Genetic Algorithm with Modified Reproduction Operators for Grammatical Inference

Hari Mohan Pandey
Computing & Informatics
Bournemouth University, United Kingdom
profharimohanpandey@gmail.com

ABSTRACT

A grammatical inference (GI) algorithm is proposed, that utilizes a Genetic Algorithm (GA), in conjunction with a pushdown automaton (PDA) and the principle of minimum description length (MDL). GI is a methodology to infer context-free grammars (CFGs) from training data. It has wide applicability across many different fields, including natural language processing, language design, and software engineering. GAs is a search methodology that has been used in many domains and we utilize GAs as our primary search algorithm. The proposed algorithm incorporates a Boolean operator-based crossover and mutation operator with a random mask. Here, Boolean operators (AND, OR, NOT, and XOR) are applied as a diversification strategy. A PDA simulator is implemented to validate the production rules of a CFG. The performance is evaluated against state-of-the-art algorithms. Statistical tests demonstrate the superiority of the proposed algorithm over the algorithms implemented in this paper.

CCS CONCEPTS

• Computing methodologies →Artificial Intelligence • Theory of Computation → Formal Language and Automata Theory • Applied Computing → Text Processing.

KEYWORDS

Context Free Grammar, Evolutionary Computation, Genetic Algorithm, Grammar Inference, Minimum Description Length Principle

ACM Reference format:
Hari Mohan Pandey. 2024. Genetic Algorithm with Modified Reproduction Operators for Grammatical Inference. In Proceedings of the Genetic and Evolutionary Computation Conference 2024 (GECCO '24), July 14 – 18, 2024, Melbourne, VIC, Australia. ACM, New York, NY, USA, 4 pages, *https://doi.org/10.1145/3638530.3654151*

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0495-6/24/07.
https://doi.org/10.1145/3638530.3654151

1 INTRODUCTION

Genetic algorithm (GA) is a stochastic search and optimization algorithm based on natural selection and genetics. During the implementation of GA, each individual is assigned a fitness, which is used to measure the quality of a solution. Then, using the reproductory operators the individual population can be modified to a new population. GAs can suffer premature convergence when the diversity of the population decreases over time and the search is stuck. Pandey et al. [1] presented a comprehensive review of the various approaches that have been proposed to prevent premature convergence within GAs. Pandey et al. [2] proposed a bit-mask-oriented GA (BMOGA) that uses a bit-mask-oriented data structure (BMODS) to perform crossover and mutation operations by creating a crossover mask (CM) and mutation mask (MM). To generate an offspring, BMOGA utilizes a Boolean-based procedure, which uses various Boolean operators. BMOGA avoids premature convergence, but the processing time of BMOGA is high. Grammatical inference (GI) or grammar learning is a subfield of machine learning (ML). Here, grammar needs to be learned from examples represented as sentences or programs in some unknown language. GI is an idealized learning procedure for grammar depending on the evidence about the languages [3] [4]. In general, GI can be described as the process of learning grammar from the set of corpora (positive and negative sentences). The solution of GI problem can be successfully utilized to solve problems in computational linguistics, natural language acquisition, pattern recognition, data mining, speech recognition, etc. Hence, it was studied extensively in [3] [7] due to its wide range of applications to solve practical problems in science and engineering. Learning context-free grammars (CFGs) is still a hard problem in ML. The primary challenge of a GI system is maintaining regularity in the data. This paper proposes a GI that utilizes a GA. We refer to the proposed algorithm as GIGA (Grammatical Inference using Genetic Algorithm). GIGA utilizes Boolean operators-based crossover and mutation operators. Employing Boolean operators is an effective decision for maintaining diversity in the population.

The main contribution can be stated as follows: (a) GIGA - a CFG learning system that utilizes a GA is proposed; (b) Boolean operators-based re-production operations (crossover and mutation) have been implemented as a diversification strategy to alleviate premature convergence; (c) A PDA simulator is implemented to validate the best grammar rules that are generated by the learning system; and (d) Statistical performance significance is analyzed.

The rest of the paper is organized as follows: Section 2 presents the proposed methodology; Experimental results are given in Section 3; Concluding remarks are described in Section 4.

2 PROPOSED METHODOLOGY

Figure 1 represents the flowchart of the GA-based learning system for GI.

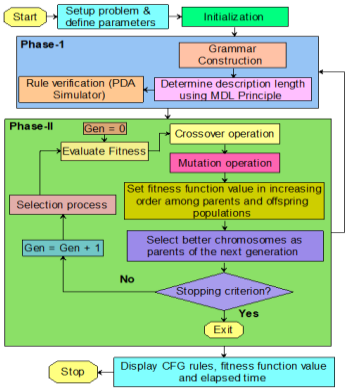

Figure 1. Flowchart illustrating the steps involved in the GIGA.

Phase I is for grammar construction and verification of production rules. Phase II is for the execution of the GA to search for the best production rules. The learning system is composed of several steps that are discussed in the subsections.

3-bit coding	000	000	001	000	011	000	000	110	011	000
SC mapping	S	S	A	S	C	S	S	?	C	S
SC block	SSASC					SS?CS				
UL/UF	UL					UL				
A/D	D					D				
3-bit coding	000	000	100	100	001	000	010	101	110	001
SC mapping	S	S	0	0	A	S	B	1	?	A
SC block	SS00A					SB17A				
UL/UF	UL					UL				
A/D	D					D				
3-bit coding	000	110	110	111	110	010	000	101	001	001
SC mapping	S	?	?	?	?	B	S	1	A	A
SC block	S????					BS1AA				
UL/UF	UF					UL				
A/D	A (S → ?)					D				
3-bit coding	000	000	011	000	001	011	000	001	111	011
SC mapping	S	S	C	S	A	C	S	A	?	C
SC block	SSCSA					CSA7C				
UL/UF	UL					UL				
A/D	D					D				
3-bit coding	000	011	010	000	010	000	110	101	100	000
SC mapping	S	C	B	S	B	S	?	1	0	S
SC block	SCBSB					S710S				
UL/UF	UL					UF				
A/D	D					A (S → 10S)				
3-bit coding	001	111	001	000	000	011	001	010	100	001
SC mapping	A	?	A	S	S	C	A	B	0	A
SC block	A?ASS					CAB0A				
UL/UF	UL					UL				
A/D	D					D				
3-bit coding	000	001	001	110	011	001	010	011	100	001
SC mapping	S	A	A	?	C	A	B	C	0	A
SC block	SAA?C					ABC0A				
UL/UF	UL					UL				
A/D	D					D				
3-bit coding	010	001	100	110	011	010	111	100	000	010
SC mapping	B	A	0	?	C	B	?	0	S	B
SC block	BA07C					B70SB				
UL/UF	UL					UL				
A/D	D					D				
Equivalent context free grammar (CFG): < {S}, {1, 0}, {S → ?, S → 10S}, S>										

Figure 2. A 3-bit representation is used as 2-symbols (1 and 0) are represented in the language. SC is split into block sizes of (equal to production rule length). UL, UF, A, and D respectively indicate useless, useful, accept, and discard production rules. PDA simulator is used for acceptance or rejection of production rules.

2.1 Initialization and Symbolic Chromosome Representation

GIGA implements the first step only once, at the beginning of the execution. A random binary chromosome comprising of sequence of 0's and 1's is generated. The binary chromosomes are then mapped to terminals and non-terminals to generate symbolic chromosomes as depicted in Figure 2 (see Figures 2 and 3 in Appendix 1.1 in supplementary data).

2.2 Fitness Function

Equation (1) is used to determine the fitness function value (F_v).

$$F_v = max \sum Q\big((P_s + N_r) - (N_a + P_r)\big) + (2 * Q - PRL) \quad (1)$$

S.T.

$P_s + N_a \leq$ Total positive strings in the corpus.

$N_r + P_r \leq$ Total negative strings in the corpus.

PRL: Production rule length.

Q: A constant.

Where P_s: accepted positive strings, N_r: rejected negative strings, N_a: accepted negative strings, P_r: rejected negative strings, PRL: production rule length, and Q: constant.

2.3 Reproduction Operators

In the GIGA, Boolean operators (AND, OR, NOT, and XOR) based crossover and mutation operations have been implemented as a diversification strategy so that diversity of the population could be maintained throughout the search for an optimal grammar. (($Terms$: P_1, P_2: parent population; $P_{1(updated)}$, $P_{2(updated)}$: up-dated parent populations are generated after applying Boolean operators on P_1/P_2 and RM; CH_1: child population after crossover operation; $CH_{1(updated)}$: child population after mutation operation; RM: random mask).

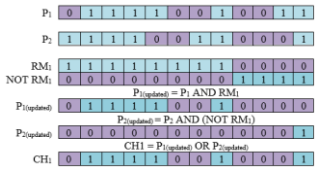

Figure 3: Boolean operators-based crossover operation.

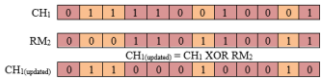

Figure 4: Mutation operation by creating an *RM* and then applying the 'XOR' operation to generate the offspring.

Figure 3 illustrates a crossover operation where parent populations P_1 and P_2 are selected. Figure 4 presents a mutation operation where Boolean XOR operation is performed to maintain population diversity.

2.4 Generalization

Let $C_L = \{c_1, c_2, \ldots, c_i, \ldots c_L\}$ is a set of corpora of length L where c_i is i^{th} string of $C_L, \forall 1 \leq i \leq L$. We can define a partial grammar set $G_k = \{g_1, g_2, \ldots g_j, \ldots g_k\}$ where g_j is a j^{th} production rule for training data. In a non-standard way, G_k represent a set of classes where $g_j = \{g'_j \rightarrow w_1\} \forall g_j \in G_k \; and \; \forall g'_j \in g_j$. In other

words, g_j shows a set of production rules where g_j' is on the left-hand side of the production rules. MDL principle is used for generalization and data regularity. Two operations – merge and construct are implemented to handle temporary production rules. Algorithm 1 presents the steps of generalization.

Algorithm-1: Generalization ()

Terms: g_i : i^{th} production rule corresponding to g_i' is present (production rules in Backus Naur form (BNF)), G: a partial grammar contains a nonstandard set of classes g_i , D_L: description length, and $D_{L(M/C)}$: description length after merge or construction operations.

```
1    Set a separate class for each string of training set.
     g1 ← {g1' → w1}, g2 = {g2' → w2}, ....
2    Merging G ← g1 ∪ g2 ∪ ...
3    Constructing gnew ← {gnew' → gl'gk'}
4    Set DL ← DL(trainingset) + DL(G)
5    Set ΔDL(M) ← difference in DL after merging.
6    Set ΔDL(C) ← difference in DL after construction.
7    If (DL(M/C) < DL) Then
8        Set DL ← DL(M/C)
9    Else
10       Set DL ← DL
```

2.5 Rule Validation

A PDA-based method is implemented for production rules validation as shown in Algorithm 2.

Algorithm-2: Rule_Validation (Str, Stack)

Terms: S_T: Symbol at the top of stack, S_F: First symbol of top of stack, TOP: Top of stack, I_{str}: input string, P_R: Set of production rules, S: Stack, EOStr: End of string, EOS: End of stack, P_{Rule}: production rule starting with start symbol, T_{str}: Temporary string, T_{STACK}: Temporary stack.

```
1    Begin
2        Set S ← "S$"
3        Set Istr ← Str
4        Set TOP ← S'
5        Set ST ← Return 1st symbol from Istr
6        Set SF ← Return 1st symbol from TOP
7        If S_Overflow () Then
8            Return "Overflow"
9        Else If (EOStr && EOS) Then
10           Return "Istr accepted"
11       Else If (SF = Terminal && ST = SF) Then
12           Delete SF from Istr
13           Delete SF from TOP
14       Else If (SF = Non_Terminal) Then
15           For ∀PRule ∈ PR
16               If (Null Production Rule) Then
17                   Delete SF
18               Else
19                   Delete SF
20                   Copy the right-hand side of the
                     production ST to TOP
21           End For
22       End If
23       Set Result ← Rule_Validation (Tstr, TSTACK)
24       If (Results = 1) Then
25           Return "String is Accepted"
26       Else
27           Return "String is Rejected"
28   End
```

2.6 GA Based Grammatical Inference

Algorithm-3 presents the pseudocode of GIGA.

Algorithm-3: Grammatical Inference Genetic Algorithm

Input: BC: Binary chromosome; PS: size of population, CS: size of chromosome, CR: crossover probability, MR: mutation probability, PRL: production rule length, Th: threshold, TR: total run, P_1/P_2: parent populations, RM: random mask, CH: child population and $G_{max:}$ maximum number of generations.

Output: CFG production rules, F_v: fitness value and T: processing time.

```
1    Begin
2      Set BC ← Generate random initial BC of PSIZE
3      Grammar constructions (Repeat 4 – 6)
4        Map BC to SC using sequential structuring and BNF
         to get CFG.
5          Rule_Validation (Str, Stack)      //Algorithm-2
6          Generalization ()                 //Algorithm-1
7      While (Fv > Th) and (TR = Gmax) Then
8          Selection operation through the roulette wheel
9          Boolean operator-based crossover operation.
10             Set P1 ← parent population
11             Set P2 ← parent population
12             Set RM← generate a random mask
13             Set P1(updated) ← P1 AND RM
14             Set P2(updated) ← P2 AND (NOT RM)
15             Set CH1 ← P1(updated) OR P2(updated)
10         Boolean operator-based mutation operation.
11             Set RM ← generate a random mask
12             Set CH1(updated)← CH1 XOR RM
13         Determine Fv using Equation (1)
14         Set Fv ←Arrange fitness value
15         Selection of parents for the next generation.
16     End While
17     Display CFG rules, Fv and T.
18   End
```

3 Experimental Results

Experiments have been conducted using Java programming on Eclipse IDE, Intel Core™ 2 processor (2.8 GHz) with 4 GB RAM. To select the corpus, the strings of terminals were generated for the length, L, starting with L = 0 and gradually increasing L to get the required size to represent the language features from the given language. The corpus of 50 strings was sufficient to represent the language features selected to perform the experiments. Four languages L1, L2, L3 and L4 were used (L1 = {Palindrome over {a + b}}, L2 = {Palindrome over {a + b + c}}, L3 = {(0*+2*)1 over

(0+1+2)*} and L4 = {ab* U cb* over (a + b + c)*}) for the experiments. The tuning process involved 5 control factors (population size (PS), production rule length (PRL), chromosome size (CS), crossover rate (CR), and mutation rate (MR)) with 2 levels. Equation (2) is used to determine SNR as an objective function where the control factors combination which gave the smaller SNR value was selected (see Figure 6 in Appendix 1.2 in supplementary data).

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

The best results were obtained using the following settings: PS: 120, PRL: 5, CS:120, CR: 0.9, MR: 0.8, and Gmax: 500. GIGA's performance is compared with BMOGA [2], ITBL [8], GAWMDL [6] and GAWS [9]. These algorithms were implemented to avoid premature convergence. GIGA was compared with ITBL [8] because it is a CFG learning algorithm, while BMOGA [2], GAWMDL [6], and GAWS [9] were proposed for handling premature convergence where the domain of inquiry was CFG learning. The result reveals that GIGA was capable of inferring CFGs as shown in Table 1. Production rules were validated with the best-known CFGs [2][9]. The standard CFG representation [10] [11] was used for representing the grammars as shown in Table 1.

Table 1: CFGs using GIGA.

L-id	CFG Representation
L1	<{S}, {a, b}, {S→bSb, S →aSa, S→ ∈}, S>
L2	<{S}, {a, b, c}, {S→cSc, S→bSb, S → aSa, S→ ∈}, S >
L3	<{S, M, L, K}, {0, 1}, {S→L, S→K, L→1, L→2L, K→0K, K→1}, S>
L4	< {S, I, L}, {a, b, c}, {S→bL, S→aL, S→ ∈, L →b, L→Sb, I→a, I→Sa} S>

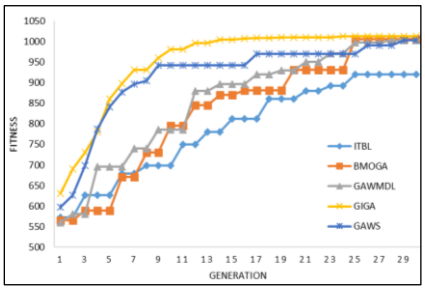

Figure 5: Fitness Vs. generations chart for algorithms.

In GIGA, a 2-point cut cyclic crossover operator and inverted mutation with an RM and then an XOR operator have been implemented to maintain population diversity. This reproduction operator combination showed a promising effect on the computer simulation. Algorithm-1 and 2 handled the complexity of search space and validated production rules successfully. The fitness vs generation chart is shown in Figure 5. GIGA has outperformed other algorithms and ITBL showed worse performance. Overall, the proposed GIGA has outperformed the other algorithms in terms of both computational cost and success rate (Appendix 1.2 in supplementary data). Statistical results confirm the superiority of the GIGA (Appendix 1.3 in supplementary data). It was noted that Boolean operators-based crossover and mutation operators have successfully maintained population diversity during a genetic evolution which allowed the GIGA to reach the global optimum successfully without getting trapped at local optimum convergence.

4 Conclusions

In this paper, a GIGA is presented for CFG induction. Exploration and exploitation are the keys to the success of any search algorithm [7]. In GA, it is achieved through the selection and recombination process. We applied Boolean operators-based crossover and mutation operations. which maintained the population diversity during the evolution. We also utilized the MDL principle for handling the complexity of the search space as it removed unwanted temporary production rules. The results showed that GIGA can infer the CFGs and greatly improve the performance. The performance of GIGA was tested against ITBL, BMOGA, GAWMDL and GAWS. These algorithms were mainly proposed to alleviate premature convergence by maintaining population diversity. Further, GIGA was compared with ITBL. ITBL uses a GA, and it was proposed for the CFG induction. Experimental results confirmed that GIGA outperformed other algorithms.

REFERENCES

[1] Hari Mohan Pandey, Ankit Chaudhary, Deepti Mehrotra. 2014. A comparative review of approaches to prevent premature convergence in GA, Applied Soft Computing, Volume 24, 2014, Pages 1047-1077, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2014.08.025

[2] Hari Mohan Pandey, Ankit Chaudhary, Deepti Mehrotra. 2016. Grammar induction using bit masking oriented genetic algorithm and comparative analysis, Applied Soft Computing, Volume 38, Pages 453-468, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2015.09.044.

[3] Colin De La Higuera. 2010. Grammatical Inference: Learning Automata and Grammars. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139194655

[4] G. K. Pullum. 2012. Learnability, Hyper Learning, and the Poverty of the Stimulus. In Proceedings of the Twenty-Second Annual Meeting of the Berkeley Linguistics Society: General Session and Parasession on The Role of Learnability in Grammatical Theory (1996), pp. 498-513

[5] King Sun Fu. 2012. Syntactic Pattern Recognition, Applications (Vol. 14). Springer Science & Business Media.

[6] H. M. Pandey, A. Chaudhary, D. Mehrotra and G. Kendall. 2016. Maintaining regularity and generalization in data using the minimum description length principle and genetic algorithm: Case of grammatical inference. Swarm and Evolutionary Computation 31 (2016): 11-23.

[7] M. Črepinšek, SH Liu, and M. Mernik. 2013. Exploration and exploitation in evolutionary algorithms: A survey. ACM computing surveys (CSUR) 45.3, pp 1-33.

[8] M. Jaworski and O. Unold. 2007. Improved TBL algorithm for learning context-free grammar". In Proceedings of the International Multiconference on Computer Science and Information Technology, ISSN 1896 -7094, pp. 267 - 274.

[9] Hari Mohan Pandey, Marcello Trovati, Nik Bessis. 2021. Statistical exploratory analysis of mask-fill reproduction operators of Genetic Algorithms, Applied Soft Computing, Volume 102, 107087, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2021.107087

[10] A. K. Yadav. 2023. Theory of Computation. Blue Rose Publishers.

[11] O. Kosheleva and V. Kreinovich. 2023. Why Chomsky Normal Form: A Pedagogical Note. In: Ceberio, M., Kreinovich, V. (eds) Decision Making Under Uncertainty and Constraints. Studies in Systems, Decision and Control, vol 217. Springer, Cham. https://doi.org/10.1007/978-3-031-16415-6_10.