Developing Genetic Algorithm Library Using Java for CFG Induction

N.S. Choubey, Hari Mohan Pandey
Computer Engineering Department, MPSTME, Shirpur (MS), India
Email: nschoubey@gmail.com , hari04top@yahoo.co.in

M.U. Kharat
PLIT&M, Buldana(MS), India
Email: mukharat@rediffmail.com

Abstract

Grammar Induction is the process of learning grammar from training data of the positive (S+) and negative (S-) strings of the language. The paper discusses the approach to develop a library for inducing the context free grammar using Genetic Algorithms. Genetic Algorithm used for the induction library produces successive generations of individuals' chromosome, computes their fitness value in every step of generation and finally select the best out of the total number of generation or when the termination condition (threshold) is meet. The library also deals with the issues in implementation of the algorithm, chromosome representation, evaluation, selection and replacement strategy and the genetic operators for crossover and mutation. The paper also addresses the solution of the problem like useless production, left recursion, left factor, and unit production etc. The library has been implemented and the results obtained for the set of various problems like balanced parenthesis, two symbol palindromes and equal number of 0s and 1s are presented.

Keywords: Context Free Grammar, Genetic Algorithms, Grammatical Inference, Machine Learning, Induction Library etc.

1. Introduction

The field which has been used as a problem solving technique is known as Evolutionary techniques, which is similar to machine learning recombination methods. There are many flavors of evolutionary computing approach available but most evolutionary computing approaches hold in common. These approaches try and find a solution to a particular problem, by recombining and mutating the individuals to get the possible solution [1]. Author uses some encoding or decoding technique known as mapping the data to obtain the solution. At the initial stage, we generate the population randomly. Fitness has been assigned to each individual for the verification and validation of the problem being solved. Since, author has used the concept of Genetic Algorithm, which works on selection, crossover and mutation operators. One can apply the concept of replacement, which is applicable usually by generations of new individuals.

This paper presents the CFG Induction Library Using Java to address various issues of context free grammar induction using GA. In this paper author has given the complete library

to induce CFG using set of corpus using crossover and mutation operators over the generated chromosome.

The organization of paper is as follows: the section II gives the detailed description about the problem definition, section III is given to understand the proposed method, section IV covers the detailed about the implementation methodology author has used. Section V is given for the experimental setup which includes the test data which authors are using for the present work, GA parameters and the results. The last section covers the conclusion and future direction of the present work done.

2. Problem Definition

In this paper authors are presenting an induction library to inference the Context Free Grammar using GAs. The learning technique author discussed so far is with informant, where a language acceptor is constructed so as to accept all positive example (S^+) and reject all the negative examples (S^-).

In Peter Wyard [2] gave the idea of different grammatical representations. After completion of experiments we will get various production rules in the form of Context Free Grammar. The CFG which author will get after experiments will be in Bakus Naur Form.

In formal language theory, a context free grammar is a grammar in which every production rule is of the form:

$$\alpha \rightarrow \beta$$

Where,

α : A single non-terminal symbol and β : A string of terminals and/or non-terminals

3. Proposed Method

Author proposes a genetic algorithm, with some assistance can solve the problem. To induce the CFG authors will train on set of samples. Over time, the set of sentences will increase, which also increases complexity of the system. By using GAs author will create the random population of Grammar using grammar designing steps. Then computing the fitness of the grammar in the whole population and get the best from the whole set. The best individual with the fitness greater than the required threshold is found or the given number of generation is completed. After that author will use the operator crossover and mutation to create new population and note down the crossover and mutation rate by using appropriate method. Again calculate the fitness of the new constructed population. Finally, merge the population and update the best individual by using the correct replacement method.

4. Implementation Methodology

In this section, author describes the methods used to induce a grammar for set of sentences. Although the approach is based on the idea of a genetic algorithm as originally presented in [3] and latter on in [4], in this paper author has made a few significant adaptations that make the process easier for the grammar induction.

4.1 Grammar Induction Process

The process of learning a grammar from set of samples is known as Grammar Induction or Language Learning. Various algorithms are given for learning regular language. These algorithms are nothing but used for largest class of languages, which can be efficiently learned. This paper focuses on grammatical inference i.e. inference of formal languages such as of the Chomsky hierarchy from positive and negative sample strings. In [2] Wyard addressed the different representations and experimental results shows that an evolutionary algorithm using Context Free Grammar (BNF). Wyard [2] also explore the concepts and various issues such as the representation of the grammars, and method for the evaluation of the chromosome.

Context free grammar learning requires more information about a set of positive and negative sample for example a set of skeleton parse trees, which makes them a more challenging task to induce grammar. In a broader sense, a learner has access to some sequential or structured data and is asked to return a grammar that should in some way explain such data. Parsing according to a grammar amounts to assigning one or more structures to a given sentence of the language the grammar has been defined. Author will surely get ambiguous grammar if there are sentences with more than one structure, as generally used in case of natural language. Parsing can be used as a search process that looks for correct structures for the input sentence. If author can establish some kind of preference among the set of correct structures, the process can be regarded as an optimize one. The idea given here suggests considering evolutionary programming techniques, which are acknowledged to be practical search and optimization methods [5].

There are verities of practical applications of Grammar Induction one can see outside the field of theoretical linguistics, such as structural pattern recognition [6] [7] (in both visual images and more general patterns), information retrieval, automatic computer program synthesis, bioinformatics etc. Syntactic processing has always been paramount to a wide range of applications, includes machine translation, speech recognition and the like. Hence, natural language syntax has always been one of the most active research areas in the field of language technology [6].

All of the typical pitfalls in language, for example ambiguity, recursion, and long distance dependencies are prominent problems in describing syntax in a computational context. The field of evolutionary computing, which we are applying is a problem solving

techniques. It is similar in intent to the Machine Learning recombination methods. Most evolutionary computing approaches hold in common that they find solution to a particular problem by recombining and mutation individuals in a society of possible solutions. This provides an attractive technique for problems involving large, complicated and non-linearly divisible search spaces.

4.2 Induction Library Development

To implement the "CFG Induction Library" author requires a proper support from the operating system to control the implementation of java. The control over the classes and methods used requires a proper communication through the system call, which should be supported by java virtual machine. The approach used to develop the induction library is explained below:

4.2.1 Process of Data Mapping

The process of mapping plays a very important role in language learning process. It is nothing but a basis for structuring the chromosomes. Suppose V be any set of terminals or non-terminal and let B be the equivalent binary for the terminals/non-terminals. Then to get a sequential structured chromosome in the form of random sequence of 0's and 1's author has to use a function, which will map the element of set B to element of set V. We can represent it as:

$$f : B \rightarrow V$$

Where, f be the function used to map sequential block of 0's and 1's to terminals or non terminals.

To decoding the grammar maps the random chromosome according to bit sequence based on the number of terminals available in the given sample. Here, for the induction library, we have used mapping from bit representation to symbolic representation symbol in 3-bit, 4-bit respectively.

4.2.2 Developing Chromosome Structure

In the field Genetic algorithm the selection of chromosome structure is an important decision. As far as grammar induction is cornered, the parameters required by the system is unknown, therefore the chromosome will be of variable length. The variable length can make the operation of the crossover operator less straightforward than before. For the induction library there are two approaches chosen by the author.

1. Generate random string consisting of 0's and 1's of a specific length.
2. Then partitioned the string in to the sequential blocks of equal length to get the desired number of production rules by mapping them to the terminals and non-terminals.

Algorithm-1: Steps used for construction of grammar from each individual chromosome

1. Initialize string of random 0's and 1's
2. Chromosome ← generate random sequence of 0's and 1's up to given size.
3. CFG rule extraction using Backus Naur form and chromosome.
4. Eliminate left recursion
5. Remove multiple production rules from the same set of terminals using rules for left factoring.
6. Stop.

After applying the above given approach author will get the production with variable length. Then the resulted productions are treated as ordinary methods such as left factoring and left-recursion removal in order to get the resultant production. To understand the working of the algorithm-1 and how to map the data let us consider an example:

Table 1: Mapping and Generating Equivalent Symbol

S.N.	Test Data	Binary Equivalent
1.	S	000
2.	A	001
3.	B	010
4.	C	011
5.	a	100
6.	b	101

Now, select the chromosome (length =120) as:

0101010001101000100000000000100010110100011110100111111100100001000101101000111110011011101111010011110000000111101111

Sequentially partitioned the chromosome into block of 3-bit as:

010|101|000|110|100|010|000|000|000|010|001|011|010|001|111|010|011|111|110|010|000|100|010|110|100|011|111|100|110|111|011|110|100|111|100|000|000|111|101|111|

Apply the mapping given in table-1

010=B|101=b|000=S|110=?|100=a|010=B|000=S|000=S|000=S|010=B|001=A|011=C|010=B|001=A|111=?|010=B|011=C|111=?|110=?|010=B|000=S|100=a|010=B|110=?|100=a|011=C|111=?|100=a|110=?|111=?|011=C|110=?|100=a|111=?|100=a|000=S|000=S|111=?|101=b|111=?|

**Symbol "|" represents wall.
Therefore, equivalent symbolic chromosome (length=40) is:

BbS?aBSSSBACBA?BC??BSaB?aC?a??C?a?aSS?b

The method which author has discussed so far, for structuring chromosome and mapping the chromosome into symbolic form known as sequential structuring of chromosome [8].

4.2.3 Developing Parser

Author has implemented a parser for the "Induction Library" where authors are dealing with the problems like removing useless production rules, removing unit productions, removing non-required epsilon, dealing with problem called as left factor and left recursion etc. Induction library contains a class "Check_Parser". The parser Check_Parser basically works on Recursive Decent Parsing technique. This parser not only just deal the problem given in this section but also it checks the correctness of the chromosome by showing the message chromosome is accepted or not accepted. For checking the validity of the chromosome we will pass the chromosome as input and then we will call a method "print_CFG" to extract production rules from the chromosome by applying the methodology given in the previous section.

For the chromosome we have selected in our example we will get the following rules:

Total number of rules: 3

S->aaM M->SM M->?

4.2.4 GA Approach for CFG Induction Library

This section describes the procedure used for Grammar Induction from a set of corpus constructed from a set of positive and negative string for the grammar to be constructed. Here fitness function is based on the nature of the string supplied in the corpus. Corpus length is taken as 50 which include positive strings and negative strings. Acceptance and rejection of the strings in the given corpus gives the fitness of an individual grammar [9]. Positive weight is added for every acceptance of the positive string and rejection of every negative string whereas penalty of the positive weight is given for every rejection of positive string and acceptance of every negative string. An additional factor of the maximum number of expected rules is used to control the number of rules required in the resultant grammar.

Algorithm-2: CFG Induction using genetic algorithm

Input: random sequence of 0's and 1's up to size of chromosomes (Array p).
Output: the optimal search time (t) and the set of grammar rule (R).
Fitness function: `((n (AP) +n (RN))-((AN) + (RP))*PW+PW*CL-(PW-MR)`

Where,

CL Corpus Length
PS Positive String
PP Penalty of Positive String
MR Addition factor of the maximum number of expected rule
PW Positive Weight
NS Negative String
PN Penalty of Negative String

1. Input (n, p)
2. Input (popsize, maxgen, poss_cross, poss, mut, prnum)
3. Population ← random population of the grammar (popsize)
4. Fitness ← ((n (PS) +n (NS))-((PN) + (PP))*PW+PW*CL) - (PW-MR)
5. Repeated step 5 to 8
   Either: Best individual with > required threshold
   OR : Number of generation completed
6. New Population ← population after crossover and mutation (figure-1)
7. Fitness ← Fitness of newly construed individual
8. Merge both populations
9. Update (best individual in the population)
10. Display (t)
11. Stop.

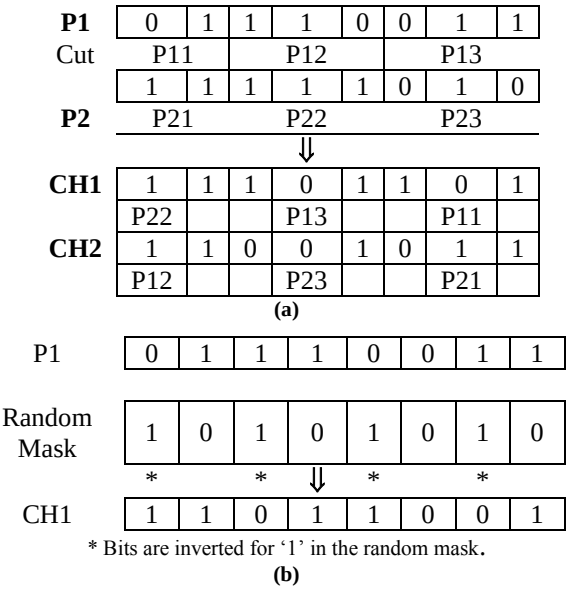

Fig.1: a. Crossover and b. Mutation methods used [10]

4.2.5 Java Classes Used

This section describes all the classes developed for inducing Context Free Grammar from the set of sample set. There are total five different classes has been developed by author for the CFG induction. The name of each class is given in the table below with small description of each.

Table-2: Java Classes Used for CFG Induction Library

S.N.	Classes Used	Description
1.	SYM_TAB	Declare variables in the form of symbols and binary.
2.	VAR_TER	Used to represents the terminals and non-terminals in binary 3-bit or 4-bit representation
3	Chromosome	Used to get random chromosomes
4.	Check_Parser	Used to check the correctness of the grammar for a set of strings.
5.	Gen_Ope	This class basically deals with operations of genetic algorithm which we have declared earlier. In this class we have applied the operators called as crossover and mutation.

5. Experimental Setup

To develop Context Free Grammar Induction library author has selected java. The experiment is carried out to collect result of first ten successful run for the grammar induction of various languages over the combination of crossover and mutation operators. The work done in this paper can run on sequential as well as parallel environments. Java is one of the best solutions for this. There are many parallel programming instruction presents in java which author can use for the parallel implementation of GAs. Java provides the facilities of machine independent method of distributing the code to perform the computation on different machine. To perform the experiment we have used JDK 1.6 on Intel Core ^{TM}2 CPU with 2.66 GHZ and 1 GB RAM. Other parameters which also play an important role in the experimental process are explained below.

i. Test Data: To understand the working of Genetic Algorithm, it is tested on various languages. The problem author has selected are common test cases for effectiveness of grammatical inference methods. The test languages are given in the table as follows:

Table 3: Test Language

L-id	Description
L-1	Balanced parentheses problem
L-2	Two symbol palindrome over {a, b}.
L-3	(10)* over {0,1}

The problem which author has selected for the implementation is based on the set of positive and negative strings.

ii. GA Parameters: The Genetic Algorithm parameters we have selected for the implementation of the problem are given in the table below:

Table 4: Parameters for Genetic Algorithm

S.N.	Parameters	Size/Value
1.	Population Size	50
2.	Chromosome Size	240
3.	Corpus size	50
3.	Maximum Generation	10
4.	Probability of Crossover	0.9
5.	Probability of Mutation	0.8
6.	Selection Strategy	Roulette Wheel Selection

iii. Results

Language-1: Balanced Parenthesis Problem

Table-5: Showing the Generation of Chromosomes

Generation	Worst	Average	Best
1	490	514.78	539
2	507	518.09	481
3	512	519.19	851
4	514	520.20	992
5	515	521.05	992
6	515	525.63	992
7	515	530.71	992
8	515	538.33	992
9	532	554.68	1013

Best Chromosome is
011001010011010110000111111111100111000110010011100100001100110110101100001000010010111001011011010110010100001001011100000100001101000011010111001111100001110111000111111110001000

The Best over generations is

S->? S->(A)S A->S

Total number of rules: 3
Its fitness is 1013

Total time required in milliseconds : 116625
Days : 0
Hours : 0
Minutes : 1
Seconds : 56
Mili-Seconds : 625

Language-2: Two Symbol Palindrome over {a, b}

Table-6: Showing the Generation of Chromosomes

Generation	Worst	Average	Best
1	493	510.22	531
2	513	515.3	534
3	515	517.94	590
4	515	521.76	590
5	515	533.04	772
6	530	554.08	911
7	533	563.20	911
8	535	610.74	950
9	590	644.34	950
10	590	682.42	1013

Best Chromosome is
0010011001100000101010100010100000101010001010001001111111010100001000001000100111010000010100100001100100010000000110100101000001111100100010010100000101010001001000110101010100

The Best over generations is

S->? S->aSa S->bSb

Total number of rules: 3
Its fitness is 1013

Total time required in milliseconds : 203344
Days : 0
Hours : 0
Minutes : 3
Seconds : 23
Mili-Seconds : 344

Language-3: (10)* over {0, 1}

Table-7: Showing the Generation of Chromosomes

Generation	Worst	Average	Best
1	350	504.58	554
2	501	512.87	554
3	509	516.57	554
4	515	518.74	554
5	515	521.43	554
6	515	526.33	572
7	515	546.61	970
8	532	571.71	970
9	554	592.23	970
10	554	626.38	1014

Best Chromosome is

00100111011010000001000111010010100001110100011101101001110011111000000
00010001011101000000111010000111100010000000110101011010000100010001110
11011000000101001011000000110111111110

The Best over generations is

S->? S->10S

Total number of rules: 2
Its fitness is 1014

Total time required in milliseconds	:	89639
Days	:	0
Hours	:	0
Minutes	:	1
Seconds	:	29
Mili-Seconds	:	639

Now, on the basis of the results authors have achieved for the languages L1 to L3 author can represent the summary of the result. The result set is given in the table-8 below:

Table-8: Resultant Grammar with Fitness Value

L-id	Gen	FV	EG
L1	09	1013	S->?, S->(A)S , S->S
L2	10	1013	S->?, S->aSa , S->bSb
L3	10	1014	S->?, S->10S

L-id: Language-id, FV: Fitness Value
G: Generation EG: Equivalent Grammar
Note: Epsilon is denoted by "?"

The grammars shown in the table-8 are the grammar equivalent to the chromosome shown above. The grammar with fitness value shown in table-8 will accept all the positive examples and rejects the negative example as considered for the experiment. The grammar out of 10 generation (or threshold reached) is represented as <V, T, P, S> where V is finite set of Variables, T is finite set of Terminals, P is finite set of Production rules and S is a starting Variable.

6. Conclusion and Future Enhancement

In this paper author has given the detailed idea for grammar induction using genetic algorithm. The contribution of the work is as follows:

a) Basic concepts of Inducing Grammars.
b) Mapping methodology and structuring chromosome.
c) CFG learning technique from the chromosome.
d) Solution of various problems with fitness value and time.
e) Java's classes used for inducing the CFG from the set of corpus (Positive and Negative)

As a future work we can do the following:

a) Dealing with the problem of local optimum which we are facing during the implementation of the library.
b) Implementing the library on different topologies.
c) Similar approach can be applied Natural Language Grammar Induction process.
d) Similar approach can be applicable for parallel implementation of GAs.

References

[1] Afra Zomorodian, Context-free language induction by evolution of deterministic pushdown automata using genetic programming. In E. S. Siegel and J. R. Koza, editors, Working Notes for the AAAI Symposium on Genetic Programming, pages 127-133, MIT, Cambridge, MA, USA, 10--12 November 1995. AAAI.

[2] Wyard, P., Representational Issues for Context-Free Grammar Induction Using Genetic Algorithm in Proceedings of the 2nd International Colloquim on Grammatical Inference and Applications, Lecture Notes in Artificial Intelligence, Vol 862, pp. 222-235, 1994.

[3] J. H. Holland, Adaptation in natural and artificial system, University of Michigan Press, Ann Arbor, 1975.

[4] D.E. Goldberg, Genetic Algorithms in search, optimization, and machine learning, Addison-Wesley, Boston, 1989.

[5] F. Javed, B. R. Bryant, M.Crepinek, Mernik, Sprague, "Context-free Grammar Induction using Genetic Programming", ACMSE, Huntzville, 2004.

[6] Evolutionary Computing as a Tool for Grammar Development, Guy De Pauw, CNTS – Language Technology Group, UIA – University of Antwerp, Antwerp – Belgium, E. Cant´u-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 549–560, 2003.,c_Springer-Verlag Berlin Heidelberg 2003.

[7] Genetic Programming with Incremental Learning for Grammatical Inference Ernesto Rodrigues and Heitor Silvério Lopes, Graduate Program in Electrical Engineering and Computer Science, Federal University of Technology – Paraná, Av. 7 de setembro, 3165 80230-901, Curitiba, Brazil.

[8] Sequential Structuring Element for CFG Induction Using Genetic Algorithm, Dr. N.S. Choubey, Head CE Department, MPSTME, M.U. Kharat, Institute of Engineering, Bhujbal Knowledge City Nashik, India.

[9] Choubey N.S. and M.U. Kharat, 2009. "Grammar Induction and Genetic Algorithm: An overview " Pacific Journal of Science & Technology 10(2): 884-888

[10] Choubey N.S., Kharat M.U. "Stochastic Mutation for Grammar Induction Using Genetic Algorithm, Electronic Computer Technology (ICECT), 2010 International Conference. pp. 142-146, 7-10 May 2010.

[11] Kharat MU, N.S. Choubey, "Sequential Structuring element for CFG induction using genetic algorithm", International journal of computer application, 1(1) 12-16, February 2010, published by foundation of computer science.