Evolutionary TBL Template Generation

Ruy Luiz Milidiú[1], **Julio Cesar Duarte**[1,2] **and Cícero Nogueira dos Santos**[1]

[1]Departamento de Informática
Pontifícia Universidade Católica
Rua Marquês de São Vicente, 225, Gávea
Phone: +55 (21) 3527-1500
Zip 22453-900 - Rio de Janeiro - RJ - BRAZIL
{milidiu | jduarte | nogueira }@inf.puc-rio.br

[2]Centro Tecnológico do Exército
Av. das Américas, 28705, Guaratiba
Phone: +55 (21) 2410-6200
Zip 23020-470 - Rio de Janeiro - RJ - BRAZIL
jduarte@ctex.eb.br

Abstract

Transformation Based Learning (TBL) is a Machine Learning technique frequently used in some Natural Language Processing (NLP) tasks. TBL uses rule templates to identify error-correcting patterns. A critical requirement in TBL is the availability of a problem domain expert to build these rule templates. In this work, we propose an evolutionary approach based on Genetic Algorithms to automatically implement the template generation process. Additionally, we report our findings on five experiments with useful NLP tasks. We observe that our approach provides template sets with a mean loss of performance of 0.5% *when compared to human built templates*

Keywords: Machine Learning, Genetic Algorithms, Transformation Error-Driven Based Learning.

1. INTRODUCTION

Transformation Based error-driven Learning (TBL) is a symbolic machine learning method introduced by Eric Brill [1]. The TBL technique builds an ordered set of rules that correct mistakes of a base line classifier. It has been used for several important linguistic tasks, such as part-of-speech (POS) tagging [1], parsing [8], prepositional phrase attachment [2] and phrase chunking [9, 6], achieving state-of-the-art performance in many of them.

Although TBL has been surpassed by other techniques for some advanced tasks, it is still competitive for several relevant NLP tasks, with the advantage of producing explicit rules that can be understand by humans. One possible reason for TBL's modeling limitations in some advanced tasks is that not all problems can be solved by investigating a small, local window of features around a given token. In some problems, there are unbounded dependencies which do not fit into this small window and are hard for experts to determine.

Within the TBL framework, the generated rules must follow patterns called templates, which are meant to capture the relevant feature combinations. The accuracy of the TBL classifier is highly dependent on the template set used in the learning process. Unfortunately, the process of generating *good* templates is highly expensive and depends on the problem expert skills.

In [7], we address the problem of automatic TBL template generation based on Genetic Algorithms (GAs). We introduce four genetic approaches, each one with a different degree of understanding of the problem. The better the understanding, the better is the accuracy of the generated classifier.

In this work, we provide a detailed description of our evolutionary approach. Additionally, we include more empirical evidence to support our findings. Our experiments show that we can achieve the same quality as the best template set for some benchmark problems. The overall training time is also compatible since it is increased by a factor that is smaller than the human-driven template generation process.

The remainder of this paper is organized as follows.

Section 2 presents a brief overview of GAs and TBL. In Section 3, we describe our genetic approaches. Section 4 presents the five NLP problems used and also our experimental findings. In the final section, we present our conclusions and some future work.

2. TECHNIQUES

2.1. GENETIC ALGORITHMS

Genetic Algorithms (GAs) [5] are a family of computational models inspired both by the Evolution and Natural Selection processes. They model the solution of the problem into a data structure called **chromosome**, or *genotype* or *genome*, which represents the possible solutions, called **individuals**, or *creatures* or *phenotypes*. A series of genetic operators are applied to these chromosomes in order to achieve a high optimization of the problem.

Two components play an important role in the GA method: the problem codification and the evaluation function. The problem codification is the mapping that is made between the chromosomes and the individuals. Usually, the individuals are mapped into a string of 1's and 0's indicating the presence, or not, of some feature or characteristic as shown in Figure 1. The evaluation function takes one individual and calculates its fitness. Usually, the fitness is a performance measure of the individual as a solution to the problem.

f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
1	1	1	0	0	1	1	0	1	0

Figure 1. Chromosome Example

A genetic algorithm starts with a random population of individuals. The current population is changed by the genetic operators into a new generation of individuals. The main objective of a generation is to keep the *best* individuals, enhancing the overall fitness of the population, until some stopping criteria is achieved. This criteria can be a fitness threshold, the number of generations or the lack of improvement.

There are two kinds of genetic operators: selection and recombination. Selection operators use the evaluation function to decide which individuals have the highest potential. These individuals may persist in the population and will be used by the other kind of operators.

The recombination operators are used to create new individuals using one or more high potential individuals. They are meant to diversify the search process. The most famous operators in this class are cross-over and mutation. The cross-over operator uses two or more fractions of high potential individuals to build a new individual which is appended to the next generation of the population. The mutation operator, on other hand, takes one high potential individual and makes a slight change in one of its components. The new individual is also appended in the next generation of the population.

A pseudo-code for a genetic algorithm is given in Algorithm 1.

Algorithm 1 A Genetic Algorithm Pseudo-Code

```
1: Choose an initial random population of individuals
2: Evaluate the fitness of the individuals
3: repeat
4:     Select the best individuals to be used by the genetic
       operators
5:     Generate new individuals using crossover and mu-
       tation
6:     Evaluate the fitness of the new individuals
7:     Replace the worst individuals of the population by
       the best new individuals
8: until some stop criteria
```

2.2. TRANSFORMATION BASED LEARNING

Transformation Based error-driven Learning (TBL) uses a greedy error correcting strategy. Its main objective is to generate an ordered list of rules that correct classification mistakes in the training set, which have been produced by a baseline system.

The baseline system is an algorithm that classifies the unlabeled training set by trying to guess the correct class for each sample. In general, the baseline system is based on simple statistics of the labeled training set.

The requirements of the TBL algorithm are: a training corpus, a template set, a baseline system and a score threshold. The learning method is a mistake-driven greedy procedure that iteratively acquires a set of transformation rules from the template set maximizing its score. The score from a rule can be defined as the number of corrections that it achieves in the training corpus in some iteration of the learning process, discounting the number of mistakes it makes in the same corpus. At each iteration, the rule with best score (better than the threshold) is chosen to be used in the generated classifier. The threshold value can be tuned to avoid overfitting to the training corpus. The classification process of a new sample can be done by simply applying the baseline system BLS and the ordered rule set R. The pseudo-code of the TBL algorithm is shown in Algorithm 2.

2.2.1. TBL Templates: A TBL template can be any combination of features within a specified window. This combination of features is meant to be relevant to the task being learned. We define a template as being a sequence

Algorithm 2 The TBL Algorithm Pseudo-Code

input A training corpus C_0, a template set T, a baseline system BLS and an integer threshold τ

1: Apply BLS to C_0 generating C_1
2: $R \leftarrow \{\}$
3: $k \leftarrow 1$
4: **repeat**
5: Generate CR_k, instantiating all candidate rules from T using C_k,
6: **for all** r such that $r \in CR_k$ **do**
7: score(r) $\leftarrow$ #(good corrections of r) - #(bad corrections of r) in C_k
8: **end for**
9: Choose r_M from CR_k with highest positive score above τ
10: **if** r_M exists **then**
11: Apply r_M to C_k generating C_{k+1}
12: $R \leftarrow R + r_M$.
13: **end if**
14: $k \leftarrow k + 1$
15: **until** *not* r_M exists

output R

of *Atomic Terms* (ATs). An AT is the smallest template unit that indicates the feature and conditions to be instantiated in a template. It is meant to identify one peace of the context that a TBL rule needs to test when applying to the target token. Some examples of ATs are:

1. **f[ds]**, which checks the feature *f* of a token, located *ds* tokens to the left or right (depending of the sign) of the target token. For example: f_1[-1];
2. **f[ds,de]**, which checks the feature *f* in an interval of tokens positioned between *ds* and *de* (included), in relation to the target token. For example: f_1[-1,1];
3. **f[ds,de]_where(f'=v')**, which checks the feature *f* of the token nearest to the target token, within the closed interval of *ds* and *de*, for which the feature *f'* equals *v'* [4]. For example: f_1[-1,-5]_where(f_2=v).

More complex atomic terms can be defined in order to create more specialized rules.

3. APPROACHES

In this section, we show the genetic coding used in our experiments. The use of genetic algorithms in conjunction with TBL has already been examined in [13], where they are used in the TBL training process to generate the instantiated rules and to provide an adaptive ranking. Nevertheless, they have not been used in the evaluation of template sets what is our proposal. In all codings, the template ordering is not taken into account, since it is the last criteria to be used when two or more rules have the same score.

3.1. GENETIC CODING

3.1.1. Fixed Context Window: In this approach, the *chromosome* is composed by several sequences of possible atomic terms (ATs) of the simplest form $f[ds]$. The value in the *chromosome* determines the presence or absence of the corresponding AT in the template. The input for this coding is composed by the following items: the list of possible features to be used, an integer value maxOffset, the number of templates to be generated and an expected number of atomic terms in each template. The generated templates are sequences of atomic terms of the form $f[ds]$, where $ds \in$ {-maxOffset, +maxOffset}. An example of this coding is given in Table 1, showing two templates with expected size 3, using 2 features, f_1 and f_2, and maxOffset equals to 1.

The *chromosome* shown in the Table 1 generates the following two templates:

1. f_1[-1] f_1[+1] f_2[-1] f_2[+1]
2. f_2[-1] f_2[0]

3.1.2. Fixed List of Atomic Terms: Usually, it is easier to identify candidate atomic terms by looking at the output errors of a Machine Learning Algorithm. In Fixed List of Atomic Terms, the *chromosome* is very similar to the previous one, but it can be composed by sequences of a given set of atomic terms. The *chromosome* value also indicates the presence or the absence of the corresponding atomic term in the template. The input for this coding is the list of possible atomic terms to be used, and, as well, the number of templates to be generated and the expected number of atomic terms. An example of this coding is given in Table 2, showing two templates with expected size 3, using 6 different possible atomic terms $f_1[-1]$, $f_1[-2]$, $f_2[0]$, $f_2[1]$, $f_1[0,2]$ and $f_2[-2,-0]\_where\{f_1 = v_1\}$.

The *chromosome* shown in the Table 2 generates the following two templates:

1. $f_1[-2]$ $f_2[0]$ $f_2[-2,-0]\_where\{f_1 = v_1\}$
2. $f_1[-1]$ $f_2[0]$ $f_1[0,2]$

3.1.3. Maximum Template Size: In this approach, the *chromosome* is quite similar to the previous one, but instead of having an expected template size we establish a maximum size for all templates. The *chromosome* value indicates the position of the corresponding atomic term in the list. A value -1 indicates the absence of an atomic term. The repetition of atomic terms in the same template is now a possibility, but they are discarded. The input for

	Template 1						Template 2					
	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]
C_1	1	0	1	1	0	1	0	0	0	1	1	0

Table 1. Example of the Fixed Context Window Approach

	Template 1						Template 2					
	AT_0	AT_1	AT_2	AT_3	AT_4	AT_5	AT_0	AT_1	AT_2	AT_3	AT_4	AT_5
C_1	0	1	1	0	0	1	1	0	1	0	1	0

Table 2. Example of the Fixed List of Atomic Terms Approach

this coding is the list of possible atomic terms to be used, the number of templates to be generated and the maximum template size. An example of this coding is given in Table 3, showing three templates with maximum size 4, using the same six possible previous atomic terms.

The *chromosome* shown in the Table 3 generates the following three templates:

1. $f_1[-1]\ f_1[-2]\ f_2[1]$
2. $f_1[-2]\ f_2[0]\ f_2[1]\ f_2[-2,-0]\_where\{f_1 = v_1\}$
3. $f_1[-2]\ f_2[0]\ f_2[1]$

3.1.4. Template List: In this approach, the *chromosome* is composed of a sequence of predefined templates. The idea here is to find a better subset of templates than the one provided by an expert. Since TBL is a greedy algorithm, using all templates may not lead to better results than using just one of its subsets. The *chromosome* value indicates the presence or absence of the corresponding template. The input for this coding is the list of possible templates to be used and the expected number of templates to be used. An example of this coding is given in Table 4, showing templates from the fixed template list, $\{\tau_{00}, \tau_{01}, \tau_{02}, \tau_{03}, \tau_{04}, \tau_{05}, \tau_{06}, \tau_{07}, \tau_{08}, \tau_{09}, \tau_{10}, \tau_{11}\}$, with an expected number of seven templates.

The *chromosome* shown in the Table 4 generates the following template set: $\{\tau_{00}, \tau_{02}, \tau_{05}, \tau_{06}, \tau_{08}, \tau_{09}, \tau_{10}\}$.

3.2. Fitness function

Using a training set, we train a TBL classifier for each individual. The F-measure of the generated classifier for a validation set is used as the fitness value of the individual.

3.3. Cross-over operator

The cross-over operator generates a new *chromosome* by breaking apart two *chromosomes* in a random point and combining them. Table 5 shows an example of the cross-over operator for the *chromosome* described in the Fixed Context Window approach.

3.4. Mutation operator

The mutation operator generates a new *chromosome* by changing the value of the atomic term in a template. Table 6 shows an example of the mutation process for the *chromosome* described in the Fixed Context Window approach.

For the Maximum Template Size approach, instead of changing the value from 0 to 1 and vice-versa, the value is changed to another value in the interval [-1, number of atomic terms - 1].

4. Experiments

4.1. NLP Problems

We conduct five experiments with useful NLP tasks: English Base Noun Phrase Chunking (BNP), English Text Chunking (CK), Portuguese Named Entities Recognition(NE), Portuguese Noun Phrase Chunking (SN) and Portuguese Appositive Extraction (AP). This experiments are meant to illustrate the quality of our genetic approaches.

Noun Phrase Chunking consists in recognizing non-overlapping text segments that correspond to noun phrases (NPs). *Text chunking* consists in dividing a text into syntactically correlated parts of words. *Named Entity Recognition* is the problem of finding all proper nouns in a text and to classify them among several given categories such as Person, Organization or Location. Finally, *Appositive Extraction* consists in recognizing structures composed by semantically related noun phrases that must be identical in reference, that is, they are co-referential.

4.1.1. Corpora: Table 7 shows some characteristics of the corpora used in the NLP experiments. The data used in the English Text Chunking is the CoNLL-2000 corpus [11]. This corpus is tagged with POS and Text Chunk tags. The data used in the Base Noun Phrase Chunking is the one of Ramshaw & Marcus [9]. It is composed by the same texts as the CoNLL-2000 corpus. The difference is that the Chunk Tags only identify base NPs.

	Template 1				Template 2				Template 3			
	AT_1	AT_2	AT_3	AT_4	AT_1	AT_2	AT_3	AT_4	AT_1	AT_2	AT_3	AT_4
C_1	1	3	-1	0	5	1	3	2	1	2	1	3

Table 3. Example of the Maximum Template Size Approach

	τ_{00}	τ_{01}	τ_{02}	τ_{03}	τ_{04}	τ_{05}	τ_{06}	τ_{07}	τ_{08}	τ_{09}	τ_{10}	τ_{11}
C_1	1	0	1	0	0	1	1	0	1	1	1	0

Table 4. Example of the Template List Approach

The Named Entities corpus used is the same reported in [10] which contains POS and NE tags. In the Portuguese Noun Phrase experiments, we use the corpus reported in [12]. This corpus has both POS and SN tags. The last conducted experiment is Portuguese Appositive Extration (AP) and uses the corpus reported in [3]. This corpus is derived from the SN corpus containing the AP tags.

4.2. Experimental Setup

In all experiments, a small excerpt of the training corpus is used by the genetic approach. Two corpora are built: a GA-training set and a validation set. The GA-training and validation sets are used by the genetic algorithm to, respectively, train and evaluate the performance of the individuals. The *best* individual returned by the genetic algorithm is applied to the whole training corpus, generating a TBL classifier. The classifier is, then, applied to the test corpus and its performance is evaluated.

We use F-measure as our key statistics to evaluate the performance of the generated classifiers. **F-measure** is the harmonic mean between precision and recall. **Precision** informs how many good classifications the model predicted amongst all predictions made. **Recall** informs how many good classifications were predicted amongst all true classifications.

For the four genetic formulations, we report the performance of the classifier trained with the *best* template set produced by the use of different slices of the GA-training set in the genetic approach. These results are compared with the Baseline System (BLS) and a handcrafted template set (HC) obtained from the referenced articles. We start with 50 sentences for the genetic training process, increasing with 50 more examples in each experiment.

Although, we fixed the τ parameter used in all experiments to the same value used by the handcrafted templates, it could also be encoded and determined by the genetic algorithm without considerable loss of performance, since its set of optimal values is very limited (usually, $0 \leq \tau \leq 2$).

We also report the training time for each approach, in terms of percentage of the training time for the experiment with 50 sentences. The reported training time includes both the generation of the best template set by the genetic algorithm and the training of the TBL classifier. The BLS training time is not reported since it is very small.

In all approaches, the performance of the population in the validation set over the ten fixed generations shows a consistent increase.

4.3. Results

4.3.1. English Base Noun Phrase Chunking: The results for the Fixed Context Window (FCW) approach are reported in Figure 2. The experiment is conducted using the three possible features (word, POS and NP tag) with a window size of five ([-2, +2]). The genetic algorithm generated 20 templates with an expected atomic term size of 3. As we can see, the results are very good since we generate only 20 templates with the simplest atomic term. The loss of F-measure is smaller than 1% in the best ga-training sets. Also the genetic approaches takes few training time, since the templates are very simple. It is interesting to notice here that the training time decreases in the first increases of the genetic training corpus. This fact occurs because the BLS depends on the provided training corpus and, since the training corpus is very small, its accuracy is very poor, providing many errors for the genetic TBL templates to correct, which increases the overall training time.

Figure 3 shows the results for the Maximum Template Size (MTS) approach. The atomic term list used is $\{$npt[0], npt[-1], npt[-2], npt[1], npt[2], pos[0], pos[1], pos[2], pos[-2], pos[-1], pos[$-3, -1$], pos[$1, 3$], word[0], word[1], word[2], word[-1], word[-2], word[$-3, -1$], word[$1, 3$]$\}$. The results are almost the same. We do not use very complex atomic terms in order to maintain the simplicity of the approaches, avoiding the need of a specialist to determine the atomic term list. The genetic algorithm generated 20 templates with maximum atomic term size of 5. The overall training time is increased, since we added atomic terms that may instantiate more candidate

	Template 1						Template 2					
	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]
C_1	**1**	**0**	**1**	**1**	**0**	**1**	**0**	0	0	1	1	0
C_2	1	1	0	0	0	1	1	**1**	**0**	**1**	**0**	**0**
$C_1 \otimes C_2$	1	0	1	1	0	1	0	1	0	1	0	0

Table 5. Example of the Cross-over operator

	Template 1						Template 2					
	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]
C_1	1	0	1	1	0	1	0	0	0	**1**	1	0
$\odot C_1$	1	0	1	1	0	1	0	0	0	**0**	1	0

Table 6. Example of the Mutation operator

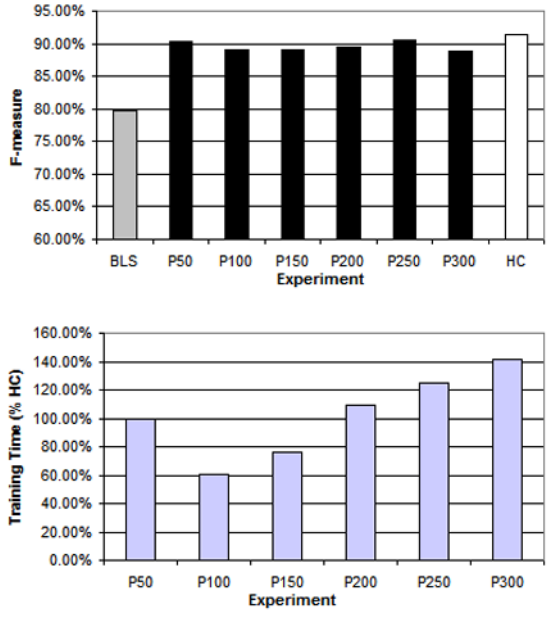

Figure 2. Results for the Fixed Context Window approach (BNP)

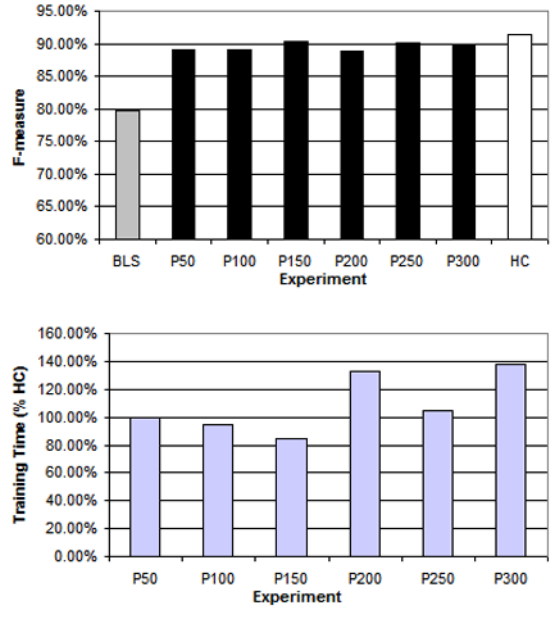

Figure 3. Results for the Maximum Template Size approach (BNP)

rules.

The experiment using the Fixed List of Atomic Terms (FLAT) approach is quite similar to the previous one, with same main parameters, and is reported in Figure 4. The only difference is that we define the expected template size, which was fixed in 4. We can see that the results are very similar to the previous one, in terms of F-measure and training time, since the two approaches are quite equivalent.

The last experiment for BNP uses the Template List (TL) approach. In this experiment, we try to find out a better combination of templates than the one provided by a specialist. Here, we use the template set proposed in [9]. The genetic generations are started with 80% of the templates activated. Figure 5 shows the results for this experiment. We can see that the template combination found by our approach achieve better results than the template set proposed by the specialist. However, this achievement implies in an increase of the overall training time.

4.3.2. English Text Chunking: In all following experiments, we use the same parameter setting as the one for BNP. This shows the power and simplicity of our scheme.

Figures 6, 7, 8 and 9 show the results for the four approaches. Our results are very close to the reference

NLP Problem	Train Data		Test Data	
	Sentences	Tokens	Sentences	Tokens
English Base Noun Phrase English Text Chunking	8936	211727	2012	47377
Portuguese Named Entities Recognition	1722	27055	378	6084
Portuguese Noun Phrase Chunking	10051	210044	2206	46847
Portuguese Appositive Extraction	3681	79560	807	17942

Table 7. NLP Tasks Corpora.

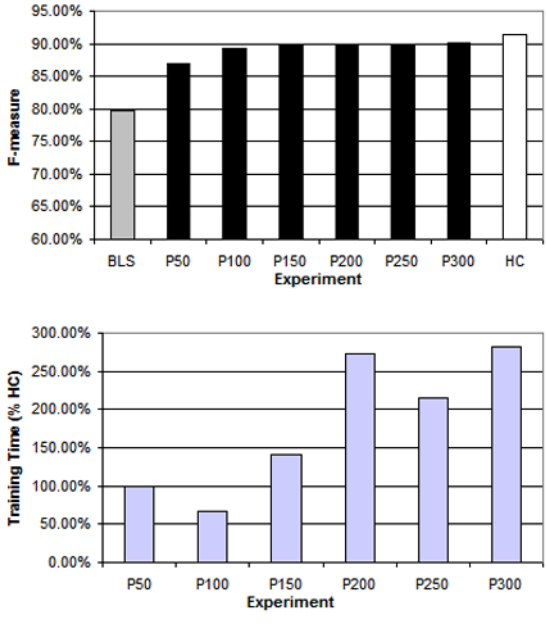

Figure 4. Results for the Fixed List of Atomic Terms approach (BNP)

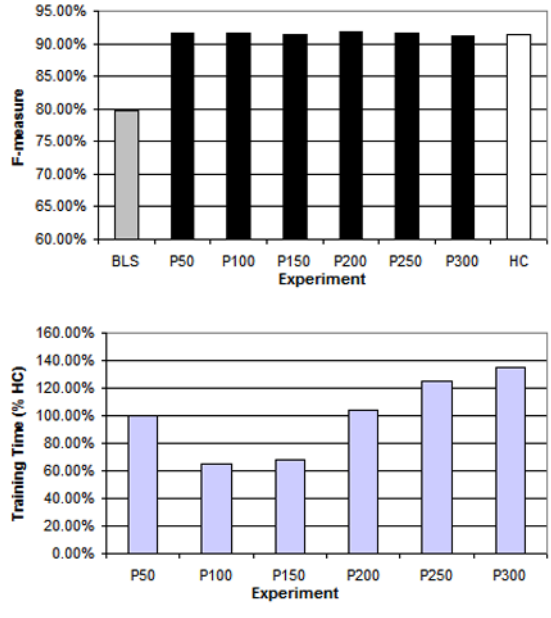

Figure 5. Results for the Template List approach (BNP)

ones, even with the simplest approach.

4.3.3. Portuguese Named Entity Recognition: We show the results of the genetic approaches for NE in Figures 10, 11, 12 and 13. Here, the results with FLAT and TL are even better than the ones provided by the handcrafted templates. The only difference is that the automatically generated templates took much more time to train since they are much more generic than the handcrafted templates.

4.3.4. Portuguese Noun Phrase Chunking: We compare the results with the handcrafted templates described in [12]. We must observe that this template set contains very sharp domain knowledge. This certainly required a lot of engineering effort to be acquired. Figures 14, 15, 16 and 17 report the results for all genetic approaches. The results with MTS are very close to the reference values. The template set reduction by the TL approach does not improve performance. This indicates that all the handcrafted templates are critical to the model performance.

4.3.5. Portuguese Appositive Extraction: Figures 18, 19, 20 and 21 show these results in all described approaches.

Here, the BLS in not an AP extractor. Its function is just to provide hints that help TBL learning. Therefore, it does not have a performance to evaluate and compare.

This task is more difficult than the others, since it is a relational task. In this case, only the MTS approach

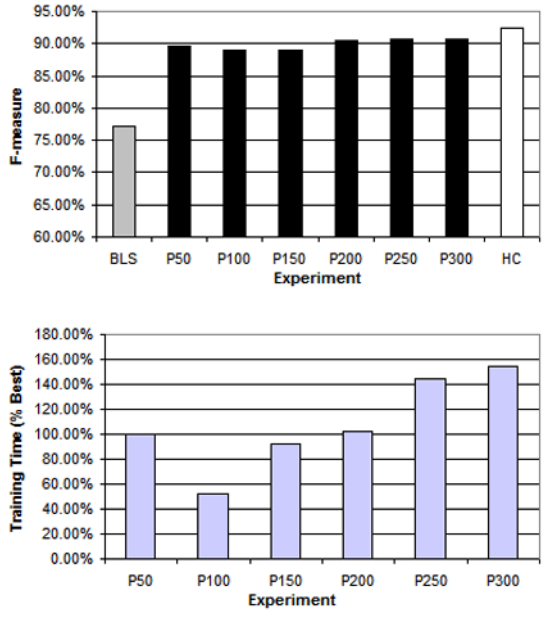

Figure 6. Results for the Fixed Context Window approach (CK)

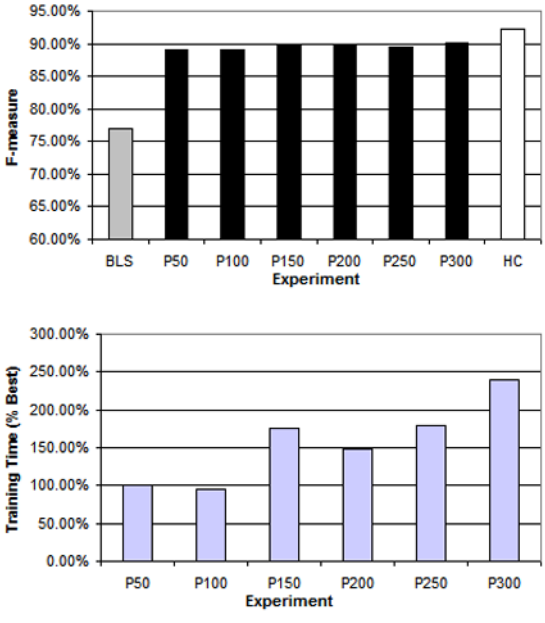

Figure 7. Results for the Maximum Template Size approach (CK)

shows a good performance, close by 1% to the hand-crafted templates.

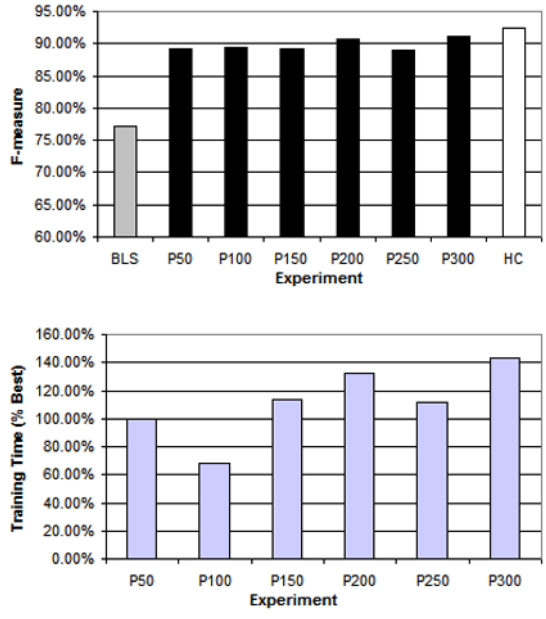

Figure 8. Results for the Fixed List of Atomic Terms approach (CK)

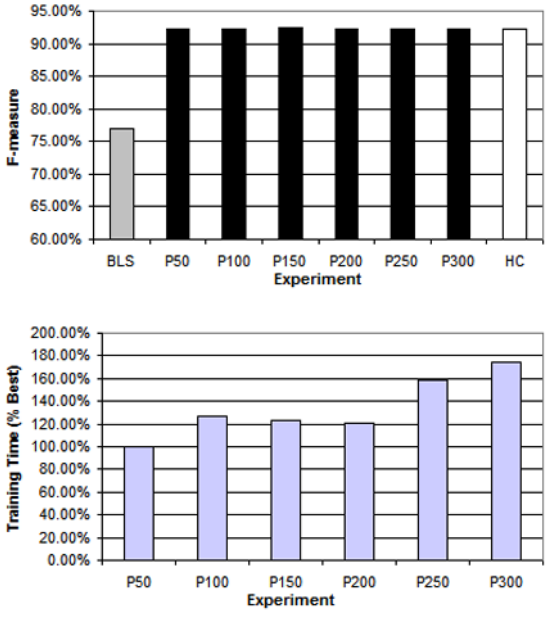

Figure 9. Results for the Template List approach (CK)

4.4. Experimental Findings

In all tasks, we manage to automatically find a new template set that provides compatible results with the ones provided by specifically handcrafted templates. This is very attractive since human driven template generation is

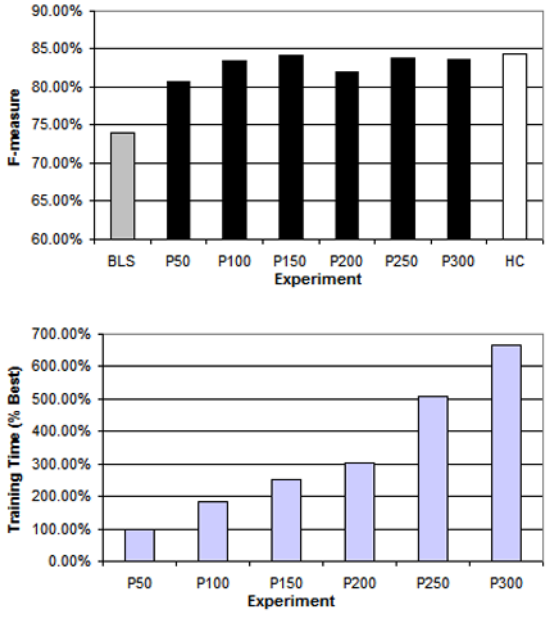

Figure 10. Results for the Fixed Context Window approach (NE)

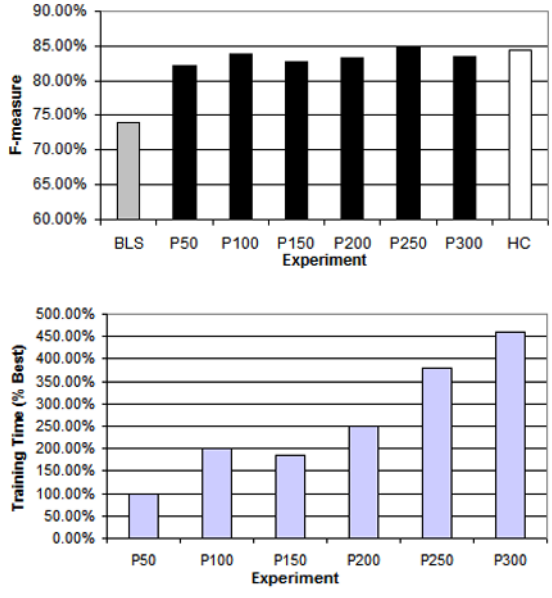

Figure 12. Results for the Fixed List of Atomic Terms approach (NE)

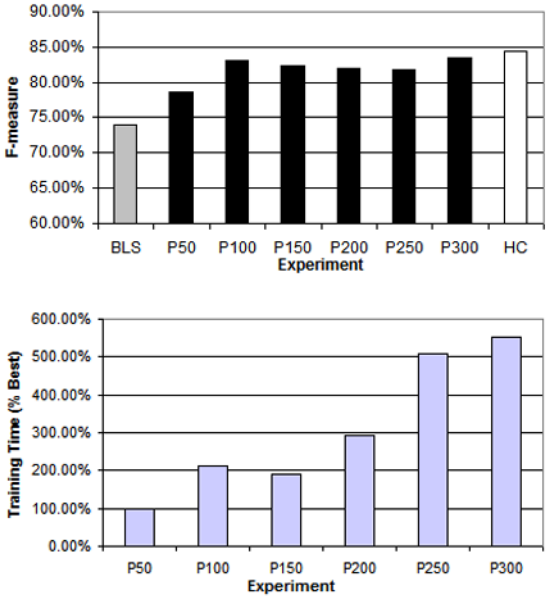

Figure 11. Results for the Maximum Template Size approach (NE)

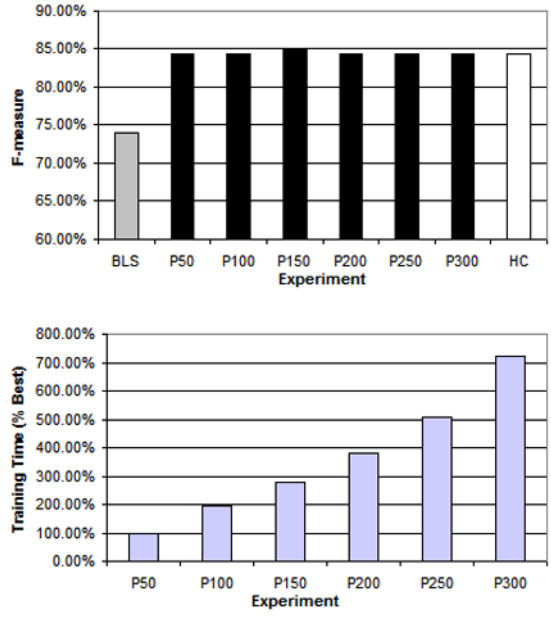

Figure 13. Results for the Template List approach (NE)

a very expensive task and requires TBL engine expertise.

The training time in all cases increased from a few minutes, with the handcrafted templates, to a few hours, when we include the evolutionary template generation. Observe, however, that handcrafting the templates requires a series of trial-and-error TBL modeling and training, what increases the overall training time to days or even weeks, depending on the complexity of the task. Therefore, the overall computing effort is also reduced with evolutionary template generation.

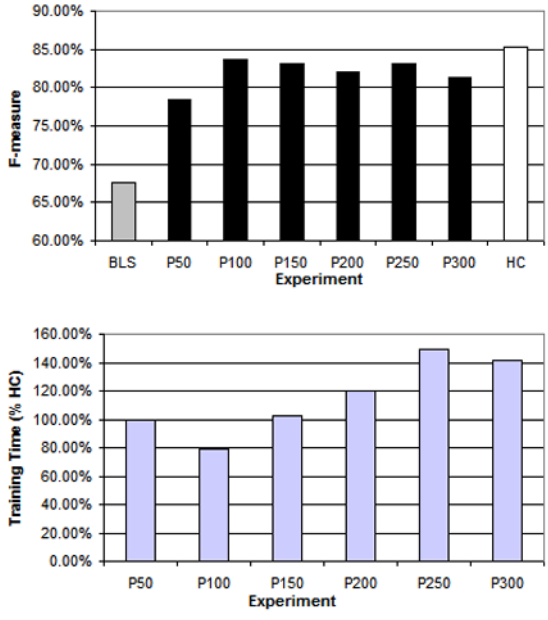

Figure 14. Results for the Fixed Context Window approach (SN)

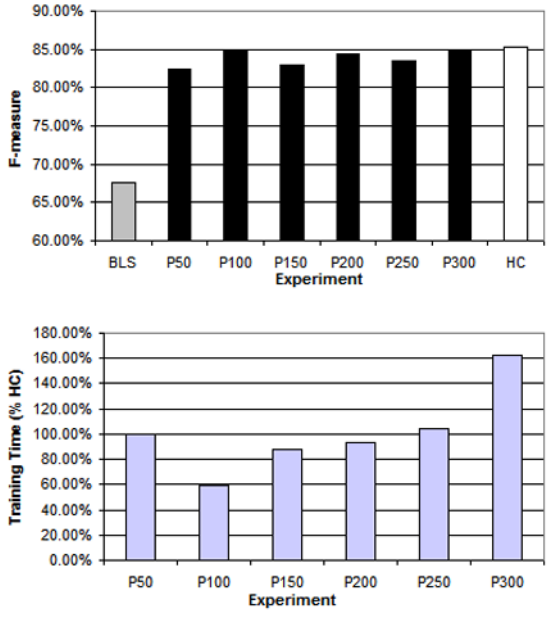

Figure 15. Results for the Maximum Template Size approach (SN)

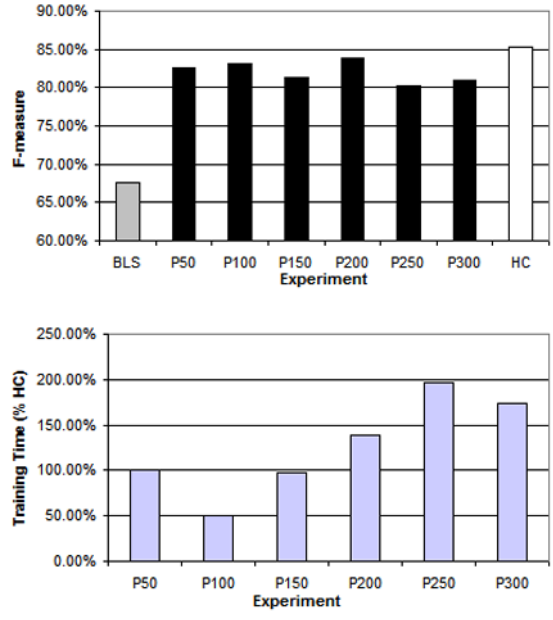

Figure 16. Results for the Fixed List of Atomic Terms approach (SN)

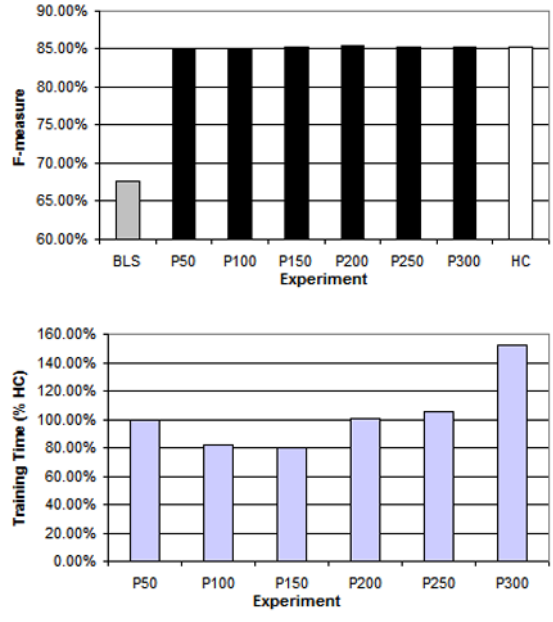

Figure 17. Results for the Template List approach (SN)

5. CONCLUSIONS

TBL Template construction is a highly expensive process with strong impact in the classifier's accuracy. In this paper, we present an evolutionary approach to help the generation of TBL templates. Our schemes use simple template design and very little training data to develop a set of templates.

We show a set of experiments that illustrate the applicability and the effectiveness of the proposed method. In all experiments, we managed to get compatible results

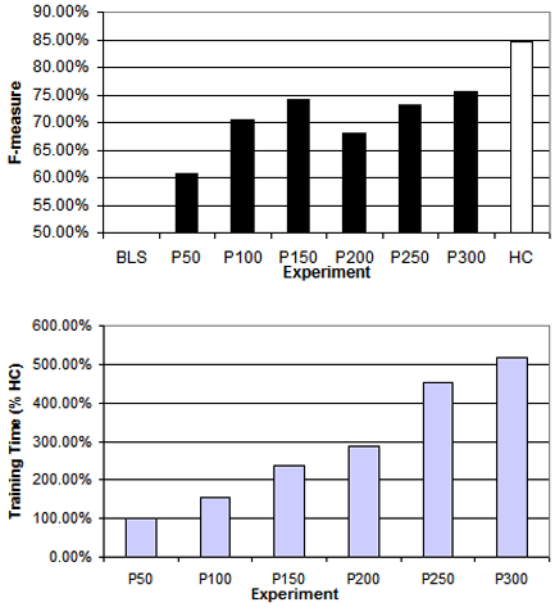

Figure 18. Results for the Fixed Context Window approach (AP)

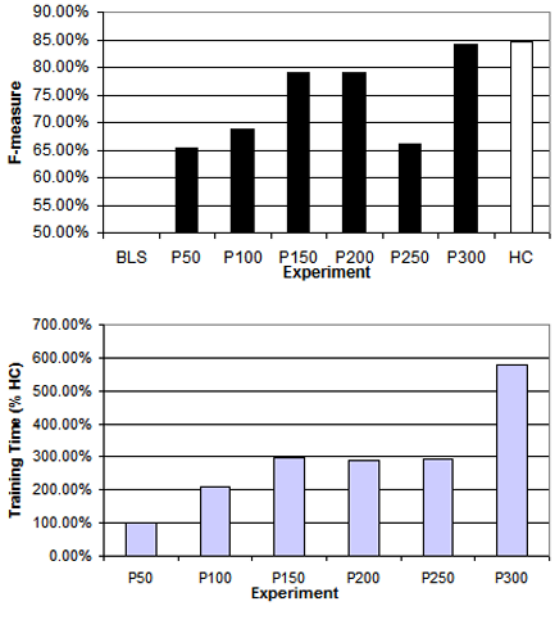

Figure 19. Results for the Maximum Template Size approach (AP)

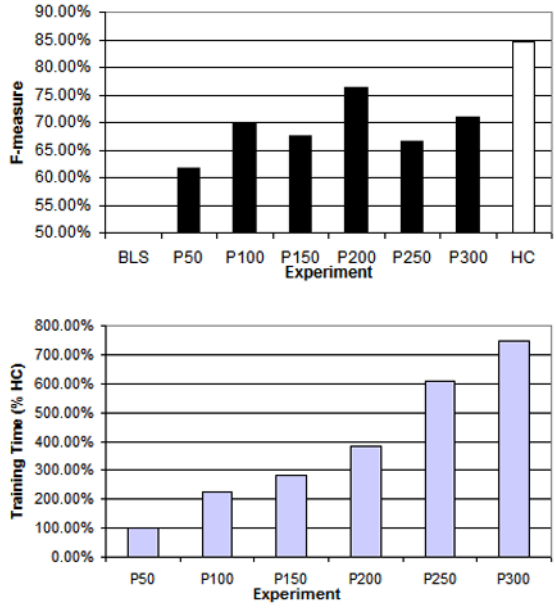

Figure 20. Results for the Fixed List of Atomic Terms approach (AP)

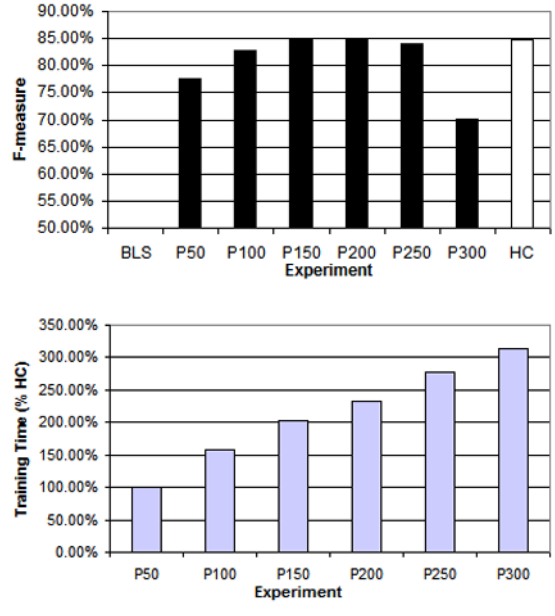

Figure 21. Results for the Template List approach (AP)

when comparing to the ones obtained by handcrafted templates. In some cases, the results provide even a better F-score. Moreover, the overall training time is reduced since the domain expert is removed from the template generation process.

In the future work, we plan to analyze the fitness function time-complexity to improve the efficiency of our solution. We also plan to validate it on harder problems, even the ones that require investigation with a large feature offset.

REFERENCES

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

[2] Eric Brill and Philip Resnik. A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of COLING'94*, Kyoto, Japan, 1994.

[3] Maria Claudia de Freitas, Julio Cesar Duarte, Cícero Nogueira dos Santos, Ruy Luiz Milidiú, Raúl P. Rentería, and Violeta Quental. A machine learning approach to the identification of appositives. In Jaime Simão Sichman, Helder Coelho, and Solange Oliveira Rezende, editors, *IBERAMIA-SBIA*, volume 4140 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2006.

[4] Cícero Nogueira dos Santos and Claudia Oliveira. Constrained atomic term: Widening the reach of rule templates in transformation based learning. In Carlos Bento, Amílcar Cardoso, and Gaël Dias, editors, *EPIA*, volume 3808 of *Lecture Notes in Computer Science*, pages 622–633. Springer, 2005.

[5] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[6] Beáta Megyesi. Shallow parsing with pos taggers and linguistic features. *Journal of Machine Learning Research*, 2:639–668, 2002.

[7] Ruy Luiz Milidiú, Julio C. Duarte, and Cícero Nogueira dos Santos. Tbl template selection: An evolutionary approach. In Daniel Borrajo, Luis A. Castillo, and Juan M. Corchado, editors, *CAEPIA*, volume 4788 of *Lecture Notes in Computer Science*, pages 180–189. Springer, 2007.

[8] Miloslav Nepil. Learning to parse from a treebank: Combining tbl and ilp. In Céline Rouveirol and Michèle Sebag, editors, *ILP*, volume 2157 of *Lecture Notes in Computer Science*, pages 179–192. Springer, 2001.

[9] Lance Ramshaw and Mitch Marcus. Text chunking using transformation-based learning. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, New Jersey, 1995. Association for Computational Linguistics.

[10] Roberto Cavalcante Ruy Luiz Milidiú, Julio Cesar Duarte. Machine learning algorithms for portuguese named entity recognition. In Solange Oliveira Rezende and Antonio Carlos Roque da Silva Filho, editors, *Fourth Workshop in Information and Human Language Technology (TIL'06) in the Proceedings of International Joint Conference, 10th Ibero-American Artificial Intelligence Conference, 18th Brazilian Artificial Intelligence Symposium, 9th Brazilian Neural Networks Symposium, IBERAMIA-SBIA-SBRN, Ribeirão Preto, Brazil, October 23-28, 2006*, 2006.

[11] Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational Natural Language Learning*, pages 127–132. Association for Computational Linguistics, 2000.

[12] Cícero Nogueira Santos. Aprendizado de máquina na identificação de sintagmas nominais: o caso do português brasileiro. Master's thesis, IME, Rio de Janeiro - RJ, fevereiro 2005.

[13] Garnett Wilson and Malcolm Heywood. Use of a genetic algorithm in brill's transformation-based part-of-speech tagger. In Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantu-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llora, Spiros Mancoridis, Martin Pelikan, Guenther R. Raidl, Terence Soule, Andy M. Tyrrell, Jean-Paul Watson, and Eckart Zitzler, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 2067–2073, Washington DC, USA, 25-29 June 2005. ACM Press.