Genetic Programming with Incremental Learning for Grammatical Inference

Ernesto Rodrigues and Heitor Silvério Lopes[*]
Graduate Program in Electrical Engineering and Computer Science
Federal University of Technology – Paraná
Av. 7 de setembro, 3165 80230-901, Curitiba, Brazil
ernesto@cpgei.cefetpr.br, hslopes@pesquisador.cnpq.br

Abstract

We present an evolutionary algorithm for the inference of context-free grammars from positive and negative examples. The algorithm is based on genetic programming and uses a local optimization operator that is capable of improving the learning task. Ordinary genetic operators are modified so as to bias the search. The system was evaluated using Tomita's language examples and results were compared with another similar approach. Results show that the proposed approach is promising and more robust than the other one.

1. Introduction

Grammatical Inference (also known as grammar induction) is the problem of learning a grammar for a language from a set of examples. In a broad sense, a learner has access to some sequential or structured data and is asked to return a grammar that should in some way explain such data. The inferred grammar can then be used to classify unseen data or provide some suitable model for this data.

Applications of grammatical inference include such diverse fields as pattern recognition, information retrieval, programming language and bioinformatics, among others [1].

Researchers have developed various algorithms that have in common that the largest class of languages, which can be efficiently learned by provably converging algorithms are regular languages. Therefore, learning context-free grammars is still a real challenge in grammatical inference [1].

In this paper, we present a Genetic Programming-based approach for grammatical inference using a local search mechanism. Genetic Programming (GP) is the automatic generation of computer programs, using a process analogous to biological evolution [2]. This technique exploits the process of natural selection based on a fitness measure to breed a population of candidate solutions that improves over time. GP is a robust method that has been successfully applied to a number of areas such as biotechnology, electrical engineering, image processing, pattern recognition, natural language and many others.

The next section defines the class of context-free grammars we are attempting to learn. Next, we describe the genetic programming approach used in this paper. We then present experiments demonstrating the effectiveness of the algorithm on a suite of grammatical inference benchmark problems. The last section presents conclusions and future work.

2. Context-Free Grammars

A context-free grammar (CFG) is defined by a quadruple $G = (N, \Sigma, P, S)$, where N is an alphabet of nonterminal symbols, Σ is an alphabet of terminal symbols such that $N \cap \Sigma = \varnothing$, P is a finite set of production rules of the form $A \rightarrow \alpha$ for $A \in N$ and $\alpha \in (N \cup \Sigma)^*$, and S is a special nonterminal called the start symbol. The language generated by a CFG G is denoted L(G) [3].

The context-free language L(G) produced from grammar G is the set of all strings consisting only of terminal symbols that can be derived from the start symbol S by sequential application of production rules, that is, $L(G) = \{ x \mid S \Rightarrow^* x, x \in \Sigma^* \}$ [3].

A context-free grammar G is in Chomsky Normal Form (CNF) if, and only if, all production rules are of the form $A \rightarrow BC$ or $A \rightarrow \alpha$ for $A, B, C \in N$ and $\alpha \in \Sigma$.

[*] This work was partially supported by a research grant to H.S.Lopes from the Brazilian National Research Council.

To determine whether a string can be generated by a given context-free grammar in CNF, the Cocke-Younger-Kasami (CYK) algorithm can be used. It also can devise how the string can be generated. This is known as parsing the string. The CYK works as follow with V_{rs} representing the triangular matrix cells [4]:

```
for r = 1 to n do
  $V_{r1}$ = { A | A → $a_r$ ∈ P }
for s = 2 to n do
  for r = 1 to n-s+1 do
    $V_{rs}$ = ∅
    for k = 1 to s - 1 do
      $V_{rs}$ = $V_{rs}$ ∪ {A | A → BC ∈ P,
                     B ∈ $V_{rk}$ and
                     C ∈ $V_{(r+k)(s-k)}$ }
if S ∈ $V_{1,n}$ then w ∈ L(G)
           else w ∉ L(G)
```

Figure 1 shows an example of a triangular matrix obtained from the parse of string "aba" from the set of rules P = {S → AA; S → AS; S → b; A → SA; A → AS; A → a}.

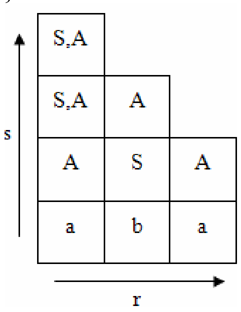

Figure 1: An example of a triangular matrix.

3. Genetic Programming (GP)

Genetic Programming is an induction method used to search over a huge state space consisting of structured representations that are trees. Traditionally, each tree represents a possible solution written in some arbitrary language. The GP algorithm can be summarized as follows [2]:

- Randomly create a population of individuals;
- Perform the following three steps until the termination criterion is satisfied:
  - Evaluate the fitness of each program;
  - Select programs from the current population according to fitness;
  - Apply genetic operators, such as: reproduction, crossover and mutation.

The evaluation of a solution is accomplished by using a set of training examples known as fitness cases which, in turn, is composed by input and output data for each instance. Usually, the fitness is a measure of the deviation between the correct output for each input and the computed value given by GP.

There are two main selection strategies used in GP: fitness proportionate and tournament. In the fitness proportionate selection, programs are selected randomly with probability proportional to its fitness. In the tournament selection, a fixed number of programs are taken randomly from the population and the one with the best fitness within this group is chosen.

Reproduction is a genetic operator that simply copies a program to the next generation. Crossover, another operator, combines parts of two individuals to create two new ones.

Each run of that loop crates a new generation of computer programs that substitutes the previous one. The evolution is stopped when a satisfactory solution is achieved or a predefined maximum number of generations is reached.

4. Grammatical Inference with GP

It is possible to represent a CFG as a list of structured trees. Each tree represents a production with its left-hand side as a root and the derivations as leaves. Figure 2 shows the grammar G = (N, ∑, P, S) with ∑ = {a, b} , N = {S, A} and P = {S → A S ; S → b; A → SA ; A → a }.

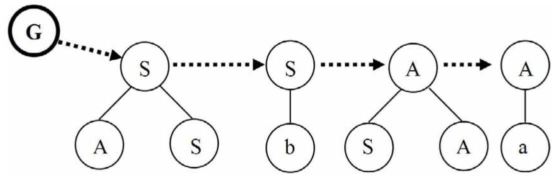

Figure 2: An example of a CFG represented as a list of structured trees.

The initial population can be created with random productions, provided that all the productions are reachable direct or indirectly starting with S.

Using this representation, the crossover operator cannot be applied directly because the grammar consistency can be broken. To overcome this problem we use a point-typed crossover that simply swaps productions with the same left-side.

Preliminary experiments using only the conventional reproduction, crossover and mutation operators have achieved limited performance. These experiments suggested a new operator capable of extracting from positive examples useful productions.

4.1. The Incremental Learning Operator

The crossover and mutation operators are global operators and they are unable to make local optimization such as adding a production which is required to achieve the solution. Also, the fitness measure itself does not indicate which productions are missing. Thus, we propose a new operator named Incremental Learning to perform a local search. This operator is applied before the evaluation of each grammar in the population. It uses the CYK derivation matrix obtained from the positive examples to allow the creation of a useful new production. For each positive example, it repeats the following:

```
Construct the CYK derivation matrix with $V_{r,s}$
If the example is not recognized:
o  If $V_{1,n}$ is not empty, clone the
   root production changing the left-
   hand side by S.
o  If $V_{1,n}$ is empty, add a new
   production in the form $S \rightarrow AB$ such
   that A matches the first half and
   B matches the second half of the
   CYK derivation matrix. If A or B
   is empty, the operator is not
   applied.
```

Once this process is completed with success, hopefully, there will be a set of positive examples (possible all) recognized by the grammar. Although, there is no warranty that some negative examples will still remain being rejected by the grammar.

4.2. Grammar Evaluation

In Grammatical Inference, we need to train the system with both positive and negative examples to avoid overgeneralization [1]. To accomplish that, we use a confusion matrix that is a visualization tool typically used in supervised learning (Table 1). Each column of the matrix represents the number of instances predicted either positively or negatively, while each row represents real classification of the instances.

Table 1: Confusion matrix for a two-class classification problem.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

The entries in the confusion matrix have the following meaning in the context of our study:

- TP is the number of positive examples recognized by the grammar.
- TN is the number of negative examples rejected by the grammar.
- FP is the number of negative examples recognized by the grammar.
- FN is the number of positive examples rejected by the grammar.

There are a several measures that can be obtained from the confusion matrix. In this paper we compute two measures: sensitivity (Equation 1) and specificity (Equation 2). These measures evaluate how positive and negative examples are correctly recognized by the classifier.

$$specificity = \frac{TN}{TN + FP} \quad (1)$$

$$sensitivity = \frac{TP}{TP + FN} \quad (2)$$

The fitness is computed by the product of these measures leading to a balanced heuristic. This fitness measure was proposed by [6] and widely used in other classification problems.

During the grammar evaluation, each production used for recognizing a positive example receives an increment in its positive score. On the other hand, each production that accepts a negative example receives an increment in its negative score. It is important to say that there are two scores for each grammar and both are used to bias the crossover and mutation.

4.3. Genetic Operators

Some adaptations in the ordinary crossover and mutation operators are necessary. In crossover, we use point-typing to preserve grammar consistency. A pair of grammars is selected from the current population based on the selection method. A production with the maximum negative score is chosen. If there are two or more productions with the same negative score, the production with the minimum positive score among them is chosen.

If the second grammar has no production with the same left-hand side of the production chosen, crossover is rejected. Otherwise, the productions are swapped.

In the mutation operation, only one grammar is selected. A production is then chosen using the same mechanism of crossover. A new production with the same left-hand side and with a randomly right-side replaces the production chosen.

5. Experiments

Several experiments using the Tomita language set were done. The standard experimental methodology for most Tomita language induction experiments in the literature is to attempt to induce a grammar which properly classifies all positive and negative examples in a limited training set of binary strings up to some length. The Tomita language set used is shown in Table 2 and the parameters used are shown in Table 3.

To maintain a high diversity in the population during the run, the mutation rate was empirically set to 30%.

Table 2: Tomita's languages.

1	1*
2	(10)*
3	(0|11)*(1*|(100(00|1)*))
4	1*((0|00)11*)*(0|00|1*)
5	(((1|0)(1|0))*(1|0))|((11|00)*((01|10)(00|11)* (01|10)(00|11)*)*(11|00)*)
6	((0(01)*(1|00))|(1(10)*(0|11)))*

Table 3: Parameters used in the experiments.

Population size	500
Initial productions of the form $P \rightarrow \alpha\beta$; $\alpha, \beta \in N$	8
Initial productions of the form $S \rightarrow \alpha\beta$; $\alpha, \beta \in N$	4
Maximum number of productions per grammar	50
Crossover rate	60%
Mutation rate	30%
Reproduction rate	10%
Tournament size	4

A total of 50 independent runs was done for each language and results are shown in Table 4. Results are compared with those provided by [5] who used a similar Evolutionary Computation approach, but with different characteristics. The training set used was the same.

A successful run occurs when a grammar accepts all positive examples and rejects all the negative ones. Table 4 shows the number of successful runs within the 50 runs.

Table 4: Number of successful runs within 50 runs.

Language	**EC approach** [5]	**Our approach**
1	31	50
2	7	50
3	1	12
4	3	16
5	0	7
6	47	50

6. Conclusions

We presented a new Genetic Programming approach for grammatical inference. Each individual is a list of structured trees representing their productions with their left-hand side as a root and the derivations as leaves. The genetic operators are biased by the positive and negative scores of each production. These scores are obtained from the evaluation phase.

We also proposed a new local search operator, named Incremental Learning, which is capable of adjusting each grammar according to the positive examples. This is accomplished by using the information obtained from the CYK triangular matrix. This operator tries to discover which production is missing and then adds it to the grammar.

Results obtained by the proposed GP using the Tomita language set were better than those obtained by other EC-based approach. From a set of 6 languages, in all runs we obtained a solution for 3 of them. This shows that the proposed approach is promising.

A drawback of the approach is that the solution found is not necessarily the smallest one. Depending on the run, the grammar inferred varies in size and, sometimes, it can be difficult to understand. Further work will focus on devising a mechanism able to favor shorter partial solutions.

7. References

[1] C. de la Higuera, "A Bibliographical Study of Grammatical Inference", Pattern Recognition, v. 38, no. 9, 2005, pp. 1332-1348.

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

[3] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA, 1979.

[4] D. H. Younger "Recognition and parsing of context-free languages in time n^3" Information and Control, v. 10, no. 2, 1967, pp. 189–208.

[5] S. Luke, S. Hamahashi and H. Kitano ""Genetic" programming". Proceedings of the Genetic and Evolutionary Computation Conference 1999, San Francisco: Morgan Kaufmann, pp. 76-83.

[6] H. S. Lopes, M. S. Coutinho and W. C. Lima, "An evolutionary approach to simulate cognitive feedback learning in medical domain" Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives. Singapore: World Scientific, 1998, pp. 193-207.