Genetic Programming for Natural Language Parsing *

Lourdes Araujo

Dpto. Sistemas Informáticos y Programación.
Universidad Complutense de Madrid. Spain.
lurdes@sip.ucm.es

Abstract. The aim of this paper is to prove the effectiveness of the genetic programming approach in automatic parsing of sentences of real texts. Classical parsing methods are based on complete search techniques to find the different interpretations of a sentence. However, the size of the search space increases exponentially with the length of the sentence or text to be parsed and the size of the grammar, so that exhaustive search methods can fail to reach a solution in a reasonable time. This paper presents the implementation of a probabilistic bottom-up parser based on genetic programming which works with a population of partial parses, i.e. parses of sentence segments. The quality of the individuals is computed as a measure of its probability, which is obtained from the probability of the grammar rules and lexical tags involved in the parse. In the approach adopted herein, the size of the trees generated is limited by the length of the sentence. In this way, the size of the search space, determined by the size of the sentence to parse, the number of valid lexical tags for each words and specially by the size of the grammar, is also limited.

keywords: Genetic programming, Parsing, Probabilistic Context Free Grammar

1 Introduction

The syntactic structure of a sentence represents the way that words in the sentence are related to each other. This structure includes information on how words are grouped, about what words modify other words, etc. The syntactic structure is needed for later processing in very different applications, such as extracting information from documents, translating documents from one language into another, and also to extract the meaning of the sentence. Accordingly, it would be very interesting to be able to perform parsing in an automatic manner.

However, nowadays parsing is still a difficult task which often produces incomplete or ambiguous structures. Because some observations and experiments [1] suggest that human parsing does not perform a complete search, as traditional parsers do, it can be worthwhile to investigate other alternative search methods with heuristic and random components.

* Supported by projects TIC2003-09481-C04 and 07T/0030/2003.

The application of Genetic Programming (GP) [2] to this problem is very natural, since GP is an evolution-based search method which processes a population of hierarchical structures, or trees, which is the most common and clear representation of a parse. Furthermore, probabilistic grammars provide a way to define a measure of the performance of a parse as a probability. Context Free Grammars (CFG)[1], which represent sentence structure in terms of what components are subparts of other components, are the basis of most syntactic representations of language. The structure of the sentence according to one of these grammars is usually represented as a tree. Probabilistic context free grammars (PCFGs) [3–6], obtained by supplementing the elements of algebraic grammars with probabilities[2], represent an important part of the statistical methods in computational linguistics. They have allowed important advances in areas such as disambiguation and error correction, being the base for many automatic parsing systems. PCFGs for parsing are automatically extracted from a large corpus [7].

This paper presents the implementation of a probabilistic bottom-up parser based on genetic programming which works with a population of partial parses, i.e. parses of sentence segments. The quality of the individuals is computed as a measure of its probability, which is obtained from the probability of the grammar rules and lexical tags involved in the parse. In the approach adopted herein, the size of the trees generated is limited by the length of the sentence. In this way, the size of the search space, determined by the size of the sentence to parse, the number of valid lexical tags for each words and specially by the size of the grammar, is also limited.

Probabilistic grammar-based genetic programming has been previously proposed [8] to incorporate domain knowledge to the algorithm and also to reduce the bloat phenomenon, resulting from the growth of non-coding branches or introns in the GP individuals. This approach that has been proved effective in other problems is particularly suitable for the parsing problem, in which the domain knowledge is the very grammar of the language. In our case, the initialization step does not represent a problem because the initial population is composed only of trees corresponding to grammar rules of the form *lexical tag* $\rightarrow$ *word*, and is limited by the length of the sentence and the lexical tags (noun, verb, etc.) of its words. Furthermore, individuals contains no introns, since only valid parse trees are allowed. The price to pay is the design of more complex genetic operators which only produce valid parses.

[1] A CFG = (T, N, S, R) is defined by a set of terminals, T, a set of nonterminals, N, an initial symbol $S \in \{N\}$, and a set of rules, $\{N^i \rightarrow \eta^j\}$ (η^j is a sequence of terminals and nonterminals).

[2] A PCFG is defined as a CFG along with a set of probabilities on the rules such that

$$\forall i \sum_j P(N^i \rightarrow \eta^j) = 1$$

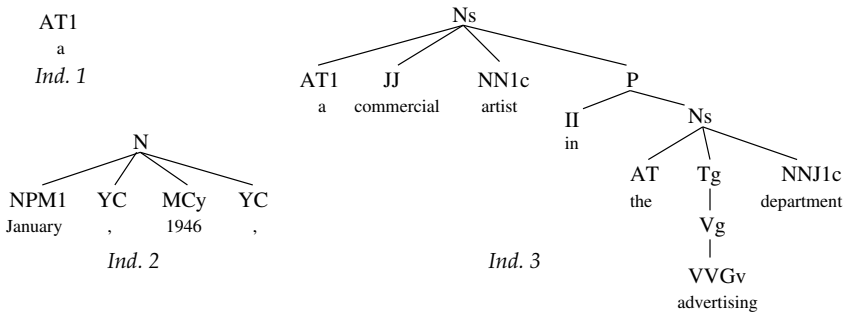

Fig. 1. Examples of individuals for the sentence *The new promotion manager has been employed by the company since January +, 1946 +, as a commercial artist in the advertising department +.*

Evolutionary Algorithms (EAs) have already been applied to some issues of natural language processing [9], and to parsing in particular. In [10], parse trees are randomly generated and combined. The fitness function is in charge of assigning low probability rates of surviving to those trees which do not match the grammar rules properly. This system has been tested on a set of simple sentences, but the size of the population required to parse real sentences with real grammars, as those extracted from a linguistic corpus, is too large for the system to work properly.

The rest of the paper is organized as follows: Section 2 describes the GP parser, including the main elements of the algorithm. Then Section 3 presents and discusses the experimental setup. The paper ends with some conclusions and perspectives for future work.

2 The Algorithm

Parsing a sentence can be sought as a procedure that searches for different ways of combining grammatical rules to find a combination which could be the structure of the sentence. A bottom-up parser starts with the sequence of lexical classes of the words and its basic operation is to take a sequence of symbols to match it to the right-hand side of the rules. We obtain the grammar from a large collection of hand-processed texts or *corpus* in which grammatical structure has already been marked. Apart from the probabilistic grammar and the genetic parameters, the input data of the algorithm are the sentence to be parsed and the dictionary from which the lexical tags of the words can be obtained along with their frequencies.

Let us now consider each element of the algorithm.

2.1 Chromosome Representation

An important issue in designing a chromosome representation of solutions to a problem is the implementation of constraints on solutions. There are two main

techniques to handle this question. One way is to generate potential solutions without considering the constraints, and then to penalize them to reduce their probability of survival. Another way to handle constraints consists in adopting special representations which guarantee the generation of feasible solutions only, and also in defining genetic operators which preserve the feasibility of the solutions. Parsing can be formulated as the search in the set of trees constructed over the grammar alphabet (terminals, P, and nonterminals, N, symbols) of those which satisfies the constraints of the grammar rules. Thus, we can consider any of the two mentioned alternatives to handled this constraint problem.

The first alternative has been adopted in [10]. Though this method works for simple sentences, when we consider real sentences and real grammars extracted from a linguistic corpus, the search space of trees is too large for the GP system to reach a solution in a reasonable amount of time. Accordingly, the algorithm presented herein follows the second alternative.

Individuals in our system are parses of segments of the sentence, that is, they are trees obtained by applying the probabilistic CFG to a sequence of words of the sentence. Each individual is assigned a syntactic category: the left-hand side of the top-level rule of the parse. The probability of this rule is also registered. The first word of the sequence parsed by the tree, the number of words of that sequence and the number of nodes of the tree are also registered. Each tree is composed of a number of subtrees, each of them corresponding to the required syntactic category of the right-hand side of the rule. Figure 1 shows some individuals for the sentence *The new promotion manager has been employed by the company since January +, 1946 +, as a commercial artist in the advertising department +.*, used as a running example, which has been extracted from the Susanne corpus. We can see that there are individuals composed of a single word, such as *1*, while others, such as *3*, are a parse tree obtained by applying different grammar rules. For the former, the category is the chosen lexical category of the word (a word can belong to more than one lexical class); e.g. the category of *1* is *AT1*. For the latter, the category is the left hand-side of the top level rule; e.g. the category of *3* is *Ns*.

Initial Population The first step to parse a sentence is to find the possible lexical tags of each word. They are obtained, along with their frequencies, from a dictionary. Because parses are built in a bottom-up manner, the initial population is composed of individuals that are leave trees formed only by a lexical category of the word. The system generates a different individual for each lexical category of the word. In order to improve the performance, the initial population also includes individuals obtained by applying a grammar rule provided that all the categories of the right-hand side of the rule are lexical. The individual *2* of Figure 1 is one such example.

2.2 Genetic Operators

Chromosomes in the population of subsequent generations which did not appear in the previous one are created by means of two genetic operators: *crossover*

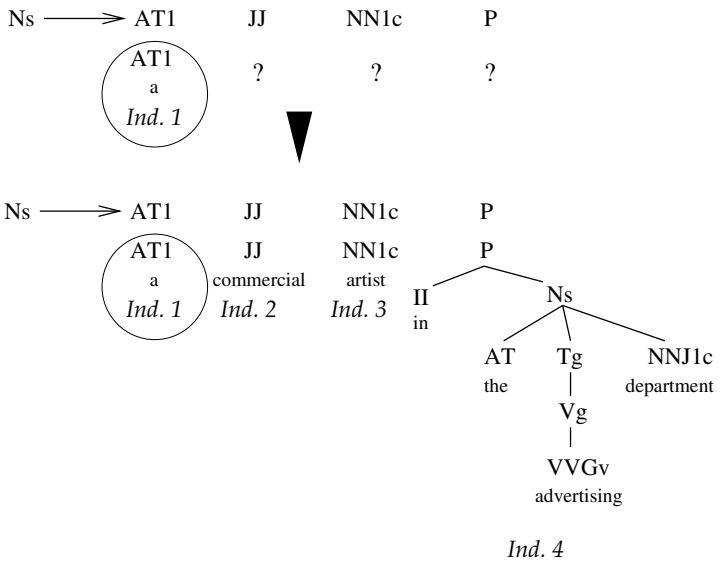

Fig. 2. Example of application of the crossover operator. Individual 1, whose syntactic category is AT1, is randomly selected for crossover. The rule Ns → AT1 JJ NN1c P is selected among those rules whose right-hand side begins with AT1. Finally, the population is searched for individuals corresponding to the remaining syntactic categories of the rule, provided its sequence of words is appropriate to compose a segment of the sentence.

and *cut*. The crossover operator combines a parse with other parses present in the population to satisfy a grammar rule; cut creates a new parse by randomly selecting a subtree from an individual of the population.

At each generation genetic operators produce new individuals which are added to the previous population that in this way is enlarged. The selection process is in charge of reducing the population size down to the size specified as an input parameter. Selection is performed with respect to the relative fitness of the individuals, but it also takes into account other factors to ensure the presence in the population of parses containing words that can be needed in later generations. Elitism has also been included to accelerate the convergence of the process. The evolutionary process continues until a maximum number of generations have passed or until the convergence criterion is reached. This criterion requires to have reached a complete parse of the sentence which does not change during a specific number of generations.

Crossover The crossover operator produces a new individual by combining an individual selected from the population with an arbitrary number of other ones. Notice that the crossover in this case does not necessarily occurs in pairs. The individuals to be crossed are randomly selected. This selection does not consider

the fitness of the individuals because some grammar rules may require, to be completed, individuals of some particular syntactic category for which there are no representatives with higher fitness.

Let us assume that the individual *1* of Figure 1 is selected. The syntactic category (label of the root) of this individual is *AT1*. The next step requires selecting among the grammar rules those whose right-hand side begins with this syntactic category, i.e *AT1*. Some examples from the grammar used in this work are (Ns $\rightarrow$ AT1 JJ NN1c P), (Ns $\rightarrow$ AT1 JJ NN1n P), (Ns $\rightarrow$ AT1 JJ Tg NN1c P), etc. Let us assume that we choose the first of these rules. Now, the crossover operator searches in the population for individuals whose syntactic category matches the remaining categories at the right-hand side of the rule, and whose sequence of words is the continuation of the words of the previous individual (Figure 2). In the example, we look for an individual of category *JJ*, another of category *NN1c* and a third one of category *P*. The sequence of words of the individual of category *JJ* must begin with the word *commercial*, the one following the words of individual *1*. Accordingly, the individual *2* of Figure 2 is a possible candidate (likewise, individuals *3* and *4* are also chosen for the crossover). This process produces the individual *3* of Figure 1 whose syntactic category is the left-hand side of the rule (Ns), and which is composed of the subtrees selected in the previous steps. This new individual is added to the population.

With this scheme, the crossover of one individual may produce no descendant at all, or may produce more than one descendant. In this latter case all descendants are added to the population. The process of selection is in charge of reducing the population down to the specified size.

Crossover increases the mean size of the individuals every generation. Though this is advantageous because at the end we are interested in providing as solutions individuals which cover the whole sentence, it may also induce some problems. If the selection process removes small individuals which can only be combined in later generations, the parses of these combinations will never be produced. This situation is prevented by applying some constraints in the selection process, as well as by introducing the *cut* operator.

Cut operator This operator produces a new individual out of another one by cutting off a subtree of its parse tree at random. The new individual is added to the population.

The rate of application of the cut operator increases with the length of the individuals. Accordingly, the application of the cut operator depends on two parameters, *per_cut* and *threshold_cut*. *Per_cut* is the percentage of application of cut, while *threshold_cut* is the minimum number of words of the individual required to allow the application of *cut*. It is given as a percentage of the length of the sentence being parsed.

2.3 Fitness: Chromosome Evaluation

Because the system only constructs individuals that are valid parses of the sequence of words considered, we do not need to include in the fitness any measure

of feasibility. Thus, the fitness function is basically a measure of the probability of the parse. It is computed as the average probability of the grammar rules used to construct the parse:

$$fitness = \frac{\sum_{\forall s_i \in T} prob(s_i)}{nn(T)}$$

where T is the tree to evaluate, s_i each of its nodes and $nn(T)$ is the number of nodes. For the lexical category, the probability is the relative frequency of the chosen tag.

Selection usually replaces some individuals of the population (preferably those with lower fitness) by others generated by the genetic operators. However, there are two issues that make selection a bit different in our case. First at all, our genetic operators include every new individual in the population, which in this way grows arbitrarily and therefore needs to be reduced to a suitable size. And secondly, if fitness were the only criterion to select the individuals to be eliminated, individuals that are the only ones parsing a particular word of the sentence could disappear, thus making impossible to generate a complete parse of the sentence in later generations. Accordingly, our selection process reduces the size of the population by erasing individuals according to their fitness but always ensuring that each of their words is present in at least another individual.

3 Experimental Results

The GP parser, implemented on a PC in C++ language, has been applied to a set of sentences extracted from the Susanne corpus [11], a database of English sentences manually annotated with syntactic information. The probabilistic grammar for parsing has also been obtained from the Susanne corpus [3]. Each grammar rule is assigned a probability computed as its relative frequency with respect other rules with the same left-hand side [4].

In order to evaluate the quality of the obtained parses, we have used the most common measures for parsing evaluation: recall, precision and accuracy. They are defined assuming a bracket representation of a parse tree. *Precision* is given by the number of brackets in the parse to evaluate which match those in the correct tree; *recall* measures how many of the brackets in the correct tree are in the parse, and *accuracy* is the percentage of brackets from the parse which do not cross over the brackets in the correct parse.

A necessary condition for a parser to produce the correct parse for a sentence is that the required rules are present in the grammar. The Susanne corpus is

[3] In order to simplify the process, those sentences which make reference to elements outside them (*trace* sentences) have not been used to extract the grammar

[4] If we are considering the rule r of the form $A \rightarrow \cdots$, the probability of r is computed as:

$$P(r) = \frac{\#r}{\sum_{r'=A\rightarrow\ldots} \#r'}$$

where $\#r$ is the number of occurrences of r

	225 r.		446 r.		795 r.	
	BFCP	GP	BFCP	GP	BFCP	GP
Precision	99.23	100	99.23	99.01	99.23	97.48
Recall	99.23	100	99.23	99.01	99.23	94.86
Accuracy	98.20	100	98.20	99.01	98.20	97.42
Tag. accuracy	100	100	100	100	100	99.61

Table 1. Results obtained for different sizes of the grammar with a best-first chart parser (BFCP) and with the genetic programming algorithm (GP).

annotated with very large sets of lexical and syntactic tags, what leads to a lack of statistic for many grammar rules, in such a way that many sentences are parsed with rules which do not appear in any other sentence. Because we are mainly interested in evaluating a parser, this problem can be circumvented by applying the parser to sentences from the training corpus. Thus we have tested the parser on a set of 17 sentences from the training corpus (the average length of the sentences is 30 words). In order to compare the GP parser with a classic parser, we have implemented a classic *best-first chart parsing* (BFCP) algorithm. It uses a data structure called *chart* to store partial results of matches already done, thus avoiding to try the same matches again and again, and explores the high-probability components first. Table 1 shows the precision, recall, accuracy and tagging [5] results obtained for grammars of different sizes (best results achieved in ten runs). We can observe that the results of the GP parser improve those of a classic chart parser for the first grammar. Though these results get a bit worse when the size of the grammar is enlarged, they can be improved again by modifying the parameters of the GP algorithm (those employed are suitable for the grammar of 225 rules). Anyway, the Susanne corpus produces too large grammars, inappropriate for the GP parser, so we expect to improve the results by using a more appropriate corpus.

The most remarkable point of the obtained results is that GP is able to reach a 100% in all three measures, while the probabilistic chart parsing does not reach this value simply because the correct parse of some sentences is not the most probable one. In this way the heuristic component of the GP algorithm shows its usefulness for parsing.

3.1 Studying the GP parameters

Some experiments have been carried out in order to determine the most appropriate values for parameters of the GP algorithm. Table 2(a) shows the results obtained for different sizes of the population. A population size of 100 is the minimum required to reach the complete parse of all the sentences in 40 generations. With this number of generations, enlarging the population worsens the results. Table 2(b) shows that results improve with the number of generations for a fixed population size. A number of generations smaller than 30 is insufficient to achieve the complete parse of all sentences with the parameters chosen.

[5] rate of words which have been assigned the correct lexical tag

Population Size	Precision	Recall	Accuracy
100	100	100	100
150	98.89	98.89	97.42
200	98.89	98.89	97.42

(a)

Generations Number	Precision	Recall	Accuracy
30	98.12	98.12	95.63
35	99.23	99.23	98.20
40	100	100	100
45	100	100	100

(b)

Table 2. Results obtained for different population sizes with a maximum number of generations of 40 (a) and for different numbers of generations and a population size of 100 individuals (b). Crossover rate is 40%, cut rate %30 and the threshold value to apply cut is |s| / 3, where |s| is the length of the sentence.

Crossover Rate	Precision	Recall	Accuracy
25	97.14	97.14	94.65
30	97.91	97.91	96.44
40	100	100	100
50(a)	99.23	99.23	98.20
50(b)	100	100	100

(a)

Cut Rate	Precision	Recall	Accuracy
5	97.19	97.19	95.63
10	98.12	98.12	96.44
20	98.12	98.12	96.44
30	100	100	100

(b)

Table 3. Results obtained for different rates of crossover(a) (with a cut rate of %30) and for different rates of the cut operator(b) (with a crossover rate of 40%), a population size of 100 individuals (except the last row, in which it is 120), a maximum number of generations of 40, and a threshold value to apply cut of |s| / 3.

Other parameters which have been investigated are the rates of application of the genetic operators. Table 3(a) shows the results for different rates of the crossover operator and Table 3(b) for different rates of the cut operator. From Table 3(a) we can observe that the results improve with the crossover rate until a certain rate (40%). Enlarging the crossover rate beyond this value slightly spoils the results, because higher crossover rates require larger populations, as the last row of the table shows. Results also improve with the rate of the cut operator as Table 3(b) shows. Another parameter which has been investigated is the threshold value of the length of the sequence of words parsed by an individual to allow the application of the cut operator to it. The best results are obtained when cut is only applied to individuals which parse a sequence of words longer than a third of the sentence length. Notice that for all the parameters studied, there is at least a value for which precision, recall and accuracy are 100%. This strongly scores for the GP as compared to classical methods.

4 Conclusions

This paper describes a genetic programming scheme for natural language parsing. This parser uses statistical information to select the most appropriate interpretation of an input sentence. This information is given by a probabilistic

grammar as well as by the frequencies of the lexical categories corresponding to each word. Because this information is automatically obtained from a linguistic corpus, the parser is language independent.

The described parser has been applied to a number of sentences, some of which present some lexical or grammatical ambiguity which can originate multiple parses. In this situation the GP parser has been able to improve the results of the classical parsers.

The population of the GP parser is composed of partial parses, i.e. parses of sentence segments, which are always valid. In this approach the size of the trees generated is limited by the length of the sentence. In this way, the size of the search space, determined by the size of the sentence to parse, by the number of valid lexical tags for each words and specially by the size of the grammar, is also limited. Because the system is devoted to bottom-up parsing, which builds the parse starting from the words of the sentence, the initial population is composed only of leave trees, corresponding to the assignment of lexical tags to words. In this way, the initialization step does no represent a bottleneck. This incremental approach to build the trees can be applied to other problems in which the size and composition of the programs to be generated by the GP system are limited.

Future experiments are planned to improve the behaviour of the system including the introduction of other genetic operators and the study of other fitness functions, as well as a possible combination of the parsing problem and the tagging problem of assignment of lexical tags to the words of a text.

References

1. Pinker, S.: The Language Instinct. Harper Collins (1994)
2. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
3. Charniak, E.: Statistical Language Learning. MIT press (1993)
4. Brew, C.: Stochastic hpsg. In: Proc. of the 7th Conf. of the European Chapter of the Association for Computational Linguistics, Dublin, Ireland, University College (1995) 83–89
5. Abney, S.: Statistical methods and linguistics. In Klavans, J., Resnik, P., eds.: The Balancing Act. MIT Press (1996)
6. Charniak, E.: Statistical techniques for natural language parsing. AI Magazine **18** (1997) 33–44
7. Charniak, E.: Tree-bank grammars. In: Proc. of the Thirteenth National Conference on Artificial Intelligence. Volume 2., AAAI Press / MIT Press. (1996) 1031–1036
8. Ratle, A., Sebag, M.: Avoiding the bloat with probabilistic grammar-guided genetic programming. In Collet, P., Fonlupt, C., Hao, J.K., Lutton, E., Schoenauer, M., eds.: Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001. Volume 2310 of LNCS., Creusot, France, Springer Verlag (2001) 255–266
9. Kool, A.: Literature survey (2000)
10. Araujo, L.: A parallel evolutionary algorithm for stochastic natural language parsing. In: Proc. of the Int. Conf. Parallel Problem Solving from Nature (PPSNVII). (2002)
11. Sampson, G.: English for the Computer. Clarendon Press, Oxford (1995)