# Evolving Stochastic Context-Free Grammars Using Genetic Algorithm

Anupam Shukla, National Institute of Technology, Raipur, India; E-mail: anupamshukla2005@yahoo.com
Devesh Narayan, National Institute of Technology, Raipur, India; E-mail: devesh_nar@yahoo.com

## ABSTRACT

*The learning of stochastic context-free grammars from corpus using genetic algorithm is explored in this work. Minimum description length principle is used for deriving the fitness function of the genetic algorithm. Stochastic context-free grammars are evolved by optimizing the parameters of the covering grammars. I provide details of my fitness function for grammars and present the results of a number of experiments in learning grammars for a variety of languages.*

## INTRODUCTION

Stochastic context-free grammars (SCFGs) are perhaps best known as a tool for expressing the syntactic structure of natural languages. Practical techniques for grammar induction have many important applications or a wide range of natural language (NL). In recent years SCFGs have been widely applied to problems in computational biology, such as modeling the secondary structure of RNA families. Other applications include visual recognition of activities and language modeling for speech recognition, robotics. A problem of central importance in each of these applications is inducing SCFGs from data.

Inferring a stochastic grammar from data, as revealed by most previous practice, involves in general two essential sub-tasks, one to infer a set of phrase structure rules (or productions), another to estimate a correspondent set of probabilistic parameters (i.e., production probabilities) The inference can be viewed as a search process for the best grammar allowable in a predefined grammar (or hypothesis) space. There are many sophisticated algorithms exist to facilitate the searching, e.g., genetic algorithm and stimulated annealing algorithm. However, no matter how sophisticated is the search method in use, the goodness criterion to guide the searching remains a critical issue. It is this criterion that tells a search algorithm which grammar is better. In this project we developed a suitable criterion for the estimation of the parameters of the stochastic context-free grammars. It is based on the classic and algorithmic information theory and on the Minimum Description Length (MDL) principle and genetic algorithm.

## STOCHASTIC CONTEXT-FREE GRAMMAR (SCFG)

A stochastic context-free grammar (SCFG) is a variant of ordinary context-free grammar in which each grammar rule is associated with a probability, a real number in the range [0,1]. The set of production probabilities will be referred to as the parameters of the SCFG. For a SCFG to be proper, the probabilities associated with all rules that expand the same non-terminal symbol must be one.

The language L(G) generated by a SCFG G comprises the set of all strings of terminal symbols derivable from the start symbol of the grammar. In addition, the parameters define a probability distribution over strings in L(G). For a string $\alpha \in$ L(G), the probability of a parse tree for $\alpha$ is given by the product of the probabilities of all the grammar rules involved in its construction. The probability $P_G(\alpha)$ of the string $\alpha$ is the sum of the probabilities of all of its parses.

$S \rightarrow AB$ (1.0)
$A \rightarrow a$ (0.6)
$A \rightarrow CS$ (0.4)
$B \rightarrow b$ (1.0)
$C \rightarrow a$ ( 1.0)

The above SCFG, with the probability associated with each production is given in parentheses, generates the language $\{ a^n b^n | n \geq 1\}$, where $P_G(ab)=0.6$, $P_G(aabb)=0.24$ and so on.

## BIASED WEIGHT GRAMMARS

Biased Weight Grammars are similar to SCFGs in that they associate numerical parameters with the rules of the grammars, a bias and a weight. Any BWG G can be converted to an equivalent SCFG G'. Let Grj denote the set of rules in G that expand the same nonterminal symbol as rule rj. Then each rule rj in G with bias bj and weight wj has a corresponding rule rj' in G' with associated probability pj given by

$$P_j = \frac{b_j w_j}{\sum_{r_k \in Gr_j} b_k w_k} \quad (1)$$

The language generated by a BWG G to be the same as L(G'), the language generated by its equivalent SCFG G', with the same associated probability distribution over its sentences.

## CORPUS-BASED GRAMMATICAL INFERENCE

A corpus C for a language L is a finite set of strings drawn from L, where each string $\alpha \in C$ is associated with an integer $f_\alpha$ representing its frequency of occurrence. The size $N_c$ of the corpus is defined as the sum of the frequencies of the individual strings in C. That is

$$N_c = \sum_{\alpha \in C} f_\alpha \quad (2)$$

The relative frequency $p_\alpha$ of a string $\alpha \in C$ is defined as $p_\alpha = f_\alpha / N_c$ Given a corpus C as training data, the inference problem is to identify a SCFG that (a) models the corpus as accurately as possible and (b) generalizes appropriately to the wider language from which the corpus was drawn. This problem is tackled by trying to identify a BWG with these properties taking its associated SCFG as the one learnt by our system.

For any probabilistic language model, natural measure of accuracy is the probability of the corpus data given the model. In this case, the most accurate model in this sense is that grammar G" given by

$$G" = \text{argmax}_G P(C|G) \quad (3)$$

Where P(C|G) ( the conditional probability of the language data C given by the grammar G) is defined as

$$P(C|G) = \frac{N_C!}{\prod_{\alpha \in C} f_\alpha!} \prod_{\alpha \in C} P_G(\alpha)^{f_\alpha} \quad (4)$$

On the other hand, simply maximizing the probability of the corpus data will not generally meet the further requirement of generalization. A perfectly accurate model is one which generates exactly the finite corpus and assigns to each string the correct relative frequency. In otherwords, the most accurate grammar will over-fit the training data. What actually require is the grammar that is most probable given the training data. That is, a grammar G" such that

$$G" = \text{argmax } G\ P(G|C) \quad (5)$$

Unfortunately, it is not clear how to calculate P(G|C) directly. From Bayes rule

$$P(G|C) = \frac{P(G)P(C|G)}{P(C)} \quad (6)$$

Ignoring P(C), which is a constant, maximizing P(G|C) just corresponds to maximizing the product of P(C|G) ( which can be calculated directly) and P(G), the prior probability of the grammar G. This poses the problem of fixing an appropriate prior probability distribution over grammars. In principle there are many different priors that could be chosen, but it seems reasonable to assume that we should prefer smaller and simpler grammars to larger, more complex ones. Our choice of prior is therefore related to the minimum description length principle of Risannan as well as earlier work on inductive due to Solomanoff.

## THE MINIMUM DESCRIPTION LENGTH PRINCIPLE

Given some data D, we should pick that theory T which minimizes:

$$L(T)+L(D/T)$$

Where L(T) is the number of bits needed to minimally encode the theory T, and L(D/T) is the number of bits needed to minimally encode the data D given the theory T.

From Shannon's information theory, we know that if we have a discrete set X of items with a probability distribution P(x) defined over it, then in order to send a message identifying x ε X we need approximately $L(x)= -\log_2 (P(x))$bits. In other words,

$$P(x)= 2^{-L(x)} \quad (7)$$

This enables us to interpret the M D L principle in Bayesian terms. From the equation it can easily be seen that minimizing L(T)+L(D/T) corresponds to maximizing P(T)P(D/T) and hence P(T/D).

It should be noted that for us the most useful feature of the M D L Principle is that it can be used "inverse". Information theory (Shannon 1948) tells us how work out minimum code lengths given prior knowledge of the probability distributes over items in some set of interest. The M D L Principle enables us to assign prior probabilities to items in some set in a meaningful way, even if we do not really have enough prior knowledge. We can do this by attempting to find minimal length encoding for the items and then use equation (4) to work out the probabilities.

## THE GENETIC ALGORITHM FOR SCFGS

Given a corpus C as training data, our approach to grammatical inference involves the following steps

1. Construct a covering grammar that generates the corpus as a (proper) subset.
2. Set up a population of individuals encoding parameter settings for the rules of the covering grammar.
3. Repeatedly apply genetic operations ( crossover, mutation) to the selected individuals in the population until an optimal set of parameters in found.

The covering grammar is in the Chomsky Normal Form (CNF) and contains every rule of the form A → BC and every rule of the form A→ a.

A member of the population encodes a set of weights for the rules of the covering grammar. Each weight is encoded as a binary integer, using w a fixed length bit string. The bias associated with each rule is determined in advance according to a prior probability distribution p" over grammar rules. Thus, for a rule r in the covering grammar the associated bias is given by p"(r). The prior distribution is chosen to reflect a preference for shorter, simpler rules. This makes it easier for the genetic algorithm to learn grammars of the sort we prefer(simpler and shorter) because changes to the weights of more heavily biased rules have a greater effect on the resulting probabilities of the sentences in the corpus. Consequently the algorithm is far more sensitive to simpler, shorter rules.

The members of the initial population are generated randomly after which the genetic algorithm repeatedly executes the following select-breed-replace cycle.

**Select** a random member of the population for breeding using roulette selection method.
**Breed** by applying crossover and mutation to produce two children.
**Replace** the weakest parent by the fittest child.

In crossover operation, variable size cromosomes are used. In making a chromosome for a set rules, only those rules are included for which the weight is not zero.

## THE FITNESS FUNCTION

In practice, it is not convenient to compute the conditional probability P(G|C) directly as a means of evaluating the fitness of grammars. Instead, the genetic algorithm uses an objective function F given by

$$F(G) = \frac{K_C}{L(C|G) + L(G)} \quad (8)$$

Maximizing F(G) corresponds to minimizing the denominator of equation. This in turn just amounts to maximizing P(G|C). The numerator Kc is a problem(corpus) dependent normalization factor that yields fitness values in the range [0,1].

Since $L(C|G)= -\log_2 P(C|G)$ it is given by

$$L(C|G) = -\log_2 \left( \frac{N_C!}{\prod_{\alpha \varepsilon C} f_\alpha!} \right) - \sum_{\alpha \varepsilon C} f_\alpha \log_2 (P_G(\alpha)) \quad (9)$$

The first term is a constant (depending only on the corpus) and therefore can be ignored in minimizing the denominator of equation 5. The second term is Nc times the cross entropy of the corpus C given the model provided by the grammar G. It can be interpreted as the number of bits needed to communicate the corpus (with the frequencies given) using a code guaranteed to minimize message length if the sentences had occurred in the corpus with probabilities as given by the grammar.

In order to compute L(G), the grammar is represented as a code. Different choices of coding scheme give rise to different probability distribution over the set of grammars. In genetic algorithm, a coding scheme is used in which a grammar can be represented in any convenient fashion as a genome, compute its length according to our chosen coding scheme and hence assign it a prior probability.

For BWGs with a fixed determined set of biases, a grammar is completely specified by some set of pairs of the form (r,w), where r is a rule and w its weight. Pairs are omitted if their weight is zero. The length for the whole grammar is given by the

sum of the lengths of the codes for each of its rules, where the length of the code of a rule is given by the length of a code for r plus length of a code for w.

To encode a weight w, one of a family of prefix codes for integers which all form good approximations to the minimal encoding can be used. These codes represent an integer by a code for the integer itself, preceded by a code for its length. The code for the integer w requires

approximately $\log_2(w)$ bits while the code for its length needs roughly $\log_2(\log_2(w))$ bits(using normal binary encoding). However this does not give any way of deciding where the code for the length ends and the code for the integer w itself begins. Therefore a code is used which duplicates every binary digit of the integer representing the length followed by a single 0. This means that an integer w will need approximately $\log_2(w)+2\log_2(\log_2(w))+1$ bits.

To compute the code lengths for r, a probability distribution P"(r) over the rules is defined. A natural way of computing the probabilities of each $r_j$ is to look at the probabilities of the symbols in each position of the set of rules in the covering grammar. In particular, we look at the set of symbols allowed at a given position in the rules given the preceding symbols. If all legal symbols at that position can occur with equal probability, then P"(r) can be computed as follows

Let N be the set of non-terminal symbols in the covering grammar G and let n be the length of right hand side of the longest rule. Now write each rule $r_j$ in the form $S_{j_0} \rightarrow S_{j_1} S_{j_2} \ldots S_{j_n}$ where $S_{j_m}$ is defined to be a special "blank" symbol for all m greater than the length of the right hand side of rule $r_j$. Define $S_{j_k}$ $(0 \le k \le n)$ to be that set of symbols given by

$$S_{j_k} = \begin{cases} N & \text{if } k=0 \text{ and} \\ \{ Sjk \mid \ni r_i \in G \text{ such that } S_{i_l} = S_{j_l} \ (0 \le l \le k) \} & \text{otherwise} \end{cases} \quad (10)$$

The probability P"($r_j$) of rule $r_j$ is than given by

$$P"(r_j) = \prod_{0 \le k \le n} P_{j_k} \quad (11)$$

Where $Pjk(0 \le k \le n)$ is defined as $1/|S_{j_k}|$

Given the probability distribution P over rules, the number of bits needed to represent a particular weighted grammar is given by

$$\sum_{(r,w) \in G} ( -\log_2(P"(r) + \log_2(w)+2\log_2(\log_2(w))+1 \quad (12)$$

This quantity is of course $L(G) = -\log_2(P(G))$

## ALGORITHM FOR EVOLVING STOCHASTIC CONTEXT-FREE GRAMMAR

To identify a Stochastic Context Free Grammar for a given corpus as training data that (a) models the corpus as accurately as possible and (b) generalizes appropriately to the wider language from which the corpus was drawn, the steps involved are as follows

**Step 1**: Identify a covering grammar for a given corpus as training data using Sequitur algorithm. The grammar must be as small as possible and simple.

**Step 2**: Generate all parses for all strings in the corpus derivable from the covering grammar using All Parse algorithm.

**Step 3**: Convert a stochastic context free grammar in to a biased weight grammar. Associate numerical parameters with the rules of the grammar, a bias and a weight

**Step 4**: Code grammar using prefix coding scheme for integers. This scheme is used to represent a grammar a convenient fashion as a genome. Compute its length according to our chosen coding scheme. The length of the whole grammar is given by the sum of the lengths of the codes for each of each of its rules, where the length of the code of a rule is given by the length of a code for r plus length of a code for w.

**Step 5**: Determine an objective function for the genetic algorithm to evaluate the fitness of grammars.

**Step 6**: Apply genetic algorithm for setting the parameters of the stochastic context-free grammar.Genetic algorithm is repeated until a threshold value is achieved for parameters of SCFG or a maximum iteration value is reached. Threshold value is dependent of the type of the language generated by the grammar G.

## EXPERIMENTAL RESULTS

We have conducted a number of experiments in learning grammars for a range of formal languages. These languages are representative of those considered in other studies.

1. (a+b)*bb: the language is the set of strings ending with a sequence of at least two **b**s.
2. **EQ**: the language of all strings consisting of equal number of **a**s and **b**s, (ab+ba)(ab+ba)*.
3. The language $a^n b^n$ $(n \ge 1)$.
4. **BRACKET**: The language of balanced brackets.
5. **PAL**: palindromes over {a,b}.

For each experiment, a corpus was first produced automatically using a hand-crafted SCFG for the target language. This involved randomly generating on the order of 1000 strings up to a pre-specified 'maximum sentence length' (typically 6 or 8). For each problem, the population size was fixed to 25.

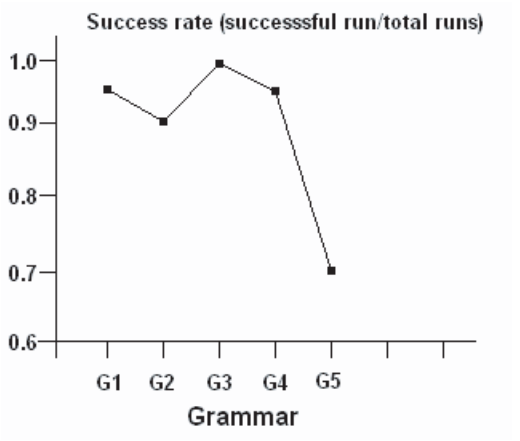
In order to assess the performance of the genetic algorithm, 20 runs were completed on each language learning task. A run of the genetic algorithm was terminated as 'successful' if a SCFG was found with fitness above a threshold value of 0.93. While this figure is somewhat arbitrary, experience has shown that grammars attaining this fitness are usually correct in the sense that they generate the target language exactly, and assign appropriate probabilities to the strings. Runs of the genetic algorithm that failed to attain the threshold value were terminated after a maximum number of select-breed-replace cycles. The number of cycles was set individually for each problem and was high enough to ensure convergence in the population.

The results of the experiments are summarized in the table given above. For each learning task, the table gives the number of non-terminals used in the covering grammar, the number of parameters to be optimized, the success rate (number of runs that attained the threshold fitness value) as well as the maximum fitness value found on the best and the worst runs of the genetic algorithm. As can be seen,

*Table 1. Experimental result for different type of grammars*

| Language | Non-terminals | Parameters | Success rate | Best fitness value | Worst fitness value |
|---|---|---|---|---|---|
| **G1**: (a+b)*bb | 3 | 6 | 19/20 | 0.972 | 0.946 |
| **G2**: EQ | 3 | 5 | 18/20 | 0.970 | 0.681 |
| **G3**: BRACKET | 3 | 5 | 20/20 | 0.955 | 0.949 |
| **G4**: $a^n b^n$ | 4 | 5 | 19/20 | 0.978 | 0.866 |
| **G5**: PAL | 5 | 10 | 14/20 | 0.951 | 0.869 |

*Figure 1. Graph between success rate and grammar given in table 1*




the first four tasks ((a+b)*bb, EQ, $a^n b^n$, BRACKET) presented little difficulty. Inspection of the grammars produced on successful runs for these experiments showed that they were indeed correct. For the occasional unsuccessful runs the relatively poor fitness values attained suggest the presence of local maxima around which the population has converged. The palindrome example PAL provided the algorithm with a rather harder test. Here, the success rate fell to around 65% to 70%. On runs where the algorithm failed to find the correct grammar, inspection of the best grammars showed that the population had converged on solutions, which had the correct rules for palindromes starting with any of the available symbols, except for one. Because these grammars covered the palindromes while assigning generally low probability to a range of other sentences they still managed to attain a relatively high fitness.

Figure 1 shows that success rate is generally greater than 90% for the grammars. It depends on the complexity of the grammar.

## CONCLUSIONS

The approach to grammatical inference described in this project differs from previous works using genetic algorithms in addressing the problem of corpus-based inference of stochastic context-free grammar. In our approach number of rules in covering grammar is reduced by using sequitur algorithm and convergence rate of genetic algorithm is speedup by using variable crossover over variable length chromosome algorithm. This makes direct comparison our results with those of other algorithms difficult. However, the experiments that we have conducted are typical of those in other studies and the results reported in this project appear promising. The approach also appears to compare well with other (non-genetic) techniques for stochastic grammatical inference.The main limitation of our approach is the cost involved in evaluating the fitness of each candidate solution, which requires parsing every string in the corpus in all possible ways. Success rate of our algorithm depends on the complexity of the grammar. Although inference can be performed very quickly for small covering grammars, the number of parses that must be considered increases exponentially with the number of rules in the grammars. This problem can be solved by including the possibility of a massively parallel implementation of this algorithm.

## REFERENCES

[1] K.L.P. Mishra, Theory of Computer Science (Automata, Languages and Computation), PHI, 2000.
[2] Dan W. Patterson, Introduction to Artificial Intelligence and Expert Systems, PHI, 2001.
[3] Elaine Rich and Kevin Knight, Artificial Intelligence, Tata McGraw-Hill, 2000.
[4] Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman, Compilers Principles, Techniques and Tools, Low Price Edition, 2000.
[5] Herbert Schildt, The Complete Reference C++, Tata McGraw-Hill, 1999.
[6] Sahni, Data Structures, Algorithms, and Applications in C++, McGraw-Hill, 2000.
[7] Marjan Mernik, Matej Crepin¡sek, Goran Gerli¡c, Viljem Zumer, " Learning Context-Free Grammars using an Evolutionary Approach", University of Maribor, 2000.
[8] Lukasz Debowski, "On the Best Theories for Learningful Texts", Institute of Computer Science-Polish Academy of Sciences, 2002.
[9] F. Amaya, J.M. Benedi and J.A. Sanchez, "Learning Of Stochastic Context-Free Grammars From Bracketed Corpora By Means Of Reestimation Algorithms*", Department of system information and computation.
[10] Moses Charikar, Eric Lehmany, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, abhi shelat, "Approximating the Smallest Grammar: Kolmogorov Complexity in Natural Models", 2002.
[11] Steve Lawrence, C. Lee Giles_, Sandiway Fong, "Natural Language Grammatical Inference with Recurrent Neural Networks", NEC Research Institute.
[12] "Stochastic context-free grammar", http://www.cse.ucsc.edu/research/compbio/scfg.html
[13] "Automata and Formal grammars", http://www.csee.umbc.edu/~sdoshi1/Inferring%20Graph%20Grammars.pdf
[14] "Minimum Description Length Principle", http://citeseer.nj.nec.com/cache/paperscs/24767/http:zSzzSzwww3.oup.co.ukzSzcomputer_journalzSzhdbzSzVolume_42zSzIssue_04zSzpdfzSz420260.pdf/rissanen98hypothesis.pdf
[15] http://citeseer.nj.nec.com/cache/papers/cs/1016/ftp:zSzzSzftp.cogs.sussex.ac.ukzSzpubzSzuserszSzbillkzSzicml97.pdf/evolving-stochastic-context-free.pdf
[16] "Language definitions", http://www.csee.umbc.edu/help/theory/lang_def.shtml
[18] "Introduction to genetic algorithms", http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/