Genetic Grammatical Inference *

Willem-Olaf Huijsen
Research Institute for Language and Speech (O.T.S.)
Utrecht University
Trans 10, room 2.05
NL-3512 JK Utrecht
Tel. +31-30-536006
e-mail: Willem-Olaf.Huijsen@let.ruu.nl

Abstract

This research is concerned with the genetic grammatical inference of formal languages: the induction of pushdown automata and context-free grammars for context-free languages from finite sets of example sentences from these languages using genetic algorithms. Setup of experiments and analyses of their results are presented. The induction of both deterministic and nondeterministic pushdown automata was successful; the induction of context-free grammars proved more difficult. A number of propositions are made for extensions and improvements.

1 Introduction

The goal of this research is to build a system that can construct a recognizer or a context-free grammar from a finite number of example sentences of a context-free language. A recognizer is a formal structure which can determine whether a specific sentence belongs to a given language. A grammar of a language is a scheme for specifying the sentences allowed in the language, indicating the syntactic rules for combining words into well-formed phrases and clauses. Grammars are more versatile than recognizers since grammars can not only be used to parse but also to generate sentences of the language. Construction of the recognizer or grammar for the context-free language is to be done using genetic algorithms.

Section 2 introduces grammatical inference; section 3 discusses the genetic algorithm paradigm; section 4 provides the reader with the insight in how genetic algorithms are to be applied to the problem of grammatical induction; section 5 describes the experiments conducted; finally, section 6 presents the conclusions.

*This research was conducted during the author's master's thesis, *Genetic Grammatical Inference: Induction of Pushdown Automata and Context-Free Grammars from Examples using Genetic Algorithms*, at the Department of Computer Science at Twente University, Enschede, The Netherlands, summer 1993 (supervisors prof.dr.ir. A. Nijholt, dr. M. Poel, dr.ir. P.R.J. Asveld)

2 Grammatical Inference

Grammatical inference (Gold 1967; Pinker 1979; Angluin and Smith 1983) is the problem of learning a grammar for a (formal) language on the basis of a set of sample sentences. Although a great deal of work has been done, a complete and applicable theory is still a distant goal.

A fundamental theorem of mathematical linguistics states that there is an infinite number of grammars that can generate any finite set of strings. Therefore, it is impossible for *any* learner to observe a finite sample of sentences of a language and always produce a correct grammar for the intended language. Thus, only guesses can be made. The quality of these guesses depends on the quality of the examples. Good examples must characterize the main features of the concept they represent.

Showing the inductor only a set of positive examples $S^+ \subseteq L$ (i.e. strings in the language) is generally not sufficient for inducing a language $L \subseteq \Sigma^*$. If one guesses too large a language, the positive examples will never tell you that you are wrong. Thus, both positive examples S^+ *and* negative examples S^- (i.e. strings *not* in the language) are required to provide the contrast needed to outline L in Σ^*. Formally, Gold showed that if both S^+ and S^- are available to the inductor, the class of primitive recursive languages (which by assumption include the natural languages) are learnable. If only positive examples S^+ are available, *no* language class other than the finite cardinality languages is learnable. See Gold (1967).

3 Genetic Algorithms

Genetic algorithms (Holland 1975; De Jong 1985; Davis 1987; Goldberg 1989; Michalewicz 1992) have been investigated for almost twenty years now, with a marked increase in interest within the last few years.

3.1 What are Genetic Algorithms ?

Genetic algorithms are optimization and search algorithms based on the mechanisms of natural selection and genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search strategy with some of the innovative flair of human search. Every generation in this evolution, new artificial creatures (the strings) are created using bits and pieces of the old, sometimes trying out a completely new part.

Unlike many other search algorithms, GA's search from a population of candidate solutions, simultaneously climbing many hills in parallel, thus reducing the probability of getting stuck in a local optimum. It is believed that GA's possess desirable properties for solving problems with large search spaces, high order, multimodality, discontinuity and noise disturbance.

Outline of a Simple Genetic Algorithm

The basic construction is to consider a population of individuals that each represent a potential solution to the given problem. The relative success of each individual on this problem is considered its *fitness*, and used to selectively reproduce fitter individuals to produce similar but not identical offspring for the next generation. By iterating this process, the population efficiently samples the space of potential individuals and eventually converges on the most fit.
More specifically, consider a population of N individuals x_i, each represented by a *chromosomal* string of L *allele* values. An initial population is constructed at random; call this *generation* g_0. Each individual is evaluated by some objective function (the *fitness function*) that returns the fitnesses $\mu(x_i) \,\epsilon\, \mathcal{R}$ of each individual in g_0. The evolutionary algorithm then performs two operations. First, its *selection* algorithm uses the population's N fitness measures to determine how many offspring each member of g_0 contributes to g_1. More fit individuals are more likely to have offspring than less fit individuals. Second, a set of *genetic operators* are applied to these offspring to make their genetic information different from their parents. The resulting population is now g_1; these individuals are again evaluated, and the cycle is repeated. The iteration is terminated by some measure suggesting that the population has converged.

3.2 Implementing Genetic Operators

This section sketches a model for a simple genetic algorithm. Subsection 3.2.1 gives a formal definition for the datastructure; subsections 3.2.2 and 3.2.3 define the two principal genetic operators: mutation and crossover. Subsection 3.2.4 discusses the fitness function, which is used in the definition of reproduction in subsection 3.2.5.

3.2.1 The Datastructure

The fundamental datastructure of genetic algorithms is the chromosome: an array of 'genes' that may assume any of several allelic forms. The GA population is defined as a set of n individuals: $g = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$ each individual $\bar{x}$ being a vector of length l, representing a chromosome: $\bar{x} = [\ x_1\ x_2\ \ldots\ x_l\ ]$ $\quad (x_i\ \epsilon\ Alphabet)$, where $Alphabet = \{0, 1\}$, or any arbitrary integer range.

3.2.2 Mutation

Mutation is the sudden inheritable change of a gene from one allelic form to another. The probability of mutation of an element of the chromosome is termed *mutation probability*. Mutation probability p_m is usually very low (e.g. 0.001 – 0.05).
Binary gene mutation is straightforward bit flipping (see figure 3.1): $mutation(gene)$ $= 1 - gene$ if $Bernoulli(p_m)$, or $gene$, otherwise, where $gene\ \epsilon\ \{0, 1\}$ and $Bernoulli(p)$ defined as the result of a Bernoulli experiment with probability p.

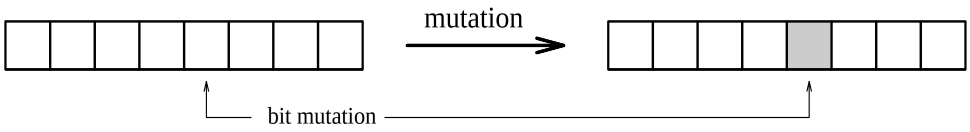

Fig.3.1. Binary Point Mutation .

3.2.3 Crossover

Crossover is the exchange of genetic material between two chromosomes. *Crossover probability* is the probability of two individuals exchanging genetic information during reproduction. Usual values for crossover probability p_c are 0.3 – 0.8. When two chromosomes $\bar{x}$ and $\bar{y}$ participate in crossover, a break occurs somewhere along the chromosomes, and the fragments are recombined:

$$\begin{aligned} crossover(\bar{x},\bar{y}) &= ([x_1 \ldots x_b\ y_{b+1} \ldots y_l],\ [y_1 \ldots y_b\ x_{b+1} \ldots x_l]) &&, \textit{if Bernoulli}(p_c) \\ &= (\bar{x},\bar{y}) &&, \textit{otherwise} \end{aligned}$$

The crossover mechanism defined is the so-called *one-point crossover*. Other types of crossover are *two-point crossover* and *uniform crossover* (Huijsen 1993b, p.30-31).

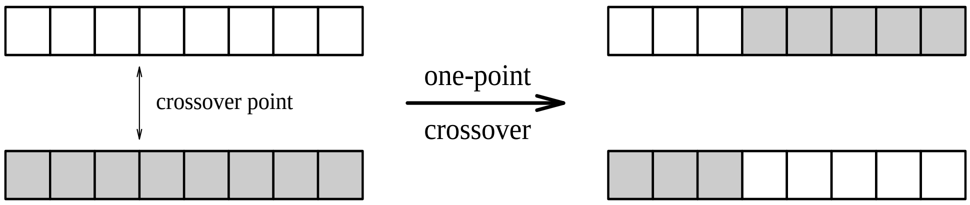

Fig.3.2. One-Point Crossover .

3.2.4 Fitness Function

The fitness function $fitness$ is a function assigning a measure of success to an individual $\bar{x}$. To do this, the chromosome has to be decoded into the problem's parameter set values that can be evaluated to measure its performance on the given problem. Since this is the only feedback the genetic algorithm gets, the fitness function has to be designed with great care.

3.2.5 Reproduction

Reproduction recombines the genetic information of two individuals to produce offspring using the genetic operators mutation and crossover. More fit individuals have a greater probability of being selected for reproduction than less fit individuals. Each individual $\bar{x}_i$ gets selected with a probability p_i proportional to its fitness:

$$p_i = \frac{fitness(\bar{x}_i)}{\sum_{j=1}^{n} fitness(\bar{x}_j)}$$

Subsequently, the two chosen individuals $\bar{x}$ and $\bar{y}$ are subjected to the genetic operators mutation and crossover: $\mathit{offspring}(\bar{x},\bar{y}) = \mathit{crossover}(\mathit{mutation}(\bar{x}),\mathit{mutation}(\bar{y}))$. The generated offspring are then placed in the new population. This process is repeated until the new population again contains n individuals.

4 Genetic Algorithms & Grammatical Inference

The problem to be attacked is the induction of recognizers for context-free languages from finite sets of examples from these languages. A recognizer is a structure which can determine whether a specific sentence belongs to a given language. Construction of the recognizers is to be done using genetic algorithms.
Learning a context-free language from examples has been proven to be *NP*-complete by Gold (1967). Genetic algorithms impressively outperform other techniques on several *NP*-complete problems such as dynamic programming, hill-climbing, and random search (Grefenstette et al. 1985). This encouraging performance of genetic algorithms stimulated the present research.

4.1 Formulation of Strategy

The grammatical induction of a recognizer can be formulated as a search in the space of all possible recognizers. Genetic algorithms being general optimization algorithms, they may readily be applied to this problem. A fitness function has to be defined over the search space, which assigns a fitness value to the chromosomes, indicating their success at recognizing the intended language. The fitness value is used by the genetic algorithm in its search for the optimum.

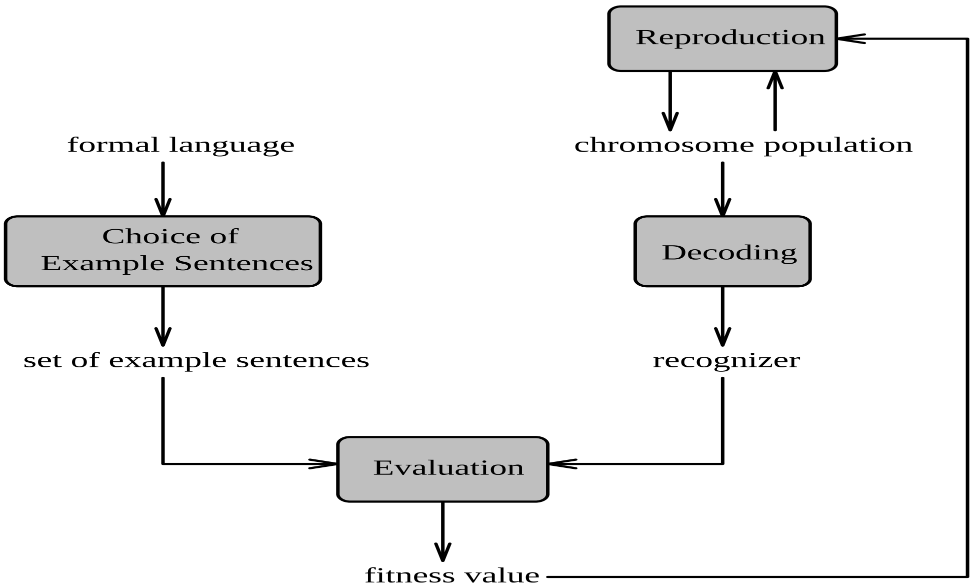

Fig.4.3. Structure of the Genetic Inductor .

In the experiments the *genetic inductor* is tested using a number of *formal languages* as test problems. The formal languages are used to establish a *set of example sentences.* The example sentences are used to determine the performance of the recognizers inferred by the inductor. The genetic inductor is based on a regular GA system: a *population of chromosomes* is maintained on which the recognizers are encoded. To obtain the fitness value for a chromosome, the fitness function first decodes the chromosome to the specific form of the recognizer. The *recognizer* now is evaluated for each sentence in the set of example sentences. A combination of these results finally yields the *fitness value* associated with the chromosome. The fitness value of a chromosome is used in the reproduction process of the genetic algorithm. The *reproduction process* constructs the next generation of chromosomes. The reproduction/evaluation loop continues until either an optimum has been found (i.e. until a recognizer has been evolved that correctly classifies all example sentences), or a predetermined maximum generation has been reached.

4.2 Test Problems

This section defines the test problems used in the experiments of section 5. The following test problems are defined: (i) *PBP*. Parenthesis balancing, deterministic; (ii) *AB*. Equal number a's/b's, deterministic; (iii) WW^R. Palindromes, nondeterministic. For every test problem language L, all sentences from the sublanguage L_6 (sentences from L with maximum length 6) are used as the positive example sentences. The complement (i.e. $\Sigma_6^* \backslash L_6$) constitutes the negative examples.

4.2.1 The Parenthesis Balancing Problem

The *parenthesis balancing problem* (PBP) is concerned with matching parentheses. A context-free grammar for this language may be written as: { S $\rightarrow$ E, E $\rightarrow$ λ, E $\rightarrow$ (E), E $\rightarrow$ E E }.
A pushdown automaton for language PBP may be defined as:

$$\begin{array}{ll} Q = \{q_0\} & \delta(q_0, (, \$) = \{[q_0, (]\} \\ \Sigma = \{(,)\} & \delta(q_0, (, () = \{[q_0, ((]\} \\ \Gamma = \{(,)\} & \delta(q_0,), () = \{[q_0, \lambda]\} \end{array}$$

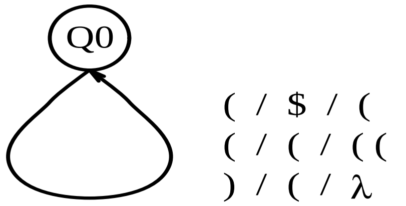

Fig.4.4. A PBP Pushdown Automaton .

In successful computations the machine accumulates the opening parentheses by pushing them onto the stack until the first closing parenthesis is inputted. Each closing parenthesis cancels an opening parenthesis by popping it from the stack.

4.2.2 The AB Problem

The AB language is the language of all strings in $\{a, b\}^*$ having an equal number of a's and b's. A context-free grammar for this language may be written as: { S $\rightarrow$ E, E $\rightarrow$ λ, E $\rightarrow$ a E b, E $\rightarrow$ b E a, E $\rightarrow$ E E }.
A pushdown automaton for language AB may be defined as:

$$\begin{array}{lll} Q = \{q_0\} & \delta(q_0, a, \$) = \{[q_0, a]\} & \delta(q_0, b, \$) = \{[q_0, b]\} \\ \Sigma = \{a, b\} & \delta(q_0, a, a) = \{[q_0, a\,a]\} & \delta(q_0, b, a) = \{[q_0, \lambda]\} \\ \Gamma = \{a, b\} & \delta(q_0, a, b) = \{[q_0, \lambda]\} & \delta(q_0, b, b) = \{[q_0, b\,b]\} \end{array}$$

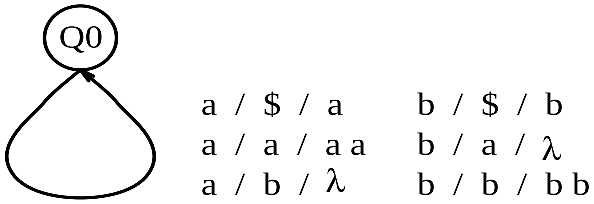

Fig.4.5. An AB Pushdown Automaton .

The pushdown automaton has to keep record of the number of a's and b's it has seen on the input. The administration is done in a way very much like the previous problem: by counting symbols. At any point during processing, one of three cases applies (na is the number of a's that have been inputted, and nb the number of b's that have been inputted): (i) $na = nb$ In this case, the stack is empty. (ii) $na > nb$ The stack contains $na - nb$ a's. (iii) $na < nb$ The stack contains $nb - na$ b's. This invariant is maintained during execution of the PDA.

4.2.3 The ww^R Problem

The ww^R language consists of the even-length palindromes over $\{a, b\}$. Formally, it is defined as $\{ww^R | w \in \{a, b\}^*\}$. A context-free grammar for this language may be written as: { S $\rightarrow$ E, E $\rightarrow$ λ, E $\rightarrow$ a E a, E $\rightarrow$ b E b }.
A pushdown automaton for language ww^R may be defined as:

$$\begin{array}{lll} Q = \{q_0, q_1\} & \delta(q_0, a, \$) = \{[q_0, a]\} & \delta(q_0, b, a) = \{[q_0, b\,a]\} \\ \Sigma = \{a, b\} & \delta(q_0, a, a) = \{[q_0, a\,a], [q_1, \lambda]\} & \delta(q_0, b, b) = \{[q_0, b\,b], [q_1, \lambda]\} \\ \Gamma = \{a, b\} & \delta(q_0, a, b) = \{[q_0, b\,a]\} & \delta(q_1, a, a) = \{[q_1, \lambda]\} \\ & \delta(q_0, b, \$) = \{[q_0, b]\} & \delta(q_1, b, b) = \{[q_1, \lambda]\} \end{array}$$

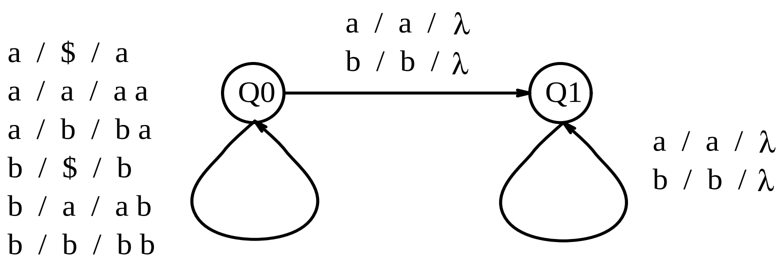

Fig.4.6. A WWR Pushdown Automaton .

A successful computation remains in state q_0 while processing the string w, and enters state q_1 upon reading the first symbol in w^R. The strings in ww^R do not contain a middle marker to trigger the state change. Therefore, the machine has to 'guess' when the middle of the string has been reached, introducing nondeterminism.

4.3 Related Research

Related research efforts are: Zhou and Grefenstette (1986), Wyard (1991), Sen and Janakiraman (1992), Kohler (1993), and Lankhorst (1994). Here I briefly discuss the findings of Sen and Janakiraman (1992), because their results are most readily comparable to mine. For further discussion, see Huijsen (1993b).
Sen and Janakiraman (1992) apply genetic algorithms to learn DPDA's that accept context-free languages, given a number of positive and negative examples of the language. They use only DPDA's, transitions always change the stack (either pushing or popping), and transitions always process one input symbol (i.e. there are no lambda-rules). The experiments use languages a^nb^n and PBP . Their experiments show that GA's can effectively evolve control rules for a DPDA to accept instances of a deterministic context-free language. However, their conclusions are based on single runs, as opposed to an average over a number of runs in my experiments. An improvement compared to the experiments of Zhou and Grefenstette is the automatic selection of example sentences: all sentences up to a specified length are used.

5 Experiments

The experiments have been grouped according to the recognizer types. Three types of recognizers are considered: *deterministic pushdown automata* (subsection 5.2); *non-deterministic pushdown automata* (subsection 5.3); *context-free grammars*[1] (subsection 5.4).

[1]Strictly, context-free grammars are not recognizers. In combination with a parser they *can* be used as a recognizer. For the sake of simplicity all three are called *recognizers*.

5.1 Mapping Chromosomes into Recognizers

The experiments implement chromosomes as integer arrays. In decoding the chromosome into a recognizer, the chromosome is read from one end to the other. Specific integer values are taken to be start markers, signalling the beginning of a segment that encodes one transition (or grammar rule, for CFG's). The subsequent array values are interpreted as the details of the specification of the transition (grammar rule). After interpreting the segment, decoding scans the chromosome for the next start marker. In this manner decoding yields a number of transitions (grammar rules), which together constitute the PDA (CFG). The exact workings are more elaborately described in section 6.1 of Huijsen (1993b).

5.2 Inferring DPDA's

These experiments concentrate on learning the transitions rules in δ of a DPDA. To facilitate implementation, I use an alternative definition for pushdown automata.
A DPDA M is modeled as a septuple $M = (Q, \Sigma, \Gamma, q_0, q_r, Z_0, \delta)$:

- Q is a finite, nonempty set of *states*. $S = |Q|$.
- Σ is a finite, nonempty set of *input symbols*.
- Γ is a finite, nonempty set of *stack symbols*.
- q_0 is the *start state*.
- q_r is the only *reject state*. Reject state q_r is special: changing state to q_r terminates evaluation immediately. Q^- is Q minus the reject state q_r.
- $Z_0 \,\epsilon\, \Gamma$ is the *initial stack symbol*. Γ^- is Γ minus the initial stack symbol.
- $\delta : Q^- \times \Sigma \times \Gamma \rightarrow Q \times \Omega \times \Gamma^-$ where $\Omega = \{\text{nop,pop,push}\}$.

Acceptance is by legal state and empty stack. Transitions of the form $\delta(q, a, Z) = (p, \omega, Y)$ are represented by a directed arc from state q to state p labeled '$a/Z/\omega$' for $\omega = \text{nop}$ or $\omega = \text{pop}$, and '$a/Z/$push Y' for $\omega = \text{push}$.

5.2.1 DPDA Results for Parenthesis Balancing

Two experiments were conducted; one using the cumulative fitness function, and one using the productive fitness function[2]. Parameter values are: population size 100, chromosome length 200, elite percentage[3] 5, maximum generation 2000, 20 runs.
Finding a global optimum took the genetic inductor 14.6 generations on the average when using cumulative fitness, and 4.2 generations when using productive fitness.
Three actual DPDA's found are shown in figure 5.7:

[2]Cumulative fitness simply computes the number of correctly classified samples; productive fitness computes the product of the percentage correctly classified postive examples and the percentage of correctly classified negative examples.

[3]A special *elite mechanism* has been added to the standard genetic algorithm: a small percentage of individuals from the old generation is retained in the next. This way the fittest solutions are maintained.

(A) The first inferred DPDA is straightforward, equivalent to the PDA proposed in subsection 4.2.1.

(B) The second only *seems* to be different. Nevertheless, it is equivalent to the first type: only the stack symbols '(' and ')' have exchanged roles.

(C) The third is a variation on the first: it differs from (A) in that it rejects strings that have a closing parenthesis that doesn't match an opening parenthesis – by popping the empty stack instead of going to the reject state.

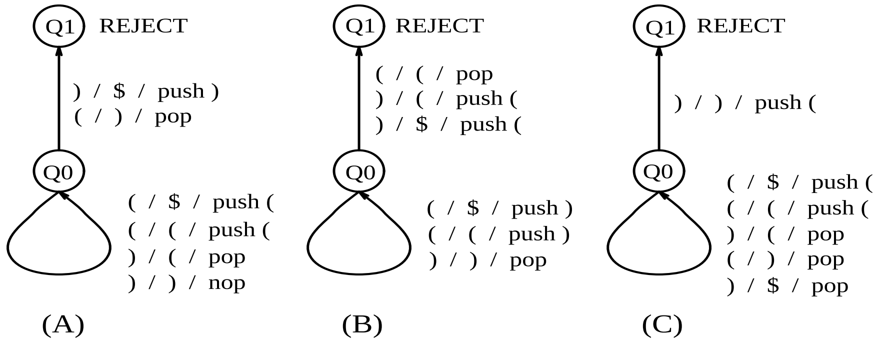

Fig.5.7. Inferred Deterministic Pushdown Automata for PBP .

5.2.2 DPDA Results for the AB Problem

Parameter values are: population size 200, chromosome length 200, elite percentage 5, maximum generation 2000, number of runs 20. Finding a global optimum takes the genetic inductor 13.7 generations on the average when using productive fitness.
As in the case of the PBP problem, the DPDA's found consist of two symmetrical solutions that are functionally equivalent to the 'prototype' (see figure 5.8).

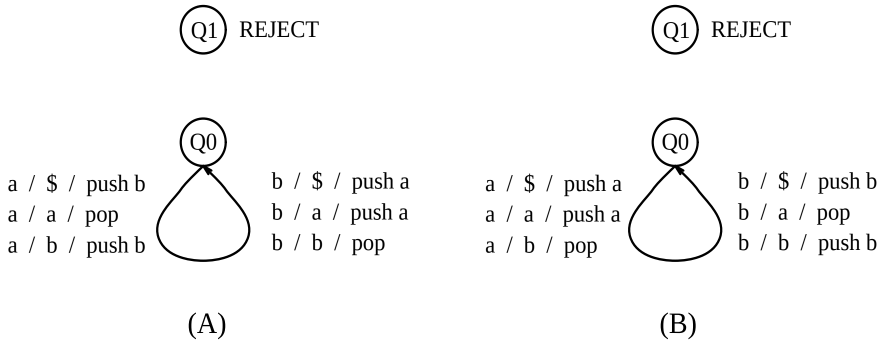

Fig.5.8. Inferred Deterministic Pushdown Automata for AB .

5.2.3 DPDA Conclusions

- The genetic inductor has been shown to effectively evolve DPDA's for the test problems selected.
- The results obtained support the theorem that there are an infinite number of languages that agree on any finite set of sample strings.
- The comparison of the cumulative and productive fitness functions shows that the productive fitness function significantly increases convergence speeds: the speed-up factor is 3.5 for PBP .
- The performance analysis shows that the GI is highly efficient in the experiments: only a very small part of the search space has to be sampled. The productive fitness experiment for PBP searched 0.16% of the total search space to arrive at a global optimum; 1.04% in the case of AB.

5.3 Inferring NPDA's

These experiments concentrate on learning the transitions rules in δ of an NPDA. The model used in the experiments on nondeterministic pushdown automata is an extension of the DPDA model: the transition function now is defined as $\delta : Q^- \times \Sigma \times \Gamma \rightarrow \mathcal{P}\ (Q^- \times \Omega \times \Gamma^-)$.

5.3.1 NPDA Results for Parenthesis Balancing

Two experiments were conducted: one using the cumulative fitness function, and one using the productive fitness function. Parameter values are: population size 100, chromosome length 100, elite percentage 5, maximum generation 2000, number of runs 20.
Finding a global optimum takes the genetic inductor 8.9 generations on the average when using cumulative fitness, and 4.2 generations when using productive fitness.
The actual NPDA's found can be classified into two types (see figure 5.9):

(A) The first type of inferred NPDA is again equivalent to the PDA proposed in subsection 4.2.1.

(B) The second type is a symmetrical equivalent of the first: only the stack symbols '(' and ')' have exchanged roles.

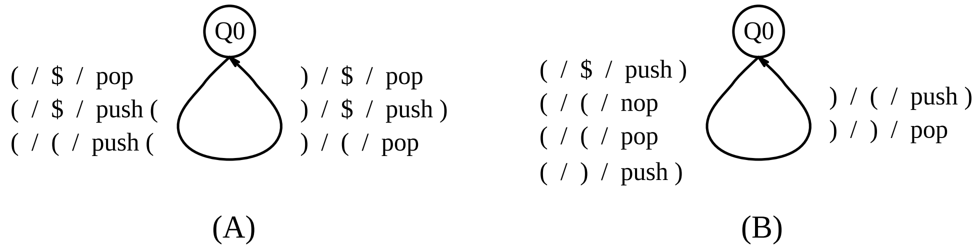

Fig.5.9. Inferred Nondeterministic Pushdown Automata for PBP .

5.3.2 NPDA Results for the ww^R Problem

Three experiments were run, buth using productive fitness. Parameter values were: population size 100 and 200, chromosome length 200, elite percentage 10, maximum generation 1000 and 2000, number of runs 10. In a total of 21 completed runs of the inductor, only two optimal NPDA's were found (see figure 5.10):

(A) The third run of the first experiment found an optimal NPDA after 635 generations. The NPDA correctly classifies all sample sentences, but significantly differs from the 'prototype' of section 4.2, most notably so in the transition $\delta(q1, b, b) = \{[q1, \lambda, b\,b]\}$.

(B) The seventh run of the third experiment found an optimal NPDA after 1156 generations. This automaton is functionally equivalent to the prototype. The additional transition 'a / \$ / pop' from state $q1$ leads to a rejection, which is equivalent to the action the prototype would take.

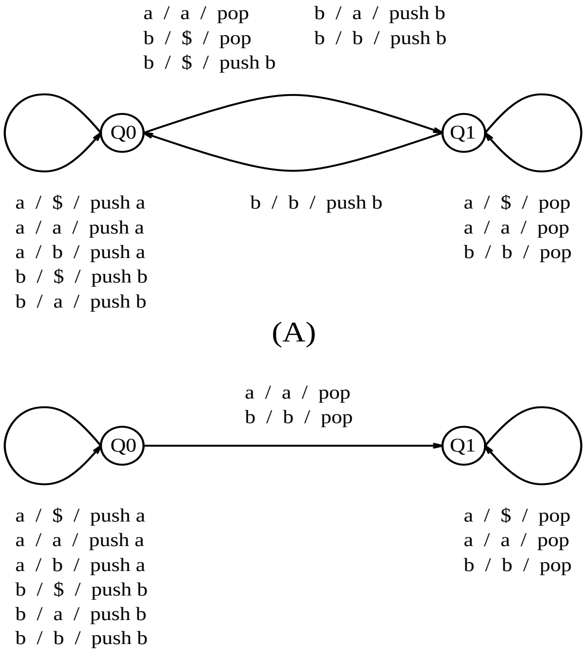

Fig.5.10. Inferred Nondeterministic Pushdown Automata for WWR .

5.3.3 NPDA Conclusions

- It has been shown that the genetic inductor can effectively evolve nondeterministic pushdown automata for the test problems selected as well.
- The comparison of the cumulative and productive fitness functions again shows that the productive fitness function significantly increases convergence speeds. The speed-up factor is 2.1 for PBP .

- The performance analysis again shows that the GI is highly efficient in the experiments: The productive fitness experiment for PBP searched only 0.0025% of the total search space to arrive at a global optimum.
- Comparing the DPDA and NPDA experiments for the PBP experiments, I conclude that the NPDA is faster than the DPDA:

experiment	deterministic	nondeterministic
cumulative PBP	14.6	8.9
productive PBP	4.2	4.2

Fig.5.11. DPDA/NPDA Comparison: No. of Generations.

This can be explained by the fact that a DPDA has to meet more stringent demands: an NPDA already accepts a sentence if *any* of all possible computations succeeds.

5.4 Inferring Context-Free Grammars

The aim is to evolve context-free grammar rules. The context-free grammar rules are required to be in Chomsky Normal Form.

5.4.1 CFG Results

Although a lot of time has been spent on the implementation and improvement of the direct inference of context-free grammars, the experiments did not produce perfect grammars. The most extensive experiment is presented here. Parameter values are: population size 100, chromosome length 200, elite percentage 10, maximum generation 300, 10 runs. None of the experiments converged within 300 generations.

5.4.2 CFG Conclusions

- The direct inference of context-free grammars has not been successful. Possible explanations are: (i) The search space is much larger than with PDA's; (ii) The fitness function used is not adequate.
- To get the direct inference to work, I see two approaches: (i) Run larger (and thus more computationally expensive) experiments. (ii) Improve on the fitness function. This is suggested by the belief that the fitness function is not adequate.

6 Conclusions

- First and foremost, we conclude that the genetic inductor implemented as part of the thesis work has been proven to effectively evolve both DPDA's and NPDA's from a finite number of example sentences.

- Compared to the related research of section 4.3, the present work: (i) uses a more general – and therefore more powerful – PDA model; (ii) is the first to infer nondeterministic pushdown automata; (iii) tests the inductor more thoroughly (for more testproblems); (iv) provides data on a sufficient numbers of runs for each experiment to allow for conclusions on efficiency.
- Quantitatively demonstrates the superiority of productive fitness over cumulative fitness.

Further research could include (i) better tuning of the parameters to the genetic algorithm; (ii) exploration of the use of other genetic operators (e.g. two-point crossover); (iii) inclusion of λ-transitions in the PDA-model; (iv) other selection strategies for sample sentences; (v) more sophisticated fitness functions; (vi) induction of natural language fragments; (vii) induction of context-sensitive languages.

References

• Angluin and Smith 1983; Dana Angluin, Carl H. Smith, 'Inductive Inference: Theory and Models', *Computing Surveys 15*, 3 (Sept.), pp. 237-269.
• Davis 1987; Lawrence Davis (ed.), *Genetic Algorithms and Simulated Annealing*, London: Pitman.
• De Jong 1985; Kenneth A. De Jong, 'Genetic Algorithms: A 10 Year Perspective', In: John J. Grefenstette (ed.), *Proc. of an Int. Conf. on Genetic Algorithms and Their Applications*, Erlbaum.
• Gold 1967; E. Mark Gold, 'Language Identification in the Limit', *Information and Control 10*, pp. 447-474.
• Goldberg 1989; David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, Massachusetts: Addison-Wesley.
• Grefenstette et al. 1985; J.J. Grefenstette, R. Gopal, B.J. Rosmaita, V. Gucht, 'Genetic Algorithms for the Travelling Salesman Problem', In: *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, p. 160 ff..
• Holland 1975; John H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and art. intelligence*, Ann Arbor: Univ. of Michigan Press.
• Huijsen 1993a; W. Huijsen, *Exercises in Genetic Algorithms*, internal report, Dept. of Computer Science, University of Twente, Enschede, The Netherlands.
• Huijsen 1993b; 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.
• Kohler 1993; John D. Kohler, *Genetic Algorithm Evolving Finite Automata Given Sample Strings*, personal communication. Kohler can be contacted at 'drsa@netcom.com', or written to: John D. Kohler, President of Artificial Systems, 1043 Electric St., Gardena, CA 90248.
• Lankhorst 1994; Marc M. Lankhorst, *Breeding Grammars: Grammatical Inference with a Genetic Algorithm*, Comp. Science Report CS-R9401, C.S. Department, Univ. of Groningen, The Netherlands.
• Michalewicz 1992; Zbigniew Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Berlin: Springer Verlag.
• Pinker 1979; Steven Pinker, 'Formal Models of Language Learning', *Cognition 7*, pp. 217-283.
• Sen and Janakiraman 1992; Sandip Sen, Janani Janakiraman, 'Learning to Construct Pushdown Automata for Accepting Deterministic Context-Free Languages', In: Gautam Biswas (ed.), *SPIE Vol. 1707: Applications of Artificial Intelligence X: Knowledge-Based Systems*, pp. 207-213.
• Wyard 1991; Peter Wyard, *Context-Free Grammar Induction using Genetic Algorithms*, In: R. Belew, L.B. Booker (eds.), *Proc. of the 4th Conf. on Genetic Algorithms ICGA'92*, Morgan Kaufmann.
• Zhou and Grefenstette 1986; Ha-Yong Zhou, John J. Grefenstette, 'Induction of Finite Automata by Genetic Algorithms', In: *Proc. of the 1986 IEEE Int. Conf. on Systems, Man, and Cybernetics.*