Evolutionary Algorithm for Noun Phrase Detection in Natural Language Processing

J. Ignacio Serrano
Insitituto de Automática Industrial, CSIC.
Ctra. Campo Real, km 0.200
Arganda del Rey. 28500 Madrid (Spain)
nachosm@iai.csic.es

Lourdes Araujo
Departamento de Sistemas Informáticos
y Programación, UCM.
C/ Prof. José García Santesmases, s/n.
28040 Madrid (Spain)
lurdes@sip.ucm.es

Abstract- Noun phrases of a document usually are the main information bearers. Thus, the detection of these units is crucial in many applications related to information retrieval, such as collecting relevant documents by search engines according to a user query, text summarizing, etc. We present an evolutionary algorithm for obtaining a probabilistic finite-state automaton, able to recognize valid noun phrases defined as a sequence of lexical categories. This approach is highly flexible in the sense that the automaton is able to recognize noun phrases similar enough to the ones given by the inferred noun phrase grammar. This flexibility can be allowed thanks to the very accurate set of probabilities provided by the evolutionary algorithm. It works with both, positive and negative examples of the language, thus improving the system coverage, while maintaining its precision. Experimental results show a clear improvement of the performance with respect to others systems.

1 Introduction

The amount of text data in electronic form which appears every day overwhelms the human capacity to extract the needed information. This has lead 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 [11]. Noun phrases (NP), i.e. units whose head or principal word is a noun, are the main bearers of the content of a document. 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 parsing, text summarizing, machine translation, 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 [4] uses a handcrafted NP grammar along with some heuristics for identifying NPs of maximal length, and Voutilainen [24] uses a constraint grammar formalism. Other proposals follow a learning approach based on the use of corpora. Church [7] uses a probabilistic model automatically trained on the Brown corpus to detect NPs as well as to assign parts of speech. Ramshaw and Marcus [16] use the supervised learning methods proposed by Brill [5] to learn a set transformation rules for the problem. Pla and Prieto [15] use grammatical inference to obtain a finite-state automaton (FSA) which recognizes NPs. Their method has inspired this work.

We present an evolutionary algorithm (EA) to implement a new model for a *flexible* identification of basic (nonrecursive) noun phrases in an arbitrary text. EAs have previously been applied to different NLP issues [12, 2], such as text categorization [19], inference of context-free grammars [25, 20], and parsing [1]. EAs allow dealing with the large search spaces of some complex tasks appearing in NLP whose complexity frequently increases due to the presence of ambiguity. The EA that we introduce to recognize NPs produces a probabilistic FSA. Correct NPs are defined as sequences of lexical categories, usually called part-of-speech (POS) tags. Thus, "DT NN", "DT JJ NN" are examples of NPs, where DT stands for *determiner*, NN for *noun* and JJ for *adjective*. The FSA is initially generated with the Error Correcting Grammatical Inference (ECGI) algorithm of grammatical inference, and initial probabilities are assigned using the collected bigram probabilities, i.e. conditional probabilities of pairs of consecutive lexical tags assigned to the words of the training text. The FSA probabilities provide a method for a flexible recognition of input chains, 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. This flexibility can be allowed thanks to the very accurate set of probabilities provided by the EA. It works with both, positive and negative examples of the language, thus improving the system coverage, while maintaining its precision.

Though it is difficult to compare different approaches to NP detection because they differ in multiple elements, some attempts have been made. Pla [14] gathers results of a number of parses trained on the data set used by Ramshaw [16]: the one proposed by Cardie & Pierce [6], 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 [23], 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

[3], which learns sequences of POS tags and brackets, and the hybrid approach of Tjong-Kim-Sang [21], 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. The evaluation of different aspects of the system, and a comparison of the overall performance with other systems is presented in Section 4. Finally, conclusions are drawn in Section 5.

2 Dynamics Overview

This work presents the design and implementation of a flexible recognizer of basic NPs given as chains of POS tags. This recognizer is a probabilistic FSA constructed from examples of objective syntagmas with an algorithm of grammar inference will be used (ECGI). In its turn, this algorithm uses the Viterbi algorithm, a dynamic programming one, so that the FSA has an optimal number of states and paths. In order to extend the recognition capabilities of the FSA beyond the set of training examples, it is necessary to endow 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 with a percentage of error. If the number of errors incurred in parsing an NP is below a threshold value, then the new NP is also recognized. An error appears whenever the input tag does not produce any transition from the actual state of the FSA. In order to allow the FSA to find NPs similar to those of the input, their edges are labeled with the probability of each transition between tags within an NP. These probabilities are initially extracted from the NPs examples, and afterwards an evolutionary algorithm is applied to optimize the probabilities. 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 flexible recognizer. Let us now describe the different elements of the system.

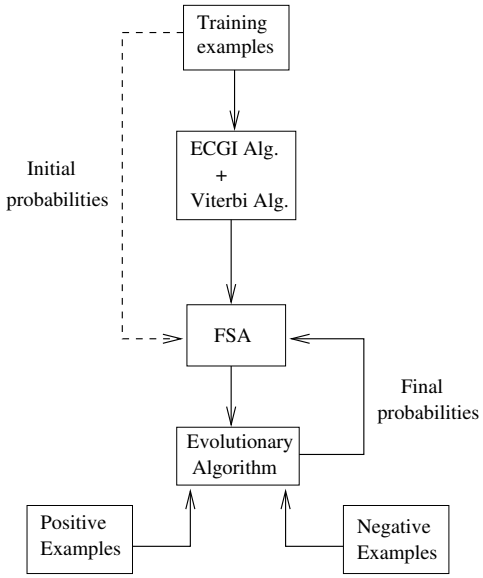


Figure 1: Scheme of the process to generate the NP flexible recognizer. The FSA is constructed from a set of training examples by applying the ECGI algorithm and using Viterbi to find the most similar chain of tags for a given input. Initial probabilities for the FSA are also obtained from this set of training examples. Then, the probabilities are tuned by applying an evolutionary algorithm which uses new examples, both positive and negative.

2.1 Construction of the Initial Finite State Automata

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[10, 15, 17], which allow obtaining grammar fragment representations of the language (in this case basic NPs). Dupont [8] 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 [18] since we are interested in a heuristic construction of a regular grammar such that the final result allows for a flexible recognition. This technique involves 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 syntagmas recognized as nominal), it is necessary to remove this noise by applying a preprocessing 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 handily detected and eliminated from the training list. After the preprocessing, the algorithm proceeds as follows. First, a simple FSA is generated which recognizes the first example of the training set. This FSA is 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 less errors. Then, the FSA is extended by adding states and/or transitions which correct the produced 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 chain and the recognized ones are of three types: insertion, substitution and deletion. Figure 2 shows examples of each of them.

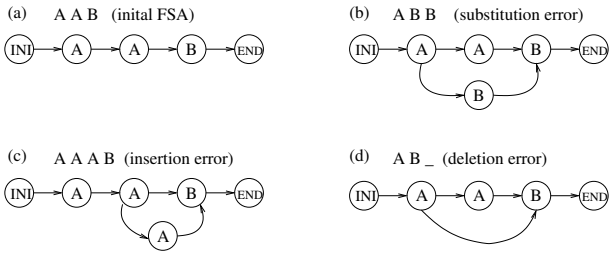


Figure 2: Examples of application of the ECGI algorithm.

When the FSA of Figure 2(a) tries to recognize the input chain (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 chain 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 chain 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.

We have introduced a simplification of this algorithm already proposed in the literature [22]. It aims at reducing the complexity of the algorithm to find in the FSA the closest path to the input chain. This simplification amounts to ignoring the deletion errors. This can reduce the recognition complexity below 10%. However, this simplification also reduces the generality of the language, since it discards chains with lengths different from the training examples. Thus, these examples must be sufficiently varied and heterogenous.

The selection of the most similar chain to the input one is carried out by applying the Viterbi algorithm. This algorithm was initially designed to find the most probable path in a Markov chain which produces a given sequence of tags [9]. 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 chain.

The algorithm uses a trellis diagram, which is a weighed multi-stage graph. Accordingly, the problem is to search the minimum path in a trellis. For a given FSA and an input chain, the number of stages of the corresponding trellis is the length of the input chain. The number of states of each stage is the number of the states in the FSA. Edges from a state to other in the following stage only exists if the transition between those two states is present in the FSA. Each edge of the trellis is weighed with the cost of the transition (1 or 0 according to whether it produces an error or not).

We have modified the Viterbi algorithm to process input chains with a length different from the examples chains. Thus, if the input chain is longer than the longest chain the FSA recognizes, there is no path in the trellis from the initial state to the end state, since it does not exists in the FSA either. Thus, the trellis is extended with edges which do not exist in the FSA and which introduce cycles.

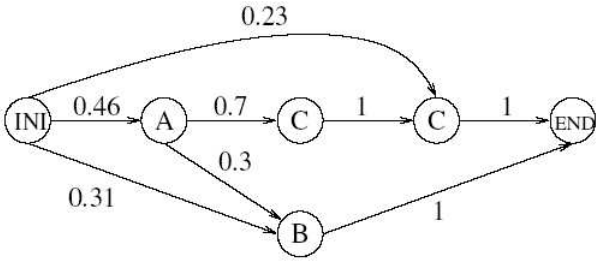
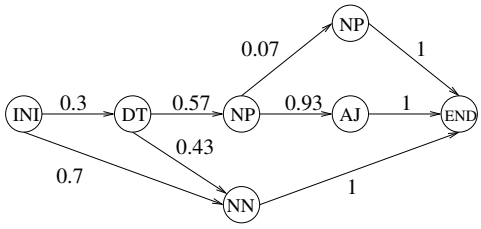


Figure 3: Probabilistic FSA for flexible NPs recognition.

2.2 Endowing the FSA with Generalization: Allowing Errors

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 chain if it is sufficiently similar to an accepted chain, i.e. if the number of differences is below a certain threshold value. This procedure does not introduces too many chains outside the language, as the experiments have shown. For example, the FSA of Figure 3 does not recognize the chain (D C C). However, if the error threshold value is 1, the chain will be recognized, since the chain (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 chain which presents errors. In this case, the parse arrives at a situation in which no transition is possible, since the next input tag does not matches any of the state 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 probability in the target language. Then, to process an error tag, the parse follows the most probable transition. The probabilities are relative to each state, in such a way that the values of all the transitions leaving a same state add up to one. Each probability 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 chain (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 chain (A C C) is obtained with one error with respect to the input. In this way, the parse produces a chain very similar



(a)

	INI	DT	NP	AJ	NN	NP	FIN
INI	-1	0.3	-1	-1	0.7	-1	-1
DT	-1	-1	0.57	-1	0.43	-1	-1
NP	-1	-1	-1	0.93	-1	0.07	-1
AJ	-1	-1	-1	-1	-1	-1	1
NN	-1	-1	-1	-1	-1	-1	1
NP	-1	-1	-1	-1	-1	-1	1
FIN	-1	-1	-1	-1	-1	-1	-1

(b)

Figure 4: Individuals representation.

to the input one and which occurs with high probability in the language. Thus, we can expect that a chain which has been only partially recognized, belongs to the language if the error is small.

3 Evolutionary Algorithm Design

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.

In our case, individuals represent probabilistic FSAs, all of them with the same structure but different probabilities. They are implemented as an adjacency matrix, what facilitates the application of the genetic operators. Thus, the probability 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 -1. For example, the FSA of Figure 4(a) is implemented as the matrix of Figure 4(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, which combines 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.

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 the zero or greater than one, then the edge is assigned zero or one respectively, and the real variation is computed according to this value. Both operators have associated a application rate which stands for the probability of each individual to be selected by operators.

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 NPs:

$$recall = \frac{\text{number of recognized NPs}}{\text{number of proposed NPs}} \tag{1}$$

On the other hand, the fitness function must also take into account the precision achieved by the individual:

$$precision = \frac{\text{number of NPs recognized} + \text{number of non NPs discarded}}{\text{number of proposed syntagmas}} \tag{2}$$

We have considered as fitness function two F-measures which combine these two parameters in different ways:

$$Fmeasure = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

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

$$F2measure = \frac{5 \cdot \text{Precision} \cdot \text{Recall}}{4 \cdot \text{Precision} + \text{Recall}} \tag{4}$$

In order to apply these measures to an individual, we must establish the cases in which an NP is considered recognized. Obviously those input chains 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 probabilities of the FSA, and only those chains 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 chain. 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.

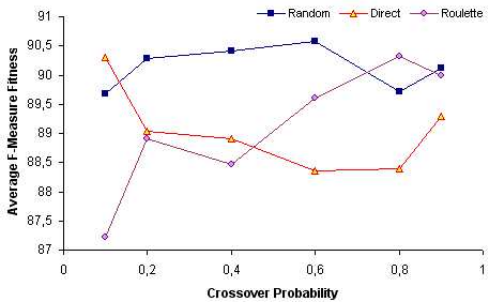
4 Experiments

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 [13], a database of English sentences manually annotated with syntactic information.

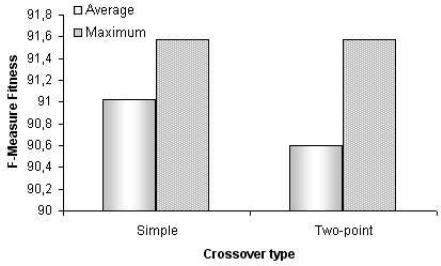
The first kind of the experiments is intended to strengthen the algorithm parameters. Beginning with a baseline configuration, such as 50 generations, population size of 25, seed variation rate of 0.5, simple crossover, mutation probability of 0.1, mutation variation rate of 0.1, F-measure as fitness 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 5(a). The graphic shows that the best average 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 5(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 6(a). As can be observed best results are obtained between values of 0.4 and 0.6. The former 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 6(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 7(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 the formers seem to produce degradation. Respecting to seed variation, Figure 7(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, is presented in Table 1. 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 syntagmas, 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. We have found out that the higher the error threshold the better the improvement. The reason is that higher error thresholds require more accurate probabilities, as those

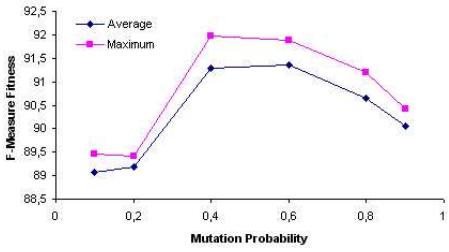


(a)

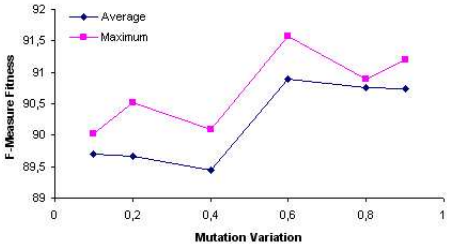


(b)

Figure 5: (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.

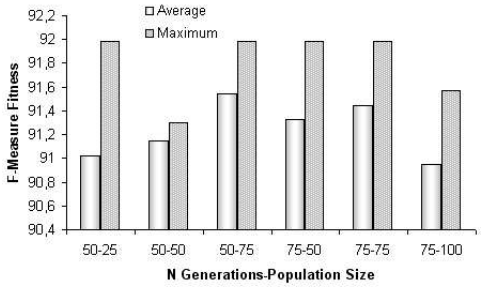


(a)

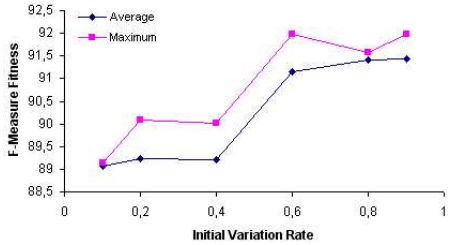


(b)

Figure 6: (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.



(a)



(b)

Figure 7: (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.

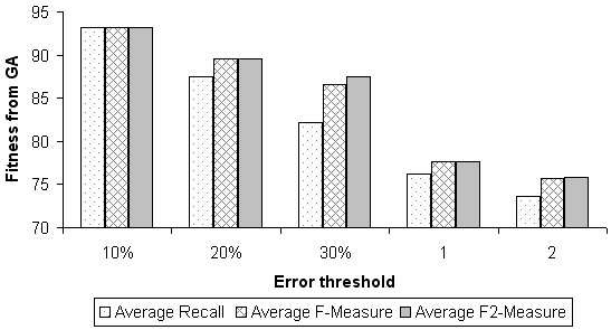
Number of generations	50
Population size	75
Initial variation	0.9
Selection	Random
Crossover type	Simple
Crossover probability	0.6
Mutation variation	0.6
Mutation probability	0.6
Fitness function	F1-Measure
Elitism[1]	Yes

Table 1: EA optimum parameters.

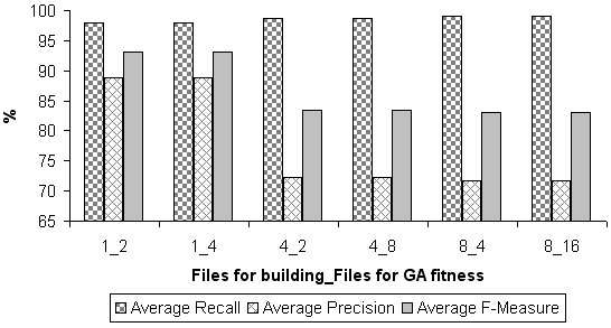
provided the EA. For relative error thresholds of 10%, 20% and 30% the F-Measure over the test increases 4%, 10% and 14%, respectively. For absolute error threshold values of 1 and 2 the increase is about 5% and 12%, respectively.

We also have performed experiments to determine the most appropriate error threshold allowed in the recognition. Figure 8 (a) shows the results obtained with different performance measures. 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.

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 8 (b) shows the results of the different considered measures for different combination of



(a)



(b)

Figure 8: (a) Error thresholds and fitness function types comparison. (b) F-Measure values for different combinations of number of files for building the recognizer-number of files for EA fitness.

these values (C-F). 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.

Since the system also uses training examples to establish the initial probabilities 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. As observed in Figure 9 (a), 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 probabilities become overfitted.

We have also performed experiments to determine the impact of using different categories of topics as training sets. Figure 9 (b) shows the results obtained using training files of category B (press editorial), E (skills and hobbies) and F (popular lore). 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. In order to compare the overall performance with other related

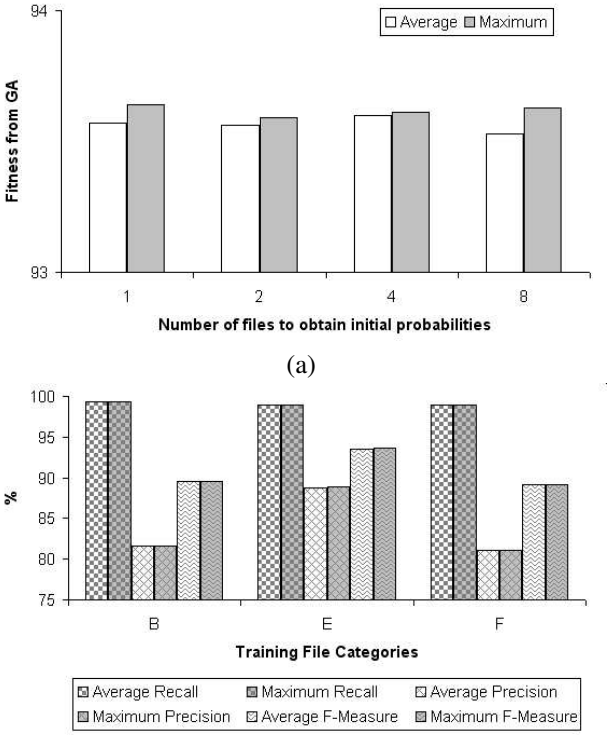


Figure 9: (a) F-measures values from the GA using different number of files to obtain the initial probabilities or seed. (b) Accuracy values for different categories of the file used to build the recognizer.

Table 2: Comparison of precision(Pr), recall(Rc) and F-measure(F) values with similar systems on the Ramshaw standard corpus.

	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
[Tjong-Kim-Sang00]	93.6	92.9	93.3
[This work]	91.6	97.8	94.5

systems, we have applied the system to a standard subset of the Wall Street Journal portion of the Penn Treebank, also used in the other works. Table 2 compares the precision, recall and F-measure values. We can observe that our results improve on those of all the other systems, both in recall and F-measure, and only the Ramshaw95 and Tjong-Kim-Sang00 systems provide a slightly better precision. 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 probabilities that the EA provides.

5 Discussion and Future Work

This paper describes an EA designed to optimize the probabilities of a probabilistic FSA able to detect NPs in arbitrary texts, what is very useful in different tasks of information retrieval and natural language processing. The EA optimizes the probabilities with respect to both, positive and negative training examples. The accurate probabilities provided this way, allow us to define a highly flexible mechanism for the recognition: an input chain of lexical categories is considered an NP if it is similar enough to an accepted one. Experimental results have shown that this mechanism for the flexible recognition of NP clearly improves the coverage of the system, while the fitted probabilities provided by the EA yield a high level of precision, thus improving the overall performance. Results obtained by testing the system on the same set of texts that other systems improve the overall performance of all of them.

We have carried out experiments which show the important improvement in performance achieved by using the accurate set of probabilities provided by the EA, instead of the initial set of probabilities. We have also studied different criteria to consider an input chain 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 percentage of the input chain 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 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.

For the future, we plan to investigate different issues which can improve the system. We will study the possibility of introducing cycles in the construction of the FSA, and also different ways of further improving the performance of the EA, such as the definition of other genetic operators and fitness functions for better precision/recall balance. We have observed that the most frequent errors affecting precision are mainly due to some training examples used in the construction of the FSA, that can be viewed as noise data, 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 in which many of their tags are

not typical NP tags. Accordingly, we are also investigating mechanisms for automatically filtering the training sets to improve precision.

Acknowledgments

This work has been partially supported by projects TIC2003-09481-C04 and FIT150500-2003-373.

Bibliography

[1] L. Araujo. 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*, pages 81–92. Springer-Verlag, 2004.

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

[3] S. Argamon, I. Dagan, and Y. Krymolowski. A memory-based approach to learning shallow natural language patterns. In *Proc. of joint International Conference COLING-ACL*, pages 67–73, 1998.

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

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

[6] C. Cardie and D. Pierce. Error-driven pruning of treebank grammars for base noun phrase identification. In *Proc. of COLING-ACL'98*, pages 218–224, 1998.

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

[8] P. Dupont. Inductive and statistical learning of formal grammars. Technical report, Reseach talk, Departement ingenerie Informatique, Universite Catholique de Louvain, 2002.

[9] G. D. Forney. The viterbi algorithm. *Proceedings of The IEEE*, 61(3):268–278, 1973.

[10] K. Fu and T. Booth. Grammatical inference: introduction and survey. parts i and ii. *IEEE Transactions on Systems, Man and Cybernetics.*, SMC-5(4):409–423, 1975.

[11] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987.

[12] Anne Kool. Literature survey. Technical report, Center for Dutch Language and Speech. University of Antwerp, 2000.

[13] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[14] F. Pla. Etiquetado léxico y análisis sintáctico superficial basado en modelos estadísticos, 2000.

[15] F. Pla, A. Molina, and N. Prieto. Tagging and chunking with bigrams. In *Proc. of the 17th conference on Computational linguistics*, pages 614–620, 2000.

[16] L. Ramshaw and M. Marcus. Text chunking using transformation-based learning. In *Proc. of the third Workshop on Very Large Corpora (ACL)*, pages 82–94, 1995.

[17] H. Rulot. Un algoritmo de inferencia gramatical mediante correcci'on de errores. Technical report, Facultad de Ciencias Físicas, Universidad de Valencia, Phd Thesis, 1992.

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

[19] J.I Serrano, M.D. Del Castillo, and M.P Sesmero. Genetic learning of text patterns. In *Proc. of CAEPIA03*, pages 231–234, 2003.

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

[21] E. F. Tjong-Kim-Sang. Noun phrase representation by system combination. In *Proc. of ANLP-NAACL*, pages 50–55, 2000.

[22] F. Torró, E. Vidal, , and H. Rulot. Fast and accurate speaker independent speech recognition using structurals models learnt by the ecgi. In *Signal Proccesing V: Theories and Applications*. Elsevier Science Publishers B.V., 1990.

[23] J. Veenstra. Fast np chunking using memory-based learning techniques. In *Proc. of BENELEARN-98: Eighth Belgian-Ducth Conference on Machine Learning*, pages 71–78, 1998.

[24] A. Voutilainen. Nptool, a detector of english noun phrases. In *Proc. of the Worshop on Very Large Corpora (ACL)*, pages 48–57, 1993.

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