Grammar Induction and Genetic Algorithms: An Overview.

Nitin S. Choubey, Ph.D.[1*] and Madan U. Kharat, Ph.D.[2]

[1]MPSTME, NMIMS University, Shirpur Campus, Shirpur, Dhule, Maharastra, India.
[2]Institute of Engineering, Bhujbal Knowledge City, Nashik, Maharastra, India.

[*]E-mail: nschoubey@gmail.com
mukharat@rediffmail.com

ABSTRACT

Grammar Induction (also know as Grammar Inference or Language Learning) is the process of learning of a grammar from training data. This paper discusses the various approaches for learning context-free grammar (CFG) from the corpus of string and presents the approach of informant learning in the form of result for two standard grammar problems namely Balanced Parenthesis Grammar and Palindrome Grammar.

(Keywords: grammar, context free grammar, CFG, genetic algorithm, GA, crossover, mutation, grammar induction)

INTRODUCTION

Genetic Algorithms (GAs) were invented by John Holland in the 1960s. Wyard [3] 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 paper discusses a brief overview of GAs, strategies for CFG induction with Gas, and the details of the implementation done by the authors for CFG induction with GA.

GENETIC ALGORITHM

In the 1950s and the 1960s several computer scientists independently studied evolutionary systems with the idea that evolution process could be used as an optimization tool for engineering problems. The central idea in all of these systems was to evolve a population of candidate solutions to a given problem, using operators inspired by natural genetic variation and natural selection. Rochenberg (1965, 1973) introduced "Evolution Strategies", a method he used to optimize real-valued parameters. This idea was further developed by Schwefel (1975, 1977). The field of Evolution Strategies has remained an active area of research. Fogel, Owens, and Walsh (1966) developed "Evolutionary Programming", a technique in which candidate solutions to given tasks were represented as finite-state machines, which were evolved by randomly mutating their state-transition diagrams and selecting the fittest. Together, Evolution Strategies, Evolutionary Programming, and Genetic Algorithms form the backbone of the field of Evolutionary Computation.

GAs 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 GA 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.

The concept of optimization pervades our every day activities. Optimization can become more complex if it is required to be done to optimize more than one objective at the same time, especially when these objectives are conflicting. Given an optimization problem, it is a challenging task to design an algorithm that can find the optimal solution. It is even more challenging to find an algorithm that does so as fast as possible and to use it to solve problems with as many parameters as possible. Typically, real world optimization problems are characterized by constraints, multiple objectives, and dynamic properties.

When comparing GA's to other conventional optimization methods is that the fitness function can be nearly anything that can be evaluated by a computer or even something that cannot. In the latter case it might be a human judgment that cannot be stated as a crisp program, like in the case of eyewitness, where a human being selects among the alternatives generated by GA. So, there are not any definite mathematical restrictions on the properties of the fitness function. It may be discrete, multimodal etc. Although GA's results in approximate result, they are the suitable mechanism to explore the induction mechanism for Context free grammar with the perspective of optimization because of the way they differ from the conventional optimization methods.

Genetic algorithm differs from conventional optimization techniques in following ways [2]:

1. GA's operate with coded versions of the problem parameters rather than parameters themselves i.e., GA works with the coding of solution set and not with the solution itself.

2. Almost all conventional optimization techniques search from a single point but GA's always operate on a whole population of points (strings) (i.e., GA uses population of solutions rather than a single solution for searching). This plays a major role to the robustness of genetic algorithms. It improves the chance of reaching the global optimum and also helps in avoiding local stationary point.

3. GA uses fitness function for evaluation rather than derivatives. As a result, they can be applied to any kind of continuous or discrete optimization problem. The key point to be performed here is to identify and specify a meaningful decoding function.

4. GAs use probabilistic transition operates while conventional methods for continuous optimization apply deterministic transition operates (i.e., GAs does not use deterministic rules).

GRAMMAR INDUCTION

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

$$V \rightarrow w,$$

where, V is a single non-terminal symbol, and w is a string of terminals and/or non-terminals (possibly empty) [1]. The term "context-free" expresses the fact that non-terminals can be rewritten without regard to the context in which they occur. A formal language is context-free if some context-free grammar generates it. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automata.

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 [3] 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.

The inferred grammar can then be used to classify unseen data or provide some suitable model for this data. Parsing according to a grammar amounts to assigning one or more structures to a given sentence of the language the grammar defines. If there are sentences with more than one structure, as in natural language, the grammar is ambiguous. Parsing can be sought as a search process that looks for correct structures for the input sentence. Besides, if we can establish some kind of preference between the set of correct structures, the process can be regarded as an optimization one. This suggests considering evolutionary programming techniques, which are acknowledged to be practical search and optimization methods [4].

Grammar induction has several practical applications outside the field of theoretical linguistics, such as structural pattern recognition [5][9] (in both visual images and more general patterns), automatic computer program synthesis and programming by example, information retrieval, programming language and bio-informatics. 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 [6]. All of the typical pitfalls in language like ambiguity, recursion and long-distance dependencies, are prominent problems in describing syntax in a computational context. 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.

INFORMANT LEARNING APPROACH FOR CFG INDUCTION

The informant learning approach is used in the paper (based on the corpus constructed from the set of positive and negative string for the grammar to be constructed). The Crossover and Mutation operators used are shown in the Figure 1.

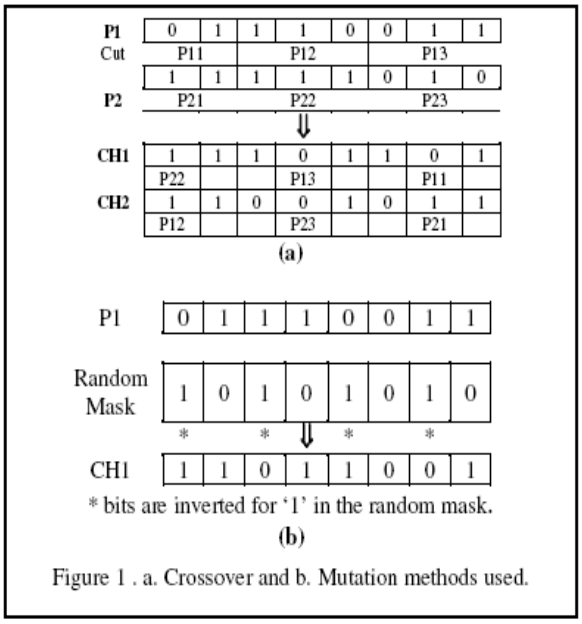


Figure 1 . a. Crossover and b. Mutation methods used.

Crossover method used divides the selected parents in to three parts similar to the two point crossover method and then reorder them in cyclic form to get two children.

Parents:

P1: P11 → P12 → P13
P2: P21 → P22 → P23

Resultant Children:

Ch1: P22 → P13 → P11
Ch2: P12 → P23 → P21

The corpus of Positive and Negative Strings work as the learning parameter for grammar construction. Procedure adapted Grammar

induction using Genetic Algorithm for implementation is shown in the Figure 2 and Figure 3.

1. Create the Random population of Grammar by using above Grammar Construction steps.
2. Calculate the fitness of the grammars in the entire population by using fitness function and get the best individual by using following formula
3. Fitness of the grammar = $((n(AP)+n(RN))-((AN)+(RP))*PW+PW*CL)-(PW-MR)$
4. Perform steps 4 to till 7 **either** the best individual with fitness greater than the required threshold is found **or** the given number of generation are completed.
5. Create new population by performing crossover and mutation operation (use crossover Rate and Mutation Rate & by using appropriate method for the same).
6. Calculate the fitness of the entire newly constructed individuals by using fitness function.
7. Merge the both population by using appropriate replacement method.
8. Update the best individual in the population.
9. Stop.

Figure 2 . Genetic Algorithm for CFG Induction.

Fitness function is based on the nature of the string supplied in the corpus. Corpus length (CL) is taken as 30 which include 15 positive strings and 15 negative strings. For the acceptance of every positive string appropriate amount of weight is added whereas for the acceptance of every negative string, appropriate weight is subtracted.

The fitness of an individual grammar is calculated based on the acceptance and rejection of the strings in the corpus. Positive weight (PW) is added for every acceptance of positive string (AP) and rejection of every negative string (RN) whereas penalty of the PW is given for every rejection of Positive string (RP) and acceptance of every Negative string (AN). An additional factor of the maximum number of expected rules (MR) is used to control the number of rules required in the resultant grammar.

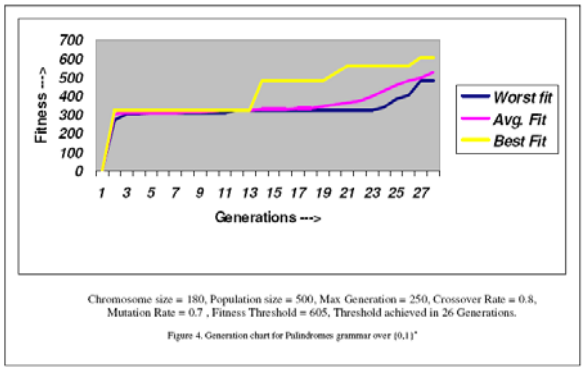
1. Create a random sequence of 0's and 1' up to the size of chromosome.
2. Map the chromosome into symbolic form by using appropriate structuring element and Backus-Naur form to get the rules of the CFG.
3. Eliminate left recursion.
4. Perform Left factoring of the Grammar to avoid multiple production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

Figure 3 . Steps for grammar construction from an individual chromosome

RESULTS

Experiment is done with JDK 1.4 on a Intel® Core™2 CPU with 1.66GHZ and 1 GB RAM. Two different corpus sets are used.

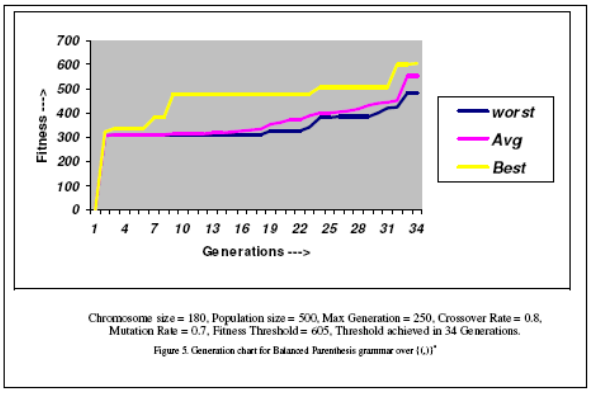
1 . Set of Palindromes over {0,1}* .
The resultant Grammar is <{S},{0,1},P,S> where P is {S->?, S->1S1, S-> 0S0, }. Fitness of the resultant grammar is 607 Generation chart is shown in Figure 4.



Chromosome size = 180, Population size = 500, Max Generation = 250, Crossover Rate = 0.8, Mutation Rate = 0.7 , Fitness Threshold = 605, Threshold achieved in 26 Generations.

Figure 4. Generation chart for Palindromes grammar over {0,1}*

2 . Set of Balanced Parenthesis over {(,)}*.
The resultant Grammar is <{S,L},{(,)},P,S> where P is {S->?, S->(S)L, L->S, L->? } Fitness of the resultant grammar is 606 Generation chart is shown in Figure 5.



Chromosome size = 180, Population size = 500, Max Generation = 250, Crossover Rate = 0.8, Mutation Rate = 0.7, Fitness Threshold = 605, Threshold achieved in 34 Generations.

Figure 5. Generation chart for Balanced Parenthesis grammar over {(,)}*

REFERENCES

1. Hopcroft, J.E., R. Motwani, and J.D. Ullman. 2007. *Introduction to Automata Theory, Languages, and Computation, 3/E*. Addison-Wesley: New York, NY.

2. Sivanandam, D. 2008. *Introduction to Genetic Algorithm*. Springer: Berlin, Germany.

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

4. Javed, F., B.R. Bryant, M. Crepinek, Mernik, and Sprague. 2004. "Context Free Grammar Induction Using Genetic Programming". ACMSE: Huntsville, AL.

5. de la Higuera, C. 2005. "A Bibliographical Study of Grammatical Inference". *Pattern Recognition*. 38(9):1332-1348.

6. De Pauw, G. 2003. "Evolutionary Computing as a Tool for Grammar Development". CNTS – Language Technology Group, UIA – University of Antwerp: Antwerp, Belgium.

7. Cant´u-Paz, E. et al. (eds.). 2003. GECCO 2003, LNCS 2723. 549–560. Springer-Verlag: Berlin, Germany.

8. Marcus, M.P., Santorini, B., and Marcinkiewicz, M. 1993. "Building a Large Annotated Corpus of English: The Penn Treebank". *Computational Linguistics.* 19:313–330. Reprinted in S. Armstrong, ed. 1994, *Using Large Corpora*. MIT Press: Cambridge, MA. 273–290.

9. Daelemans, W., van den Bosch, A., and Zavrel, J. 1999. "Forgetting Exceptions is Harmful in Language Learning". *Machine Learning, Special issue on Natural Language Learning*. 34: 11–41.

10. Rodrigues, E. and H.S. Lopes. 2000. "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.

ABOUT THE AUTHORS

Dr. N. S. Choubey , BE , ME , MBA , Ph.D. was educated at Amravati University, India and also holds a Diploma in TQM & ISO 9000. Presently he is working at Mukesh Patel School of Technology Management and Engineering at SVKM's NMIMS University, Shirpur Campus, Shirpur, Dhule, India, as a Associate Professor and Head of the Computer Engineering Department. He has presented papers at National and International conferences and also published papers in National Journals on various issues of such as Computer Engineering and Management. To his credit, he has published books on various topics in Computer Science and Management subjects. He is also a Certified Internal Quality Auditor. His areas of interest include Algorithms, Theoretical Computer Science, and Computer Networks and Internet.

Dr. M. U. Kharat, BE, MS, Ph.D. was educated at Amravati University. Presently he is working at the Institute of Engineering, Bhujbal Knowledge City, Nashik, Maharastra, India, as Professor & Head of the Computer Engineering Department. He has presented papers at National and International conferences and also published papers in National and International Journals on various aspects of Computer Engineering and Networks. He has worked in various capacities in academic institutions at the level of Professor, Head of Computer Engineering Department, and Principal. His areas of interest include Digital Signal Processing, Computer Networks, and the Internet.

SUGGESTED CITATION

Choubey, N.S. and M.U. Kharat. 2009. "Grammar Induction and Genetic Algorithms: An Overview". *Pacific Journal of Science and Technology.* 10(2):884-888.

Pacific Journal of Science and Technology