**IAES International Journal of Artificial Intelligence (IJ-AI)**
Vol. 6, No. 3, September 2017, pp. 100~111
ISSN: 2252-8938, DOI: 10.11591/ijai.v6.i3.pp100-111


# Genetic Algorithm for Grammar Induction and Rules Verification through a PDA Simulator

**Hari Mohan Pandey**
Department of Computer Science and Engineering, Amity University, India

**Article Info**

***Article history:***

Received Jun 6, 2017
Revised Aug 8, 2017
Accepted Aug 22, 2017

***Keyword:***

Context free grammar
Genetic algorithm
Grammar induction
Parsing
Pushdown automata

**ABSTRACT**

The focus of this paper is towards developing a grammatical inference system uses a genetic algorithm (GA), has a powerful global exploration capability that can exploit the optimum offspring. The implemented system runs in two phases: first, generation of grammar rules and verification and then applies the GA's operation to optimize the rules. A pushdown automata simulator has been developed, which parse the training data over the grammar's rules. An inverted mutation with random mask and then 'XOR' operator has been applied introduces diversity in the population, helps the GA not to get trapped at local optimum. Taguchi method has been incorporated to tune the parameters makes the proposed approach more robust, statistically sound and quickly convergent. The performance of the proposed system has been compared with: classical GA, random offspring GA and crowding algorithms. Overall, a grammatical inference system has been developed that employs a PDA simulator for verification.



***Corresponding Author:***

Hari Mohan Pandey,
Department of Computer Science and Engineering,
ASET, Amity University,
Sector 125, Noida, U.P. India.
Email: hari04top@yahoo.co.in

## 1. INTRODUCTION

The Context free grammars (CFG) (BNF Grammars) have been used extensively to describe the syntax of programming languages and natural languages. Parsing algorithms for CFGs play a significant role in the implementation of compilers and interpreters for the programming languages and of programs, which "understand" or translate the natural languages. There exist two different types of parser: top down and bottom up parser. Several good algorithms were proposed in literatures conduct sequential parsing of the context free languages (CFLs). Some of the popular parsing algorithms are CYK parsing [1, 2]; Earley's parsing [3], and Tomita parsing [15]. There are many other standard parsing algorithms also available, which include operator precedence parsing, predictive parsing, recursive descent parsing, non-recursive descent parsing, LR parsing, and LALR parsing [16]. Noam Chomsky (1956) presented a classification scheme known as the Chomsky hierarchy [17]. The classification scheme proposed is connected to the classification of automata as shown in the Figure 1, which can be used to recognize a specific language as well as the grammar used to recognize them.

The problem of recognizing a language can be explained as: "let $L$ be any language over an alphabet $\sum$, the problem is to design an automata $M$ that accept an input sequence in $\sum^*$. The $M$ only accept the input sequence in $\sum^*$ if they are valid element of $L$, otherwise rejects".

Figure 1 depicts that each class of language in the Chomsky hierarchy is generated by a specific type of grammar, which will then recognize by an appropriate type of automata. Moving up in the hierarchy of the languages, the type of automata required to recognize the language is challenging, whereas the type of

grammar needed to create the language becomes more general. One can see that Type 2 languages, generated by the CFG, are the class of languages that can be recognized by pushdown automata (equivalent to finite automata), have an unbounded stack available.

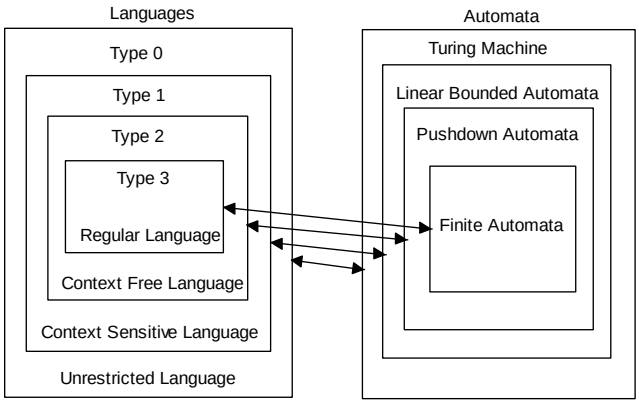


Figure 1. Chomsky hierarchy

Grammatical inference (GI) or grammar learning deals with idealized learning procedure to acquire grammars on the basis of the evidence about the languages [4-6], was extensively studied in [6-14] due to its wide fields of applications to solve the particle problems. In GI process, an individual grammar rule needs to be verified using an input string of a specific language over finite state controller, whereas a pushdown automaton is used to generate the equivalent proliferation for the language.

This paper presents a GI algorithm combines with the genetic algorithm (GA), which has a powerful global exploration capability. A PDA simulator has been implemented in the computational experiment that employs a recursive method for the parsing that results in terms of acceptance or rejection of the string. The procedure for the PDA simulator has been implemented and explained with the help of examples. The author has implemented two points cut crossover based cyclic crossover and inverted mutation with random offspring then applied 'XOR' operation for the reproduction. The reproduction operators employed introduces diversity in the population helps the GIGA to explore the search space adequately. The author has compared the performance of the proposed GIGA with three existing algorithms namely classical genetic algorithm (CGA) [38], random offspring generation genetic algorithm (ROGGA) [36] and crowding algorithm [37]. The comparative results lead to a conclusion that the proposed GIGA outperformed theses algorithms.

The rest of the paper is organized as follows: Section 2 reviews the related researches on GI problem. Section 3 presents the CFG induction approach using GA adapted in this paper. This section shows the chromosome structuring, fitness function and reproduction operators that have been employed for CFG induction. Section 4 shows the push down automata simulator with the various methods incorporated for the verification purpose. The experimental setup, results and discussion on results are given in Section 5. Section 6 concludes the paper and assesses the future perspectives.

## 2. RELATED WORK

Gold [18] proposed the first learning model to address "*Is the information sufficient to determine which of the possible languages is the unknown language?*", but it was suffered because sufficient information about the identification of correct grammar does not exist. To address the issue of [18], Angluin [19] proposed tell tales. Although, Gold [18] laid the foundation of learning model, but Bunke and Alberto [20] proposed the first usable learning model also suffered since it was unable to deal with negative data, was not fit for noisy data, does not suitable for a finite state machine, therefore good formal language theory was lost. Teacher and query learning model, also referred as an oracle was proposed in [21] is a supervised learning model in which an oracle knows the answer. It was found capable in answering a particular type of query using an inference system, but implementing an oracle is a matter of concern, which needs vast information, hence less commonly used, whereas Gold's model is more popular.

Valiant [22] combined the best features of [18] and [21] and presented probably approximately correct (PAC) learning model. The PAC learning model suffered due to the following reasons:

1) PAC learning assumes that the inference algorithm must learn in polynomial time under all distribution, but this believes is too string in reality.
2) PAC learning is not fit for negative and NP hard equivalence results. A modified version of PAC model was present in [23] in which simplicity was measured using Kolmogorov complexity.

Inductive inference is the process of making generalization from the input. Wyard [24] showed the impact of different grammar representation. The experimental results presented that the evolutionary algorithm (EA) using standard context free grammar (CFG) (Backus Naur Form (BNF)) outperformed others [24]. Thanaruk and Okumaru [25] classified GI into three categories: supervised, semi-supervised and unsupervised. Javed et. al [26] proposed a genetic programming (GP) based approach to learn the CFG. The work presented in [26] was the extension of the work done in [24]. A sequential structuring approach was proposed in [28] that perform coding and decoding of binary coded chromosomes into terminals and non-terminals and vice-versa. A GA based CFG induction library was proposed in [28, 29]. A case study on GI was proposed in [27] includes the basics of GA in which simple crossover (one and two points) and bit inversion mutation operators were utilized. Hrncic et al [31, 32] implemented a memetic algorithm (MA) for GI, which assist domain experts and software language engineers to develop domain-specific languages (DSLs) by automatically producing a grammar.

Theorists and empiricists are the two main groups contributing in GI [33, 34]. Language classes and learning models were considered by the theorists group, whereas empiricists group dealt with practical problems. A detailed survey of various GI algorithms has been presented in [4], [33, 35].

## 3. GRAMMATICAL INFERENCE USING GENETIC ALGORITHM

In this section, the author has presented a block diagram, shows the process of the GI uses GA. There exists different GI approaches were proposed as discussed in Section 2. The authors have implemented a GA based approach for CFG induction. Figure 2 depicts the block diagram for CFG induction uses a GA.

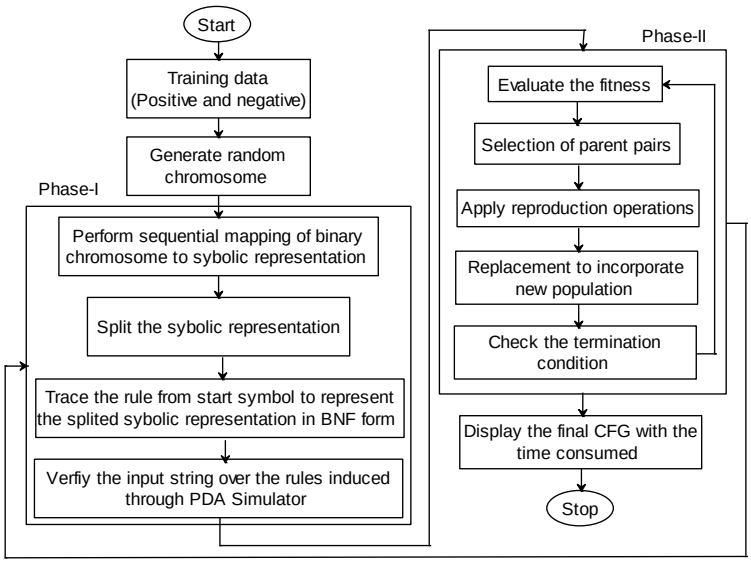


Figure 2. Block diagram for context free grammar induction using GA.

The overall process has been divided into two different phase. The Phase-1 is responsible for production rule generation and verification over PDA simulator, whereas Phase-2 shows the steps of GA incorporated to optimize the search process and explore the search space adequately.

The GI process starts generating the initial random binary chromosome (BC), which is then mapped to appropriate symbolic chromosome (SC) representation of terminals and non-terminals in a sequential manner. The SC has been divided into equal block size of five equal to production rule length. Each SC has been traced from the start symbol 'S' to terminal to remove use less production and rest of the productions have been tested for removal of left recursion, unit production, ambiguity and left factor. A string to be tested has been selected and passed for the validity, i.e. acceptability of the CFG equivalent to the chromosome.

The test string and the CFG rules are the input to the finite state controller, which verifies the acceptability through the proliferation on the PDA.

### 3.1. Chromosome Structure

A random initial binary chromosomes (BC) consist of sequence of 0's and 1's have been created (see Figure 4). The BC is mapped into terminals and non-terminals in a sequential manner following 3-bit/4-bit coding that depend on the number of symbols present in the language. If more than two symbols are used in the language then 4-bit representation has been used otherwise use 3-bit representation. The start symbol 'S' has been mapped at "000" and '?' represents the null symbol mapped at "010", whereas for others symbols (terminals and non-terminals) are used as appropriate.

Binary chromosome:
0001000100000100100001010011110001010001100100000100111010110010000110010011101010100011000001000101101100000001101101110

Coding of terminal and non-terminals

| Non-terminals | Terminals |
|---|---|
| S → 000 | 1→100 |
| A→001 | 0→101 |
| B→111 | ?→010 |
| C→011 | ?→110 |

Symbolic chromosome representation: S1?S??S0ABS0S??S?C0CASCAA?0?A1S1???SA00?

The SC's are divided into block size of five equal to the production rule length as shown in Table 2, which are then traced from 'S' to remove the insufficiency presents. The PDA simulator accepts test string and the production rules as an input for any specific language, verifies the acceptability via proliferation on the PDA.

### 3.2. Fitness Function

The fitness of an individual has been calculated in each GA run and then selection of parent string is done. In GI problem, the fitness of an individual chromosome largely depends upon the acceptance (rejection) of the positive and negative sample strings. The fitness value increase for accepting positive (AP) and rejecting negative (RN) sample, whereas it decreases for accepting negative (AN) and rejecting positive (RP) sample. The problem specific factor (s) also plays a significant role in GA's performance. In case of GI, production rule length (PR) is an important factor, has been considered in the fitness calculation. Equation (1) has been applied to calculate the fitness of an individual.

$$Fitness = \sum C*((AP+RN)+(AN+RP))+(2*C-PR) \quad (1)$$

The following convention has been followed for the selection of the best grammar rules: "A grammar that accepts all the positive strings and rejects the entire negative string from set of training data with minimum number of production rules". The value of constant (C= 10) is found sufficient to accommodate grammar rules blocks present in the symbolic chromosome.

### 3.3. Reproduction Operators

The GA's performance largely depends on the two most commonly used genetic operators are crossover and mutation. The operators' crossover and mutation play a significant role in the population diversity management and therefore improves the convergence speed.

A variation of two point crossover based on the cyclic crossover has been incorporated to perform the crossover operation. The inverted mutation method has been applied with random mask. As we know the mutation operator introduces diversity in the population helps to keep the search process alive. Random mask is useful in achieving the diversity. The following convention has been applied: "simply apply "XOR" operation between the parent strings received after crossover operation and the random offspring". An example for both crossover and mutation operations have been represented in Figure 3.

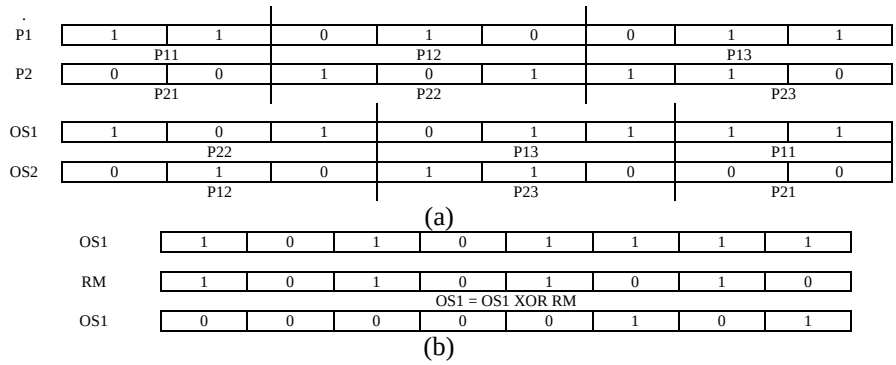


Figure 3. Demonstrations of crossover and mutation operations. (a) Representing the two point cut crossover based on cyclic crossover. (b) Inverted mutation by generating random mask (RM) and then applying XOR operation.

### 3.3. Verification of Rules using PDA Simulator

This section explains the working of pushdown automata incorporated during CFG induction. The PDA simulator utilizes various methods for the verification purpose. The description of each method used in the implementation of PDA simulator is outlined below:

*PDA_Simulator (input_String, Stack):* It accepts input string and stack as an input. The purpose of this procedure is to simulate the overall working. It uses top of stack (TOS) for the simulation purpose. If TOS = input symbol = $, indicate that the input string is accepted for the grammar. If TOS = Terminal and the first symbol matches with the symbol present at TOS, then both the symbols are removed. If TOS = non-terminal (X), then in such situation, all the production of 'X' replace the TOS with the right side of the production rule and call the *PDA_Simulator()* recursively. In case of non-acceptance, the selected production rules repeat the process for another production that starts with 'X'.

*get_input (T_input_String):* It simply returns the next terminal present in the input string.

*get_Top_S (T_Stack):* Returns TOS symbol present in the stack.

*remove_first_input ():* It is used to remove the first symbol of the input string.

*remove_TOP_Stack ():* It is used to remove TOS item from the stack.

*copy_right (T_Stack, X):* It accept stack and production rules as an input. It is used to replace the TOS with right side of the production rule.

*verify_string (String str):* This procedure is executed to take the decision about the acceptance or rejection of the input string "str"

The procedure *PDA_Simulator (input_String, Stack)* and its associated functions definitions are given below:

***Procedure: PDA_Simulator (input_String, Stack)***

```
Begin
 Set T_input_String = input_String
 Set T_Stack = Stack
 Set X = get_input()
 Set S = get_Top_S()
 If stack overflow() then
 return (2)
 End If
 If (End_of_String && End_of_Stack) then
 return (1)
 End If
 If(S == Terminal && X = = S)
 perform remove_first_input ()
 perform remove_TOP_Stack ()
 End If
 If(S = = non-terminal) then
 For all production rules in P starting with S
 If production rule is Null then
 perform remove_TOP_Stack ()
```

```
        Else
        perform remove_TOP_Stack ()
        perform copy_right (S, x)
        End For
        End If
Set Found = PDA_Simulator (Temp_String, Temp_Stack)
        If (Found = =1) Then
        return 1
        Else
        return 0
        End If
       End
```

***Procedure: get_input (T_input_String)***

```
Begin
 return first symbol from T_input_String
End
```

***Procedure: get_Top_S (T_Stack)***

```
Begin
 return first symbol from the T_Stack
End
```

***Procedure: remove_first_input()***

```
Begin
 delete first symbol from the T_input_String
End
```

***Procedure: remove_TOP_Stack ()***

```
Begin
 delete first symbol from the T_Stack
End
```

***Procedure: copy_right(T_Stack, X)***

```
Begin
 Copies the RHS of the production X to T_Stack
End
```

***Procedure: verify_string(String str)***

```
Begin
 STK = "S$"
 STT = append "$" to str
 Result = PDA_Simulator (str, STK)
 Result (?)
 Case 0:
 Msg = "Rejected";
 Case 1:
 Msg = "Accepted"
 Case 2:
        Msg = "Stack overflow"
End
```

Computational simulation results have been shown for both the case, i.e. for the acceptance and rejection of the input string. The s*imulation result-1* represent the acceptance of the input string "()((()())()())((()))" in Table 1. The best possible CFG used for this purpose is <{S, M}, {(,)}, {S→M M→? M→(S) M}, S>.

Table 1. PDA simulation result-1 for language "balanced parenthesis" for the test string "((()())()())((()))"

| | |
|---|---|
| BC | 011101111101010101101001011001000001010000000000000100000101000110110110010101000011001001000001 000011000110000101110100100 |
| SC | C ) B ) ? )) A C A S A S S? S S (S) S? C A ? S?? ? ? S A A ? ? (A C ((C) B)?) ) A C A S A S S ? S S (S) S C A?? S?? ? ? S A A?? ( A C ( ( |

Splitting of SC
C)B)? ))ACA SASS? SS(S) S?CA? S???? SAA?? (AC(( C)B)? ))ACA SASS? SS(S) SCA?? S???? SAA?? (AC((
Removal of unwanted rules
C)B)? ))ACA SASS? SS(S) SCA?? S???? SAA?? (AC((
Rule added are: S???? SS(S)
Rule after useless removal from terminal side: S???? SS(S)
Removal of unwanted rules over n number of rules added are : 2 Original rules are ::8
Rule after useless removal from S side: S???? SS(S)
Test for recursion at: S???? SS(S)
SC M ( S ) M M ( S ) M S ? ? M ? ? M ? ? ? ? ? S ? M ? ? ? M ? ? ? ? ? S M ? ? ? ? M ? ? ? ? ?
Final rules induced are: S→M M→? M→(S)M
PDA Simulator proliferation for verification of string: ((()())()())((()))

| | | | |
|---|---|---|---|
| STRING IS | ((()())()())((()))$ | Stack is | M$ |
| STRING IS | (()())()())((()))$ | Stack is | M)M$ |
| STRING IS | ()())()())((()))$ | Stack is | M)M)M$ |
| STRING IS | ())()())((()))$ | Stack is | M)M)M$ |
| STRING IS | ()())((()))$ | Stack is | M)M$ |
| STRING IS | ())((()))$ | Stack is | M)M$ |
| STRING IS | ((()))$ | Stack is | M$ |
| STRING IS | (()))$ | Stack is | M)M$ |
| STRING IS | ()))$ | Stack is | M)M)M$ |

((()())()())((())) string is accepted

These grammars have been induced from the training data employing a GA in Figure 2. It can be seen that the input string and production rules are the input for the finite state controller, which verifies the string and production rules through proliferation via the PDA simulator.

## 4. RESULTS AND ANALYSIS

The computational experiments have been conducted. Net Beans IDE 7.0.1, Intel Core TM 2 processor (2.8 GHz) with 2 GB RAM has been used. Four different types of languages of varying patterns have been taken, are given in Table 2.

Table 2. Test Languages Description

| L-id | Descriptions | Standard Set |
|---|---|---|
| L1 | (10)* over (0 + 1)* | Tomita /Dupont Set |
| L2 | All string not containing '000' over (0+1)* | Tomita /Dupont Set |
| L3 | Balanced Parentheses | Huijsen /Keller & Lutz set |
| L4 | Odd binary number ending with 1 | Dupont set |

### 4.1. Parameters Selection and Tuning

The orthogonal array method has been used for the parameter selection. Orthogonal array is helpful in setting the well balanced experiments and Taguchi signal-to-noise ratios (SNR), which are log functions of the desired output, serve as an objective function for optimization that helps in data analysis and prediction of optimum results. Equation (2) has been used to evaluate SNR.

$$SNR_i = -10\log\left(\sum_{u=1}^{N_u}\frac{y_u^2}{N_i}\right) \quad (2)$$

Where, $i$ = experiment number, $u$ = trial number, $N_i$ = number of trials for the experiment, and $y_u$ = number generations taken in each trials to reach to the solution.

Table 3. Parameter selection by orthogonal array method

| Ex. No. | | PS | PRL | CS | CR | MR | Means | Coff. Variation | Std.dev. | SNR |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 120 | 5 | 120 | 0.6 | 0.5 | 3.56667 | 0.0428278 | 0.152753 | -11.0506 |
| 2 | | 120 | 5 | 120 | 0.9 | 0.8 | 4.06667 | 0.0375621 | 0.152753 | -12.1889 |
| 3 | | 120 | 8 | 240 | 0.6 | 0.5 | 3.26667 | 0.0467610 | 0.152753 | -10.2884 |
| 4 | | 120 | 8 | 240 | 0.9 | 0.8 | 3.56667 | 0.0901276 | 0.321455 | -11.0687 |
| 5 | | 360 | 5 | 240 | 0.6 | 0.8 | 3.50000 | 0.0285714 | 0.100000 | -10.8837 |
| 6 | | 360 | 5 | 240 | 0.9 | 0.5 | 3.59000 | 0.0460243 | 0.165227 | -11.1080 |
| 7 | | 360 | 8 | 120 | 0.6 | 0.8 | 3.30000 | 0.0606061 | 0.200000 | -10.3809 |
| 8 | | 360 | 8 | 120 | 0.9 | 0.5 | 3.50000 | 0.0494872 | 0.173205 | -10.8884 |
| Response for SNR (Small is better) | Low | -11.15 | -11.31 | -11.13 | -10.65 | -10.83 | For (PS:PRL:CS:CR:MR= 120: 5: 120: 0.6: 0.5) | | | |
| | High | -10.82 | -10.66 | -10.84 | -11.31 | -11.13 | For (PS:PRL:CS:CR:MR= 360: 8: 240: 0.9: 0.8) | | | |
| | Delta | 0.33 | 0.65 | 0.29 | 0.66 | 0.30 | Delta: difference between Low: High | | | |
| | Rank | 3 | 2 | 5 | 1 | 4 | Rank: based on delta (1: Highest and 5: Lowest) | | | |
| Suitable Combination | | 120 | 5 | 120 | 0.9 | 0.8 | SNR (smaller is better) | | | -12.1889 |

SNR: Signal to noise ratio

The performance of the GA is significantly affected by population size (PS), chromosome size (CS), crossover rate (CR) and mutation rate (MR). The performance of GA largely depends on PS, CS, CR and MR. In GI, production rule length (PRL) is also an important factor that affects the result. The orthogonal array involves five control factors with two levels: PS= [120, 360], PRL= [5, 8], CS= [120, 240], CR= [0.6, 0.9] and MR= [0.5, 0.8], where following setting gave the best results PS= 120, PRL= 5, CS= 120, CR= 0.9 and MR= 0.8 (See Table 3 experiment number 2, SNR=-12.1889) is taken for the robust process design and to conduct the experiments.

### 4.2. Performance Comparison

The author has compared the performance of the proposed approach with CGA, ROGGA [36] and crowding method [37]. The same parameters setting have been used for CGA, ROGGA and crowding in the proposed GA.

The CGA works using random initial population of fixed length chromosomes. It starts with a set of solution represented by chromosome. Using an appropriate selection technique a solution from one population is picked depending on its fitness and used to form a new offspring. This process repeated until the GA reached to the threshold or maximum number of generation. For the purpose of an evolution the CGA perform a onetime crossover and mutation operator for reproduction. Then individuals are picked depending on their fitness value to act as parents to generate offspring in the new generation.

In ROGGA, before applying reproduction, test for similarity of the genetic material has been conducted – if found similar, then generate offspring – producing a random solution otherwise apply reproduction in normal fashion.

De Jong (1975) introduced crowding approach to preserve population diversity which results in preventing premature convergence. It was applied in the survival solution step of the GA to decide which individual among those in the current population and their offspring individual will pass to the next generation. It eliminates the most similar individual whenever a new one enters in a subpopulation.

### 4.3. Results and Discussion

The experimental results show that GA is capable in CFG induction. The minimum description length (MDL) principle has been applied and found effective in generalization and specialization of the training data.

Table 4. Generated grammar with fitness value and total number of rules

| L-id | Fitness | Grammar $< V, \sum, P, S >$ |
|---|---|---|
| L1 | 1014 | <{S}, {0, 1}, {S→?, S→10S}, S> |
| L2 | 1011 | <{S,C,M}, {0, 1}, {S→CCM, M→?, M→1SM, C→?, C→0}, S> |
| L3 | 1014 | <{S}, {(, )}, {S→?, S→ (S)S}, S> |
| L4 | 1012 | <{S, M}, {0, 1}, {S→1M, S→0SM, M→SM, M→?}, S> |

Table 5. Comparison based on Generation range, threshold, time consumed, mean (μ) and standard deviation (Std. Deviation) for each language L1 through L4

| Approach | L-id | Gen. Range | Threshold | Time (in millisecond) | μ | Std.Deviation |
|---|---|---|---|---|---|---|
| GIGA | L1 | 5±2 | 7 | 38801.1 | 3.4 | 2.5 |
| | L2 | 22±11 | 26 | 3192827 | 18.2 | 6.9 |
| | L3 | 8±3 | 18 | 365851.8 | 9.1 | 5.22 |
| | L4 | 9±4 | 23 | 235360.3 | 10.8 | 5.41 |
| CGA | L1 | 6±2 | 7 | 26064.1 | 4.1 | 2.73 |
| | L2 | 14±10 | 22 | 2690711 | 17.4 | 6.43 |
| | L3 | 8±4 | 12 | 196347 | 8.2 | 2.44 |
| | L4 | 7±6 | 14 | 73749.7 | 8.1 | 3.87 |
| ROGGA | L1 | 4±2 | 7 | 56012.3 | 3.9 | 1.8 |
| | L2 | 20±12 | 24 | 3701925 | 25.7 | 10.34 |
| | L3 | 7±3 | 13 | 268276.2 | 7.2 | 2.97 |
| | L4 | 12±6 | 25 | 380766.3 | 12 | 8.01 |
| Crowding | L1 | 8±6 | 32 | 359077.3 | 8.7 | 2.7 |
| | L2 | 14±12 | 27 | 3688203 | 14.3 | 7.8 |
| | L3 | 7±6 | 24 | 258723.1 | 9.8 | 2.5 |
| | L4 | 8±6 | 15 | 80078.5 | 9.2 | 1.6 |

The training set and test set, which are required for the learning has been generated with length 'L' (L= 0, 1, 2, …..) such that it covers all the possible valid strings of the length L till the sufficient number of valid strings of corpus that has been generated. The invalid strings generated during this process are considered as negative strings. The resultant grammars are validated against the best known available grammar. The standard representation $< V, \sum, P, S >$ has been adapted to represent the grammars as presented in Table 4. The results are collected as an average of twenty runs; hence generation range has been used (see Table 5). The PDA simulator and its associated methods have been found effective in validating the strings using the resultant grammar and its equivalent proliferation on the stack. The proliferation through PDA simulator presented in Table 1, 2, and 3 are for best grammars received over twenty successful GA runs. The author has followed instantaneous descriptor "(current state, current input string, stack content)" for the proliferation purpose. The '$' symbol indicates the end of the input and bottom of the stack respectively. The author has shown best average fitness vs. generation chart for each language L1 through L4, indicates that the proposed GIGA does not showed any premature convergence. It was observed that the population has converged to the best value earlier in case of the simple grammars, whereas it has shown slow convergence for relatively complex grammar. The inverted mutation operator with random masks and then XOR operation introduces sufficient diversity in the population helps in avoiding premature convergence that is the main reason; the author has compared the algorithm with the existing GA proposed for avoiding premature convergence.

### 4.4. Statistical Analysis

A statistical test has been performed considering the hypothesis: "*there is no significant difference in the mean of samples at the 5% level of confidence*". Total 15 samples have been drawn from each algorithm for the randomly selected language. The descriptive analysis is depicted in Table 7 represents the minimum, maximum and average fitness. The main ANOVA result is represented in Table 6. The p- value 0.001<0.05 leads to rejection of null hypothesis. Therefore, multiple comparison tests: TukeyHSD test has been adapted. The results of TukeyHSD test is depicted in Table 8 indicates that the performance of the GIGA is significantly better than CGA and ROGGA since the p-value 0.002 and 0.05 is either less or equal to the 0.05, whereas the proposed GIGA with PDA simulator perform better than the crowding method.

Figure 4 graphically displays the average fitness value of each algorithm. The X-axis represents the methods and the Y-axis represents the estimated marginal average fitness. The average fitness of the GIGA is found better than other approaches. The CGA's performance has been found worst.

Table 6. ANOVA table showing group comparison results

| | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|
| Between Groups | 319382.836 | 3 | 106460.945 | 6.389 | 001 |
| Within Groups | 933149.033 | 56 | 16663.376 | | |
| Total | 1252531.870 | 59 | | | |

Table 7. Descriptive analysis

| Method | N | Mean | Std. Deviation | Std. Error | 95% Confidence Interval for Mean | | Minimum | Maximum |
|---|---|---|---|---|---|---|---|---|
| | | | | | Lower Bound | Upper Bound | | |
| CGA | 15 | 747.9867 | 138.76471 | 35.82889 | 671.1413 | 824.8320 | 540.20 | 944.50 |
| ROGGA | 15 | 759.3867 | 139.45049 | 36.00596 | 682.1616 | 836.6118 | 560.30 | 950.50 |
| Crowding | 15 | 855.9933 | 112.48940 | 29.04464 | 793.6988 | 918.2879 | 634.80 | 1003.00 |
| GIGA | 15 | 925.6267 | 123.68328 | 31.93489 | 857.1331 | 994.1202 | 615.90 | 1011.80 |
| Total | 60 | 822.2483 | 145.70296 | 18.81017 | 784.6093 | 859.8874 | 540.20 | 1011.80 |

Table 8. Multiple comparisons test (Posthoc test: Tukey HSD test)

| (I) Method | (J) Method | Mean Difference (I-J) | Std. Error | Sig. | 95% Confidence Interval | |
|---|---|---|---|---|---|---|
| | | | | | Lower Bound | Upper Bound |
| CGA | ROGGA | -11.40000 | 47.13580 | .995 | -136.2103 | 113.4103 |
| | Crowding | -108.00667 | 47.13580 | .112 | -232.8169 | 16.8036 |
| | GIGA | -177.64000* | 47.13580 | .002 | -302.4503 | -52.8297 |
| ROGGA | CGA | 11.40000 | 47.13580 | .995 | -113.4103 | 136.2103 |
| | Crowding | -96.60667 | 47.13580 | .183 | -221.4169 | 28.2036 |
| | GIGA | -166.24000* | 47.13580 | .005 | -291.0503 | -41.4297 |
| Crowding | CGA | 108.00667 | 47.13580 | .112 | -16.8036 | 232.8169 |
| | ROGGA | 96.60667 | 47.13580 | .183 | -28.2036 | 221.4169 |
| | GIGA | -69.63333 | 47.13580 | .458 | -194.4436 | 55.1769 |
| GIGA | CGA | 177.64000* | 47.13580 | .002 | 52.8297 | 302.4503 |
| | ROGGA | 166.24000* | 47.13580 | .005 | 41.4297 | 291.0503 |
| | Crowding | 69.63333 | 47.13580 | .458 | -55.1769 | 194.4436 |

*. The mean difference is significant at the 0.05 level.

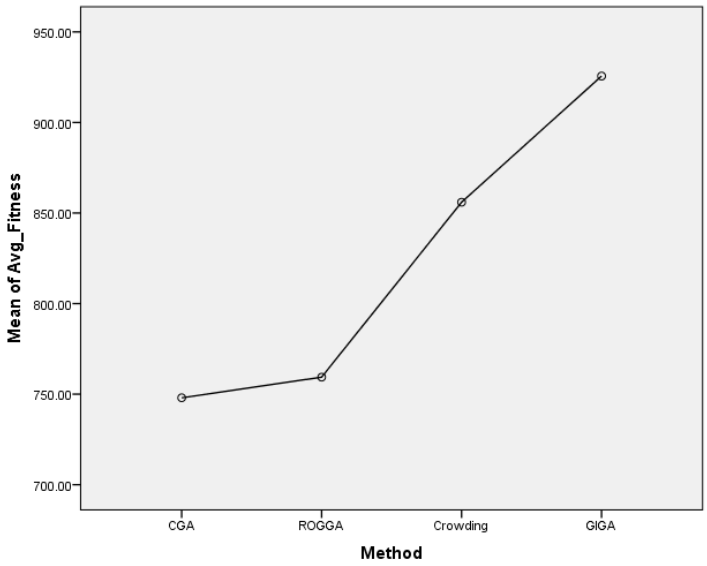


Figure 4. Means of average fitness with respect to the algorithms (methods) chart

## 5. CONCLUSION

In this paper, the GIGA has been presented for CFG induction. The results reported provide the significant evidence about the performance of the proposed algorithm. The 2-level orthogonal array and the Taguchi method's SNR have been incorporated to find the appropriate parameters' setting that introduces robustness, fast convergence and statistical soundness. The author has executed the proposed GIGA for CFLs and regular languages of varying complexities. The experimental results have been found encouraging as the proposed GIGA has been found capable in CFG induction and greatly improves the performance. The PDA simulator has been incorporated in the experiments have been found working successfully for the light weight examples, which can be improved for the larger length description of the training data set. The result received applying proposed algorithm has been compared with other approaches indicates the superiority of the GIGA. The author has performed statistical tests explains the performance significance of the proposed approach. The current results are encouraging but it would be interesting to implement the proposed approach

for more complex problems and natural languages. Therefore the next goal is to implement the proposed GIGA for more complex problems.

## REFERENCES


[1] Chandwani, M., M. Puranik, and N. S. Chaudhari. "On CKY-parsing of context-free grammars in parallel." TENCON'92. "Technology Enabling Tomorrow: Computers, Communications and Automation towards the 21st Century." *1992 IEEE Region 10 International Conference. IEEE, 1992*.

[2] Younger, Daniel H. "Recognition and parsing of context-free languages in time n" Information and control. 10.2 (1967): 189-208.

[3] Earley, Jay. "An efficient context-free parsing algorithm." Communications of the ACM 13.2 (1970): 94-102.

[4] Pullum, Geoffrey K. "Learnability, hyperlearning, and the poverty of the stimulus." *Annual Meeting of the Berkeley Linguistics Society*. Vol. 22. No. 1. 2012.

[5] De La Higuera, Colin. "A bibliographical study of grammatical inference." *Pattern recognition* 38.9 (2005): 1332-1348.

[6] De la Higuera, Colin. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[7] Sakakibara, Yasubumi. "Recent advances of grammatical inference." *Theoretical Computer Science* 185.1 (1997): 15-45.

[8] Tsoulos, Ioannis G. and Isaac E. Lagaris. "Grammar inference with grammatical evolution." (2006).

[9] Sebastian, Neetha, and Kamala Krithivasan. "Learning Algorithms for Grammars of Variable Arity Trees." *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*. IEEE, 2007.

[10] Angluin, Dana, and Carl H. Smith. "Inductive inference: Theory and methods."*ACM Computing Surveys (CSUR)* 15.3 (1983): 237-269.

[11] Fu, King Sun. *Syntactic pattern recognition and applications*. Vol. 4. Englewood Cliffs: Prentice-Hall, 1982.

[12] Harrison, Michael A. *Introduction to formal language theory*. Addison-Wesley Longman Publishing Co., Inc., 1978.

[13] Lang, Kevin J. "Random DFA's can be approximately learned from sparse uniform examples." *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992.

[14] Oliveira, Arlindo L., ed. Grammatical Inference: Algorithms and Applications: 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11-13, 2000 Proceedings. No. 1891. Springer, 2000.

[15] Tomita, Masaru. Efficient parsing for natural language: A fast algorithm for practical systems. Vol. 8. Springer Science & Business Media, 2013.

[16] Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman.: *Compilers, Principles, Techniques*. Addison wesley, 1986.

[17] Hopcroft, John E. Introduction to automata theory, languages, and computation. Pearson Education India, 1979.

[18] Gold, E. Mark. "Language identification in the limit." *Information and control*10.5 (1967): 447-474.

[19] Angluin, Dana. "Inductive inference of formal languages from positive data."*Information and control* 45.2 (1980): 117-135.

[20] Bunke, Horst, and Alberto Sanfeliu, eds. *Syntactic and structural pattern recognition: theory and applications*. Vol. 7. World Scientific, 1990.

[21] Angluin, Dana. "Queries and concept learning." *Machine learning* 2.4 (1988): 319-342.

[22] Valiant, Leslie G. "A theory of the learnable." *Communications of the ACM*27.11 (1984): 1134-1142.

[23] Li, Ming, and Paul MB Vitányi. "Learning simple concepts under simple distributions." *SIAM Journal on Computing* 20.5 (1991): 911-935.

[24] Wyard, Peter. "Representational issues for context free grammar induction using genetic algorithms." *Grammatical Inference and Applications*. Springer Berlin Heidelberg, 1994. 222-235.

[25] Theeramunkongy, Thanaruk, and Manabu Okumuray. "Grammar acquisition and statistical parsing by exploiting Local Contextual Information." *Journal of Natural Language Processing Vol* 2.3 (1995).

[26] Javed, Faizan, et al. "Context-free grammar induction using genetic programming." *Proceedings of the 42nd annual southeast regional conference*. ACM, 2004.

[27] Pandey, Hari Mohan, Anurag Dixit, and Deepti Mehrotra. "Genetic algorithms: concepts, issues and a case study of grammar induction." *Proceedings of the CUBE International Information Technology Conference*. ACM, 2012.

[28] Choubey, N. S., and M. U. Kharat. "Sequential structuring element for CFG induction using genetic algorithm." *International Journal of Futuristic Computer Application* 1 (2010).

[29] Choubey, Nitin Surajkishor, Hari Mohan Pandey, and M. U. Kharat. "Developing Genetic Algorithm Library Using Java for CFG Induction." *International Journal of Advancements in Technology* 2.1 (2011): 117-128.

[30] Pandey, Hari Mohan. "Context free grammar induction library using Genetic Algorithms." *Computer and Communication Technology (ICCCT), 2010 International Conference on*. IEEE, 2010.

[31] Hrncic, Dejan, and Marjan Mernik. "Memetic grammatical inference approach for DSL embedding." *MIPRO, 2011 Proceedings of the 34th International Convention*. IEEE, 2011.

[32] Hrncic, Dejan, Marjan Mernik, and Barrett R. Bryant. "Improving Grammar Inference by a Memetic Algorithm." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 42.5 (2012): 692-703.

[33] Stevenson, Andrew, and James R. Cordy. "Grammatical inference in software engineering: An overview of the state of the art." *Software Language Engineering*. Springer Berlin Heidelberg, 2013. 204-223.

[34] Stevenson, Andrew, and James R. Cordy. "A Survey of Grammatical Inference in Software Engineering." *Science of Computer Programming* (2014).
[35] Coste, Alexander Clark François, and Laurent Miclet. "Grammatical Inference: Algorithms and Applications." (2008).
[36] Rocha, Miguel, and José Neves. "Preventing premature convergence to local optima in genetic algorithms via random offspring generation." *Multiple Approaches to Intelligent Systems*. Springer Berlin Heidelberg, 1999. 127-136.
[37] De Jong, Kenneth Alan. "Analysis of the behavior of a class of genetic adaptive systems." (1975).
[38] John Henry Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, *control, and artificial intelligence*. MIT press, 1992.

## BIOGRAPHY OF AUTHOR

Hari Mohan Pandey did M.Tech in Computer Engineering from Mukesh Patel School of Technology Management & Engineering, NMIMS University, Mumbai. He is perusing Ph.D. in Computer Science & Engineering. He has published research papers in various journals. He has written many books in the field of Computer Science & Engineering for McGraw-Hill, Pearson Education, and University Science Press. He is associated with various International Journals as reviewer and editorial board member. His area of interest Machine Learning Computer, Artificial Intelligence, Soft Computing etc.