Structuring Chromosomes for Context-Free Grammar Evolution

Simon Lucas
Department of Electronic Systems Engineering
University of Essex
Colchester CO4 3SQ

Abstract

This paper investigates the use of genetic algorithms for inferring small regular and context-free grammars. Applied simply, a genetic algorithm is not very effective at this. To overcome this problem we investigate two methods of structuring the chromosomes. The first is to bias the distribution of '1's in the population of chromosomes according to an algebraic expansion technique previously developed by the author. This 'design' of the chromosome distribution, shows no bias to any particular type of language (i.e. full generality is retained) yet improves convergence. The second method involves performing the evolution (i.e. making the mutations) in a different space, where the grammars are represented in 'embedded normal form'. The latter approach structures the chromosome to represent context-free rather than regular grammars, for example. It is shown that biasing the chromosome in this fashion produces extremely fast convergence, and 3-symbol palindromes grammars are learned in typically less than 10 generations.

1 Introduction

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.

As with many domains, there is a tradeoff between the complexity of a model and the cost of estimating it. An example of this may be seen in speech recognition, where hidden Markov models (equivalent to stochastic regular grammars) are applied with great success. These are actually not the most fitting model for speech recognition (for example, context-free would be a theoretically more appropriate choice), but due to the existence of fast robust training algorithms, they have achieved great popularity.

Nonetheless, there is still a great deal of interest in learning more powerful classes of grammar, such as context-free.

The main contribution of this paper is the idea of biasing the probability distribution of the population of chromosomes, using an according to two distinct techniques.

The first is to bias the distribution of '1's in the population of chromosomes according to an algebraic expansion technique previously developed by the author [1, 2]. This 'design' of the chromosome distribution, shows no bias to any particular type of language (i.e. full generality is retained) yet improves convergence. The second method involves performing the evolution (i.e. making the mutations) in a different space, where the grammars are represented in 'embedded normal form' (ENF). The latter approach structures the chromosome to represent context-free rather than regular grammars, for example. This 'design' of the chromosome distribution, while showing no bias to any particular type of language (i.e. full generality is retained) seems to be much more effective than experimenting with parametric details such as the mutation and crossover rates.

The next Section considers the problem of designing chromosomes in general, while Section 3 considers the specific problem of mapping from encoding a grammar for evolutionary purposes, and considers how to bias the probability distribution of bit strings in favour of those corresponding to reasonable grammars, or fragments of reasonable grammars. Section 4 presents some initial results, and Section 5 concludes.

2 Chromosome Design

In many applications of genetic algorithms (GAs), the choice of chromosome tends to be taken for granted. Hence, the parameters of the model to be estimated are each encoded as a string of bits (for example,

using a binary or gray-code representation), and the chromosome is then formed from the concatenation of these strings.

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) CFGs have been inferred (e.g. [3]) using GAs.

Kitano [4] was perhaps the first to publish results on the effects of chromosome design of relevance to us here. His results compared the effects of using a direct encoding of a neural architecture (as a binary matrix indicating the presence or absence of a neuron), with using a parallel rewriting grammar as the chromosome. In the latter case, the grammar was then run for a fixed number of cycles to produce a matrix of the desired size. Kitano found that the grammar outperformed the direct encoding by an order of magnitude (where performance measured by the average number of generations taken by the GA to converge on a solution within the specified error bounds).

Since then Gruau [5] has done some interesting work on designing more advanced grammars for neural network evolution, and has made significant progress since the work of Kitano. In particular, Gruau identifies a set of characteristics that a good chromosome should have. These include such properties as modularity, compactness and completeness.

Also of interest is the work of Boers and Kuiper [6], whose approach is to model the artificial chromosomes on their biological counterparts, rather than paying special attention to the formal properties of the codes.

Rather than studying how grammars may be designed to evolve neural architectures, this paper considers the design of chromosomes for evolving good grammars. While eventually we should like to employ some kind of structured approach (such as the use of a grammar) to create the chromosomes, we have not done this yet. Note however, that the effect of imposing a higher level structure on the chromosomes (as would be the case if we used a grammar to generate them) is to deform the probability distribution of the chromosomes; this is precisely the effect we achieve here, but by using a different technique.

3 Chromosomes for grammar representation

There are several ways of representing language-equivalent grammars. They may be represented in a free format, where each production is of the form: $A \rightarrow \alpha$ where A is a non-terminal and α 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 α.

Given any form of a grammar, it is possible to conjure up mutation and crossover operators, and these will be different depending on the chosen form. It would be interesting to experiment on the effect that the form of the grammar has on the operation of the GA, but this is left to future work for the moment.

Here our approach is to take the simplest normal form for constructing binary codes. This is a form I call Lucas Normal Form (LNF), which is closely related to Chomsky Normal Form (CNF). Grammars in LNF tend to look uglier than their CNF counterparts, but the advantage here is that their more rigid structure reduces the size of the search space (given the same number of non-terminals) and simplifies parsing.

3.1 Chomsky Normal Form (CNF)

In CNF each rule is either of the form $A \rightarrow BC$ or of the form $A \rightarrow a$ where A, B, C are non-terminals, and a is a terminal.

3.2 'Embedded Normal Form' (ENF)

ENF is not really a normal form, but a template for a self-embedded production (hence the quotes). However, examination of many toy CFGs (when represented in their most natural form reveals that many of the productions are of the the form $A \rightarrow bAc$ or of the form $A \rightarrow b$ where A is a non-terminal, and b and c are terminals.

3.3 Lucas Normal Form and Direct Encoding

In LNF, each production right-hand side (RHS) now has exactly two symbols, but these may be either terminals or non-terminals. Hence, each production is now of the form $A \rightarrow \alpha_1\alpha_2$, where $A \in N$ and $\alpha_1, \alpha_2 \in (N \cup \Sigma)$.

In other words, there are exactly 4 types of production: $A \rightarrow aa|aB|Ba|BB$

The fact that each RHS now has exactly two symbols means we can perform all parsing reductions (i.e.

replacing the appearance of an RHS with an LHS) using a binary two-dimensional parsing look-up-table (PLUT) for each LHS.

3.4 Example

For experimentation purposes I have implemented an algorithm which takes any regular or context-free grammar, and converts it to LNF. Here we illustrate the effects of doing this with a two-symbol palindrome grammar (PG). It should be pointed out that LNF grammars have the restriction that all strings they generate are at least two symbols long. This is not a problem in practice.

Consider the following grammar, which is in 'ENF', and generates the complete (infinite) set of 2-symbol palindromes.

$$S \rightarrow 0S0|1S1|0|1|\epsilon$$

For illustration, we show the actual inputs and outputs of the algorithm in LISP:

```
> (lnf '((S -> 0 S 0) (S -> 1 S 1)
         (S ->) (S -> 0) (S -> 1)))

((B -> S 0) (B -> 1 0) (B -> 0 0)
 (A -> S 1) (A -> 1 1) (A -> 0 1)
 (S -> 0 B) (S -> 0 0) (S -> 1 A) (S -> 1 1))
```

$$\begin{aligned} S &\rightarrow 0B|1A|00|11 \\ A &\rightarrow S1|11|01 \\ B &\rightarrow S0|10|00 \end{aligned}$$

Table 1 shows how to package this into a binary chromosome in this case the following string:

```
000000000000000001100100100010000000
000000010000100001000000000000100001
```

3.5 Biased Chromosomes

Some initial experiments suggested that GAs would not perform well at CFG inference. This appeared to be due to the fact that in many cases to improve a grammar (and hence escape a local minima), many chance mutations were needed simultaneously. Crossover would take care of this in many applications, but recall that a grammar is a complex structure, and that particular genes are only good in relation to other genes i.e. there is often an absence of simple building blocks, and hence a rather shaky foundation on which the GA must build.

The idea to overcome this is to bias the probability distribution of chromosomes. In addition to normal mutation, and crossover, we would now also mix in grammar sub-components (from a *gene-pool*) The gene-pool is generated from an algebraic expansion of all the strings in the positive sample.

S		RHS-2				
		S	A	B	**0**	**1**
RHS-1	S	0	0	0	0	0
	A	0	0	0	0	0
	B	0	0	0	0	0
	0	0	0	1	1	0
	1	0	1	0	0	1

A		RHS-2				
		S	A	B	**0**	**1**
RHS-1	S	0	0	0	0	1
	A	0	0	0	0	0
	B	0	0	0	0	0
	0	0	0	0	0	1
	1	0	0	0	0	1

B		RHS-2				
		S	A	B	**0**	**1**
RHS-1	S	0	0	0	1	0
	A	0	0	0	0	0
	B	0	0	0	0	0
	0	0	0	0	1	0
	1	0	0	0	1	0

Table 1: Binary Parsing Tables for PG. We unwind the chromosome as a bit-string by working through each table in turn (from top to bottom), and going through the rows from top to bottom, left to right.

This almost sounds like cheating, but it is not, for the following reason: the biasing process is completely general, employing an algebraic expansion which has no human input, and does not restrict the languages that may be inferred.

Using this technique, grammars that parse all the positive strings evolve quite rapidly; the main problem is then to avoid over-generalisation. There may be scope for creating a pool of 'negative' genes from the negative sample, to help overcome this, but this possibility has not been investigated here.

3.6 Algebraic Expansion

The algebraic expansion of a string gives all the grammars that could have possibly parsed that string, given the set of non-terminals.

Full details of the algebraic expansion may be found in [2], here consider the following example. We take the algebraic parse of the string 0100; this has 45 terms, each corresponding to a possible set of productions that could have been used to parse it. We show one of these 45, with its corresponding grammar below:

```
000000000000010000000000000000000000
000000000000000000000000010000100000
=> ((S -> B 0) (B -> B 0) (B -> 0 1))
```

3.7 Fitness Function

The choice of fitness function is critical to the operation of the GA. For example, without thinking about things too deeply, we might award $+1$ for each positive string parsed, and -1 for each negative string parsed. However, this does not give enough weight to the fact that positive strings are typically much rarer than negative strings, and hence perhaps should be more greatly rewarded. Also, it is desirable to build into the system other factors, such as preferring a small grammar over a larger grammar with the same language score.

However, care should be taken to make the maximum reward for smallness smaller than the minimum reward for superior language recognition. In this paper the following fitness function F was applied to evaluate each chromosome c. F is split into two components: L measures language performance, while W acts as a penalty for excess rules. In the equations below, PS is the positive sample, NS is the negative sample, P is the number of positive strings parsed by the chromosome and N is the number of negative strings parsed.

$$F(c) = L(c) - W(c)$$

where

$$L(c) = \frac{P(c)}{|PS|} - \frac{N(c)}{|NS|}$$

and

$$W(c) = \frac{\Sigma_{i=1}^{|c|} c_i}{|c| * (|PS| + |NS|)}$$

i.e. the 'weight' of the chromosome is simply the number of bits in it set to one, or in other words, the number of production in the grammar, divided by the product of the possible number of productions and the size of the largest sample (positive or negative). This adjustment ensured that the largest possible penalty for being large is smaller than the smallest possible reward for getting an extra production correct.

4 Results

In this Section we report results comparing an unbiased direct encoding with an algebraically biased direct encoding on the toy languages of parity, a regular language, and 2 and 3 symbol palindromes, which are context-free. The goal of the GA is to evolve grammars that parse all the strings in the positive training sample, and none of the strings in the negative training sample. While the fitness function is based entirely on the training sample, we also measure the performance of the inferred grammars on an unseen test set. In all the results better performance on the training set has led to better performance on the test set, and usually (but not always) 100% on the training set implies 100% on the test set.

4.1 Sample Sets

There are two basic alternatives with sample creation: we may either deal with a static sample that is sufficiently large to capture the main types of syntactic structures that characterise the language, or we may use a small dynamically updated sample that would be insufficient for learning purposes at any given point in time, but over time its trajectory takes it through a greater breadth of information than we could reasonably include in the static set.

Here we choose a small static set for training, using all strings of length 2,3 and 4. This allows fast computation of the fitness of each chromosome, and tests the generalisation powers of the GA to the limit. Previous results [1] had proven the possibility of perfect generalisation from such small sample sets.

4.2 Implementation

All the code for the results presented here is written in Sun Common-LISP and all experiments were performed on a Sun SPARC-ELC rated at 22 MIPs. For example, evaluation of a single generation of 40 individuals took between 2 and 3 seconds on the 2-symbol palindrome problem.

The basic tools necessary are the LNF parser, a GA, and a tool to construct the LNF-grammars from the chromosomes. The LNF parser is required to be as fast as possible and must be robust. By robust we mean that it should be able to cope with ANY grammar, no matter how ridiculous without taking excessive time or halting with an error message. Hence, our LNF parser is based on the CYK (matrix) parsing algorithm [7], as for grammars that are highly ambiguous (such as are often produced by the GA) this is much faster than using an Multiple-LR parser [8] or a chart parser (a chart parser is a matrix parser where the matrix entries are represented in sparse form).

4.3 Experimental Details

Each generation consisted of about 40 individuals, from which between 3 and 10 of the best parents were

selected for breeding the next generation (the exact number depended on details of the mutation and crossover rates). In the algebraically biased experiments a special crossover was used, which OR-ed the string with a chromosome from the biased gene-pool.

The mutation and crossover rates were adjusted in each experiment to keep the number of fitness evaluations constant; this avoids giving an unfair advantage to one approach over the other.

Some experimentation was done varying things such as the mutation and crossover rates, and while this was not as varied and as exhaustive as would have been ideal, it did not result in any significant differences from the results shown here (once we take into account the total number of fitness evaluations, rather than the number of generations).

The GA was run for 20 generations each time on the parity problem, and 50 generations each time on the 2 and 3 symbol palindrome problems.

For the ENF chromosomes, productions conforming to ENF were generated at random to produce an initial population. Evolution then proceeded by mutation only (there was insufficient time to implement a crossover operator for this representation). ENF grammars were mutated by removing a production at random, or by adding a random production, with equal probability.

As mentioned above, each training set was a complete sample of strings of length 2,3 and 4. Each test set was a complete sample of strings of length 6 and 7. Hence, the GA is not only having to generalise to unseen strings, but to strings of a length never seen during evolution.

The results in Table 2 show that there is a distinct performance gain in algebraically biasing the chromosomes. However, Table 3 shows that this biasing is insufficient for evolving the more difficult 3-symbol palindromes. The problem here is that the length of the binary chromosome is now 196 bits, and to evolve an a self embedded production in this form requires two chance mutations to happen cooperatively. This becomes increasingly unlikely as the size of the problem (and hence the length of the binary encoding) grows. However, we see that the ENF structuring is very effective, and from the the graph in Figure 1 we see that convergence is normally achieved within well under 10 generations. Incidentally, there ENF structuring was developed after the algebraic biasing, and there has not been time yet to test it on the easier problems in Table 2, but it would be expected to perform poorer than the algebraic method on the parity problem (due to its context-free bias), and much better than the algebraic method on 2-symbol palindromes.

	Training Set		Test Set	
Grammar	Normal	Biased	Normal	Biased
Parity	97 (5.9)	99 (2.1)	92 (11.3)	98 (3.4)
2-Pal	87 (7.3)	97 (5.3)	68 (14.8)	92 (13.4)

Table 2: Percentage accuracy (with standard deviations in parentheses) of the unbiased and algebraically biased GA, based on 10 repetitions of each experiment.

	Training Set		Test Set	
Grammar	ALG	ENF	ALG	ENF
3-Pal	83 (4.4)	100 (0.0)	59 (5.5)	100 (0.0)

Table 3: Percentage accuracy (with standard deviations in parentheses) of the unbiased and algebraically biased GA, based on 10 repetitions of learning 3-symbol palindromes.

It is perhaps worth mentioning that inferring 3-symbol palindromes is generally regarded as a reasonably difficult problem, and stochastic re-estimation methods (such as the inside/outside algorithm) take many hours to learn this, and are prone to getting trapped in local minima [9], but with the ENF structured chromosomes we evolve solutions to this problem reliably in less than a minute.

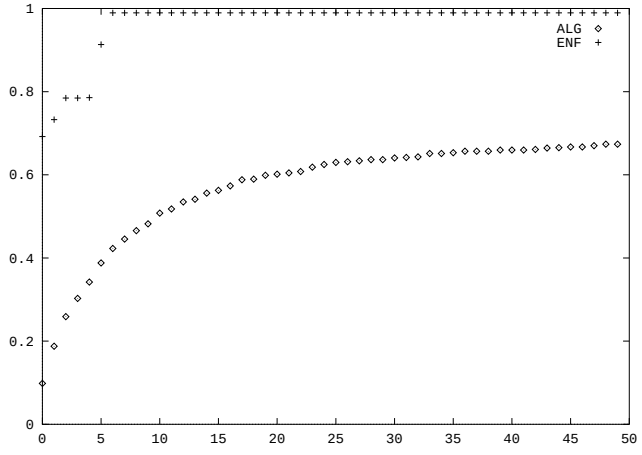

Figure 1: Comparison of algebraically biased chromosomes (ALG) with 'embedded normal form' (ENF) structured chromosomes on 3-symbol palindromes.

5 Conclusions

In this paper we investigated various ways of biasing the chromosomes to improve the process of evolving

context-free grammars to fit and generalise datasets.

Grammatical inference is a difficult problem for genetic algorithms, due to the lack of natural building blocks in binary-encoded grammars. Two approaches were investigated to overcome this. In the first method, the distribution of chromosomes was biased towards chromosomes containing automatically generated partial grammars (i.e. sets of productions that by themselves could parse at least one positive string). This improved the convergence characteristics of the GA, and more significantly, the generalisation performance of the evolved grammars.

However, when dealing with the more difficult 3-symbol palindromes, even this biasing proved ineffective. To overcome this, we biased the chromosomes in a more direct manner – by generating them according to production templates – many of which produced productions in 'embedded normal form'. This proved dramatically more successful. This approach now needs to be tested on a wider range of grammars.

There is something slightly dissatisfying about the ENF structuring,, in that the biasing was generated from high-level human knowledge i.e. from the fact that the very essence of a CFG is self-embedding, and also by examining the most natural form of many CFGs for toy problems (e.g. $a^n b^n$ etc.). More interestingly, the biasing could have been evolved naturally, given suitable chromosome structures, and a suitable environment for the evolution to proceed. This natural biasing will be explored further in a future paper, using the idea of cross-generalisation, but it is worth a mention here.

The idea behind cross-generalisation is that we could evolve grammars for several test languages, and build up probability distribution function of production templates found in the fittest grammars. Then we could test the effect of using evolved-biased chromosome structures on unseen languages. Ultimately, these techniques may provide answers to some fundamental problems in grammatical inference. For example, many theoretical results exist that show that non-trivial context-free grammars should be practically unlearnable. Yet people learn them without difficulty. This apparent contradiction might be due to the fact that we are biased (i.e. we have evolved to be biased) to learn classes of natural language, for example, rather than the toy languages presented here.

Finally, it is worth emphasising the main result of this paper: that the design of the chromosome has a far more profound effect on the performance of evolutionary computation than messing about with algorithmic details of the GA.

References

[1] S. Lucas, "An algebraic approach to learning in syntactic neural networks," in *Proceedings of the International Joint Conference on Neural Networks (Baltimore '92)*, pp. I, 877 – 882, San Diego, CA: IEEE, (1992).

[2] S. Lucas, "Algebraic grammatical inference," in *Proceedings of IEE Colloquium Grammatical Inference: Theory, Applications and Alternatives, 22nd-23rd April 1993, Digest No. 1993/092*, London: IEE, (1993).

[3] P. Wyard, "Context-free grammar induction using genetic algorithms," in *Proceedings of the fourth international conference on Genetic Algorithms* (R. Belew and L. Booker, eds.), pp. 514 – 518, San Mateo, CA: Morgan Kaufman, (1991).

[4] H. Kitano, "Designing neural networks using genetic algorithm with graph generation system," *Complex Systems*, vol. 4, pp. 461 – 476, (1990).

[5] F. Gruau, "Cellular encoding of genetic neural networks," *Laboratoire de l'Informatique du Parallelisme Technical Report 92-21, Ecole Normale Superieure de Lyon*, (1992).

[6] E. Boers and H. Kuiper, "Biological metaphors and the design of modular artificial neural networks," *Masters thesis, Department of Computer Science and Experimental and Theoretical Psychology, Leiden University, the Netherlands*, (1993).

[7] D. Younger, "Recognition and parsing of context-free languages in time n^3," *University of Hawaii Technical Report, Department of Computer Science*, (1967).

[8] M. Tomita, *Efficient Parsing for Natural Language: a fast algorithm for practical systems.* Dordrecht, Netherlands: Kluwer, (1985).

[9] K. Lari and S. Young, "The estimation of stochastic context-free grammars using the inside–outside algorithm," *Computer Speech and Language*, vol. 4, pp. 35 – 56, (1990).