Verb Sense Disambiguation in STRING

Ricardo Pires[1,2], Nuno Mamede[1,2], and Jorge Baptista[2,3]

[1] Universidade de Lisboa - Instituto Superior Técnico
[2] L^{2}F - Spoken Language Systems Laboratory - INESC-ID Lisboa
[3] Universidade do Algarve - Faculdade de Ciências Humanas e Sociais
ricardo.pires@tecnico.ulisboa.pt
nuno.mamede@inesc-id.pt
jbaptis@ualg.pt

Abstract. Word Sense Disambiguation has a great influence over the performance of several Natural Language Processing tasks, since the exact meaning of an ambiguous word in a given context may have an impact on the syntactic parsing of the sentence, for example in the parsing of the dependency relations among its constituents. This paper focuses on Verb Sense Disambiguation, a sub-problem of Word Sense Disambiguation, in the context of STRING, a NLP chain developed for Portuguese. The paper presents an approach for the generation of disambiguation rules to be used in STRING. This approach uses a set of declarative rules and the knowledge present in ViPEr, a Lexicon-Grammar for the European Portuguese verbs, to generate disambiguation rules. An implementation of the Transformation-based Learning machine-learning algorithm is also presented, as a way to optimize the order of application of the generated disambiguation rules.

Keywords: natural language processing, verb sense disambiguation, rule-based disambiguation, machine-learning

1 Introduction

Nowadays, Natural Language Processing (NLP) is a field of research with a wide array of applications. However, it still faces several unsolved challenges, posed by the very nature of language, among which *ambiguity* is paramount. Ambiguity is present at several levels of analysis and in many forms. One of the most difficult aspects of this complex phenomena is word sense ambiguity, that is, the possibility of the same word being assigned different semantic values depending on its given context. This has a large impact on the results of NLP, since an inadequate disambiguation can change the meaning of the text being processed. This is particularly relevant when it comes to verbs, which, for the most part, express semantic predicates, thus are one of the most important components for determining the meaning of sentences and assigning them an adequate syntactic parse.

1.1 Goal

This work aims at improving the Verb Sense Disambiguation (VSD) mechanisms currently in place on the STatistical and Rule-based Natural lanGuage processing chain (STRING) [1] system, using recent developments on ViPEr [2] - a database of the European Portuguese Verbs. For each verb, this database contains a set of semantic and syntactic features that describe its use.

Two approaches are currently used, a rule-based [3] and a machine-learning-based [3, 4] approach.

The rule-based approach is implemented through a module that uses the knowledge contained in ViPEr to automatically produce disambiguation rules, which are then applied within the STRING chain. However, this approach suffers from some shortcomings: it is not very flexible, which makes it hard to keep up with ViPEr's recent developments; and all the knowledge used in the rule generation process is embedded in the code of the module itself.

The machine learning approach uses *corpora* annotated according to the information provided by ViPEr and a machine learning algorithm to build models for each verb, which are then used to choose between a set of verb senses for that verb.

This work aims at refining verb sense disambiguation in STRING by improving upon the current rule-based approach shortcomings. In addition, this work will use the Transformation-based Learning (TBL) machine-learning algorithm to help improve rule-based disambiguation.

2 STRING

STRING is a hybrid, statistical and rule-based NLP chain for the Portuguese developed at the Spoken Language Systems Laboratory (L^2F). It has a modular structure, and it performs basic text processing tasks, such as tokenization and text segmentation, part-of-speech tagging, morphosyntactic disambiguation, shallow parsing (chunking), and deep parsing (dependency extraction). It also performs additional NLP tasks, such as Named Entity Recognition, Anaphora Resolution and Time Normalization, among others. The system's architecture is shown in Figure 1.

The first stage of the chain is responsible for performing segmentation, part-of-speech (POS) tagging, and sentence splitting. This is performed by the Lexical Morphological Analyser (LexMan) [5, 6] module. The second stage is the disambiguation stage, which is comprised of two steps: a rule-driven morphossyntactic disambiguation, performed by the Rule Driven Converter (RuDriCo2) [7, 8] module; and a statistical disambiguation, performed by the Morphossyntactic Ambiguity Resolver (MARv4) [9, 10] module. The final stage is the syntactic analysis stage (parsing). This stage is performed using the XEROX Incremental Parser (XIP) parser [11]. This is also where the VSD task is performed. Finally, a set of post-processing modules uses the output of STRING to perform specific NLP tasks. These include anaphora resolution, slot filling, time expression normalization and temporal event ordering.

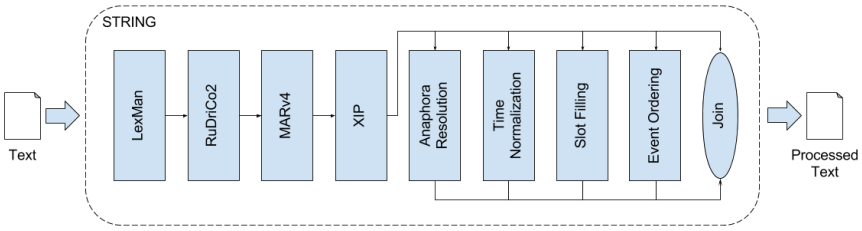

Fig. 1. STRING architecture

The first three stages are always performed, whereas the fourth stage can be skipped if not necessary. In addition, not all of the modules of the fourth stage need to be used in every case. The communication between the different modules is done using eXtensible Markup Language (XML) files.

2.1 XIP

XIP [11] is the module responsible for performing syntactic analysis in the STRING chain. First, it allows for the introduction of additional lexical, syntactic and semantic information to the result of the previous modules; and, then, it performs syntactic analysis through the sequential application of local grammars and morphossyntactic disambiguation rules, the calculation of chunks and the extraction of dependencies between chunks. XIP uses nodes as the basic data representation structure. Each node has a category (such as *noun*), a set of feature-value pairs (*plural*, `plural=+`), and a set of brother nodes.

XIP itself is composed of several modules:

- Lexicons: used to add information to tokens. XIP supports the definition of custom lexicons, which are used to add new features not yet present on the pre-existing lexicon;
- Local Grammars: consisting in pattern-matching rules that consider the context of a given pattern. These rules are used to identify entities composed by several different elements, and then group those elements into a single entity (e.g. *Ministério da Justiça*, Ministry of Justice);
- Chunking Module: This module is responsible for grouping sequences of nodes into structures called *chunks*. These chunks represent basic phrase types, such as noun phrase (NP), prepositional phrase (PP) or finite verb phrase (VF). This is done through rule-based syntactic analysis. The NP, VP and PP chunks are processed afterwards by the Dependency Extraction Module;
- Dependency Extraction Module: This module extracts the syntactic-semantic relations between nodes. To do so, it uses a set of dependency rules, which

it applies using a logic programming paradigm. This is where VSD is performed, by using a set of dependency rules that disambiguate between pairs of verb senses, represented as different verb classes, for verbs that are present in a given dependency.

3 ViPEr

ViPEr is a Lexicon-Grammar for European Portuguese Verbs [2]. It is a linguistic resource with the syntactic and semantic properties of the most frequent verbs of the Portuguese language. These fine-grained properties, along with the constraints the verbs impose on their arguments (such as the number of arguments, the prepositions each verb uses to introduce its complements or the main shape-changes these structures can undergo) are meant to facilitate the correct identification of verb meanings in texts. ViPEr focuses on distributional (or lexical) verbs, whose meaning allows for an intensional definition of their respective syntactic construction and the semantic constraints on their arguments.

Each verb is described by a syntactic frame composed of the basic constituents of the sentence. The frame takes the form: *N0, Prep1, N1, Prep2, N2, Prep3, N3*. *N0-N3* describe the verb's arguments. *N0* corresponds to the subject of the sentence, and *N1-N3* represent the verb's complements. *Prep1-Prep3* are the prepositions that introduce the respective complement. *Prep1* may be absent in the case of direct-transitive verbs (see below).

Each component of the frame has a set of boolean properties, associated with each verb class. The properties accept one of two values, '+' if the property must be positive, and '-' otherwise. For example, if a verb requires its subject to be a human noun, then it has the property *N0=Hum* marked with a '+'. Other properties include *Npl*, for plural nouns, *QueF* for completive subclauses, *vse* for intrinsically pronominal (reflex) constructions, among others.

4 TBL

TBL [12] is a machine-learning algorithm that has been successfully used in several NLP problems, such as POS tagging and syntactic parsing. It derives linguistic information from an unannotated *corpus* to build and improve a linguistic model by manipulating the *corpus* in order to find the closest result to the truth. The truth, in this context, is a manually annotated *corpus* that is used as reference. Figure 2 shows the architecture of the algorithm.

The algorithm takes unannotated text, and through an initial state annotator produces the annotated text to be used. Different initial state annotators have been used with this algorithm: For POS tagging, the output of a stochastic n-gram tagger, labelling all words with the most likely tag or labelling all the words as nouns have all been used; for syntactic parsing, initial state annotators have ranged from the output of sophisticated parsers to random tree structures with random non-terminal labels.

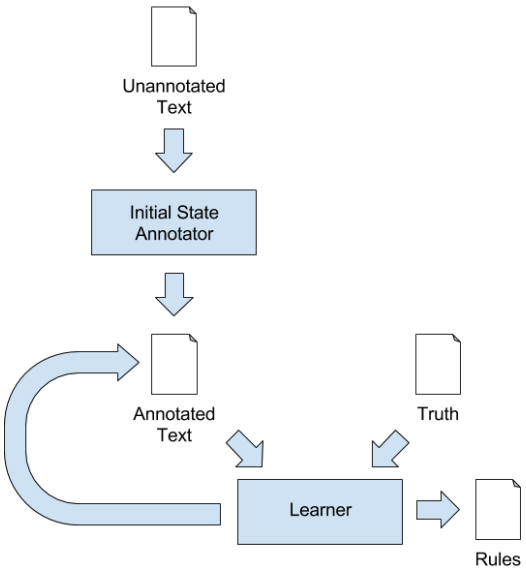

Fig. 2. Transformation-Based Learning - Architecture

Following the initial state annotation, a learner compares the annotated text to the truth, in order to produce transformations that drive the annotated text closer to the reference. These transformations are composed by a rewrite rule, and a triggering environment. The rewrite rule indicates how to transform the text, and the triggering environment determines the conditions in which this transformation should be applied.

The learner follows an iterative process, in which different transformations are applied individually to the annotated text, and the best one is chosen and added to an ordered list. To produce this list of transformations, several approaches may be used. In the original TBL implementation [12], a greedy best-first approach was used. This means that at every iteration, the transformation that drives the annotated text closer to the truth is chosen and then added to the ordered list. However, greedy best-first approaches are usually not optimal, thus other approaches may be used to improve upon this.

Finally, when no more transformations can be reordered in such a way that brings the annotated text closer to the truth, the learner stops, and outputs the final ordered list of transformations.

5 Rule-generation Module

Figure 3 shows how the rule-generation module here developed fits in the STRING chain. This module takes ViPEr and a set of declarative rules as input, and produces a set of XIP dependency rules as output. This set of rules is then used by XIP to perform the actual verb sense disambiguation.

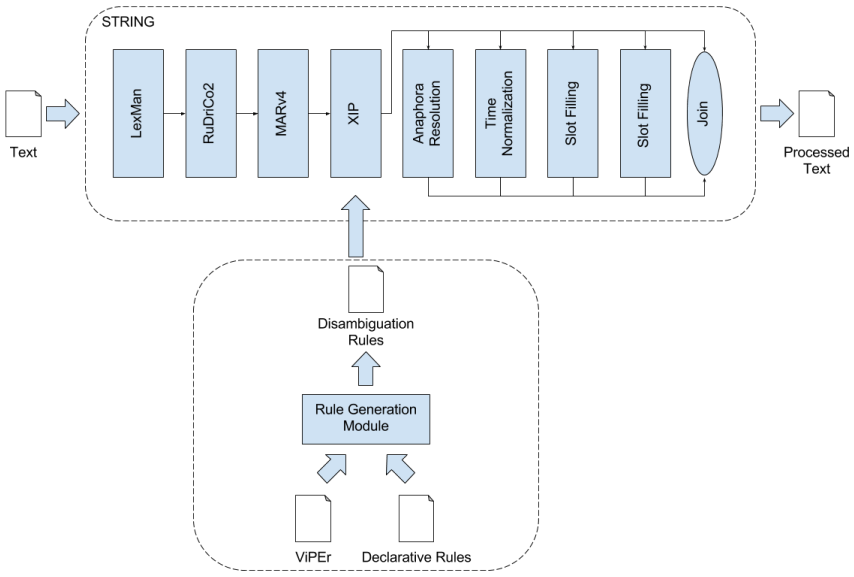

Fig. 3. Rule-generation Module in STRING

Figure 4 shows the architecture of the new rule-generation module. The module's processing is carried out in two stages. In the first stage, both ViPEr and the set of declarative rules are parsed, and the information contained therein is stored. In the second stage, the module uses the stored information and applies the declarative rules to the verbs present in ViPEr.

5.1 Declarative Rules

The previous approach to rule-generation had a major disadvantage, in that most of the knowledge required for this process was embedded in the code of the module. In order to avoid this, a new way of representing this knowledge was devised. Instead of being embedded in the code, this knowledge is now represented under the guise of declarative rules.

These rules follow the format:

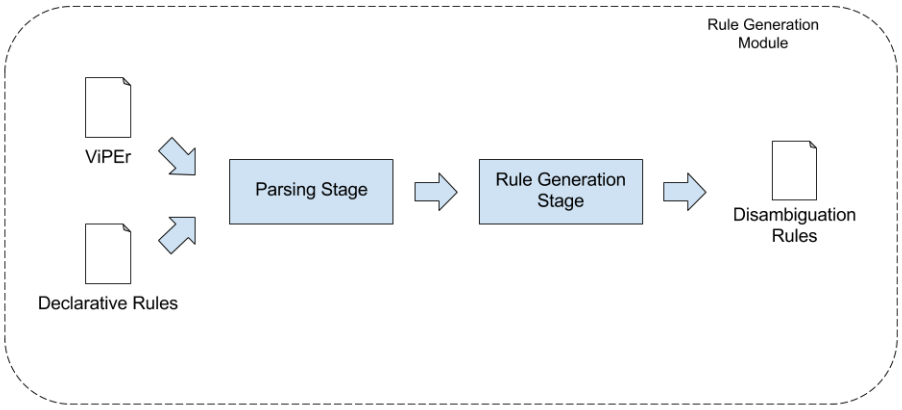

Fig. 4. Rule Generation Module Architecture

```
VIPER  v1{<props +>}{<props ->}{<pairs +>}{<pairs ->}
       v2{<props +>}{<props ->}{<pairs +>}{<pairs ->}
       {{<XIP condition>}}
```

This means:

- `v1` refers the first ViPEr class of the pair to disambiguate
- `v2` refers the second ViPEr class of the pair to disambiguate
- `<props +>` is the set of ViPEr properties that the ViPEr class must have in order for the XIP rule to be generated;
- `<props ->` is the set of ViPEr properties that the ViPEr class must not have in order for the XIP rule to be generated;
- `<pairs +>` is the set of pairs[1] of ViPEr properties that the ViPEr class must have in order for the XIP rule to be generated;
- `<pairs ->` is the set of pairs of ViPEr properties that the ViPEr class cannot have in order for the XIP rule to be generated;
- `XIP Condition` is the XIP condition of the generated rule (The argument `#1` refers to the verb being disambiguated);

Consider Declarative Rule 1.

```
VIPER v1{vse}{}{}{}
      v2{}{vse}{}{}
      {{CLITIC(#1,?[ref])}}
```

Declarative Rule 1. Single Property Rule

This rule means that, for a given pair of ViPEr classes `v1` and `v2`, class `v2` is removed from the verb if the following conditions are true:

[1] A pair of features is identified by the ampersand (`&`) that join them. A ViPEr class is considered to have a pair of features if both features are marked with a '+'

– If `v1`, in ViPEr, has a '+' in the ViPEr property `vse`;
– If `v2`, in ViPEr, has a '-' in the ViPEr property `vse`;
– If the verb is the first argument of a `CLITIC` XIP dependency, and the second argument of the `CLITIC` XIP dependency has the `ref` XIP feature. In plain English, an intrinsically pronominal verb construction (`vse`) must show a reflex clitic pronoun attached to that verb.

5.2 Rule-generation

The module first parses both the ViPEr file, as well as the file that contains the declarative rules. Afterwards, the module determines which rules apply to which pairs of verb classes of each verb present in ViPEr. The following pseudo-code illustrates this process:

```
for each (ViPEr verb v)
 for each (Declarative rule r)
  for each (Verb class vc1 of verb v)
   for each (Verb class vc2 of verb v)
    if (r matches verb classes vc1 and vc2)
       {
        generate xip rule for vc1 and vc2
          with the xip condition in declarative rule r
       }
```

A declarative rule matches a pair of verb classes if:

– verb class `vc1` contains all the properties in `<props +>` of `v1`;
– verb class `vc1` contains all the property pairs in `<pairs +>` of `v1`;
– verb class `vc1` does not contain any of the properties in `<props ->` of `v1`;
– verb class `vc1` does not contain any of the property pairs in `<pairs ->` of `v1`;
– verb class `vc2` contains all the properties in `<props +>` of `v2`;
– verb class `vc2` contains all the property pairs in `<pairs +>` of `v2`;
– verb class `vc2` does not contain any of the properties in `<props ->` of `v2`;
– verb class `vc2` does not contain any of the property pairs in `<pairs ->` of `v2`;

If a rule matches a pair of verb classes, a XIP disambiguation rule is generated, that follows the format:

```
//Rule no <rule number>
//<vc1> - <ViPEr example>
//<vc2> - <ViPEr example>
//Generated by rule: <declarative rule>
|#1[markviper,lemma:<verb lemma>,<vc1>,<vc2>,<vc2>=~]|
if( <XIP condition> )
~
```

6 Transformation-based Learning Module

This module does not perform verb sense disambiguation. Instead, it attempts to improve the basis of the actual disambiguation performed by XIP. This is done by using the TBL algorithm to order the disambiguation rules produced by the module presented in the previous section, since rule-order is critical to the VSD task. Figure 5 shows the architecture of the system.

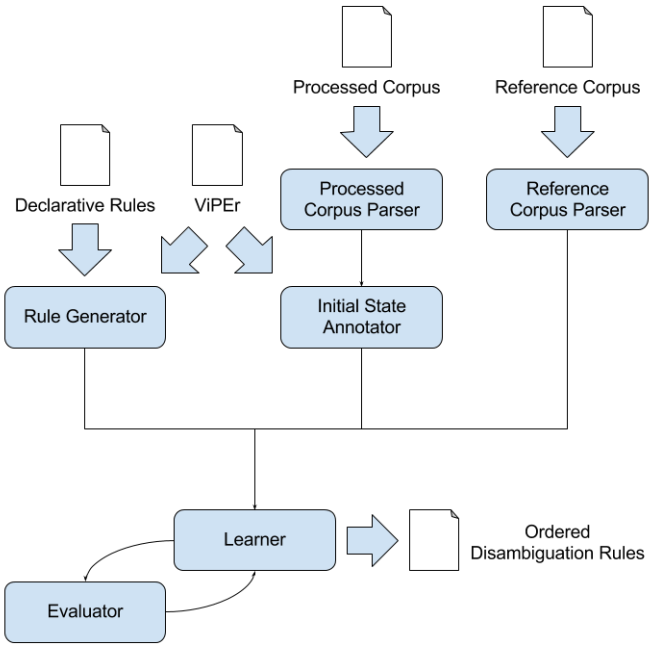

Fig. 5. TBL Module - Architecture

The system is composed by six modules:

1. **Rule Generator** - This is the same module described in the previous section. The TBL module operates upon the rules generated by this module. Thus, including the module in the system prevents having to maintain two code bases that perform the same exact functions;
2. **Processed Corpus Parser** - This module is responsible for parsing a *corpus* that has been previously processed by XIP. The result of this module will be used as the learning *corpus* used by the TBL algorithm implementation;
3. **Reference Corpus Parser** - This module, like the previous one, is responsible for parsing the reference *corpus* into the system. This *corpus* is then used by the Evaluator module in the learning process;

4. **Initial State Annotator** - This module takes as input both ViPEr and the result of the parsing performed by the Processed Corpus Parser module, and adds all appropriate ViPEr classes to every verb in the *corpus*;
5. **Learner** - This module uses the result of the Processed Corpus Parser module, along with the generated XIP disambiguation rules, to perform the learning process. This process then produces a set of ordered XIP disambiguation rules;
6. **Evaluator** - This module evaluates the results of applying a given XIP disambiguation rule to the working *corpus*. These results are used in the learning process.

7 Evaluation

To evaluate the impact of the proposals presented in the previous section, the PAROLE *corpus* [13] was used. This *corpus* contains around 250,000 words, of which 12,494 are ambiguous full verbs. Since there is no other large enough annotated *corpus*, the PAROLE *corpus* was partitioned in ten folds, and the TBL module was evaluated with a 10 fold cross-validation technique to avoid bias of the results.

The baseline considered for evaluating the new modules is the accuracy of the STRING chain using the rules generated by the previous approach.

7.1 Rule-generation module evaluation

Table 1 presents the results of evaluating the STRING chain without the use of the post-XIP classifiers: the Most Frequent Sense (MFS) classifier; and the Naive-Bayes classifier.

Meanings	Instances	Correctly Disamb.	Fully Disamb.	Wrongly Disamb.	Not Disamb.	Ambiguous Left
Former rules	12,494 (100%)	1,959 (15.68%)	1,458 (11.67%)	490 (3.92%)	10,045 (80.4%)	10,546 (84.41%)
New rules	12,494 (100%)	4,556 (36.47%)	2,541 (20.34%)	1,125 (9%)	6,813 (54.53%)	8,828 (70.66%)

Table 1. Results comparison (no classifiers)

The new rules are an overall improvement over the former rules, increasing the accuracy of the system from 15.68% to 36.46%, a 20.78% increase. Note that the number of rules increased from 1,487 to 10,691 (a 618.96% increase). This means STRING takes 647 seconds to process the profiling *corpus* (a 5,000 word

partition of the CETEMPúblico *corpus*) with the new rules, as opposed to the 548 seconds it took to process the *corpus* with the previous set of rules.

Table 2 presents the results of evaluating STRING with the addition of the MFS classifier. Note that the **Fully Disambiguated**, **Not Disambiguated** and **Ambiguous Left** columns were omitted, since the MFS classifier fully disambiguates every verb, thus these data are not relevant.

Meanings	Instances	Correctly Disamb.	Wrongly Disamb.
Former rules	12,494 (100%)	10,130 (81.08%)	2,364 (18.92%)
New rules	12,494 (100%)	9,700 (77.64%)	2,794 (22.36%)

Table 2. Results comparison (with the MFS classifier)

An additional 430 verbs were wrongly disambiguated by the new set of rules. This is due to a subset of the verbs that were wrongly classified by the new rules while also being reduced to a single ViPEr class, meaning they are no longer considered ambiguous. As a consequence, they are not considered by the MFS classifier. In addition, despite the new rules fully disambiguating more verbs, they do not overlap completely with those fully disambiguated by the former rules. This causes the MFS classifier to apply to verbs to which it would not apply to using the former rules, and where, unfortunately, the MFS is not the correct choice. These results indicate that, since the MFS classifier has such a high accuracy, it is more important to have a low error count than a high success count.

Finally, Table 3 presents the results of evaluating STRING with the addition of the MFS and the Naive-Bayes classifiers. These settings represent the use of the STRING chain in a live environment.

Meanings	Instances	Correctly Disamb.	Wrongly Disamb.
Former rules	12,494 (100%)	10,233 (81.9%)	2,261 (18.1%)
New rules	12,494 (100%)	9,864 (78.95%)	2,630 (21.05%)

Table 3. Results comparison (with the MFS and the Naive-Bayes classifiers)

The Naive-Bayes classifier improves the accuracy of the system with both sets of rules. Unfortunatelly, the new set of rules still displays lower accuracy than the former set of rules.

7.2 TBL module evaluation

Table 4 presents the results of evaluating the STRING chain with both classifiers, comparing the former rules with the rules reordered by the TBL module.

Meanings	Instances	Correctly Disamb.	Wrongly Disamb.
Former rules	12,494 (100%)	10,233 (81.9%)	2,261 (18.1%)
TBL reordered rules	12,494 (100%)	9967 (79.77%)	2527 (20.23%)

Table 4. Results comparison (with the MFS and the Naive-Bayes classifiers)

The reordered rules present a slight improvement of 0.82% over the unordered rules, from 78.95% to 79.77%. However, the former rules are still 2.13% more accurate, showing an accuracy of 81.9% as opposed to the 79.77% of the reordered rules.

8 Conclusions

This paper set out to improve the current verb sense disambiguation task within the STRING chain. The major flaw of the previous system had to do with the fact that much of the linguistic knowledge was embedded in the code of the rule generation module. Besides, the rules followed a pre-set manual ordering. In this project, a new rule generation module was built, based on declarative rules, which render explicit the liguistic knowledge required for VSD and allow for a more efficient generation and debugging. Furthermore, a TBL algorithm was implemented for the rule order problem. Besides, new corpora for VSD have been annotated.

Though results have not improved from the previous settings, the new set of generated rules is several times larger (10,691 rules, as opposed to the 1,487 that made up the former set of rules), and the declarative rules on which they are based can be used with the new debugging tools to improve the system's overall performance. For example, when applied to a large corpus annotated with verbs' senses, some of the generated rules were never used, while others yet to be declared may still be formed. The MFS eliminates all the remaining readings of a given verb, but other, less drastic approaches can be implemented and tested. Thus, there is still room to improve in future work.

References

1. Nuno Mamede, Jorge Baptista, Cláudio Diniz, and Vera Cabarrão. STRING: A Hybrid Statistical and Rule-Based Natural Language Processing Chain for Portuguese. In *Proceedings of the 12th International Conference on Computational Processing of the Portuguese Language - Demo sessions* `http://www.propor2012.org/demos/DemoSTRING.pdf`, PROPOR 2012, 2012.
2. Jorge Baptista. ViPEr: A Lexicon-Grammar of European Portuguese Verbs. In *Proceedings of the 31st International Conference on Lexis and Grammar*, pages 10–16, September 2012.
3. Tiago Travanca. Verb Sense Disambiguation. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, June 2013.
4. Gonçalo Suíssas. Verb Sense Classification. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, November 2014.
5. Alexandre Vicente. LexMan: um Segmentador e Analisador Morfológico com Transdutores. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, June 2013.
6. Hugo Almeida. Suffix Identification in Portuguese using Transducers. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, November 2016.
7. Cláudio Diniz. Um conversor baseado em regras de transformação declarativas. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, October 2010.
8. Cláudio Diniz, Nuno Mamede, and João Pereira. RuDriCo2 - a faster disambiguator and segmentation modifier. In *II Simpósio de Informática (INForum)*, pages 573–584, September 2010.
9. Ricardo Ribeiro. Anotação Morfossintáctica Desambiguada do Português. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa.
10. David Rodrigues. Uma evolução do sistema ShRep. Optimização, interface gráfica e integração de mais duas ferramentas. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, November 2007.
11. Salah Aït-Mokhtar, Jean Pierre Chanod, and Claude Roux. Robustness beyond shallowness: Incremental deep parsing. *Nat. Lang. Eng.*, 8(3):121–144, June 2002.
12. Eric Brill. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, 21(4):543–566, December 1995.
13. M. Nascimento, P. Marrafa, L. Pereira, R. Ribeiro, R. Veloso, and L. Wittmann. LE-PAROLE - Do corpus à modelização da informação lexical num sistema-multifunção. In *XIII Encontro Nacional da Associação Portuguesa de Linguística*, pages 115–134. Lisboa: APL/Colibri, 1998.