Symbiosis of Evolutionary Techniques and Statistical Natural Language Processing

Lourdes Araujo

Dpto. Sistemas Informáticos y Programación, Univ. Complutense, Madrid 28040, SPAIN (email: lurdes@sip.ucm.es)

Abstract

This work presents some applications of Evolutionary Programming to different tasks of Natural Language Processing (NLP). First of all, the work defines a general scheme of application of evolutionary techniques to NLP, which gives the mainstream for the design of the elements of the algorithm. This scheme largely relies on the success of probabilistic approaches to NLP. Secondly, the scheme has been illustrated with two fundamental applications in NLP: tagging, i.e. the assignment of lexical categories to words, and parsing, i.e. the determination of the syntactic structure of sentences. In both cases, the elements of the evolutionary algorithm are described in detail, as well as the results of different experiments carried out to show the viability of this evolutionary approach to deal with tasks as complex as those of NLP.

Index Terms

Evolutionary programming, natural language processing, lexical tagging, parsing, probabilistic methods

I. Introduction

Natural Language Processing (NLP) includes a large amount of complex tasks such as the determination of the lexical tag which corresponds to each word in a text (tagging), the parsing of sentences, the determination of the antecedent of pronouns and relative clauses (anaphora resolution), the determination of words belonging to an expression, etc. Features such as ambiguity make these tasks complex search processes, too complex in fact to be tackled with classic search algorithms in a reasonable time. For instance, the parsing of a sentence is a search problem that requires exploring a tree of possible parses. The size of this tree increases exponentially with the length of the sentence or text to be parsed. This complexity suggests applying heuristic techniques to speed up the search. The price to pay is that we are no more certain about the correctness of the solution found, although good heuristic methods allow to systematically decrease the probability of error (for instance by increasing the number of points explored). Evolutionary Algorithms (EAs) are among such heuristic techniques; they are stochastic algorithms based on a search model which emulates evolutionary phenomena.

An important aspect concomitant with the introduction of statistical techniques in text processing, is that it transforms NLP tasks into optimization processes, for which evolutionary techniques (already applied to areas such as planning or machine learning) have proven to be very useful [29]. Other optimization techniques have been also applied to NLP, like e.g. neural networks [5], [30], [32]. In the last decades, statistical methods have become a fundamental approach to computational linguistics, bringing significant advances in issues such as disambiguation, error correction or grammar induction. Applied to NLP, these methods rest on the assumption that we can extract knowledge about a language (that is, on its structure and semantics) by defining a general model with free parameters, whose values are then determined by statistical techniques. Statistical measurements are extracted from an annotated corpus [27], i.e., a set of texts linguistically marked-up. Probabilistic approaches have already been successfully used to deal with a large number of NLP problems. Research on these methods has grown a lot in the last years, probably due to the increasing availability of large tagged corpora.

One NLP task for which statistical methods have been very successfully applied is lexical tagging, i.e. the disambiguation in the assignment of an appropriate lexical tag to each word in a text [15]. There are different proposals to automatically perform this disambiguation, all of which use a large amount of data to establish the probabilities of each situation. Most of these systems are based on Hidden Markov models and variants [21], [18], [17], [15], [34], [28], [4] and neither require knowledge of the rules of the language nor try to deduce them. There are also rule-based approaches [31], [9], [10] which apply language rules, also extracted from an annotated corpus, to improve the accuracy of the tagging. The accuracy of both approaches is so far comparable.

Another task carried out by statistical techniques [11], [33] is word sense disambiguation. It consists in determining which of the senses of one ambiguous word is referred to in a particular occurrence of the word in a text. It differs from tagging in that it uses semantic tags instead of lexical tags. These two issues have to be differentiated because nearby structures are more influential in lexical tagging, while relatively distant words are more useful for word sense disambiguation.

Parsing is a preceding step for many NLP tasks. In most cases, there are many different parses for the same sequence of words, that is, syntactic ambiguity occurs very often. Stochastic grammars [7], [15], [8], [16], [2] represent another important part of the statistical methods in computational linguistics. These grammars give us an evaluation of the tree representing a possible parse of a sentence. The goal is then to find one of the best parses according to this measurement.

Machine translation, i.e. automatic translation from one language to another, is one of the most important applications of NLP. A previous fundamental step to perform machine translation is text alignment, that is, establishing the correspondence between paragraphs, sentences or words of parallel texts in different languages. A number of publications deal with statistical alignment [19], [12], [36]. There are also some other approaches on statistical machine translation [12], [6]. And there still remains a large amount of topics of NLP to which statistical techniques have been applied, such as different aspects of information retrieval or word clustering.

The kind of non-exhaustive search performed by EAs makes them suitable to be combined with probabilistic techniques in NLP. The purpose of most EAs is to find a good solution and not necessarily the best solution, and this is enough for most natural languages statistical processes. EAs provide at the same time a reasonable accuracy as well as a unique scheme of algorithm when applied to different problems. The symbiosis between EAs and statistical NLP come from the simplicity to develop systems for statistical NLP using EAs —because EAs provide a unified treatment of different applications— and from the fact that the more trickiest elements of the EA, such as the fitness function or the adjustment of the parameters, are highly simplified by resorting to the statistical NLP models. EAs have already been applied to some issues of natural language processing [23], such as query translation [25], inference of context-free grammars [37], [35], [24], [22], tagging [4] and parsing [2]. The study of these approaches enable us to observe patterns of similarity in the application of these techniques.

This work aims to show the natural symbiosis of EAs and Statistical NLP. The statistical measurements extracted by the stochastic approaches to NLP provide in a natural way an appropriate fitness function for EAs. Furthermore, the annotated corpus or training texts required for the application of statistical methods also provide a nice benchmark for fine-tuning the algorithm parameters (population size, crossover and mutations rates, etc; see below), a fundamental issue for the behaviour of the algorithm. Accordingly, this work presents a general framework for the application of evolutionary techniques to statistical NLP. The mainstream for the definitions of the different elements of an evolutionary algorithm to solve some statistical NLP tasks are presented. These ideas are exemplified by means of particular algorithms for two central problems in NLP: tagging and parsing.

The rest of the paper proceeds as follows: Section 2 describes the generic structure of an EA for NLP; section 3 applies this scheme to tagging, and includes experimental results; section 4 presents an evolutionary parser for natural language, including a parallel version and experimental results for each version; finally, section 5 summarizes the main conclusions of this work.

II. Evolutionary Algorithms and Natural Language Processing

An EA is basically a sequence of generations (iterations), each of which produces a new population of individuals out of the previous one. Each individual represents a potential solution to the problem, whose "fitness" (a quantification of "closeness" to an exact solution) can be measured. The population of a new generation is made of the survivors from the previous generation, as well as of individuals resulting from the application of "genetic" operators to members of the previous population, randomly selected with a probability proportional to their fitness. After a number of generations, the population is expected to contain some individuals representing a near-optimum solution. Thus, EAs perform a random search, potentially able to reach any region in the search space because of the random selection. However, at the same time they present a marked tendency to keep the most promising points for possible improvements, because the fittest individuals have a higher probability to survive. EAs can also be designed as learning systems in which the evaluation of the individuals can be defined in such a way that the AE is trained to solved a particular problem. In this case, the system gives the best fitness value to a number of solved situations that are provided and then uses this information to evaluate new inputs.

Different EAs can be formulated for a particular problem. Such programs may differ in many ways: the representation of individuals, the genetic operators, the methods for creating the initial population, the parameters, etc. The approach adopted in this paper is based on the evolutionary programming method, that considers any set of data structures for the representation of individuals [29]. Thus, it differs from classical genetic algorithms, which use fixed-length binary strings as the representation of individuals.

An evolutionary program for a particular problem requires the following elements:

- A representation of the potential solutions to the problem, i.e., of the individuals of the population.
- A method for producing an initial set of individuals, which will represent the initial population.
- An evaluation function to compute the *fitness* of an individual. This function indicates "how suitable" is an individual to be a solution to the problem at hand according to a training set.
- Genetic operators to modify the individuals of a population and to produce new ones for the next generation.
- Values for different parameters which characterize the algorithm: population size, survival probability to the next generation, rates of application of the genetic operators, etc.

Now, let us consider the similarities and differences between EAs applied to different tasks of Statistical NLP. It is clear that the tasks of NLP are of a very different nature, and they require different structures to represent the data to which to apply the computational process (although some of the structures, such as the parse trees, are common to many of them). The most relevant similarity is the use of the measurements provided by the statistical model in the definition of a fitness function. This function guides the search of the algorithm and strongly determines its performance. At the same time it logically represents one of the greatest difficulties in the design of an EA. Hence, the great advantage of combining of these techniques: EAs provide their generality to the complex search processes of NLP, and the development of the system is highly simplify by using the statistical models of NLP to define the most delicate elements of the algorithm. Furthermore, the training text used to provide the statistical measurements required by the statistical model of NLP, also provides a nice benchmark to tune the parameters of the algorithm, another critical point to produce efficient EAs.

According to the previous considerations, we can define a general framework or *NLP-EA scheme* to deal with NLP tasks:

- **Individuals Representation**

  It must be a natural representation of the solutions to the particular problem at hand. Thus, in tagging for instance, individuals can be represented by sequences of tags assigned to each word in the sentence to be tagged. In parsing, individuals can be trees, one of the most extended representations of a parse.
- **Generations and Genetic Operators**

  In each generation some individuals from the current population are applied genetic operators to produce new individuals and renew the population. The genetic operators used herein are *crossover*, which combines two individuals to generate a new one, and *mutation*, which creates a new individual by changing a randomly selected gene in an individual of the population. The genetic operators must be designed in such a way as to maintain a delicate equilibrium between the inheritance of the ancestors' properties and the exploration of new areas of the search space. Their particular designs strongly depend on the representation chosen for the individuals.
- **Fitness Function: the training text**

  The definition of the fitness function is provided by the statistical model of the NLP task considered. It is a function whose maximum is the most likely solution to the problem. This function is evaluated on the data provided by a previously processed corpus. That is, we need to have a corpus available for which the particular task we are dealing with has already been achieved: tagging requires a hand-tagged text, parsing a hand-parsed text or tree-bank, and so on. This corpus is then used as a *training text*. For example, in the tagging problem, if we consider contexts without right-hand side —which reduces the model to a Markov chain— the fitness function would be the function whose maximum is achieved by the most likely path in a Markov chain, and would be evaluated according to the frequencies of the contexts in the training text. However, we also want to consider possible representations of the individuals which are not feasible solutions to the problem. For example, in the tagging problem, if we consider individuals which are assignments of a sequence of tags to the sentence where each tag is chosen among the valid tags of the corresponding word, any individual will be a valid solution. Therefore, the fitness function is only a measurement of the probability of the sequence of tags chosen. However, in the parsing problem, if we take individuals which are trees containing components that do not correspond to the tags of the words, there will appear individuals which are not valid parses of the sentence. Thus the fitness function must include both, a measure of the feasibility of the individual as well as a measure of its probability. In essence, the steps to obtain the fitness function are:

  - Defining the relations between words, positions, lexical or syntactic tags, etc., which determine the solution to the problem at hand. In the tagging problem, this relation refers to the words (and their tags) in the neighbourhood of the one to be tagged (the so called *context* of the word). In the parsing problem, this relation is given by the grammar rules used to build the parse.
  - Extracting statistics about those relationships from the training corpus. In the case of tagging, we will compute the probability of the different contexts appearing in the training text. In the case of parsing, we will compute the probability of the grammar rules. It is obvious that the larger the corpus the better these estimated probabilities. The size of the corpus is strongly influenced by the complexity of the relationships defined in the first step (conversely, the complexity of those relationships is limited by the size of the available corpus).
  - Defining a measure of the probability of an individual according to the model provided by the statistics.
  - If the representation chosen allows non feasible individuals, then we will also have to define a measure of the feasibility and define the fitness function as a combination of both measures.

- **EA parameters: the test text**

  In EAs there is always a trade-off between two fundamental factors in the evolution process: population diversity and selective pressure. An increase in the selective pressure, which decreases the diversity of the population, can lead to a premature convergence of the EA, while a weak selective pressure can made the search ineffective. The relation between these factors is mainly determined by the following EA parameters:

  - Population Size

    It is clear that population diversity and selection pressure are related to the size of the population, which is one of the most relevant parameters of an EA. If this size is too small the population will quickly uniformize and the algorithm usually converges to a bad result; but if it is too large, the algorithm will take too long to converge due to the higher diversity of individuals.
  - Rates of crossover and mutation

    These parameters, also critical for the effectiveness of the algorithm, must be correlated with the population size to provide the EA with the suitable population diversity.

  The most appropriate values for these parameters are strongly problem dependent. Again, the statistical approach to NLP provides an automatic mechanism for tuning these parameters by training the algorithm with a new test text —not included in the training text (Figure 1). The *training text* provides the data used by the *evolutionary algorithm* to compute the fitness function. The parameters of this algorithm are initially set to some particular values. Then the algorithm is applied to the new *test text* producing an *annotated text* which is compared with the correct answer. The accuracy obtained is then fed back to the evolutionary algorithm to modify its parameters. The process is iterated until finding the parameters which achieve the best accuracy.

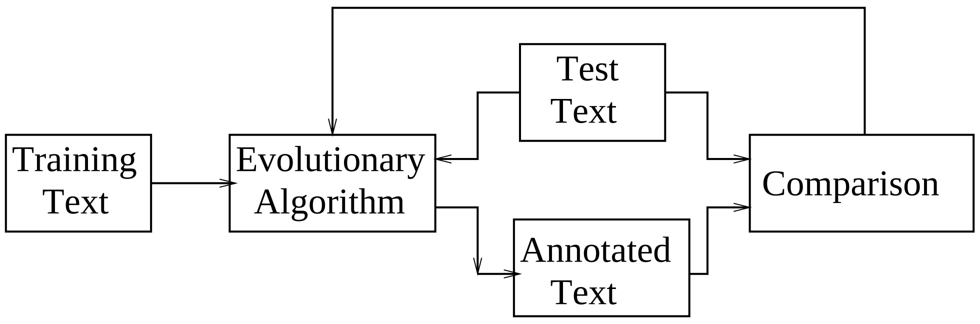

Fig. 1. Evolutionary algorithm operating scheme.

In what follows we will apply this NLP-EA scheme to two of the fundamental tasks of NLP: tagging and parsing. Individuals are represented in the most usual way for the corresponding problem in NLP. Classic taggers give the results as a sequence of tags for the words of the sentence to be tagged; hence the representation chosen will be tag sequences. In classic parsing, parses are usually given as a parse tree, and so this has been the representation chosen. The genetic operators have been defined to suit the representation. The approach of this work starts from the assumption that there is a statistical model for the task considered.

III. TAGGING

Usually, the first step in the parsing process is to identify which lexical category (noun, verb, etc) each word in the sentence belongs to, i.e to tag the sentence. The difficulty of the tagging process comes from the lexical ambiguity of the words: many words can be in more that one lexical class. The assignment of a tag to a word depends on the assignments to the other words. Let us considered the following words and their tags.

Rice:	NOUN
flies:	NOUN, VERB
like:	PREP, VERB
sand:	NOUN

The choice for the word *like* depends on the choice for the word *flies* and vice versa. The syntactic structure of the sentence depends on its tagging, as Figure 2 shows, and this structure is fundamental to determine its meaning, a crucial requirement in very different applications, from machine translation to information recover. The complexity of the problem increases with the length of the sentence to be tagged, and thus lexical tagging can be sought as a *search process* that looks for a correct assignment of lexical tag to every word in a sentence.

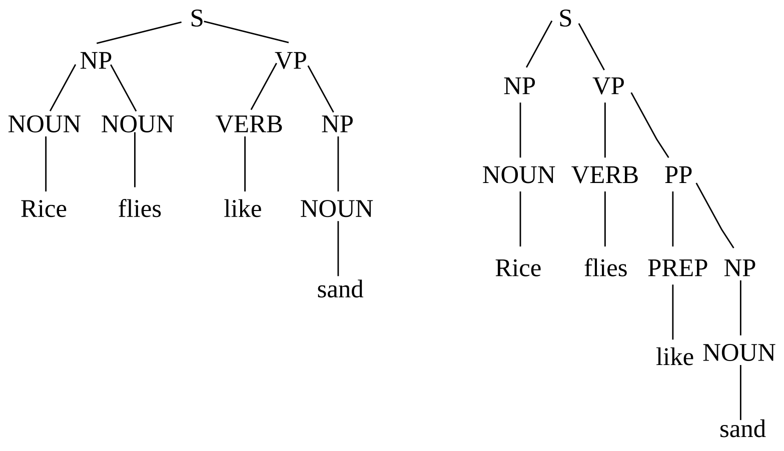

Fig. 2. Different parses for the sentence *Rice flies like sand*.

The most common statistical models for tagging are based on assigning a probability to a given tag according to its neighboring tags (*context*). Then the tagger tries to maximize the total probability of the tagging of each sentence. This is usually done with a Hidden Markov Model (HMM) [15]. A HMM is a finite-state automaton in which a given node may have several transitions out of it, all with the same symbol. These transitions are assigned certain probabilities. In the tagging problem each node is a different assignment of tag to word for every word in the sentence. This representation is only accurate under the Markov assumption, that is, if the probability of a lexical category only depends on the category before it. To find the most likely sequence of tags for a sentence we can look for the most likely path in the HMM, which is traditionally computed by means of the Viterbi Algorithm [15].

If only a number of preceding tags are considered as context, the model is equivalent to a Markov chain; however, if the context includes successive tags, the model is no more a Markov chain, but a more complex process in which the tag of surrounding words influence the tag of a word and the former are in turn influenced by the latter.

Now, according to our scheme to define an EA, we can design an algorithm which works with individuals consisting of sequences of tags assigned to each word in the sentence, because this is the form of the solutions we expect to obtain. Furthermore, with this representation the position of the sentence to which each tag corresponds does not need to be explicitely coded. Let us consider the sentence of a previous example *Rice flies like sand*. Then two possible individuals when tagging this sentence are shown in Figure 3.

NOUN	NOUN	VERB	NOUN

NOUN	VERB	VERB	NOUN

Fig. 3. Examples of individuals for the sentence *Rice flies like sand*.

With this representation, genetic operators of crossover and mutation are easily defined as the exchange of pieces of the sequence between individuals, and as the alteration of a particular tag, respectively. Furthermore, the probabilistic model, which assigns a probability to a sequence of tags according to the context of each word in the sentence, provides the fitness function, and thus we have the basic elements of the algorithm.

The probabilistic model requires to generate a table of context frequency (*training table*) from a training tagged corpus. This training table will be used to evaluate individuals and to determine their fitness. The table is constructed by recording every context of each tag along with the number of its occurrences in the training text. This allows to estimate the probability of all contexts of a tag. The entry in the training table corresponding to tag *T* has the following form: it begins by a line

T #C #T

where #C stands for the number of different contexts of tag T and #T stands for the number of occurrences of tag T in the whole corpus. This is followed by a line for each of those contexts of the form (l_{LC} denotes the length of the context to the left of the tag and l_{RC} that of the context to the right of the tag and both are fixed for each run):

l_{LC} tags to the left T l_{RC} tags to the right # occurrences

For example, if $l_{LC} = l_{RC}$ = 2, the entry in the table for tag *JJ* could have the form:

```
JJ 4557 9519

VBD AT JJ NN IN 37
IN PP$ JJ NNS NULL 20
PPS BEZ JJ TO VB 18
NN IN JJ NN WDT 3
       ...
```

denoting that *JJ* has 4557 different contexts and appears 9519 times in the text, and that in one of those contexts, which appears 37 times, *JJ* is preceded by tags *VBD* and *AT* and succeeded by *NN* and *IN*, and so on until all 4557 different contexts have been listed. The contexts corresponding to the position at the beginning and the end of the sentences lack tags on the left-hand side and on the right-hand side respectively. This event is managed by introducing a special tag, NULL, to complete the context. In the construction of the training table, the contexts corresponding to the ends of the sentences are also completed with the NULL mark. In this way, the tags at the ends of the sentences fit in the general scheme.

A. Evolutionary Tagging

The evolution process is run for each sentence in the text to be tagged. Evolution aims to maximize the total probability of the tagging of the sentences in the test corpus. The process finishes either when the fitness deviation lies below a threshold value (convergence) or when the evolutionary process has been running for a maximum number of generations, whatever occurs first. Let us analyze in detail each of the elements of the algorithm.

1) Individual Representation: Individuals are sequences of genes, each consisting of the tag of a word in the sentence to be tagged plus some additional information useful in the evaluation (such as counts of different contexts for this tag according to the training table).

2) Initial Population: For a given sentence of a test corpus, the initial population is composed of a number of individuals constructed by taking from a dictionary one of the valid tags for each word, selected with a probability proportional to their frequencies.

Words not listed in the dictionary are assigned the tag which appears more often with that given context in the training text, provided there are no other unknown words in the context. Since the number of unknown words is very small, this is what usually happens. Otherwise the word is assigned a randomly chosen tag.

3) Fitness: Individual Evaluation: In this case, this function is related to the total probability of the sequence of tags of an individual. The raw data to obtain this probability are extracted from the training table. The fitness of an individual is defined as the sum of the fitness of its genes ($\sum_i (f(g_i))$). The fitness of a gene is defined as

$$f(g) = \log P(T|LC, RC)$$

where $P(T|LC, RC)$ is the probability that the tag of gene g is T, given that its context is formed by the sequence of tags LC to the left and the sequence RC to the right. This probability is estimated from the training table as

$$P(T|LC, RC) \approx \frac{occ(LC, T, RC)}{\sum_{T' \in \mathcal{T}} occ(LC, T', RC)}$$

where $occ(LC, T, RC)$ is the number of occurrences of the list of tags LC, T, RC in the training table and $\mathcal{T}$ is the set of all possible tags of g_i. For example, if we are evaluating the first individual of Figure 3 and we are considering contexts composed of one tag on the left and one tag on the right of the position evaluated, the first gene, for which there is only one tag, will be evaluated as

$$\frac{\#(\text{NULL NOUN NOUN})}{\#(\text{NULL NOUN NOUN})} = 1$$

where $\#$ represents the number of occurrences of the context. The second gene, for which there are two possible tags, NOUN (the one chosen in this individual) and VERB, will be evaluated as:

$$\frac{\#(\text{NOUN NOUN VERB})}{[\#(\text{NOUN NOUN VERB}) + \#(\text{NOUN VERB VERB})]}$$

The remaining genes are evaluated in the same manner.

A special situation needs to be handled separately. It may happen that a particular sequence LC, T, RC is not listed in the training table. This may be so because its probability is strictly zero (if the sequence of tags is forbidden for some reason) or, most likely, because there are insufficient statistics (it is easy to realize that even short contexts exhibit such a huge number of possibilities that gigantic corpora would be needed in order to have reliable statistics of every possibility). When this occurs we proceed this way. If RC is not empty, we ignore the rightmost tag of the context and consider the new (shorter) sequence. We seek in the training table for all sequences matching this shorter context and take for the number of occurrences the sum of the number of occurrences of the new context. If RC is empty or there are no sequences matching the shorter one, we carry on ignoring the leftmost tag of the context (provided LC is not empty). We repeat the search with this even shorter sequence. The process continues alternatively ignoring the rightmost and then the leftmost tag of the remaining sequence (skipping the corresponding step whenever either RC or LC are empty) until one of these shorter sequences matches at least one of the training table entries or until we are left simply with T. In this latter case we take for $F(i)$ the logarithm of the frequency with which T appears in the corpus (also listed in the training table).

The whole population is evaluated according to this procedure every generation, and the average fitness of the population is also computed.

4) Genetic Operators: Because of the representation of the individuals, genetic operators have in this case a straightforward design. To apply *crossover,* two individuals are selected with a probability proportional to their fitness. Then a crossover point is randomly selected, thus dividing both individuals in two parts. In the selection of this partition point, positions corresponding to genes with low fitness are preferred. The first part of one parent is combined with the second part of the other parent thus producing two offsprings.

Mutation is then applied to every gene of the individuals resulting from the crossover operation with a probability given by the mutation rate. The tag of the mutation point is replaced by another of the valid tags of the corresponding word. The new tag is randomly chosen according to its probability (the frequency it appears in the corpus).

Individuals resulting from the application of genetic operators replace an equal number of individuals, selected with a probability inversely proportional to their fitness. The number of individuals that remain unchanged in one generation depends on the crossover and mutation rates.

B. Tuning the Model

The evolutionary algorithm has also been improved mainly by adjusting some parameters along the evolution:

- *Fitness scaling:*

  It is very common that the first generations of individuals present some extraordinary individuals among a mass of average ones. With the basic selection rule, these extraordinary individuals would take soon a significant portion of the population, thus leading to a premature convergence. On the contrary, after some generations, we can obtain a population of individuals of very similar fitness. In this case, average and best individuals have similar opportunities of surviving and the evolutionary process would have a random result. In both cases, fitness scaling can be useful. A linear scaling function $f' = Cf + b$ has been applied. C is a scaling factor that takes the value 1.5 when the difference between the maximum and average fitness values is less than 25% of the average fitness, and 0.5 otherwise. b is defined in such a way that the average values of the fitness with and without scaling are the same, i.e. $b = f_{av}(1 - C)$ where f_{av} is the fitness average.
- *Variable Crossover and Mutation rates*:

  The rates of crossover and mutation can vary along the evolution process, in such a way that they decrease with the number of generations.
- *Preservation of the best individual*:

  The best solution up to the moment is saved.

C. Experimental Results

The corpus used to train the tagger has a huge influence on the performance. It must be a good sample of the language and it must be as domain-specific as possible, in order to present similar constructions. In this work we used the *Brown* corpus. The tag set is not too large, what favors the accuracy of the system, but at the same time it is large enough to make the system useful.

Different experiments have been carried out in order to study the main factors affecting the accuracy of the tagging: the size and shape of the contexts used for the training table, the influence of the size of the training corpus and the evolutionary parameters.

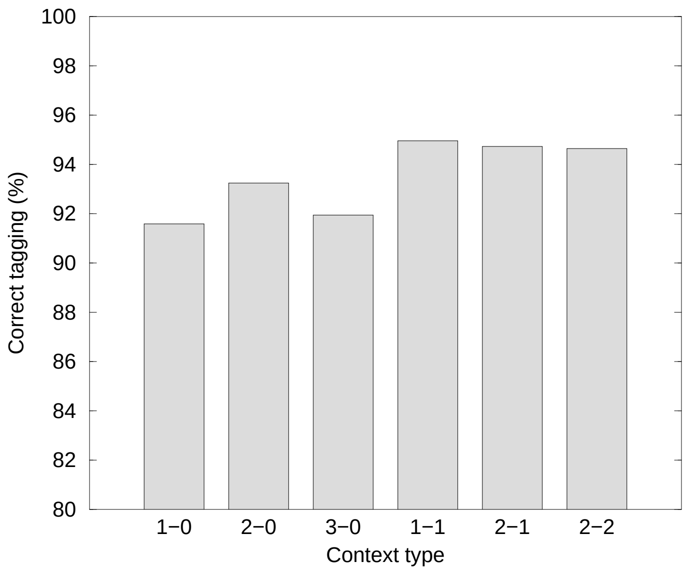

Fig. 4. Accuracy rate obtained for different sizes of the context, using a training corpus of 185000 words, a test text of 2500 words, a population size of 20 individuals, a crossover rate of 50%, and a mutation rate of 5 %.

1) Influence of the amount of context information: The way in which the context information is used by the tagger affects its performance. This information can be fixed or variable and have different sizes. A statistical analysis of the Brown corpus allows to determine the maximum length for which the contexts are meaningful. This is done by checking correlation between tags, that is, for a given tag X_0 we have computed $P(X_d|X_0)$ for different values of d, the distance (in number of tags) between X_d and X_0. From those data, we can determine the smallest value of d for which X_d and X_0 are statistically independent, i.e.

$$P(X_d|X_0)/P(X_d) \approx 1.$$

This analysis has shown that $d = 3$ is a safe value and in most cases $d = 1$ is enough. Therefore, contexts longer than 3 are irrelevant.

Figure 4 shows the accuracy rates reached with different context sizes and shapes (always shorter than $d = 3$). Results show that the best performance is reached for small context sizes, such as 1-1, probably due to the fact that for larger contexts the number of occurrences of many entries of the training table is not statistically significant.

Another remarkable feature is that left-handed contexts perform slightly worse than symmetric ones. This suggests that models more complex than Markov chains might capture better the structure of language.

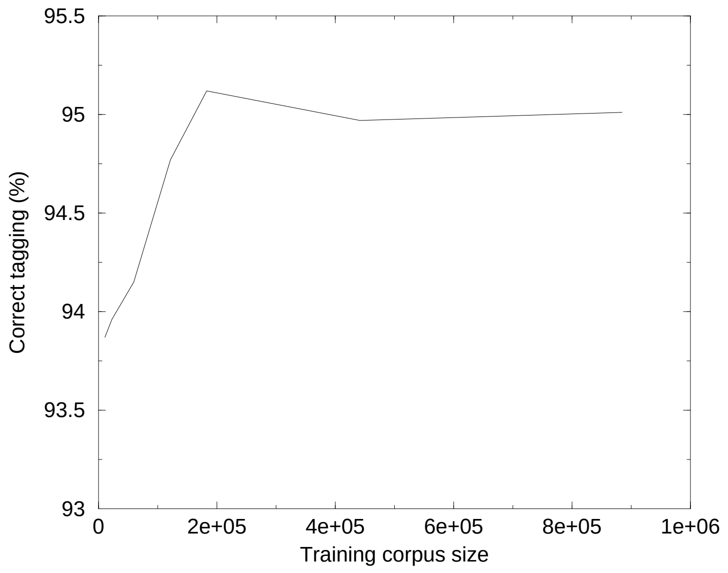

Fig. 5. Accuracy rate reached with different sizes of the training corpus, using contexts of the form 1-1, a test text of 2500 words, a population size of 20 individuals, a crossover rate of 50%, and a mutation rate of 5 %.

2) Influence of the size of the training text: The next step has been the study of the influence of the size of the training text. For a given context pattern, increasing the size of the corpus increases the quality of the statistical data and hence the accuracy of the algorithm. But this saturates when the maximum accuracy for that context is reached, so further increasing the size does not improve the algorithm. This is the optimal size for that context. Enlarging the context will again require larger training text due to the proliferation of new patterns, but this has a drawback in that the training table also grows, which slows down the algorithm. So going to larger contexts is only justified if the increase in accuracy is significant. Figure 5 shows the increase of the accuracy with the size of the training corpus, and clearly illustrates the saturation occurring beyond a certain size (around 200000 words in this case).

3) Study of the Evolutionary Algorithm parameters: We have also investigated the parameters of the evolutionary algorithm: population size and crossover and mutation rates. Figure 6 shows the results obtained. We can observe that small populations are sufficient to obtain high accuracy rates, because the sentences are tagged one by one and in general a small population is enough to represent the variety of possible taggings. This leads to a quicker algorithm. Crossover and mutation rates must be in correspondence with the population size: the larger the population, the higher the rates required. It is therefore not only unnecessary but also inconvenient to increase the population.

4) Comparison with other systems: The best Hidden Markov models typically perform at about a level of correctness of the 95% [15]. Brill's model [10], based on transformation rules, with a training text of 120,000

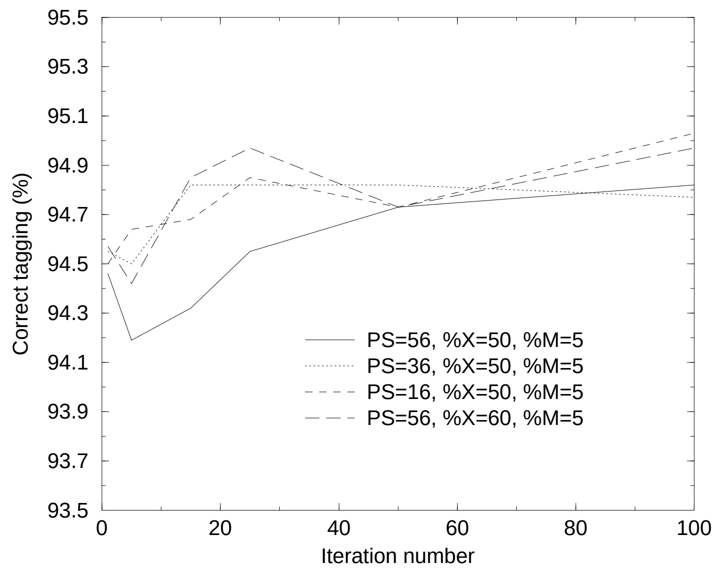

Fig. 6. Accuracy rate reached as a function of the number of generations. PS stands for the population size, %X for the crossover rate, and %M for the mutation rate.

words an a separate test text of 200,000 words, obtained a tagging accuracy of 95.6%, which increased up to 96.0% by expanding the training set to 350,000 words. Therefore our results are comparable to those of other probabilistic and rule-based approaches, in spite that the systems using those approaches have been specifically designed for this problem. Furthermore, in this case the algorithm turns out to be particularly fast because the best tagging can be reached with small populations and just a few generations.

A study of the most frequent errors in the tagging has revealed the following points:

- As expected, words that require a tag that is not the most frequent or that appears in an odd context tend to be tagged incorrectly.
- In general, the longer the sentence to be tagged, the better the results, because in large sentences there will be enough contexts to compensate the weight of some erroneous taggings.
- To decide the size of the training corpus we must take into account that in tagging sentences whose words require one of its frequent tags and that appear in a frequent context, the longer the training text, the more accurate the tagging. However, when tagging sentences with words that adopt some of their most rare tags, the length of the training corpus can spoil the results.
- Another observation is the correlation between the parameters of the algorithm and the complexity (number of ambiguous words) of the texts to be analyzed. The more complex the text, the larger the population size required to quickly reach a correct tagging.
- Furthermore, as the population size increases, higher rates of crossover and mutation are required to maintain the efficiency of the algorithm.

IV. Evolutionary Parsing

Let us now consider the process of looking for the structure of a sentence according to a particular grammar, i.e. parsing. Parsing is an extremely important step for both recognition and generation of natural language. It is required in different applications such as the extraction of information from documents, machine translation (which has to find the correspondence between the structure of sentence in two languages) or the generation of answers when consulting a database. Because of syntactic ambiguity, parsing is a complex problem that has been tackled with probabilistic methods. Among these probabilistic approaches are Probabilistic Context Free Grammars (PCFGs), the ones considered in this work. In a PCFG each grammar rule is assigned a probability proportional to the frequency with which the rule appears in the training text in such a way that the probabilities for all the rules that expand the same non-terminal add up to one. The capabilities of the parser depend on the probabilistic grammar it uses. The probabilistic grammar is extracted from a tree-bank, i.e. a text with syntactic annotations. The test set is formed by the sentences to be parsed, which can be also extracted from a corpus.

Traditional parsers [1] try to find grammar rules to complete a parse tree for the sentence. For instance, the main basic operation in a bottom-up parser is to take a sequence of symbols and match them to the right hand side of the grammar rules. Accordingly, the parser can be simply formulated as a search process of the matching operation. These parsers can be easily modified to take into account the probabilities of the rules. The main idea of a *best-first* parser is to consider the most likely constituents first.

According to our scheme of construction of the EA, the population consists of potential parses for the sentence and grammar considered. These parses are represented in the classical way, i.e. as trees. Genetic operators are then operations over these trees, which exchange or modify some of their subtrees.

We define a fitness function based on a measurement of the “distance” between the individual to evaluate and a “feasible” and probable parse for the sentence. The feasibility of the sentence is measured by the number of grammar rules that have been correctly applied in the parsing. The probability of the parse is given by the product of the probabilities of its grammar rules.

The complexity of this process makes a parallel approach [3] particularly suitable to speed up the convergence of the algorithm. This parallelization can be included in the NLP-EA scheme for all problems whose complexity requires large populations and number of generations to obtain results of quality.

Next, the elements of the algorithm are described, along with as some experiments and the parallel implementation.

A. Individual Representation

The individuals of our evolutionary algorithm represent potential parses for an input sentence according to a PCFG. Figure 7 shows some rules and probabilities of the PCFG used in the experiments. The input sentence is given as a sequence of words whose set of possible categories is obtained from a dictionary or “lexicon” in a pre-processing step. Let us consider a simple running example. For the sentence “the man sings a song” we can obtain

Rule	Prob.	Rule	Prob.
S $\rightarrow$ NP VP	1.		
NP $\rightarrow$ Noun	0.1	VP $\rightarrow$ Verb	0.05
NP $\rightarrow$ Pronoun	0.1	VP $\rightarrow$ Verb, AP	0.15
NP $\rightarrow$ Det, NP	0.2	VP $\rightarrow$ Verb, NP	0.3
NP $\rightarrow$ NP, PP	0.1	...	...
NP $\rightarrow$ NP, AP	0.1	VP $\rightarrow$ Verb, WP	0.2
...	...	VP $\rightarrow$ Verb, S	0.1
PP $\rightarrow$ Prep, NP	0.2	...	...
...	...	AP $\rightarrow$ Adj	0.3
WP $\rightarrow$ WH, S	0.3	AP $\rightarrow$ Adj, NP	0.2
...	...	...	...

Fig. 7. PCFG rules. Pro stands for pronoun, Det for determiner, Adj for adjective, VP for verb phrase, NP for nominal phrase, PP for prepositional phrase, AP for adjective phrase, DP for adverbial phrase, and WP for wh-phrase.

the(Det) man(Noun) sings(Verb) a(Det) song(Noun)

An individual is represented as a data structure containing the following information:

- Fitness value.
- A list of *genes* representing the parse of different sets of words in the sentence.
- The number of genes in the individual.
- The depth of the parse tree.

Each gene represents the parse of a consecutive set of words in the sentence. If this parse involves non-terminal symbols, the parse of the subsequent partitions of the set of words is given in later genes. Accordingly, the information contained in a gene is the following:

- The sequence of words in the sentence analyzed by the gene. It is described by two data: the position in the sentence of the first word in the sequence, and the number of words in the sequence.
- The rule of the grammar used to parse the words in the gene.
- If the right hand side of the rule contains non terminal symbols, the gene also stores the list of references to the genes in which the analysis of these symbols continues.
- The depth of the node corresponding to the gene in the parse tree. It will be used in the evaluation function.

Figure 8 exhibits some possible individuals for the sentence of the running example. The data structure used to represent the *individual2* of Figure 8 appears in Figure 9. The first gene tells us that the sequence of words is segmented between the second and third words, i.e., the first two words correspond to the main NP, and the

```
NP: The man
VP: sings a song

NP -> Det,NP:
   Det: The
   NP: man

   NP -> Noun
      Noun: man

VP -> Verb, NP:
   Verb: sings
   NP: a song

   NP -> NP, AP
     NP -> Noun
        Noun: a
     AP -> Adj
        Adj: song
```

Individual 1

```
NP: The man
VP: sings a song

NP -> Adj,NP:
   Adj: The
   NP: man

   NP -> Noun
      Noun: man

VP -> Verb, PP:
   Verb: sings
   PP: a song

   PP -> Prep, NP
     Prep: a
     NP -> Noun
        Noun: song
```

Individual 2

Fig. 8. Possible individuals for the sentence *The man sings a song*.

following three words to the main VP. The gene also tells us that the parse of the NP is given by gene 2, and the parse of the VP is given by the gene 3. Since the rule in gene 2 has only terminal symbols in its right hand side, there is no subsequent gene decomposition. On the contrary, the rule for gene 3 presents a NP symbol in its right hand side, whose parse is done in gene 4. The process continues in the same manner. The other genes are represented in a similar way.

1) Initial Population: The first step of an EA is the creation of a initial generation of PS individuals, where PS is the population size. In our case the individuals of this population are generated by a random selection of rules weighted with their probability.

Let us assume the individuals of Figure 8 belong to the initial population and let us consider their generation process. The input sentence is randomly divided in two parts assigned to main NP and VP, enforcing that the sequence of words assigned to the main VP contains at least a verb. In *individual1* the rule chosen at random to

(fitness): X		
(number of genes): 4		
(genes):		
(gene id.)	(rule)	(gene decomposition):
		(first word, number of words, gene):
(1)	$S \rightarrow NP, VP$	NP:(1, 2, 2)
		VP:(3, 3, 3)
(2)	$NP \rightarrow Det, Noun$	Det: The
		Noun: man
(3)	$VP \rightarrow Verb, NP$	Verb: $sings$
		NP:(4, 2, 4)
(4)	$NP \rightarrow NP, AP$	NP:(4,1,5)
		AP:(5,1,6)
(5)	$NP \rightarrow Noun$	Noun: a
(6)	$AP \rightarrow Adj$	Adj: $song$

Fig. 9. Data structure which represents *individual 1* in Figure 8.

parse the NP is $NP \rightarrow Det, NP$. Therefore, the words assigned to NP are randomly divided in two sequences assigned to Det and NP. In this case, the word "The" is assigned to Det and the word "man" to NP. Since NP is a non terminal symbol the parsing process continues randomly choosing a NP-rule to parse the words assigned to NP. The NP-rule is randomly selected but only among those rules able to parse the number of words in the corresponding sequence, one word in this case. The rule chosen is $NP \rightarrow Noun$. Notice that a wrong rule could have been chosen as well. The parsing process proceeds in the same way for the VP, as well as for the *individual 2*.

Summarizing, the steps for creating individuals in the initial population are the following:

- The set of words in the sentence is randomly partitioned, making sure that there is at least one verb in the second part, which will correspond to the main VP. This decision has been taken to reduce the search space and improve the performance. Notice that this kind of constraints can be introduced to improve the performance without limiting the coverage of the system if a study of the sentence previous to the parse is introduced to determine the kind of components that can be safely excluded (not only verbs, but also other types of components). For example, a study previous to the generation of the population can determine that the sentence does not present any preposition and then the parses will be generated without PP components, etc.
- The set of words corresponding to the NP is parsed by randomly generating any of the possible NP rules. The same is done for generating the parse of the VP with the VP rules. The process is improved by enforcing the application of just those rules able to parse the right number of words in the gene.

- If the rules applied contain some non terminal symbol in its right hand side, the parsing process is applied to the set of words which are not yet assigned a category. Rules are selected accordingly to their probability.
- The process continues until there are no terminal symbols left pending to be parsed.

B. Fitness: Individual Evaluation

In each cycle, every individual is tested for its ability to parse the target sentence. Individuals with a larger than average ratio of correctly applied grammar rules should have a higher probability to survive for the next cycle. A rule is correctly applied if each component of its right-hand side parses a sub-tree of the kind of the component. The precise definition is given later as the concept of *coherent* gene. But the probability of the grammar rules of the individual must also be taken into account in this evaluation. Accordingly, the fitness value will be given by a couple of values, f_{coher}, which measures the ability of an individual to parse the target sentence, and f_{prob}, which measures the probability of the rules employed in the parse.

f_{coher} is based on the relative number of *coherent* genes. A gene will be considered *coherent* if

a) it corresponds to a rule whose right hand side is only composed of terminal symbols, and they correspond to the categories of the words to be parsed by the rule, or
b) it corresponds to a rule with non-terminal symbols in its right hand side and each of them is parsed by a coherent gene.

Accordingly, f_{coher} is computed as

$$\log\left(\frac{\text{number of coherent genes} - \sum_{i\in\text{incoherent genes}} \frac{\text{penalization}}{\text{depth}(i)}}{\text{total number of genes}}\right).$$

The formula takes into account the relative relevance of the genes: the higher in the parse tree is the node corresponding to an incoherent gene, the worse is the parse. Thus the fitness formula introduces a penalization factor which decreases with the depth of the gene.

f_{prob} is computed as

$$\log(\prod_{i=1}^{n} \mathrm{Prob}(g_i))$$

where $\mathrm{Prob}(g_i)$ is the probability assigned to the grammatical rule corresponding to gene g_i in the individual.

The fitness is then computed as a linear combination of both:

$$Fitness = wf_{\text{coher}} + (1-w)f_{\text{prob}}$$

where $w(>0)$ is a parameter which allows to tune the computation during the evolution process. In the first generations w is higher in order to produce individuals corresponding to valid parse trees, while later, w is lowered in order to select the most probable individuals.

C. Genetic Operators

The genetic operators considered herein are *crossover*, which combines two parses to generate a new one, and *mutation*, which creates a new parse by replacing a randomly selected gene in a previous individual.

```
     function Crossover( $C_1$, $C_2$, $\mathcal{S}$ ):  $C_off$
     begin
 1     selected_word := random_choose(1..length( S))
 2     gene1 := identify_gene($C_1$,selected_word,long_chro($C_1$))
 3     gene2 := identify_gene($C_2$,selected_word,long_chro($C_2$))
 4     while (words(gene1) <> words(gene2)) and
             (position(gene1) <> MAIN_NP) and
             (position(gene1) <> MAIN_VP) and
             (position(gene2) <> MAIN_NP) and
             (position(gene2) <> MAIN_VP) do
 5       gene1 := identify_gene($C_1$, selected_word, gene1)
 6       gene2 := identify_gene($C_2$, selected_word, gene2)
 7     end-while
 8     if (words(gene1) = words(gene2))
       and (type(gene1) = type(gene2)) then
 9       $Caux_1$ := exchange_gene($C_1$, $C_2$, gene1, gene2)
10       $Caux_2$ := exchange_gene($C_2$, $C_1$, gene2, gene1)
11       $C_off$ := select_best($Caux_1$, $Caux_2$)
12     else
13       $Caux_{11}$ := $C_1$; $Caux_{12}$ := $C_1$;
14       erase_gene($Caux_{11}$, gene1); erase_gene($Caux_{12}$, gene2)
15       generate_parse($Caux_{11}$, gene1)
16       generate_parse($Caux_{12}$, gene2)
17       $Caux_{21}$ := $C_2$; $Caux_{22}$ := $C_2$;
18       erase_gene($Caux_{21}$, gene1); erase_gene($Caux_{22}$, gene2)
19       generate_parse($Caux_{11}$, gene1)
20       generate_parse($Caux_{12}$, gene2)
21       $C_off$ := select_best($Caux_{11}$, $Caux_{12}$, $Caux_{21}$, $Caux_{22}$)
22     end-if
23     return $C_{off}$
     end
```

Fig. 10. Crossover operation algorithm. C_1 and C_2 are the parent individuals. C_{off} is the offspring individual. S is the input sentence.

1) Crossover: The crossover operator generates a new individual which is added to the population in the new generation. The part under a randomly selected parent's tree point is exchanged with the corresponding part of the other parent to produce two offsprings, subject to the constraint that the genes exchanged correspond to the same type of syntactic category (NP, VP, etc). This avoids wrong references of previous genes in the individual. Of course those exchanges which produce parses inconsistent with the number of words in the sentence must be avoided. Therefore, the crossover operation (Figure 10) performs the following steps:

- Select two parents, C_1 and C_2.
- Randomly select a word (`selected_word`) from the input sentence (line 1).
- Identify the innermost gene (`identify_gen`) to which the selected word corresponds in each parent (lines 2 and 3).
- If the genes correspond to different sets of words, the next gene in the inner most order is selected. This process continues until the sequences of words whose parses are to be exchanged are the same, or until the main NP or VP are reached (lines 4 to 7).
- If the two selected genes parse the same sequence of words the exchange is performed (lines 8 to 11).
- If the process to select genes leads to the main NP or VP, and the sequence of words do not match yet (lines 12 to 22), the exchange can not be performed. In this case a new procedure is followed: in each parent one of the two halves is maintained while the other one is randomly generated (`generate_parse`) to produce a parse consistent with the number of words in the sentence. This produces four offsprings.
- Finally, the best offspring `select_best`) is added to the population (line 23).

As an example let us consider the individuals in Figure 8 and let us assume that the word selected in the input sentence is the determiner *a*. Figure 11 shows the new individuals resulting from the crossover operation. The selected word corresponds to genes of different kind in each individual (Noun and Preposition) and therefore can not be exchanged. The next gene containing the selected word in its sequence of words is now considered. This corresponds to the main VP in both individuals which now can be exchanged. This exchange produces two new individuals (Figure 11). The best individual out of the two offsprings is number 1, which is added to the population.

2) Mutation: Selection for mutation is done in inverse proportion to the fitness of an individual. Mutation operation changes the parse of some randomly chosen sequence of words.

The mutation operation (Figure 12) performs the following steps:

- A gene is randomly chosen from the individual (lines 1 to 2).
- The parse of the selected gene, as well as every gene corresponding to its decomposition, are erased (line 3).
- A new parse is generated for the selected gene (line 4).

Let us consider the *individual 1* of Figure 8, and let us assume the randomly chosen gene for mutation is the one corresponding to the parse of the main VP. Then a new parse is generated for the sequence of words of this gene, producing the individual of Figure 13, which provides a correct parse of the input sentence.

```
NP: The man
VP: sings a song

NP -> Det,NP:
   Det: The
   NP: man

   NP -> Noun
      Noun: man

VP -> Verb, PP:
   Verb: sings
   PP: a song

   PP -> Prep, NP
     Prep: a
     NP -> Noun
        Noun: song
```

Offspring 1

```
NP: The man sings
VP:  a song

NP -> Adj,NP:
   Adj: The
   NP: man

   NP -> Noun
      Noun: man

VP -> Verb, NP:
   Verb: sings
   NP: a song

   NP -> NP, AP
     NP -> Noun
        Noun: a
     AP -> Adj
```

Offspring 2

Fig. 11. Individuals produced by a crossover operation on the individuals in Figure 8.

```
function Mutation($C$, $\mathcal{S}$)
begin
  n := number_genes($C$)
  selected_gene := random_choose(1..n)
  erase_parse($C$, selected_gene)
  generate_parse($C$, selected_gene)
end
```

Fig. 12. Mutation operation algorithm.

```
NP: The man
VP: sings a song

NP -> Det,NP:
   Det: The
   NP: man sings

   NP -> Noun
      Noun: man sings

VP -> Verb, NP
   Verb: sings
   NP -> Det, NP
      Det: a
      NP -> Noun
        Noun: song
```

Fig. 13. Individual resulting from a mutation operation of the individual 1 in Figure 8.

D. Experimental Results

The algorithm has been implemented in C++ language and run on a Pentium II processor. In general, the probabilistic grammar is extracted from a text syntactically annotated. However, the sentences of real texts present additional problems, such as punctuation symbols, expressions of multiple words, anaphora, etc, which have not been considered in this implementation. Because of this, the grammar and the sentences used in the experiments presented herein are artificial, constructed to test the system. In order to evaluate the performance we have considered the parsings of the sentences appearing in Table I. The average length of the sentences is around 10 words. However, they present different complexities for the parsing, mainly arising from the length and the number of subordinate phrases.

1) Study of the EA parameters: Experiments include the study of the performance with respect to the population size. Figure 14 shows the number of generations required to reach a correct parse for each sentence versus the population size. The behavior is different for each sentence. In general, the higher the "sentence complexity", i.e. the length and the number of subordinate phrases, the larger the population size required to reach the correct parse in a reasonable number of steps. Accordingly, we can consider that the sentences are approximately sorted by their increasing complexity. We can observe in the graph that the simplest sentences, 1 and 2, reach the correct parse in a few generations even for small populations, while *sentence 3* requires a large number of generations to reach the solution for small populations. On the other hand, sentences 4 and 5 only reach the correct parse for large

1	Jack(noun) spoke(verb) with(prep) Sue(noun) about(prep) the(det) book(noun)
2	Jack(noun) thinks(verb) Sue(noun) is(verb) happy(adj) in(prep) her(det) job(noun)
3	Jack(noun) regretted that(wh) he(pro) ate(verb) the(det) whole(adj) thing(noun)
4	The(det) man(noun) who(wh) gave(verb) Bill(noun) the(det) money(noun) drives(verb) a(det) big(adj) car(noun)
5	The(det) man(noun) who(wh) lives(verb) in(abv) the(det) red(adj) house(noun) saw(verb) the(det) thieves(noun) in(abv) the(det) bank(noun)

TABLE I

SENTENCES USED IN THE PARSING EXPERIMENTS.

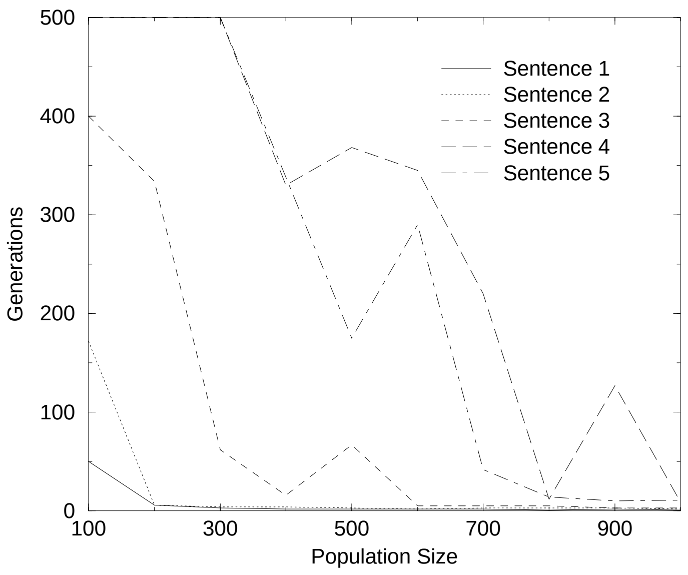

Fig. 14. Number of generations required to reach the correct parse for different input sentences, when using a crossover rate of 50% and a mutation rate of 20%.

populations. But large populations lead to replication of individuals and slow evolutions, so high percentages of genetic operators are required to speed up the process.

In order to illustrate the performance of the evolution process, Figure 15 represents the fitness of the best individual in the population versus the number of generations for sentence 2, with a population of 50 individuals and different crossover and mutation rates. The first observation is that the fitness grows abruptly at certain number of generations. We can also observe that a minimum percentage of application of genetic operators is required in order to reach the correct parse. Thus, for the population size considered, with a crossover percentage of 10% and one of mutation of 5% the solution is not reached in 500 generations, while with percentages of 30% and 10%

respectively the solution s reached after 450 generations, and with percentages of 50% and 20% respectively the solution is reached after 100 generations.

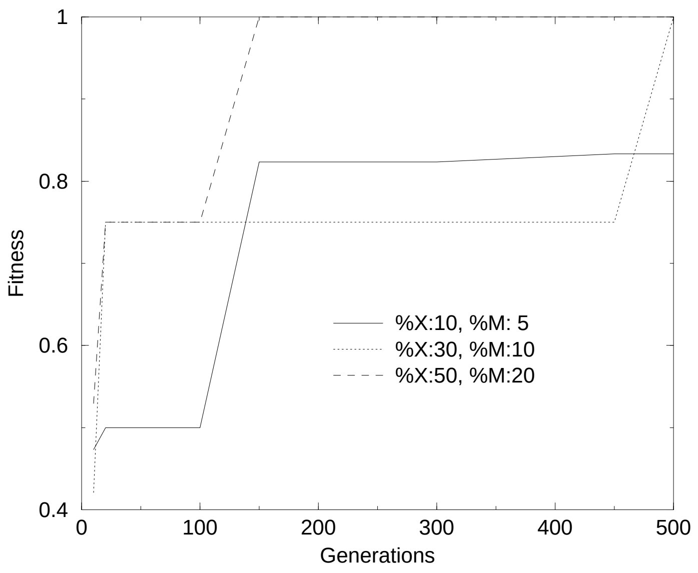

Fig. 15. Performance of the sentence 2 for different parameter settings, with a population size of 50 individuals.

E. *Parallel Parsing*

Despite the ability of evolutionary algorithms to find a "good" solution, though perhaps approximate, to optimization problems, if such problems are too hard they require a very long time to given an answer. This has led to different efforts to parallelize these programs. Basically, the approaches to this parallelization can be classified in two groups [13], [26]: *global* parallelization, and *island* or *coarse-grained* parallelization. In the first method there is only one population, as in the sequential model, and it is the evaluation of individuals, and the application of genetic operators what is parallelized. Thus, the behavior of the sequential algorithm does not change. This method can obtain significant speedup if the communication overhead is low, as it happens in shared memory machines.

The island or coarse_grain model is probably the most popular approach for parallelization of EAs because it does not require a high cost in communications, and it therefore exhibits a high portability. In this model, the population is divided in subpopulation or *demes*, which usually evolve isolated except for the exchange of some individuals or *migrations* after a number of generations. In this case, we can expect a different behavior with respect to the parallel model, both because this model employs different parameters of the algorithm (such as population size, which is smaller in each deme) and because the dynamics is completely changed by migrations. This can result in a faster convergence. Though such a faster convergence may, in principle, reduce the quality of the solutions, results show [14] that the parallel model with smaller populations but with migration among demes can improve the quality of the sequential solutions, and that there is an optimal number of demes which maximizes the performance.

The complexity of the parsing problem makes it appropriate for implementing a parallel version of the EA. Both the fitness function and the genetic operators are expensive enough to make the evolutionary program appropriate for a coarse grain parallel model. Accordingly, the parallel implementation follows an island model, in which, after a number of generations, demes exchange some individuals. The system is composed of a number of processes, called *cooperative parsers*, each of which performs, by evolutionary programming, a parse for the same input sentence. The seed in each deme (random number generation) is different in order to obtain different individuals in different populations. There is a special process, known as *main selector*, which selects the best individual among the best ones of each deme. In order to reduce the communication, migrations are not performed from one deme to each other, but only to the next deme in a round-robin manner (Figure 16). The N cooperative parsers are assigned an arbitrary number between 0 and $N-1$, and the $N-1$ parser considers that its next parser is numbered 0. Nevertheless, this policy has been compared to an all-to-all policy in order to guarantee that the adopted option does not implies a significant reduction of the solutions quality. The model is asynchronous, and after a fixed number

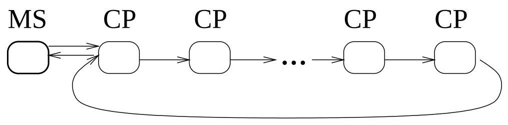

Fig. 16. Migration Policy. CP stands for Cooperative parser, MS for Main Selector.

of generations, a parser sends a fixed number of individuals to the next parser and then continues the evolution checking in each generation the arrival of the same number of individuals from the previous parser. The selection of individuals to be sent in a migration is done with a probability proportional to their fitness.

Different experiments have been carried out in order to tune this basic model. These experiments are described in the next section.

F. Experiments

The algorithm has been implemented in the C++ language on a SGI-CRAY ORIGIN 2000 computer using PVM (*Parallel Virtual Machine*) [20], a software package which allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. In order to evaluate the performance we have considered the parsing of the most complex sentences appearing in Table I.

Table II shows the improvement in performance obtained when increasing the number of processors. This experiment has been carried out with a population size of 200 individuals, the minimum required for the sequential version to reach the correct parse, a crossover rate of 50%, a mutation rate of 20%, a size of migrated population of 40 and an interval of migration of 15 generations. We can observe that the parallel execution achieves a significant improvement even with only 2 processors. We can also observe that performance reaches saturation for a certain number of processors (around 8). However, these figures are expected to increase with the complexity of the analyzed sentences.

Sentence	Sequential	Parallel				
		2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
sentence3	16.55	10.48	3.08	3.09	2.09	2.09
sentence4	50.03	19.12	15.02	10.64	3.48	3.49
sentence5	52.70	25.40	22.71	19.34	14.93	14.79

TABLE II

TIME IN SECONDS REQUIRED TO REACH A CORRECT PARSE WHEN PROCESSING SEQUENTIALLY AND IN PARALLEL.

V. CONCLUSIONS

This work develops a general framework, the NLP-EA scheme to design evolutionary algorithms devoted to tasks concerning Statistical Natural Language Processing (NLP). The statistical measurements extracted by the stochastic approaches to NLP provide an appropriate fitness function for the EA in a natural way. Furthermore, the corpus or training texts required for the application of statistical methods also provide a perfect benchmark for adjusting the algorithm parameters (population size, crossover and mutations rates, etc), a fundamental issue for the behaviour of the algorithm. On the other hand, EAs provide their characteristic way of performing a non-exhaustive search for solving NLP tasks. Accordingly, a natural symbiosis between the statistical NLP techniques and the EAs arises.

The NLP-EA scheme has been illustrated with an evolutionary algorithm for tagging that works with a population of potential taggings for each input sentence in the text. The evaluation of individuals is based on a training table composed of contexts extracted from a set of training texts. Results indicate that the evolutionary approach is a robust enough approach for tagging texts of natural language, obtaining accuracies comparable to other statistical systems. The tests indicate that the length of the contexts extracted for the training is a determining factor for the results.

Another example of application of the EA-NLP scheme has been an evolutionary algorithm for parsing. In this case, the algorithm works with a population of potential parses for a given PCFG and an input sentence. The appropriate data structure to represent potential parses has been designed in order to enable an easy evaluation of individuals and application of genetic operators. Again the results indicate the validity of the EA approach for parsing positive examples of natural language, and thus, the system developed here can provide a promising starting point for parsing sentences of realistic length and complexity. The tests show that the EA parameters need to be suitable for the input sentence complexity. The more complex the sentence (length and subordination degree), the larger the population size required to quickly reach a correct parse. Furthermore, as the population size increases, higher rates of crossover and mutation are required to maintain the efficiency of the algorithm. This algorithm has been parallelized in the form of an island model, in which processors exchange migrating populations asynchronously and in a round-robin sequence. Results obtained for these experiments exhibit a clear improvement in the performance, thus showing that the problem has enough granularity for the parallelization.

These experiments indicate the effectiveness of an evolutionary approach to tagging and parsing. The accuracy obtained in tagging with these methods is comparable to that of other probabilistic approaches, in spite that other systems use algorithms specifically designed for this problem. The evolutionary approach does not guarantee the right answer, but it always produces a close approximation to it in a reasonable amount of time. On the other hand, it can be a suitable tool for NLP because natural languages posses some features, such as some kinds of ambiguities, not even human beings are able to resolve. Furthermore, the result obtained from the EA can be the input for a classic method, which in this way will perform a smaller search, with methods such as logic programming or constraint programming.

Other possible future works are the unsupervised processing of different NLP tasks. For instance, we could be interested in performing unsupervised tagging of texts because we do not have an appropriate tagged corpus available for training. In this case there would be no training texts and all statistical information would have to be extracted from the text to be tagged. EAs can be applied to perform this kind of tasks, using generalizations for the tags until they get enough information to decide the most appropriate tag.

EAs might also provide new insights and a new directions to investigate the statistical nature of the language. In particular, they can give a new approach to natural language generation, for which the space of the results is very much open.

REFERENCES

[1] J. Allen. *Natural Language Understanding*. The Benjamin/Cumming Publishing Company, Inc., 1994.

[2] L. Araujo. Evolutionary parsing for a probabilistic context free grammar. In *Proc. of the Int. Conf. on on Rough Sets and Current Trends in Computing (RSCTC-2000), Lecture Notes in Computer Science 2005*, pages 590–597. Springer-Verlag, 2000.

[3] L. Araujo. A parallel evolutionary algorithm for stochastic natural language parsing. In *Proc. of the Int. Conf. Parallel Problem Solving from Nature (PPSNVII)*, 2002.

[4] L. Araujo. 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*, pages 230–239. Springer-Verlag, 2002.

[5] G. Berg. A connectionist parser with recursive sentence structure and lexical disambiguation. In *Proc. of the Tenth National Conf. on Artificial Intelligence*, pages 32–37. MIT Press, 1992.

[6] A. Berger, P. Brown, S. Della Pietra, V Della Pietra, J. Lafferty, H. Printz, and L. Ures. The candide system for machine translation. In *Proc. of the ARPA Conference on Human Language Technology*, pages 152–157. Morgan Kaufmann, 1994.

[7] T.L. Booth and R.A. Thomson. Applying probability measures to abstract languages. *IEEE Transaction on Computers C*, 22:442–450, 1973.

[8] C. Brew. Stochastic hpsg. In *Proc. of the 7th Conf. of the European Chapter of the Association for Computational Linguistics*, pages 83–89, Dublin, Ireland, University College, 1995.

[9] E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4), 1995.

[10] E. Brill. Unsupervised learning of disambiguation rules for part of speech tagging. In S. Armstrong, K. Church, P. Isabelle, S. Manzi, E. Tzoukermann, and D. Yarowsky, editors, *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Press, 1997.

[11] P. Brown, J. Cocke, S. Della Pietra, F. Jelinek, R. Mercer, and P. Roossin. Word sense desambiguation using statistical methods. In *Proc. of Annual Conference of the Association for Computational Linguistics*, pages 264–270. Association for Computational Linguistics, 1991.

[12] P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, and P. Roosin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, 1990.

[13] E. Cantú-Paz. A survey of parallel genetic algorithms. Technical report, Illinois Genetic Algoritms Laboratory, IlliGAL Report No. 97003, 1997.

[14] E. Cantu-Paz and D. E. Goldberg. Predicting speedups of idealized bounding cases of parallel genetic algorithms. In T. Back, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 113–120. CA: Morgan Kaufmann, 1997.

[15] E. Charniak. *Statistical Language Learning*. MIT press, 1993.

[16] E. Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18(4):33–44, 1997.

[17] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proc. of the Third Conf. on Applied Natural Language Processing*. Association for Computational Linguistics, 1992.

[18] C. DeMarcken. Parsing the lob corpus. In *Proc. of the 1990 of the Association for Computational Linguistics*. Association for Computational Linguistics, 1990.

[19] W.A. Gale and K.W. Church. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102, 1994.

[20] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. Pvm: Parallel virtual machine. Technical report, Oak Ridge National Laboratory, 1994.

[21] F. Jelinek. Self-organized language modelling for speech recognition. In J. Skwirzinski, editor, *Impact of Processing Techniques on Communications*. Dordrecht, 1985.

[22] B. Keller and R. Lutz. Evolving stochastic context-free grammars from examples using a minimum description length principle. In *In Worksop on Automata Induction, Grammatical Inference and Language Acquisition. ICML097*. ICML, 1997.

[23] Anne Kool. Literature survey, 2000.

[24] Robert M. Losee. Learning syntactic rules and tags with genetic algorithms for information retrieval and filtering: an empirical basis for grammatical rules. *Information Processing & Management*, 32(2):185–197, 1996.

[25] T. Dunning M. Davis. Query translation using evolutionary programming for multilingual information retrieval II. In *Proc. of the Fifth Annual Conf. on Evolutionary Programming*. Evolutionary Programming Society, 1996.

[26] R. Poli M. Nowostawski. Review and taxonomy of parallel genetic algorithms. Technical report, School of Computer Science, The University of Birmingham, UK, Technical Report CSRP-99-11, 1999.

[27] T. McEnery and A. Wilson. *Corpus Linguistics*. Edinburgh University Press, 1996.

[28] B. Merialdo. Tagging english text with a probabilistic model. *Computational Linguistics*, 20(2):155–172, 1994.

[29] Z. Michalewicz. *Genetic algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 2nd edition, 1994.

[30] R. Miikkulainen. *Subsymbolic Natural Language processing: An Integrated Model of Scripts, Lexicon, and Memory*. MIT press, 1993.

[31] J.R. Quinlan. *C 4.5: Programs for Machine Learning*. Morgan Kaufmann Publisher, 1993.

[32] H. Schmid. Part-of-speech tagging with neural networks. In *Proc. of the Int. Conf. on Computational Linguistics*, pages 172–176, 1994.

[33] H. Schutze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):7–124, 1998.

[34] H. Schutze and Y. Singer. Part od speech tagging using a variable memory markov model. In *Proc. of the 1994 of the Association for Computational Linguistics*. Association for Computational Linguistics, 1994.

[35] Tony C. Smith and Ian H. Witten. Probability-driven lexical classification: a corpus-based approach. In *Pacific Association for Computational Linguistics Conference*, pages 271–283. Association for Computational Linguistics, 1995.

[36] D. Wu. Aligning a parallel english-chinese corpus statistically with lexical criteria. In *Proc. of Annual Conference of the Association for Computational Linguistics*, pages 80–87. Association for Computational Linguistics, 1994.

[37] P. Wyard. Context free grammar induction using genetic algorithms. In *Proc. of the 4th Int. Conf. on Genetic Algorithms*, pages 514–518, 1991.