. Sakakibara's works [12,7] are also closely related and provides some theoretical foundations for the tabular method. Nakamura's works [13,14] are motivated by our tabular method and uses some heuristic search to find CFGs. While they have done several intensive computational experiments, their heuristic methods have no theoretical background.

Our future work will also include comparisons with other practical learning methods for inductive learning of grammars and rules.

Another interesting and important research direction is to investigate the relation between the degree of structural information (that is, how well structured the given example is) and the number of necessary unstructured examples and

further to theoretically analyze the possibility that the structured examples could decrease the number of unstructured examples (especially, the number of negative examples) required for correct learning.

Acknowledgments

This work was performed in part through Special Coordination Funds for Promoting Science and Technology from the Ministry of Education, Culture, Sports, Science and Technology, the Japanese Government.

References

[1] Y. Sakakibara, Recent advances of grammatical inference, Theoret. Computer Sci. 185 (1997) 15–45.
[2] E.M. Gold, Complexity of automaton identification from given data, Inform. Control 37 (1978) 302–320.
[3] D. Angluin, On the complexity of minimum inference of regular sets, Inform. Control 39 (1978) 337–350.
[4] D. Angluin, Inference of reversible languages, J. ACM 29 (1982) 741–765.
[5] P. Dupont, Regular grammatical inference from positive and negative samples by genetic search: the GIG method, in: Proceedings of Second International Colloquium on Grammatical Inference (ICGI-94), Lecture Notes in Artificial Intelligence, vol. 862, Springer, Berlin, 1994, pp. 236–245.
[6] D. Angluin, A note on the number of queries needed to identify regular languages, Inform. Control 51 (1981) 76–87.
[7] Y. Sakakibara, Efficient learning of context-free grammars from positive structural examples, Inform. Comput. 97 (1992) 23–60.
[8] A.V. Aho, J.D. Ullman, The Theory of Parsing, Translation and Compiling, vol. I: Parsing, Prentice-Hall, Englewood Cliffs, NJ, 1972.
[9] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin, 1996.
[10] G. von Laszewski, Intelligent structural operators for the k-way graph partitioning problem, in: Proceedings of Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991, pp. 45–52.
[11] P. Dupont, L. Miclet, E. Vidal, What is the search space of the regular inference?, in: Proceedings of Second International Colloquium on Grammatical Inference (ICGI-94), Lecture Notes in Artificial Intelligence, vol. 862, Springer, Berlin, 1994, pp. 26–37.
[12] Y. Sakakibara, Learning context-free grammars from structural data in polynomial time, Theoret. Computer Sci. 76 (1990) 223–242.
[13] K. Nakamura, T. Ishiwata, Synthesizing context free grammars from sample strings based on inductive CYK algorithm, in: Proceedings of the Fifth International Colloquium on Grammatical Inference (ICGI-2000), Lecture Notes Artificial Intelligence, vol. 1891, Springer, Berlin, 2000, pp. 186–195.
[14] K. Nakamura, M. Matsuno, Incremental learning of context-free grammars, in: Proceedings of Sixth International Colloquium on Grammatical Inference (ICGI-2002), Lecture Notes in Artificial Intelligence, vol. 2484, Springer, Berlin, 2002, pp. 174–184.
[15] Y. Sakakibara, M. Kondo, GA-based learning of context-free grammars using tabular representations, in: Proceedings of 16th International Conference on Machine Learning (ICML-99), Morgan-Kaufmann, Los Altos, CA, 1999, pp. 354–360.
[16] Y. Sakakibara, H. Muramatsu, Context-free grammars from partially structured examples, in: Proceedings of Fifth International Colloquium on Grammatical Inference (ICGI-2000), Lecture Notes in Artificial Intelligence, vol. 1891, Springer, Berlin, 2000, pp. 229–240.

About the Author—YASUBUMI SAKAKIBARA is an associate professor at the department of Biosciences and Informatics at Keio University, Japan. He received his degree of Doctor of Science from Tokyo Institute of Technology in 1991. His research interests include bioinformatics, DNA computers, machine learning, and formal language theory. He is a member of International Society for Computational Biology, Japanese Society of Bioinformatics, and Japanese Society of Artificial Intelligence.