Applied Artificial Intelligence
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713191765

STOCHASTIC PARSING AND EVOLUTIONARY ALGORITHMS

Lourdes Araujo [a]
[a] Languages and Computing Systems Department, UNED (Universidad Nacional de Educaction a Distancia), Madrid, Spain

Online Publication Date: 01 April 2009

To cite this Article Araujo, Lourdes(2009)'STOCHASTIC PARSING AND EVOLUTIONARY ALGORITHMS',Applied Artificial Intelligence,23:4,346 — 372
To link to this Article: DOI: 10.1080/08839510902830650
URL: http://dx.doi.org/10.1080/08839510902830650

Applied Artificial Intelligence, 23:346–372

ISSN: 0883-9514 print/1087-6545 online
DOI: 10.1080/08839510902830650

Taylor & Francis
Taylor & Francis Group

STOCHASTIC PARSING AND EVOLUTIONARY ALGORITHMS

Lourdes Araujo
Languages and Computing Systems Department, UNED (Universidad Nacional de Educaction a Distancia), Madrid, Spain

□ *This article aims to show the effectiveness of evolutionary algorithms in automatically parsing sentences of real texts. Parsing methods based on complete search techniques are limited by the exponential increase of the size of the search space with the size of the grammar and the length of the sentences to be parsed. Approximated methods, such as evolutionary algorithms, can provide approximate results, adequate to deal with the indeterminism that ambiguity introduces in natural language processing. This work investigates different alternatives to implement an evolutionary bottom-up parser. Different genetic operators have been considered and evaluated. We focus on statistical parsing models to establish preferences among different parses. It is not our aim to propose a new statistical model for parsing but a new algorithm to perform the parsing once the model has been defined. The training data are extracted from syntactically annotated corpora (treebanks) which provide sets of lexical and syntactic tags as well as the grammar in which the parsing is based. We have tested the system with two corpora: Susanne and Penn Treebank, obtaining very encouraging results.*

INTRODUCTION

When we listen to a sentence, independently of its meaning, we know if it is or is not correctly built. The mental representation that we have of the grammar of the language allows us to decide on the correctness of the construction, even if we have not listened to it before, and even if it contains words that we do not know. The grammar is an example of a combinatorial discrete system (Pinker 1994) in which a finite number of elements, the words, are combined to create more extensive structures, which are the sentences with different properties from those of the elements that compose them. This combinatorial feature enables the grammar to generate virtually infinite correct constructions from a finite number of words. Another property of the grammar is that it

Supported by projects TIN2007-68083-C02-01 and TIN2007-67581-C02-01.
Address correspondence to Lourdes Araujo, UNED, Dpto. Lenguajes y Sistemas Informáticos, c/Juan del Rosal 16, Madrid 28040, Spain. E-mail: lurdes@lsi.uned.es

is autonomous with respect to other cognitive capacities. The grammar establishes the form in which the words should be combined to express meanings, and that form is independent of the meanings themselves. Because of this, we are commonly able to understand the meaning of sentences that do not exactly match the rules of the grammar. Similarly, we can recognize meaningless sentences as grammatical.

From these considerations it makes sense to regard parsing as an autonomous search process for grammatical structures in a combinatorial discrete space. Classical parsing methods are based on complete search techniques to find the different interpretations of a sentence. On the one hand, human parsing does not seem to perform a complete search but some kind of heuristic process, among other things, because parsing begins even before the sentence is complete. This suggests exploring alternative search methods where a degree of uncertainty is allowed to achieve tractability and robustness. Evolutionary algorithms are among these kinds of techniques. On the other hand, exhaustive search techniques can overwhelm the system capabilities when applied to extensive grammars like the ones automatically obtained from corpora.

Evolutionary algorithms (EAs) are not guaranteed to reach the optimum solution but a reasonably good approximation, according to the resources assigned (time and memory). Moreover, the tasks involved in natural language processing, and parsing in particular, are a special case due to ambiguity, which in general makes it difficult to determine what is the best solution. Even if we consider some techniques to resolve the ambiguous cases, they do not provide a general solution. For example, if we focus on probabilistic techniques to solve ambiguity, we must take into account that the most probable parse is not always the correct one. These reasons make approximate techniques very appropriate for Natural Language Processing (NLP).

This work investigates the application of evolutionary algorithms to parsing. Evolutionary algorithms imitate nature in deciding the way the system is going to change (to evolve), i.e., they are based on the production of offsprings and on natural selection or survival of the fittest to the environment. In an evolutionary algorithm, the best candidates at a given time are favored, but the remainder also has chances of surviving (although less). This is important because if the circumstances (the environment) change, some individuals which were little adapted can pass to be the fittest in the new environment. It is of course true that nature spends millions of years to carry out those changes, while an evolutionary algorithm is expected to provide an answer in a much shorter time. Evolutionary algorithms include mechanisms to explore the search space without carrying out an exhaustive exploration, and at the same time to focus the search in a specific direction (they do not carry out a random blind search), which can vary along the evolution.

Evolutionary algorithms have already been applied to some issues of natural language processing (Kool 1999), such as query translation (Davis and Dunning 1996), inference of context-free grammars (Wyard 1991; Smith and Witten 1995; Losee 1996; Keller and Lutz 1997), phonological parsing (Belz 1998), morphological analysis (Kazakov 1997; Kazakov and Manandhar 2001), part-of-speech (POS) tagging (Araujo 2002b), semantic interpretation (Rose 1999) and dialogue (Nettleton and Garigliano 1994).

In this article we explore different alternatives for implementing an evolutionary bottom-up parser. Individuals representation, which should facilitate its evaluation and the application of the genetic operators, is highly determined by the kind of parsing. Previous experiments (Araujo 2002a, 2004b) have shown the convenience of working with a bottom-up parser, which works with a population of partial parses (Araujo 2004a), which is the alternative explored in this work. Another consideration is that in natural language not all the constructions are equally frequent. Thus we can bias towards the most probable constructions, although it is important to keep in mind that not always the most probable construction is the most appropriate one. Applying evolutionary algorithms to natural language processing, we can assimilate the most adapted constructions with the most probable ones, so that the most probable parses are favored with a higher probability, although permitting a certain degree of survival to others, which can eventually be part of a better solution. Apart from guiding the search, statistical parsing provides a way of dealing with disambiguation (Charniak 1993). Although the first research in probabilistic context free grammar (PCFGs) suggested that they were poor models for language, later works (Charniak 1996, 1997) shown that they can produce useful results on parsing. More recent works on best-first strategies have achieved further and further improvements on the performance. Caraballo and Charniak (1998) provide a detailed study of different figures of merit (FOM) which can be used to compare the probability of the constituents of a parse. They proposed a particular FOM, which outperformed the others in their experiments. In this FOM, which considers contextual information, a constituent probability is estimated according to a simple model of the context on both sides of the constituent, and uses a trigram model for the estimation of the probability of the sequence of tags, which is part of the estimation.

The work by Charniak, Goldwater, and Johnson (1998) uses the FOM proposed by Caraballo and Charniak (1998). They present a parser that also ranks incomplete constituents, or edges, along with complete constituents. This change leads to an important improvement in the performance, with a small reduction of the accuracy. This parser is implemented by transforming the grammar in a binary one, in which every rule is unary or binary. Blaheta and Charniak (1999) achieve a further improvement of the performance, with very little decrease in the accuracy.

This improvement is based on the observation that parsers based on FOMs tend to spend too much time in one part of the sentence, finding multiple parses for the same substring, while other parts of the sentence are often ignored in the meantime.

There have also been other works on parsing that focus on other kinds of grammars or additional information. Charniak (1997) refines the statistical model for CFG by using the notion of lexical head of a constituent. He also finds in the experiments reported in this work that statistics on individual words outperform statistics on word classes. Collins (1997) presents a generative model of lexicalized CFG for statistical parsing. This model includes the treatment of subcategorization and wh-movement. Charniak and Carroll (1994) propose an approximation to a context-sensitive model, in which the probability that a nonterminal expands using a particular grammar rule depends on its parent. Klein and Manning (2003a) present an extension of the A^* search algorithm to tabular PCFG parsing. This method guarantees to find the most likely parse, not just an approximation. Collins (1999) provides a detailed review of different works on the topic.

There also exists a number of available parsers, some of which have been developed on the basis of some of the above-mentioned statistical models. The CMU link parser (Sleator and Temperley 1993) uses link grammars for parsing, which assign to a sentence a structure very different from the one assigned by a PCFG. The Stanford parser (Klein and Manning 2003b) used unlexicalized PCFG, reaching a performance close to that of the lexicalized model by using some extra linguistic annotations. The parser by Collins (1996) uses lexical information by modeling head-modifier relations between pairs of words. The Bikel parser (2004) is also based on a lexicalized statistical model, which tries to reduce the complexity of Collins' parsing model. The parser by Charniak (2000) is based on a maximum-entropy model. We have used some of them in the evaluation of our system.

It is not the purpose of this work to propose a new statistical model for parsing. What we propose instead is a new parsing algorithm that can be used in place of a best first chart parser.[1] Any statistical model to assign FOMs to the constituents of the parsing process, complete or incomplete, can also be used in the evolutionary parser. The FOM, which a best first chart parser uses to compare the probability of the constituents, is used in the evolutionary parser as the "fitness" which guides the selection process during the evolution. Because of this, we have adopted a simple statistical model, which simplifies the implementation. What we want to compare in this work are the results, both in efficiency and accuracy, obtained with a chart parser and the evolutionary one. The training data used are extracted from corpora syntactically annotated, or treebanks, which provide the sets of lexical and syntactic tags as well as the grammar in

which the parsing is based. We have tested the system on two corpora: Susanne and Penn Treebank.

EVOLUTIONARY ALGORITHMS: MAIN ELEMENTS

Nowadays, evolutionary algorithms have been shown to be practical search and optimization methods, applied in diverse areas, such as planning or machine-learning (Michalewicz 1994). Evolutionary algorithms mimic the principles of natural evolution: heredity and survival of the fittest individuals. Different evolutionary programs can be formulated for a particular problem. Such programs may differ in many ways, depending on the representation of individuals, on the genetic operators for transforming the individuals, on the methods for creating the initial population, on the parameters, etc. Genetic algorithms, introduced by Holland (1975) were originally proposed as a general model of adaptive processes, but by far, their largest application is in the domain of optimization. Evolutionary programming, introduced by Fogel (1962), was originally offered as an attempt to create artificial intelligence. The approach was to evolve finite state machines (FSMs) to predict events on the basis of former observation. Evolution strategies, as developed by Rechenberg (1973) and Schwefel (1975), were initially designed with the goal of solving difficult discrete and continuous parameter optimization problems. Another interesting approach, called genetic programming, was proposed by Koza (1992). In this case, instead of building an evolutionary algorithm to solve a problem, the algorithm searches the space of possible programs (in a particular language) for the best one. The space of programs can be regarded as a space of rooted trees, i.e., structures without a predefined size. The evaluation of an individual is based on its ability to solve a selected set of test cases.

Nevertheless, the different kinds of EAs share a common structure, shown in Figure 1. Systems based on evolutionary algorithms maintain a population P of potential solutions, and are provided with some selection process (*individuals_selection*) based on the ability of the individual to solve the problem, which is called its fitness F (*evaluation*). The population is renewed (*new_generation*) by replacing individuals with those obtained by applying "genetic" operators to selected individuals. The usual "genetic" operators are *crossover* and *mutation*. Crossover obtains new individuals by mixing, in some problem-dependent way, two individuals, called parents. Mutation creates a new individual by performing some kind of change on an individual. The production of new generations continues until resources are exhausted (*termination_condition*) or until some individual in the population is fit enough (*required_fitness*).

The approach adopted herein is close to genetic programming since we work with variable size trees. However, in our case the evaluation is

```
evolution program
begin
    generation = 0
    P = initialize_population
    F = evaluation(P)
    while not required_fitness(F) and
          not termination_condition do
    begin
       generation = generation + 1
       I = individuals_selection(P, F) %for reproduction
       P = new_generation(P, I) % genetic operators
       F = evaluation(P)
    end
end
```

FIGURE 1 Structure of an evolutionary algorithm.

based on the ability to solve a particular case, i.e., parsing a particular sentence, as it is usual in genetic algorithms.

EVOLUTIONARY PARSER DESIGN: TOP-DOWN OR BOTTOM-UP

Parsing a sentence can be sought as a procedure which searches for different ways of combining grammatical rules to find a combination that could be the structure of the sentence. A top-down parser starts with the initial symbol of the grammar, *S*, and searches for rules to rewrite nonterminal symbols (rules having this symbol at the left-hand-side) until achieving a sequence of terminal symbols which matches the lexical classes of the words in the input sentence. A bottom-up parser starts with the sequence of lexical classes of the words, and its basic operation is to match a sequence of symbols to the right-hand-side of a rule. Thus, this parser can be implemented simply as a search procedure for this matching process.

Though we can define different representations for the individuals, they must always represent in some way potential solutions, in our case, parse trees. Furthermore, the selected representation must facilitate the evaluation of the individuals, in this case, its ability to parsing the sentence considered. Because of this, it is reasonable that a top-down parser works with complete parses of the sentence, in order to be able to evaluate the distribution of the sentence words among the grammar rules. On the other hand, the intermediate constructions of a bottom-up parser can be evaluated, because they are naturally associated with a segment of the sentence.

Another 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 of adopting special representations that guarantee the generation of only feasible (valid) solutions, and also in defining genetic operators that 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 that satisfy the constraints of the grammar rules. Thus, we can consider any of the two mentioned alternatives to handle this constraint problem.

The first alternative has been adopted in the top-down parser (Araujo 2002a). Since it works with complete parses, if we enforce individuals to perfectly match the grammar rules, there would be no space for diversity and evolution under natural selection in most cases (only in the presence of ambiguity). Because of this, in this case, the parse trees are randomly generated, only with some minor constraints. This leads to parse trees that can be inconsistent with the grammar, being the fitness function in charge of penalizing them. However, the search space to be explored with this approach is too large. The system was 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, was too large for the system to work properly.

On the other hand, the bottom-up parser proposed in this article works with partial parses (Araujo 2004a) corresponding to different segments of the sentence. It leaves space for diversity, not only because of the possibility of applying different grammar rules to parse the same segment of the sentence, but also because the sentence can be partitioned in different ways. Accordingly, in this case, the parse (sub)trees are generated in such a way that they are always coherent with the grammar.

BOTTOM-UP EVOLUTIONARY PARSER

This section is devoted to a probabilistic bottom-up parser which works with a population of partial parses, i.e., parses of sentence segments. Only valid parse trees are allowed, and thus, the measure of the quality of the individuals does not require an inclusion of any contribution accounting for the feasibility. Because individuals are partial parses, the fitness can measure how far they are from completing the whole parse of the sentence, and also their probabilities, if we consider probabilistic grammars. We obtain the grammar from a treebank, i.e., a large collection of hand-processed texts in which the grammatical structure has already been marked.

Let us now consider each element of the algorithm separately.

Chromosome Representation

Individuals in this system are parses of segments of the sentence, that is, they are trees obtained by applying the CFG (possibly probabilistic) 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 2 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 *Individual 1* is *AT1*. For the latter, the category is the left-hand-side of the top-level rule, e.g., the category of *Individual 3* is *Ns*.

Initial Population

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, such as individual *1* of Figure 2. The possible lexical tags of each word are obtained, along with their frequencies, from a dictionary. 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 of the categories of the right-hand-side of the rule are lexical. Individual *2* of Figure 2 is one such example.

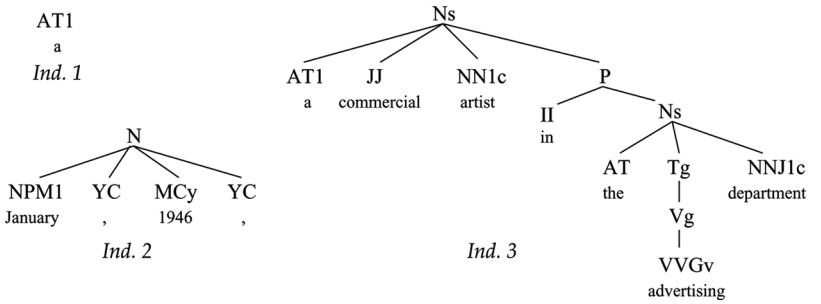

FIGURE 2 Examples of individuals for the sentence *The new promotion manager has been employed by the company since January +, 1946 +, as a commercial artist in the advertising department +.*

Termination Condition

Though the proposed system can be useful in performing partial parsings, since the evolutionary algorithm works with these types of parses, the results we present here correspond to the parsing of the whole sentence. The evolutionary process continues until a maximum number of generations have passed or until the convergence criterion is reached. This criterion requires reaching a complete parse of the sentence which does not change during a specific number of generations.

Genetic Operators

Chromosomes in the population of subsequent generations, which did not appear in the previous one, are created by means of different genetic operators: *crossover*, *mutation*, and *cut*. The crossover operator combines a parse with other parses present in the population to satisfy a grammar rule. We have investigated different approaches for this operator. The *mutation* operator changes a subtree of an individual by another one which parses the same sequence of words with a grammar rule of the same type. The *cut* operator creates a new parse by randomly selecting a subtree from an individual of the population. The rates of application of these operators performed at each step are input parameters. The efficiency of parsing is very sensitive to them.

At each generation, genetic operators produce new individuals which are added to the previous population that in this way are 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 are needed in later generations. Elitism[2] has also been included to accelerate the convergence of the process.

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 occur 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.

Crossover begins by selecting an individual from the population to be combined with others. The next step is selecting among the grammar rules those whose right-hand-side begin with the syntactic category X of the selected individual. Then, for each category of the right-hand-side of

the rule after the first one, the population is searched for an individual whose syntactic category matches the required one, and whose sequence of words is the continuation of the words of the previous subtree. Finally, a new individual is created, whose syntactic category is the one of the chosen rule and is composed of the subtrees of the selected individuals. The 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 that cover the whole sentence, it may also generate 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.

We have investigated two different schemes for crossover. *Conservative* crossover always produces complete parses of a segment of the sentence. *Speculative* crossover may produce parses which lack some subtrees.

Conservative Crossover

This operator produces complete parses of a segment of the sentence, in such a way that if the parse cannot be completed, the offspring being constructed is discarded.

Let us assume that the individual *1* of Figure 2 is selected for crossover. 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 → AT1 JJ NN1c P), (Ns → AT1 JJ NN1n P), (Ns → 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 3). 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 3 is a possible candidate (likewise, individuals *3* and *4* are also chosen for the crossover). This process produces the individual *3* of Figure 2, whose syntactic category is the left-hand-side of the rule (Ns)

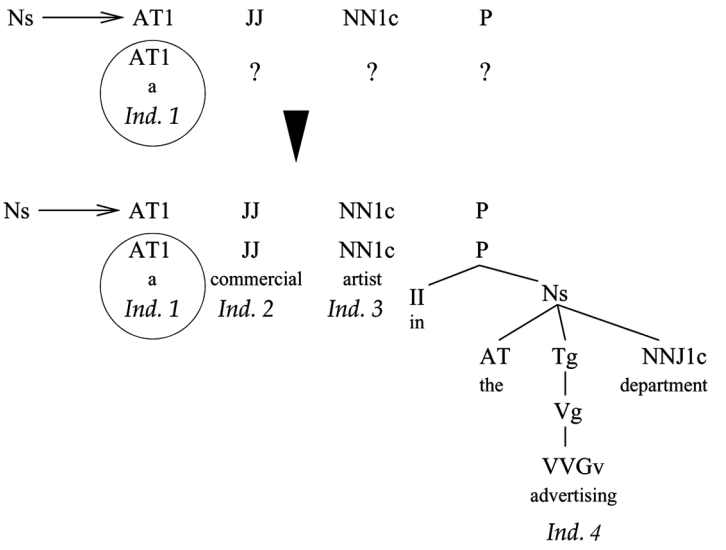

FIGURE 3 An example of application of the conservative crossover operator. Individual 1, whose syntactic category is AT1, is randomly selected for crossover. The rule Ns → AT1 JJ NN1c P is selected among those rules whose right-hand-side begins with AT1. Finally, the population is searched for individuals corresponding to the remaining syntactic categories of the rule, provided its sequence of words is appropriate to compose a segment of the sentence.

chosen at the beginning of the process, and which is composed of the subtrees selected in the previous steps. This new individual is added to the population. The search in the population for each required individual is repeated a number of times, and therefore more than one alternative can be found and thus more than one individual can be produced.

Speculative Crossover

This operator can give rise to *incomplete* individuals, which lack the subtree corresponding to some subsequence of words. The operator works in a different manner depending on whether the selected individual to be crossed is *complete* or not. If it is, the operator selects a grammar rule whose right-hand-side begins with its category, and then searches in the population for individuals to satisfy the remaining categories of the right-hand-side of the rule. However, if some of them cannot be found, the offspring individual is created anyway as an *incomplete* one. Figure 4 shows an example of this case. The resulting *Individual 2* lacks the subtree corresponding to the syntactic categories *Vzfp* and *P*. If, on the contrary, the selected individual is incomplete, the operator randomly searches in the population for individuals to complete each missing subtree. Figure 5 shows an example. *Individual 1* is selected for crossover, and because it is incomplete, the population is searched for individuals corresponding

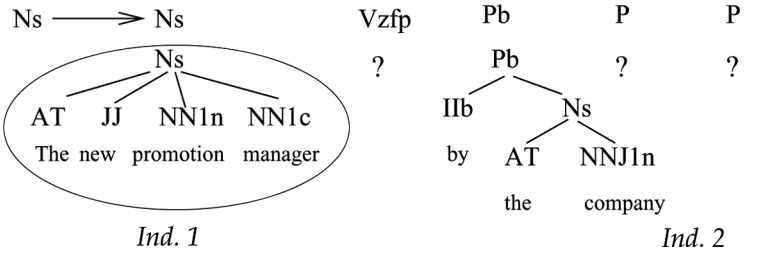

FIGURE 4 An example of application of the speculative crossover operator to a complete individual. Individual 1, which is complete and whose syntactic category is Ns, is randomly selected for crossover. The rule Ns → Ns Vzfp Pb P P is selected among those rules whose right-hand-side begins with Ns. Finally, the population is searched for individuals corresponding to the remaining syntactic categories of the rule. Only an appropriate individual of category *Pb* is found and thus the operator produces an incomplete individual.

to the absent subtrees, according to the syntactic categories of the rule and the sequence of words to be parsed. *Individual 2* properly matches the subtree for the absent category Vxfp and it is added to the individual producing a new one.

Because this crossover can produce parses with subtrees impossible to be combined with others to complete the parse, a mechanism is required to eliminate them. With this purpose, we introduce aging of incomplete individuals. An incomplete individual starts to age as soon as it is created. Its age is incremented every new generation as long as the individual is not combined with another. When the age of an individual reaches a threshold value, given as an input parameter, the individual dies (is discarded).

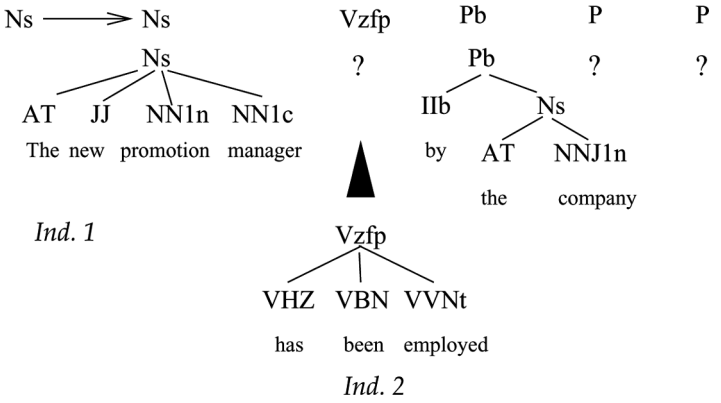

FIGURE 5 Example of application of the speculative crossover operator to an incomplete individual. Individual 1, which is incomplete, is selected for crossover and the population is searched for individuals of category Vzfp, and *P*, which parse the appropriate sequence of words. Individual 2 is found in the population and it is inserted in Individual 1.

Mutation Operator

This operator randomly changes a subtree of an individual by another one with the same syntactic category than the replaced one. This operator works in a slightly different way depending on the kind of crossover, conservative or speculative, that is being used, i.e., depending on whether it is applied only to complete individuals or to incomplete ones as well. When applied to complete individuals, mutation substitutes a subtree by another one provided the new one parses exactly the same sequence of words and has the same syntactic category as root, but performs the parsing in a different manner. Figure 6 shows an example for a possible individual when parsing the sentence from the Susanne corpus *The Fulton County Grand Jury said Friday an investigation of Atlanta +⟨apos⟩s recent primary election produced ⟨ldquo⟩ + no evidence +⟨rdquo⟩ that any irregularities took place +*. The subtree under the node (Ns) selected for mutation is substituted by an individual of syntactic category Ns, which parses the same sequence of words in a different manner.

When applied to incomplete individuals, the new subtree can parse a sequence of words different from the replaced one provided this sequence does not overlap the sequences corresponding to other subtrees of the individual, and it is the continuation of the sequence of the previous subtree if it is present, and the preceding sequence of the following

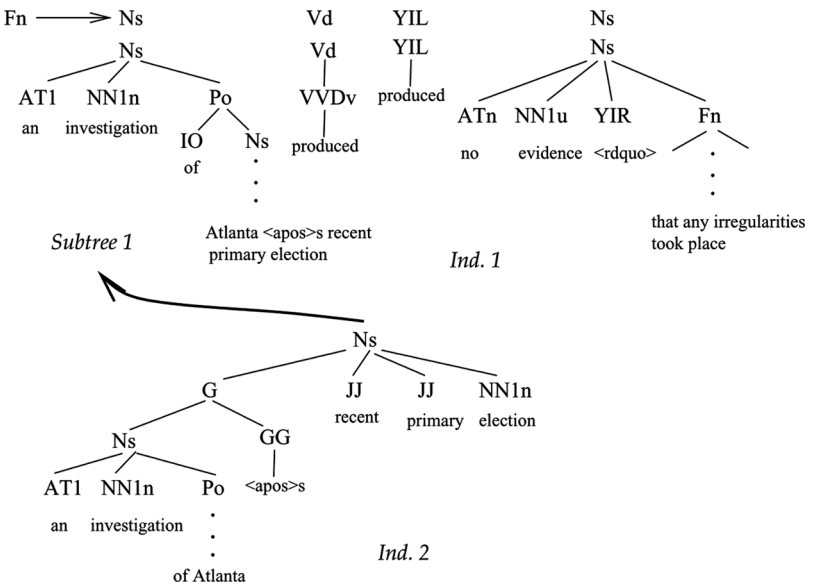

FIGURE 6 Example of an application of the mutation operator to a complete individual. Mutation is applied to Individual 1 and the subtree corresponding to the syntactic category *Ns* is randomly selected for mutation. Then, the population is searched for individuals with syntactic category *Ns*, which parse the same sequence of words (*an investigation of Atlanta's recent primary election*) in a different manner.

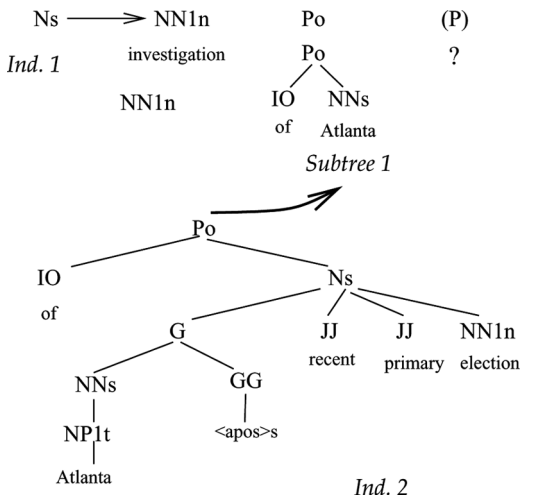

FIGURE 7 Example of an application of the mutation operator to an incomplete individual. The subtree under the node of category *Po*, which parses *of Atlanta*, selected for mutation is substituted by *Individual 2*. It has the category *Po* and parses a longer segment of sentence *of Atlanta* ⟨*apos*⟩*s recent primary election*, since there is no conflict with the remaining subtrees, which are absent.

subtree if it is present too. Figure 7 shows an example of this case. The subtree corresponding to the category Po is substituted by Individual 2, which parses a longer segment of sentence, since there is no conflict with other subtrees.

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. This operator is introduced in order to recover parses previously produced, which may have disappeared during the evolution. The rate of application of the cut operator needs to increase 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.

Fitness: Chromosome Evaluation

As 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. Because this bottom-up parser works with partial parses of the sentence, the fitness measure can include some measure

indicating how far the partial parse is of parsing the whole sentence. On the other hand, if we use probabilistic grammars for parsing, the fitness function can also include some measure of the probability of the parse, which can be defined in different ways. According to these considerations, we have adopted different fitness functions which include one or more of the following criteria (measures (a) and (b) are related to the closeness to parsing the complete sentence, while measures (c) and (d) are related to the probability of the parse):

(a) Length of the segment of sentence being parsed by the individual. This criterion favors the construction of large individuals, what can quickly lead to complete the parse of the whole sentence. It is an open question to investigate whether a fast convergence of the algorithm can produce high quality individuals.
(b) Number of nodes of the parse tree. This criterion can be used to favor the generation of parses for large segments of the sentence, as the previous one. However, this criterion also favors deeper parses.
(c) Measure of the probability of the parse computed as the sum of the logarithm of the probabilities of the grammar rules included in the parse:

$$\mathit{fitness} = \sum_{s_i \in T} \log \mathit{prob}(s_i),$$

where T is the tree to evaluate and the s_i denote its nodes. For the lexical category, the probability is the relative frequency of the chosen tag. This measure can decrease with the size of the parse, thus penalizing the parsing of long segments of the sentence, which can hinder the parsing of the whole sentence. It is also an open question to investigate the convenience of combining this measure with some other measure related to the length of the parsed segment.
(d) Measure of the probability of the parse computed as the average probability of the grammar rules used to construct the parse:

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

where T is the tree to evaluate, the s_i denotes its nodes, and $nn(T)$ is the number of nodes of T. This measure does not depend on the size of the parse.

The next section reports the results of a study carried out to determine the most appropriate definition of the fitness function.

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 it 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 only if each of their words is present in at least another individual. Otherwise, the individual is kept.

EXPERIMENTAL RESULTS

The bottom-up parser, implemented on a PC in C++ language, has been applied to two sets of sentences extracted from the Susanne corpus (Sampson 1995) and from the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1994), databases of English sentences manually annotated with syntactic information. The probabilistic grammar for parsing has also been obtained from the corpora.[3] Each grammar rule is assigned a probability computed as its relative frequency with respect to other rules with the same left-hand-side.[4]

In order to evaluate the quality of the obtained parses, we have used some 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 that 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 grammars directly obtained from a corpus are composed of very specific rules, what lead to a lack of statistic for many grammar rules, in such a way that many sentences are parsed with rules that 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 sets of sentences from the training corpus (17 sentences form the Susanne corpus, with average length of the sentences of 30 words, and 11 sentences form Penn Treebank, with average length of 25 words). In order to compare this evolutionary parser with a classic one, we have implemented a classic *best-first chart parsing* (BFCP) algorithm. In a chart parser, the *chart* structure stores the partial results of the matchings already done. Matches are always

attempted from one component, called *key*. To find rules that match a string involving the key, the algorithm looks for rules that start with the key, or for rules that have already been started by early keys and require the present key either to extend or to complete the rule. The chart records all components derived from the sentence so far in the parse. It also maintains the record of rules that have partially matched but are incomplete. These are called *active arcs*. The basic operation of a chart parser consists in combining an active arc with a completed component. The result is either a new completed component or a new active arc that is an extension of the original active arc. Completed components are stored in a list called *agenda* until being added to the chart. This process is called *arc extension algorithm*, of which Figure 8 shows an scheme. To add a component C into the chart from position p_1 to position p_2, C is inserted into the chart between those positions. Then, for any active arc of the form $X \rightarrow X_1, \ldots, \circ C, X_n$ (where $\circ$ denotes the key position) from p_0 to p_1, a new active arc $X \rightarrow X_1, \ldots, C \circ X_n$ is added from position p_0 to p_2. Finally, for each active arc $X \rightarrow X_1, \ldots, X_n \circ C$ from position p_0 to p_1, which only requires C to be completed, a new component of type X is added to the agenda from position p_0 to p_1. Figure 9 shows a scheme of the chart-parsing algorithm. It consists of a loop repeated until there is no input left. At each iteration, if the agenda is empty, the lexical categories for the next word of the sentence are added to the agenda. Then a component C is selected from the agenda. Let us assume it goes from position p_1 to p_2. For each grammar rule of the form $X \rightarrow CX_1, \ldots, X_n$, a new active arc $X \rightarrow \circ CX_1, \ldots, X_n$ from p_1 to p_2 is added from position p_1 to p_2. Finally, C is added to the chart by means of the arc extension algorithm.

Best first chart parsing algorithms consider the most likely components first. The main idea is to implement the agenda as a *priority queue*—where the highest rate elements are always first in the queue. Accordingly, the parser always removes the highest ranked component from the agenda and adds it to the chart.

Table 1 compares the results of the BFCP and the bottom-up evolutionary parser (buEP) (average and deviation in 10 runs, where

```
function Arc-Extension(chart, agenda, grammar, C, p1, p2)
    Insert(chart, C, p1, p2);
    for each active arc X → X1, ···, ∘C, Xn from p0 to p1 do {
        AddNewActiveArc(chart, X → X1, ···, C ∘ Xn, p0, p2);
    }
    for each active arc X → X1, ···, Xn ∘ C from p0 to p1 do {
        AddNewComponent(agenda, X, p0, p2);
    }
end
```

FIGURE 8 Arc extension algorithm to add a component from position p_1 to position p_2.

```
function ChartParser(sentence, chart)
    while cont < Length(sentence) do{
    if Empty(agenda) {
       AddInterpretation(sentence[cont]);
       cont++;
    }
    SelectComponent(agenda, C, p1, p2);
    for each grammar rule X → CX1 ··· Xn from p1 to p2 do{
       AddNewActiveArc(chart, X → ∘CX1 ··· Xn, p1, p2);
    }
    ArcExtension(chart, agenda, grammar, C, p1, p2);
end
```

FIGURE 9 Bottom-up chart parsing algorithm.

fluctuations of different runs are within a 1% interval in the Susanne corpus and within a 2% interval in the Penn Treebank). We can observe that in both cases, the buEP improves the results of the BFCP. This is due to the correct parse of some sentences not being the most probable one. In this way, the heuristic constituent of evolutionary algorithms shows its usefulness for parsing. We can observe that the values of precision and recall are slightly worse than the one for accuracy. This is because the parse obtained has a structure similar to the one in the corpus, but a different depth in some subtrees.

We have also studied the impact of the size of the grammar on the algorithm. Table 2 shows the precision, recall, accuracy, and tagging[5]

TABLE 1 Comparison of the Bottom-Up Evolutionary Parser (buEP) with a Best First Chart Parser (BFCP) with Grammars of 800 Rules (Susanne) and 120 Rules (Penn). The Column Labeled buEP Presents the Average Values of the Results Obtained in 10 Runs, Along with the Standard Deviation. *Tag. acc.* Stands for the Accuracy of the Part-of-Speech Tagging Achieved. The buEP Uses a Population Size of 200 and a Maximum Number of Generations of 500. Crossover Rate is 40%, Mutation Rate 10%, Cut Rate 20%, and the Threshold Value to Apply Cut is One-Third of the Length of the Sentence

	Susanne			Penn		
		buEP			buEP	
Measure	BFCP	Mean	SD	BFCP	Mean	SD
Precision	94.52	98.88	0.86	88.66	89.44	1.57
Recall	96.41	98.73	0.61	85.81	87.80	1.74
Accuracy	97.47	99.11	0.99	92.20	94.87	1.07
Tag. acc.	97.30	99.85	0.16	99.22	99.68	0.32

TABLE 2 Results Obtained in the Susanne Corpus for Different Sizes of the Grammar with a Best-First Chart Parser (BFCP) and with the Bottom-Up Evolutionary Parser (buEP), in this Case Average of 10 Runs, and Deviation in Bracket

	225 r.		446 r.		800 r.	
	BFCP	buEP	BFCP	buEP	BFCP	buEP
Precision	99.41	99.28 (0.56)	96.41	99.21 (0.41)	94.52	99.88 (0.86)
Recall	97.14	99.28 (0.56)	97.14	99.16 (0.41)	96.41	98.73 (0.61)
Accuracy	94.65	98.32 (1.01)	94.65	98.08 (0.88)	94.65	99.11 (0.99)
Tag. accuracy	99.61	100 (0.0)	99.61	100 (0.0)	97.30	99.85 (0.16)
Time (s.)	1.15	1.28 (0.14)	2.67	3.11 (0.57)	7.75	8.76 (0.71)

results obtained for grammars of different sizes (average and deviation of 10 runs). We can observe that the results of the evolutionary parser improve those of a classic chart parser. As expected, results for a particular corpus get a bit worse when the size of the grammar is enlarged, since there is a higher degree of indeterminism. The most remarkable point of those data is that the buEP results are very close to 100% in all three measures, while the probabilistic chart parser results are far from this value, again because the correct parse of some sentences is not the most probable one. It may be surprising that the buEP results improve those of the BFCP when the fitness measure is the parse probability. However, we must take into account that in the case of buEP, the composition of the population is constrained by the input sentence and the crossover and mutation operators have a high impact on the kind of individuals generated.

Another result worth noticing is the high accuracy of the part-of-speech tagging obtained.

Studying the Algorithm Design

Different experiments have been conducted in order to investigate the most appropriate definition of the fitness function and genetic operators. Table 3 shows the results for the Penn Treebank obtained with different fitness measures when using conservative crossover, while Table 4 shows the results with the Susanne corpus. The first column in these tables indicates the definition of the fitness function that has been used: (a) for a fitness based on the length of the parsed segment sentence; (b) for the number of nodes in the parse tree; (c) for a probabilistic measure defined as the sum of the logarithm of the probabilities of the grammar rules; (d) for a probabilistic measure defined as the average probability of the grammar rules; (c)∗(a) for a function defined as the product of the measures (c) and (a), and (d)∗(a) for one defined as the product of (d) and (a). The

TABLE 3 Results Obtained in the Penn Treebank with Different Definitions of the Fitness Function (Average and Deviation of 10 Runs). The Population Size is 200 and the Maximum Number of Generations is 100. Crossover Rate is 40%, Mutation Rate 10%, Cut Rate 20%, and the Threshold Value to Apply Cut is One-Third of the Length of the Sentence

Fitness measure	Tag. acc.	Precision	Recall	Accuracy
(a)	97.96 (0.74)	85.13 (1.13)	76.95 (1.31)	90.18 (1.21)
(b)	97.62 (0.45)	78.93 (0.98)	75.39 (1.02)	88.22 (1.29)
(c)	99.78 (0.32)	91.64 (1.57)	90.80 (1.74)	95.87 (1.07)
(d)	98.76 (0.69)	85.78 (0.94)	84.09 (0.92)	92.10 (1.49)
(c) ∗ (a)	96.55 (1.06)	85.44 (1.21)	78.01 (1.04)	89.79 (0.99)
(d) ∗ (a)	95.43 (1.32)	87.91 (1.01)	80.33 (1.33)	91.12 (1.10)

worse results are those obtained with the definition (b), based on the number of nodes, because though this measure leads to build parses of longer segments of sentences, like (a), it also tends to produce larger parse trees, too deep to be correct. We can observe that in both corpora, the best measure is (c), the one defined as the logarithm of the probability of the parse tree, which is a correct definition of probability (see Charniak 1993) of a parse tree for a probabilistic CFG.[6] Accordingly, this has been the fitness function adopted in the remaining experiments. The combination of the probabilistic measures with the measure based on the length of the parsed segment does not improve the results or even spoil them.

We have also studied the most appropriate type of crossover operator, conservative or speculative. Tables 5 and 6 show the results. Though the speculative crossover can provide results as good as the conservative one (see Table 6), the execution time increases significantly with this crossover, since it allows for a higher indeterminism. Therefore, the conservative crossover has been adopted.

TABLE 4 Results Obtained in the Susanne Corpus with Different Definitions of the Fitness Function (Average and Deviation of 10 Runs). The Population Size is 200 and the Maximum Number of Generations is 40. Crossover Rate is 40%, Mutation Rate 10%, Cut Rate 20%, and the Threshold Value to Apply Cut is One-Third of the Length of the Sentence

Fitness measure	Tag. acc.	Precision	Recall	Accuracy
(a)	97.33 (0.54)	95.89 (0.76)	94.89 (0.76)	96.59 (0.88)
(b)	99.03 (0.41)	95.74 (0.83)	93.37 (0.63)	95.67 (0.84)
(c)	99.85 (0.16)	98.88 (0.86)	98.73 (0.61)	99.11 (0.99)
(d)	99.50 (0.23)	96.47 (0.73)	94.99 (0.64)	94.94 (0.86)
(c) ∗ (a)	99.84 (0.61)	96.48 (0.81)	95.01 (0.63)	96.59 (0.91)
(d) ∗ (a)	99.44 (0.24)	96.32 (0.77)	95.91 (0.71)	95.88 (0.94)

TABLE 5 Results Obtained in the Penn Treebank with Different Types of Crossover Operator (Average and Deviation of 10 Runs). The Parameters for the Execution with Speculative Crossover are a Population Size of 600 and the Maximum Number of Generations is 500. Crossover Rate is 40%, Mutation Rate 10%, Cut Rate 20%, and the Threshold Value to Apply Cut is One-Third of the Length of the Sentence. For the Conservative Crossover the Parameters are Population Size of 200 and the Maximum Number of Generations is 500. Crossover Rate is 40%, Mutation Rate 10%, Cut Rate 20%, and the Threshold Value to Apply Cut is Again One-Third of the Length of the Sentence

Cross. type	Tag. acc.	Precision	Recall	Accuracy	Ex. time
Speculative	99.45 (0.50)	85.73 (2.54)	81.75 (1.98)	92.16 (1.84)	46.69 (2.16)
Conservative	99.6 (0.32)	89.44 (1.57)	87.80 (1.74)	94.87 (1.07)	4.13 (0.45)

Studying the Evolutionary Parameters

Some experiments have been carried out in order to determine the most appropriate values for the parameters of the algorithm. The results presented here (average and deviation of 10 runs) corresponds to experiments with Penn Treebank, which is more sensitive to the variations in the parameters. Table 7 shows the results obtained for different sizes of the population. A population size of 150 is the minimum required to reach the complete parse of all the sentences with the chosen crossover rate and the maximum number of generations. With the chosen parameter setting, enlarging the population to more than 200 individuals worsens the results.

Other parameters that have been investigated are the rates of application of the genetic operators. Table 8 shows the results for different rates of the crossover operator, Table 9 for different rates of mutation, and Table 10 for different rates of the cut operator. From Table 8 we can observe that the results improve with the crossover rate until a certain rate

TABLE 6 Results Obtained in the Susanne Corpus with Different Types of Crossover Operator (Average and Deviation of 10 Runs). The Parameters for the Execution with Speculative Crossover are a Population Size of 600 and a Maximum Number of Generations of 500. Crossover Rate is 30%, Mutation Rate 40%, Cut Rate 10%, and the Threshold Value to Apply Cut is One-Third of the Length of the Sentence. For the Conservative Crossover the Parameters are Population Size of 200 and a Maximum Number of Generations of 500. Crossover Rate is 40%, Mutation Rate 10%, Cut Rate 20%, and the Threshold Value to Apply Cut Again One-Third of the Length of the Sentence

Cross. type	Tag. acc.	Precision	Recall	Accuracy	Ex. time
Speculative	99.53 (0.36)	98.92 (0.76)	98.20 (1.30)	98.21 (1.05)	28.11 (5.23)
Conservative	99.85 (0.16)	98.88 (0.86)	98.73 (0.61)	99.11 (0.99)	8.76 (0.71)

TABLE 7 Results Obtained (Average and Deviation) in the Penn Treebank for Different Population Sizes with a Maximum Number of Generations of 500. Crossover Rate is 40%, Mutation Rate 10%, Cut Rate 20%, and the Threshold Value to Apply Cut is One-Third of the Length of the Sentence

Population size	Tag. acc.	Precision	Recall	Accuracy
150	99.68 (0.32)	87.12 (1.49)	85.17 (1.25)	93.16 (1.59)
200	99.68 (0.32)	89.44 (1.57)	87.80 (1.74)	94.87 (1.07)
250	99.16 (0.45)	85.61 (1.66)	84.47 (1.63)	92.34 (1.12)
300	98.96 (0.19)	86.09 (1.45)	84.97 (1.67)	91.06 (1.23)
400	98.14 (0.44)	84.26 (1.78)	83.66 (1.45)	89.96 (1.30)

(40%). Enlarging the crossover rate beyond this value slightly spoils the results. Results also improve with the rate of mutation and the cut operator until a threshold value, as Tables 9 and 10 show. Table 9 indicates that the mutation rate does not affect the results too much, since the applicability of the mutation, which is restricted to complete individuals (the only ones produced with the conservative crossover), is severely limited: it can only exchange a subtree by another one with the same syntactic category, which parses exactly the same sequence of words, and this is not always possible. We can observe that the crossover rate has a higher impact on the results than the ones of mutation or cut. The crossover operator is the most important one in our algorithm, because of the kind of individuals it deals with. Crossover is responsible for extending the parsed segments until completing the parse of the whole sentence and thus, its rate of application has the highest impact on the results.

Another parameter that 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. Table 11 shows the results. The best results are obtained when cut is only applied to individuals that parse a sequence of words longer than one-third of the sentence length.

TABLE 8 Results Obtained (Average and Deviation) in the Penn Treebank for Different Rates of Crossover, with a Mutation Rate of 10%, a Cut Rate of 20%, a Population Size of 200 Individuals, a Maximum Number of Generations of 500, and a Threshold Value to Apply Cut of One-Third of the Length of the Sentence

Crossover rate	Tag. acc.	Precision	Recall	Accuracy
10	98.17 (0.51)	85.41 (1.32)	83.22 (1.44)	91.83 (2.54)
20	98.54 (0.35)	87.83 (1.45)	86.25 (1.76)	91.91 (1.33)
30	99.23 (0.43)	89.13 (1.34)	86.81 (1.62)	93.97 (1.67)
40	99.68 (0.32)	89.44 (1.57)	87.80 (1.74)	94.87 (1.07)
50	99.16 (0.34)	86.26 (1.54)	84.14 (1.51)	92.18 (1.59)
60	99.12 (0.45)	85.96 (1.71)	83.08 (1.70)	92.32 (1.54)

TABLE 9 Results Obtained (Average and Deviation) in the Penn Treebank for Different Rates of Mutation, with a Crossover Rate of 40%, a Cut Rate of 20%, a Population Size of 200 Individuals, a Maximum Number of Generations of 500, and a Threshold Value to Apply Cut of One-Third of the Length of the Sentence

Mut. rate	Tag. acc.	Precision	Recall	Accuracy
0	99.52 (0.31)	86.79 (1.34)	82.88 (1.89)	92.96 (1.31)
5	99.74 (0.25)	86.25 (1.39)	83.46 (1.45)	93.92 (0.93)
10	99.68 (0.32)	89.44 (1.57)	87.80 (1.74)	94.87 (1.07)
20	99.68 (0.32)	87.49 (1.45)	83.34 (1.89)	92.67 (1.32)
30	99.35 (0.42)	86.92 (1.63)	83.04 (1.91)	92.03 (1.13)

TABLE 10 Results Obtained (Average and Deviation) in the Penn Treebank for Different Rates of Cut, with a Crossover Rate of 40%, a Mutation Rate of 10%, a Population Size of 200 Individuals, a Maximum Number of Generations of 500, and a Threshold Value to Apply Cut of One-Third of the Length of the Sentence

Cut rate	Tag. acc.	Precision	Recall	Accuracy
0	99.45 (0.42)	83.59 (1.89)	81.67 (2.13)	90.89 (1.53)
5	99.32 (0.24)	84.62 (1.78)	81.61 (2.04)	90.92 (1.34)
10	99.65 (0.38)	87.91 (1.65)	85.03 (1.89)	91.67 (1.39)
20	99.68 (0.32)	89.44 (1.57)	87.80 (1.74)	94.87 (1.07)
30	97.17 (0.44)	87.21 (1.69)	86.06 (1.81)	92.11 (1.18)

TABLE 11 Results Obtained (Average and Deviation) in the Penn Treebank for Different Threshold Values of the Length of the Sequence of Words Required to Apply Cut, with a Cut Rate of 20%, a Population Size of 200 Individuals, a Maximum Number of Generations of 500, a Crossover Rate of 40%, and a Mutation Rate of 10%

Threshold (cut)	Tag. acc.	Precision	Recall	Accuracy
|s|	98.89 (0.35)	84.63 (2.36)	81.10 (2.00)	90.27 (1.23)
|s|/1.5	99.12 (0.22)	84.91 (1.81)	82.46 (2.21)	90.07 (1.15)
|s|/2.0	99.36 (0.38)	84.45 (2.05)	82.99 (1.74)	93.71 (1.36)
|s|/2.5	99.36 (0.38)	87.05 (1.78)	85.68 (1.85)	94.22 (1.10)
|s|/3.0	99.68 (0.32)	89.44 (1.57)	87.80 (1.74)	94.87 (1.07)
|s|/3.5	99.85 (0.38)	85.97 (1.78)	84.74 (1.62)	92.32 (1.23)
|s|/4.0	99.10 (0.49)	85.34 (1.67)	84.91 (1.83)	90.18 (1.15)

|s| length of the sentence.

Comparison with Other Parsers

This article presents a search method valid for different tagging models, and thus our goal is not to compete with other models. However, in order to give an idea of the quality of the particular model that we have used, as well as the evolutionary search method, we present a comparison with other parsers. Among the different available parsers we have chosen

TABLE 12 Comparison of Results Obtained with the Bikel Parser, Charniak Parser, and the Parser Presented Here (Average and Deviation of 10 Runs), Applied to the Same Set of Sentences, Extracted from Penn Treebank, and Trained with the Same Set of Files from this Corpus

Threshold (cut)	Tag. acc.	Precision	Recall	Accuracy
Bikel parser	96.89	81.46	80.70	97.90
Charniak parser	94.96	82.56	82.58	98.98
Evolutionary parser	99.68 (0.12)	89.44 (1.32)	87.80 (1.67)	94.87 (1.13)

Bikel's (2004) and Charniak's (2000) parsers because they can be trained with a particular set of parsed files, and thus we can use the same set used in our system, making the comparison more fair.

Table 12 compares the results of parsing the same set of sentences extracted from the Brown section of the Penn Treebank, when the system has been trained with the files from the section *cf* of this corpus. Though the underlying statistical model is different in each parser, we can observe that the results are rather similar; even for some of the measures considered, the evolutionary parser outperforms the other parsers. In this way, the evolutionary approach is proven a valid search technique, which can be applied to any proposed parsing model, avoiding the design of particular algorithms for the chosen model. Besides, the methodology developed for evolutionary algorithms, such as very refined genetic operators, and in particular, parallelization models, can be applied to the evolutionary parsers, allowing the improvement of different aspects of the system performance.

CONCLUSIONS

We have performed a detailed study of different alternatives for designing an evolutionary algorithm for bottom-up parsing. This study includes the investigation of different definitions of the fitness function used in the evaluation of the individuals, the use of different genetic operators, and a search of the most appropriate parameters. The information required to apply the evolutionary approaches that have been proposed are the grammar and the lexical categories corresponding to each word. Because this information is automatically obtained from a linguistic corpus, the evolutionary parsers are language independent.

The main conclusion of this work is that evolutionary algorithms provide a valid approach for parsing. They can improve the results of classic parsers based on exhaustive search techniques. For example, best-first chart parsers always produce the most probable parses according to the provided probabilistic grammar. However, the most probable parse

is not always the correct one, according to the corpus. An evolutionary parser however, although biased to select the most probable parses, can also produce others, thus improving the results of the chart parser. In fact, the evolutionary parser can be regarded as a kind of chart parser with a random component. The individuals in the population represent the completed constituent that the chart parser stores to be combined according to the grammar rules. The arc extension algorithm, used by a chart parser to extend a parsed segment of the sentence to a longer one, is represented by the crossover operator, which combines partial parses. The most important difference is the way of selecting the rules to be applied. A best-first parsing algorithm always selects the most likely rule, while the evolutionary parser can select any rule, though those with a higher probability have more chances to be selected.

Another conclusion from the experiments is the advantage of working with partial parses. This allows setting the constraint given by the grammar rules in the generation of new individuals, so that the algorithm works only with valid ones. This highly reduces the search space, what has proven essential for the system to properly work with real text sentences. This representation of the individuals as partial parses fits well with a bottom-up parser, which can begin with the partial parses given by the assignment of lexical tags to the words of the sentence, and then combine them according to the grammar rules.

Experiments on the fitness function have revealed the usefulness of using a probabilistic grammar. The best results are obtained with a fitness function based on the probability of the grammar rules used to build the parse. Specifically, the most appropriate fitness function is the one defined as the sum of the logarithm of the probability of the rules.

Concerning the genetic operators, a more conservative crossover provides best results, particularly on the execution time, than a more speculative one. The conservative crossover, which produces only complete individuals where each category of the applied grammar rule has been satisfied, highly reduces the search space, thus improving the results.

The main limitation of the system is due to the grammars used, which have been directly extracted from a corpus. These grammars have two main problems. On the one hand, their rules are very specific, and in many cases they are only used for parsing a single sentence. When we try to parse a sentence outside the training text, it is common to require rules that have not appeared before, thus impeding the parsing. On the other hand, the large size of such too specific grammars limits the performance of the system. These problems are not specific to the evolutionary parser, but of any parser that uses these grammars. Accordingly, we plan to work in obtaining more appropriate grammars, which allow the evolutionary parser to be applied to any sentence in the language considered.

REFERENCES

Araujo, L. 2002a. A parallel evolutionary algorithm for stochastic natural language parsing. In: *Proc. of the Int. Conf. Parallel Problem Solving from Nature (PPSNVII), Lecture Notes in Computer Science* 2439, pp. 700–709. Heidelberg, Germany: Springer-Verlag.

Araujo, L. 2002b. Part-of-speech tagging with evolutionary algorithms. In: *Proc. of the Int. Conf. on Intelligent Text Processing and Computational Linguistics (CICLing-2002), Lecture Notes in Computer Science* 2276, pp. 230–239. Heidelberg, Germany: Springer-Verlag.

Araujo, L. 2004a. A probabilistic chart parser implemented with an evolutionary algorithm. In: *Proc. of the Int. Conf. on Intelligent Text Processing and Computational Linguistics (CICLing-2004), Lecture Notes in Computer Science* 2945, pp. 81–92. Heidelberg, Germany: Springer-Verlag.

Araujo, L. 2004b. Symbiosis of evolutionary techniques and statistical natural language processing. *IEEE Transactions on Evolutionary Computation* 8(1):14–27.

Belz, A. 1998. Discovering phonotactic finite-state automata by genetic search. In: *Proc. of COLING-ACL '98*, pp. 1472–1474. San Francisco, USA: Morgan Kaufmann.

Bikel, D. M. 2004. A distributional analysis of a lexicalized statistical parsing model. In: *Proc. of the Conf. on Empirical Methods in Natutal Language Processing, EMNLP* 2004, pp. 182–189. Stroudsburg, USA: Association for Computational Linguistics.

Blaheta, D. and E. Charniak. 1999. Automatic compensation for parser figure-of-merit flaws. In: *Proc. of Annual Conference of the Association for Computational Linguistics*, pp. 513–518. Association for Computational Linguistics.

Caraballo, S. and E. Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics* 24(2):275–298.

Charniak, E. 1993. *Statistical Language Learning*. Boston: MIT Press.

Charniak, E. 1996. Tree-bank grammars. In: *Proc. of the Thirteenth National Conference on Artificial Intelligence*, Vol. 2, pp. 1031–1036. Cambridge, USA: AAAI Press/MIT Press.

Charniak, E. 1997. Statistical parsing with a context-free grammar and word statistics. In: *Proc. of the 14th National Conference on Artificial Intelligence*, pp. 598–603. Cambridge, USA: AAAI Press/MIT Press.

Charniak, E. 2000. A maximum-entropy-inspired parser. In: *Proc. of the conf. on North American Chapter of the Association for Computational Linguistics*, pp. 132–139, San Francisco, CA: Morgan Kaufmann Publishers.

Charniak, E. and G. Carroll. 1994. Context-sensitive statistics for improved grammatical language models. In: *AAAI*, pp. 728–733.

Charniak, E., S. Goldwater, and M. Johnson. 1998. Edge-based best-first chart parsing. In: *Proc. of the 6th Workshop for Very Large Corpora*, pp. 127–133. Stroudsburg, USA: Association for Computational Linguistics.

Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In: *Proc. of the Annual Meeting of the Association for Computational Linguistics*, pp. 16–23. Stroudsburg, USA: Association for Computational Linguistics.

Collins, M. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. PhD dissertation, Philadelphia, PA: University of Pennsylvania.

Collins, M. J. 1996. A new statistical parser based on bigram lexical dependencies. In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, eds. A. Joshi and M. Palmer, pp. 184–191, San Francisco: Morgan Kaufmann Publishers.

Davis, M. and T. Dunning. 1996. Query translation using evolutionary programming for multilingual information retrieval II. In: *Proc. of the Fifth Annual Conf. on Evolutionary Programming*. San Diego, USA: Evolutionary Programming Society.

Fogel, L. J. 1962. Autonomous automata. *Ind. Res.* 4:14–19.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Kazakov, D. 1997. Unsupervised learning of naive morphology with genetic algorithms. In: *Workshop Notes of the ECML/MLnet Workshop on Empirical Learning of Natural Language Processing Tasks*, pp. 105–112. Prague, Czech Republic.

Kazakov, D. and S. Manandhar. 2001. Unsupervised learning of word segmentation rules with genetic algorithms and inductive logic programming. *Machine Learning* 43:121–162.

Keller, B. and R. Lutz. 1997. Evolving stochastic context-free grammars from examples using a minimum description length principle. In: *Workshop on Automata Induction, Grammatical Inference and Language Acquisition ICML097*. Int. Conf. on Machine Learning, Nashville, Tennessee, USA.

Klein, D. and C. Manning. 2003a. A* parsing: Fast exact viterbi parse selection. In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL* Feb. 2003. Stroudsburg, USA: Association for Computational Linguistics.

Klein, D. and C. D. Manning. 2003b. Accurate unlexicalized parsing. In: *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pp. 423–430. Stroudsburg, USA: Association for Computational Linguistics.

Kool, A. 1999. Literature survey. Unpublished manuscript, The Netherlands: University of Antwerp.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, USA: MIT Press.

Losee, R. M. 1996. Learning syntactic rules and tags with genetic algorithms for information retrieval and filtering: an empirical basis for grammatical rules. *Information Processing & Management* 32:185–197.

Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The penn treebank. *Computational Linguistics* 19:313–330.

Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs*. 2nd ed. Heidelberg, Germany: Springer-Verlag.

Nettleton, D. J. and R. G. Garigliano. 1994. Evolutionary algorithms for dialogue optimization in the lolita natural language processor. In: *Proc. of the Seminar on Adaptive Computing and Information Processing*, pp. 810–815.

Pinker, S. 1994. *The Language Instinct*. New York: Harper Collins.

Rechenberg, I. 1973. Evolutionsstrategie (Evolutionary Strategies), Technical Report. Stuttgart, Germany: Frommann-Holzboog.

Rose, C. P. 1999. A genetic programming approach for robust language interpretation. In: *Advances in Genetic Programming 3*, eds. L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, pp. 67–88. Cambridge, USA: MIT Press.

Sampson, G. 1995. *English for the Computer*. Oxford: Clarendon Press.

Schwefel, H. P. 1975. Evolutionary strategies and numerical optimization disertation. Technical Report, Berlin, Germany: Technische Universität.

Sleator, D. D. and D. Temperley. 1993. Parsing English with a link grammar. In: *Third International Workshop on Parsing Technologies*, pp. 277–292.

Smith, T. C. and I. H. Witten. 1995. A genetic algorithm for the induction of natural language grammars. In: *Proc. IJCAI-95 Workshop on New Approaches to Learning Natural Language*, pp. 17–24. Montreal, Quebec, Canada.

Wyard, P. 1991. Context free grammar induction using genetic algorithms. In: *Proc. of the 4th Int. Conf. on Genetic Algorithms*, pp. 514–518. San Francisco: Morgan Kaufmann,

NOTES

1. 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 highest probability constituents first.
2. By elitism, we refer to the technique of retaining in the population the best individuals found so far.
3. In order to simplify the process, those sentences that 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
$$P(r) = \frac{\#r}{\sum_{r'=A\rightarrow\cdots} \#r'}$$
where $\#r$ is the number of occurrences of r.
5. Rate of words that have been assigned the correct P05 tag.
6. The probability of each parse is the product of the probabilities of all the rules used in the parse tree. The probability of a sentence in a PCFG is the sum of the probabilities of all possible parses for the sentence. Then, the probabilities of all the sentences generated by the grammar add up to one.