Sequential Structuring Element for CFG Induction Using Genetic Algorithm

N. S. Choubey
MPSTME, NMIMS University,
Shirpur, Maharastra, India

M. U. Kharat
Institute of Engineering
Bhujbal Knowledge City Nashik, India

ABSTRACT
This paper investigates the induction of Context free Grammar with genetic algorithm. The genetic algorithm is not very effective at this [1]. To overcome this problem we investigate combined effect of two methods for structuring the chromosomes. The first is to bias the distribution of Non-terminals in the chromosome at the time of chromosome generation as well as updating. The latter approach is sequential structural mapping from chromosome to grammar. It is shown that biasing the chromosome in this fashion produces extremely fast convergence as shown in the result section. Grammars are an extremely general and useful tool with many applications. These include the higher levels of signal processing, such as pattern recognition. However, the application of grammars is limited by the algorithms we can apply to infer them from samples of data. The main contribution of this paper is the effective decoding of the context free grammar from chromosomes with the distinct approaches mentioned.

Categories and Subject Descriptors
F. [MATHEMATICAL LOGIC AND FORMAL LANGUAGES]- **F.4.2** Grammars and Other Rewriting Systems (D.3.1)- Grammar types (context-free)

General Terms
Experimentation, Languages, Theory.

Keywords
Genetic Algorithm, Grammar Induction, Pattern Recognition, Context Free Grammar, Informant Learning.

1. INTRODUCTION
The Genetic Algorithms (GA's) were invented by John Holland in the 1960s. Wyard [2] explored the impact of different grammar representations and experimental results show that an evolutionary algorithm using standard context-free grammars (BNF) outperformed other representations. The section II of the paper discusses the brief overview of the Genetic Algorithm, the details of the structure implementation for the chromosome along with an example is covered in section III, whereas the section IV and section V includes the details of the implementation done by the authors for CFG induction with Genetic Algorithm. Section 6 of the paper displays the result obtained for eight different set of languages.

2. STRATEGIES FOR GRAMMAR INDUCTION AND GENETIC ALGORITHM

2.1 Role of Genetic Algorithm
Genetic Algorithms (GA's) were invented by John Holland in the 1960s. In contrast with Evolution Strategies and Evolutionary Programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be utilized into computer systems. Holland's 1975 book "Adaptation in Natural and Artificial Systems" presented the Genetic Algorithm as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GA. Holland's GA is a method for moving from one population of "Chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a kind of "natural selection" together with the genetics-inspired operators of Crossover, Mutation, and Inversion. Each Chromosome consists of "Genes" (e.g., bits), each Gene being an instance of a particular "allele" (e.g., 0 or 1). The Selection Operator chooses those Chromosomes in the Population that will be allowed to reproduce, and on average the fitter Chromosomes produce more offspring than the less fit ones. Crossover exchanges subparts of two Chromosomes, roughly mimicking biological recombination between two single-Chromosome ("haploid") organisms; Mutation randomly changes the allele values of some locations in the Chromosome; and Inversion reverses the order of a contiguous section of the Chromosome, thus rearranging the order in which genes are arrayed.

2.2 Grammar Induction Process
The Grammar Induction (or Grammar Inference or Language Learning) is the process of learning of a grammar from training data. Various algorithms exist for learning regular languages, which represent the largest class of languages which can be efficiently learned. Inductive inference involves making generalizations from examples. The generalizations sought in this research are languages. The focus is on grammatical inference, i.e. the inference of formal languages such as those of the Chomsky hierarchy from positive (and negative) sample strings.

Wyard[2] explored the impact of different grammar representations and experimental results show that an evolutionary algorithm using standard context-free grammars (BNF) outperformed other representations. This performance differential was attributed to the larger grammar search space of the other representations, which was a consequence of them having a more complex grammar form.

In the conventional grammatical induction, a language acceptor is constructed to accept all the positive examples. Learning from

positive examples is called text learning. A more powerful technique uses negative samples as well. This is learning with an informant. In informant learning, the language acceptor is constructed so as to accept all the positive examples and reject all the negative examples. By comparison, context-free grammar (CFG) learning requires more information than a set of positive and negative samples (e.g. a set of skeleton parse trees) which makes them a bigger challenge for grammatical inference. In a broad 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.

Grammar induction has several practical applications outside the field of theoretical linguistics, such as structural pattern recognition [3][4] (in both visual images and more general patterns), automatic computer program synthesis and programming by example, information retrieval, programming language and bioinformatics. Syntactic processing has always been paramount to a wide range of applications, such as machine translation, information retrieval, speech recognition and the like. It is therefore natural language syntax has always been one of the most active research areas in the field of language technology [5]. All of the typical pitfalls in language like ambiguity, recursion and long-distance dependencies, are prominent problems in describing syntax in a computational context. Historically, most computational systems for syntactic parsing employ hand-written grammars, consisting of a laboriously crafted set of grammar rules to apply syntactic structure to a sentence. But in recent years, a lot of research efforts are trying to automatically induce workable grammars from annotated corpora, i.e. large collections of pre-parsed sentences [6]. Since the tree-structures in these annotated corpuses already implicitly contain a grammar, it is a relatively trivial task to induce a large-scale grammar and parser that is able to acquire reasonably high parsing accuracies on a held-out set of data. Yet, data-analysis of the output generated by these parsers still brings to light fundamental limitations to these corpus-based methods. Even though they generally provide a much broader coverage as well as higher accuracy than hand-built grammars, corpus-induced grammars will still not hold enough grammatical information to provide an important trend in the field of Machine Learning sees researchers employing combinatory methods to improve the classification accuracies of their algorithms. Natural language problems in particular benefit from combining classifiers to deal with the large datasets and expansive arrays of features that are paramount in describing this difficult and disparate domain that typically features a considerable amount of sub-regularities and exceptions [7]. The field of evolutionary computing has been applying problem-solving techniques that are similar in intent to the Machine Learning recombination methods. Most evolutionary computing approaches hold in common that they try and find a solution to a particular problem, by recombining and mutating individuals in a society of possible solutions. This provides an attractive technique for problems involving large, complicated and non-linearly divisible search spaces.

3. SIMULATION AND CONSTRUCTION OF AN INDIVIDUAL CHROMOSOME

3.1 Chromosomes for Context Free Grammar

In Genetic Algorithm application the choice of the chromosome structure is an important decision. When dealing with grammars, however, the number of parameters required by the model is unknown, and hence (ideally) the chromosomes can be of variable lengths, making the operation of the crossover operator less straightforward than before. Furthermore, the interactions of the individual genes is now profound: the flipping of a single bit (and the corresponding removal or addition of a production) can render a previously perfect grammar utterly useless. Perhaps as a result of these problems, only relatively simple (and deterministic) Context Free Grammars have been inferred (e.g. [2]) using GAS.

The authors have adapted two approaches of generating the effective context free grammar. First approach is to generate random string consisting of 0's and 1's of a specific length and then divide 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. Effective use of empty symbol (Epsilon) results in to the production with variable length. The resulted productions are then treated with traditional methods such as left factoring and left-recursion removal in order to get the resultant production. Lateral approach involves use of a masking operator to get the symbol on the left hand side of every production as a non-terminal.

1. Create a random sequence of 0's and 1' with fixed size.
2. Map the chromosome, from first step, into symbolic form by using appropriate structuring element and Backus-Naur Form to get the rules of the CFG.
3. Apply the masking operator
4. Eliminate left recursion.
5. Perform Left factoring of the Grammar to avoid multiple rules starting with same variable or terminal.

Figure 1. Steps for the individual grammar construction.

3.2 Example

There are several ways of representing language equivalent grammars. They may be represented in a free format, where each production is of the form: $A \rightarrow BC$, where A is a non-terminal and BC is a (possibly empty) string of terminals and non-terminals, or they may be represented in some normal form, where restrictions are placed on the form of BC. Authors used a fixed length chromosome of length 240 which is decoded by a three bit pattern. First bit is used for categorizing the symbols (0- for non-terminal and 1- for terminal) and remaining bits are used for representing non-terminals/terminals. Masking operator used to get left hand side of production used the first bit effectively for the desired output. The example, figure 2, derives a grammar with two terminals and four variables from a chromosome with length 240.

4. FITNESS FUNCTION

The fitness function is based on the weights associated with the acceptance and rejection of the positive and negative string samples. Weights (W_p) are added for every acceptance of positive string and rejection of negative string similarly weight (W_n) is subtracted for every rejection of positive string and rejection of positive string. Fitness function also includes a component for limiting the number of rules in the resultant production (W_r). Fitness function is given by

Step 1: Binary Chromosome (Length = 240)
011101101001010000101111001110010110011111101000000010000010100010111100100000100100101111000001100001111110011010100011010010111001100010111111000111010001101111111100111100111000110010000001110100000001000010001000010000010010111000100111001000010100

Step 2&3 : Mapping is done & Results are

Symbol	Bits used	Symbol	Bits used
S	000	C	011
A	001	(	100
B	010	)	101

Equivalent Symbolic Chromosome (length =80)
C))ABS)?A?B?C?)SS(S)S)?ASAAAC(S?S??A)BA)AC(?AC?(C)S????C?C(CSS?BSS(AS(S()?AA?BS)

Productions Extracted (Total rules = 15)

C->))AB	S->)A	B->C)	S->S(S)
S->)AS	A->AAC(	S->S	A->)BA)
A->C(A	C->(C)	S->?	C->C(C
S->SBS	S-> (AS(	S-> ()A	A->BS)

Step 4: Productions after rule shift & Useless removal

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

Productions after left recursion removal

M->(S)M S->M???? M->?????

Step 5: Total number of valid Production :: 3

S->M M->? M->(S)M

Note : Epsilon is denoted by '?'

Figure 2. Example for the grammar construction.

$f(x) = A-B+C$

$= (W_p*N_p) - (W_n*N_n) + (W_r-N_r).$

Here, N_p, N_n & N_r denotes the number of accepted positive & rejected negative strings, the number of rejected positive string & accepted negative string and the number of rules available in the context free grammar respectively. W_r is balanced with respect to the complexity of the language to be constructed. It is inversely proportional to the complexity of the language.

5. LANGUAGE SET

To asses the performance of the adapted approach, the set of six different languages is used which includes four languages from the language set given by Lankhorst[9] along with two other languages. The definition of the language set is

Language 1 : Balanced Parenthesis Problem.

Language 2 : Even length Palindrome over Zero-One Problem

Language 3: (10)* Language.

Language 4: Language containing No Odd Zero after Odd ones over {0,1}*

Language 5: No Triplets of Zeros over an alphabet {0,1}*

Language 6: 0*1*0*1* Problem over an alphabet {0,1}*

6. METHODOLOGY ADAPTED FOR IMPLEMENTATION

Diversity in the population is the best method to get fast convergence. The effective Crossover and mutation methods are the best way to achieve diversity in the population. The method adapted in the experiment is based o Simple Genetic Algorithm (SGA). The crossover and mutation method adapted for the experimentation are shown in figure 3.

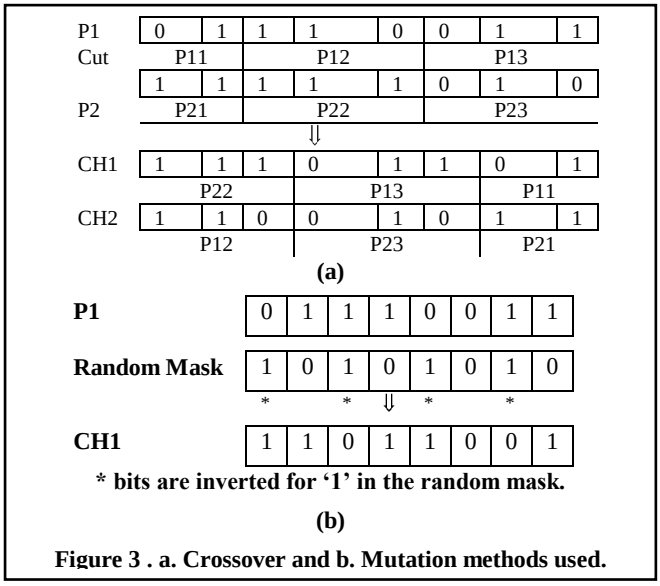
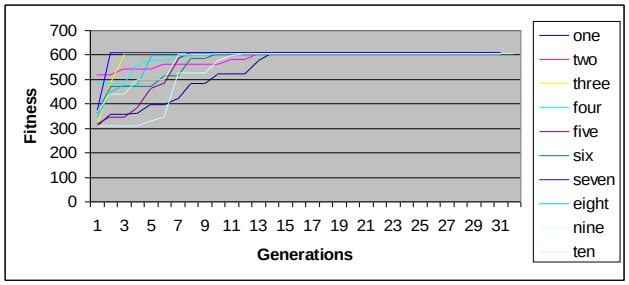


Figure 3 . a. Crossover and b. Mutation methods used.

The crossover method used is a variation of two point crossover method based on the cyclic crossover method. The mutation method uses a random mask of 0's and 1's. for every bit '1' of the random mask the respective bit in the child is inverted.

7. RESULTS & CONCLUSION

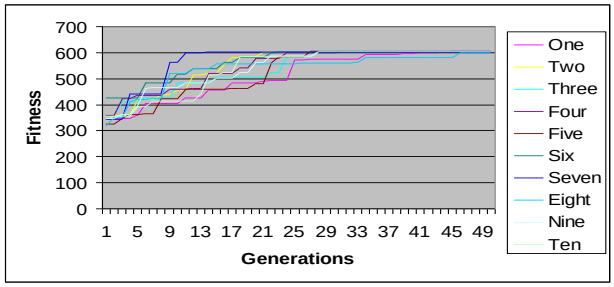
Experiment is done with JDK 1.4 on Intel® Core™2 CPU with 1.66GHZ and 1 GB RAM. The population size, size of the chromosome, Maximum number of generations, crossover rate and mutation rate is taken as 100, 240, 500, 0.9 and 0.8 respectively. Sample set of 30 strings is taken as learning set for grammar induction



L1: Balanced Parenthesis Problem.

Generations required : 17±14

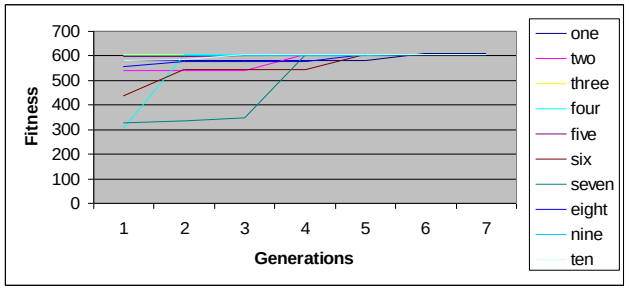
Best Grammar: {S->?,S->(S)L, L->S, L->? }



L2: Even length palindrome over Zero-One Problem

Generations required : 30±15

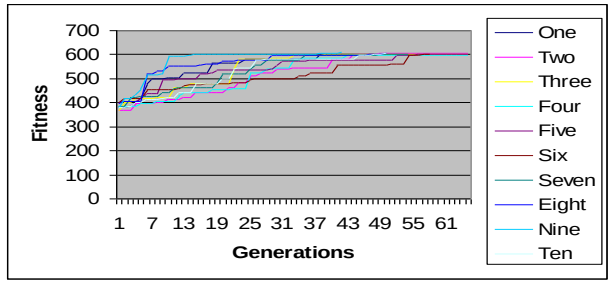
Best Grammar : {S->? , S-> 0S0, S-> 1S1}



L3: (10)*

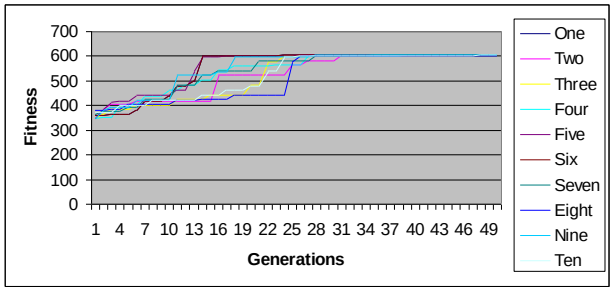
Generations required : 4±3

Best Grammar : {S->? ,S->10S}



L4: No Odd Zero after Odd ones

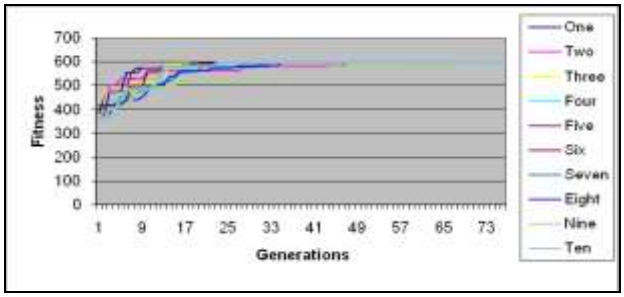
Generations required : 52±10 Best Grammar : {S->M, S->11SM, S->0SM, M->?, M->1M, M->00M}



L5: No Triplets of Zeros

Generations required : 36±12

Best Grammar :{ S->M, S->0J, M->?, M->1SM, J->M, J->0M}



L6: 0*1*0*1* Problem

Generations required : 48±28 , Best Grammar : { S->M, S->C0M, S->0SM, M->CM, M->?, M->1A1M, C->1, C->00C, A->01}

Figure 4 . Results Sets for first ten successful runs of the language sets

Results obtained for first ten successful run on each language set, which does not end with local optima, are summarized in the figure 4. Figure 4 shows the best grammar received over all successful run of the experiment. Methodology adapted found to be working successfully and efficiently on the language set considered. The language set used are the example of the lightweight grammars. There is further scope for adoption of the same method for more complex grammar sets.

8. ACKNOWLEDGMENTS

We sincerely extend our acknowledgements to Dr. V. M. Thakre, P. G. Department of Computer Science, Sant Gadge Baba Amravati University, Amravati, Maharastra, India, for his kind support in providing Laboratory infrastructural facility required for carrying out of research work.

9. REFERENCES

[1] Simon Lucas, IEEE, 2000, A. Structuring Chromosomes for Context-Free Grammar Evolution

[2] Wyard P.,1994, 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,

[3] De la Higuera, 2005, "A Bibliographical Study of Grammatical Inference", Pattern Recognition, v. 38, no. 9, 2005, pp. 1332-1348.

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

[5] Guy De Pauw, 2003, Evolutionary Computing as a Tool for Grammar Development, 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.

[6] Marcus, M.P., Santorini, B., Marcinkiewicz, M., 1994, Building a large annotated corpus of english: the penn treebank. Computational linguistics 19 (1993) 313–330 Reprinted in Susan Armstrong, ed., Using large corpora, Cambridge, MA: MIT Press, 273–290.

[7] Daelemans,W., van den Bosch, A., Zavrel J, 1999, Forgetting exceptions is harmful in language learning. Machine Learning, Special issue on Natural Language Learning 34 (1999) 11–41.

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

[9] Marc Lankhorst, "A Genetic Algorithm for the Induction of Nondeterministic Pushdown Automata", University of Groningen, The Netherlands, May 1995.