A Genetic Algorithm for the Induction of Context-Free Grammars *

Marc M. Lankhorst
Dept. of Computing Science,
University of Groningen,
e-mail: `lankhors@cs.rug.nl`

Abstract

This paper presents a genetic algorithm used to infer context-free grammars from legal and illegal examples of a language. It discusses the representation of grammar rules in the form of bitstrings by way of an interval coding scheme, genetic operators for reproduction of grammars, and the method of evaluating the fitness of grammars with respect to the training examples.

Results are reported on the inference of several of these grammars. Grammars for the language of correctly balanced and nested brackets, the language of sentences containing an equal number of a's and b's, a set of regular languages, and a micro-NL language were inferred. Furthermore, some possible improvements and extensions of the algorithm are discussed.

1 Introduction

Genetic algorithms, introduced by Holland Holland (1975), are probabilistic search techniques especially suited for difficult search and optimization problems where the problem space is large, complex and contains possible difficulties like high dimensionality, multimodality, discontinuity and noise. Applied to such problems, genetic algorithms consistently outperform both gradient techniques and various forms of random search (Bethke, 1981). They manipulate a population of strings defined over some alphabet, which encode possible solutions to the problem at hand. These alternative solutions are being processed in parallel, giving the algorithm the ability to efficiently search a large problem space for global optima.

An example of such a difficult optimization task is grammatical inference, the problem of learning a grammar based on a set of sample sentences (Fu and Booth, 1986). Many researchers have attacked this problem (for a survey, see (Angluin and

*Most of the computations were carried out on the Connection Machine CM-5 of the University of Groningen, the investments in which were partly supported by the Netherlands Computer Science Research Foundation (SION) and the Netherlands Organization for Scientific Research (NWO).

Smith, 1983)), e.g. trying to induce finite-state automata to accept regular languages (Berwick and Pilato, 1987) or to learn context-free grammars directly from examples (VanLehn and Ball, 1987). Genetic algorithms have been applied to the induction of finite-state automata (Zhou and Grefenstette, 1986), context-free grammars (Wyard, 1992), and push-down automata (Sen and Janakiraman, 1992; Huijsen, 1993).

This paper presents a genetic algorithm that is used to infer context-free grammars from legal and illegal examples of a context-free language.

2 Genetic Algorithms

Genetic algorithms are search and optimization techniques inspired by the "survival of the fittest" principle of natural evolution. A genetic algorithm maintains a population $P(t) = \langle x_1(t), \ldots x_n(t) \rangle$ of candidate solutions $x_i(t)$ to the objective function $F(x)$, represented in the form of "chromosomes". These chromosomes are strings defined over some alphabet that encode the properties of the individual. More formally, using an alphabet $A = \{0, 1, \ldots, k-1\}$ and a value-based encoding, we define a chromosome $C = \langle c_1, \ldots, c_l \rangle$ of length l as a member of the set $S = A^l$. Each element c_i of C is called an *allele*, and the subscript i itself is called the *locus number* of that allele.

At each step t of the algorithm—called a generation—the fitness f_i of the individuals x_i is evaluated with respect to the optimization criterion; the fittest individuals are then selected and allowed to reproduce in order to create a new population. A sketch of the algorithm is shown in Figure 1.

Usually, reproduction is performed using two operations: *crossover* and *mutation*. Crossover is used to create offspring from two parent individuals by exchanging parts of their chromosomes, which can be performed in various ways. An example of one-point and two-point crossover on bitstrings is given in Figure 2 and Figure 3, respectively. Subsequently, mutation may be applied to individuals by randomly changing pieces of their representations, as shown in Figure 4.

The genetic algorithm may be terminated if a satisfying solution has been obtained, after a predefined number of generations, or if the population has converged to a certain level of genetic variation.

The operation of a genetic algorithm is very simple. It starts with a random population of n strings, copies strings with some bias toward the best, mates and partially swaps substrings, and randomly mutates a small part of the population. On a deeper level, this explicit processing of strings causes an implicit processing of *schemata*. A schema (Holland, 1975) is a similarity template describing a subset of strings with similarities at certain string positions. Each schema is represented as a list made up of characters from the set $A \cup \{\#\}$. A character from A (i.e., an allele) at any position in the schema means that the value of the chromosome must have the same value at that position for it to contain the schema. The #s function as "don't cares", i.e., a # at any position in the schema means that the value of the chromosome at that position is irrelevant to determine whether the chromosome contains the schema.

It is easy to see that if we have an alphabet of size k, there are $(k+1)^l$ schemata

```
procedure Generational GA
begin
        t := 0;
        initialize P(t)
        evaluate structures in P(t);
        while termination condition not satisfied do
                t := t + 1;
                select P(t) from P(t - 1);
                recombine structures in P(t);
                evaluate structures in P(t)
        end
end;

procedure recombine
begin
        for i := 1 to population_size/2 do
            pick mom, dad from P(t);
            kids := crossover mom,dad based on crossover rate;
            mutate kids based on mutation rate
        end
end
```

Figure 1: A Genetic Algorithm

$$\left.\begin{array}{c} \circ\bullet\bullet\circ\bullet|\bullet\bullet\circ\circ\bullet\circ \\ \bullet\circ\circ\bullet\circ|\circ\bullet\circ\circ\circ\bullet \end{array}\right\} \Rightarrow \left\{\begin{array}{c} \circ\bullet\bullet\circ\bullet|\circ\bullet\circ\circ\circ\bullet \\ \bullet\circ\circ\bullet\circ|\bullet\bullet\circ\circ\bullet\circ \end{array}\right.$$

Figure 2: One-point crossover

$$\left.\begin{array}{c} \circ\bullet|\bullet\circ\bullet\bullet\bullet|\circ\circ\bullet\circ \\ \bullet\circ|\circ\bullet\circ\circ\bullet|\circ\circ\circ\bullet \end{array}\right\} \Rightarrow \left\{\begin{array}{c} \circ\bullet|\circ\bullet\circ\circ\bullet|\circ\circ\bullet\circ \\ \bullet\circ|\bullet\circ\bullet\bullet\bullet|\circ\circ\circ\bullet \end{array}\right.$$

Figure 3: Two-point crossover

$$\begin{array}{c} \circ\bullet\bullet\circ\bullet\bullet\bullet\circ\circ\bullet\circ \\ \Downarrow \\ \circ\bullet\bullet\circ\bullet\circ\bullet\circ\circ\bullet\circ \end{array}$$

Figure 4: Single mutation

```
01101
0##01
#11#1
#####
```

Figure 5: Examples of schemata in a single bit string

defined on strings of length l. Every string is a member of 2^l different schemata: each locus can have an allele or a don't care. It follows that a population of n members may contain at most $n \cdot 2^l$ schemata. Figure 5 shows a single chromosome and some of the schemata represented in that chromosome. Just like chromosomes represent points in the search space, schemata represent hyperplanes. A hyperplane consists of those points that are represented by the strings that match its accompanying schema.

If we use a reproduction technique that makes reproduction chances proportional to chromosome fitness, then we can use Holland's *Schema Theorem* or Fundamental Theorem of Genetic Algorithms (Holland, 1975) to predict the relative increase or decrease of a schema in the next generation of the genetic algorithm.

In effect, the Schema Theorem says that a schema occurring in chromosomes with above-average evaluations will tend to occur more frequently in the next generation, and one occurring in chromosomes with below-average evaluations will tend to occur less frequently. Since short schemata are less likely to be disrupted by mutation and crossover than longer ones, genetic algorithms tend to construct new chromosomes from small, contiguous pieces of chromosomes that score above-average fitness.

This feature of genetic algorithms, also called the *building blocks hypothesis*, is the main motivation for applying GAs to the problem of grammatical inference. Parts of different grammars and rules can be recombined to form new, and possibly better performing grammars and rules.

3 Representation

We have used an interval encoding that represents a vector of integers in one bit-string encoded number. Each grammar rule is taken as a vector of symbol numbers (with a fixed maximum length n). We can encode such a vector $\mathbf{p} = [p_1, \ldots, p_n]$ with $0 \leq p_i < m_i$ by successively subdividing the interval $[0, 1)$ in m_k equal subintervals. At each level k, the subinterval corresponding to p_k is used as the basis for the next stage, i.e., if we want to encode an integer p_k at stage k, we choose the subinterval $[a_{k+1}, b_{k+1}) \subset [a_k, b_k)$ with $a_{k+1} = a_k + p_k \cdot (b_k - a_k)/m_k$ and $b_{k+1} = a_k + (p_k + 1) \cdot (b_k - a_k)/m_k$. With $a_1 = 0$ and $b_1 = 1$, the number $E(\mathbf{p})$ that encodes the complete vector is given by:

$$E(\mathbf{p}) \;=\; \sum_{i=1}^{n} \left(p_i \cdot \prod_{j=1}^{i} \frac{1}{m_j} \right) \qquad (1)$$

4 Genetic Operators

In our algorithm, we have used the following genetic operators.

- Selection: Selection is based on a ranking algorithm, i.e., in each generation the individuals are sorted by fitness, and the probability of selecting an individual for reproduction is proportional to its index in the sorted population. The selection itself is carried out by a stochastic universal sampling algorithm (Baker, 1987). This ranking algorithm is used to help prevent premature convergence by preventing "super" individuals from taking over the population within a few generations. Furthermore, we employ an elitist strategy in which the best individual of the population always survives to the next generation.

- Mutation: Following the heuristic given by Bäck (Bäck, 1993), we have used a mutation rate of $1/\ell$ throughout all the experiments (ℓ being the number of bits in the chromosome), i.e., each bit in each chromosome has a probability of 1 in ℓ of being mutated.

- Reproduction: The representation scheme we used, allowed crossover operations to break the right-hand side of production rules. Crossover within an individual right-hand side of a production rule has the advantage of creating a larger variety of right-hand sides in the population. Furthermore, it might be beneficial to construct a new right-hand side from parts of the parents' right-hand sides. Not allowing this crossover to occur, (Wyard, 1992) had to rely on mutation alone to change the right-hand sides of production rules.

  We have used two-point crossover, which has some theoretical advantages over one-point (DeJong, 1975), and a crossover probability of 0.9.

5 Fitness Evaluation

The most important issue in constructing a genetic algorithm is the choice of a particular evaluation function. Suppose we have sets S_{POS} of positive and S_{NEG} of negative examples of a language $\mathcal{L}$, and a grammar $G = \langle N, \Sigma, P, S \rangle$. Defining the fraction of correctly analyzed sentences as follows

$$\begin{aligned} cor(G,\sigma) &= \begin{cases} 1 & \text{if } \sigma \in \mathcal{L} \cap L(G) \text{ or } \sigma \in \overline{\mathcal{L}} \cap \overline{L(G)} \\ 0 & \text{otherwise} \end{cases} \\ cor(G,S) &= \frac{1}{|S|} \sum_{\sigma \in S} cor(G,\sigma) \end{aligned} \tag{2}$$

a simple evaluation function would be

$$F_1(G, S_{POS}, S_{NEG}) = cor(G, S_{POS}) \times cor(G, S_{NEG}) \tag{3}$$

which yields fitness values between 0 and 1.

However, to evaluate the fitness of a particular grammar with respect to the positive and negative training examples, it does not suffice to simply count the correctly accepted (rejected) positive (negative) examples. In this way, a grammar that can analyze large parts of the examples correctly, but fails to recognize the complete sentences, would receive a very low fitness value. Although recombination of such a partially correct grammar might yield a better result, this low fitness will cause it to be thrown away, thereby destroying valuable information.

To evaluate a grammar, we would like to credit correctly analyzed substrings of each positive training example $a_1 \ldots a_n$. To do so, we can look at the length of the longest substring derivable from a single nonterminal

$$\begin{aligned} &sub(G, a_1 \ldots a_n) = \\ &\quad MAX\{l \mid 0 \leq l \leq n \wedge \exists A \in N : \exists i : 0 \leq i \leq n - l \wedge A \Rightarrow^* a_{i+1} \ldots a_{i+l}\} \end{aligned} \tag{4}$$

and, with $m(S)$ being the maximum sentence length of S, we can take the normed total over the set of sentences:

$$sub(G, S) = \frac{1}{m(S) \cdot |S|} \sum_{\sigma \in S} sub(G, \sigma) \tag{5}$$

Naturally, this is only meaningful for legal examples, so we define our new evaluation function to be

$$F_2(G, S_{POS}, S_{NEG}) = (cor(G, S_{POS}) + sub(G, S_{POS})) \times cor(G, S_{NEG}) \tag{6}$$

A second aspect of the quality of a grammar consists of its predictions on the next symbol a_{k+1} of a string, given the previous symbols $a_1 \ldots a_k$. The more accurate these predictions are, the tighter the grammar fits the sentence. Hence, this information might be helpful in rejecting grammars that are too permissive. A criterion for this accuracy is given by:

$$pred(G, a_1 \ldots a_n) = \frac{1}{n+1} \sum_{j=0}^{n} \frac{1}{|\{\alpha \in \Sigma \mid \exists \delta \in V^* : S \Rightarrow^* a_1 \ldots a_j a \delta\}|} \tag{7}$$

and

$$pred(G, S) = \frac{1}{|S|} \sum_{\sigma \in S} pred(G, \sigma) \tag{8}$$

Incorporating this into our evaluation function, we get

$$\begin{aligned} &F_3(G, S_{POS}, S_{NEG}) = \\ &\quad (cor(G, S_{POS}) + sub(G, S_{POS}) + pred(G, S_{POS})) \times cor(G, S_{NEG}) \end{aligned} \tag{9}$$

Furthermore, we might include information on the generative capacity of a grammar. We can use the grammar G to generate a set of strings $S_{GEN}(G)$ and test whether

1 “brackets”;
2 “AB”;
3 (10)*;
4 no odd zero strings after odd one strings;
5 no triples of zero’s;
6 pairwise, an even sum of 01’s and 10’s;
7 number of 1’s - number of 0’s = $3n$;
8 0*1*0*1*;
9 “micro-NL”.

Figure 6: The test languages

these strings belong to the language. We can augment our evaluation function with this information, obtaining

$$\begin{aligned} &F_4(G, S_{POS}, S_{NEG}) = \\ &\quad F_3(G, S_{POS}, S_{NEG}) \times cor(S_{GEN}(G)) \end{aligned} \tag{10}$$

Unfortunately, this requires a teacher with prior knowledge of the underlying structure of the language for which we want to infer a grammar.

6 Results

The genetic algorithm has been tested with 9 different languages, which are listed in figure 6 and discussed in the following sections. For each experiment, we have randomly generated an equal number of positive and negative examples, with a Poisson-like length distribution and a maximum sentence length of 30. An example of this distribution for the “AB” language is shown in figure 7. Domain knowledge was used to determine the terminal symbols and the size—i.e. the maximum size of the right-hand sides of rules and the number of rules and nonterminals—of the grammars to be inferred. These features could also be encoded on the chromosomes, but that would impose an extra computational burden upon the genetic algorithm.

The implementation of the genetic algorithm we used was based on Genesis 5.0, a genetic algorithm package written by John J. Grefenstette (Grefenstette, 1990). We ported it to a 16-node Connection Machine CM-5 by parallelizing the evaluation of the chromosomes. This was done using a master-slave programming model, in which the host (the master) executed the genetic algorithm, and the CM-5 nodes (the slaves) conducted the chromosome evaluation. At each generation, this meant parsing several hundred example sentences per chromosome. Since parsing consists mainly of symbol manipulation, we could not use the extensive floating point processing capabilities of the CM-5. Hence we only employed the Sparc processor of each of the nodes. Despite the computing power of the CM-5, the longest run of the algorithm took about two days to complete.

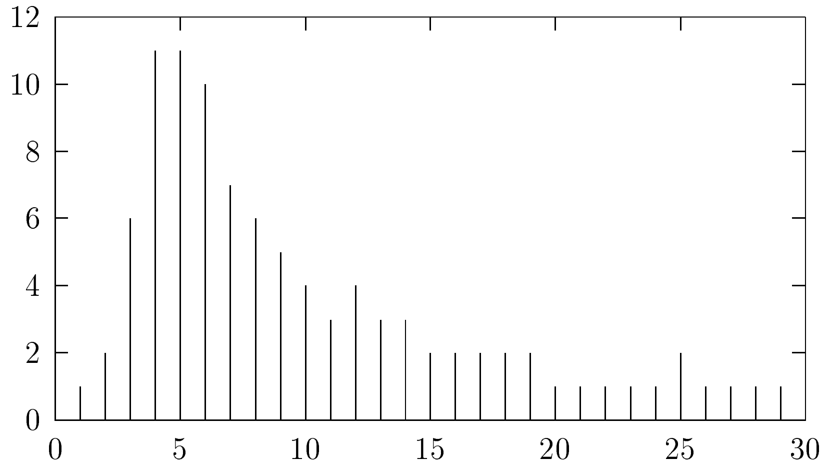

Figure 7: Sentence length distribution for the "AB" language

6.1 The "Brackets" Language

The "Brackets" language consists of all correctly nested and balanced brackets expressions. We performed five runs, each with a population of 48 chromosomes, 100 legal and 100 illegal example strings. The maximum number of rules per grammar was set at 5, the size of the right-hand side of the grammar rules was fixed at 2, and the grammars were allowed to have two nonterminal symbols other than the start symbol S.

$$
\begin{array}{lcl}
\mathrm{S} & \rightarrow & \mathrm{A} \\
\mathrm{A} & \rightarrow & (\ \mathrm{B} \\
\mathrm{A} & \rightarrow & \epsilon \\
\mathrm{B} & \rightarrow & \mathrm{A}\ ) \\
\mathrm{B} & \rightarrow & \mathrm{A}\ \mathrm{B}
\end{array}
$$

Figure 8: Grammar inferred for the "Brackets" language

Since every string of brackets can be followed by a '(' or a ')' in some sentence of the language, we have used evaluation function F_2 (definition 6), which does not contain information on the predictive qualities of grammars. Every run resulted in a correct grammar, which took on average 592 generations. As an example, one of the inferred grammars is given in figure 8.

6.2 The "AB" Language

This experiment was conducted on the "AB" language, which consists of all sentences containing equal numbers of a's and b's. Four different runs were performed, each with a population size of 48 chromosomes, 100 positive and 100 negative example strings. The maximum number of rules per grammar was fixed at 11, and the maximum size of the right-hand side of the grammar rules was 3.

For reasons similar to those stated in the previous section, the evaluation function we employed was F_2 (see definition 6). In all five runs a correct grammar of the "AB" language was found, after 271 generations on average. An example of such a grammar is shown in figure 9. In this example, we have deleted multiple occurrences of rules and inaccessible rules.

$$\begin{array}{lcl}
\mathrm{S} & \rightarrow & \mathrm{X} \\
\mathrm{X} & \rightarrow & \mathrm{X\ Y} \\
\mathrm{X} & \rightarrow & \mathrm{Y\ X} \\
\mathrm{X} & \rightarrow & \epsilon \\
\mathrm{Y} & \rightarrow & \mathrm{X} \\
\mathrm{Y} & \rightarrow & \mathrm{a\ Y\ b} \\
\mathrm{Y} & \rightarrow & \mathrm{b\ X\ a}
\end{array}$$

Figure 9: Grammar inferred for the "AB" language

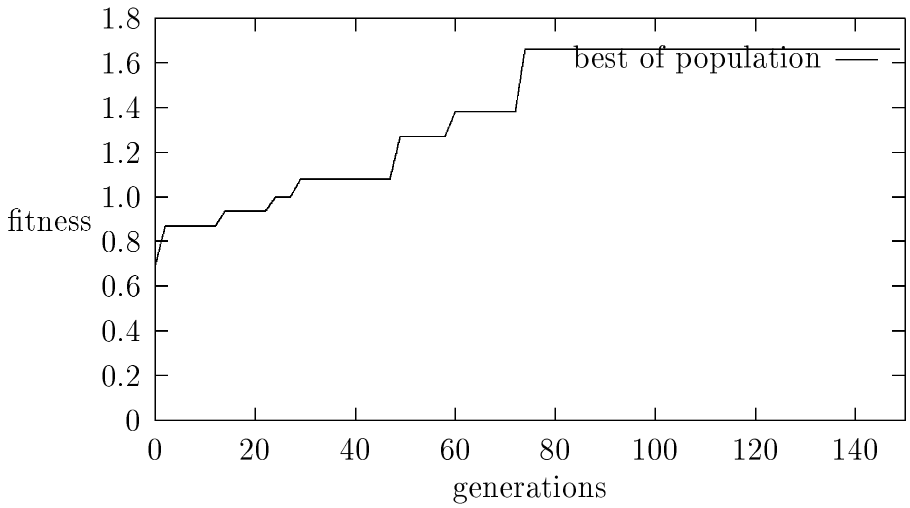

Figure 10: Convergence of the GA for the "AB" language

To illustrate the discontinuous character of the convergence of the GA, a graph of the fitness function during one of the runs is shown in figure 10. Small changes to a grammar can cause it to parse many more of the examples correctly, which explains the sometimes quite substantial jumps in the convergence process.

$$\begin{array}{rcl} \mathrm{S} & \rightarrow & \mathrm{NP\ VP} \\ \mathrm{NP} & \rightarrow & \mathrm{det\ n} \\ \mathrm{NP} & \rightarrow & \mathrm{n} \\ \mathrm{NP} & \rightarrow & \mathrm{NP\ PP} \\ \mathrm{PP} & \rightarrow & \mathrm{prep\ NP} \\ \mathrm{VP} & \rightarrow & \mathrm{v\ NP} \\ \mathrm{VP} & \rightarrow & \mathrm{VP\ PP} \end{array}$$

Figure 11: Grammar for the "Micro-NL" language

6.3 A Set of Regular Languages

Further tests were conducted using a set of six regular languages, used by Tomita (Tomita, 1982) in a number of experiments in inducing finite automata using hill-climbing. The easiest language, 1^*, was omitted from this set since it posed no challenge to our algorithm. The remaining languages are listed as numbers 3–8 in figure 6. Numbers 4, 7, and 8 were also used in (Zhou and Grefenstette, 1986).

For all these languages we used evaluation function F_2 and a set of 100 positive and 100 negative examples. For some languages, grammars evolved that could parse all examples correctly, but generated a number of incorrect strings. This cannot be prevented, since these languages are sparse, i.e., the number of correct strings formed from the terminal alphabet is much smaller than the number of incorrect strings.

All grammars were also tested on a set of positive examples that did not occur in the training set. The grammars scoring 100% on the positive training examples, also analyzed the test set correctly. The other grammars scored within 5% of the training set score.

The results obtained are summarized in figure 12.

6.4 The "Micro-NL" Language

The "Micro-NL" language can be described by the grammar of figure 11. A first experiment with this language, using 250 positive and 250 negative example sentences, a population of 128 individuals, and fitness function F_2 (definition 6), did not result in a correct grammar. Neither enlarging the population nor using more examples could improve the results significantly.

Some grammars evolved that scored a high fitness value by analyzing all examples correctly, but generated many illegal sentences. The cause of this problem is the fact that the set of correct sentences forms a very small part of the total set of sentences that can be generated from the given nonterminals. Therefore, restricting the grammar just by training it on illegal sentences is a very hard job. To overcome this problem, we decided to use evaluation function F_4 (definition 10), that includes information on the predictive and generative capacity of the grammar at hand.

To overcome this problem, we decided to train the GA incrementally. First, we only offered it noun phrases as positive training examples. After a correct grammar

nr.	pop.	rules	syms	bits	gen's	eval's	pos.	neg.	gen.	test
1	48	4	4	28	592	28416	100	100	100	100
2	48	8	5	88	271	13008	100	100	100	100
3	48	4	4	44	17	816	100	100	100	100
4	48	14	8	154	933	44784	85	100	100	81
5	48	14	7	154	1438	69024	80	100	100	75
6	48	14	10	154	2085	100080	100	99	61	100
7	48	8	6	88	1067	51216	100	91	84	100
8	48	8	7	88	825	39600	100	86	87	100
9	128	10	8	140	1984	231763	100	100	55	100

"nr.": number of test language (see figure 6);
"pop.": population size;
"rules": maximum number of rules;
"syms": number of symbols (terminals and nonterminals);
"bits": number of bits in chromosomes;
"gen's": avg. nr. of generations until first occurrence of best solution;
"eval's": avg. nr. of chromosome evaluations until first occurrence of best solution;
"pos.": percentage of positive examples analyzed correctly;
"neg.": percentage of negative examples analyzed correctly;
"gen.": percentage correct of generated sentences;
"test": percentage of test set analyzed correctly.

Figure 12: Results

had been inferred, the training set was augmented with verb phrases. This approach resulted in the inference of grammars that could analyze all positive and negative examples correctly, and generated only a modest number of incorrect sentences. The results are shown in figure 12.

7 Conclusions and Plans

In this paper, genetic algorithms have been shown to be a useful tool for the induction of context-free grammars from positive and negative examples of a language. Grammars for the language of correct brackets expressions, the language of equal numbers of a's and b's, a set of regular languages, and a micro-NL language have been inferred more or less correctly. Further experimentation will have to show whether this technique is applicable to more complex languages.

We are planning to investigate several extensions of this work. The evaluation functions we used weighed different aspects of grammars and condensed these into a single scalar. Instead of using such a scalar fitness, we could employ a multiobjective algorithm such as Schaffer's Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985), that uses multidimensional fitness vectors.

We can further enhance our fitness function by regarding the "educational value" of the example sentences. If many of the grammars in the population can judge an example correctly, this educational value is quite low. On the other hand, difficult sentences that are often classified incorrectly should be valued higher. To include these educational values in the fitness function, we could assign a weight factor to each sentence, which is proportional to the number of grammars of the population that do not analyze this sentence correctly. This has, however, the disadvantage that we cannot work with a "moving target" approach in which new sets of examples are generated in every generation.

Another possibly useful approach, introduced by Hillis (Hillis, 1992), is to use the concept of co-evolution. The example sentences form a population of parasites that compete with the population of grammars. The fitness of a sentence is based on the difficulty with which the grammars can analyze this sentence, i.e., the more difficult a sentence is, the higher its fitness will be.

The reproduction of correct sentences can be implemented using De Weger's tree crossover (TX) operator (de Weger, 1990), or recombination operators analogous to those of Koza's Genetic Programming paradigm (Koza, 1992). In this paradigm, Lisp expressions are represented as parse trees, and crossover is implemented by taking suitable subtrees and exchanging them.

Reproducing incorrect sentences is even simpler, since there is no tree structure to be preserved. Therefore, a straightforward recombination of parts of incorrect examples (which is likely to result in new incorrect sentences), combined with a test whether the offspring is incorrect, will suffice.

As Wyard already pointed out (Wyard, 1992), a bucket-brigade algorithm (Holland and Reitman, 1978), in which the population consists of rules instead of complete grammars, might prove to be useful. In such an algorithm, a population member's fitness is determined by scoring its ability to correctly analyze the examples in conjunction with the other rules of the population. This approach has been employed successfully in the inference of classifier systems (Goldberg, 1989). A possible advantage of this approach is that in a population of rules we only have to evaluate the merit of different grammar rules once, as opposed to a population of grammars, in which a single rule might appear in many different grammars. This could alleviate the computational burden of the algorithm.

References

D. Angluin and C.H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, 1983.

T. Bäck. Optimal mutation rates in genetic search. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA'93*, pages 2–9, San Mateo, CA, 1993. Morgan Kaufmann.

J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference*, pages 14–21. LEA, Cambridge, MA, July 1987.

R.C. Berwick and S. Pilato. Learning syntax by automata induction. *Machine Learning*, 2:39–74, 1987.

A.D. Bethke. *Genetic Algorithms as Function Optimizers.* PhD thesis, Computer Science Dept, University of Alberta, 1981.

M. de Weger. Generalized adaptive search: Analysis of codings and extension to parsing. Paper written for the course "Seminarium Theoretische Informatica" at the Dept. of Computer Science, University of Twente, The Netherlands, 1990.

K.A. DeJong. *Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, Dept. of Computer and Communication Sciences, University of Michigan, 1975.

K.S. Fu and T.L. Booth. Grammatical inference: Introduction and survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:343–375, 1986.

D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, MA, 1989.

J.J. Grefenstette. *A User's Guide to GENESIS Version 5.0*, 1990.

W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C.G. Langton, C. Taylor, J. Doyne Farmer, and S. Rasmussen, editors, *Proceedings of the Second International Conference on Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. X, pages 313–324, 1992.

J.H. Holland and J. Reitman. Cognitive systems based on adaptive algorithms. In Waterman and Hayes-Roth, editors, *Pattern-directed Inference Systems.* Academic Press, 1978.

J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

W. Huijsen. Genetic Grammatical Inference: Induction of Pushdown Automata and Context-Free Grammars from Examples using Genetic Algorithms Master's thesis, Dept. of Computer Science, University of Twente, Enschede, The Netherlands, 1993.

J.R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

J.D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms ICGA'85*, pages 93–100. Lawrence Erlbaum, 1985.

S. Sen and J. Janakiraman. Learning to construct pushdown automata for accepting deterministic context-free languages. In G. Biswas, editor, *SPIE Vol. 1707: Applications of Artificial Intelligence X: Knowledge-Based Systems*, pages 207–213. 1992.

M. Tomita. Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pages 105–108, Ann Arbor, MI, 1982.

K. VanLehn and W. Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2:39–74, 1987.

P. Wyard. Context free grammar induction using genetic algorithms. In R.Belew and L.B.Booker, editors, *Proceedings of the Fourth Conference on Genetic Algorithms ICGA'92*. Morgan Kaufmann, 1992.

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