Tagging with Disambiguation Rules
A New Evolutionary Approach to the Part-of-Speech Tagging Problem

Ana Paula Silva[1], Arlindo Silva[1] and Irene Rodrigues[2]
[1]*Escola Superior de Tecnologia, Instituto Politécnico de Castelo Branco, Av[a] do Empresário, Castelo Branco, Portugal*
[2]*Departamento de Informática, Universidade de Évora, Évora, Portugal*

Keywords: Part-of-Speech Tagging, Disambiguation Rules, Evolutionary Algorithms, Genetic Algorithms, Natural Language Processing.

Abstract: In this paper we present an evolutionary approach to the part-of-speech tagging problem. The goal of part-of-speech tagging is to assign to each word of a text its part-of-speech. The task is not straightforward, because a large percentage of words has more than one possible part-of-speech, and the right choice is determined by the surrounding word's part-of-speeches. This means that to solve this problem we need a method to disambiguate a word's possible tags set. Traditionally there are two groups of methods used to tackle this task. The first group is based on statistical data concerning the different context's possibilities for a word, while the second group is based on rules, normally designed by human experts, that capture the language properties. In this work we present a solution that tries to incorporate both these approaches. The proposed system is divided in two components. First, we use an evolutionary algorithm that for each part-of-speech tag of the training corpus, evolves a set of disambiguation rules. We then use a second evolutionary algorithm, guided by the rules found earlier, to solve the tagging problem. The results obtained on two different corpora are amongst the best ones published for those corpora.

1 INTRODUCTION

The words of a language are grouped by lexical categories, normally designated by part-of-speech tags or word classes, such as nouns, verbs, adjectives, and adverbs. These categories represent the type of functions that words can assume in a sentence. The process of classifying words into their parts-of-speech, and labeling them accordingly, is known as part-of-speech tagging, POS tagging, or, simply, tagging. Tagging is a very important task in natural language processing (NLP), because it is a necessary step in a large number of more complex processes like parsing, machine translation, information retrieval, speech recognition, etc. In fact, it is the second step in the typical NLP pipeline, following tokenization (Steven Bird and Loper, 2009).

An important aspect of this task is that the same word can assume different functions depending on how it is used in the sentence, more specifically depending on it's surrounding words (context). For instance, the word *fly* can assume the function of a **noun**, or a **verb**, depending on how we choose to use it on a sentence: *The fly is an insect* and *How insects fly is a very complex subject*. These means that in order to assign to each word of a sentence it's correct tag, we have to consider the context in which each word appears.

A part-of-speech tagger processes a sequence of words and attaches a part-of-speech tag (from a predefined tag set) to each word. Most current taggers are based on statistical models defined on a set of parameters whose values are extracted from texts marked manually. The aim of such models is to assign to each word in a sentence the most likely part-of-speech, according to its context, i.e, according to the lexical categories of the words that surround it. In order to do this, statistics on the number of occurrences of different contexts, for each word part-of-speech assignment possibilities, are collected.

The simplest stochastic tagger, called the unigram tagger, makes decisions based only on the word itself. It assigns the tag that is most likely for one particular token. The training step just investigates all the words presented in the training corpus, and saves the most frequent tag for each word. The tagger then works like a simple lookup tagger, assigning to each word the tag learned on the training step. A n-gram tagger

Paula Silva A., Silva A. and Rodrigues I..
Tagging with Disambiguation Rules - A New Evolutionary Approach to the Part-of-Speech Tagging Problem.
DOI: 10.5220/0004112000050014
In *Proceedings of the 4th International Joint Conference on Computational Intelligence* (ECTA-2012), pages 5-14
ISBN: 978-989-8565-33-4

is a generalization of a unigram tagger whose context is the current word together with the part-of-speech tags of the $n-1$ preceding tokens. In this case, the training step saves, for each possible tag, the number of times it appears in every different context presented on the training corpus.

Since the surrounding words can also have various possibilities of classification, it is necessary to use a statistical model that allows the selection of the best choices for marking the entire sequence, according to the model. These stochastic taggers, usually based on hidden markov models, neither require knowledge of the rules of the language, nor try to deduce them. Therefore they can be applied to texts in any language, provided they can be first trained on a corpus for that language.

Other type of taggers are rule-based systems, that apply language rules to improve the tagging's accuracy. The first approaches in this category were based on rules designed by human linguistic experts. There are also attempts to automatically deduce those rules, with perhaps the most successful one being the Brill Tagger (Brill, 1995). The Brill's system automatic extract rules from a training corpus, and applies them in a iterative way in order to improve the tagging of the text. The results presented by Brill on the Wall Street Journal data set, with a closed vocabulary assumption, (97.2%) are among the bests results obtained so far in this task. Brill's rules are called transformation rules, and they allow to consider not only the tags that precede one particular word, like the traditional probabilistic taggers, but also the tags of the words that follow it.

Brill conduced experiments with two types of transformation rules: nonlexicalized transformation rules, which contemplate only the tags that surround one particular word, and lexicalized transformation rules, which consider the words itselves.

Considering Brill's work, it seams that a model based on rules can be more flexible, since it allows to consider not only the tags that precede but also the tags that follow one particular word. Information about the words itselves can also be used. Moreover, the format of the information collected, in the form of rules, is easier to analyze than a extreme high number of probabilistic values.

More recently, several evolutionary approaches have been proposed to solve the tagging problem. These approaches can also be divided by the type of information used to solve the problem, statistical information (Araujo, 2002; Araujo, 2004; Araujo, 2006; Araujo, 2007; Araujo et al., 2004; Alba et al., 2006), and rule-based information (Wilson and Heywood, 2005). Shortly, in the former, an evolutionary algorithm is used to assign the most likely tag to each word of a sentence, based on a context table, that basically has the same information that is used in the traditional probabilistic approaches. Notwithstanding, there is an important difference related with the context's shape, i.e they also take into account context information about the tags that follow a particular word.

On the other hand, the later is inspired by the Brill's tagger. In this case a genetic algorithm (GA) is used to evolve a set of transformations rules, that will be used to tag a text in much the same way as the Brill's tagger. While in Araujo's work the evolutionary algorithm is used to discover the best sequence of tags for the words of a sentence, using an information model based on statistical data, in Wilson's work the evolutionary algorithm is used to evolve the information model, in the form of a set of transformation rules, that will be used to tag the words of a sentence.

There are also some other aspects that can be used to determine a word's category beside it's context in a sentence (Steven Bird and Loper, 2009). In fact, the internal structure of a word may give useful clues as to the word's class. For example, *-ness* is a suffix that combines with an adjective to produce a noun, e.g., *happy* $\rightarrow$ *happiness*, *ill* $\rightarrow$ *illness*. Therefor, if we encounter a word that ends in *-ness*, it is very likely to be a noun. Similarly, *-ing* is a suffix that is most commonly associated with gerunds, like *walking*, *talking*, *thinking*, *listening*. We also might guess that any word ending in *-ed* is the past participle of a verb, and any word ending with *'s* is a possessive noun.

In this work we investigate the possibility of using an evolutionary algorithm to evolve a set of disambiguation rules, that contemplate not only context information, but also some information about the word's morphology. This rules are not transformation rules like Brill's or Wilson's rules, but a form of classification rules, which try to generalize the context information that is used in probabilistic taggers. We look at the problem as a classification problem, where the classes are the different part-of-speeches, and the predictive attributes are the context information, and some aspects about the words' internal structure. Our goal is to achieve a model that captures both of the advantages of statistical and rule based systems.

The tagging itself is also made by a second evolutionary algorithm, that uses the disambiguation rules to find the most likely sequence of tags for the words of a sentence. So, our system is composed by two steps. First, a set of disambiguation rules are discovered by an evolutionary algorithm, and than an evolutionary tagger is used to tag the words of a sentence, using the rules found in the first step.

The rest of the paper is organized as follows:

Section 2 describes the evolutionary algorithm used to discover the disambiguation rules. In section 3 we present the evolutionary tagger and the results achieved. Finally, Section 4 draws the main conclusions of this work.

2 EVOLUTIONARY ALGORITHM FOR DISAMBIGUATION RULES DISCOVERY

In this section we describe the use of a genetic algorithm to discover a set of disambiguation rules that solve the part-of-speech tagging problem. The algorithm works on a set of annotated texts. We will approach the problem as if we were trying to solve a classification problem, by discovering a set of classification rules. The motivation for using a genetic algorithm (GA) in this task, is that genetic algorithms are robust, adaptive search methods that perform a global search in the space of candidate solutions. As a result of their global search, they tend to cope better with attribute predictions than greedy data mining methods (Freitas, 2003).

We begin by selecting the predictive attributes that we will use, then discuss the aspects that concern the individuals' representation, genetic operators, selection and, finally, the fitness function.

2.1 Attribute Selection

Our aim is to discover a set of rules that take into consideration not only context information but also information about the words' morphology. For the context, we decided to consider the same information that was used in the work of Brill (Brill, 1995) and (Wilson and Heywood, 2005). Thus, we consider six attributes:

- The lexical category of the third word to the left.
- The lexical category of the second word to the left.
- The lexical category of the first word to the left.
- The lexical category of the first word to the right.
- The lexical category of the second word to the right.
- The lexical category of the third word to the right.

For the words' morphology information we decided to include the following attributes:

- The word is capitalized.
- The word is the first word of the sentence.
- The word ends with ***ed*** or ***ing*** or ***es*** or ***ould*** or ***'s*** or ***s***.
- The word has numbers or '.' and numbers.

The possible values for each of the first six attributes are the values of the corpus tag set from which the evolutionary algorithm will extract the rules. This set will depend on the annotated corpus used, since the set of used labels will vary for different annotated corpora. The last four attributes are boolean, and so the possible values are simply 0 and 1.

2.2 Individuals

Genetic algorithms for rule discovery can be divided into two dominant approaches, based on how the rules are encoded in the population of individuals. In the Michigan approach each individual encodes a single rule, while in the Pittsburgh approach each individual encodes a set of prediction rules. The choice between these two approaches depends strongly on the type of rules we want to find, which in turn is related to the type of data mining task we are interested to. In the case of classification tasks, we are interested in evaluating the quality of the rule set as a whole, as opposed to the individual assessment of a rule. That is, the interaction between the rules is important and therefore, for classification, the Pittsburgh approach seems to be more natural (Freitas, 2003). Examples of GAs following the Pittsburgh's approach are Gabil, (De Jong et al., 1993), GIL, (Janikow, 1993).

In our work, we are interest in a set of rules that will not be used for a standard classification problem, but will help the disambiguation task necessary to solve the tagging problem. In this sense, the Pittsburgh's approach seems to be more appropriate. However there is an important question to consider when we adopt this type of representation, and that concerns the size of the individuals. We could adopt a traditional fixed length representation, or we could adopt a non standard variable length representation. In the first case, the problem is to define which size to consider, since we usual don't know how many rules are necessary for a certain classification task. In the other hand, in the non standard variable length representation, there is a very difficult problem to deal with, which concerns the control of the individuals' length. Individuals tend to grow through the evolutionary algorithm generations, making it increasingly slower - this problem is the well known bloat problem.

Since we will have a very large training set, and therefore the algorithm will be very time consuming, we have chosen to adopt the Michigan's approach, so that we don't have to deal with the bloat problem. However, we didn't consider all the population as a set of rules representing a solution to the classification problem. Instead, we adopted a covering algorithm

approach, i.e, we run the genetic algorithm as many times as necessary to cover all the positive examples of the training set, evolving a rule in each run. After each execution ends, we store the rule represented by the best individual in the population. We also update the training set, removing all the positive examples that were covered by the best individual obtained in that run (see algorithm 1).

Algorithm 1: Covering Algorithm. sp and sn represent the sets of positive and negative examples. ps and gm give the population size and the maximum number of generations.

Require: sp, sn, ps, gm
Ensure: set_of_rules
  while $sp \neq \varnothing$ **do**
    $best\_rule \Leftarrow GeneticAlgorithm(sp, sn, ps, gm)$
    $sp \Leftarrow RemoveExamples(sp, best\_rule)$
    $set\_of\_rules \Leftarrow Add(set\_of\_rules, best\_rule)$
  end while

Algorithm 2: Genetic Algorithm. sp and sn represent the sets of positive and negative examples. ps and gm give the population size and the maximum number of generations.

Require: sp, sn, ps, gm
Ensure: $bestRule$
  pop=GenerateInitialPop(ps)
  while $gm \neq 0$ **do**
    Evaluate(pop)
    $mating\_pool \Leftarrow Selection(pop)$
    $new\_pop \Leftarrow Crossover(pop)$
    $new\_pop \Leftarrow Mutation(new\_pop)$
    $best\_old \Leftarrow GetBestInd(pop)$
    $worst \Leftarrow GetWorstInd(new\_pop)$
    $pop \Leftarrow Replace(worst, best\_old, new\_pop)$
    $gm \Leftarrow gm - 1$
  end while
  $best\_ind \Leftarrow GetBestInd(pop)$
  $best\_rule \Leftarrow Fenotypel(best\_ind)$

In our approach each individual represents a rule of the form *IF Antecedent THEN Consequent*, where *Antecedent* consists of a conjunction of predictive attributes and *Consequent* is the predicted class. In the next sections we explain how we encode the antecedent and consequent of a rule.

2.2.1 The Rule's Antecedent

A simple way to encode the antecedent of a rule (a conjunction of conditions) in an individual is to use a binary representation. Let's assume that a given attribute can take *k* discrete values. We can encode these values using *k* bits. The *i*-th attribute value, with $(i = 1, ..., k)$, is part of the rule condition if and only if the ith bit equals 1.

For instance, let's assume that we want to represent a rule antecedent that takes only one attribute into consideration, let's say, *WeatherCondition*, whose possible values are *Sunny*, *Raining*, *Foggy*, and *Windy*. Thus, a condition involving this attributes may be encoded at the expense of four bits. The interpretation of a sequence like 1001 would result in the following antecedent:

$$IF\ \ , WeatherCondition = \text{"}Sunny\text{"}\ \ OR\ WeatherCondition = \text{"}Windy\text{"}$$

As we have seen, this type of representation allows conditions with disjunctions. If we want to include a new attribute, we just need to include the sequence of bits required to encode the respective values. The representation can thus be extended to include any number of attributes, assuming that all are connected by logical conjunction. An important feature of this type of representation concerns the situation where all bits of a given attribute are 1. This means that any value is acceptable for that particular attribute, which in terms of interpretation indicates that this attribute should be ignored.

As we saw above, for our particular problem, we have a relatively large number of possible values for most of the attributes considered. Thus, a representation such as the one described above would lead to very long individuals. For this reason we adopted a slightly different representation, inspired by the representation used by Wilson.

For each of the first six attributes we used six bits. The first bit indicates whether the category should or should not be considered, and the following five bits represent the assumed value of the attribute in question. We adopted a table of 29 entries, and used the binary value represented by five bits to index this table. If the value exceeds the number 29, we used the remainder of the division by 29. The extra bit for each attribute allows us to ignore, in the antecedent of the rule, a given attribute, as in the previous representation when all the bits are 1. The remaining attributes were encoded by nine bits, each one indicating whether the property is, or is not, present. In short, each individual is composed by $6 \times 6 + 9 = 43$ bits.

Like in the standard representation, the attributes are linked by logical conjunction. However, the rules do not contemplate internal disjunctions between different allowable values for a given attribute. Nevertheless, this knowledge can be expressed by different rules for the same class.

2.2.2 The Rule's Consequent

In general, there are three different ways to represent the predicted class in an evolutionary algorithm (Freitas, 2003). One way is to encode it into the genome of the individual, opening the possibility of subjecting it to evolution (De Jong et al., 1993; Greene and Smith, 1993). Another way is to associate all individuals of the population to the same class, which is never modified during the execution of the algorithm. Thus, if we are to find a set of classification rules to predict *k* distinct classes, we need to run the evolutionary algorithm, at least *k* times. In each *i*-th execution, the algorithm only discover rules that predict the i-th class (Janikow, 1993). The third possibility consists in choosing the predicted class in a more or less deterministic way. The chosen class may be the one that has more representatives in the set of examples that satisfy the antecedent of the rule (Giordana and Neri, 1995), or the class that maximizes the performance of the individual (Noda et al., 1999) . We adopted the second possibility, so we didn't need to encode the rule's consequent. Since we used a covering approach, we run the covering algorithm for each class independently.

2.2.3 Initial Population

50% of the individuals of the initial population were randomly generated and the other half were obtained by randomly choosing examples from the set of positive examples. These examples were first converted to the adopted binary representation and then added to the population.

2.3 Training Set

We used the Brown Corpus to create the training sets that we provided as input to the evolutionary algorithm. The examples considered were extracted from 90% of the corpus. For each word of the corpus we collected the values for every attribute included in the rule's antecedent, creating a specific training example. Then, for each tag of the tag set, we built a training set composed by positive and negative examples of the tag. Usually, the set of positive (negative) examples of a class is composed by examples that do (do not) belong to that particular class. However in our case we are not interested in finding typical classification rules, our goal is not to solve a classification problem, we just need rules that allow us to choose the best tag from a set of possible tags. This set is not all the tag set, but a subset of it, usually composed by a few number of elements. When we have a word that has only one possible lexical class, the tagging is straightforward. The problematic words are the ones that are ambiguous. Thus, our training set only includes examples corresponding to ambiguous words. With this in mind, we decided to use as positive examples of a class c_i only the examples concerning words that are ambiguous and are tagged with class c_i in the training corpus. As negative examples we consider every example that correspond to a word that could be used as c_i, but is tagged with a class different from c_i.

2.4 Fitness Function

Rules must be evaluated during the training process in order to establish points of reference for the evolutionary training algorithm. The rule evaluation function must not only consider instances correctly classified, but also the ones left to classify and the wrongly classified ones. To evaluate our rules we used the well known F_β-measure:

$$F_\beta = (1+\beta)^2 \times \frac{precision \times recall}{(\beta^2 \times precison) + recall} \quad (1)$$

$$precision = \frac{TP}{TP+FP} \quad (2)$$

$$recall = \frac{TP}{TP+FN} \quad (3)$$

where:

- TP - True Positives = number of instances covered by the rule that are correctly classified, i.e., its class matches the training target class;
- FP - False Positives = number of instances covered by the rule that are wrongly classified, i.e., its class differs from the training target class;
- FN - False Negatives = number of instances not covered by the rule, whose class matches the training target class.

The F_β-measure can be interpreted as a weighted average of precision and recall. We used $\beta = 0.09$, which means we put more emphasis on precision than recall.

2.5 Genetic Operators and Selection

Since our representation is a typical binary representation, we didn't need to use special operators. We used a traditional two point crossover and binary mutation as genetic operators. In the two point crossover operator, two crossover points were randomly selected, and the inner segments of each parent were switched, thus producing two offsprings. The mutation operator used was the standard binary mutation:

if the gene has the allele 1, it mutates to 0, and vice versa. We used a mutation probability of 0.01 and a 0.75 crossover probability. These values were empirically determined.

For the selection scheme we used a tournament selection of size two with $k = 0.8$. We also used elitism, preserving the best individual of each generation by replacing the worst individual of the new population by the best of the old one (see algorithm 2).

2.6 Experimental Results

We developed our system in Python and used the resources available on the NLTK (Natural Language Toolkit) package in our experiences. The NLTK package provides, among others, the Brown corpus and a sample of 10% of the Penn Treebank corpus. It also provides several Python modules to process those corpora. Since different corpora use different formats for storing part-of-speech tags, the NLTK's corpus readers were very useful, by providing a uniform interface.

As we said before, tagged corpora use many different conventions for tagging words. This means that the tag sets vary from corpus to corpus. To extract the disambiguation rules from a set of annotated texts, we need to run our algorithm for each of the tags belonging to the tag set. However, if we want to test the resulting rules in a different corpus, we will not be able to measure the performance of our tagger, since the corpus tag set may be different. To avoid this, we decided to use the *simplify_tags=True* option of the *tagged_sentence* module of NLTK corpus readers. When this option is set to *True*, NLTK converts the respective tag set of the corpus used to a uniform simplified tag set, composed by 29 tags. This simplified tag set establishes the set of classes we use in our algorithm. We ran the covering algorithm for each one of the classes that had ambiguous words in the training corpus. There were 20 lexical classes in these conditions and for each one we defined the respective sets of positive and negative examples. We used 90% of the Brown corpus to extract the examples used to discover the disambiguation rules, using the process described in the previous section. This resulted in a total of 61113 examples.

In the next table (1) we present for each class the number of distinct examples (negative and positive) used. Each one appears at least one time in the training corpus. To construct the training set of positive and negative examples for each class, we reduced the number of distinct examples by eliminating those that were less frequent. This reduction is intended to make the algorithm faster, by eliminating examples that are not meaningful or even just noise. It is worth noting that each example has associated the number of times it occurs in the training corpus. This way we guarantee that the statistical information of each instance is not lost.

Table 1: TP (TN) column shows the number of distinct positive (negative) examples that where found in the training corpus for each of the 20 tags considered; SP (SN) shows the number of distinct positive (negative) examples that were used in the discovery of the disambiguation rules. This simplification results from eliminating the less frequent examples.

Class	TP	TN	SP	SN
ADV	25420	99481	1194	8366
VD	17689	23200	484	683
TO	13754	10363	1895	1248
CNJ	21838	22387	1822	2151
PRO	13998	4411	1235	339
VG	8107	3176	145	83
DET	46427	40160	4648	2622
VN	21400	19790	669	683
N	66576	92770	2582	6336
UH	209	10499	198	866
P	66750	37380	7055	3613
NUM	5199	837	416	40
EX	2070	471	192	26
V	18474	63185	477	3046
NP	4185	11219	165	1182
VBZ	2773	8380	38	179
WH	2748	8512	233	822
ADJ	28466	29652	1065	1182
FW	279	25130	279	2990
MOD	5308	541	591	25

The genetic algorithm was run with a population size of 200 individuals for a maximum of 80 generations. These values were established after some preliminary experiments. The number of rules that result in the best tagging, discovered by the algorithm for each of the 20 ambiguous classes, are presented in table 2. A total number of 2834 rules were found. The list below shows some examples of the discovered rules:

- If **Following** tag is **ADJ** and **Second Following** tag is **N** THEN **DET** with $Precision = 0.976$ and $TP = 4702$.
- If **Previous** tag is **V** and **Following** tag is **N** THEN **DET** with $Precision = 0.981$ and $TP = 1720$.
- If **Previous** tag is **V** and **ends** with ***-ed*** THEN **VN** with $Precision = 1$ and $TP = 753$.
- If **Following** tag is **V** THEN **TO** with $Precision = 1$ and $TP = 7113$.

Table 2: Number of rules discovered by the genetic algorithm for each of the 20 ambiguous classes considered.

Class	Number of Rules
ADV	396
VD	179
TO	5
CNJ	378
PRO	177
VG	43
DET	421
VN	62
N	364
UH	6
P	289
NUM	42
EX	13
V	121
NP	67
VBZ	15
WH	67
ADJ	196
FW	42
MOD	15

3 EVOLUTIONARY TAGGER

In the previous section we presented an evolutionary algorithm designed to discover a set of sets of disambiguation rules for a predefined set of tags. Each set is composed by a certain number of rules, and each rule has associated a precision value and the total number of positive examples covered by that rule (including repetitions).

We now want to use these rules so that we can mark the words of a sentence with the appropriate part-of-speech. Therefore, if we want to decide which tag to choose for a particular word, from the set of possible tags for that word, we should be able to do that by applying the set of rules previously found.

We could do this by applying the correspondent sets of rules to the particular word, choosing the class indexing the set which includes the best rule. Thus, if we want to tag the word w_i, and w_i can be tagged with one of the tags of the set S_i, for each possible tag $t_k \in S_i$, we should apply the correspondent set of disambiguation rules R_{t_k} to w_i. The best tag should be the one that indexes the set which includes the best rule. To measure the rule's quality, we could use the product between the precision value and the total number of positive examples covered by that rule.

However, the decision we need to make so that we can solve the tagging problem, does not only concerns which tag is the best for a particular word, but also the best sequence of tags to mark a sequence of words. Since one tagging decision affects the tagging choices for all the word's neighbors, we need to use a method that could give us the optimal sequence of choices. We used an approach similar to the one presented in (Araujo, 2002). However, instead of using the training table, we use the disambiguation rules. Our tagger receives as input a non annotated sentence, a dictionary with all the words present in the corpus (and their possible tags), and a set of sets of disambiguation rules. The tagger returns the same sentence with each word associated with a tag.

An important aspect of the tagging problem, besides the existence of ambiguous words, is the possibility of occurring unknown words. In this work we adopt a closed vocabulary assumption, i.e. there are no unknown words in the test set. Our goal is to compare our results to the ones achieved by the three approaches presented earlier, based on the same assumptions. However, we designed the evolutionary tagging algorithm to be able to deal with this possibility.

3.1 Representation

An individual is represented by a chromosome made of a sequence of genes. The number of genes in a chromosome equals the number of words in the input sentence. Each gene proposes a candidate tag for the word in the homologous position. For example, consider the input sentence: "The cat sat on the mat." A possible individual would be represented by a chromosome made of six genes, such as the one below:

g_1	g_2	g_3	g_4	g_5	g_6
DET	N	VD	P	DET	N

To evaluate the individual we need to apply the disambiguation rules. However, as we discussed in the previous section, our rules have six attributes related with the word context, and other nine attributes concerning some morphological properties of the words. Therefore, so that we can apply the disambiguation rules, we need to extract the needed attributes from the input sentence and from the tags proposed by the genes. This way each pair w_i/g_i gives rise to a 15-tuple of properties with the following alignment:

1. The lexical category proposed by the third gene to the left;
2. The lexical category proposed by the second gene to the left;

3. The lexical category proposed by the first gene to the left;
4. The lexical category proposed by the first gene to the right;
5. The lexical category proposed by the second gene to the right;
6. The lexical category proposed by the third gene to the right;
7. True if the homologous word is capitalized, false otherwise;
8. True if the homologous word is the first word of the sentence, false otherwise;
9. True if the homologous word ends with ***ed***, false otherwise;
10. True if the homologous word ends with ***ing***, false otherwise;
11. True if the homologous word ends with ***es***, false otherwise;
12. True if the homologous word ends with ***ould***, false otherwise;
13. True if the homologous word ends with ***'s***, false otherwise;
14. True if the homologous word ends with ***s***, false otherwise;
15. True if the homologous word has numbers or '.' and numbers, false otherwise.

When there is no gene (no corresponding word) in one of the positions contemplated in the context, we adopted an extra tag named 'None'. This can happen with the first three and last three genes of the individual.

We adopted a symbolic representation, i.e. the possible alleles of a gene are the the tags of the tag set adopted for the corpus in which the experiences will be executed. However, the allowed alleles of a gene are only the ones that correspond to the possible tags of the word the gene represents.

The initial population is generated by choosing, for each gene, one of the possible tags for the corresponding word. If the word is not in the dictionary, the algorithm chooses randomly one of the classes whose rule set has a rule which covers the example defined by the 15-tuple of the corresponding gene. If none of the rules cover the 15-tuple, the algorithm chooses by default the class N, which is the most frequent lexical class in the english language.

3.2 Genetic Operators and Selection

We used a typical one point crossover with a 0.8 probability. The mutation operator randomly chooses another allele from the set of possible alleles for the particular gene and was applied with a 0.05 probability. Again, if the word is unknown, the sets of rules will be used to determine which ones include a rule that covers the 15-tuple, and one of the possibilities will be randomly chosen and assigned to the corresponding gene. We adopted a tournament selection of size two with $k = 0.7$ and also used elitism, replacing the worst individual of each new population with the best of the old one. All the values were empirically determined in a small set of preliminary experiments.

3.3 Fitness Function

The performance of an individual is measured by the sum of the performances of his genes. Let's consider t_i to be the lexical category proposed by the gene g_i for the word w_i , and p_i to be the 15-tuple of properties determined by g_i. If R_{t_i} represents the set of disambiguation rules for the lexical category t_i, and $RP_i \subset R_{t_i}$ the set of all rules $r_k \in R_{t_i}$ that cover the 15-tuple p_i, the evaluation of g_i, is defined by

$$FG(g_i) = \begin{cases} \max\{P(r) \times TP(r) | r \in RP_i\} & if \ RP_i \neq \varnothing \\ Prob(t_i, w_i) & otherwise \end{cases} \quad (4)$$

where $P(r)$ gives the precision of rule r, $TP(r)$ gives the number of examples of the training corpus covered by r and $Prob(t_i, w_i)$ the probability of the word w_i appearing with tag t_i in the corpus.

The fitness of an individual i with n genes is given by:

$$Fitness(i) = \sum_{j=1}^{n} FG(g_j) \quad (5)$$

3.4 Experimental Results

We tested our evolutionary tagger on 12006 words of the Penn Treebank corpus and on 7527 words of the Brown corpus. We achieved an accuracy of 96.9% on the Pen Treebank of the Wall Street Journal corpus and a 96.49% accuracy on the Brown corpus (table 3). We ran the evolutionary tagger with a population of 20 individuals, during 30 generations. The experiments that we performed show that the evolutionary tagger usually finds a solution very quickly. In fact the difficulty level of the tagging task depends on the number of ambiguous words of the sentence we want to tag. Although it is possible to construct sentences in which every word is ambiguous (Hindle, 1989), such as the following:

Her hand had come to rest on that very book.

Table 3: Results achieved by the Evolutionary Tagger on the Penn Treebank corpus and on the Brown corpus, along with the results achieved by the approaches more similar to the one presented here.

Tagger	Corpus	Train	Test	Accuracy
Evolutionary Tagger	Brown	61113	7527	96.5
Evolutionary Tagger	Penn Treebank WSJ	-	12006	96.9
Wilson's	Penn Treebank WSJ	600000	-	89.8
Brill's	Penn Treebank WSJ	600000	150000	97.2
Araujo's	Brown	185000	2500	95.4

those situations are not the most common. After counting the number of ambiguous words that appear in the sentences of the 10% of the Brown corpus we reserved for testing the tagger, we observed that, in average, there are 6.9 ambiguous words per sentence. This explain the considerable low number of individuals and generations needed to achieve a solution. We could argue that in those conditions the use of a genetic algorithm is unnecessary, and that a exhaustive search could be applied to solve the problem. However, we can not ignore the worst case scenario, where, like we see above, all the words, or a large majority of the words, on a very long sentence may be ambiguous. Furthermore, we observed that the sentence average size of the Brown corpus is of 20.25 tokens, with a maximum of 180. The largest number of ambiguous words on a sentence belonging to this corpus is 68. Even for the smallest degree of ambiguity, with only two possible tags for each word, we have a search space of 2^{68}, which fully justifies the use of a global search algorithm such as a GA.

The results achieved show that there are no significant differences on the accuracy obtained by the tagger on the two test sets used. At this point, it is important to recall that the disambiguation rules used on the tagger were extracted from a subset (different from the test set used in this experiments) of the Brown corpus. Which bring us to the conclusion that the rules learned on step one are generic enough to be used on different corpora, and are not domain dependent.

4 CONCLUSIONS

We described a new evolutionary approach to the part-of-speach tagging problem that achieved results comparable to the best ones found in the area bibliography. Although there are other approaches to this problem that use, in some way, evolutionary algorithms, as far as we know this is the first attempt that uses these algorithms to solve all aspects of the task. In the previous works the evolutionary approach was applied in two different ways:

- to perform the tagging (Araujo, 2002). Here the evolutionary algorithm was oriented by statistical data, that was collected in much the same way as in the statistical approaches;
- to discover a set of transformation rules (Wilson and Heywood, 2005). Here the tagging is not done by the evolutionary algorithm. The author uses an evolutionary algorithm to discover a list of transformations rules, that is then used to perform the tagging in a deterministic way.

In our approach to the problem, we used an evolutionary algorithm to discover a set of disambiguation rules and then used those rules to evaluate the sequences of tags for the words of a sentence, with the sequences being evolved by another evolutionary algorithm.

When we first begun to research this problem, we concluded that there were two main approaches. The most frequent were based on statistical data collected from a training corpus, concerning the words' left context (the tags appearing left of a word); the others were based on rules: disambiguation rules, generally constructed by human experts, and transformation rules, firstly presented by Brill. Our intention was to capture the positive aspects of this two main approaches. We wanted to use rules since they are more flexible in terms of the kind of information we can use to solve the disambiguation problem, and are also more comprehensible than pure statistical data. However, we couldn't ignore the good results presented by the statistical approaches. And this led us to the idea of transforming the statistical data in a set of disambiguation rules.

We wanted to generalize the statistical information normally used on the traditional approaches, and, simultaneously, include other type of information, presenting it in a way that could be easily interpreted, i.e. in the form of rules. This generalization was achieved by the discovery of the disambiguation rules, with the statistical information being reflected on the quality measure of each rule. Our expectations were that this generalization could reduce the size of the data needed to do the disambiguation in the tagging task and that the information acquired would be less domain dependent than in previous approaches.

We tested our approach on two different corpora: in a test set of the corpus used to discover the disambiguation rules, and on a different corpus. The results obtained are among the best ones published for the corpora used in the experiments. Also there were no significant differences between the results achieved in the subset belonging to the same corpus from which we defined the training set, used to discover the rules, and the results obtained on the sentences of the other corpus. This confirms our expectations concerning the domain independence of the obtained rules.

Although we consider our results very promising, we are aware of the necessity of test our approach with a larger tag set, and to apply it to more corpora. We intend to test the tagger on other languages, as well. We also think that this approach could be applied to other natural language processing tasks like noun phrase chunking and named-entity recognition.

REFERENCES

Alba, E., Luque, G., and Araujo, L. (2006). Natural language tagging with genetic algorithms. *Information Processing Letters*, 100(5):173 – 182.

Araujo, L. (2002). Part-of-speech tagging with evolutionary algorithms. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, volume 2276 of *Lecture Notes in Computer Science*, pages 187–203. Springer Berlin / Heidelberg.

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

Araujo, L. (2006). Multiobjective genetic programming for natural language parsing and tagging. In Runarsson, T., Beyer, H.-G., Burke, E., Merelo-Guervós, J., Whitley, L., and Yao, X., editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 433–442. Springer Berlin / Heidelberg.

Araujo, L. (2007). How evolutionary algorithms are applied to statistical natural language processing. *Artificial Intelligence Review*, 28(4):275–303.

Araujo, L., Luque, G., and Alba, E. (2004). Metaheuristics for natural language tagging. In *Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference*, volume 3102 of *Lecture Notes in Computer Science*, pages 889–900. Springer.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. *Comput. Linguist.*, 21:543–565.

De Jong, K. A., Spears, W. M., and Gordon, D. F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188. 10.1023/A:1022617912649.

Freitas, A. A. (2003). *A survey of evolutionary algorithms for data mining and knowledge discovery*, pages 819–845. Springer-Verlag New York, Inc., New York, NY, USA.

Giordana, A. and Neri, F. (1995). Search-intensive concept induction. *Evol. Comput.*, 3:375–416.

Greene, D. P. and Smith, S. F. (1993). Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257.

Hindle, D. (1989). Acquiring disambiguation rules from text.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228. 10.1007/BF00993043.

Noda, E., Freitas, A., and Lopes, H. (1999). Discovering interesting prediction rules with a genetic algorithm. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages 3 vol. (xxxvii+2348).

Steven Bird, E. K. and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

Wilson, G. and Heywood, M. (2005). Use of a genetic algorithm in brill's transformation-based part-of-speech tagger. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 2067–2073, New York, NY, USA. ACM.