Genetic Programming for Induction of Context-free Grammars

Ernesto Rodrigues [1,2] and Heitor Silvério Lopes [2]
[1]*Fundação de Estudos Sociais do Paraná, Brazil*
[2]*Universidade Tecnológica Federal do Paraná, Brazil*
ernesto@fesppr.br, hslopes@pesquisador.cnpq.br

Abstract

We present an evolutionary algorithm for the induction 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 were modified so as to bias the search and a new operator was proposed. The system was evaluated using benchmark problems and results were compared with another recent approach. Results show that the proposed approach is very promising.

1. Introduction

Grammar induction, also known as grammatical inference, is the task 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.

Grammar induction can be applied to diverse fields such as pattern recognition, information retrieval, programming language and bioinformatics, among others [1].

Several algorithms for grammar induction have been developed lately. They have in common that the most complex class of languages which can be efficiently learnt by provably converging algorithms are the regular languages. For context free languages some recent approaches have shown limited success [2], because the space of possible grammars is infinite. Therefore, learning context-free grammars is still a real challenge for grammar induction approaches.

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

In this paper, we propose an approach for grammar induction based on Genetic Programming (GP) using a local search mechanism. The use of GP in context-free grammar induction is relatively new and only recently promising results have appeared [3, 4].

GP is the automatic generation of computer programs using a process analogous to biological evolution [5]. This technique exploits the process of natural selection, and it is based on a fitness measure to breed a population of candidate solutions that improves over generations. Actually, GP is not restricted to evolve computer programs and it can be used to evolve complex structures such as digital circuits or neural networks [6]. 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 that the proposed method attempts to learn. Next, we describe the GP approach used in this paper. Then, we present experiments demonstrating the effectiveness of the algorithm on a suite of grammar induction benchmark problems. Finally, conclusions and future work are presented.

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 $\alpha \in (N \cup \Sigma)^*$ where * represents the set of symbols that can be formed by taking any number of them, possibly with repetitions. The S is a special nonterminal called the start symbol. The language generated by a CFG G is denoted $L(G)$ [7].

The context-free language $L(G)$ produced from grammar G is the set of all strings consisting only of

0-7695-2976-3/07 $25.00 © 2007 IEEE
DOI 10.1109/ISDA.2007.117

terminal symbols that can be derived from the start symbol S by application of production rules, that is, $L(G) = \{ x \mid S \Rightarrow^* x, x \in \Sigma^* \}$.

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 $\alpha \in \Sigma$.

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. In the CYK algorithm, we construct a triangular table. The horizontal axis corresponds to the positions of the string $w = a_1 a_2 .. a_n$. The table entry V_{rs} is the set of variables $A \in$ P such that $A \Rightarrow^* a_r a_{r+1} ... a_s$. We are interested in whether the start symbol S is in the set V_{1n} because that is the same as saying $S \Rightarrow^* w$, i. e., $w \in L(G)$.

To fill the table, we work row-by-row upwards. Each row corresponds to one length of substrings; the bottom row is for strings of length 1, the second-from-bottom row for strings of length 2 and so on, until the top row corresponds to the one substring of length n which is w itself [8]. The pseudocode is represented as follows:

```
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)  }
```

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

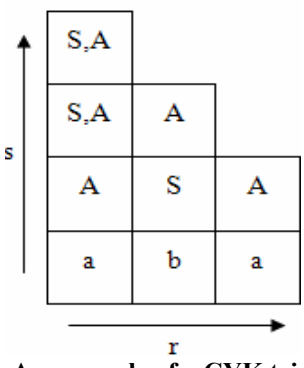

Figure 1: An example of a CYK triangular table.

3. Genetic Programming (GP)

GP is an evolutionary technique used to search over a huge state space of structured representations (computer program). Each program represents a possible solution written in some arbitrary language. The GP algorithm can be summarized as follows [5]:

- Create at random a population of individuals (programs);
- Perform the following three steps until a predefined termination criterion is satisfied:
  - Evaluate the fitness of each individual;
  - Apply a selection method to the current population to select individuals according to their fitness;
  - Modify selected individuals by applying 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 sets of input and output data. Usually, the fitness is a measure of the deviation between the expected output for each input and the computed value given by GP.

There are two main selection methods used in GP: fitness proportionate and tournament selection. 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 in this group is chosen. In this work, we use the tournament selection.

Reproduction is a genetic operator that simply copies a program to the next generation. Crossover, on the other hand, combines parts of two individuals to create two new ones. Mutation changes randomly a small part of an individual.

Each run of the main loop of GP creates 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. GP for Grammar Induction

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, Σ, P, S) with Σ = {a, b}, N = {S, A} and $P = \{S \rightarrow A\ S\ ; S \rightarrow$ b; $A \rightarrow SA; A \rightarrow$ a $\}$.

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

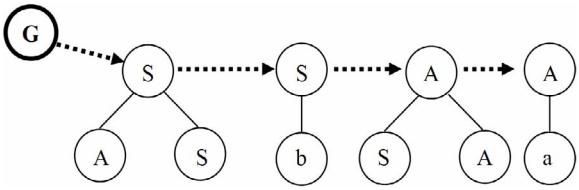

Figure 2: An example of a CFG represented as a list of structured trees.

During the evaluation of a given grammar, 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 operators.

The crossover operator is applied over a pair of grammars and works as follows. First, 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.

The mutation operation is applied to a single selected grammar. A production is then chosen using the same mechanism of crossover. A new production with the same left-hand side and with a random right-side replaces the production chosen.

Unfortunately, using only the biased genetic operators mentioned, the convergence of the algorithm is not guaranteed. In our recent work [4], we demonstrated that the use of an incremental learning operator is needed. They used the information obtained from a CYK triangular table to discover which production is missing to cover the sentence.

In our experiments, we noted that it is also desirable an operator that is capable of extending the grammar, that is, adding new productions.

4.1. The Incremental Learning Operator

Crossover and mutation operators cannot add a new production as required for constructing a satisfactory solution. Also, the fitness measure, by itself, does not indicate which productions are missing. Therefore, we proposed a new operator named Incremental Learning to perform a guided local search [4]. This operator is applied before the evaluation of each grammar in the population. It uses the CYK triangular table obtained from the parsing of positive examples to allow the creation of a useful new production. For each positive example, the following steps are repeated:

- Construct the CYK triangular table with V_{rs}
- If the example is not recognized:
  - If V1n is not empty, clone the root production changing the left-hand side by S.
  - If V1n is empty, add a new production in the form S → AB such that A matches the first half and B matches the second half of the CYK triangular table. 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 remain being rejected by the grammar.

4.2. A New Expansion Operator

In our previous work [4], the set of productions in the initial population affects the entire process. That is, if some type of required production does not exists in any grammar, the algorithm is unable to generate it, unless if the incremental operator generates it. In some experiments, we observed that a limited convergence because the grammar was unable to be expanded.

Therefore, we propose a new genetic operator named expansion. It adds a new nonterminal to the grammar and generates a new production with this new nonterminal as a left side. This new approach allows grammars to grow dynamically in size. To avoid a new useless production, a production with another non-terminal in the left side and the new non-terminal in the right side is generated. It is important to emphasize that the new operator adds two productions to the grammar.

This operator promotes diversity in the population that is required in earlier generations.

4.3. Grammar Evaluation

In grammar induction, 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 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 instances recognized by the grammar.
- TN is the number of negative instances rejected by the grammar.
- FP is the number of negative instances recognized by the grammar.
- FN is the number of positive instances rejected by the grammar.

There are a several measures that can be obtained from the confusion matrix. The most common is total accuracy that is obtained from the total of correct classified examples divided by the total number of instances. In this paper we used two other 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 [9] and widely used in many classification problems.

Before the evaluation, all grammars are verified and corrected if needed, that is, useless and redundant productions are removed.

5. Computational Experiments

The proposed algorithm was implemented in C++ and it was run under Linux operating system, on a desktop computer with Pentium IV 2.4 GHz processor with 1 GBytes of RAM. Table 2 shows the parameters used for the GP. All parameters were determined after preliminary experiments. To maintain a high diversity in the population during the run, the mutation rate was set to 30%. In a future work a sensitivity analysis will be done with the running parameters of GP, aiming at finding optimized parameters.

The initial population was created with a uniform distribution of productions, ranging from 10 to 59. This is only a start point, since before the evaluation phase, all grammars are corrected (useless productions are removed).

Table 2: GP Parameters

Runs	10
Population size	500
Initial population:	
Minimum number of productions	10
Maximum number of productions	59
Tournament size	7
Probability of crossover	60%
Probability of mutation	30%
Probability of expansion	10%

Ten runs were done to accomplish a 10-fold stratified cross-validation procedure [10]. Data was divided randomly into ten parts, preserving in each partition the same class proportionality of the full dataset. Each part is set aside in turn and the learning procedure was done with the remaining nine-tenths. The evaluation is done with the part left behind. The results presented are the average of results obtained in the evaluations.

We need both positive and negative data for training and testing to avoid overgeneralization [1]. Negative examples are hard to construct. Simply generating random strings does not produce a test set sufficiently difficult to distinguish the corrected grammar from a similar, but wrong one. The organizers of the Omphalos competition [2] encountered the same problem.

Recently, Clark and colleagues [11] presented a new grammar inference approach based on String Kernels (SK) and compared with other known approaches, e.g., Probabilistic Context-free Grammars (PCFG) and Hidden Markov Models (HMM). They applied SK to context-free and context sensitive grammars.

We use the same datasets and compared the results with those provided by [11]. The brackets dataset has 537 positive examples and 463 negative examples. The

palindrome dataset has 510 positive examples and 490 negative examples.

We chose the brackets and palindrome languages because they are common test cases for the evaluation of grammatical inference methods. The brackets language is deterministic context-free language, but the palindrome language is nondeterministic. Both languages are context-free.

Table 3 shows the results obtained with our approach. The negative error rate (NER) and positive error rate (PER) are calculated as follows (Equations 3 and 4). In the table, they are represented by N and P, respectively. For each dataset, we report the percentage error rate separately for positive and negative data (the lower, the better).

$$NER = \frac{FP}{FP + TN} \quad (3)$$

$$PER = \frac{FN}{TP + FN} \quad (4)$$

Table 3: Results for brackets (B) and palindrome (P) datasets, and comparison with other approaches

dataset	PCFG		HMM		SK		GPGI	
	N	P	N	P	N	P	N	P
B	0	0	3	1	10	0	0	0
P	6	0	84	3	16	0	14	0

Our approach (represented as GPGI in Table 3) successfully found a correct grammar for the brackets language in all ten runs in at most four generations. In the palindrome language, we achieved an average of 90% of total accuracy in the evaluation phase. However, our approach was not better than PCFG, because our algorithm tries to find an exact grammar not a probabilistic one, as PCFG does. Further work must be done to apply our approach to the inference of probabilistic grammars.

The average execution time was 15 seconds per generation for the brackets language and 6 minutes for the palindrome language.

In Figure 3 we show the values of sensitivity (Sens), specificity (Spec) and total accuracy obtained from the best grammar of one of the runs of the palindrome dataset, from generation 0 to 15 (after the 15th generation there was no significant improvement).

In this figure, during the first seven generations, the algorithm tries to improve both sensitivity and specificity. Between the 8th and 11th generations the specificity and sensitivity values alternate each other. This behavior is caused by the incremental operator which tries to improve the coverage of positive examples (improves sensitivity) and causes the grammar to cover an uncovered negative one (reduces specificity). The expansion operator may cause the same behavior.

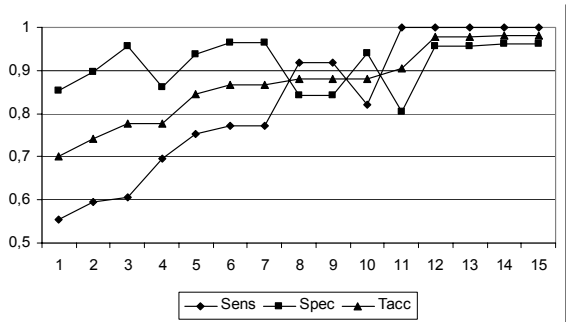

Figure 3: The evolution of the best grammar in the palindrome dataset

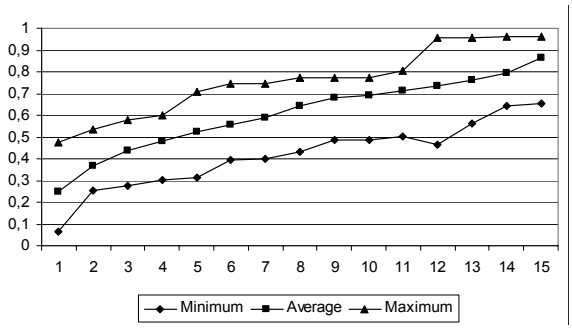

Figure 4: The fitness variation during a run of the palindrome dataset

Figure 4 shows the variation of the minimum, average and maximum fitness during the first 15 generations of a training phase of the palindrome dataset. The slow convergence helps the evolutionary algorithm to avoid local optima.

A frequently found problem of GP implementations is called bloat, the uncontrolled growth of the size of an individual in the population [12]. Figure 5 shows the variation of the size of the grammars during the first 15 generations of a training phase of the palindrome dataset. The curves indicate that the expansion operator is useful during the first 8 generations. Comparing this figure with figures 3 and 4, we observe that the expansion operator creates useful productions in the early generations, thus providing genetic diversity. After the 8th generation, the size of the grammars grows slightly because, before the evaluation, all useless production are removed. Overall, the bloat effect was not detected.

6. Conclusions

We proposed a GP approach for context-free grammar induction. In this approach, an individual is a list of structured trees representing their productions with their left-hand side as the root and the derivations as leaves. The regular genetic operators were biased by the positive and negative scores of each production. These scores are obtained from the evaluation phase.

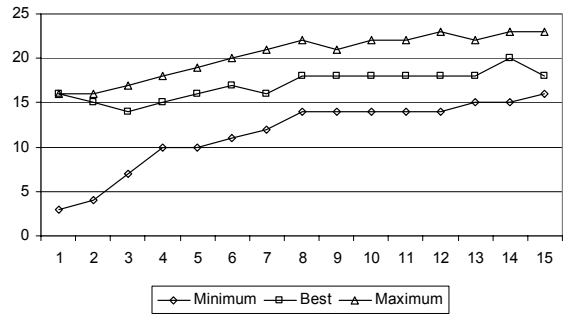

Figure 5: Variation of the size of grammars during a run of the palindrome dataset

We used a local search operator, named Incremental Learning [4], capable of adjusting each grammar according to the positive examples. We also presented a new operator, named expansion, which adds a new production to the grammar allowing the grammars to grow in size. This operator promotes diversity in the population that is required in the earlier generations.

Results obtained by the proposed GP using the Brackets and Palindrome languages set were better than those obtained by recently published algorithms, namely Hidden Markov Models (HMM) and String Kernels (SK) [11]. This shows how promising is the proposed GP approach.

However, 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*, vol. 38, no. 9, pp. 1332–1348, 2005.

[2] B. Starckie, F. Costie and M. van Zaanen, "The Omphalos context-free grammar learning competition," In *Proceedings of the International Colloquium on Grammatical Inference*, 2004, Athens, Greece, pp. 16–27.

[3] F. Javed, B. Bryant, M. Crepinsek, M. Mernik and A. Sprague, "Context-free grammar induction using genetic programming," *Proceedings of the 42nd Annual ACM Southeast Conference '04*, Huntsville, AL, 2004, pp. 404-405.

[4] E. Rodrigues and H.S. Lopes, "Genetic programming with incremental learning for grammatical inference", In *Proceedings of the 6th International Conference on Hybrid Intelligent Systems (HIS'06)*, IEEE Press, Auckland, 2006, pp. 47–50.

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

[6] Banzhaf, W., Nordin, P., Keller, R.E. Francone, F.D. *Genetic Programming : An Introduction,* 3rd ed, Morgan Kaufmann, San Francisco, CA, 2001.

[7] Hopcroft, J. E., Motwani, R. and Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, 2nd Ed., Addison-Wesley, Reading, MA, 2001.

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

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

[10] Witten, I. H. and Frank, E. *Data Mining*, Morgan Kaufmann, San Francisco, CA, 2000.

[11] A. Clark, C. C. Florêncio and C. Watkins, "Languages as hyperplanes: grammatical inference with string kernels," In *Proceedings of European Conference on Machine Learning*, LNAI v. 4212, Springer-Verlag, Berlin, 2006, pp. 90–101.

[12] P. Monsieurs and E. Flerackers, "Reducing bloat in genetic programming," In *Proceedings of the International Conference, 7th Fuzzy Days on Computational Intelligence, Theory and Application*, LNCS v. 2206, Springer-Verlag, 2001, pp. 471–478.