Highly Accurate error-driven method for Noun Phrase Detection

Lourdes Araujo [a,*,1] J. Ignacio Serrano [b,2]

[a] *Dpto. Lenguajes y Sistemas Informáticos, ETSI Informática, Universidad Nacional de Educación a Distancia, C/ Juan del Rosal, 16. Madrid 28040, Spain.*

[b] *Instituto de Automática Industrial, CSIC. Ctra Campo Real, km. 0,200. La Poveda. 28500 Arganda del Rey. Madrid. Spain.*

Abstract

We present a new model for detection of noun phrases in unrestricted text, whose most outstanding feature is its flexibility: the system is able to recognize noun phrases similar enough to the ones given by the inferred noun phrase grammar. The system provides a probabilistic finite-state automaton able to recognize the part-of-speech tag sequences which define a noun phrase. The recognition flexibility is possible by using a very accurate set of rankings for the FSA transitions. These accurate rankings are obtained by means of an evolutionary algorithm, which works with both, positive and negative examples of the language, thus improving the system coverage while maintaining its precision. We have tested the system on different corpora and evaluated different aspects of the system performance. We have also investigated other ways of improving the performance such as the application of certain filters in the training sets. The comparison of our results with other systems has revealed a considerable performance improvement.

Key words: noun phrase detection, evolutionary programming, grammar induction, information retrieval

1. Introduction

The huge size which internet has reached nowadays has converted information retrieval, which aims to discover the relevant information for a particular user, in one of the most active research areas. The amount of text data in electronic format which appears every day overwhelms the human capacity to extract the needed information. This has led to different approaches to automate the process, which rely on the automatic derivation of indexes for any document. Indexes are very useful in searching information because they characterize the documents by providing a valid list of *topic terms*. Identifying index terms is one of the hardest parts of the search process (Furnas et al., 1987). Noun phrases (NP), i.e. units whose head or principal word is a noun, are the main bearers of information content (Justeson and Katz, 1995; Boguraev and Kennedy, 1997). Therefore, the identification of NPs is crucial in many information retrieval processes. Besides, NP identification is also useful in many natural language processing (NLP) tasks, such as word sense disambiguation (Stevenson, 2003), machine translation (Koehn et al., 2003), compound detection (Lapata and Lascarides, 2003), etc. Different approaches have been proposed for the NP identification problem. Some of them rely on linguistic knowledge and use a hand-coded language model. Bourigault (Bourigault, 1992) uses a handcrafted NP grammar along with some heuristics for identifying NPs of maximal length, and Voutilainen (Voutilainen, 1993) uses a constraint grammar formalism. Other proposals follow a machine learning approach based

* Corresponding author.
Email addresses: lurdes@lsi.uned.es (Lourdes Araujo), nachosm@iai.csic.es (J. Ignacio Serrano).

[1] Supported by the Regional Government of Madrid under the Research Network MAVIR (S-0505/TIC-0267).
[2] Supported by project FIT150500-2003-373

on the use of labeled corpora. Church (Church, 1988) uses a probabilistic model automatically trained on the Brown corpus to detect NPs as well as to assign parts of speech. Ramshaw and Marcus (Ramshaw and Marcus, 1995) use the supervised learning methods proposed by Brill (Brill, 1995) to learn a set of transformation rules for the problem. Pla and Prieto (Pla et al., 2000) use grammatical inference to obtain a finite-state automaton (FSA) which recognizes NPs. Their method has inspired this work. Other approaches for chunking in general have also been explored, such as networks of linear units (SNoW) (Muñoz et al., 1999), Support Vector Machines with polynomial kernel (Kudo and Matsumoto, 2000) voting between different representations (Kudo and Matsumoto, 2001), a Winnow algorithm (Zhang et al., 2002), conditional random fields (Sha and Pereira, 2003) and maximum likelihood trigram models also with different representation voting (Shen and Sarkar, 2005).

This paper presents a new model for a *flexible* identification of NPs in an arbitrary text. The system generates a probabilistic finite-state automaton (FSA) able to recognize the sequences of part-of-speech (POS) tags which form an NP. An NP is a phrase whose head is a noun or a pronoun, possibly accompanied by a set of modifiers. Accordingly an NP can include any kind of POS tag. The FSA is generated with the Error Correcting Grammatical Inference (ECGI) algorithm and initial probabilities are assigned using the collected bigram probabilities. The FSA rankings assigned to the transitions of each state provide a method for a flexible recognition of input sequences, because they are considered to be NPs even if they are not accepted by the FSA but are similar enough to an accepted one. Thus, the system is able to recognize NPs not present in the training examples, what has proven very advantageous for the performance of the system. Some examples of NPs that we have found during our experiments in the test sets and were not present in the training sets are the following:

NP	tagged text
PDT DT NN	(Such a basis)
RBS JJ NNS	(Most political analysts)

where PDT stands for predeterminer, DT for determiner, NN for noun (singular or mass), RBS for adverb (superlative), JJ for adjective, and NNS for noun (plural).

The FSA rankings, which are crucial for the flexibility in recognition, are optimized by applying an evolutionary algorithm (EA). EAs allow dealing with the large search spaces of some complex tasks appearing in NLP (Kool, 2000; Araujo, 2004b). The EA that we introduce here to produce an accurate set of rankings for the FSA transitions, searches in a large space of combinations of transition rankings in an efficient way. The EA uses both, positive and negative training examples, which contributes to improve the coverage of the system while maintaining a high precision. The accuracy of the system is further improved by introducing additional mechanisms, such as a filtering process to the training set. The aim of this filter is to exclude some sequences of tags which make the system prone to recognize false NPs.

Some comparisons of different approaches to NP detection have been made. Pla (Pla, 2000) gathers results of a number of parsers trained on the data set used by Ramshaw (Ramshaw and Marcus, 1995): the one proposed by Cardie & Pierce (Cardie and Pierce, 1998), whose technique consists in matching part-of-speech (POS) tag sequences from an initial NP grammar extracted from an annotated corpus and then ranking and pruning the rules according to their achieved precision; the memory-based approach presented in (Veenstra, 1998), which introduces cascaded chunking, a process in two stages in which the classification of the first stage is used to improve the performance of the second one; the Memory-Based Sequence Learning (MBSL) algorithm (Argamon et al., 1998), which learns sequences of POS tags and brackets, and the hybrid approach of Tjong-Kim-Sang (Tjong-Kim-Sang, 2000), which uses a weighted voting to combine the output of different models. In order to compare our results with other systems, we have also tested our system on the same test set.

The remainder of the paper describes the model, its implementation and its evaluation. Section 2 describes the different phases of our approach for NP identification in more detail. Section 3 presents the EA used to training the FSA. Section 4 describes a filtering mechanism applied to the training set. The evaluation of different aspects of the system, as well as a comparison of the overall performance with other systems, are presented in Section 5. Finally, conclusions are drawn in Section 6.

2. System Overview

This work presents the design and implementation of a flexible recognizer of NPs given as sequences of POS tags. This recognizer is a probabilistic FSA constructed from examples of objective syntagmas by using an algorithm of grammar inference (ECGI). In its turn, this algorithm uses the Viterbi (Viterbi, 1967; Forney, 1973) algorithm, a dynamic programming one, so as to find the tag sequence most similar to a given one. In order to extend the recognition capabilities of the FSA beyond the set of training examples, it is necessary to enhance the system with a mechanism to recognize also other NPs not appearing in the examples, but very similar to the ones gathered in the grammar. This mechanism to introduce flexibility amounts to allowing the FSA to recognize NPs even if a certain number of errors occur during the process. If the number of errors incurred in processing an NP is below a threshold value, then the new NP is also recognized. An error appears whenever the input tag does not cause any transition from the current state of the FSA. In order to allow the FSA to find NPs similar to those of the input, their edges are labeled with a ranking of each transition between tags within an NP. These rankings are initially assigned the probabilities extracted from the NPs examples, and afterwards an evolutionary algorithm is applied to optimize the rankings. This algorithm uses NP examples different from those used in the FSA construction, as well as negative examples, i.e. syntagmas which must not be recognized as NPs. Figure 1 shows a scheme of the relationships between the different components and phases involved in the construction of the flexible recognizer.

Let us now describe the different elements of the system.

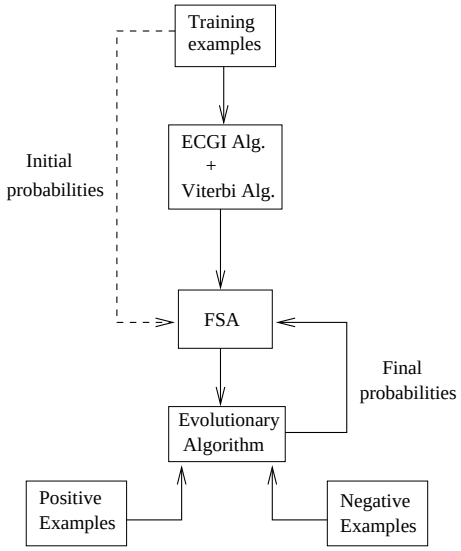


Fig. 1. Scheme of the process to generate the NP flexible recognizer. The FSA is constructed from a set of training examples by applying the ECGI algorithm and using Viterbi to find the most probable sequence of tags for a given input. Initial transition rankings for the FSA are also obtained from this set of training examples. Then, the rankings of the FSA transitions are tuned by applying an evolutionary algorithm which uses new examples, both positive and negative. The result of this process is a FSA which can recognize NPs autonomously.

2.1. *Initial Finite State Automata for NP detection*

The technique used to obtain the FSA is an algorithm of Grammar Inference, i.e. a process which, from a set of examples, obtains structural patterns of the language and represents them by means of a grammar. Different Grammar Inference algorithms have been proposed(Fu and Booth, 1975; Pla et al., 2000; Rulot, 1992), which allow obtaining grammar fragment representations of the language (in this case NPs). Dupont (Dupont, 2002) proposes a general scheme for selecting the most appropriate algorithm of grammar inference under different conditions. According to these ideas, we have chosen the Error Correcting Grammatical Inference (ECGI) algorithm (Rulot and Vidal, 1987), which generates a loop-free FSA. FSA cycles could lead to recognize many wrong cases, thus reducing precision. The ECGI algorithm performs a progressive construction of the FSA from positive examples. The FSA is modified in order to correct the errors appearing in the parsing of a new example, obtaining non recursive grammars (a FSA without cycles).

The algorithm constructs the grammar by adding in an incremental way the training examples. In order to avoid introducing noise in the recognizer which could drive it to increase the number of false positives (non-nominal syntagma s recognized as nominal), it is necessary to remove this noise by applying a filtering step to the training examples. Thus, the typically non-nominal syntagmas which behave as nominal in the training set (the nominalization of a verb, for example) are statistically detected and eliminated from the training list. After the filtering, the algorithm proceeds as follows. First, a simple FSA is generated which recognizes the first example of the training set. This FSA is

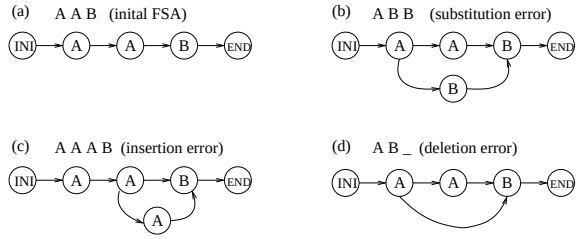


Fig. 2. Examples of application of the ECGI algorithm.

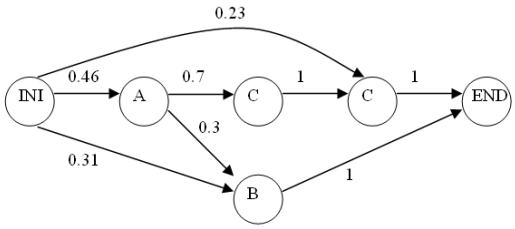


Fig. 3. Probabilistic FSA for flexible NPs recognition.

```
FSA_state ⟵ INI
error_number ⟵ 0
while not empty(sequence) do
  tag ⟵ extract_next_tag(sequence)
  if exists_transition(FSA_state, tag) then
    FSA_state ⟵ transition(FSA_state, tag)
  else
    FSA_state ⟵ highest_rank_trans(FSA_state)
    error_number ⟵ error_number + 1
```

Fig. 4. Scheme of the algorithm applied to process a tag sequence.

then extended with the other examples. To introduce a new example, the Viterbi algorithm is used to find the sequence of states which recognizes the example with least number of errors. Then, the FSA is extended by adding states and/or transitions which correct the errors. In this way, when the process of parsing all the examples finishes, the resulting FSA is able to recognize all of them and does not present cycles.

The errors which can appear between an input sequence and the recognized ones are of three types: insertion, substitution and deletion. Figure 2 shows examples of each of them. When the FSA of Figure 2(a) tries to recognize the input sequence (A B B) (Figure 2(b)), a substitution error is detected and solved by adding a new state to the FSA with the label of the error. If the new sequence is (A A A B) (Figure 2(c)) an insertion error appears, which is solved by adding a new state with the missing label. Finally, if the sequence is (A B) (Figure 2(d)), the deletion error is managed by adding a transition between the states neighbor to the deleted one. The result of the algorithm is a regular grammar which generates a finite language, because by construction it has no cycles.

The selection of the most similar sequence to the input one is carried out by applying the Viterbi (Viterbi, 1967; Forney, 1973) algorithm. This algorithm was initially designed to find the most probable path in a Markov sequence which produces a given sequence of tags (Forney, 1973). This algorithm has also been applied to search the minimum path in a multi-stage graph. In our case, the goal is to minimize the number of errors in the FSA for an input sequence.

2.2. *Flexible Recognition of NPs*

The FSA built so far basically recognizes those NPs present in the training examples (some other sequences can also be recognized, though in general they are not NPs). Because we are interested in a recognizer as general as possible and because the training examples usually include only a small sample of the language, it is necessary to endow the system with a mechanism of generalization. This mechanism amounts to allowing the FSA to recognize a sequence if it is sufficiently similar to an accepted sequence, i.e. if the number of differences is below a certain threshold value. This procedure does not introduces too many sequences outside the language, as the experiments have shown. For example, the FSA of Figure 3 does not recognize the sequence (D C C). However, if the error threshold value is 1, the sequence will be recognized, since the sequence (A C C) is in fact recognized. Obviously, the error threshold value is a determining factor for the generalization of the model, and it is object of a detailed study in the experiments.

Now the question is how to parse a sequence which presents errors. Figure 4 shows a scheme of the way in which a tag sequence is processed, considering the case in which the parse arrives at a situation in which no transition is possible because the next input tag does not match any of the states for which there is a transition. At this point, we use the statistics derived from the training examples. Each transition is labeled with a real value, which represents its ranking in the target language. Then, to process an error tag, the parse follows the most highest ranking transition (*highest_rank_trans*). The transition ranking is relative to each state, in such a way

that the values of all the transitions leaving a same state add up to one. Each value is initially computed as the number of occurrences of a transition relative to the others of the same state according to the training examples. For example, in the FSA of Figure 3, if transition AC appears 7 times in the examples and transition AB appears 3 times, 10 transitions exist from A to C or B, and thus the probability of the edges that leave state A are 0.7(C) and 0.3(B) (AC = 7/10; AB = 3/10). With the probabilities of the edges so determined, if the sequence (D C C) is parsed, as there is not transition from the initial state to D, the parse will take the most probable transition, which leads to A, and afterwards the process continues with the other tags of the input, (C C), which are correct transitions. Thus, the sequence (A C C) is obtained with one error with respect to the input. In this way, the parse produces a sequence very similar to the input one and which occurs with high probability in the language. Thus, we can expect that a sequence which has been only partially recognized, belongs to the language if the error is small. Notice that once the final FSA is generated only sequences whose length differ less than the threshold error can be recognized. However, we have found that it is really infrequent to find base NP sequences with a length different from any of the training examples.

To chunk a text into NPs and non-NPs, sequences of consecutive words in the text are tested with the FSA recognizer, in a constrained exhaustive search. The sequences are constructed progressively, appending the next word in the text to the corresponding sequence under two conditions: first, the sequence must be identified as NP by the FSA (perhaps with some error) and second, for the pair involving the last POS tag of the current sequence and the appended POS tag, the probability of appearance inside an NP must be higher than the probability of appearance outside an NP. If any of the latter conditions is not fulfilled then the current sequence is considered the last FSA output and the next sequence is built from the word that was intended to be appended the last.

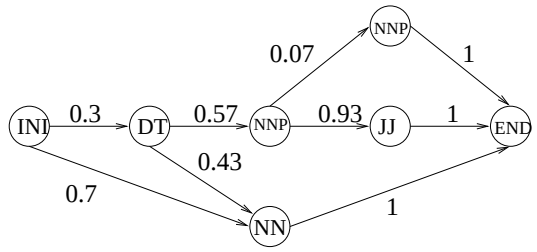
3. Evolutionary Algorithm to Optimize the Automata Probabilities

Now the goal is to find a set of probabilities for the edges of the FSA described in the previous section, which allow it to recognize as many NPs as possible, avoiding at the same time recognizing false NPs. Accordingly, the search space is composed of the different sets of probabilities for the edges of the FSA and the algorithm looks for those which optimize the recognition of a set of training examples. This complex search is performed with an evolutionary algorithm.

Systems based on evolutionary algorithms maintain a population of potential solutions and are provided with some selection process based on the fitness of individuals, as natural selection does. The population is renewed by replacing individuals with those obtained by applying "genetic" operators to selected individuals. The most usual "genetic" operators are *crossover* and *mutation*. Crossover obtains new individuals by mixing, in some problem dependent way, two individuals, called parents. Mutation produces a new individual by performing some kind of random change on an individual. The production of new generations continues until resources are exhausted or until some individual in the population is fit enough. Evolutionary algorithms have proven to be very useful search and optimization methods, and have previously been applied to different issues of natural language processing (Kool, 2000; Araujo, 2004b), such as text categorization (Serrano et al., 2003), inference of context-free grammars (Wyard, 1991; Smith and Witten, 1995), tagging (Araujo, 2002) and parsing (Araujo, 2004a).

In our case, individuals represent probabilistic FSAs, all of them with the same structure but different rankings for their transitions. They are implemented as a transition matrix, what facilitates the application of the genetic operators. Thus, the probability of an edge going from state i to state j is the value which appears in row i column j, A_{ij}. Transitions which do no exist are assigned the value 0. For example, the FSA of Figure 5(a) is implemented as the matrix of Figure 5(b).

The genetic operators applied to renew the population are crossover and mutation. Two variants of crossover have been implemented. The first one is the classic one point crossover (Goldberg, 1989), which mixes two individuals by combining the first part of one parent up to a crossover point with the second part of the other parent and vice versa. The second variant uses two crossover points and exchanges the bit of the two parents between these points. In both variants, the crossover points are randomly selected. In this case, crossover points indicate state levels. The level of a state S is defined as the maximum number of states to reach S from



(a)

	INI	DT	NNP	JJ	NN	NNP	FIN
INI	0	0.3	0	0	0.7	0	0
DT	0	0	0.57	0	0.43	0	0
NNP	0	0	0	0.93	0	0.07	0
JJ	0	0	0	0	0	0	1
NN	0	0	0	0	0	0	1
NNP	0	0	0	0	0	0	1
FIN	0	0	0	0	0	0	0

(b)

Fig. 5. Individuals representation.

the initial state, being the level of this initial state equal to 0. This way, given the transition matrix, we can know which side of the crossover points each transition is at, and thus the corresponding weights can be swapped between two individuals.

The mutation operator amounts to choosing an edge at random and varying its probability. This variation may be positive or negative, what is decided at random, and the amount is an input parameter to the algorithm, studied in the experiments. Notice that when an edge is modified the remaining edges of the same state must be updated in order to maintain the value of its addition equal to 1. If the variation produces a value smaller than zero or greater than one, then the edge is assigned zero or one respectively, and the real variation is computed according to this value.

3.1. *The Fitness Function*

The fitness function must be a measure of the capability of the FSA to recognize NPs and only them. On the one hand, the fitness function must include a measure of the coverage, or *recall* achieved by the individual, i. e. the number of NPs which have been recognized from the set of proposed(true) NPs:

$$recall = \frac{\#\text{ recognized NPs}}{\#\text{ proposed NPs}} \quad (1)$$

where # stands for number of.

On the other hand, the fitness function must also take into account the precision achieved by the individual. The precision metric used in the fitness function, called precision*, is a variation of the standard precision measure. In this case, precision* takes into account not only the correctly identified NPs but also the correctly discarded non-NPs.

$$precision* = \frac{\#\text{ NPs recognized} + \#\text{ non NPs discarded}}{\#\text{ FSA proposed syntagmas}} \quad (2)$$

where proposed NPs stands for the number of syntagma that the FSA recognized as NPs. This special form of precision* was used because prior experiments showed that the consideration of correct negatives in fitness calculation drives the EA to obtain much better individuals. Notice that precision* was only used for EA parameter optimization. All the other results presented in this paper were calculated by using the standard precision measure(NPs correctly recognized/NPs proposed by the FSA), as in the related works. We have considered as fitness function two alternative F-measures which combine these two parameters in different ways:

$$F1\text{-}measure = \frac{2 \cdot \text{precision*} \cdot \text{recall}}{\text{precision*} + \text{recall}} \quad (3)$$

and F2-measure, which gives a higher weight to precision*,

$$F2\text{-}measure = \frac{5 \cdot \text{precision*} \cdot \text{recall}}{4 \cdot \text{precision*} + \text{recall}} \quad (4)$$

In order to apply these measures to an individual, we must establish the cases in which an NP is considered recognized. Obviously those input sequences corresponding to a path from the initial to the final state are recognized by any individual in the population. But in the remaining cases, the path will depend on the transition rankings of the FSA, and only those sequences with a number of errors below the threshold will be recognized. This threshold value can be defined as an absolute value or as a percentage of the length of the sequence. Both alternatives have been evaluated in the experiments.

3.2. *Initial Population*

Individuals for the initial population are randomly generated from a seed solution, which helps

to guide the search. The seed, which is included in the population as another individual, is the set of probabilities obtained from the training examples used to construct the FSA. The individuals of the remaining population are generated by applying the mutation operator several times to the seed. The variation produced by each mutation in this phase is a parameter studied in the experiments. Notice that a small variation would produce individuals too similar to the seed, which in spite of having a high quality according to the fitness function, do not help to explore other areas of the search space where better solutions could be found, while a great variation can lead to lose the advantages of the seed.

4. Filtering the Training Set

Some preliminary experiments have shown that the most frequent errors affecting precision are mainly due to some training examples used in the construction of the FSA in which a sequence of tags, which usually does not correspond to an NP, is working as an NP in that particular case, and also to sequences many of whose tags are not typical NP tags. Specifically, we consider noise the transitions corresponding to a single tag or two consecutive tags (bigram) whose occurrence ratio within the NPs patterns is below a threshold value. Some among such examples are:

JJ VB
JJ
VB
VB NN

where JJ stands for adjective, VB for verb (base form) and NN for noun (singular or mass). It is clear that precision is going to be highly reduced if the FSA recognizes a verb (VB) as an NP because it has appeared once in the training set in the role of a noun.

We have studied mechanisms for automatically filtering those NPs with noise in the training set to improve precision. For example, let us assume that the threshold is set to 50%. If the tag sequence $\langle VB \;\; WJ \rangle$ appears 100 times in the training set and 73 of them belongs to an NP, then it is not considered noise. On the contrary, if only 42% of its occurrences are inside NPs, it is considered noise and every NP containing such a sequence is removed from the training set and do not contribute to the FSA construction.

Obviously, this filtering mechanism, devoted to improve the precision of the system by avoiding false NPs, reduces the system coverage. It is necessary to check out if the precision increase is large enough to compensate the lost of coverage. Results presented in section 5 show that this mechanism provides a significant improvement of the overall performance.

5. Experiments

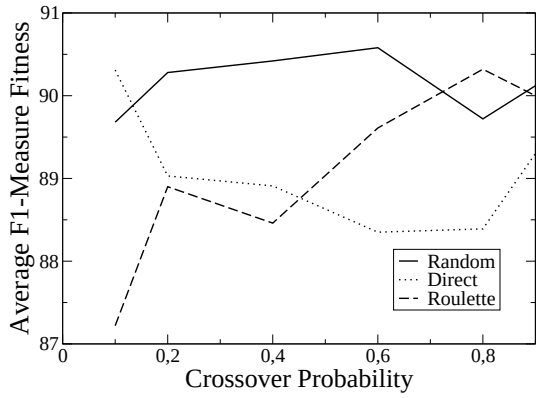
The algorithm has been implemented using C++ language and run on a Pentium III processor. The CPU time spent on generating an automaton from 45 examples, with a maximum length of 7, was 0.3 seconds, and from 156 examples, with 9 as maximum length, 1.6 seconds. The time spent on analyzing a text composed of 1108 different syntagmas, being NPs 570 of them, was 1.1 seconds. For the experiments we have used training and test sets from the Brown corpus portion of the Penn Treebank (Marcus et al., 1994), a database of English sentences manually annotated with syntactic information.

The first kind of the experiments is intended to strengthen the genetic algorithm parameters. Beginning with a baseline configuration, such as 50 generations, population size of 25, seed variation rate of 0.5, simple crossover, mutation probability of 0.1, mutation variation rate of 0.1, F1-measure as fitness function and error threshold standing for 10%, the algorithm was run five times for different crossover probabilities from 0.1 up to 0.9 together with different selection methods, as can be observed in Figure 6(a). We have used the first file of category B of the corpus (627 NPs, 424 non-NPs) to build the FSA and to obtain the initial transition rankings. To calculate the fitness function we used the second and third files of category B (1226 NPs, 944 non-NPs). The graphic shows that the best average and maximum results are obtained by using a crossover probability of 0.6 and random selection. Using this probability value and selection method the algorithm was run five times applying the two crossover types, as seen in Figure 6(b), concluding that simple crossover performs better.

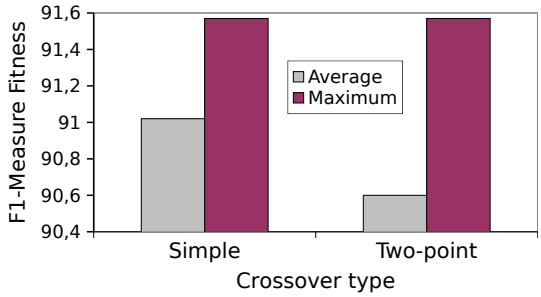
Thus, using the previous values the next parameters to be optimized are mutation related. Analogously, the mutation probability is varied from 0.1 to 0.9 running five times the algorithm for each value. The results are presented in Figure 7(a). As can be

observed best results are obtained between values of 0.4 and 0.6. The latter was the selected value because it presents a better average value. The same process was carried out for the rate of variation produced by the mutation operator. Figure 7(b) shows that the best average and maximum fitness is achieved for a variation of 0.6.

Finally, the number of generations, population size and seed variation rate are settled. Figure 8(a) presents the average and maximum fitness values for five executions of the algorithm using different number of generations and number of individuals. Both number of generations and population size produce better results as they increase until 50 generations and 75 individuals. Greater values than those seem to produce degradation. With respect to seed variation, Figure 8(b) shows that the greater the variation the better the results, meaning that to include individuals very different from the initial seed within the initial population leads the algorithm to better solutions. So, the final configuration of the algorithm, which was used in the experiments above,

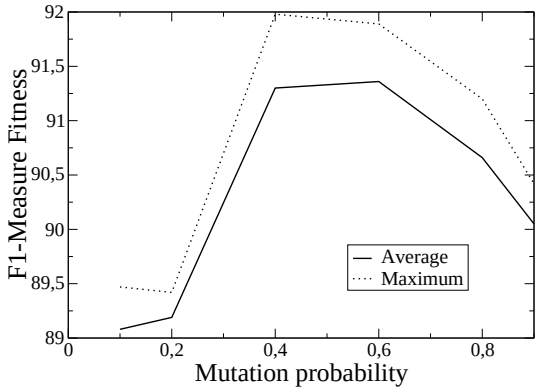


(a)

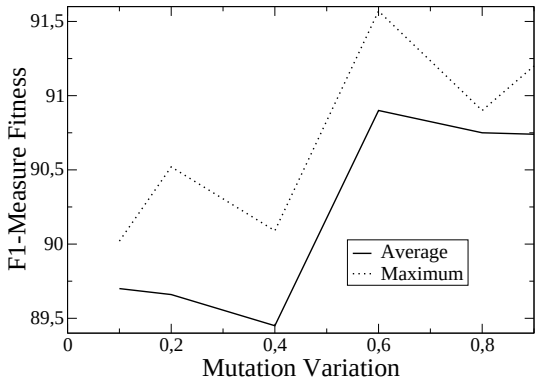


(b)

Fig. 6. (a) Average fitness values of five executions of the EA using different individual selection methods and crossover probabilities. (b) Average and maximum fitness values of five executions of the EA using different types of crossover.



(a)

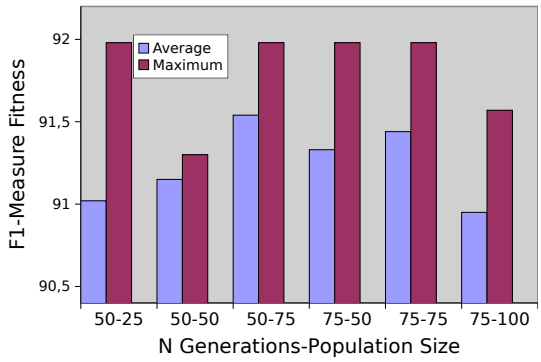


(b)

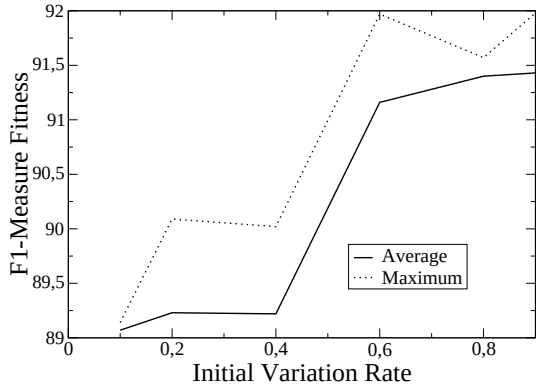
Fig. 7. (a) Average and maximum fitness values of five executions of the EA using different mutation probabilities. (b) Average and maximum fitness values of five executions of the EA using different rates of variation produced by mutation.

is presented in Table 1. We use elitism, a technique of retaining in the population the best individuals found so far. Setting the EA parameters to these values, the time spent on ten executions of the EA was roughly 3 hours and a half, using a test set of 1569 different syntagma, 543 NPs, for the fitness function calculation.

Before any other empirical trials, we have carried out a test in order to show the benefits of the EA over the recognizer, by testing on the files 4 to 12 of category B of the corpus (4034 NPs, 3422 non-NPs). Figure 9 shows the results obtained. We have found out that the higher the error threshold the better the improvement. The reason is that higher error thresholds require more accurate transition rankings, as those provided by the EA. For relative error thresholds of 10%, 20% and 30% the F1-measure over the test increases 4%, 10% and 14%, respectively. For absolute error threshold values of 1 and



(a)



(b)

Fig. 8. (a) Average and maximum fitness values of five executions of the EA using different number of generations and population sizes. (b) Average and maximum fitness values of five executions of the EA using different rates of variation used to generate the initial population by modifying the seed.

Number of generations	50
Population size	75
Initial variation	0.9
Selection	Random
Crossover type	Simple
Crossover probability	0.6
Mutation variation	0.6
Mutation probability	0.6
Fitness function	F1-Measure

Table 1
EA optimum parameters.

2 the increase is about 5% and 12%, respectively.

We also have performed experiments to determine the most appropriate error threshold allowed in the recognition. Figure 10 shows the results obtained with different performance measures over the 74 first

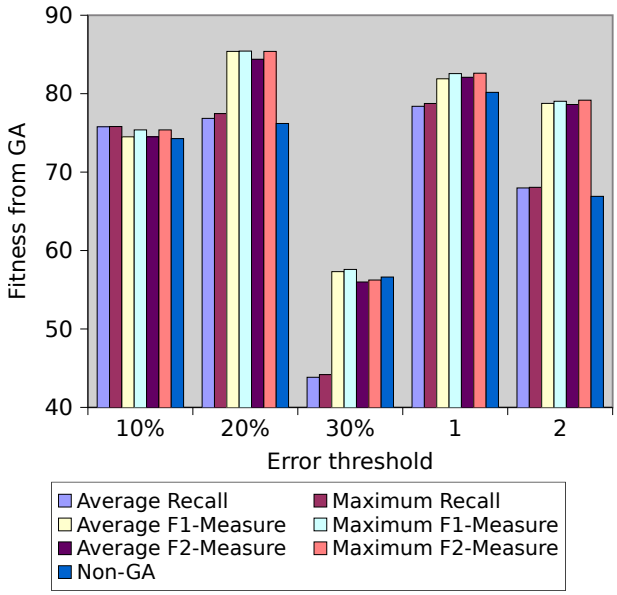


Fig. 9. Improvement achieved by using a FSA optimized with the evolutionary algorithm.

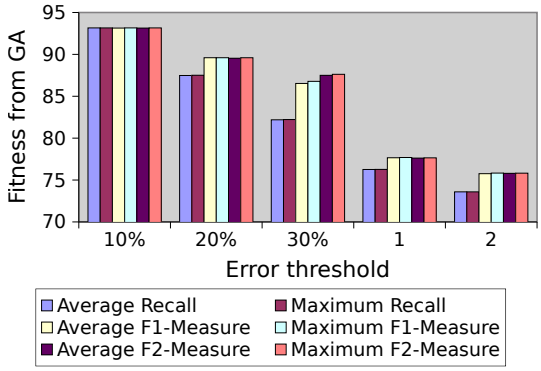


Fig. 10. Error thresholds and fitness function types comparison.

files of the category J of the corpus (44,479 NPs and 35,712 non-NPs). Two different ways of defining the threshold value have been studied: as an absolute number of errors, and as a percentage of the NP length. Best results are obtained with the threshold defined as a percentage and for relative low values, such as 10%. This value has been used in the following experiments.

Figure 11 shows the comparison, for different error thresholds, of results for absolute number of syntagmas, i.e. each different appearance of a syntagma is considered, and relative numbers, i.e. only one appearance of each type of syntagma is considered. As can be observed, the results are better for absolute number of syntagmas, which means that the correctly identified syntagmas are the most fre-

quent and the mistakes have a very low number of appearances, being the most significant at 30% error threshold. However, this is not the case for error thresholds values defined as absolute number of errors (1 and 2), mainly because in these cases there is a larger number of wrong recognitions of non-nominal examples with a high frequency. This fact boosts the preference for relative error thresholds over absolute error thresholds.

Another question which has been investigated is the most appropriate size of the training sets for both, the construction of the FSA (C), and for the evaluation of fitness measure of the EA (F). Figure 12 shows the results of the different considered measures for different combination of these values (C-F), using disjoint combinations of the first 16 files of the category B of the corpus (10,095 NPs and 7761 non-NPs). Best results for all measures are obtained when using a small number of files for the FSA construction, and a large number of files for the EA. Using a larger training set in the construction of the FSA produces too large FSAs, which recognize too much false NPs and deteriorate the system performance. We can in fact observe that the recall measures are high when using a larger set in the construction of the FSA, at the price of producing very low precision measures.

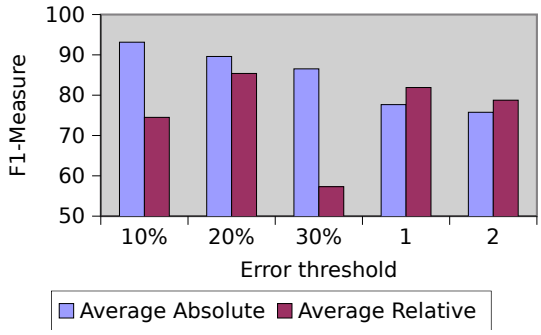


Fig. 11. F-measure values obtained over a test set for different error thresholds and for relative and absolute number of syntagmas.

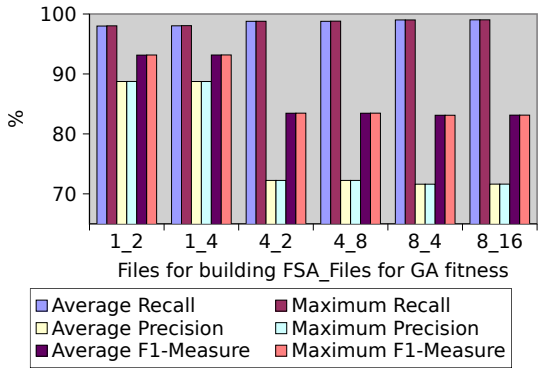


Fig. 12. Accuracy values for different combinations of number of files for building the recognizer-number of files for EA fitness.

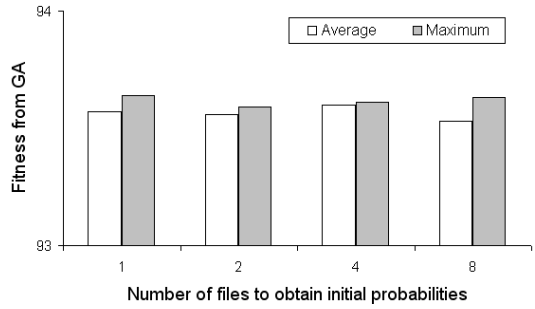


Fig. 13. F1-measures values from the GA using different number of files to obtain the initial transition rankings or seed.

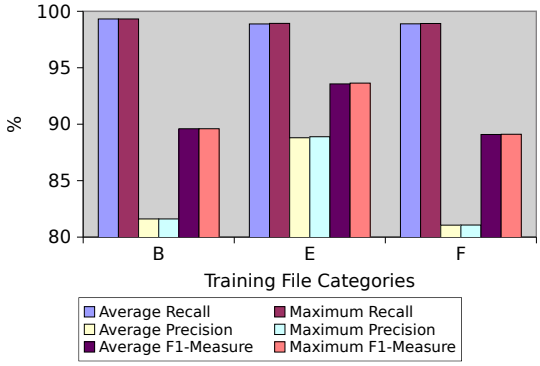


Fig. 14. Accuracy values for different categories of the file used to build the recognizer.

Since the system also uses training examples to establish the initial transition rankings from which the initial population of the EA is generated, we have performed a test in order to study the influence of the number of examples used this way in the EA performance, using subsets of the first 8 files in E category of the corpus (3560 NPs, 2505 non-NPs) and tested on the first 74 files in category J. As observed in Figure 13 there exists an improvement the more examples are used, but it is not very significant. However, the decrease in performance when using eight files indicates that it is not convenient to use too many training examples, because the extracted transition rankings become overfitted.

We have also performed experiments to determine the impact of using different categories of topics as training sets. Figure 14 shows the results obtained using training files of category B (press editorial), E (skills and hobbies) and F (popular lore) from the

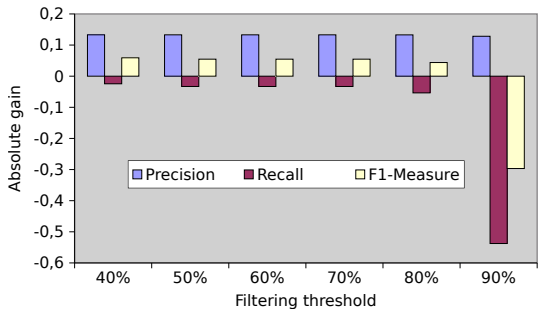


Fig. 15. Accuracy values for different values of the filtering threshold applied to the training NP set.

Penn Treebank, that we are using in the experiments (the first 3 files of each one). We used the first 3 files of each category: the first file to build the FSA and set initial transition rankings (600 NPs and 500 non-NPs approximately), and the other two for the fitness function calculation (about 1200 NPs and 1000 non-NPs). The algorithm was tested on the first 74 files of category J, as above. We can observe that the results are quite different depending on the used category, since the kind and frequency of NPs is different in each of them.

We have tested the effect of applying the filtering mechanism to the set of training examples. In this experiment we use 40 documents from the standard subset of the Wall Street Journal portion of the Penn Treebank (Ramshaw and Marcus, 1995), 10 of each training category (from categories "15" to "18"), and all the files of the test category "20" for testing. The automata transition rankings are obtained with the best values of the parameters according to the results of the experiments described above. Figure 15 shows the improvement in precision, recall and F1-measure achieved by introducing this filter to detect the NPs in the test set with different threshold values for filtering. We can observe that for every filtering threshold, except 90%, recall decreases about 2.5% because the absence of the filtered NPs in the training set makes it impossible to recognize NPs similar to them. However, precision increases more than 10% for any filtering threshold. Because this improvement is larger than the decrease of recall, the F1-measure also increases in more than 5%, except for a filtering threshold of 90%. In this case the amount of excluded NPs yields a huge decrease in recall and thus in F1-measure too. Accordingly, taking an intermediate value, such as 50%, for the filtering threshold, we obtain values of 97.79%, 95.92% and 96.84% for precision, recall and F1-measure, respectively, which represents a neat improvement.

Table 2
Comparison of precision(Pr), recall(Rc) and F1-measure(F) values with similar systems on the Wall Street Journal standard portion.

	Pr(%)	***Rc(%)***	***F(%)***
[**Ramshaw95**]	91.8	92.3	92.0
[**Cardie98**]	90.7	91.1	90.9
[**Veenstra98**]	89.0	94.3	91.6
[**Argamon98**]	91.6	91.6	91.6
[**Muñoz99**]	92.4	93.1	92.8
[**Tjong-Kim-Sang00**]	93.6	92.9	93.3
[**Kudo00**]	93.7	94.0	93.9
[**Kudo01**]	94.2	94.3	94.2
[**Shen05**]	95.1	95.3	95.2
[**This work**]	97.8	95.9	96.8

In order to compare the overall performance with other related systems, we have applied the system to the above mentioned standard subset of the Wall Street Journal portion of the Penn Treebank (used in CoNLL-2000 shared task), with the same POS tags used in the other works. We used the bracket representation since experiments carried out in (Muñoz et al., 1999) show an improvement of this formalism over the IOB representation of (Ramshaw and Marcus, 1995). Table 2 compares the precision, recall and F1-measure values. We can observe that our results improve on those of all the other systems, either in recall or precision and F1-measure, only the system of (Shen and Sarkar, 2005) reaching a very similar recall. The results of our system are even better when experiments are performed using the later version of 1995 of the Penn Treebank, in which many errors have been corrected. Using this corpus the system reaches a precision rate of 99.31%, a recall rate of 97.03% and an F-measure rate of 96.84 %. It proves the usefulness of the mechanism of flexible recognition, since it achieves, as expected, a significant improvement of the performance, maintaining at the same time a high level of precision thanks to the accurate transition rankings that the EA provides as well as to the filtering process, providing nearly optimal values for the test set.

6. Discussion and Future Work

We have presented a scheme for the detection of NPs in unrestricted texts. The process relies on different techniques, some of which aim to enlarge the

coverage of the NP detection mechanism, while others aim to improve the precision, excluding the detection of false NPs. The result of training the system with positive and negative NP examples is a probabilistic FSA. This is able to detect a wide range of NPs thanks to the flexible recognition mechanism which allows recognizing NPs similar enough to an accepted one. However the system can also provide a very high precision thanks to the accurate transition rankings assigned to the FSA by the evolutionary algorithm.

We have carried out experiments which show the important improvement in performance achieved by using the accurate set of transition rankings provided by the EA, instead of the initial set of probabilities. We have also studied different criteria to consider an input sequence similar enough to an NP to be recognized. The best results are obtained when the threshold value for the number of allowed errors is defined as a fraction of the input sequence length and for relative low values, such as 10 %. Other experiments have been carried out to determine the most appropriate size of the training sets for both, the construction of the FSA, and the evaluation of the fitness measure of the EA. The best results are obtained when using a small number of files for the FSA construction, and a large number of files for the EA. Using a larger training set in the construction of the FSA produces too large FSAs, which recognize too many false NPs.

Experimental results have shown that the mechanism for the flexible recognition of NPs clearly improves the coverage of the system, while the fitted rankings for the FSA transitions provided by the EA yield a high level of precision, thus improving the overall performance. The precision of the FSA is further improved by applying the filtering mechanism to the training set in order to eliminate those sequences of tags which frequently lead the system to accept false NPs. Results obtained by testing the system on the same set of texts as other systems did, improve the overall performance of all of them, reaching a very high level of accuracy.

Other approaches, such as those used in sequence alignment, could be used to find the sequence most similar to a given one. However the complexity of such approaches would be much higher than that of our approach, which despite its simplicity, definitely improves the results obtained with alternative methods.

We believe that the accurate detection of NPs provided by our system can be useful in information retrieval tasks. Accordingly, for the future we plan to test the application of the system to tasks such as document indexing and web search.

References

Araujo, L., 2002. 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. Springer-Verlag, pp. 230–239.

Araujo, L., 2004a. A probabilistic chart parser implemented with an evolutionary algorithm. In: Proc. of the Int. Conf. on Intelligent Text Processing and Computational Linguistics (CICLing-2004), Lecture Notes in Computer Science 2276. Springer-Verlag, pp. 81–92.

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

Argamon, S., Dagan, I., Krymolowski, Y., 1998. A memory-based approach to learning shallow natural language patterns. In: Proc. of joint International Conference COLING-ACL. pp. 67–73.

Boguraev, B., Kennedy, C., 1997. Salience-based content characterisation of text documents.

Bourigault, D., 1992. Surface grammatical analysis for the extraction of terminological noun phrases. In: Proc. of the Int. Conf. on Computational Linguistics (COLING-92). pp. 977–981.

Brill, E., 1995. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. Computational Linguistics 21 (4), 543–565.

Cardie, C., Pierce, D., 1998. Error-driven pruning of treebank grammars for base noun phrase identification. In: Proc. of COLING-ACL'98. pp. 218–224.

Church, K. W., 1988. A stochastic parts program and noun phrase parser for unrestricted text. In: Proc. of 1st Conference on Applied Natural Language Processing, ANLP. pp. 136–143.

Dupont, P., 2002. Inductive and statistical learning of formal grammars. Tech. rep., Reseach talk, Departement ingenerie Informatique, Universite Catholique de Louvain.

Forney, G. D., 1973. The viterbi algorithm. Proceedings of The IEEE 61 (3), 268–278.

Fu, K., Booth, T., 1975. Grammatical inference: in-

troduction and survey. parts i and ii. IEEE Transactions on Systems, Man and Cybernetics. SMC-5 (4), 409–423.

Furnas, G. W., Landauer, T. K., Gomez, L. M., Dumais, S. T., 1987. The vocabulary problem in human-system communication. Communications of ACM 30 (11), 964–971.

Goldberg, D. E., 1989. Genetic Algorithms in search, optimization and machine learning. Addison Wesley.

Justeson, J., Katz, S., 1995. Technical terminology: some linguistic properties and an algorithm for identification in text. Natural Language Engineering 1, 9–27.

Koehn, P., Och, F. J., Marcu, D., 2003. Statistical phrase-based translation. In: HLT-NAACL. pp. 48–54.

Kool, A., 2000. Literature survey. Tech. rep., Center for Dutch Language and Speech. University of Antwerp.

Kudo, T., Matsumoto, Y., 2000. Use of support vector learning for chunk identification. In: Proceedings of the 4th Conference on CoNLL-2000 and LLL-2000. pp. 142–144.

Kudo, T., Matsumoto, Y., 2001. Chunking with support vector machines. In: NAACL'01: North American Chapter of the Association for Computational Linguistics on Language technologies 2001. Association for Computational Linguistics, Morristown, NJ, USA, pp. 1–8.

Lapata, M., Lascarides, A., 2003. Detecting novel compounds: The role of distributional evidence. In: EACL. pp. 235–242.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., 1994. Building a large annotated corpus of english: The penn treebank. Computational Linguistics 19 (2), 313–330.

Muñoz, M., Punyakanok, V., Roth, D., Zimak, D., 1999. A learning approach to shallow parsing. In: Proc. of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP-VLC'99). Association for Computational Linguistics, pp. 168–178.

Pla, F., 2000. Etiquetado léxico y análisis sintáctico superficial basado en modelos estadísticos. Tesis doctoral en informática, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia.

Pla, F., Molina, A., Prieto, N., 2000. Tagging and chunking with bigrams. In: Proc. of the 17th conference on Computational linguistics. pp. 614–620.

Ramshaw, L., Marcus, M., 1995. Text chunking using transformation-based learning. In: Proc. of the third Workshop on Very Large Corpora (ACL). pp. 82–94.

Rulot, H., 1992. Un algoritmo de inferencia gramatical mediante correcci'on de errores. Tech. rep., Facultad de Ciencias Físicas, Universidad de Valencia, Phd Thesis.

Rulot, H., Vidal, E., 1987. Modelling (sub)string-length-based constraints through a grammatical inference method. In: Pattern Recognition: Theory and Applications. Springer-Verlag, pp. 451–459.

Serrano, J., Castillo, M. D., Sesmero, M., 2003. Genetic learning of text patterns. In: Proc. of CAEPIA03. pp. 231–234.

Sha, F., Pereira, F., 2003. Shallow parsing with conditional random fields. In: NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology. Association for Computational Linguistics, Morristown, NJ, USA, pp. 134–141.

Shen, H., Sarkar, A., 2005. Voting between multiple data representations for text chunking. In: Advances in Artificial Intelligence: Proceedings of the Eighteenth Meeting of the Canadian Society for Computational Intelligence, Canadian AI 2005. Springer, pp. 389–400.

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

Stevenson, M., 2003. Word Sense Disambiguation: The Case for Combinations of Knowledge Sources. CSLI.

Tjong-Kim-Sang, E. F., 2000. Noun phrase representation by system combination. In: Proc. of ANLP-NAACL. pp. 50–55.

Veenstra, J., 1998. Fast np chunking using memory-based learning techniques. In: Proc. of BENELEARN-98: Eighth Belgian-Dutch Conference on Machine Learning. pp. 71–78.

Viterbi, A. J., 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory 13 (2), 260–269.

Voutilainen, A., 1993. Nptool, a detector of english noun phrases. In: Proc. of the Worshop on Very Large Corpora (ACL). pp. 48–57.

Wyard, P., 1991. Context free grammar induction

using genetic algorithms. In: Proc. of the 4th Int. Conf. on Genetic Algorithms. pp. 514–518.

Zhang, T., Damerau, F., Johnson, D., 2002. Text chunking based on a generalization of winnow. J. Mach. Learn. Res. 2, 615–637.