GA-based Learning of Context-Free Grammars using Tabular Representations.

Conference Paper · January 1999

Source: DBLP

CITATIONS
43

READS
107

2 authors, including:

Yasubumi Sakakibara
Keio University
200 PUBLICATIONS **3,056** CITATIONS

SEE PROFILE

GA-BASED LEARNING OF CONTEXT-FREE GRAMMARS USING TABULAR REPRESENTATIONS

Yasubumi SAKAKIBARA, Mitsuhiro KONDO

Department of Information Sciences, Tokyo Denki University

ABSTRACT: We propose a new algorithm for learning context-free grammars from a finite sample of positive and negative examples using genetic algorithms. The problem of learning context-free grammars from examples has two aspects: determining the grammatical structure (topology) of the unknown grammar and identifying nonterminals in the grammar. To solve these problems efficiently, we propose a new hypothesis representation method which uses a table similar with the parse table used in the efficient parsing algorithm for context-free grammars like the Cocke-Younger-Kasami algorithm. By employing this representation method, the problem of learning context-free grammars from examples can be reduced to the partitioning problem of nonterminals. Then we use genetic algorithms for solving this partitioning problem. We also demonstrate some experimental results using this algorithm.

1 Introduction

Inductive learning of formal languages, often called *grammatical inference*, is an active research area in machine learning and computational learning theory [6]. Especially, 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, identification of minimum-state finite automata from positive and negative examples is known as NP-hard [3]. Nevertheless, one advantage in learning finite automata is that once we construct a prefix-tree automaton for 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. In [2], the genetic algorithm is applied to solve the partitioning problem for the set of states in the prefix-tree automaton.

In this paper, we study the problem of inductively learning context-free grammars from positive and negative examples. This is more difficult learning problem than learning finite automata. The difficult reason is that the problem of learning context-free grammars from examples has two specific aspects: determining the grammatical structure (topology) of the unknown grammar, and identifying nonterminals in the grammar. The first problem is especially hard because the number of all possible grammatical structures to be considered for a given positive example becomes exponential of the length of the positive example. Thus, the hypothesis space of context-free grammars is very large (too large) to search a correct context-free grammar consistent with the given examples. It has been shown in [5] that if information on the grammatical structure of the unknown context-free grammar to be learned 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 using a table which is similar with the parse table used in the Cocke-Younger-Kasami parsing algorithm (CYK algorithm) for context-free grammars of Chomsky normal form. By employing this representation method, the problem of learning context-free grammars from examples can be reduced to the partitioning problem of nonterminals. This hypothesis representation method for CFGs corresponds to the prefix-tree automaton. Then we use genetic algorithms for solving this partitioning problem. We also demonstrate some experimental results using this algorithm.

2 Tabular method for representing hypotheses

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 $A \to BC$

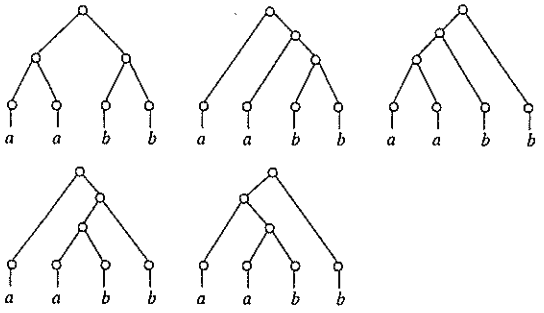

Figure 1: All possible grammatical structures for the string "*aabb*".

or $A \to a$ where $A, B \in N$ and $a \in \Sigma$.

In the following, we fix a terminal alphabet Σ and without loss of generality, we only consider ϵ-free context-free grammars.

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) to search 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 nonterminals in the grammar.

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 as follows when we consider the Chomsky normal form of CFGs:
Thus, the number of all possible grammatical structures to be considered for a string of length n becomes exponential of n. In order to solve this hardness, we propose a new representation method using a table which is similar with the parse table used in the Cocke-Younger-Kasami parsing algorithm (CYK algorithm) [1] for CFGs of Chomsky normal form.

First we introduce 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 for learning CFGs. 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 $w = a_1 a_2 \cdots a_n$ in Σ^* to be parsed according to G, each element of T, denoted $t_{i,j}$, for $1 \le i \le n$ and $1 \le j \le n - i + 1$, 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 $A \Rightarrow a_i a_{i+1} \cdots a_{i+j-1}$, 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 $1 \le i \le n$ and $1 \le j' < j$, 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 ($1 \le k < j$) 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 "*abaab*" becomes as follows:

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
	(a	b	a	a	a)

Figure 2: The parse table T of G for the string "*abaaa* ".

The above parse table can efficiently store all possible parse trees of G for *abaaa*.

Now we propose a representation method using this parse table for efficiently representing hypotheses of CFGs. Given a positive example $w = a_1 a_2 \cdots a_n$, the *tabular representation* for w is the triangular table $T(w)$ where each element, denoted $t_{i,j}$, for $1 \le i \le n$ and $2 \le j \le n - i + 1$, 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 $\{X_{i,1,1}\}$. The *primitive* CFG $G(T(w)) = (N, \Sigma, P, S)$ derived from the tabular representation $T(w)$ is defined to be as follows:

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

Thus for each nonterminal $X_{i,j,k}$ at entry $t_{i,j}$, we have the production rules $X_{i,j,k} \to X_{i,k,l_1} X_{i+k,j-k,l_2}$ with the right-hand sides of the nonterminals at entries $t_{i,k}$ and $t_{i+k,j-k}$. The number of nonterminals contained in $G(T(w))$ is at most n^3 and the size (the number

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

Figure 3: The tabular representation for w.

of production rules) of the primitive CFG $G(T(w))$ is very roughly $O(n^5)$, 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 $G(T(aabb))$ becomes as follows:

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

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

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

While the size of $G(T(aabb))$ is polynomial of the length of the given example, this CFG $G(T(aabb))$ can generate all possible grammatical structures for *aabb* as shown in Figure 5.

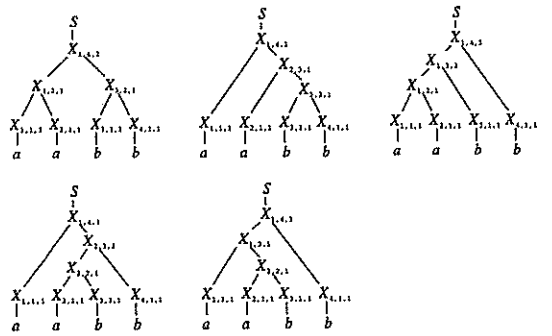

Figure 5: All possible parse trees of $G(T(aabb))$ for "*aabb*".

3 Learning algorithm using GA

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 $w \in L(G)$. The learning is completely a reverse process that 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.

Our learning algorithm for CFGs will be 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 merges distinct nonterminals to be consistent with the given positive and negative examples and minimize the number of nonterminals in $G(T(w))$. Then, we use the genetic algorithm to solve the partitioning problem for the set of nonterminals $\{X_{i,j,k} \mid 1 \le i \le n, 1 \le j \le n-i+1, 1 \le k < j\}$. This partitioning problem contains the problem of finding minimum-state finite automata consistent with the given examples and hence it is NP-hard. Therefore, it is reasonable to use the genetic algorithm for solving the computationally hard problem of partitioning nonterminals. The figure 6 illustrates our scenario.

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 live production rule in G_*, that is, every live production is used at least once to generate the subset.

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:

$$
\begin{aligned}
N' &= \pi \quad \text{(the set of blocks of } \pi),\\
P' &= \{K(A,\pi) \to K(B,\pi)\ K(C,\pi) \mid A \to BC \in P\}\\
&\quad \cup \{K(A,\pi) \to a \mid A \to a \in P\},\\
S' &= K(S,\pi).
\end{aligned}
$$

Now we assume that a finite sample 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 described as follows:

The Learning Algorithm TBL:

1. Construct the tabular representation $T(w)$ for each positive example w in U_+;
2. Derive the primitive CFG $G(T(w))$ for each w in U_+;

Given a positive example "*abaaa*"

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

STEP 1:

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
	(a	b	a	a	a)

⇓ **merge nonterminals**

STEP 2:

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
	(a	b	a	a	a)

⇓

STEP 3:

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

Figure 6: The tabular representation $T(abaaa)$, merging nonterminals, and the resulting CFG.

3. Take the union of those primitive CFGs, that is,

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

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

We employ the 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 [4] and we take a successful approach by Von Laszewski for partitioning n elements into k categories (blocks) [4]. This approach encodes partitions using *group-number encoding*, that is, partitions are represented as strings of integer number of length n,

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

where the j-th integer $i_j \in \{1, \ldots, k\}$ indicates the group number assigned to element j. This representation is supported by specific GA operators called "intelligent structural operators": structural crossover and structural mutation.

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 decode the following partitions:

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

First, a random block is selected, say block #2. This block is copied (introduced) from p_1 into p_2:

$$p_2' = (112423222235)$$

and the resulting partition is:

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

structural mutation: it replaces a single component of a string by some random number. Thus, a parent

$$p_2' = (112423222235)$$

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

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

In addition to those operators, we introduce a specific 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 contributes to deleting unnecessary nonterminals in a CFG $G(T(w))$.

Finally, 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 a partition which p encodes for the set of nonterminals in the CFG $G(T(U_+))$. Let U_+ be the given positive examples and U_- be the given negative examples. The fitness function f is defined as follows:

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

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

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

where C_1, C_2 are some constants.

4 Experimental results

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

Experiment 1: The unknown (target) context-free language is $\{a^m b^m c^n \mid m,n \geq 1\}$ over $\Sigma = \{a,b,c\}$. This context-free language is especially important to see 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 2000 iterations:

$$\begin{aligned} N &= \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 104\rangle, \langle 113\rangle, \langle 121\rangle, \langle 148\rangle\}, \\ P &= \{\ S \to \langle 104\rangle, \\ &\quad \langle 104\rangle \to \langle 148\rangle\ \langle 121\rangle, \quad \langle 104\rangle \to \langle 148\rangle\ \langle 3\rangle, \\ &\quad \langle 148\rangle \to \langle 113\rangle\ \langle 2\rangle, \quad \langle 113\rangle \to \langle 1\rangle\ \langle 148\rangle, \\ &\quad \langle 148\rangle \to \langle 1\rangle\ \langle 2\rangle, \\ &\quad \langle 121\rangle \to \langle 3\rangle\ \langle 121\rangle, \quad \langle 121\rangle \to \langle 3\rangle\ \langle 3\rangle, \\ &\quad \langle 1\rangle \to a, \quad \langle 2\rangle \to b, \quad \langle 3\rangle \to c \qquad \}. \end{aligned}$$

Experiment 2: The unknown context-free language is $\{ac^m \mid m \geq 1\} \cup \{bc^m \mid m \geq 1\}$ 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:

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

5 Concluding remarks

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 partitioning problem of nonterminals, and we have used the genetic algorithm for solving the partitioning problem.

We are now working on more experiments and more heavy experiments. Our future works are comparisons with other practical learning methods and investigations of theoretical analyses for the tabular representation methods for learning CFGs from examples.

Acknowledgments

This work is supported by "The TDU Foundation for Advancement of Research" No. Q97J-07.

References

[1] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing.* Prentice Hall, 1972.

[2] P. Dupon. Regular grammatical inference from positive and negative samples by genetic search: the GIG method. *Proceedings of Second International Colloquium on Grammatical Inference (ICGI-94)* (LNAI 862, Springer-Verlag, 1994), 236–245.

[3] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[4] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1996.

[5] Y. Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97:23–60, 1992.

[6] Y. Sakakibara. Recent Advances of Grammatical Inference. *Theoretical Computer Science*, 185:15–45, 1997.