Applied Soft Computing 38 (2016) 453–468

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Grammar induction using bit masking oriented genetic algorithm and comparative analysis

Hari Mohan Pandey[a,*], Ankit Chaudhary[b], Deepti Mehrotra[c]

[a] *Department of Computer Science & Engineering, Amity University, Uttar Pradesh, India*
[b] *Department of Computer Science, Truman State University, USA*
[c] *Amity School of Engineering & Technology, Amity University, Uttar Pradesh, India*

ARTICLE INFO

Article history:
Received 2 May 2014
Received in revised form
24 September 2015
Accepted 24 September 2015
Available online 17 October 2015

Keywords:
Bit-masking oriented data structure
Context free grammar
Genetic algorithm
Grammar induction mask-fill operator
Premature convergence

ABSTRACT

This paper presents bit masking oriented genetic algorithm (BMOGA) for context free grammar induction. It takes the advantages of crossover and mutation mask-fill operators together with a Boolean based procedure in two phases to guide the search process from ith generation to $(i+1)$th generation. Crossover and mutation mask-fill operations are performed to generate the proportionate amount of population in each generation. A parser has been implemented checks the validity of the grammar rules based on the acceptance or rejection of training data on the positive and negative strings of the language. Experiments are conducted on collection of context free and regular languages. Minimum description length principle has been used to generate a corpus of positive and negative samples as appropriate for the experiment. It was observed that the BMOGA produces successive generations of individuals, computes their fitness at each step and chooses the best when reached to threshold (termination) condition. As presented approach was found effective in handling premature convergence therefore results are compared with the approaches used to alleviate premature convergence. The analysis showed that the BMOGA performs better as compared to other algorithms such as: random offspring generation approach, dynamic allocation of reproduction operators, elite mating pool approach and the simple genetic algorithm. The term success ratio is used as a quality measure and its value shows the effectiveness of the BMOGA. Statistical tests indicate superiority of the BMOGA over other existing approaches implemented.

1. Introduction

Grammar induction or grammar learning deals with idealized learning procedures for acquiring grammars on the basis of the evidence about the languages [45,62,63]. It was extensively studied [6,46–50,63] due to its wide fields of application to solve practical problems in a variety of different fields, which includes a compilation and translation, human machine interaction, graphic languages, design of programming language, data mining, computational biology, natural language processing, software engineering and machine learning, etc.

The first learning model was proposed by Gold in 1967 [26]. Gold addressed the question "*Is the information sufficient to determine which of the possible languages is the unknown language*?" [26]. It was shown that an inference algorithm can identify an unknown language in the limit from complete information in a finite number of steps. The key issue with the Gold's approach is that there is no information present with inference algorithm about identification of correct grammar because it is always possible that next sample may invalidate the previous hypothesis. Angluin [58] proposed "*tell tales*" (a unique string makes the difference between languages) to avoid the drawback. Although Gold [26] laid the foundation of grammar inference, Bunke and Sanfeliu [38] presented the first usable grammatical inference algorithm in syntactic pattern recognition community with the aim of classification and analysis of patterns, classification of biological sequence, character recognition, etc. The main drawback of this algorithm was it only dealt with positive data, was unable to deal with noisy data, did not fit exactly into a finite state machine and therefore good formal language theories were lost. Stevenson and Cordy [39,40] explain that theorists and empiricists are the two main groups contributing in the field of grammar inference. Language classes and learning models were considered by theorists group to set up the boundaries of what is learnable and how efficiently it can be learned. On the other hand empiricists group dealt with a practical problem by solving it; finally they have made a significant contribution

* Corresponding author. Tel.: +91 9810625304.
E-mail address: profharimohanpandey@gmail.com (H.M. Pandey).

http://dx.doi.org/10.1016/j.asoc.2015.09.044
1568-4946/© 2015 Elsevier B.V. All rights reserved.

in grammatical inference. Teacher and query is another learning model, where a teacher also referred to as an Oracle knows the target languages and is capable to answer particular types of questions/queries from the inference algorithm. Six types of queries were described by Angluin [59], two of which are membership and equivalence queries that have significant impact on learning. In case of membership queries, the inference algorithm presents either "*yes*" or "*no*" as answer to the oracle, whereas oracle receives "*yes*" if the hypothesis is true and "*no*" otherwise by inference algorithm. Valiant [60] presented *Probably Approximately Correct* (PAC) Learning Model, which takes the advantages of both identification of the limit and *teachers and queries* learning models. The PAC learning model is different from other two former learning models because of two reasons: first it does not guarantee exact identification with certainty, second compromises between accuracy and certainty. The problem with the PAC model is that the inference algorithm must learn in polynomial time under all distributions, but it is believed to be too strict in reality. These problems occur because many apparently simple classes are either known to be NP-hard or at least not known to be polynomial learnable for all distributions [39]. To mitigate this issue, Li et al. [61] proposed inference algorithm to consider the simple distribution only. Apart from the above popular learning models, many researchers explain the suitability of the neural network for grammar inference problem. Neural network shown the ability to maintain a temporal internal state like a short term memory [40]. In case of the neural network, a set of inputs, and their corresponding outputs (Yes: string is in the target language, No: otherwise) and a defined function need to learn, which describes those input-output pairs [40]. Alex et al. [54] conducted experiments for handwriting recognition using neural network and it was explained that this network has the capability to predict subsequent elements from an input sequence of elements. Cleeremans et al. [53] implemented a special case of a recurrent network presented by Elman [55] known a simple recurrent network to approximate a DFA (deterministic finite automata). Delgado and Pegalajar [56] presented a multi-objective genetic algorithm to analyze the optimal size of a recurrent neural network to learn positive and negative examples. Authors of [56] utilized the merits of self organizing map to determine the automation, once the training is completed. Although neural network is widely used for grammar inference since it was found good at simulating an unknown function, but it was observed that there is no way to reconstruct the function from the connections in a trained network [40]. A detailed survey of various grammar inference algorithms is presented in [6,39,40,51,52,57]. Inductive inference is the process of making generalization from the input (string). Wyard [3] presented the impact of different grammar representation and experimental result show that the evolutionary algorithm (EA) using standard context free grammar (CFG) (Backus Naur Form (BNF)) outperformed others. Thanaruk and Okumaru [27] classified grammar induction methods into three categories, namely; supervised, semi-supervised and unsupervised depending on the type of required data. Javed et al. [28] presented genetic programming (GP) based approach to learn CFG. The work presented in [28] was the extension of the work done in [3] by utilizing grammar specific heuristic operator. In addition, better construction of the initial population was suggested. Choubey and Kharat [29] presented a sequential structuring approach to perform coding and decoding of binary coded chromosomes into terminal and non-terminals and vice versa. A CFG induction library was presented using the GA contains various Java classes to perform grammar inference process [30,9]. The proposed approach for grammar induction is discussed in Section 2.

Section 3 discusses an important issue in GA known as premature convergence. In evolutionary search, the diversity (difference among individual at the genotype or phenotype levels) decreases as the population converges to a local optimum. It is the situation when an extraordinary individual takes over a significant proportion of finite population and leads towards an undesirable convergence. Diverse population is a prerequisite for exploration in order to avoid premature convergence to local optima [44,64]. On the other hand, promoting diversity in an evolutionary process is good where exploitation is needed. Hence, to reach to the global solution and explore the search space adequately, maintaining population diversity is very important. Also, it had been explained clearly that degree of population diversity is one of the main cause of premature convergence [42,43]. It is necessary to select the best solution of the current generation to direct the GA to reach to the global optimum. The tendency to select the best member of the current generation is known as selective pressure. It is also a key factor that plays an important role in maintaining genetic diversity. A proper balancing is required between genetic diversity and selection pressure to direct the GA search to converge in a time effective manner and to achieve global optima. High selective pressure reduces the genetic diversity; hence in this situation premature convergence can occur while the little selective pressure prohibits the GA to converge to an optimum in reasonable time. An approach to increase the population size is not sufficient to alleviate premature convergence because any increase in population size will add twofold cost both extra computing time and number of generations to converge on global optima. Applying mutation alone converts GA search into a random search whereas crossover alone generates a sub-optimal solution. Also, applying selection, crossover and mutation together may result the GA search to noise tolerant hill climbing approach. Several approaches to handle the premature convergence are suggested [41]. Although these approaches address the premature convergence in their ways by maintaining population diversity, these methods suffer with its own strengths and weaknesses. Pandey et al. [41] presented a detailed and comprehensive comparison of various premature convergence handling approaches on the basis of their merits, demerits and other important factors.

The focus of this paper is towards development of a grammar induction tool and addressing premature convergence in GA. An effort is made towards utilizing the applicability of the bit masking oriented data structures to apply mask-fill reproduction operators and the Boolean based procedure. The diversity of the population is created through the Boolean based procedure during offspring generation, which helps in avoiding premature convergence in a grammar inference problem using the GA. A learning system uses a finite set of examples and to train the system sufficient training set is needed, which can be achieved employing generalization and specialization. Authors [65–67] discussed the importance of generalization and specialization for the learning systems. It was discussed that overgeneralization is a major issue in the identification of grammars for formal languages from the positive data [40] [65]. Therefore, the minimum description length principle is used to address this issue (discussed in Section 2.4). This paper makes five main contributions, as enumerated below:

(a). The four steps of grammar induction are presented. An example is considered to illustrate the step by step process of grammar induction.
(b). A sequential mapping of chromosome is presented. In addition, the applicability of the parser is shown helps in removing insufficiencies and decide acceptance and rejection of training data.
(c). An algorithm "*BMOGA*" is presented for CFG induction from the positive and negative corpus. Furthermore, we show the crossover and mutation mask-fill reproduction operators and detailed procedure of the new offspring generation.

(d). Applicability of the minimum description length principle is shown in CFG induction.
(e). We bring to light the effectiveness of the proposed algorithm by comparing it with existing algorithms such as a simple genetic algorithm (SGA), random offspring generation genetic algorithm (ROGGA), elite mating pool genetic algorithm (EMPGA) and dynamic allocation of reproduction operators (DARO).

The simulation model for grammar induction is discussed in Section 4 followed by concluding remarks for the paper in Section 5. Lastly but not the least important literatures for grammar inference and approaches to alleviate premature convergence is presented in “References”.

2. Proposed approach for grammar induction

Earlier work conducted shows the relatively simple and deterministic CFG has been inferred [3,9,10] using the GA’s. A grammar induction library for CFG induction is presented [9,30], which consists of a four Java classes and dealt only with simple languages. A case study on grammar induction is presented in [10], which utilizes the basics of the GA that uses simple crossover (one and two points) and mutation (bit inversion) operators for grammar induction. This section presents a grammar induction algorithm, which takes the advantages of the bit-masking oriented data structure and the GA known as bit mask oriented genetic algorithm (BMOGA) for CFG induction. Fig. 1 shows the block diagram of the implemented GA. From the Fig. 1 it can be seen that the BMOGA implemented for the experiment, forms the sub-section of the intermediate population by applying crossover and mutation using the mask-fill operations (separately for crossover and mutation) (Phase-1) and a *procedure* (*P1, P2,crossover, mutation*) (Phase-2) and then merges them with the original population to get populated in the next generation. The reproduction operators adapted for the purpose of experimentation are explained in Section 2.2 whereas Section 2.1 talks about the bit masking oriented data structure and its representations. The process of grammar induction can be broken down into four steps:

(a) *Mapping:* Mapping binary chromosome into terminals and non-terminals.
(b) *Representation in Backus Naur Form (BNF)*: Apply biasing on the symbolic chromosome to get non-terminal on the left hand side.
(c) *Production rules generation*: Divide the symbolic chromosome to produce the production rules up to the desired length.
(d) *Resolve the insufficiency*: Addressing the issues such as unit production, left recursion, left factoring, multiple production, useless production, ambiguity [7,8], etc.

A parser has been implemented for the validity of the CFG rules based on the acceptance or rejection of the rules. An individual is revised in each generation by testing the ability of every chromosome to parse the objective sentence [9].

Example 1: A case of palindrome over $(0+1)^*$ is given to demonstrate the grammar induction process (Fig. 1). The isomorphic mapping of binary chromosome into terminals and non-terminals is shown. The coding mechanism, i.e. 3-bit/4-bit have been used which depends on the number of symbols used in the language. In the present scenario 3-bit mapping scheme has been applied where symbol ‘S’ represents the start symbol and mapped at “000” and ‘?’ shows the null symbol, mapped at “010” and “110”. Encoding procedure of the grammar maps the binary chromosome into terminals and non-terminals of symbolic chromosome in a sequential manner. The symbolic representation contains the block size of five, equal to the production rule length (PRL = 5) chosen for the experiment. Symbolic grammar is traced from start symbol ‘S’ to terminal to remove useless productions. The remaining production rules are tested for removal of left recursion, unit production, ambiguity and left factor. At this stage, string to be tested from the selected sample set is passed for the validity (acceptability of the CFG rules equivalent to the chromosome). The test string and CFG rules are the input to the finite state controller, which verifies the acceptability through proliferation on the push down automata.

```
*******Isomorphic Mapping (Algorithm-4: Step-2) *******
Binary Chromosome:
0001000100000100100001010011110001010001100100000100
1110101100100001100100111010101000110000010001011011
0000001101101110
Mapping Scheme:
Symbol: = S        Mapped at: = 000
Symbol: = A        Mapped at: = 001
Symbol: = B        Mapped at: = 111
Symbol: = C        Mapped at: = 011
Symbol: = 1        Mapped at: = 100
Symbol: = 0        Mapped at: = 101
Symbol: = ?        Mapped at: = 010
Symbol: = ?        Mapped at: = 110
Binary chromosome representation:
 000  S  100  1  010  ?  000  S  010  ?  010  ?
 101  0  001  A  111  B  000  S  101  0  000  S
 110  ?  010  ?  000  S  010  ?  011  C  101  0
 011  C  001  A  000  S  011  C  001  A  001  A
 110  ?  101  0  010  ?  001  A  100  1  000  S
 100  1  010  ?  110  ?  110  ?  000  S  001  A
 101  0  101  0  110  ?
Symbolic Chromosome Mapping:
S1?S??S0ABS0S??S?C0CASCAA?0?A1S1???SA00?
Splitting the symbolic chromosome (block size equal to
production rule length (PRL = 5)
      S1?S?          ?S0AB          S0S??          S?C0C

     ASCAA           ?0?A1          S1???          SA00?
Simplifying the symbolic chromosome (epsilon removal)
S1S??   S0AB?   S0S??   SC0C?   ASCAA    0A1??    S1???

SA00?   S1S??   S0AB?   S0S??    SC0C?    ASCAA   0A1??

S1???
Total symbolic chromosome = 15
Splitting the symbolic chromosome (unwanted rules removal)
      S1S??          S0AB?          S0S??          SC0C?

     ASCAA           0A1??          S1???          SA00?
Total original rules are: = 8
Total rules after removal of unwanted rules: = 3
Updated symbolic chromosome: S1???S1S??S0S??
BNF form for the updated symbolic chromosome
S→1???    S→1S??     S→0S??
After Useless Production: = S1???S1S??S0S??
String after rule shift & Useless removal:
S->1???    S->1S??   S->0S??
String after left recursion removal:
 S->1???    S->1S??    S->0S??
Final Rules are: = S->1L     S->0S    L->S     L->?
The total number of rules constructed: = 4
Testing rules for string: 10101
Stack Processing
STRING: = $          Stack Item: =S$     Level: = 13
STRING: = $          Stack Item: =L$     Level: = 12
STRING: = $          Stack Item: =L$     Level: = 11
STRING: = 1$         Stack Item: =S$     Level: = 10
STRING: = 1$         Stack Item: =S$     Level: = 9
STRING: = 01$        Stack Item: =S$     Level: = 8
STRING: = 01$        Stack Item: =L$     Level: = 7
STRING: = 01$        Stack Item: =L$     Level: = 6
STRING: = 101$       Stack Item: =S$     Level: = 5
STRING: = 101$       Stack Item: =S$     Level: = 4
STRING: = 0101$      Stack Item: =S$     Level: = 3
STRING: = 0101$      Stack Item: =L$     Level: = 2
STRING: = 0101$      Stack Item: =L$     Level: = 1
STRING: = 10101$     Stack Item: =S$     Level: = 0
10101: string is accepted
```

‘?’ Represent the null, i.e. *epsilon*.

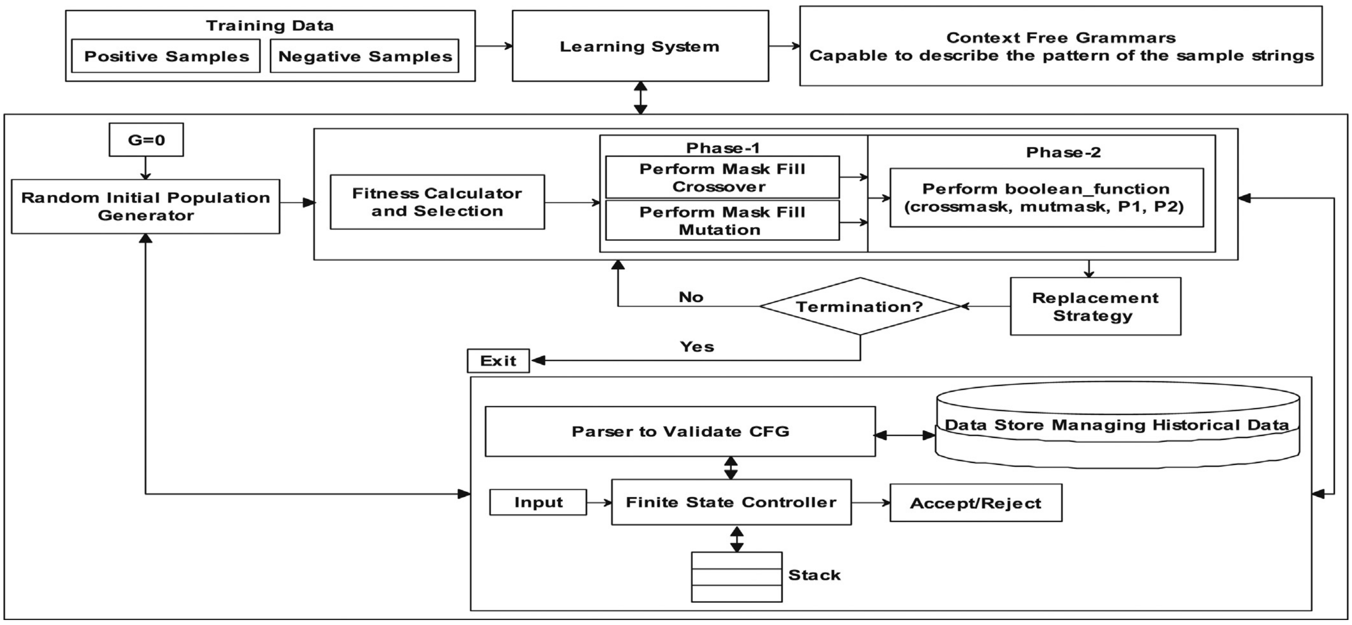

Training Data
Positive Samples
Negative Samples
Learning System
Context Free Grammars
Capable to describe the pattern of the sample strings
G=0
Random Initial Population Generator
Fitness Calculator and Selection
Phase-1
Perform Mask Fill Crossover
Perform Mask Fill Mutation
Phase-2
Perform boolean_function (crossmask, mutmask, P1, P2)
No
Termination?
Replacement Strategy
Yes
Exit
Parser to Validate CFG
Data Store Managing Historical Data
Input
Finite State Controller
Accept/Reject
Stack

Fig. 1. Block diagram for context free grammar induction using bit-masking data structure and Boolean based procedure in the reproductive phase within a genetic algorithm for the new generation.

In evolutionary algorithms (EAs), an individual can only survive depending on its fitness value [1,2]. In case of grammar induction, the fitness value of an individual chromosome largely depends on acceptance and rejection of the positive and negative corpus. It increases for accepting each positive sample (APS) and rejecting each negative sample (RNS) while decreases for the accepting each negative sample (ANS) and rejecting each positive sample (RPS). The number of rules (NPR) also influences the fitness value. As discussed the number of input strings accepted or rejected affects the fitness of an individual chromosome. Therefore, the fitness of an individual chromosome is calculated using Eq. (1), which then map to the grammars in the proposed BMOGA:

$$Fitness = \sum C * ((APS + RNS) + (ANS + RPS)) + (2 * C - NPR) \quad (1)$$

S.T.

APS + RNS ≤ Number of positive samples in corpus data
ANS + RPS ≤ Number of negative samples in corpus data
NPR: number of grammar rules
C: constant

The fitness calculation, for example-1: The chromosome size = 120 is taken which derives maximum 8 rules. The value of APS = RNS = 25 because total 25 each positive and negative sample string are considered for the simulation. In order to achieve the maximum fitness value, ANS = RPS = 0 is assumed, i.e. the proposed system is not accepting any negative sample and not rejecting any positive sample. NPR = 4, is the number of grammar rules. *C* is a constant (*C* = 10), is taken, so that grammar with less rules should have high fitness value. Putting these values in Eq. (1), we get Fitness = 516. This is the initial fitness value; it increases in a generative manner. The noticeable thing is: any increase in the value of *C*, would lead to high value of fitness by that factor. But according to the chromosome size, only 8 (=120/15) grammar rules can be extracted. Further, substitution/break, for removal of left recursion and other pre-processing, leads to at most of additional 4–5 rules approximately. Therefore, *C* = 10 (i.e. 2**C* = 20) is assumed, which differentiate between various grammar based on the number of rules. As discussed, increasing *C* will produce high fitness value, but it will be just for the sake of increasing the fitness value and not for representing the difference between various grammars. Hence, if the chromosome size is increased to produce more rules, higher value of *C* may be taken, but there is no need of doing this because by setting *C* = 10, the same task can be done satisfactorily.

2.1. Bit masking oriented data structure

Iuspa and Francesco [12] presented bit masking oriented data structure to enhance the GA search. The masked data structure was suggested to improve the implementation of crossover and mutation operator as it replaces various algorithms and codify specialized rules of mating. Working of standard procedural method and bit-masking approach was given based on vector function and relative arguments. Compared to standard procedure, bit-masking provides the facility of formal separation between the searching for the proper bit composition and effective string achievement of offspring individuals. As suggested in previous research, binary code based GAs can be grouped into explicit and implicit binary formulation [13]. But using bit-masking approach, there is no need to use explicit data structure. The reason behind this is: only high level operations, working on integer values mapped into a discrete representation domain are executed [12]. To construct the data structure of this type 2-D integer array is used which performs the crossover and mutation operations via two integer arrays known as *crossmask* (CM) and *mutmask* (MM).

It can be seen from Fig. 2 that an integer genome array has been defined to maintain the crossover and mutation operation, where set of integer values are linked with the design variables. Binary images are used to represent the masks and it is used to obtain the crossover and mutation masks. Following convention have been made to represent a binary image: high value, i.e. one or true for the current image bit is pointer to P1 while low value i.e. zero or false is a pointer to P2. Similarly, for mutation mask an integer sequence has been used that indicates its binary image using the following convention: "*if the pointed bit of the target string has to be inverted* (i.e. *high value*) *or not* (i.e. *low value*)".

To get the generic child individual there is a need of a vector function. The implementation of bit-masking data structure for

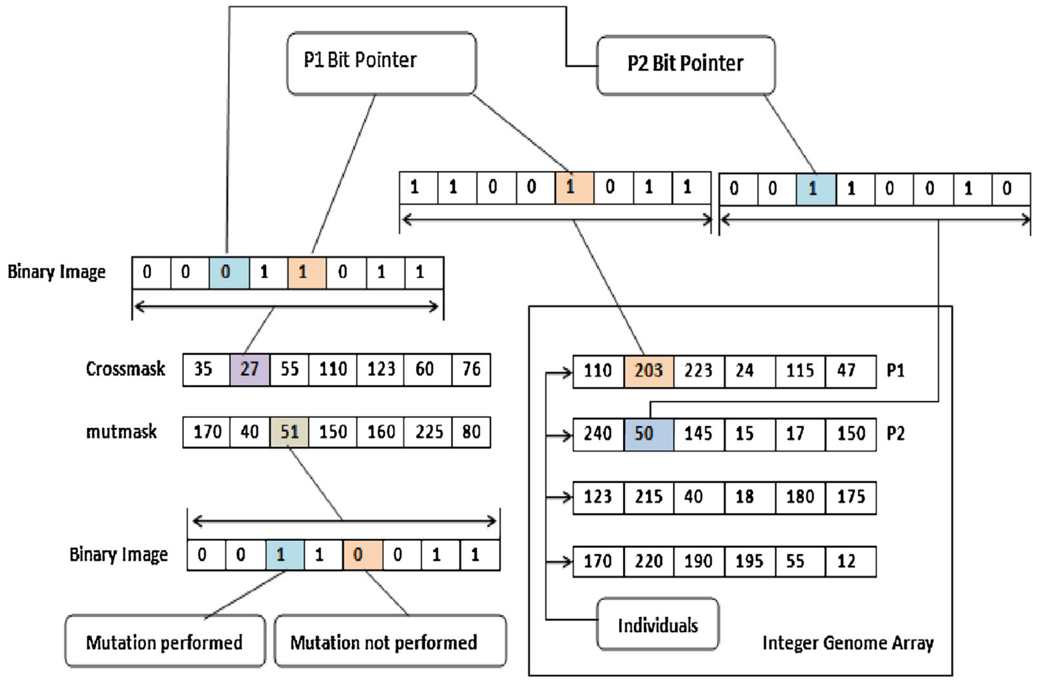


Fig. 2. Bit-masking oriented data structure with a crossover and a mutation mask with GA search (P1 and P2 are parent1 and parent2 respectively).

any real life problem or industrial problems is a two-step process: first apply crossover and mutation mask-fill and then apply mask application (see Sections 2.2 and 2.3) on selected parent strings.

2.2. Reproduction operators: crossover and mutation mask fill

CM and MM have been applied to perform the mask fill operations. Three basic optimization procedures are suggested to perform mask-fill for crossover namely: *single cut*, *bit-by-bit* and *local cut* crossover [12].

```
Algorithm-1: cutcrossover (P1, P2)
Ndv: Number of design variables, L: String length, R: Keep bit resolution
related to each design variable, C: a random cut value ranging from 1 to L-1,
RND: Random, Sum: Binary string partial length.
S1.    Apply cut randomly on (1 to L-1)
S2.    Initially Set Sum = 0
S3.    For I = 1 to Ndv do
S4.      Set Sum = Sum + R (I)
S5.      If Sum < C then
S6.        Set Mask (I) = -1
S7.      Else
S8.        Set J = I
S9.        If J ≠ Ndv then
S10.         For K = J + 1 to Ndv
S11.         Set Mask (K) = 0
S12.       Else
S13.         Set I = Ndv
S14.   End for
S15.   Set Mask (J) = $2^{R(J)} - 2^{Sum-C}$
```

Fig. 3 depicts the working of *cutcrossover* (P1, P2). The string length chosen for the example is 60. The genetic heritage from parent-to-child strings can be maintained through high/low bit strings. The value −1 (minus one) has been filled in the first section of CM array which indicate a first parent gene preemption. Under 2's complement codification, −1 indicates the whole binary string of the current mask array element set to high [12]. The second section is filled with value 0, indicates that genetic materials are coming from the second parent string, whereas the third section represent the P1 to P2 bit transition and to fill this part Eq. (2) has been used:

$$CM_{tr}^{C} = 2^{R_{tr}} - 2^{Sum(tr)-C} \text{ with } tr \in [1 \div \text{Ndv}] \tag{2}$$

where *tr* represents P1 to P2 *bit transition*.

```
Algorithm-2: bitbybit (P1, P2)
S1.    For I = 1 to Ndv do
S2.      Set Mask (I) = INT (RND * R (I)) – 1
S3.    End for
Algorithm-3: localcut (P1, P2)
S1.    For I = 1 to Ndv do
S2.        Set C = RND [1 ÷ R(I) – 1]
S3.        Set Mask (I) = $2^{R(I)} - 2^{C}$
S4.    End for
```

Similar to Eq. (2), P1 to P2 bit transition is given for Algorithm-2 and Algorithm-3 [12]. The key benefit of applying this mechanism is no extra operation is needed for bit manipulation to set the appropriate integer value according to the associate resolution. However, the redundant bit set to high for the selected bit resolution for the current design variable is ignored via specific Boolean function based technique applied to assembly of child individuals.

On the other hand, the mutation mask-fill operation is similar to the classical random binary inversion driven, depending on the specific mutation rate. Eq. (3) is used to perform mutation mask [12]:

$$mutmask_i = \sum_{j=0}^{R_j-1} 2^j.\delta_{lm} \quad i = 1, \text{Ndv} \tag{3}$$

where $\delta_{lm} = \begin{cases} l = m & if \quad rand < MutRate \\ l \neq m & \quad otherwise \end{cases}$ where, δ_{lm} is the Kronecker operator.

Kronecker product is a special operator used in matrix algebra. It gives the possibility to get a composite matrix of elements of any pair of matrices [31] [32]. The merit of Kronecker product is that it works without the assumption on the size of composing matrices, so it is well suited for large data. Equation (3) clearly explains that if the random value *rand* is less than the mutation rate *MutRate* then

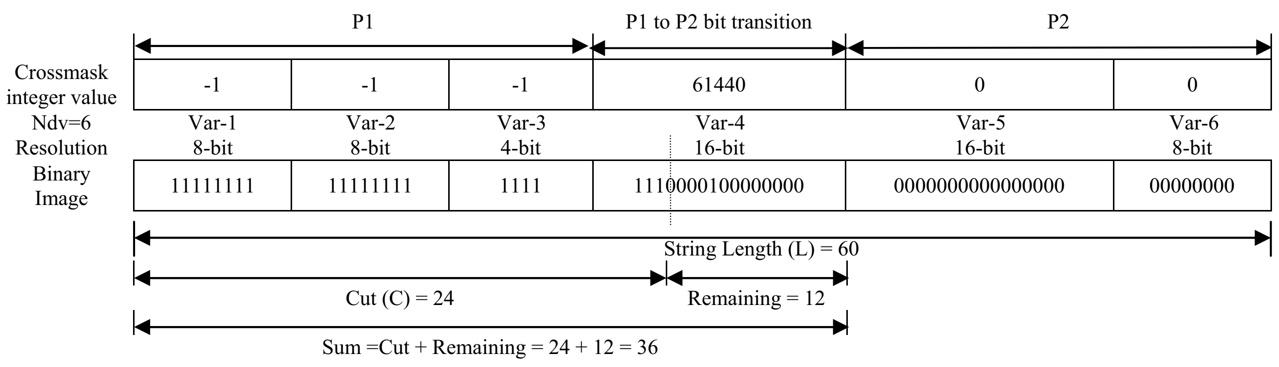


Fig. 3. An example presented to show the working of an Algorithm-1: cutcrossover (P1, P2).

apply the binary inversion otherwise there is no need of applying the mutation mask fill.

2.3. New offspring generation

The new offspring generation process starts once the crossover and mutation masks are filled in with the proper values to excel the chosen crossover and assigned mutation rate. The new offspring is generated using special Boolean based operators, which utilize parent string, CM, and MM. A couple of parent string must be chosen using an appropriate selection technique [11]. We have used *roulette wheel* selection technique for the proposed approach. Two complementary child vectors, as to crossover operator are generated, as given in Eq. (4).

$$\begin{aligned} off_1 &= f_1(P_1, P_2, CM, MM) \\ off_2 &= f_2(P_1, P_2, CM, MM) \end{aligned} \tag{4}$$

where off_j, P_j and $f_j (j = 1,2)$ are respectively the offspring, parent vectors and a Boolean function, which determines assembly style of new individuals. The arguments CM and MM are used to find the suitable crossover and mutation rules.

For the sake of simplicity, i.e. to understand crossover and mutation operation effectively, Eq. (4) can be converted into a new form to show both crossover and mutation operation separately.

Eq. (5) represents the crossover vector, a binary image, which allows P_1 or P_2 to child bit transfer as per the correlated CM value:

$$\begin{aligned} off_1 &= (P_1\ AND\ CM)\ OR\ (P_2\ AND\ (NOT\ CM)) \\ off_2 &= (P_2\ AND\ CM)\ OR\ (P_1\ AND\ (NOT\ CM)) \end{aligned} \tag{5}$$

Eq. (6) expresses the mutation section of Eq. (4), under the situation that a single MM vector of both child strings is set.

$$off_j = off_i \quad XOR \quad MM \tag{6}$$

The step by step mechanism of generating the new offspring is depicted in Fig. 4, whereas Fig. 5 demonstrates the genetic reproduction (application of Eqs. (5) and (6)) with the help of an example. The interesting thing to note that, as P1, P2, CM and MM vectors are considered as an argument of the function (f1 and f2); new individual has no strict correlation with a specific type of crossover scheme or parent pairs, as happens in an explicit binary formulation depending on the algorithm. In some situation, if the evolutionary strategies needed some couples for an identical crossover such as bit-by-bit crossover with a constant random seed, the only operation to perform and fill the mask properly, apply Eq. (5) multiple times, changing the selected parent pairs only. The Boolean based procedure introduces diversity during reproduction operations, which helps in maintaining the diversity in population and therefore avoids the premature convergence.

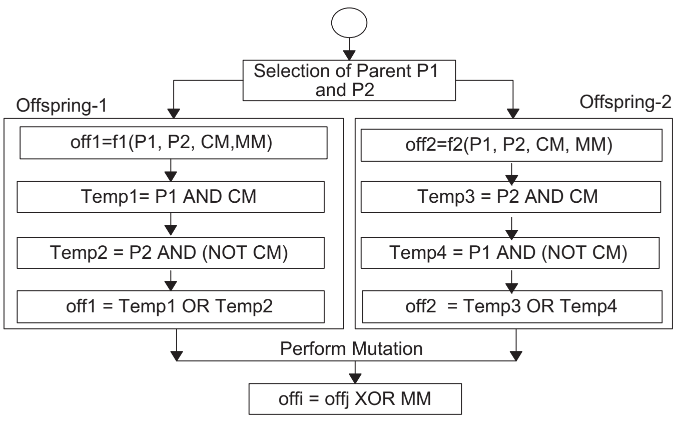


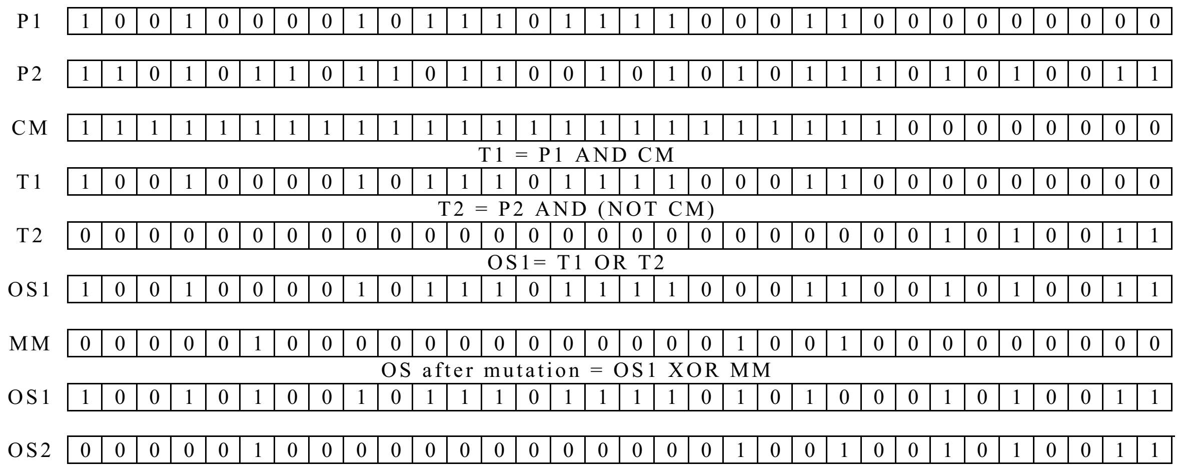
Fig. 4. New offspring generation process using crossover, mutation mask and Boolean operators.

2.4. Minimum description length principle

The problem with inductive and statistical inference systems is: How to take decision for selecting an appropriate model that should present the competing explanation of data using limited observations? An envision where a sender who wants to transmit some data to the receiver and therefore they want to select a best model which can maximally compress the observed data by which data can be delivered to the receiver using as few bits as possible.

Formally, the selection of the best model is the process to decide among model classes based on the data. The Principle of Parsimony (Occam's razor) is the soul of model selection, states that given a choice of theories, the simplest is preferable [4,5]. In other words, the purpose to implement Parsimony Principle is to find out a model which can fit well to the data. Rissanen (1978) extracted the essence of Occam's theory and presented the Principle of Minimum Description Length (MDL): "*choose the model that gives the shortest description of data*" [4,14].

For any set of corpus, one can construct a grammar without using the MDL that does not reflect regularities in data as depicted in Fig. 6(a). In such situation, constructed grammar can be considered as very simple grammar since it simply shows the validity of any combination of words. In this case, grammars do not show any regularity, hence high amount of information needed to specify them. At the opposite, one can construct grammars which can list all possible sentences/corpus but not for all sentences since grammars are not capable in terms of generalization (see Fig. 6(a)). Although this type of grammar shows some sort of regularity, but fails to present a generalization since it contains the information about each observed corpus therefore always shows poor performance and assumed to be very complex. On the other hand, construction



P1	1	0	0	1	0	0	0	0	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0
P2	1	1	0	1	0	1	1	0	1	1	0	1	1	0	0	1	0	1	0	1	0	1	1	1	0	1	0	1	0	0	1	1
CM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	T1 = P1 AND CM																															
T1	1	0	0	1	0	0	0	0	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0
	T2 = P2 AND (NOT CM)																															
T2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1
	OS1= T1 OR T2																															
OS1	1	0	0	1	0	0	0	0	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1	0	0	1	0	1	0	0	1	1
MM	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
	OS after mutation = OS1 XOR MM																															
OS1	1	0	0	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0	1	0	0	0	1	0	1	0	0	1	1
OS2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	1

Fig. 5. Demonstration of new offspring generation after applying genetic reproduction.

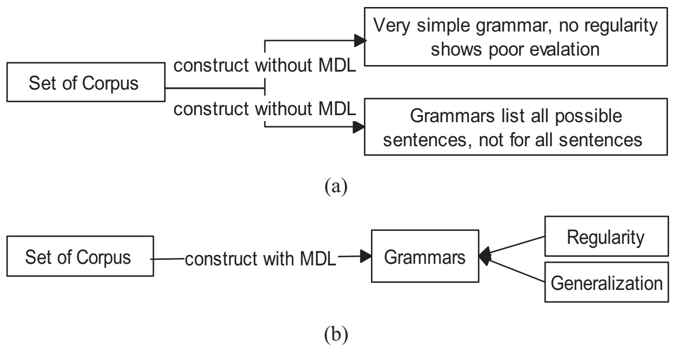
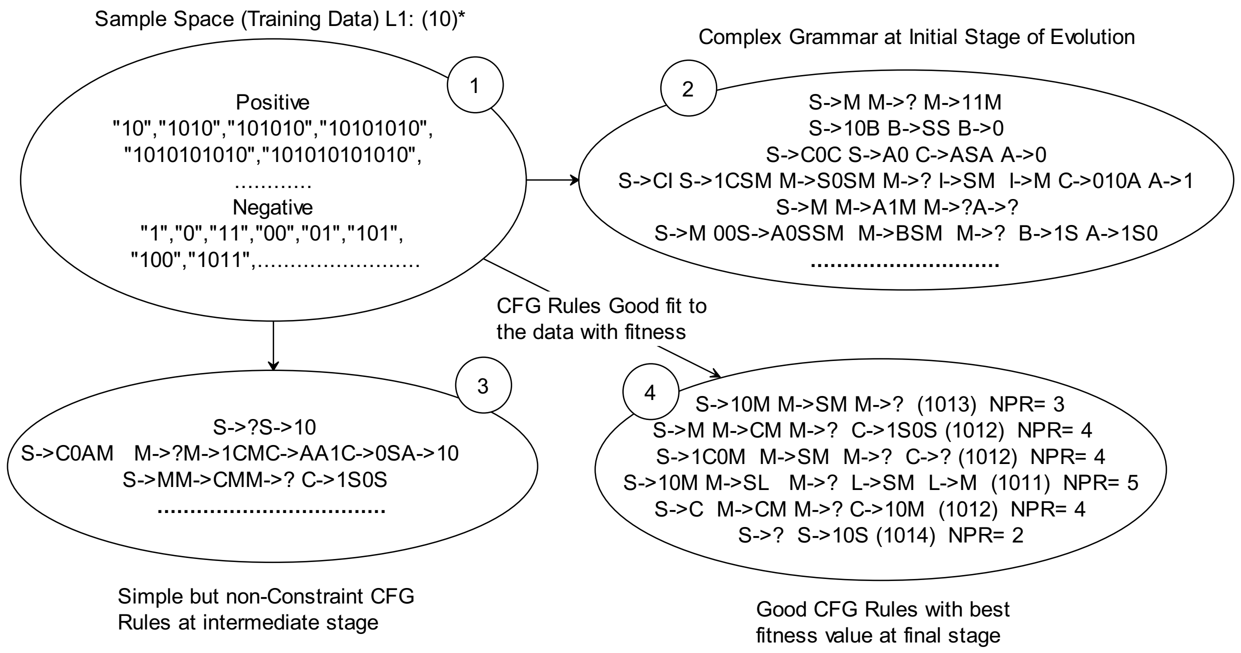


Fig. 6. MDL as a middle level for grammar construction.

of grammar using the MDL shows both regularities in the data and make generalizations beyond the observed corpus (see Fig. 6(b)). Therefore, the MDL behaves as a middle level and fills the gaps presented in Fig. 6(a).

Bayes theorem can be used to derive the MDL principle, but the working of the MDL principle is not similar to the Bayes, since the MDL uses code length rather probabilities [4,14]. The MDL was used widely in grammar induction problem [5,15–18].

In order to see the working of the MDL, an example of $L1 = (10)^*$ is presented (see Fig. 7). (1) First ellipse indicates the sample space of positive and negative training data for $L1 = (10)^*$. (2) It can be seen that initially we get very complex CFG rules with a very less fitness value which can be refined by applying the GA reproduction operator in each generation where MDL helps in compressing grammar rules and to generate positive and negative string set required during the execution. (3) After a few generations, we get a simple grammar, but non-constraint CFG rules. (4) But when the GA search process reaches to threshold/termination condition, it



Sample Space (Training Data) L1: (10)*
1
Positive
"10","1010","101010","10101010",
"1010101010","101010101010",
....
Negative
"1","0","11","00","01","101",
"100","1011",........................
Complex Grammar at Initial Stage of Evolution
2
S->M M->? M->11M
S->10B B->SS B->0
S->C0C S->A0 C->ASA A->0
S->CI S->1CSM M->S0SM M->? I->SM I->M C->010A A->1
S->M M->A1M M->?A->?
S->M 00S->A0SSM M->BSM M->? B->1S A->1S0
....
CFG Rules Good fit to the data with fitness
3
S->?S->10
S->C0AM M->?M->1CMC->AA1C->0SA->10
S->MM->CMM->? C->1S0S
....
Simple but non-Constraint CFG Rules at intermediate stage
4
S->10M M->SM M->? (1013) NPR= 3
S->M M->CM M->? C->1S0S (1012) NPR= 4
S->1C0M M->SM M->? C->? (1012) NPR= 4
S->10M M->SL M->? L->SM L->M (1011) NPR= 5
S->C M->CM M->? C->10M (1012) NPR= 4
S->? S->10S (1014) NPR= 2
Good CFG Rules with best fitness value at final stage

Fig. 7. Demonstration of MDL principle for $L1 = (10)^*$ which says that "more we are able to compress the data implies that we learned more" (NPR: number of production rules).

produces grammar's rules with fitness value. We can assume such grammars as good CFG rules with best fitness value. In the fourth ellipse six CFG rules are provided: first CFG rules have NPR = 3, fitness value = 1013. In second, third and fifth CFG, NPR = 4, fitness value = 1012 but here the noticeable thing is rules generated are different for the same language. At fourth CFG, NPR = 5, fitness value = 1011. In case of sixth CFG NPR = 2, fitness value = 1014, it indicated that the MDL compressed the data more in case of sixth CFG rules with maximum fitness value and therefore system learned more.

In the present scenario, for selecting the corpus, strings of terminals are generated for the length *L* from the given language. Initially, *L* = 0 is chosen, which gradually increases up to the required length to represent the language features. Here, a corpus of twenty five positive and negative strings is found to be sufficient to represent the chosen languages L1 through L8 for CFG induction.

2.5. Genetic algorithm implemented

The algorithm implemented using the bit-masking oriented data structure with crossover and mutation mask-fill operator is presented in this section. A *procedure* (*P1, P2, CM, MM*) is given in step 4 of the proposed algorithm to enter from *i*th generation to (*i* + 1)th generation. As soon as the system enters into the new generation, an individual population is updated with its fitness value, merges the population, and accordingly updates the best individual. Algorithm-4 is given below shows the step by step working of the proposed model.

```
Algorithm-4: CFG Induction uses BMOGA
Terminology Used: TER: Terminals, NTER: Non-terminals, PSIZE: Population
Size, BI: Best Individual, THERD: Threshold (maximum value of fitness
achieved), PO: Population, NEWPO: New Population, BS: Binary String, TOR:
Total Runs, UPFIT: Updated Fitness, APS: A set of accepted positive strings,
RNS: A set of rejected negative strings, RPS: A set of rejected positive strings,
ANS: A set of accepted negative strings, NPR: Number of production rules, P1,
P2: parents pair, CM: crossmask, MM: mutmask, Temp: Temporary
individuals, OS: offspring, Gmax: Maximum number of generations.
Input: Set of corpora (positive and negative)
Output: Context Free Grammar rules with fitness value and elapsed time.
S1. Initialize random population up to PSIZE.
S2. Grammar Induction Process
  • Generating variable length chromosome
  • Use sequential mapping to map of BS into TER or NTER and represent the
grammar using Backus Naur Form.
      $f : V \to B$ and $f : \Sigma^* \to B$
      $h : B \to V$ and $h : B \to \Sigma^*$
    Where, V represents the set of non-terminals and $\sum$ represents the set of
    terminals, while B represents set binary chromosomes, which is represented
    as 3-bit or 4-bit binary equivalent of non-terminals and terminal, f and h are
    the function used for mapping.
  • If (TER < 4 AND NTER < 4) Then
                    Apply 3-bit representation
  Else              Apply 4-bit representation
  • If (BS = = "010" OR BS = = "110") then
                    Set TER ← NULL (?) //'?' indicate null
  Else              Set symbol as appropriate
  • Eliminate left recursion, left factoring, multiple production rules, unit
    production if present
S3. Evaluate Fitness of an individual chromosome
  $Fitness = \sum C * ((APS + RNS) + (ANS + RPS)) + (2 * C - NPR)$
S4. Apply loop for performing GA operators
  • Until (BI >THERD) OR (TOR = = Gmax)
  • Apply crossover and mutation mask-fill operation using bit-masking
    oriented data structure.
  • Apply Boolean procedure based mixture (P1, P2, CM, MM)
    P1, P2 ← Select parent pairs via selection techniques as appropriate.
      CM ← initialize crossmask
      MM ← initialize mutmask
      Perform Temp1 ← P1 AND CM
      Perform Temp2 ← P2 AND (NOT CM)
      Perform Temp3 ← P2 AND CM
      Perform Temp4 ← P1 AND (NOT CM)
      Perform OS1 ← Temp1 OR Temp2
      Perform OS2 ← Temp3 OR Temp4
      Update OS1 ← OS1 XOR MM
      Update OS2 ← OS2 XOR MM
      Use OS1 and OS2 in next generation as appropriate.
  • NEWPO ← PO after crossover and mutation
  • UPFIT ← FITNESS (updated fitness)
  • Merge the population and Update the BI
S5. Display CFG rules, fitness value and elapsed time.
S6. Stop
```

3. Other approaches for premature convergence

The study is conducted on four different algorithms: a simple genetic algorithm (SGA), random offspring generation genetic algorithm (ROGGA), elite mating pool genetic algorithm (EMPGA) and dynamic allocation of reproduction operator (DARO). ROGGA, EMPGA and DARO are mainly proposed to address premature convergence. The SGA is given to lay down the basic understanding of the GA, whereas EMPGA and DARO algorithms are chosen because these approaches were implemented on CFG induction problems [25].

3.1. Simple genetic algorithm

SGA works using initial random population of fixed length chromosomes. The SGA starts with a set of solutions represented by chromosomes. 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 is repeated until the GA reaches to the threshold or the maximum number of generations. Fig. 8 shows the flowchart of the SGA. For the purpose of an evolution, the SGA performs a single crossover and a single mutation operator for reproduction. Then the individuals are picked depending on their fitness value to act as parents to generate offspring in the new generation.

Nicoară [33] presented an advanced version that uses more than one crossover and mutation reproduction operators, known as crossover-mutation combination reproduction to produce offspring based on their performance in the previous generation. Choubey and Kharat [25] executed the SGA multiple times in their proposed approach (EMPGA) to address premature convergence creating a mating pool (see Section 3.3).

3.2. Random offspring generation approach

Rocha and Jose [19] presented the ROGGA to resolve premature convergence. Fig. 9 shows the working of the ROGGA and how this

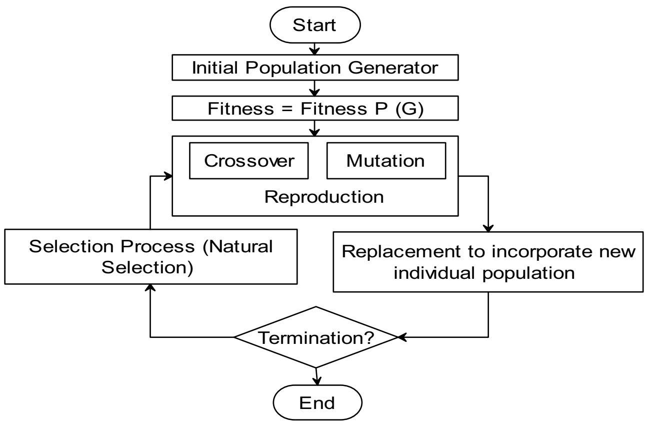

Start
Initial Population Generator
Fitness = Fitness P (G)
Crossover
Mutation
Reproduction
Selection Process (Natural Selection)
Replacement to incorporate new individual population
Termination?
End

Fig. 8. Flowchart for simple genetic algorithm [10].

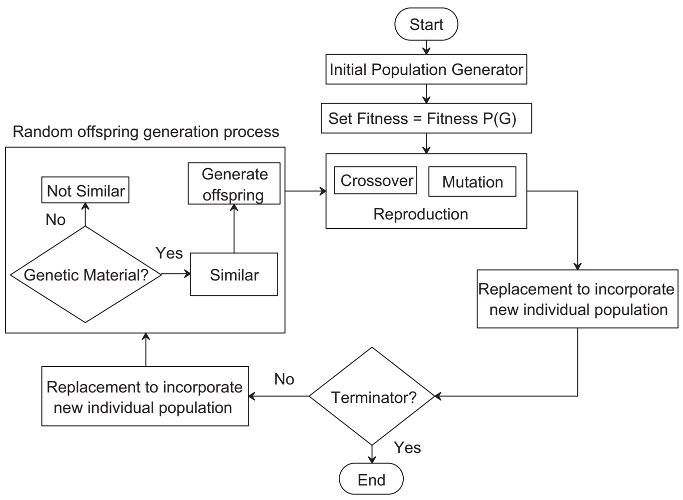

Start
Initial Population Generator
Set Fitness = Fitness P(G)
Random offspring generation process
Not Similar
Generate offspring
No
Yes
Genetic Material?
Similar
Crossover
Mutation
Reproduction
Replacement to incorporate new individual population
Replacement to incorporate new individual population
No
Terminator?
Yes
End

Fig. 9. Genetic algorithm flowchart with a random offspring generation process to prevent premature convergence.

approach prevents premature convergence. In this approach before applying reproduction operations, a test for similarity of the genetic material is done first. If found similar then generate a random offspring which will produce a random solution for the problem otherwise apply reproduction operators in normal fashion.

Two different strategies were used, namely 1-RO (one offspring per two parents) and 2-RO. In case of 1-RO only one offspring was generated and other one obtained from their parent to reach to the result while in 2-RO both the offspring was randomly used. It was well proven that the ROGGA perform better than the adaptive mutation rate (AMR) technique [20] and social disaster techniques [21]. Rocha and Jose [19] accepted that the weakness of this approach is the computational overheads disregarded since there is no change in the selection methodology compared to other approaches such as Crowding [22], Sharing [24] and Scheduled Sharing [23].

3.3. Elite mating pool approach

Choubey and Kharat [25] presented the EMPGA approach to prevent premature convergence and for context free grammar induction. Here, the SGA was executed multiple times on different populations to choose the best solution. The EMPGA approach works in the following manner:

S1 Generate a mating pool of size n<N, where N is the initial random population.
S2 Calculate the fitness of all individuals in the population.
S3 Sort the mating pool on the fitness of the individuals.
S4 Perform crossover of the first parent with the elite member of the mating pool and create two offspring.
S5 Replace old weaker individuals in the mating pool with newly created offspring in sorted order.
S6 Perform mutation on the newly created offspring.
S7 Replace old weaker individuals in the mating pool with newly created offspring in sorted order.
S8 Once repeat steps S4 through S7.

In order to test the effectiveness of the *EMPGA*, GA generations were processed with various probabilities ranging from 0% to 50%. The EMPGA had shown the capability of breaking the local optimum convergence. The main issue with this approach was extra processing time and spaces are required in maintaining the separate mating pool to get the elite members. Authors [25] accepted that the execution time of the EMPGA is assumed to be greater than a normal SGA approach.

3.4. Dynamic application of reproduction operators

Choubey and Kharat (2011) extended the work done in [33] and presented a Dynamic Application of Reproduction Operator (DARO) [25] where equal probability was assigned to each reproduction operator combination (crossover-mutation operator combination, CMOC) to begin the genetic algorithm generations. Three different crossover strategies, namely *two Point Crossover (TPC)*, *Two-Point Crossover with Internal Swapping (TPCIS)* and *Uniform Crossover (UC)* and four mutation techniques such as: *Stochastic Mutation*, *Inverse Mutation*, *Block Copier with Fixed Length Mutation* and *Block Copier with Random Length Mutation* were implemented in DARO. The same set of languages (languages used for EMPGA [25]) was considered for the experiment purpose. It was found that this method works effectively for simple languages, but there is scope for applying this method for more complex grammar sets. It was also observed that the DARO consume comparatively less time than the EMPGA, but it is not fit to handle the situation of local optimum convergence, whereas the EMPGA approach handle such situation effectively.

4. Simulation model

In order to evaluate the performance of the BMOGA experiments have been conducted, which utilize CM and MM for the CFG induction using a set of positive and negative corpora. Java programming on Net Beans IDE 7.0.1, Intel Core^{TM} 2processor (2.8 GHz) with 2 GB RAM have been used. Context Free, and Regular languages have been chosen with varying patterns of 0's and 1's is given in Table 1. Table 2 shows the parameters used for the implementation of the proposed algorithm. The MDLP has been used to generate the positive and a negative string set required during the execution [5].

GA is a stochastic search technique and therefore, the results are collected as the average of ten runs for each language. The GA search process continues until it reaches to a maximum number of generations or reaches to the threshold, where threshold indicates the highest rank solution's fitness. This stopping criterion is common for each algorithm implemented. The results show that the presented approach is capable for CFG induction. MDL principle is applied to generate a proportionate number of positive and negative sample strings. The masks fill operation for crossover

Table 1
Test languages description.

L-id	Descriptions	Standard set
L1	$(10)^*$ over $(0+1)^*$	Tomita [35]/Dupont set [36]
L2	String not containing '000' over $(0+1)^*$	–
L3	Balanced parentheses	Huijsen [34]/Keller and Luts set [5]
L4	Odd binary number ending with 1	Dupont set [36]
L5	Even binary number ending with 0 over $(0+1)^*$	–
L6	Any string with even 0 and odd 1 over $(0+1)^*$.	–
L7	$0(00)^*1$ over $(0+1)^*$	–
L8	0^*1 over $(0+1)^*$.	–

Table 2
Parameter used for GA simulation.

S.N.	Parameter	Value/size
1.	Population size	100
2.	Chromosome size	250
3.	Corpus size	50
4.	Max. generation	500
5.	CrossRate	0.9
6.	MutRate	0.8

Table 3
Generated grammar with fitness value and total number of rules.

L-id	Fit.	Grammar <V, $\sum$, P, S>	NPR
L1	1014	<{S}, {0, 1}, {S→?, S→10S}, S>	2
L2	1011	<{S,C,M}, {0, 1}, {S→CCM, M→?, M→1SM, C→?, C→0}, S>	5
L3	1014	<{S}, {(,)}, {S→?, S→ (S)S}, S>	2
L4	1012	<{S, M}, {0, 1}, {S→1M, S→0SM, M→SM, M→?}, S>	4
L5	1012	<{S, C}, {0, 1}, {S→C, S→1S, S→ 0S, C→0}, S>	4
L6	1011	<{S, M}, {0, 1}, {S→1M, S→0SM, M→SSM, M→?, M→0M}, S>	5
L7	1013	<{S, C}, {0, 1}, {S→C, S→00S, C→01}, S>	3
L8	1014	<{S}, {0, 1}, {S→1, S→ 0S}, S>	2

Fit: fitness value, L-id: language ID, NPR: number of production rules.

Table 4
Comparison based on average rules length and success ratio when reached to threshold.

L-id	SGA		ROGGA		EMPGA		DARO		BMOGA	
	APR	SR	APR	SR	APR	SR	APR	SR	APR	SR
L1	4.1	10%	4.1	10%	3.2	30%	3.6	30%	2.8	30%
L2	8.3	10%	7.5	10%	7.5	10%	7.1	20%	6.5	30%
L3	5.8	10%	5.8	10%	4.7	10%	5.2	20%	4.0	20%
L4	6.5	30%	6.2	30%	5.5	30%	5.2	40%	4.7	60%
L5	6.7	40%	5.3	40%	5.8	30%	5.6	50%	5.1	60%
L6	6.7	30%	6.4	30%	6.2	40%	6.1	40%	6.0	60%
L7	4.5	30%	4.3	40%	4.1	60%	3.7	60%	3.6	60%
L8	3.1	50%	2.6	60%	2.6	60%	2.5	60%	2.2	80%

APR: average production rules, SR: success ratio.

and mutation methods have been found effective in improving the performance. It was observed that the MDL principle works effectively in selecting the correct sample strings with minimum length. For the validation purpose experimentally obtained grammars have been tested against the best known available grammar. The standard representation <V, Σ, P, S> is considered to show the best grammar. Table 3 shows the best generated grammars with a fitness value (FV) and total number of rules received.

In syntactic pattern analysis number of rules represent the pattern is a crucial factor and it must be kept minimum. Therefore, comparative analyses have been done based on the number of grammar's rules obtained for the languages L1 through L8. The Average Production Rules (APR) has been computed for each GA approach as presented in Table 4. Eq. (7) is used to compute the APR in which term NPR represents the *Number of Production Rules* obtained from one (the best) chromosome, when the search process reached to a maximum number of generations or to the threshold. For example, the APR value for L1 in case of the BMOGA is 2.8 (see Table 4). As discussed, the GA is a stochastic search algorithm, the results are collected for ten runs, for which the value of NPR's is respectively 3, 2, 3, 3, 3, 3, 4, 3, 2, and 2, which results APR = 2.8 after applying Eq. (7).

It is clear from Fig. 10 that for each language L1 through L8 the value of the APR obtained implementing the BMOGA is the lowest, i.e. it outperformed other GAs. It can be seen from Fig. 10 and Table 4 that the EMPGA, ROGGA and the DARO showed the average values of the APR whereas the SGA showed the maximum value of APR.

The success ratio for each algorithm is calculated helps in judging the suitability of each approach. The high value of success ratio indicates that the approach is more suitable for grammar induction. Eq. (8) has been used for calculating the percentage of success ratio.

$$\text{APR} = \frac{\sum_{i=1}^{n} \text{NPR}_i}{\sum_{j=1}^{10} NR_j} \tag{7}$$

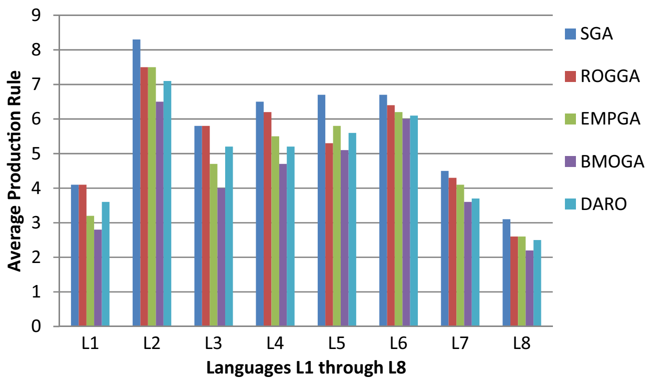


Fig. 10. Average production rules comparison chart for each language L1 through L8 w.r.t. SGA, EMPGA, ROGGA, DARO and BMOGA.

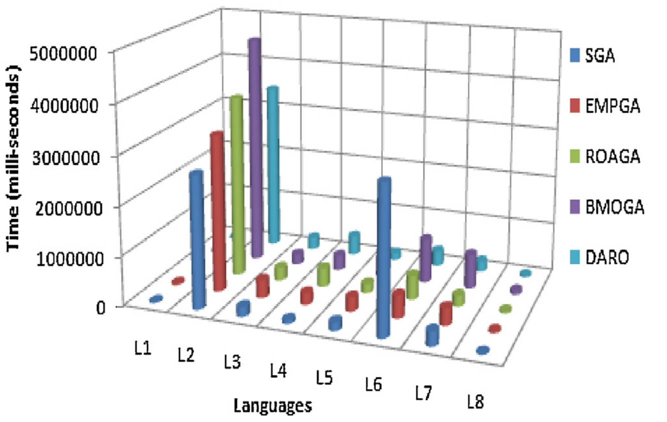


Fig. 11. Execution time comparison chart for each language L1 through L8 w.r.t. SGA, EMPGA, ROGGA, DARO and BMOGA.

where NR_j represent the number of runs and $NR_j = 1$ is chosen.

$$\text{Succ. Ratio} = \frac{\sum_{i=1}^{n} \text{NOPR}_i}{\sum_{j=1}^{10} \text{NR}_j} \times 100 \tag{8}$$

where NOPR: number of optimal production rules is taken from (the best) chromosome and NR_j ($NR_j = 1$ is chosen) represent the number of runs. The term NOPR indicates production rules for which maximum value of fitness is achieved. It indicates the learning capability of the system. The following conventions are followed: production rules with a maximum value of fitness indicate that the system has learned more and vice versa.

Table 4 shows the percentage success ratio for each language. The value of *Succ. Ratio* is found higher for the BMOGA compared to other algorithms: SGA, ROGGA, DARO and EMPGA for the chosen languages. Table 5 shows the execution time consumed to reach to the threshold value. It is depicted in Fig. 11 that the computational cost of the BMOGA is slightly higher in reaching to the global solution since the whole process runs in two separate phases. Although, the execution time for the BMOGA is comparatively higher than the other GAs, but it produces better results in both the aspects: APR and success ratio.

Table 6 presents generation range, mean and standard deviation. The results are collected as the average of first successful ten runs. The number of generations taken over ten generation run varies, therefore generation range is given. The phenomenon involved with the generation range can be understood with the help of an example: the generation range for L1 in case of the SGA is 6 ± 2, means the number of generations taken over ten generation run varies between 04 (6 – 2) and 08 (6 + 2) similarly for others.

Table 5
Comparison of threshold value and time elapsed by each approach.

L-id	SGA		ROGGA		EMPGA		DARO		BMOGA	
	Th.	Time	Th.	Time	Th.	Time	Th.	Time	Th.	Time
L1	7	26,064.1	7	56,012.3	7	38,801.1	11	24,748.5	9	68,671.3
L2	22	2,690,711	24	3,701,925	26	3,192,827	33	3,456,620	30	4,693,783
L3	12	196,347	13	268,276.2	18	365,851.8	12	244,034.4	10	205,309.5
L4	14	73,749.7	25	380,766.3	23	235,360.3	25	400,404.8	12	305,396.7
L5	21	170,964.1	14	185,769.7	29	288,390.2	14	142,451.3	13	366,416.1
L6	166	2,968,404	33	484,304.8	24	462,148.7	41	313,116.2	33	889,390.5
L7	40	292,091.9	31	218,135.7	29	343,464	24	210,542.3	43	694,897
L8	6	13,816.4	10	37,865.3	7	33,226.7	15	37,137.1	12	74,882

Th.: threshold, Time: time in milliseconds.

Table 6
Statistical analysis of each approach.

L-id	SGA			ROGGA			EMPGA			DARO			BMOGA		
	G.R.	μ	SD	G.R.	μ	SD	G.R.	μ	SD	G.R.	μ	SD	G.R.	μ	SD
L1	6 ± 2	4.1	2.73	4 ± 2	3.9	1.8	5 ± 2	3.4	2.5	4 ± 2	4.8	1.6	4 ± 2	3	2.75
L2	14 ± 10	17.4	6.43	20 ± 12	25.7	10.34	22 ± 11	18.2	6.9	16 ± 10	16.9	6.7	12 ± 9	17.5	6.64
L3	8 ± 4	8.2	2.44	7 ± 3	7.2	2.97	8 ± 3	9.1	5.22	8 ± 3	10.4	2.25	4 ± 3	6.7	2.26
L4	7 ± 4	8.1	3.87	12 ± 6	12	8.01	9 ± 4	10.8	5.41	12 ± 8	14.9	2.5	7 ± 3	6.9	3.31
L5	11 ± 6	10.4	5.88	8 ± 4	10.6	3.1	13 ± 6	9.2	7.61	10 ± 6	8.3	2.2	9 ± 5	8.8	2.97
L6	108 ± 54	113.3	42.7	15 ± 5	16.0	7.96	12 ± 9	14.1	4.86	14 ± 10	16.4	2.0	16 ± 11	15.7	7.2
L7	16 ± 13	15.4	9.45	13 ± 7	11	9.41	15 ± 5	11	10.02	9 ± 4	9.8	2.3	21 ± 11	18.8	12.58
L8	3 ± 2	2.6	1.43	8 ± 2	3.9	3.03	7 ± 2	3.6	1.89	8 ± 4	7.8	1.8	8 ± 3	4.3	3.13

G.R.: generation range; μ: mean; S.D.: standard deviation.

Table 7
Descriptive analysis.

	N	Mean	Std. deviation	Std. error	95% confidence interval for mean		Minimum	Maximum
					Lower bound	Upper bound		
SGA	15	666.731867	152.0970509	39.2712897	582.503327	750.960406	509.2420	1008.0000
EMPGA	15	777.673333	157.5512228	40.6795508	690.424374	864.922292	566.0000	1011.3000
ROGGA	15	741.960000	113.1219431	29.2079601	679.315156	804.604844	582.0000	920.0000
DARO	15	737.413333	111.3869309	28.7599819	675.729307	799.097360	580.2000	940.0000
BMOGA	15	894.246667	163.0063797	42.0880662	803.976742	984.516591	607.0000	1013.9000
Total	75	763.605040	156.5372587	18.0753657	727.589088	799.620992	509.2420	1013.9000

A statistical test is conducted to evaluate the performance significance of the BMOGA compare to the other GA approach. *F*-test is conducted on the collected sample considering the hypothesis: "*there is no significant difference in the mean of samples at the 5% level of confidence*" i.e.

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5$$

H_A: At least one mean is different than others.

F-test is used to test whether two samples may be regarded as drawn from the normal population have the same variance. The detailing of the *F*-test is given in [37]. The purpose of applying the *F*-test based on the analysis of variance (ANOVA) is to verify whether the samples collected from the five algorithms lie within the same group or is there any possibility that one or the other methods different from the group. The total of 15 samples was drawn from each algorithm for the randomly chosen languages. The descriptive analysis is depicted in Table 7 and Fig. 12 graphically displays the means for the five algorithm groups. The *X*-axis represents the five approaches (SGA, EMPGA, ROGGA, DARO and BMOGA). The *Y*-axis represents the estimated marginal mean fitness values. The performance of the BMOGA is found better than the other approaches whereas the EMPGA showed slightly better performance than the DARO and ROGGA, and the SGA's performance is found worst. The main ANOVA result is given in Table 8. The significance value ($p = 0.001$) comparing the group (algorithms) is less than 0.05 ($0.001 < 0.05$), so we could reject the null hypothesis. The result indicates that one of the samples is better than the other ones. However, this result does not indicate which methods are responsible for the difference, therefore it is necessary to apply post hoc test.

Table 8
Result received after ANOVA test.

	Sum of squares	df	Mean square	*F*	Sig.
Between groups	417061.147	4	104265.287	5.227	.001
Within groups	1396228.441	70	19946.121		
Total	1813289.588	74			

The post hoc tests compare individual group results with each other. The least significant difference (LSD) test is applied. It was developed by Fisher with the purpose to explore all possible pairwise comparisons of means comparing a factor (in this case fitness) using the equivalent of multiple t-test. It is interesting to note that, there are various tests one can choose, and they may produce different results. Table 9 shows the result of multiple comparisons using the LSD method. First column "(I) Samples" and the second column "(J) Samples" represent the combination of the paired algorithm for comparison. For example, the first row contains SGA (in (I) Samples) compared with the other approaches (EMPGA, ROGGA, DARO and BMOGA) given in the (J) sample column. The mean difference (I – J) represents the mean difference of (I) samples and (J) samples. In case of the SGA-EMPGA pair: mean of SGA and EMPGA is 666.731867 and 777.673333 respectively given in Table 7 and

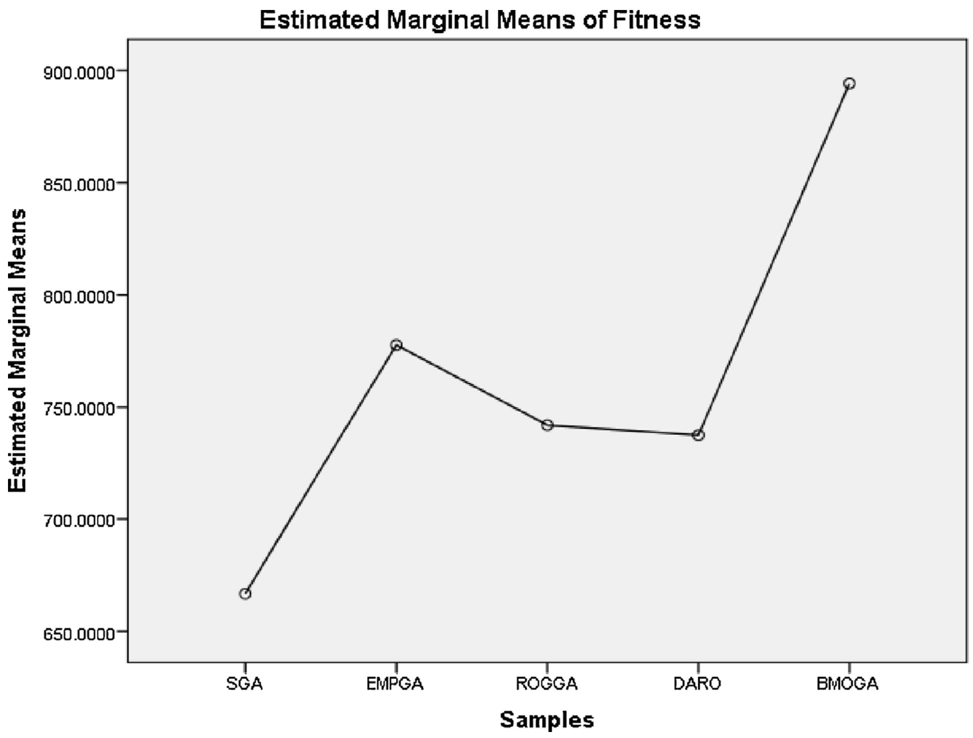


Fig. 12. Profile plot for estimated marginal means of fitness for each approach.

the difference is −110.9414667 similarly for others. One need to pay attention to the astrix mark (*) given next to the mean difference, indicates the difference is significant. The last row of Table 9 indicates that the mean difference of the BMOGA and the other algorithm is significantly different, since the mean difference has astrix (*) marks immediately to the difference value. The significant difference can also be verified by comparing the *p*-value present in column five of Table 9. If the obtained *p*-value is less than 0.05, then the algorithmic combination is significantly different. It can be seen that the algorithmic combination BMOGA and others (SGA, EMPGA, ROGGA and DARO) has the p value 0.000, .027, .004 and .003 less than 0.05, so the mean of the BMOGA is significantly different than the other algorithms and therefore we can conclude that the BMOGA's performance is better than other approaches.

Although the LSD produces results accurately, it is very sensitive to the violation to the assumptions of the ANOVA, so the most likely to lead to the Type 1 error, i.e. rejecting the null hypothesis when it is true. To avoid this situation, the Tukey HSD (Tukey–Kramer honestly significant difference) test is also conducted. It is very similar to the LSD test, but less liable to a Type 1 error. In addition, it is suitable for the experiment that have an equal number of observations (in our case $N = 15$).

Here, pay attention to the last row of Table 10, the pairwise comparison for the BMOGA-SGA is strongly significantly different ($0.000 < 0.05$), BMOGA-ROGGA has been significantly different since $0.034 < 0.05$, BMOGA-DARO has also been significantly different since $0.027 < 0.05$, but BMOGA-EMPGA is not significantly different since $0.170 > 0.05$. Hence, it can be concluded that the

Table 9
Result of multiple comparisons (post hoc tests: LSD test).

(I) Samples	(J) Samples	Mean difference (I – J)	Std. error	Sig.	95% confidence interval	
					Lower bound	Upper bound
SGA	EMPGA	−110.9414667*	51.5701730	.035	−213.794934	−8.088000
	ROGGA	−75.2281333	51.5701730	.149	−178.081600	27.625334
	DARO	−70.6814667	51.5701730	.175	−173.534934	32.172000
	BMOGA	−227.5148000*	51.5701730	.000	−330.368267	−124.661333
EMPGA	SGA	110.9414667*	51.5701730	.035	8.088000	213.794934
	ROGGA	35.7133333	51.5701730	.491	−67.140134	138.566800
	DARO	40.2600000	51.5701730	.438	−62.593467	143.113467
	BMOGA	−116.5733333*	51.5701730	.027	−219.426800	−13.719866
ROGGA	SGA	75.2281333	51.5701730	.149	−27.625334	178.081600
	EMPGA	−35.7133333	51.5701730	.491	−138.566800	67.140134
	DARO	4.5466667	51.5701730	.930	−98.306800	107.400134
	BMOGA	−152.2866667*	51.5701730	.004	−255.140134	−49.433200
DARO	SGA	70.6814667	51.5701730	.175	−32.172000	173.534934
	EMPGA	−40.2600000	51.5701730	.438	−143.113467	62.593467
	ROGGA	−4.5466667	51.5701730	.930	−107.400134	98.306800
	BMOGA	−156.8333333*	51.5701730	.003	−259.686800	−53.979866
BMOGA	SGA	227.5148000*	51.5701730	.000	124.661333	330.368267
	EMPGA	116.5733333*	51.5701730	.027	13.719866	219.426800
	ROGGA	152.2866667*	51.5701730	.004	49.433200	255.140134
	DARO	156.8333333*	51.5701730	.003	53.979866	259.686800

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

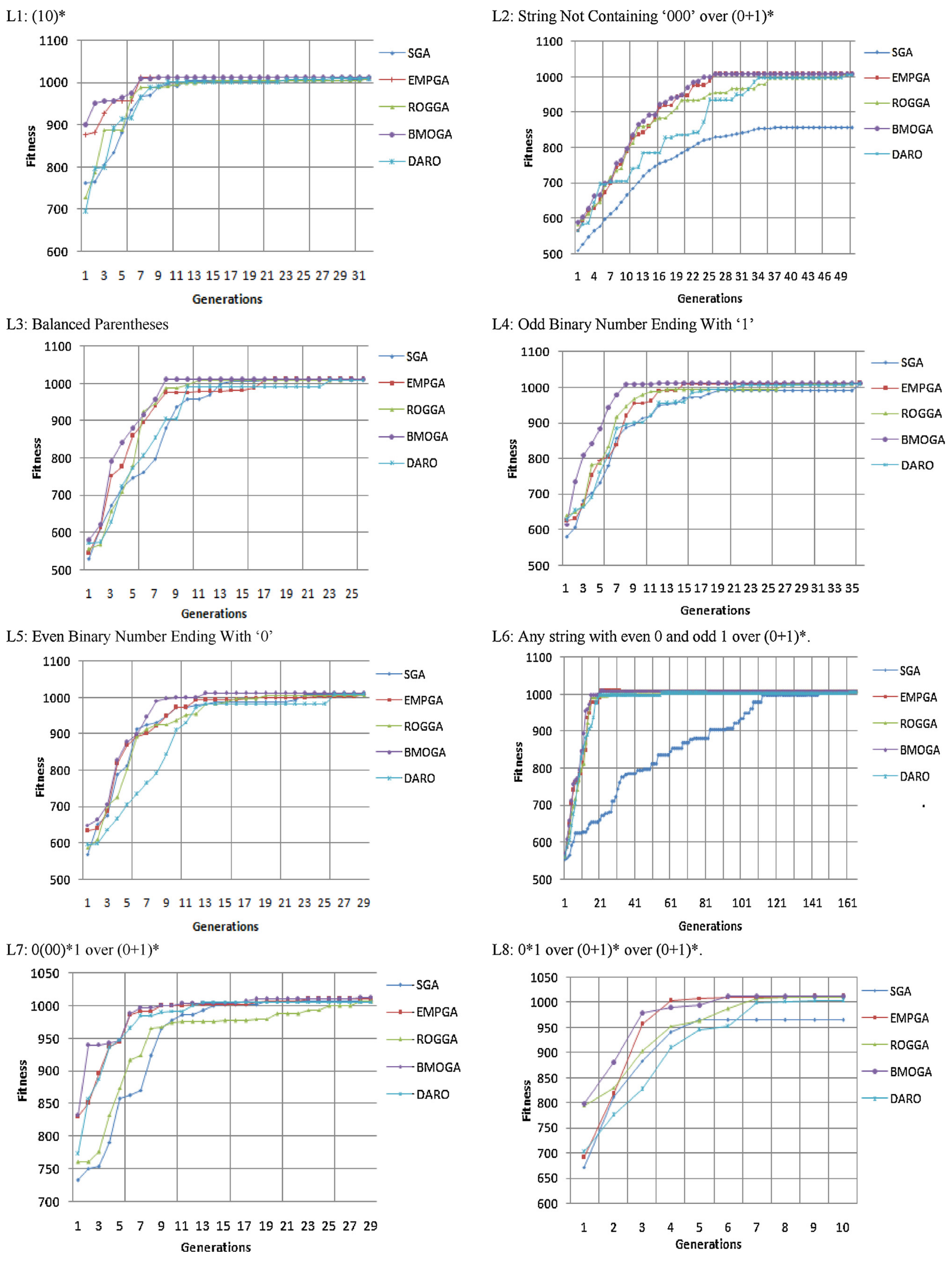


Fig. 13. Fitness vs. generations chart for languages L1 through L8 with respect to each genetic algorithm (SGA, EMPGA, ROGGA, DARO and BMOGA).

Table 10
Result of multiple comparisons (post hoc tests: TukeyHSD test).

(I) Samples	(J) Samples	Mean Difference (I – J)	Std. error	Sig.	95% confidence interval	
					Lower bound	Upper bound
SGA	EMPGA	−110.9414667	51.5701730	.211	−255.345839	33.462906
	ROGGA	−75.2281333	51.5701730	.592	-219.632506	69.176239
	DARO	−70.6814667	51.5701730	.648	−215.085839	73.722906
	BMOGA	−227.5148000[*]	51.5701730	.000	−371.919173	−83.110427
EMPGA	SGA	110.9414667	51.5701730	.211	−33.462906	255.345839
	ROGGA	35.7133333	51.5701730	.957	−108.691039	180.117706
	DARO	40.2600000	51.5701730	.935	−104.144373	184.664373
	BMOGA	−116.5733333	51.5701730	.170	−260.977706	27.831039
ROGGA	SGA	75.2281333	51.5701730	.592	−69.176239	219.632506
	EMPGA	−35.7133333	51.5701730	.957	−180.117706	108.691039
	DARO	4.5466667	51.5701730	1.000	−139.857706	148.951039
	BMOGA	−152.2866667[*]	51.5701730	.034	−296.691039	−7.882294
DARO	SGA	70.6814667	51.5701730	.648	−73.722906	215.085839
	EMPGA	−40.2600000	51.5701730	.935	−184.664373	104.144373
	ROGGA	−4.5466667	51.5701730	1.000	−148.951039	139.857706
	BMOGA	−156.8333333[*]	51.5701730	.027	−301.237706	−12.428961
BMOGA	SGA	227.5148000[*]	51.5701730	.000	83.110427	371.919173
	EMPGA	116.5733333	51.5701730	.170	−27.831039	260.977706
	ROGGA	152.2866667[*]	51.5701730	.034	7.882294	296.691039
	DARO	156.8333333[*]	51.5701730	.027	12.428961	301.237706

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

Table 11
Means for groups in homogeneous subsets are displayed.

	Samples	*N*	Subset for alpha = 0.05	
			1	2
Tukey HSD[a]	SGA	15	666.731867	
	DARO	15	737.413333	
	ROGGA	15	741.960000	
	EMPGA	15	777.673333	777.673333
	BMOGA	15		894.246667
	Sig.		.211	.170

[a] Uses harmonic mean sample size = 15.000.

performance of the BMOGA is better than SGA, ROGGA and DARO but slightly better than the EMPGA or the performance of the BMOGA and EMPGA is almost similar. Homogeneity test is conducted to verify the similarity using the TukeyHSD test. The means of algorithms in homogeneous subsets are shown in Table 11. Looking at Table 11, it can be seen that the SGA, ROGGA and DARO fall in the group1 at 0.211 significance level, whereas the BMOGA and EMPGA are in the same group at 0.170 significance level, since both the algorithms showing almost the similar performance (where BMOGA is slightly better than EMPGA).

The comparative generation chart (fitness vs. generation) for each language is depicted in Fig. 13. Overall, the performance of the BMOGA is found better than other algorithms except for L6 for which each method showed the similar performance (SGA showed worst). For each language L1 through L8 the BMOGA outperformed other approaches. It was observed that the EMPGA showed a higher tendency of getting local minima during the execution.

5. Conclusions

In this paper, we have presented the BMOGA, when compared with other approaches such as SGA, ROGGA, EMPGA and DARO is found effective for the CFG induction. The analysis explains the effectiveness of the proposed approach. Although the execution time required is higher but it showed promising results in terms of APR and success ratio. It was observed that the BMOGA outperformed SGA, ROGGA, DARO and showed better results than the EMPGA. Though the execution time varies from language to language, it follows the way that for complex language, it needs more time and vice versa. The role of the MDL is discussed, i.e. how the MDL can be used to generate the corpus of desired length to represent the language features. However, it is thereby found that the BMOGA is a better choice for grammar induction process since it produces optimal results. An important issue of the GA known as premature convergence and its effects is discussed. It was observed that the bit masking oriented data structure and the Boolean based procedure helps in alleviating premature convergence by introducing diversity in the population during offspring generation. It was discussed in [41] that the ROGGA perform better than the adaptive mutation rate (AMR) technique [20] and social disaster techniques [21], whereas the DARO showed better results than the work conducted in [33]. Therefore, one can say that the BMOGA can also perform better than the AMR, social disaster techniques and the work done in [33]. The *F*-test is conducted to verify the hypothesis H_0 and H_A (see Section 4), the results indicates that one of the samples showed better results than others, hence rejects the null hypothesis. In order to understand, which method is responsible for the difference, the post hoc tests (LSD and TukeyHSD) are conducted, which indicates that the BMOGA perform better than the SGA, ROGGA and DARO, whereas the performance of the EMPGA is close to the BMOGA. Although, the comparative results showed the suitability of the BMOGA for grammar induction and in addressing premature convergence, but still there is scope for improvements. Some of them are: improving the fitness function dynamically by setting the weight factor for every training example, applying it for more difficult problems such as natural languages and it would be interesting to compare the proposed algorithms with other approaches given to avoid premature convergence since it showed the tendency of getting trapped at local optimum convergence in some situations.

References

[1] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, MIT Press, 1992.
[2] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, 1989.

[3] P. Wyard, Representational issues for context-free grammar induction using 431 genetic algorithm, in: Proceedings of the 2nd International Colloquium on 432 Grammatical Inference and Applications, Lecture Notes in Artificial Intelligence, vol. 862, 1994, pp. 222–235, 434.
[4] M.H. Hansen, B. Yu, Model selection and the principle of minimum description length, J. Am. Stat. Assoc. 96 (454) (2001) 746–774.
[5] B. Keller, R. Lutz, Evolving stochastic context-free grammars from examples using a minimum description length principle, in: 1997 Workshop on Automata Induction Grammatical Inference and Language Acquisition, 1997.
[6] Y. Sakakibara, Recent advances of grammatical inference, Theor. Comput. Sci. 185 (1) (1997) 15–45.
[7] H.M. Pandey, Advances in ambiguity detection methods for formal grammars, Proc. Eng. 24 (2011) 700–707.
[8] Pandey, Hari Mohan, and M. Tech. “A Case Study on Ambiguity Detection Method Using Some Set of Grammars.”.
[9] N.S. Choubey, H.M. Pandey, M.U. Kharat, Developing genetic algorithm library using Java for CFG induction, Int. J. Adv. Technol. 2 (1) (2011) 117–128.
[10] H.M. Pandey, A. Dixit, D. Mehrotra, Genetic algorithms: concepts, issues and a case study of grammar induction, in: Proceedings of the CUBE International Information Technology Conference, ACM, 2012.
[11] R. Sivaraj, T. Ravichandran, A review of selection methods in genetic algorithm, Int. J. Eng. Sci. Technol. 3 (5) (2011).
[12] L. Iuspa, F. Scaramuzzino, A bit-masking oriented data structure for evolutionary operator's implementation in genetic algorithms, Soft Comput. 5 (1) (2001) 58–68.
[13] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, 1996.
[14] J. Rissanen, Modeling by shortest data description, Automatica 14 (5) (1978) 465–471.
[15] H. Hlynsson, Transfer Learning using the Minimum Description Length Principle with a Decision Tree Application, 2007.
[16] I. Jonyer, L.B. Holder, D.J. Cook, MDL-based context-free graph grammar induction and applications, Int. J. Artif. Intell. Tools 13 (01) (2004) 65–79.
[17] M. Saers, K. Addanki, D. Wu, Iterative rule segmentation under minimum description length for unsupervised transduction grammar induction, in: Statistical Language and Speech Processing, Springer, Berlin, Heidelberg, 2013, pp. 224–235.
[18] K. Lee, T.-K. Kim, Y. Demiris, Learning action symbols for hierarchical grammar induction, in: 21st International Conference on Pattern Recognition (ICPR), IEEE, 2012.
[19] M. Rocha, J. Neves, Preventing premature convergence to local optima in genetic algorithms via random offspring generation, in: Multiple Approaches to Intelligent Systems, 1999, pp. 127–136.
[20] S.H. Jung, Selective mutation for genetic algorithms, in: World Academy of Science, Engineering and Technology, vol. 56, 2009, pp. 478–481.
[21] V.M. Kureichick, V.V. Miagkikh, A.P. Topchy, Genetic algorithm for solution of the traveling salesman problem with new features against premature convergence, Proc. GA + SE 96 (1996).
[22] K.A. De Jong, Analysis of the Behavior of a Class of Genetic Adaptive Systems, 1975.
[23] M.E. Palmer, S.J. Smith, Improved evolutionary optimization of difficult landscapes: control of premature convergence through scheduled sharing, Complex Syst. 5 (5) (1992) 443–458.
[24] D.E. Goldberg, J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their Application, L. Erlbaum Associates Inc., 1987.
[25] N. Choubey, M. Kharat, Approaches for handling premature convergence in CFG induction using GA, in: Soft Computing in Industrial Applications, 2011, pp. 55–66.
[26] E.M. Gold, Language identification in the limit, Inform. Control 10 (5) (1967) 447–474.
[27] T. Theeramunkongy, M. Okumuray, Grammar acquisition and statistical parsing by exploiting local contextual information, J. Nat. Lang. Process. 2 (3) (1995).
[28] F. Javed, et al., Context-free grammar induction using genetic programming, in: Proceedings of the 42nd Annual Southeast Regional Conference, ACM, 2004.
[29] N.S. Choubey, M.U. Kharat, Sequential structuring element for CFG induction using genetic algorithm, Int. J. Futuristic Comput. Appl. 1 (2010).
[30] H.M. Pandey, Context free grammar induction library using Genetic Algorithms, in: 2010 International Conference on Computer and Communication Technology (ICCCT), IEEE, 2010.
[31] L.L. Whitcomb, Notes on Kronecker Products, 2013, Available: spray.me.jhu.edu/llw/courses/me530647/kron_1.pdf.
[32] Phillip A. Regalia, Mitra K. Sanjit, Kronecker products, unitary matrices and signal processing applications, SIAM Rev. 31.4 (1989) 586–613.
[33] E.S. Nicoară, Mechanisms to avoid the premature convergence of genetic algorithms, Petroleum-Gas University of Ploiesti Bulletin, Mathematics-Informatics-Physics Series 61 (1) (2009).
[34] W.-O. Huijsen, Genetic grammatical inference, in: CLIN IV: Papers from the Fourth CLIN Meeting, 1993.
[35] M. Tomita, Dynamic construction of finite-state automata from examples using hill-climbing, in: Proceedings of the Fourth Annual Cognitive Science Conference, 1982.
[36] P. Dupont, Regular grammatical inference from positive and negative samples by genetic search: the GIG method, in: Grammatical Inference and Applications, Springer, Berlin, Heidelberg, 1994, pp. 236–245.
[37] R.G. Lomax, D.L. Hahs-Vaughn, Statistical Concepts: A Second Course, Routledge, 2013.
[38] Horst Bunke, Sanfeliu Alberto (Eds.), Syntactic and Structural Pattern Recognition: Theory and Applications., vol. 7, World Scientific, 1990.
[39] A. Stevenson, J.R. Cordy, Grammatical inference in software engineering: an overview of the state of the art, in: Software Language Engineering, Springer, Berlin, Heidelberg, 2013, pp. 204–223.
[40] A. Stevenson, J.R. Cordy, A survey of grammatical inference in software engineering, in: Science of Computer Programming, 2014.
[41] H.M. Pandey, A. Choudhary, D. Mehrotra, A comparative review of approaches to prevent premature convergence in GA, in: Applied Soft Computing, 2014.
[42] Tobias Friedrich, et al., Analysis of diversity-preserving mechanisms for global exploration, Evol. Comput. 17 (4) (2009) 455–476.
[43] D. Whitley, An overview of evolutionary algorithms: practical issues and common pitfalls, Inform. Softw. Technol. 43 (14) (2001) 817–831.
[44] H. Shimodaira, A diversity-control-oriented genetic algorithm (DCGA): performance in function optimization, in: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, IEEE, 2001.
[45] Geoffrey K. Pullum, Learnability, hyperlearning, and the poverty of the stimulus, Proc. Annu. Meet. Berkeley Linguist. Soc. 22 (1) (2012).
[46] D. Angluin, C.H. Smith, Inductive inference: theory and methods, ACM Comput. Surv. (CSUR) 15 (3) (1983) 237–269.
[47] K.S. Fu, Syntactic Pattern Recognition and Applications, vol. 4, Prentice-Hall, Englewood, Cliffs, 1982.
[48] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley Longman Publishing Co., Inc., 1978.
[49] K.J. Lang, Random DFA's can be approximately learned from sparse uniform examples, in: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, ACM, 1992.
[50] A.L. Oliveira (Ed.), Grammatical Inference: Algorithms and Applications: 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11–13, 2000, Proceedings No. 1891, Springer, 2000.
[51] A.C.F. Coste, L. Miclet, Grammatical Inference: Algorithms and Applications, 2008.
[52] Y. Sakakibara, et al., Proceedings of Grammatical Inference: Algorithms and Applications, 2006.
[53] A. Cleeremans, D. Servan-Schreiber, J.L. McClelland, Finite state automata and simple recurrent networks, Neural Comput. 1 (3) (1989) 372–381.
[54] G. Alex, et al., A novel connectionist system for unconstrained handwriting recognition, IEEE Trans. Pattern Anal. Mach. Intell. 31 (5) (2009) 855–868.
[55] J.L. Elman, Finding structure in time, Cogn. Sci. 14 (2) (1990) 179–211.
[56] M. Delgado, M.C. Pegalajar, A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference, Pattern Recogn. 38 (9) (2005) 1444–1456.
[57] A. D'Ulizia, F. Ferri, P. Grifoni, A survey of grammatical inference methods for natural language learning, Artif. Intell. Rev. 36 (1) (2011) 1–27.
[58] D. Angluin, Inductive inference of formal languages from positive data, Inform. Control 45 (2) (1980) 117–135.
[59] D. Angluin, Queries and concept learning, Mach. Learn. 2 (4) (1988) 319–342.
[60] L.G. Valiant, A theory of the learnable, Commun. ACM 27 (11) (1984) 1134–1142.
[61] M. Li, P.M.B. Vitányi, Learning simple concepts under simple distributions, SIAM J. Comput. 20.5 (1991) 911–935.
[62] C. De La Higuera, A bibliographical study of grammatical inference, Pattern Recogn. 38 (9) (2005) 1332–1348.
[63] H. Colin de la, Grammatical Inference: Learning Automata and Grammars, Cambridge University Press, New York, NY, USA, 2010.
[64] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, ACM Comput. Surv. (CSUR) 45 (3) (2013) 35.
[65] S. Jain, A. Sharma, Generalization and specialization strategies for learning re languages, Ann. Math. Artif. Intell. 23 (1–2) (1998) 1–26.
[66] S. Winberg, C. Balkenius, Generalization and specialization in reinforcement learning, in: The Seventh International Conference on Epigenetic Robotics, vol. 134, Lund University Cognitive Science, 2007.
[67] C. De Stefano, A. Marcelli, Generalization vs. specialization: quantitative evaluation criteria for genetics-based learning systems, in: IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, vol. 3, IEEE, 1997.

Hari Mohan Pandey is major in Computer Science and Engineering and pursuing Ph.D. in Language Processing and Evolutionary Algorithms. He has published research papers in various journals. He is the author of eight books in the field of Computer Science & Engineering for McGraw-Hill, Pearson Education, University Science Press, and Scholar Press. He is associated with various International Journals as reviewer and editorial board member. His area of interest Machine Learning Artificial Intelligence, Soft Computing etc. Currently he is the assistant professor at Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University. Prior to this, he has been a faculty at MEC, Coventry University. He has served as a guest editor for many International journals also.

Ankit Chaudhary is major in Computer Science & Engineering and received his Ph.D. in Computer Vision. His current research interests are in Vision based applications, Intelligent Systems and Graph Algorithms. He was a Post-Doc at Department of Electrical and Computer Engineering, The University of Iowa. Currently he is the assistant professor at Department of Computer Science, Truman State University USA. Prior to this, he has been a faculty at Department of Computer Science, BITS Pilani. He has also worked with CITRIX R&D and AVAYA INC in past. He is on the Editorial Board of several International Journals and serves as TPC in many Conferences. He is also reviewer for Journals including IEEE Transactions on Image Processing, Machine Vision and Applications, ACM Transactions on Inter-active Intelligent Systems, Signal Image and Video Processing, Expert Systems with Applications, Robotics and Autonomous Systems and others.

Deepti Mehrotra did Ph.D. from Lucknow University and currently she is working as Professor in Amity school of Engineering and Technology, Amity University, Noida, earlier she worked as Director of Amity School of Computer Science, Noida, India. She has more than 20 year of research, teaching and content writing experience. She had published more than 60 papers in international refereed Journals and conference Proceedings. She is editor and reviewer for many books, referred journal and conferences. She is regularly invited as resource person for FDPs and invited talk in national and international conference. She guided Ph.D. and M.Tech students.