LEARNING GRAMMATICAL RULES FROM EXAMPLES USING A CREDIT ASSIGNMENT ALGORITHM

Dario Bianchi
Dipartimento di Ingegneria dell' Informazione
Universita` di Parma
Viale delle Scienze - 43100 Parma - Italy
Phone: +39 521 905 734 - Fax: +39 521 905 723
Email: bianchi@ce.unipr.it

ABSTRACT

A credit assignment algorithm is proposed for inferring a context free grammar from a set of sample sentences. Only one copy of the grammar is maintained as a population of rules. A strength, associated to each rule, takes in account its usefulness in correctly classify the test sentences. Some results are reported for the language of correctly nested and balanced brackets and for a natural language fragment. In the case of complex grammars the system may be used for grammar refinement, starting from a given set of rules and discovering the remaining ones. A comparison is made with the more usual GA approach, in which a population of complete grammar is maintained. Using a single copy of the grammar and scoring individual rules give better learning and reduces the time spent in parsing the test sentences. A further speed up is obtained doing the parser of multiple sentences in parallel.

1 INTRODUCTION

Grammatical induction is the task of learning a grammar, for a formal language, from a set of sample sentences [1,2] It has practical applications in the field of structural pattern recognition [3], in natural language processing, in information retrieval [4]. Context-free grammars are the most commonly used because they capture much of the structure of natural and artificial languages and can be parsed by efficient algorithms. The induction of the more restricted class of regular languages was also intensively studied.

All the basic methods developed for grammatical induction [1] do not run in polinomial time and the inference of large grammars is computationally infeasible.

It is well known that genetic algorithms (GA) are probabilistic techniques especially suited for difficult search and optimisation problems. So many authors have used genetic algorithms to infer recognizers and grammars for regular and context free languages. Zhou and Grefenstette [8] used GA to induce a finite-state automata while Huijsen [9] used GA for the induction of the rules of a push-down automata. Wyard [10], Lankhorst [11] and Losee [4] have applied GA to induce directly the rules of a context free grammar. Also the genetic programming approach was succesfully used to induce context free grammars[12,13,14].

Using a GA to induce a grammar, requires that each chromosome represents a complete grammar. To calculate the fitness of a single chromosome, all the sentences in the test sets should be parsed. So, considering reasonable values for the number of sentences, the dimensions of the population maintained by the GA and the number of generation required to have a good learning, the computational effort to induce a grammar is very high. Moreover the evaluation function gives a score to the entire grammar encoded in a chromosome, without any judgement about the usefulness of single rules.

A different approach can be suggested by the credit apportionment algorithm used in classifier systems (CS), such as the bucket brigade algorithm [15]. In this case a population of rules (and not of grammars) is maintained and a strength, associated to each rule, takes in account its usefulness in correctly classify the test sentences, in conjunction with the other rules of the population. The advantage of this approach is a better ability to select a coherent set of rules and a better learning rate may be expected. A reduced computation time is required because at each cycle a single grammar should be tested.

2 THE CREDIT ASSIGNMENT ALGORITHM

Given a language L, a grammar G cannot be inferred only from a set of positive sentences $S^+ \subset L$. Actually the grammar guessed from this set can generate a language larger than L. Moreover there is an infinite number of grammars that cover the finite number of strings in the positive set of examples. Also a set $S^- \subset \Sigma^* - L$ of negative examples is required as shown by Gold[2].

In this work, context free grammars in Chomsky's normal form are used. Each rule has the form

$$A \rightarrow B, C$$

where *A,B,C* are non terminal symbols. So all rules have the same length and can be represented as triple of integers, ranging from 0 to the total number of symbols. The rules that rewirite preterminal symbols are not inferred. A tagged input is given to the parser.

As usual, mutation and crossover are used as genetic operators. Mutation randomly changes a single category in a rule. The crossover operator acts at category boundaries.

The system consist of a parser that analyses the sentences of the positive and negative test sets, building a parse table. A credit assigment module gives a score to the grammar rules and a rule generator modifies the population of rules using the genetic operators. The parse table plays the role of the message list in classifier systems.

The credit assignment algorithm consists of the following steps:

1. a grammar is random generated.
2. all the sentences in S^+ and S^- are parsed.
3. terminate if all the sentences are correctly parsed or a time limit is reached.
4. compute the strength of each rule.
5. a fraction of the rules (those with lowest rank) are deleted and substituted with new ones.
6. repeat from step 2

The strength is assigned to the i^{th} rule by the expression:

$$st_i = (u_i^+ - u_i^-) \,/\, fr_i$$

where u_i^+ is the numbers of times rule i^{th} is used for a correct recognition (accepting sentences in S^+ and rejecting sentences S^-) and u_i^- is the numbers of times rule i^{th} is used for a noncorrect recognition (rejecting sentences in S^+ and accepting sentences S^-) and fr_i is the total number of times the i^{th} rule has fired (putting a category in the parse table).

The credit apportionment algorithm we have adopted is simpler than the commonly used bucket brigade one. Actually, in syntactic analysis, there is not a problem of competition between rules, as in CS, and there is no need for a bid mechanism to resolve conflicts. Each element in the parse table can contain more that one category and each category may be constructed by different rules in the case of ambiguous grammars. Moreover the best performance was obtained if the strength at cycle $n+1$ does not depend on that at cycle n (remember that in a cycle all the test sentences are examined). As a consequence, there is no tax for firing.

Both GA and classifiers systems can be implemented on SIMD and MIMD parallel hardware [16]. Usually each processor can deal with one or more (fine-grained or island models) individuals to calculate their fitness in parallel. In CS the matching of classifiers and the auction step can be parallelized. In the case of grammatical induction the most time consuming task is the parsing of the test set. It is possible a) take advantage of parallelism to speed up the parsing of a single sentence b) parse a number of sentences in parallel.

The grammar rules are mantained on the front-end computer. Generation of new rules by the genetic operators is made sequentially. Parsing of sentences and extraction of data for rule scoring is parallelized. A data parallel model was used, implementing the tabular parser of Cocke, Younger and Kasami which deals with context-free grammars in Chomsky's normal form [17]. In the parallel implementation, this algorithm requires a time $O(n)$ and a space $O(n^2)$ where n is the length of the sentence to parse [18]. It was implemented in C* and runs on a CM2 or on a CM5 [19]. It can also parse a number of sentences in parallel. The parse table for each sentence uses $n*(n-1)/2$ processors. The time spent by the parser is that required by the longest sentence (with an overload introduced by the virtual processor management). Also the syntactic tree can be examined, to assign credits to the rules, in a time $O(n)$.

3 RESULTS

Three simple grammars were used to test the system. The “brackets” language consists of all correctly nested and balanced bracket expressions with terminals “(“ and “)”. The grammar can be described by the following rules in Chomsky normal form:

S -> L R
S -> S S
L-> L S

and by the terminal rewriting rules

L-> (, R->)

where S is the starting symbol.

The test set consists of 50 positive examples and 50 negative ones. A population of 4 rules is used, with 1 rule replaced every cycle (the terminal rewriting rules were not considered). At every run a correct grammar is discovered. The mean number of cycles needed is 58 (here and in the following experiments the data are averaged over 10 runs).

A slightly more complex grammar consists of the expression containing round, square and curl brackets:

S -> La Ra
S -> Lb Rb
S -> Lc Rc
S -> S S
La ->La S
Lb -> Lb S
Lc -> Lc S

with the preterminal rewriting rules:

La -> (, Ra ->)
Lb -> [, Rb ->]
Lc -> {, Rc ->}

In this case a population of 20 rules is used with 5 rules replaced at every cycle. All runs are successful, and the mean number of cycles required is 319.

An example of grammar found is given in table 1.

#	RULE	Strength
0	S->Lc Rc	1.567568
1	Lc->Lc S	1.529412
2	La->La S	1.230770
3	S->Lb Rb	1.409091
4	S->S S	1.481482
5	Ra->Ra S	1.294118
6	S->La Ra	1.333334
7	Rb->S Rb	1.280000
8	Rc->Rc S	1.333333
9	Rc->S Rc	0.896553
10	Lc->S Lc	0.875001
11	La->S La	0.941177
12	Rb->Rb S	1.166667

Table 1. A grammar found for the round, square and curl brackets language. Rules with strength 0 are not shown.

Discarding the rules that have strength 0 and do not contribute to the parsing, there are 13 remaining rules (the rules rewriting the preterminals are not considered), which are much more than the 7 rules of the target grammar. Some rule is redundant, like 1 and 9 (both allows to embed an expression in curl brackets), or rule 5 and 11 (an expression can precede or follow round brackets) etc. So the discovered grammar has a degree

of ambiguity higher than the target one, that is it may give a higher number of parse three for a sentence of the language. Nevertheless the two grammars define the same language.

In fig. 2 is reported the learning behaviour for this example. The increments of performance happens in steps because there are subsets of rules strongly related. A new rule introduced in the grammar can activate the other rules of the subset.

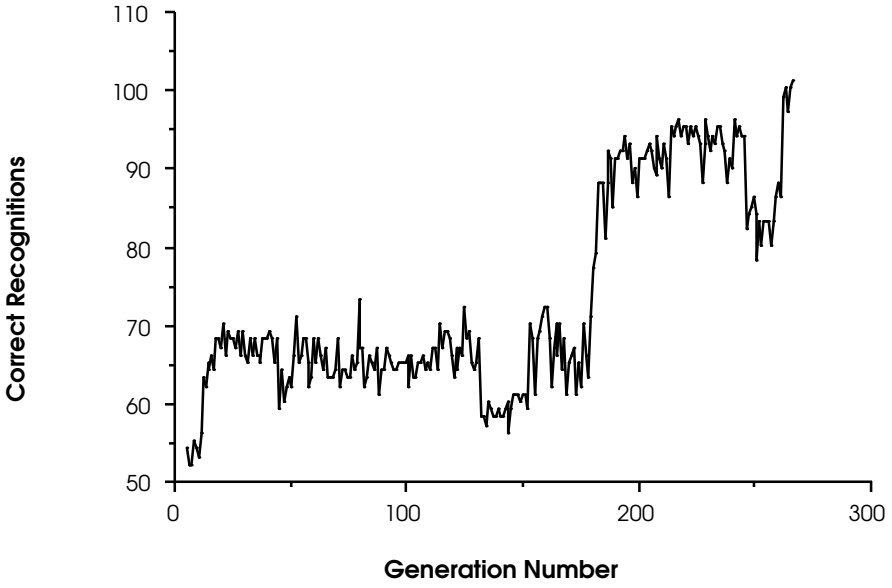

Fig.2 The number of sentences correctly analysed as a function of generation number for the round, square and curl brackets language. This example took 262 generations to reach a 100% score.

As a last example a natural language fragment (toy-NL) grammar is tested.

```
s   -> np vp
np -> det n
np -> np pp
pp -> prep n
vp -> v np
vp -> vp pp
```

In this case about 3000 cycles are required to correctly analyse the 100 sentences in the test set. Actually, it is very easy, for the system, to develop a grammar that covers all positive examples, but many of the grammars discovered overgenerate and covers also a part of the negative sentences.

A better performance can be obtained giving some initial knowledge to the system. The three rules for a simple sentence without the pp adjunct

```
s   -> np vp
np -> det n
vp -> v np
```

were initially given to the system, which can easily discover the remaining rules needed to classify correctly all the sentences. In this case a population of 10 rules is used and only 275 cycles are required.

Two of the grammars found in the refinement of toy-NL language are listed below. In table 2 a grammar shorter than the target grammar is found. We note that the target grammar is ambiguous because a prepositional phrase can be attached to a noun phrase or to a verbal phrase (an ambiguity required by natural language). The grammar discovered lacks of the rule vp -> vp pp giving a non ambiguous grammar that generates the same language of the target grammar. But other form of recursion are found like the one reported in table 3 where the pp attachment is given by the two mutually recursive rules 4 and 6.

#	RULE	Strength
0	s -> np vp	11.515152
1	vp -> v np	11.176471
2	np -> det n	11.000001
3	pp -> prep n	11.000000
4	np -> np pp	1.053764

Table 2. A grammar found for the toy-NL language

#	RULE	Strength
0	s -> np vp	11.639345
1	vp -> v np	11.176471
2	np -> det n	11.000000
3	pp -> prep n	11.000000
4	vp -> prep pp	1.269842
5	np -> np pp	1.076923
6	pp -> n vp	1.025641

Table 3. A grammar found for the toy-NL language

4 DISCUSSION

We have also tested the learning capability of the GA approach. In this case each chromosome encodes an entire grammar. A variable number of rules is allowed in the chromosome. The genetic operators used were non standard. Mutation acts changing at random a single category in a rule. Crossover preserves information exchanging only whole rules. In this way the chromosomes obtained represents always a good grammar. Moreover the offspring has a different length than the parents. This allows to optimise the number of rules in the grammar.

Also in this case the GA shell runs on the front-end while the parser runs on the Connection Machine.

The fitness, associated to each population individual, is the number or sentences correctly recognised plus the number of sentences correctly rejected. A penalty, proportional to the number of rules, can be added to the fitness to obtain smaller and simpler grammars.

The GA operational model was used. The operators act on members of the population to produce tentative members of the new population. Each new offspring is the product of only one operator. The offspring's individuals compete with each other and with parents for survival into the next generation.

The results we have obtained are summarised in table 5 (all results are mean values over 10 runs) for the brackets languages.

Grammar	# of generation required to learn	population size	# of offsprings produced at each generation
round brackets	25	50	20
round, square and curl brackets	190	100	50

Table 5. The results of the use of a direct GA for two of the test grammars.

The grammar for the round brackets requires 25 generation and for each generation the fitness of 20 new individuals should be evaluated. We have to do 500 parsing of the test sentences in comparison with the 58 parses needed by the credit assignment algorithm.

For the grammars of round, square and curl brackets the number of parses needed is 190*50 = 9500 to de compared with the 319 parses required by the credit assignment. The gain in learning rate is a factor of ten or more and increases with the complexity of the grammar to discover.

A quantitative comparison is difficult in the case of the toy-NL language. Many of the runs using GA give only a partial learning, due to premature convergence with the search stuck in a local maximum. A problem not observed when the credit assignment algorithm is used.

5 CONCLUSIONS

The system based on the credit assignment algorithm has been shown to be useful for grammar induction. In the case of complex grammars the system may be used for grammar refinement, starting from a given set of rules and discovering the remaining ones. The performance obtained is better than that given by the direct use of a GA. A weakness stays in the fact that the number of rules is kept fixed. If this number is too low,

the grammar cannot be induced, if too high, the grammar overgenerates. On the contrary using a GA it is easier to deal with a variable number of rules.

Credits are assigned counting the number of times a rule participates to a correct recognition or rejection of a test sentence. In this way no credit is given when a rule helps to analyse only a part of the sentence even if the whole sentence is not recognised. Some criterion can be developed to take in account the usefulness of a rule in recognising substring of the input, but this applies only to the positive examples. For negative ones, a substring may be correct also if the whole input string should be rejected. Also the generative power of the discovered rules may be tested in order to limit the production of too general grammars. This is possible only if we assume to know the target grammar used to generate the examples but this assumption is not valid in real situations (for example in pattern recognition tasks).

BIBLIOGRAPHY

[1] P.R.Choen and E.A.Feigenbaum, *The Handbook of Artificial Intelligence*, Vol III, Addison-Wesley, Reading, Massachusetts, 1983.

[2] E.M.Gold, *Language Identification in the Limit*, Information and Control 10, pp. 447-474, 1967.

[3] K.S.Fu, *Syntactic Pattern Recognition and application*, Springer-Verlag, New-York 1977.

[4] R.M.Losee, *Learning Syntactic Rules and Tags with Genetic Algorithms for Information Retrieval and Filtering: An Empirical Basis for Grammatical Rules*, to appear in Information Processing & Management.

[5] J.B.Pollack, *The induction of dynamical recognizers*, Machine Learning, vol. 7, no 2, pp 196-227, 1991.

[6] R.L.Watrous and G.M.Kuhn , *Induction of finite state languages using second order recurrent networks*, Neural Computation, vol. 4, no. 3, pp 406-414, 1992.

[7] Z.Zeng, R.M. Goodman and P. Smoth, *Discrete Recurrent Neural Networks for Grammatical Inference*, IEEE Transactions on Neural Networks, Vol. 5, No. 2, pp 320-329, 1994.

[8] 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, pp. 170-174, Atlanta, GA,1986.

[9] W.Hijsen, *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 Twene, Enshede, The Netherlands, 1993.

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

[11] M.M. Lankhorst, *Breeding Grammars: Grammatical Inference with a Genetic Algorithm*, Comp. Science Report CS-R9401, C.S. Department, Univ. of Gronigen, The Netherlands, 1994.

[12] J.R.Koza, *Genetic Programming*, pp.442-445, The MIT Press, Cambridge, Massachusetts, 1992.

[13] A.Zomorodian., *Context-free languages induction by evolution of deterministic pushdown automata using genetic programming*, in E.S.Siegel and J.R.Koza (eds), *Working Notes for the AAAI Symposium on Genetic Programming*, pp.127-133,MIT,Cambridge, MA, USA, 10-12 November 1995.

[14] W.B.Langdon, *Using Data Structures with Genetic Programming*, in J.R.Koza, D.E.Goldberg, D.B.Fogel,R.L.Riolo (eds) *Genetic Programming 96*, MIT Press, 1996.

[15] D.E.Goldberg, *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison Wesley, Reading, MA, 1989.

[16] M.Dorigo and V.Maniezzo, *Parallel Genetic Algorithms: Introduction and Overview of Current Research, in Parallel Genetic Algorithms*, J.Stender ed., IOS Press 1993.

[17] A.V. Aho and J.D. Ullman, *The Theory of Parsing Translation and Compiling, Volume 1: Parsing*, Prentice Hall, Englewood Cliff, NJ, 1972.

[18] R.S. Kosaraju, *Speed of recognition of context-free languages by array automata*, SIAM J.Comput, Vol.4, p. 331, 1975.

[19] D.Bianchi, P. Anelli and G.Tamborino, *Parallel lexical analysis and parsing of context-free languages*, AI*IA Notize, Anno VIII, n. 2, pp 31-33, 1995.