Improvement on Fitness Function for Learning Context-Free Grammars Using Tabular Representations

Gend Lal Prajapati, *Member, IAENG*, M.Vignesh

***Abstract*—The problem of learning context-free grammars (CFGs) using genetic algorithms is considered. In particular, we deal with a scheme for learning CFGs that first generates primitive CFGs, then genetic algorithms are used to find the smallest consistent CFGs by merging the non-terminals repeatedly (i.e., by partitioning the set of non-terminals). The primitive context-free grammar (CFG) based on the inputs that belong to the unknown target language is formed using the efficient parsing algorithm for CFG such as the Cocke-Younger-Kasami algorithm. By employing this representation method, the problem of learning CFGs from examples can be reduced to the problem of partitioning the set of non-terminals. In order to formulate the solution of this partitioning problem using genetic algorithms, the problem of calculating fitness effectively and efficiently is still persists. In this paper, we present a scheme for determining fitness value that can improve the performance of learning algorithms for CFGs.**

***Index Terms*— Context-free grammar, genetic algorithm, tabular representation, fitness function, grammatical inference**

I. Introduction

LEARNING structural models from data is known as grammatical inference (GI). The data can represent sequences of natural language corpora, biosequences (DNA, RNA, primary structure of proteins), speech etc., but can also include trees, metabolic networks, social networks or automata. Typical models include formal grammars [7], and statistical models in related formalisms such as probabilistic automata, Hidden Markov Models, probabilistic transducers or conditional random fields. The major learning models proposed for formal languages are: *identification in the limit* [4], *query learning model* [10], and the *probably approximately correct learning model*, addressed *PAC learning model*, in short [11]. GI is the gradual construction of a model based on a finite set of sample expressions. In general, the training set may contain both positive and negative examples from the language under study. If only positive examples are available, no language class other than the finite cardinality languages is learnable [4].

Manuscript received October 14, 2011; revised January 05, 2012.

G. L. Prajapati is with the Department of Computer Engineering, Institute of Engineering & Technology, Devi Ahilya University, Indore-452001 INDIA (phone: +91 731 2366800, 2462311; fax: +91 731 2764385; e-mail: glprajapati1@gmail.com).

M.Vignesh is with the Swami Vivekanand College of Engineering, Indore- 452020 INDIA (e-mail: mvignesh1990@gmail.com).

It has been proved that deterministic finite automata are the largest class that can be efficiently learned by provable converging algorithms. There is no context-free grammatical inference theory which provable converges, if language defined by a grammar is infinite. Building algorithms that learn context–free grammars (CFGs) is one of the open and crucial problems in the grammatical inference. The approaches taken have been to provide learning algorithms with more helpful information, such as negative examples. Some people have taken the notion of unlabelled derivation trees that describe the grammatical structures of the target language in order to formulate alternative representation of CFGs [8], [9].

Due to the hardness of learning CFGs many researchers have attacked the problem of grammar induction by using evolutionary methods to evolve (stochastic) CFG or equivalent pushdown automata, but mostly for artificial languages like brackets, and palindromes. In this paper, we introduce an improvement on the tabular representation algorithm (TBL) dedicated to inference of CFGs in Chomsky normal form. The TBL was proposed by Sakakibara and Kondo in [5] and later on Sakakibara analyzed some theoretical foundations for the algorithm [1].

A. *Context-Free Grammar Parsing*

A context free grammar (CFG) is a quadruple $G = (\Sigma, V, R, S)$, where Σ is finite set of terminal symbols called alphabet and V is finite set of non-terminal symbols such that $\Sigma \cap V = \emptyset$. R is set of production rules in the form $W \rightarrow \beta$ where $W \in V$ and $\beta \in (V \cup \Sigma)^*$. $S \in V$ is a special non-terminal symbol called start symbol. All derivations start using the start symbol S. A derivation is sequence of rewriting the string containing symbols $(V \cup \Sigma)^*$ using production rules of CFG G. In each step of a derivation a non-terminal symbol from string is selected and replaced by string on the right side of the production rule.

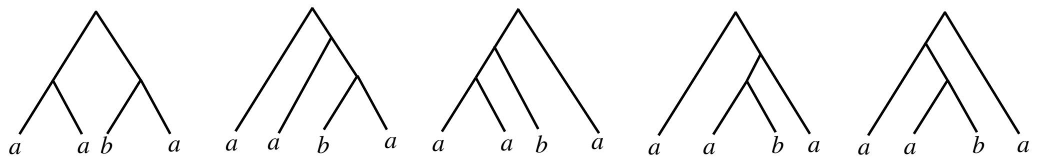
For example if we have CFG $G = (\Sigma, V, R, S)$, where

$\Sigma = \{a, b\}$,
$V = \{A, B\}$,
$R = \{A\rightarrow a, A \rightarrow BA, A \rightarrow AB, A\rightarrow AA, B\rightarrow b\}$,
$S = A$.

Then the derivation can have the form:

$A\rightarrow AA \rightarrow ABA \rightarrow AABA\rightarrow aABA \rightarrow aABa \rightarrow aAba \rightarrow aaba$.

More details regarding the CFGs can be seen in the literature [2] and [3].



(a)

$j = 4$	<12> <13> <14>			
$j = 3$	<8> <9>	<10> <11>		
$j = 2$	<5>	<6>	<7>	
$j = 1$	<1>	<2>	<3>	<4>
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	a	*a*	*b*	*a*

(b)

Fig. 1. All possible grammatical structures of word "*aaba*" (a), and its tabular representation (b)

The derivation ends when there are no more non-terminal symbols left in the string. The language generated by CFG is denoted $L(G) = \{x \in \Sigma^* |$ the string x can be derived starting from the start symbol $S\}$. The word x is a string of any number of terminal symbols derived using production rules from CFG G. A CFG $G = (\Sigma, V, R, S)$ is in Chomsky normal form if every production rule in R has the form: $A \rightarrow BC$ or $A \rightarrow a$, where $A, B, C \in V$ and $a \in \Sigma$.

A CFG describes specific language $L(G)$ and can be used to recognize words (strings) that are members of language $L(G)$. But to answer the question which words are members of language $L(G)$ and which are not we need a parsing algorithm. This problem, called membership problem can be solved using CYK algorithm. This algorithm is a bottom up dynamic programming technique that solves the membership problem in a polynomial time.

B. *CFG Induction Using TBL Algorithm*

Grammar induction target is to find a CFG structure (set or rules and set of non-terminals) using positive and negative examples in a form of words. It is a very complex problem, especially hard is finding grammar topology (set of rules). Its complexness results from great amount of possible solutions that grows exponentially with length of the examples. TBL algorithm uses tabular representations of positive examples. Tabular representation is similar to parse table of CYK algorithm and is able to remember exponential number of possible grammatical structures of example in a table of polynomial size. Fig. 1 shows all possible grammatical structures of word "*aaba*" and its tabular representation.

The following pseudo code shows all steps of the TBL algorithm presented in [1]:

```
1. Create tabular representation $T(w)$ of each positive example $w$ in $U+$.
2. Derive primitive CFG $G(T(w))$ and $G_w(T(w))$ for each w in $U+$
3. Create union of all primitive CFG's, $G(T(U+))$ = Union of all $w \in U+ \{G_w(T(w))\}$
4. Find smallest partition $\pi^*$ such that $G(T(U+))/\pi^*$ is coherent with $U$, which is compatible with all positive and negative examples $U+$ and $U-$
5. Return result CFG $G(T(U+)) / \pi^*$
```

Fig. 2 shows the tabular representation and the primitive grammar developed for the given sentence "*aaba*".

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

(a) The tabular representation

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

(b) The Primitive grammar

Fig. 2. The tabular representation $T(aaba)$ (a), and the derived primitive CFG G ($T(aaba)$) (b).

Once the partitioning has been completed, the subsequent chromosomes are generated by applying genetic operators like structural crossover, mutation and special deletion. Then we generate the next population set by choosing probabilistically the fit chromosome and forwarding it to the next generation.

The fitness function proposed by Sakakibara does not consider the negative examples role in the fitness and hence it is weak. We thereby, propose a solution to improve the quality of the fitness and lead to results with better efficiency.

Our method of the improved fitness function is given in the next section.

II. METHODOLOGY

Our main aim is to generate a grammar from a finite set of positive and negative examples. A temporary set of grammars is generated from the given set of examples. Successive populations are generated by applying the genetic operations namely structural crossover, structural mutation and special deletion. The transition of a particular set of chromosomes (grammar) into the next generation is decided by its fitness.

Step 1: Calculation of fitness

The two methods for the calculation of fitness of a particular grammar that we propose are as follows:

Method 1:

$$F = \frac{|\{w \in U+ \mid w \in G(U+)\} - \{w \in U- \mid w \in G(U-)\}|}{|U+|}$$

And Method 2:

$$f1 = \frac{|\{w \in U+ \mid w \in G(U+)\}|}{|U+|}$$

$$f2 = \frac{|\{w \in U- \mid w \in G(U-)\}|}{|U-|}$$

And $F = f1 - f2$

U+ is the no of positive examples and *U-* is the number of negative examples.

G(U+) and *G(U-)* are no of positive and negative examples accepted by the grammar respectively.

These methods are better because:

1) Sakakibara's method [1] does not consider the negative set of examples into consideration. It happens so in his method that if a grammar even accepted only one negative example, its fitness would be 0. Let's consider an example: Let we require a grammar which accepts say 100 positive example and must reject 80 negative examples.
   Case 1: Grammar accepts 99 positive and one negative example. Fitness = 0
   Case 2: Grammar accepts 99 positive and 79 negative examples. Fitness = 0
   The first grammar is almost perfect whereas the second one is nowhere near it. Yet these are rated at the same fitness levels.
2) Marcin Jaworski's and Olgierd Unold's methods [6] took into consideration the partitions too. Hence the fitness calculated was partition dependent.
3) Once any one of the methods defined by me is used, the fitness value of each grammar in the population is obtained. These fitness values are of the type: $F <= 1$.

Step 2: Generation of next population

We generate the next population based on the fitness values obtained. The pseudo code is as follows:

```
1. Choose the minimum value of fitness from the set, say it is 'x'.
2. Compute the increment value for the entire set $i = |x| + 1$.
3. Shift each value in the set, i.e. add 'i' to each element in the set.
4. Calculate ratio of each updated value to the sum of all the values in the set. This results in a new set of numbers which add up to 1.
5. Represent these ratios (parts) in a pie chart format and employ roulette wheel.
6. Generate a random number between 0 and 360. The value generated is the angle on the pie chart. The value having more area hence has a greater probability of getting selected that those with a lower area.
7. Repeated selection using the roulette wheel mechanism will lead to the entire set of the next population.
```

III. AN EXAMPLE

A. Improvement on Fitness Function

We now demonstrate the working of the fitness function and its advantage in this section. A couple sample tests and their results are as follows:

TEST 1:

Positive examples given by user: 1000
Negative examples given by user: 800
Positive examples accepted by the grammar: 600
Negative examples accepted by the grammar: 350

Fitness:
Fitness Value using Method 1: 0.25
Fitness Value using Method 2: 0.162499
Fitness Value using Sakakibara's method: 0

TEST 2:

Positive examples given by user: 1000
Negative examples given by user: 800
Positive examples accepted by the grammar: 999
Negative examples accepted by the grammar: 1

Fitness:
Fitness Value using Method 1: 0.998
Fitness Value using Method 2: 0.99775
Fitness Value using Sakakibara's method: 0

It is clearly seen that even though the second grammar (TEST 2) is a lot better than the first one (TEST 1), using

Sakakibara's method it is rated 0 in both cases. Whereas the comparative fitness is obtained in our method (0.25, 0.162499 and 0.998, 0.99775 respectively) proving that it is better capable of detecting the grammar that is more close to the answer that is needed.

B. Generation of the next population

For example let there be four fitness values say:

a) -2.3
b) .63
c) .99
d) -.235

We choose among all the fitness values in the population the least valued fitness.

Chosen value = -2.3

Then we shift each fitness value in the population by the |least value| +1.

Shift them by |least value| +1. This becomes:

a) 1
b) 3.93
c) 4.29
d) 3.065

Now calculate the ratio of each fitness value to the total of the fitness values.

Total = 1+3.93+4.29+3.065 = 12.285

Ratios:

a) 1/12.285 = .0814
b) 3.93/12.285 = .3199
c) 4.29/12.285 = .3499
d) 3.065/12.285 = .2494

Make a virtual pie chart with its area divided into parts corresponding to the ratios.

Now a roulette wheel method can be used to generate a random number between 0 and 360. The number generated specifies an angle between 0 and 360 of the roulette wheel. The selected angle denotes the selected chromosome to be sent into the next generation. The probability of the chromosome with more fitness and hence more area to be selected is greater than the probability of the chromosome with a lesser fitness. Repeated selection using the roulette wheel mechanism will lead to the entire set of the next population. The pie chart constructed for the above example is shown in Fig. 3.

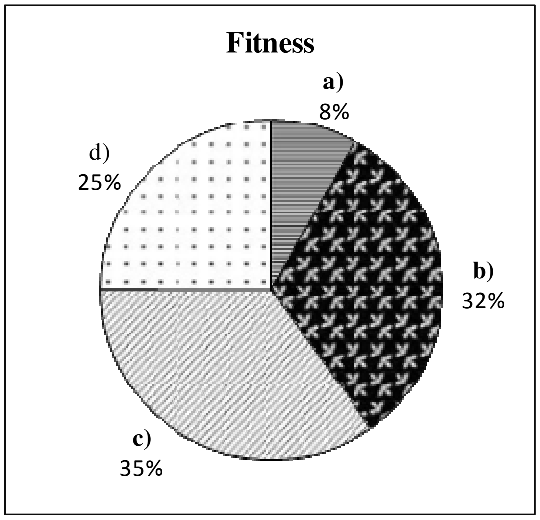


Fig. 3. The pie chart

IV. Conclusions

Our methods produced results having fitness that took into consideration, the importance of both the positive and negative examples. Hence the functioning of the algorithm won't be single minded and positive oriented anymore. Moreover, the method also eliminates the partition dependency of the results and puts forth a more straightforward way of handling the chromosomes.

Further researches would help in generating more efficient fitness functions and better algorithms for the partitioning problem that would significantly reduce the complexity of the said problem and make it a more deterministic approach.

References

[1] Yasubumi Sakakibara, "Learning context-free grammars using tabular representations", *Pattern Recognition* 38 (2005) pp. 1372 – 1383.

[2] Daniel I A Cohen, "Introduction to Computer Theory" second edition, John Wiley & Sons, Inc. ISBN 9971-51-220-3

[3] John Martin, "Introduction to Languages and the Theory of Computation" Third Edition, Tata McGraw Hill Publishing Company Ltd. ISBN 0-07-049939-X

[4] Gold E.M.: "Language Identification in the Limit", *Information Control*, 10: (1967) pp. 447-474.

[5] Sakakibara Y., Kondo M.: "GA-based learning of context-free grammars using tabular representations", in *Proc. 16th International Conference in Machine Learning* (ICML-99)

[6] Marcin Jaworski and Olgierd Unold, "Improved TBL algorithm for learning context-free grammar," in *Proc. the International Multi-conference on Computer Science and Information Technology*, (2007) pp. 267 – 274

[7] G.L. Prajapati, "Advances in Learning Formal Languages," in *Proc. International MultiConference of Engineers and Computer Scientists* (IMECS 2011), The 2011 IAENG International Conference on Artificial Intelligence and Applications (ICAIA 2011), Hong Kong, Proceedings in Lecture Notes in Engineering and Computer Science, ISSN: 2078-0958 (Print), ISSN: 2078-0966 (Online), ISBN: 978-988-18210-3-4, vol. 2188, pp. 118-126 2011

[8] Sakakibara, Y., "Efficient Learning of Context-Free Grammars from Positive Structural Examples," *Information and Computation*, vol. 97, pp. 23-60, 1992.

[9] Prajapati G.L., Chaudhari N.S., and Chandwani M., "Efficient Incremental Model for Learning Context-Free Grammars from Positive Structural Examples," in *Proc. 5th Hellenic Conference on Artificial Intelligence*, SETN-08, Syros, Greece, Proceedings in *Artificial Intelligence: Theories, Models and Applications*, Lecture Notes in Artificial Intelligence (LNAI), Springer Verlag, Berlin, ISSN: 0302-9743, ISBN: 978-3-540-87880-3, vol. 5138, pp. 250-262, 2008.

[10] Angluin, D., "Queries and Concept Learning," *Machine Learning*, vol. 2, pp. 319-342, 1988.

[11] Valiant, L.G., "A Theory of the Learnable," *Communications of the ACM* vol. 27, pp. 1134-1142, 1984.