Learning Stochastic Context-Free Grammars from Corpora Using a Genetic Algorithm

Bill Keller and Rüdi Lutz
School of Cognitive and Computing Sciences
The University of Sussex
email: {billk,rudil}@cogs.susx.ac.uk

Abstract

A genetic algorithm for inferring stochastic context-free grammars from finite language samples is described. Solutions to the inference problem are evolved by optimizing the parameters of a covering grammar for a given language sample. We describe a number of experiments in learning grammars for a range of formal languages. The results of these experiments are encouraging and compare very favourably with other approaches to stochastic grammatical inference.

Keywords: grammatical inference, genetic algorithms, stochastic grammar, formal languages, induction.

1 Introduction

This paper describes an evolutionary approach to the problem of inferring stochastic context-free grammars from finite language samples or *corpora*. Grammatical inference [Go78] is a fundamental problem in many areas of artificial intelligence and cognitive science, including speech and language processing, syntactic pattern recognition and automated programming. Although a wide variety of techniques for automated grammatical inference have been devised (for surveys see [FB86, AS83]) most are subject to limitations which severely restrict their range of application. For example, inference may be limited to grammars for the regular languages, or require access to both positive (grammatical) and negative (ungrammatical) instances of the target language. A goal of the present work is to provide a general method for inferring a wide class of grammars on the basis of just positive information about the target language.

Genetic algorithms [Ho75] are a family of robust, probabilistic optimization techniques that offer advantages over specialised procedures for automated grammatical inference. A number of researchers have already described applications of genetic algorithms to language identification problems with some success [ZG86, Lu93, SJ92, La94, Wy91, DPB94, Hu93, SO95]. However, with the exception of recent work reported by Schwem and Ost [SO95], the problem of inferring *stochastic* language models has not been addressed. This is surprising in view of the many practical applications of such models to tasks including

S → A B	(1.0)
A → a	(0.6)
A → C S	(0.4)
B → b	(1.0)
C → a	(1.0)

Figure 1: SCFG for the language $a^n b^n$ $(n \geq 1)$

speech recognition, part-of-speech tagging, optical character recognition and robust parsing. While Schwem and Ost recognize the importance of the stochastic inference problem, their approach is restricted to the inference of stochastic regular grammars. In contrast, the present work tackles the more general problem of inferring stochastic grammars for the class of context-free languages.

The following sections describe our approach to grammatical inference in more detail. Stochastic context-free grammars are briefly described in section 2. Section 3 discusses the problem of inferring stochastic grammars from corpora. Details of the genetic algorithm are then given in section 4 and in section 5 we present the results of several experiments in learning grammars for a range of formal languages.

2 Stochastic Context-Free Grammars

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 *consistent*, the probabilities associated with all rules that expand the same non-terminal symbol must sum to 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 (typically, S). 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 α 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 α is the sum of the probabilities of all of its parses.

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

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_α 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:

$$N_C = \sum_{\alpha \in C} f_\alpha$$

The *relative frequency* p_α of a string $\alpha \in C$ is defined as $p_\alpha = f_\alpha / N_C$

ab	595
$aabb$	238
$aaabbb$	97
$aaaabbbb$	49
$aaaaabbbbb$	14
$aaaaaabbbbbb$	5

Figure 2: A Corpus for the Language $a^n b^n$

An example of a corpus for the language $\{a^n b^n | n \geq 1\}$ is shown in figure 2. The frequency of the string ab is 595, the frequency of $aabb$ is 238, and so on. The total size of the corpus is 998. The relative frequencies of the strings ab and $aabb$ are given by $p_{ab} = 0.596192$, $p_{aabb} = 0.238477$.

3 Corpus-Based Grammatical Inference

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. For a stochastic grammar, a natural measure of accuracy is the likelihood of the corpus data given the grammar. The most accurate model in this sense is that SCFG $\hat{G}$ given by

$$\hat{G} \quad = \quad argmax_G \, P(C|G) \tag{1}$$

where $P(C|G)$ (the conditional probability of the language data C given the grammar G) is defined as:

$$P(C|G) = \prod_{\alpha \in C} P_G(\alpha)^{f_\alpha}$$

It may be observed that the grammar maximizing the likelihood of the corpus data according to equation 3 will not generally meet the further requirement of *generalization*. To see this, note that a perfectly accurate model of a finite corpus is one which generates exactly the finite corpus and assigns to each string the correct relative frequency. In other words, the most accurate grammar will *over-fit* the training data. Intuitively, what we actually require is the grammar that is most likely given the training data. That is, we wish to find $\hat{G}$ such that

$$\hat{G} \quad = \quad argmax_G \, P(G|C) \tag{2}$$

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

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

Ignoring $P(C)$, which is a constant, maximising $P(G|C)$ just corresponds to maximising the product of $P(C|G)$ (which we can calculate directly) and $P(G)$, the prior probability of the grammar G. Of course, this poses the problem of fixing an appropriate prior probability distribution over grammars. In principle there are many different priors that could be chosen. Other things being equal, however, it seems reasonable to assume that we should prefer smaller or simpler grammars to larger, more complex ones. Our choice of prior is related to the *minimum description length* principle of Risannen [Ri78] as well as earlier work on inductive inference due to Solomonoff [So64]. In outline, we first fix a probability distribution over (parameterized) grammar rules, such that shorter rules are more probable than longer rules. The prior probability of a grammar is then obtained simply as the product of the prior probabilities of all of its rules.

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

$$F(G) = \frac{-K_C}{\log P(C|G) + \log P(G)} \tag{3}$$

Note that minimizing the denominator in 3 (ignoring sign) just amounts to maximising $P(G|C)$. The numerator $-K_C$ is a problem (corpus) dependent normalization factor that yields fitness values in the range [0;1].

4 The Genetic Algorithm

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 (cross-over, mutation) to selected individuals in the population until an optimal set of parameters is found.

Step one involves first fixing a set of non-terminals to be used in the covering grammar. The grammar itself is a large context-free grammar in Chomsky Normal Form (CNF), which contains every rule of the form $A \rightarrow BC$ (where A, B and C are non-terminals) and every rule of the form $A \rightarrow a$ where A is a non-terminal and a a terminal symbol appearing in the corpus. By restricting attention to CNF grammars over finite sets of terminal and non-terminal symbols, the covering grammar is guaranteed to be finite[1]. On the other hand, there is no loss of generality: for any SCFG (not generating the empty string) there is a SCFG in CNF which generates exactly the same language with exactly the same probability distribution.

[1]For a covering grammar in CNF with n nonterminals and m terminals, there will be $n^3 + nm$ productions.

The population maintained by the genetic algorithm is organized as a two dimensional grid, with opposing sides of the grid identified (i.e. members of the population inhabit the surface of a torus). Thus, each member of the population has exactly eight neighbours. A member of the population encodes a set of parameters for the rules of the covering grammar, with each parameter encoded as a fixed-length bit string. If a block of n bits is used to encode each parameter, then for a covering grammar having M rules each member of the population has a genome of length Mn bits, where the jth parameter is encoded as the jth n-bit block.

Because the parameters of a SCFG are not independent of one another, we do not encode their values directly. Instead, each n-bit block is treated as an encoding of a numerical *weight*. To obtain the actual parameters of the SCFG, the weights are normalized as part of the decoding process. If w_j is the weight associated with rule r_j, then $p_j = w_j/W$ is the probability associated with this rule (where W is the sum of all those weights associated with rules expanding the same non-terminal as r_j). A weight of zero means that the corresponding rule does not belong to the rule-set of the SCFG.

An obvious scheme for encoding the weights is to treat each n-bit block as a binary representation of an integer value. However, this simple scheme has the drawback that it makes it relatively unlikely that a rule will be assigned a zero weight. In general, the covering grammar has many more rules than are required for the target SCFG, so it makes sense to use an encoding that is biased in favour of rules having zero weight rather than the other way around. To achieve this, the encoding scheme is modified by reserving a small number of initial bits in each block. If each of these initial bits is set to 1, then the remaining bits are decoded to obtain the rule weight. In all other cases the rule ¸weight is zero, and the remaining bits are ignored. The number of reserved bits effectively controls the amount of bias in favour of a rule being assigned a weight of zero, while the number of remaining bits controls the size of the rule weights and thus the precision of the rule probabilities. For the experiments described in the following section we have found that between one and three initial bits and 7 'weight' bits is sufficient[2].

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, and choose the fittest of its eight neighbours as the second parent.

Breed by applying crossover and mutation to produce two children.

Replace the weakest parent by the fittest child.

Selection and replacement are carried out within a small, local population. This allows for rival solutions to emerge at different locations and discourages too rapid a spread of successful genetic information throughout the whole population. By replacing the weakest parent, rather than the weakest neighbour, relatively unfit individuals still get a chance at breeding while useful genetic material from the weakest parent may survive through the fittest child.

[2]The actual number of initial bits is determined automatically in proportion to the size of the covering grammar. The larger the grammar, the more weight bits are used.

A characteristic of our parameter encoding scheme is that the probability associated with any given rule does not depend solely on local properties of the genome (i.e. the state of the relevant n-bit block). In general, it will also depend on the state of all the weight encodings associated with rules expanding the same non-terminal symbol. Furthermore, the fitness of a given individual may be crucially dependent on the state of weight encodings that are widely separated within the genome. In short, our representation is one which exhibits high epistasis and where global properties of the genome are in many ways more important than local properties. This presents a problem, because such global characteristics are more likely to be destroyed than preserved under the classical, one-point crossover operator.

We have experimented with a number of alternatives to the classical crossover operator. Good performance has been achieved using a novel genetic operator which we refer to as *and-or crossover*. As its name suggests, this works by inspecting corresponding positions in the parent's genomes and then performing the logical operations of *and* and *or*. Two children are built up bit-by-bit, with one child selected (with some crossover probability) to receive the value returned by the *and* operation, and the other the value of the *or*. Best results are obtained when the crossover probability is itself randomly generated at the start of each breeding phase. This operator preserves shared characteristics in the genomes of the two parents (via logical *and*), whilst permitting useful intermingling of differing characteristics (via logical *or*). A desirable property of *and-or* crossover from out point of view is that is is capable of preserving fit schemata of arbitrary length. The genetic algorithm employs a standard, point-wise mutation operator, which is performed with low probability[3].

5 Experimental Results

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

1. **EQ**: the language of all strings consisting of equal numbers of as and bs;
2. the language $a^n b^n$ $(n \geq 1)$;
3. **BRA1**: the language of balanced brackets;
4. **BRA2**: balanced brackets with two sorts of bracketing symbols;
5. **PAL1**: palindromes over $\{a, b\}$
6. **PAL2**: palindromes over $\{a, b, c\}$

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 16,000 strings up to a pre-specified 'maximum sentence length' (typically 6 or 8). For the covering

[3]The mutation rate is set inversely proportional to the length of the genome

Language	Nonterminals	Parameters	Successful	Best	Worst
EQ	3	33	9/10	0.971	0.679
$a^n b^n$	4	72	10/10	0.979	0.941
BRA1	3	33	10/10	0.956	0.951
BRA2	5	145	9/10	0.957	0.622
PAL1	5	135	2/10	0.950	0.871
PAL2	7	364	1/3	0.937	0.892

Figure 3: Results on a number of language learning tasks

grammar, the number of non-terminal symbols was fixed as the number used in the hand-crafted SCFG[4]. For each problem, the population size was determined automatically in proportion to the number of parameters (i.e. rules) in the covering grammar. We have found that setting the population size to twice the number of parameters gives good results.

In order to assess the performance of the genetic algorithm, multiple runs were completed on each language learning task. With the exception of PAL2 (three symbol palindromes), ten runs each of the genetic algorithm were performed. For PAL2, the current implementation requires considerable processor time and for this reason only three runs were executed. 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 almost invariably 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 in figure 3. For each learning task, the table gives the number of nonterminals 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 worst runs of the genetic algorithm. As can be seen, the first four tasks (EQ, $a^n b^n$, BRA1 and BRA2) presented little difficulty. In particular, for $a^n b^n$ and BRA1 all ten runs in each case terminated successfully. Inspection of the grammars produced for these experiments showed that they were indeed correct[5]. For EQ and BRA2, one run in each case failed to produce an adequate grammar. The relatively poor fitness values attained on these runs (0.679 and 0.622 respectively) suggests the presence of local maxima around which the population has converged.

[4]For more realistic problems, where the target grammar is not known in advance, this would not be possible. We are currently conducting further experiments to assess the effects of introducing surplus non-terminals.

[5]Occassionally, a grammar contained an additional, spurious rule with 'near zero' probability. A post-processing phase to prune these rules could easily be implemented.

S → C B	(0.51875)
S → D A	(0.48125)
A → *a*	(1.0)
B → S C	(0.24778)
B → *b*	(0.752212)
C → *b*	(1.0)
D → A S	(0.252066)
D → A C	(0.066116)
D → *a*	(0.681818)

Figure 4: Near-miss grammar for the language of two symbol palindromes

The results for the two palindrome languages initially appear less encouraging. For PAL1, only two runs attained the threshold fitness value, while for PAL2 only one of the three runs was terminated successfully. On the other hand, even on the worst runs in each case the algorithm found grammars with quite high fitness. Furthermore, it should be recalled that the threshold fitness value represents an arbitrary measure of success. In particular, failure to attain this threshold does not imply that the algorithm has failed to find a grammar with a correct (or nearly correct) set of rules. For example, it is possible that the grammar generates the target language exactly, but with a non-optimal probability distribution.

Inspection of the grammars produced for all runs of the PAL1 learning task showed that the algorithm had performed rather better than suggested by the 2/10 success rate given in the table. Figure 5 shows the grammar ranked fifth best (with a fitness of 0.897355) out of all those produced by the algorithm on this task. Aside from the presence of one spurious production D → A C, which has a low associated probability (0.066116), the grammar is otherwise correct. Indeed, it was found that the five fittest grammars produced by the algorithm were all of this type. Similar comments apply in the case of PAL2. In this case the second-ranked grammar actually achieved a fitness of 0.925312, narrowly missing the threshold value.

6 Conclusion

The approach to grammatical inference described in this paper differs from previous work using genetic algorithms in addressing the problem of corpus-based inference of stochastic context-free grammar. This difference makes direct comparison of our results with those of other researchers difficult. However, the experiments that we have conducted are typical of those in other studies and the results reported in this paper appear promising. The approach also compares well with other (non-genetic) techniques for stochastic grammatical inference, for example the work reported by Lari and Young [LY90] using the Inside-Outside algorithm [Ba79].

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. 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 non-terminals. In the current implementation, this makes the cost prohibitive for CNF covering grammars with more than about 8 non-terminal symbols. We are currently investigating ways of overcoming this problem, including the possibility of a massively parallel implementation of our algorithm.

References

[AS83] Angluin, D. and C. Smith (1983) Inductive inference: theory and methods, *Computing Surveys*, 15(3), 237–269.

[Ba79] Baker, J.K. (1979) Trainable grammars for speech recognition. *Proceedings of the Spring Conference of the Acoustical Society of America*, 547–550, Boston, MA.

[DPB94] Dunay, D., F. Petry and B. Buckles (1994) Regular language induction with genetic programming, *Proceedings of the First International Conference on Evolutionary Computing*, 396–400.

[FB86] Fu, K.S. and T.L. Booth (1986) Grammatical inference: introduction and survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8, 343–375.

[Go78] Gold, E. M. (1978) Language identification in the limit, *Information and Control*, 10, 447–474.

[Ho75] Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

[Hu93] Huijsen, W.-0. (1993) *Genetic Grammatical Inference: Induction of Push-down Automata and Context-Free Grammars from Examples Using Genetic Algorithms*. Master's thesis, Dept. of Computer Science, University of Twente, Enschede, The Netherlands.

[La94] Lankhorst, M.M. (1994) Grammatical inference with a genetic algorithm, in L.Dekker, W.Smit and J.C.Zuidervaart (eds.), *Proceedings of the 1994 EU-ROSIM Conference on Massively Parallel Processing Applications and Development*, 423-430, Delft, The Netherlands.

[LY90] Lari, K. and S.J. Young (1990) The estimation of stochastic context-free grammars using the inside-outside algorithm, *Computer Speech and Language*, 5, 237–257.

[Lu93] Lucas, S. (1993) Biased chromosomes for grammatical inference, *Proceedings of Natural Algorithms in Signal Processing*, IEE Workshop, Danbury Park, UK.

[Ri78] Risannen, J. (1978) Modelling by shortest data description, *Automatica*, 14, 465-471.

[SO95] Schwem, M. and A. Ost (1995) Inference of stochastic regular grammars by massively parallel genetic algorithms, *Proceedings of the Sixth International Conference on Genetic Algorithms*, 520–527, Morgan-Kaufmann, CA.

[SJ92] Sen, S. and J. Janakiraman (1992) Learning to construct pushdown automata for accepting deterministic context-free languages, in G.Biswas (ed.), *SPIE Vol. 1707: Applications of Artificial Intelligence X: Knowledge-Based Systems*, 207–213.

[So64] Solomonoff, R.J. (1964) A formal theory of inductive inference, *Information and Control*, 7, 1–22; 224–254.

[Wy91] Wyard, P. (1991) Context-free grammar induction using genetic algorithms, in R.Belew and L.B. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms, ICGA'92*, 514–418, Morgan Kaufmann. CA.

[ZG86] Zhou, H. and J.J. Grefenstette (1986) Induction of finite automata by genetic algorithms, *Proceedings of the 1986 IEEE International Conference on Systems, Man and Cybernetics*, 170–174, Atlanta, GA.