Dependency Parsing with Finite State Transducers and Compression Rules

Pablo Gamallo
Centro Singular de Investigación en
Tecnoloxías da Información (CiTIUS)
University of Santiago de Compostela, Galiza
pablo.gamallo@usc.es

Marcos Garcia
Universidade da Corunha
Grupo LyS, Departamento de Letras
Faculdade de Filologia, Corunha, Galiza
marcos.garcia.gonzalez@udc.gal

May 18, 2018

Abstract

This article proposes a syntactic parsing strategy based on a dependency grammar containing formal rules and a *compression* technique that reduces the complexity of those rules. Compression parsing is mainly driven by the 'single-head' constraint of Dependency Grammar, and can be seen as an alternative method to the well-known *constructive* strategy. The compression algorithm simplifies the input sentence by progressively removing from it the *dependent* tokens as soon as binary syntactic dependencies are recognized. This strategy is thus similar to that used in deterministic dependency parsing. A compression parser was implemented and released under General Public License, as well as a cross-lingual grammar with Universal Dependencies, containing only broad-coverage rules applied to Romance languages. The system is an almost delexicalized parser which does not need training data to analyze Romance languages. The rule-based cross-lingual parser was submitted to *CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies.* The performance of our system was compared to the other supervised systems participating in the competition, paying special attention to the parsing of different treebanks of the same language. We also trained a supervised delexicalized parser for Romance languages in order to compare it to our rule-based system. The results show that the performance of our cross-lingual method does not change across related languages and across different treebanks, while most supervised methods turn out to be very dependent on the text domain used to train the system.

1 Introduction

Syntactic analysis is a crucial module for many tasks relying on natural language processing and information extraction techniques such as summarization (Abdi et al., 2015), information retrieval (Vilares et al., 2014), topic detection (Lee et al., 2007), named entity recognition and entity linking (Derczynskia et al., 2015), text mining (Tseng et al., 2007), text classification (Uysal and Gunal, 2014; Sokolova and Lapalme, 2009), or sentiment analysis (Vilares et al., 2015; Severyn et al., 2015). Besides, syntactic information is required to improve semantic applications. More precisely, knowledge on syntactic parse trees turns out to be useful to yield accurate semantic word models and embeddings based on the distributional context of words (Saif et al., 2015), as well as to extract semantic relations (Zhang et al., 2008) and for semantic role labeling (Zhou et al., 2011).

In this article, we propose a new (rule-based) finite-state parsing strategy based on dependencies, which minimizes the complexity of rules by using a technique we call *compression*. Compression parsing is driven by the single-head constraint of Dependency Grammar. It simplifies the input string by progressively removing the *dependent* tokens as binary syntactic dependencies are recognized. At the end of the compression process, if all the dependencies in the sentence are recognized, the input string should contain just one token representing the main head (i.e., the *root*) of the sentence. This strategy was inspired by the *Right* and *Left* Reduce transitions used in deterministic dependency parsing (Nivre, 2003; Nivre et al., 2004).

Deterministic dependency parsing (called 'transition based') relies on supervised techniques requiring fully analyzed training corpora (syntactic treebanks). Given that supervised techniques tend to have a loss of precision when applied to texts of domains and genres different from those used for training (Rimell et al., 2009), they need too much manual effort to create, adapt, or modify the training corpus to the target domain. It is generally accepted that supervised classifiers require some type of domain adaptation when both the training and test data sets belong to different domains. In particular, the accuracy of statistical parsers degrades when they are applied to different genres and domains (Rimell et al., 2009; Gildea, 2001). By contrast, we propose a dependency parsing strategy based on elementary linguistic information which may be applied on different domains with similar accuracy and whose performance is close to the state-of-the-art (Section 6.4).

A system based on the compression strategy was implemented in Perl and released under General Public License: *DepPattern*. In addition, we defined a high level grammar language to define dependency-based rules and developed a grammar compiler in Ruby to generate compression parsers in several languages (Gamallo and González, 2011). DepPattern has been used for several web-based IE applications, namely Open Information Extraction from Wikipedia (Gamallo et al., 2012), extraction of semantic relations with distant supervision (Garcia and Gamallo, 2011), and extraction of bilingual terminologies from comparable corpora (Gamallo and Pichel, 2008). It has also been integrated into commercial tools, e.g. Linguakit.[1] and Avalingua[2]

Some experiments were performed to compare our rule-based system with supervised approaches. For this purpose, we implemented a specific parser with DepPattern, called *MetaRomance*, which is based on a very basic cross-lingual grammar for Romance languages. MetaRomance was compared to the systems that participated at *CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies* (Zeman et al., 2017). The reported results showed that our system's performance remains stable across related languages and different treebanks for the same language, while most supervised methods are very dependent on the specific properties of the treebank used for training (Garcia and Gamallo, 2017).

The use of grammars in recent dependency parsers is almost non-existent since they are considered too cost to build and maintain. However, this drawback can be minimized by incorporating into the parsing system *light-weight* grammars. More precisely, our system makes use of small grammars containing broad-coverage syntactic information that can be applied to several related languages and different content domains. The cost of manually creating grammar rules is also reduced by providing a suitable rule notation for linguists.

The remainder of this article is organized as follows. Sections 2 and 3 introduce different approaches on both dependency-based and finite-state parsing, including deterministic dependency parsing and constructive parsing. Then, Section 4 is focused on the description of our compression strategy. Next, Section 5 provides a general view of the implemented system: DepPattern and MetaRomance. Section 6 reports the diverse experiments performed with MetaRomance using the treebanks provided by CoNLL 2017. Finally, some conclusions are drawn in Section 7.

2 Dependency-based Syntactic Parsing

Following Nivre (2006), there are two traditions in dependency parsing: grammar-driven and data-driven parsing. Within each tradition, it is also possible to distinguish between two different approaches: non-deterministic and deterministic parsing. In the latest years, most work on dependency parsing has been developed within the approach of data-driven deterministic parsing, which is also known as *transition-based parsing*, in opposition to non-deterministic strategies such as *graph-based dependency parsing*. In addition, in the latest years there has been an important growth in cross-lingual parsing research, which is the main application field of our parsing system.

2.1 Graph-based Dependency Parsing

A graph-based dependency parser, also known as discriminative parser, starts with all valid dependencies between the nodes/words of a sentence. This am-

[1]`https://linguakit.com/`

[2]`http://cilenis.com/en/avalingua/`

biguous structure is represented as a completely connected graph whose edges are weighted according to a statistical model. Then, in the disambiguation (or discriminative) process, the parser tries to find a tree covering all nodes (words) in the graph that maximizes the sum of the weighted edges (McDonald and Pereira, 2006; Carreras, 2007; Martins et al., 2010). Therefore, as in non-deterministic parsing, this technique generates first all analyses and, then, selects the most probable one according to the statistical model.

2.2 Transition-based Dependency Parsing

This strategy consists in inducing statistical models in combination with a deterministic strategy based on shift-reduce parsing (Nivre, 2004; Yamada and Matsumoto, 2003; Gómez-Rodríguez and Fernández-González, 2012). Nivre et al. (2004) uses the arc-eager algorithm as the parsing strategy. This algorithm is simplified by just using two transitions on undirected dependencies by Gómez-Rodríguez et al. (2015), by just using two transitions on undirected dependencies (the head-dependent information is erased), so as to avoid error propagation.

There have been recently many extensions of these two data-driven strategies. In Zhang and Clark (2009), the authors propose a transition parsing process combined with a beam-search decoder instead of greedy search. The use of beam search allows the correction of local decision errors by global comparison.

In general, transition-based systems are more efficient (linear time in the best cases), but graph-based parsers tend to be more accurate in terms of performance. In this sense, the best dependency parser at the CoNLL 2017 shared task was a graph-based parser (Dozat et al., 2017).

As we will show later, the main problem of these supervised learning strategies arises when the test sentences belong to linguistic domains very different from those found in the training corpus. We will show later the negative effect on system performance when the test and training data sets does not belong to the same domain and genre. As hand labeling data in new domains is a costly enterprise, the domain adaptation problem is a fundamental challenge in machine learning applications. Note that many NLP annotated resources are based on text from the news domain (in most cases, the Wall Street Journal), which is a poor match to other domains such as biomedical texts, electronic mails, transcription of meetings, administrative language, etc. (Daumé III, 2006, 2007).

2.3 Cross-Lingual Parsing

In the last few years the work on cross-lingual parsing (analyzing a target languages using resources from one or more source languages) has increased. In this regard, we can define two main cross-lingual parsing approaches: (i) data transfer, and (ii) model transfer. On the one hand, data transfer approaches obtain annotated treebanks of the target variety by projecting the syntactic information from the source data. Some methods use parallel corpora (Hwa et al., 2005; Ganchev et al., 2009; Agić et al., 2016) while others directly create an artificial treebank taking advantage of machine translation (Tiedemann and Agić, 2016).

On the other hand, model transfer approaches train supervised system (using the source data) that can be used to parse a target language (Zeman and Resnik, 2008). The emergence of different initiatives promoting harmonized annotations increased the research in this area, testing different strategies such as delexicalization and multi-source training (McDonald et al., 2011; Täckström et al., 2012).

Besides, some recent works addressed multilingual parsing using a single model, trained in a combination of various treebanks, to analyze different languages (Vilares et al., 2016; Ammar et al., 2016).

The growth in cross-lingual parsing research has given rise to a recent shared task at VarDial 2017 (Zampieri et al., 2017), Cross-lingual Dependency Parsing (CLP) (Tiedemann, 2017). CLP is a shared task focused on parsing selected target languages without annotated training data, but having treebanks in one or two closely related languages Rosa et al. (2017).

With the emergence of Universal Dependecies as the practical standard for multilingual PoS and syntactic dependency annotation, it is possible to develop universal rule-based strategies requiring no training data, and relying on basic rules exploiting the UD criteria. The Universal Dependency Parser, described in Martínez Alonso et al. (2017), is a good example of this unsupervised strategy. Similarly, our work performs cross-lingual parsing without training data, but with two differences: the fisrt cross-lingual grammar we have written is focused on Romance languages, and the parser relies on basic rules implemented as cascades of finite-state transducers (FST).

3 Finite-State Parsing Techniques

Finite-state technology has attractive properties for syntactic parsing, such as conceptual simplicity, flexibility, and efficiency in terms of space and time. It permits to build robust and deterministic parsers. Most finite-state based parsing strategies use cascades of transducers and are known as *constructive* parsers.

3.1 Constructive Parsing

Parsing based on cascades of finite-state transducers can be viewed as a sort of string transformation. Finite-state transducers introduce progressively markings and labels within the input text. Transducers are arranged in cascades (or layers), where the subsequent transducer takes the output of the previous one as input. After a certain number of cascades, the initial input (which is a tagged sentence) is transformed into a structured text enriched with syntactic marks, such as chunk boundaries, labels for heads, special markers for functions or for relations between heads, etc. This strategy, known as *constructive*, progressively constructs the linguistic representation within the input string, by making use of rules/transducers organized at different levels (or layers) of complexity.

Most of finite state strategies aim to construct phrase based structures instead of dependency graphs (Ait-Mokhtar et al., 2002; Ciravegna and Lavelli, 2002; Kokkinakis and Kokkinakis, 1999; Ait-Mokhtar and Chanod, 1997; Abney, 1996; Joshi, 1996; Grefenstette, 1996). In general, the construction of these structures is performed with three main cascades/layer of rules: chunking, head recognition, and attachment. The first layers of rules transform the tagged input into sequences of symbols representing basic chunks. Then, further rules take those chunks as input to add new symbols marking the heads of each chunk and, finally, new rules are applied on the output of the previous ones to annotate the identified heads with labels of syntactic functions (attachment).

The number of finite-state approaches focused on constructive dependency parsing is much more limited. We can merely cite the work by Oflazer (2003), where the input string is progressively enriched by additional symbols encoding dependency relations between words.

Finally, there is also work on finite-state parsing which is not properly based on the constructive approach. For instance, the paper by Koskenniemi et al. (1992) describes a finite-state strategy for eliminative parsing, inspired by Constraint Grammar. Sekine (2000) described a data-driven dependency parser (evaluated on Japanese) with a deterministic finite-state transducer. It is much faster than other data-driven parsers using similar machine learning techniques for training, but its accuracy is lower.

3.2 Basic Properties

The finite-state strategy often relies on four fundamental properties: *easy-first parsing*, *procedural linguistic knowledge*, *robust parsing*, and *speed*.

Easy-first parsing means that the simplest tasks must be done first, leaving the harder decisions for the last steps of the parsing process. Parsing proceeds by growing *islands of certainty* (Abney, 1996). Once the easy tasks have been solved with high accuracy, then we can try to solve more difficult issues (e.g., long distance attachments). Notice that this principle does not harmonize well with the incremental (left-to-right) strategy. Whereas incrementality simulates how we process sentences, easy-first simulates how we solve problems.

Easy-first parsing is the main cause of the following property: procedural linguistic knowledge. To solve problems, it is necessary to take good decisions, and these decisions are taken, not by the parsing algorithm, but by the linguist's intuitions when writing the rules. According to Ait-Mokhtar and Chanod (1997), the ordering of transducers is in itself a genuine linguistic task. The linguist decides on the order of the transducers by addressing the easy tasks first. It means that the linguistic knowledge is not totally declarative: the linguistic information contains elements that are relevant for the parsing process. This may sound like a severe disadvantage of the approach, but Ait-Mokhtar and Chanod (1997) argue that this view of parsing is instrumental in achieving robust parsing in a principled fashion.

A common problem with traditional parsers is that correct low level decisions are rejected because full analysis cannot be performed, due to incompleteness of the grammar (Abney, 1996). However, for finite-state methods (and in general for deterministic parsing), the issue of grammaticality is independent from the parsing process. They assume that any text sentence is a valid input string. The objective is to perform robust parsing. A robust parser produces output as soon as it recognizes a piece of linguistic information within a substring of the input sentence.

Parsers based on finite-state technology are the fastest systems among those achieving linear time complexity. So, they are scalable as the input text increases in size and are easily integrated into IE applications exploring the Web as corpus.

Finite-state transducers not only give simple and efficient parsing strategies but also provide a natural and unified way of performing syntactic analysis. Some authors claim that finite-state models are one of the best formalisms to represent accurately complex linguistic phenomena (Roche, 1997, 1999). It is, therefore, assumed that the development of FST-based parsing strategies goes far beyond regular grammars. There has been a stream of work in using finite-state methods in parsing which is based on approximating context-free grammars with finite-state grammars, which are then processed by efficient methods (Roche, 1997; Laporte, 1996; Pereira and Wright, 1997; Grimley-Evans, 1997; Johnson, 1998; Nederhof, 2000). Moreover, according to Yli-Jyrä (2005), some FST-based parsing strategies are closely related to some mildly context-sensitive grammars, which may deal with the complexity of crossing dependencies and non-projectivity.

4 A Compression Parsing Strategy

We propose yet another FST based method, very similar to the constructive approaches, but making use of a similar strategy to the shift-reduce algorithm as incremental parsing. We call it *compression parsing.* It consists of a set of transducers/rules that *compress* the input sequence of tokens by progressively removing the dependent tokens as soon as dependencies are recognized. So, at each application of a rule, the systems reduce the input and make it easier to find new dependencies in further rule applications. In particular, short dependencies are recognized first and, as a consequence, the input is simplified so as to make lighter the recognition of long distance dependencies. This is inspired by the easy-first strategy.

4.1 Introduction to the Compression Method

The input of our parsing method is a sequence of disambiguated tagged tokens, where each token is associated with two pieces of information: a PoS tag representing the basic morpho-syntactic category of the token (NOUN, VERB, ADP, etc.)[3] and a feature structure containing other relevant information of the token: morphological information (number, tense, person, etc.), lemma, token string, token position, etc. Tagged tokens are the elementary objects of the parsing

[3] `http://universaldependencies.org/u/pos/all.html`

Table 1: Parsing with compression transducers of the sentence “The coach needs a long break”

L_4		V<t:**needs**,p:3>
	T_4	$(3, 6)$
L_3		V<t:**needs**,p:3> N<t:**break**,p:6>
	T_3	$(3, 2)$
L_2		N<t:**coach**,p:2> V<t:**needs**,p:3> N<t:**break**,p:6>
	T_2	$(2, 1), (6, 4)$
L_1		DT<t:**The**,p:1> N<t:**coach**,p:2> V<t:**needs**,p:3> DT<t:**a**,p:4> N<t:**break**,p:6>
	T_1	$(6, 5)$
L_0		DT<t:**The**,p:1> N<t:**coach**,p:2> V<t:**needs**,p:3> DT<t:**a**,p:4> A<t:**long**,p:5> N<t:**break**,p:6>

process, while rules, which are implemented as finite state transducers, operate on tagged tokens. More precisely, rules successively identify dependencies between tokens, remove the dependents (if required) from the input sequence, and update (if required) the feature structures of the heads. Let’s take the sentence “The coach needs a long break” analyzed in Table 1 with the following set of compression dependency rules:

$$\begin{aligned}
T_1\texttt{:} \quad & \texttt{N<\$f>} \longleftarrow \texttt{A<\$f>} \quad \texttt{N<\$f>} \\
T_2\texttt{:} \quad & \texttt{N<\$f>} \longleftarrow \texttt{DT<\$f>} \quad \texttt{N<\$f>} \\
T_3\texttt{:} \quad & \texttt{V<\$f>} \longleftarrow \texttt{N<\$f>} \quad \texttt{V<\$f>} \\
T_4\texttt{:} \quad & \texttt{V<\$f>} \longleftarrow \texttt{V<\$f>} \quad \texttt{N<\$f>}
\end{aligned} \tag{1}$$

Rules in (1) are regular expressions performing pattern matching and reduction (by replacing a string with an empty string). The right-hand side of a rule is a pattern aimed at recognizing two adjacent dependent tokens. If there is a sequence of tokens in the input sentence matching the right-hand side of the current rule, the sequence is reduced to a single node (the head), whose PoS tag is given in the left-hand side of the rule. Each element of a pattern is a token represented by a PoS tag (`V, N, A, DT`)[4] and a feature structure specifying further linguistic information: token string, lemma, position, gender, number, etc. Symbol `<$f>` stands for a generic feature structure. Rules will be described in more detail later in Section 4.2.

Each rule is implemented as a transducer that recognizes ‘head-dependent’ token relations and removes the *dependent* tokens from the input string. It is applied from left to right until it reaches the end of the input sentence. The successive application of these transducers/rules simplifies and reduces the search space of the next rule to be applied. The transducer/rule T_i is run with L_{i-1} as input, and it produces L_i as output. In Table 1, parsing begins at level L_0, which represents the tagged sentence. Each token in the sentence consists of a PoS tag and a specific feature structure (within ‘$<\ >$’). For the sake of simplicity, only two features are shown: both the token and its position

[4]`V` is a verb, `N` a noun, `A` an adjective, and `DT` a determiner.

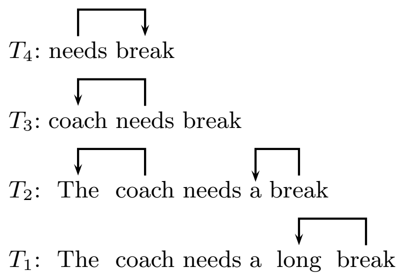

T_4: needs break
T_3: coach needs break
T_2: The coach needs a break
T_1: The coach needs a long break

Figure 1: The four levels of analysis of the sentence "The coach needs a long break"

in the sentence. Lemmatization and morphological information (number, tense, person, etc.) are missing here. The binary dependencies that are recognized are stored as pairs: *(position-of-head, position-of-dependent).*

The application of T_1 on the initial input sentence L_0 recognizes a specific adjective-noun dependency (the head is the token in position 6, and the dependent in 5), and yields a reduced sentence L_1 by removing the recognized adjective. The last level (L_4), which has been reached after applying the four rules, represents the main head (or root) of the sentence. Figure 1 helps to visualize the application of the four rules in the parsing process.

4.2 Description of rules

A compression rule is defined as a tuple $< P, Arc, \triangle, Reduce >$, where:

- P is a pattern of tagged tokens, defined as a regular expression, whose general form is $\alpha X \beta Y \gamma$. X and Y are non-empty strings representing two tagged tokens, considered as the *core elements* of the pattern; while α, β, and γ represent the left, middle, and right contexts, respectively, of the core elements; they may be empty.

- *Arc* is the action that creates a dependency link between the core elements (X and Y), when a subsequence of the tagged input is matched by the pattern; two types of arcs are distinguished: $Left\_Arc$ adds a dependency link between X and Y, X being the dependent and Y the head; *Right_Arc* creates a dependency link between X and Y, X being the head and Y the dependent. This action also assigns a label (subject, modifier, adjunct, etc) to the dependency.

- $\triangle$ is a set of operations (*Agreement*, *Add*, *Correction*, *Inherit*, etc.) that are applied on the feature structure of the core elements; they can be used to perform very different actions: verifying if the two core elements (i.e., head-dependent) share a set of feature-values, adding new feature-values to the head, modifying some values of the head, correcting PoS tags, allowing the head to inherit selected values from the dependent, etc. *Add* and *Inherit* can be seen as constructive operations.

- *Reduce* is the action that removes the *dependent* from the input string; this action can be suspended if the dependent token is likely to be the head in other dependencies that have not been recognized yet. So, the dependent will not be reduced until all its potential dependent tokens have been recognized.

For illustrative purposes, let us suppose we want to define a rule that identifies a dependency between two verbs, namely an auxiliary verb and a past participle form, to be applied on the following sequence of tagged tokens:

```
VERB<token:has,lemma:have|type:A|tense:P|mode:I|number:S|person:3|pos:0>
ADV<token:never|lemma:never|pos:1>
VERB<token:seen|lemma:see|type:M|tense:P|mode:P|number:0|person:0|pos:2>
```

This is the tagged sequence for “has never seen”. Each line stands for a token, constituted by a PoS tag and a feature structure. A feature structure consists of a set of ‘attribute:value’ elements separated by vertical bars ‘|’ . The information contained in the feature structure was mainly specified by a PoS tagger, even if previously applied rules can also add or modify some features. The rule that identifies the dependency between the auxiliary verb and the past participle form could be constituted by the following elements:

- The pattern:

  ```
  (X,Y)=(VERB<${f}type:A${f}>) (?:ADV<$f>)?  (VERB<${f}mode:P${f}>)
  ```

  where each tagged token is enclosed by round brackets and then grouped into a single element. The two core elements (the two verbs) are stored into the variables X and Y. The element starting with ‘?:’ introduces a contextual token, the adverb, which is not stored in any variable since it will not be involved in further operations triggered by the rule. Moreover, well-known wildcard characters elaborate the pattern. In this example, the question mark introduces an optional element, the adverb. Morphological information, relevant for the matching process, appears between the symbol ‘${f}’, which is used to represent a generic chain of ‘attribute:value’ pairs. The first verb is of type ‘A(uxiliary)’ and the last one has the mode ‘P(articiple)’.

- $Left\_Arc(\mathit{spec}, X, Y)$: this operation creates a labeled ‘dependent-head’ arc between the two core elements. In this case, the dependent is the left element, X, while the head is Y, and the label *spec*. Note that we make the assumption that the auxiliary verb is the dependent taking the lexical verb as the head. This choice is relevant for IE applications working with dependencies between lexical items.

- Some operations belonging to the set $\triangle$. In this example, we can apply the following two specific operations:

  - *Inherit*$(tense, mode, number, person)$: The head inherits the values from a set of morphological attributes of the dependent. This operation is required if a further rule needs to check the agreement between this verb and any candidate to be its grammatical subject.
  - *Add*$(perfect : P)$: This adds a new 'attribute:value' to the feature structure of the head, namely the P(resent) perfect tense. This information was not provided by the PoS tag since it only gives tense information on simple forms. Such an information can be relevant later for rules involved in temporal relations.

- *Reduce*: this operation removes the dependent token from the input sequence. It is implemented as a replacement operation (by replacing a token with an empty string):

  `ADV<$f>?  VERB<$f>` $\leftarrow$ `VERB<$f> ADV<$f>?  VERB<$f>`

  The right-hand side of the operation contains all the tagged tokens of the pattern (both core and contextual elements), while the left-hand side rewrites the same elements except that identified as being the dependent.

This rule rewrites the input expression to produce the following output:

```
ADV<token:never|lemma:never|pos:1>
VERB<token:seen|lemma:see|type:M|tense:P|mode:I|number:S|person:3|perfect:P|pos:2>
```

where the auxiliary verb has been removed, but not its main morphological features, which have been inherited by the lexical head. In addition, new information has been created within the feature structure of the head: the compound tense present perfect. This is the input of the next rules.

Compressing rules not only reduce the complexity of the search space (or input) of the remaining rules to be applied, but also construct relevant information (by adding linguistic features) for the application of those rules. In particular, a rule may store in the head relevant information of the removed dependent (Inherit operation), generates new attributes or modify values from existing attributes (Add operation), and also corrects odd tagged PoS tags (Correction, Garcia and Gamallo (2010)). In sum, the main contribution of our work is to define compressing rules as the integration of two parsing techniques: both transition-based and constructive parsing. On the one hand, the rules reduce the search space by removing the dependent tokens and, on the other hand, they can add relevant information to the head tokens by making use of operations such as *Add* or *Inherit*. Rules compress the input sequence of tokens so as to make it easier the identification of more distant dependencies.

The *Inherit* operation is required to avoid information loss because of the reduction operation. It permits to transfer linguistic features to heads before removing the dependent tokens from the search space. This can be considered as one of the main contributions of our dependency-based strategy. In combination with *Add*, *Inherit* is able to transfer relevant information from auxiliary, light, or modal verbs to their main verbs. It can also be used to model coordination by transferring categorial information from coordinated structures to the coordinating conjunction, which enables subject-verb agreements. For instance, in the sentence "Paul and Mary are eating", the *Inherit* operation allows the coordinating conjunction "and" to inherit the nominal category of their parts and, by means of the *Add* operation, it can get the *plural* number. In addition, *Inherit* can also be used to transfer relevant morphological information (third person, plural, present tense) to the root verb "eat" from the auxiliary "are". This way, the verb '(are) eating' (3rd person and plural) agrees with its subject "Paul and Mary". A similar solution has also been described by Holan and Zabokrtsky (2006).

4.3 Rule Ordering: Easy First

As was mentioned above (Section 3.2), the ordering of rules in a FST parser is a genuine linguistic task. Rules are ordered in such a way that the easiest tasks, for instance short dependencies, are performed first. As in Eisner and Smith (2010), Goldberg and Elhadad (2010), Tratz and Hovy (2011), or Versley (2014), we assume that correct parsers exhibit a short-dependency preference: a word's dependents tend to be close to it in the string. The fact of identifying first easy syntactic structures, such as those including modifiers and specifiers, allows us to easily find later distant links, for instance those relating verbs with subordinate conjunctions. Let us take the expression: "if the former president of USA says…". We can find here a long-distance dependency between the verb 'says' and the conditional conjunction "if". In most sentences, both words are not adjacent since a great variety of tokens can be interposed. However, in a compression approach, we can guess that dependency by making use of a very simple pattern consisting in a subordinate conjunction (type:S) appearing immediately to the left of a verb (last rule T_6 below in 2). We just need the following sequence of transductions/rules:[5]

$$\begin{aligned}
T_1:&\quad \texttt{ADP<\$f>} \longleftarrow \texttt{ADP<\$f>}\quad \texttt{N<\$f>}\\
T_2:&\quad \texttt{N<\$f>} \longleftarrow \texttt{ADJ<\$f>}\quad \texttt{N<\$f>}\\
T_3:&\quad \texttt{N<\$f>} \longleftarrow \texttt{DT<\$f>}\quad \texttt{N<\$f>}\\
T_4:&\quad \texttt{N<\$f>} \longleftarrow \texttt{N<\$f>}\quad \texttt{ADP<\$f>}\\
T_5:&\quad \texttt{VERB<\$f>} \longleftarrow \texttt{N<\$f>}\quad \texttt{VERB<\$f>}\\
T_6:&\quad \texttt{VERB<\$f>} \longleftarrow \texttt{CONJ<\$\{f\}type:S\$\{f\}>}\quad \texttt{VERB<\$f>}
\end{aligned}\tag{2}$$

[5]To simplify, rule notation is focused on just the final *Reduce* operation.

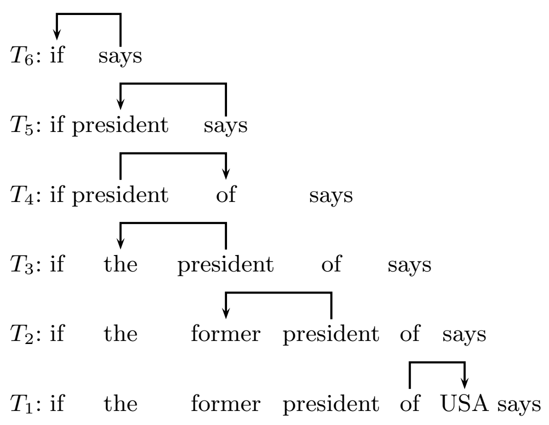

T_6: if says
T_5: if president says
T_4: if president of says
T_3: if the president of says
T_2: if the former president of says
T_1: if the former president of USA says

Figure 2: The six levels of analysis of "if the former president of USA says..."

In Figure 2, we show the application of the six rules on the input expression, as well as the effect of the *Reduce* transition at each level.

Each rule processes the input from left to right repeatedly as long as new dependencies satisfying the pattern are found. Rules are checked top-down following the rank imposed by the linguist. When the parser reaches the last rule of the ranked list, if at least one dependency has been identified, the parser starts again from the beginning until no new dependency is found. So, the parser works iteratively until no change is made. If the grammar is not complete, the parser produces partial parses.

Our approach takes into account the ideas on rule ordering by Arnola (1998). According to him, ambiguous natural language sentences can be parsed deterministically in linear time if the linguistic rules are ordered following two very basic principles: short-distance rules must be applied in first place, but in the case of ambiguous constructions, the most specific option must be applied first.

4.4 Shift-Reduce Parsing

Our rules are inspired by *D-rules*, which consist in describing possible dependency relations between word categories (PoS tags) (Covington, 2001). Nivre (2003) defined a shift-reduce parsing which uses oriented D-rules to represent left and right dependencies. It applies a set of actions bottom-up in a left to right and deterministic manner on consecutive pairs of words (A,B) (A<B) in a sentence. Our compressed parsing strategy was inspired by this type of deterministic parsing. The basic actions (also called *transitions*) in deterministic dependency parsing are the following (Nivre, 2003; Yamada and Matsumoto, 2003):

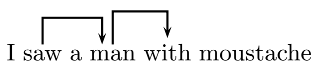
Shift adds no relation between A and B and moves to the right, by making

(B,C) the new target words, where (B<C) .

Right-Reduce constructs a dependency relation between B (the head) and A (the dependent), and A is thus eliminated for further considerations.

Left-Reduce constructs a dependency relation between A (the head) and B (the dependent), and B is thus eliminated for further considerations.

This is known as the "arc-standard algorithm". It is important to note that, as in our compressed approach, the dependent word is removed from the search space after performing *Left* and *Right Reduce.* The dependent is removed because its head was already found, given the single-head condition. However, this algorithm cannot be fully incremental (or left-to-right), since in some cases, the *Right-Reduce* operation has to be performed from right to left, namely when B is dependent of A but the head of C in the chain A<B<C. Incrementality requires each word to have found all its dependents before it is combined with its head. For instance, let's take the partial analysis (3) of the sentence 'I saw a man with moustache':



(3)

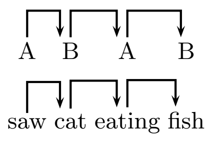
The noun "man" is the right-dependent of "saw" and the head of "with" (to its right). If we apply the *Right Reduce* action on the arc linking "saw" and " man", the noun would be removed before finding its right-dependent "with". In order to deal with multiple right-dependents and keep incrementality, it was necessary to improve the algorithm by changing *Right-Reduce.* In (Nivre, 2004), a new fully incremental algorithm was proposed: arc-eager dependency parsing. In this new approach, left dependents are processed bottom-up using the incremental left-to-right strategy with *Left-Reduce*, but right dependents are processed top-down. More precisely, *Right-Reduce* is transformed into two separate transitions: *Right-Arc* (a dependency is added without reduction) and *Reduce*, which is performed when all right dependents of a dependent word are found.

Similarly, in the compressed parsing we propose, the *Reduce* operation is not included in the actions that construct dependency arcs: *Left_Arc* and *Right_Arc.* If it is required, the grammarian can postpone such an operation for several reasons. We distinguish, at least, two reasons for suspending *Reduce* operation: the treatment of embedded structures and the introduction of syntactic ambiguity.

4.5 Embedded Structures

An embedded structure arises when the head of the relation A $\rightarrow$ B depends on the dependent token of the same kind of relation A $\rightarrow$ B (see Figure 4). This is the well-known recursive structure, which appears with completives, gerund clauses, and so on. For instance, in "I saw a cat eating fish", there is a verb

(A=eating) which is the head of its nominal argument (B=fish), but it also depends on the nominal argument (B=cat) depending, in turn, of the main verb (A=saw):



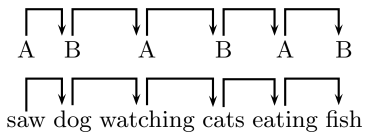
(4)

In order to deal with embedded structures, it is possible to use several strategies, involving or not the suspension of *Reduce*. A strategy with immediate reduction forces the duplication of verb-argument rules at different linguistic levels, namely at the level of main clauses and at the level of completive/gerund clauses. However, the temporal suspension of this operation seems to be the most efficient strategy, since it prevents from duplicating rules. So, we propose to postpone the reduction of the dependent verb until its dependent argument (or arguments) has been identified and removed. The strategy we propose is the following:

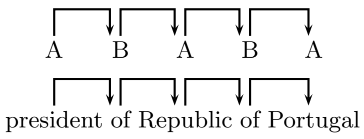
$$\begin{array}{lccc} T_1': & & \texttt{A*} & \texttt{B} \\ T_2: & \texttt{B} \leftarrow & \texttt{B} & \texttt{A} \\ T_1: & \texttt{A} \leftarrow & \texttt{A} & \texttt{B} \end{array} \qquad (5)$$

In (5), the first occurrence of T_1 (noted T_1') is an incomplete rule missing the *Reduce* operation. It is constituted by a pattern with the two core elements, A and B, as well as by the *Right_Arc* operation which identifies a head, A ("saw" and "eating"), and its dependent B ("cat" and "fish"). We use the asterisk to highlight the head. In this partial application of the rule, the reduction of B ("cat" and "fish") is postponed. The removing operation will be performed only after the identification and removal of the A element ("eating") that depends on B ("cat") in rule T_2. So, rule T_1 is only completed after the application of T_2. This final application removes the tokens B ("cat" and "fish"), and so only the main head A ("saw") is kept. Thanks to this technique we are able to separate specific rules for nominal modifiers from those used to deal with verb arguments, and thereby, we keep the simplicity of the rule system. This is a similar strategy to the arc-eager algorithm, but here applied only on embedded expressions.

However, the solution proposed in (5) is applied to those expressions with only one level of recursion. It is easy to find expressions with more recursion levels giving rise to multiple embeddings: e.g. "I saw a dog watching cats eating fish" (Figure 6). This is also the case of expressions that make a chain with several NOUN-ADP-NOUN patterns, for instance "president of the Republic of Portugal", exemplified in (7).



(6)



(7)

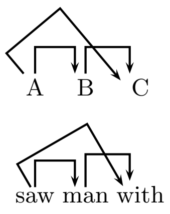
To deal with this type of complex embedded structures, we need to suspend the *Reduce* operation within the two rules required to analyse the expression. Below, in (8), the two rules, T_1' and T_2', are first applied without the *Reduce* operation as many times as necessary to recursively create all the arcs. The *Reduce* operations are only applied once all the arcs have been created (final application of T_1 and T_2); they remove all tokens that are dependent on the main head (the first A to the left):

$$\begin{array}{lrll}
T_1': & & \mathtt{A*} & \mathtt{B} \\
T_2': & & \mathtt{B*} & \mathtt{A} \\
T_2: & \mathtt{B} \leftarrow & \mathtt{B} & \mathtt{A} \\
T_1: & \mathtt{A} \leftarrow & \mathtt{A} & \mathtt{B}
\end{array} \tag{8}$$

The strategy described in (8) allows us to deal with multiple embeddings, such as the analyses shown in examples 6 and 7.

4.6 Ambiguity

The temporal suspension of the *Reduce* operation also permits to introduce ambiguous structures, such as those caused by PP-attachment. In the expression "saw a man with", the preposition "with" can be dependent either on "man" or "saw" (see example (9)).



(9)

Yet, the suspension of *Reduce* is performed here in a slightly different way. As it is shown below in (10), C can be dependent either on A (first line) or on B (second line). This complex rule is constituted by three patterns containing the same PoS tags but with different core elements and, then, with different *Arc* operations. The final reduction is performed when all dependencies involved in the rule are identified. As a result, the final output (the left-side of the rule) is just that element that has been identified as head at least once, but never as dependent in any *Arc* operation. In our example, this element corresponds to token A. Contextual tokens are enclosed in round brackets to be distinguished from the two core elements. As in the examples of the previous section, each head resulting from an *Arc* operation is noted with an asterisk:

$$T_1: \quad \mathtt{A} \leftarrow \begin{bmatrix} \mathtt{A*} & \mathtt{(B)} & \mathtt{C} \\ \mathtt{(A)} & \mathtt{B*} & \mathtt{C} \\ \mathtt{A*} & \mathtt{B} & \mathtt{(C)} \end{bmatrix} \tag{10}$$

Notice that this kind of temporal suspension can lead to breaking up the single-head principle. A dependent token may have several heads, as C in the example above. Ruling the single-head constraint out allows us to yield a richer dependency analysis. For instance, adjective complements of verbs can be seen as dependent on both the verb and a noun: in the expression "such experiences make life worthwhile", the adjective "worthwhile" can be analyzed as being dependent on both the verb "make" and the noun "life". This analysis is only possible if the *Reduce* operation is suspended.

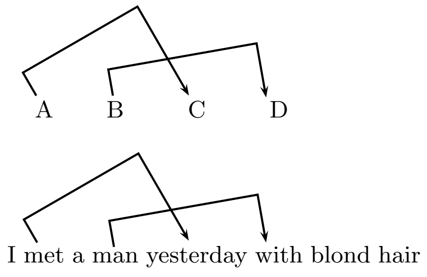
This strategy makes parsing non-deterministic.

4.7 Non-Projective Dependencies

The formalism allows us to generate non-projective trees in a straightforward way. For instance, take the following rules:

$$\begin{array}{llllll} T_1: & \mathtt{A} \leftarrow & \mathtt{A} & \mathtt{(B)} & \mathtt{C} \\ T_2: & \mathtt{B} \leftarrow & \mathtt{B} & \mathtt{(C)} & \mathtt{D} \end{array} \tag{11}$$

These two rules generate non-projective trees (trees with crossing arcs) such as that depicted in (12), which can be the analysis of expressions like: "I met a man yesterday with blond hair", where the adverb "yesterday" depends on the verb "met", the noun "man" depends on the verb "saw", and the preposition "with" depends on the noun "man". In this particular case, A=met, B=man, C=yesterday, and D=with.



(12)

However, the above rules, if they are not lexicalized, may often generate odd parses. For instance, they erroneously create crossing dependencies in the analysis of an expression like "I met a man away from home". In order to avoid odd analyses, lexicalized rules would be necessary, i.e. rules should be enriched with lexical classes. To correctly analyze the last expression, we would require to define rules with the same patterns as those in (11), but being projective and only applied on adverbs that subcategorize prepositions, e.g. "away" subcategorizes "from". Yet, lexicalized rules have low coverage and are language-dependent. As we will see in the next section, our objective will be to define grammars containing high coverage rules shared by several (even if related) languages. In the experiments described later, lexicalized rules with low coverage and those giving rise to non-projective trees will not be considered.

5 System Overview

5.1 The Modules

Our compression parsing strategy has been inserted into a more generic natural language architecture (see Figure 3), which consists of the following modules:

- A set of PoS tagging *adapters* that modify the output of three PoS taggers, namely FreeLing (Padró and Stanilovsky, 2012), Tree-Tagger (Schmid, 1995), and LinguaKit (Garcia and Gamallo, 2015; Gamallo and Garcia, 2017) so as to generate an unified PoS tagged format, where each token is represented as a pair: 'main PoS tag + feature structure'. So, the objective of the adapters is to propose a common notation for all tagsets used by different PoS taggers. The result of this process is the input of compression parsers.

- A set of grammars written with a specific grammar notation.

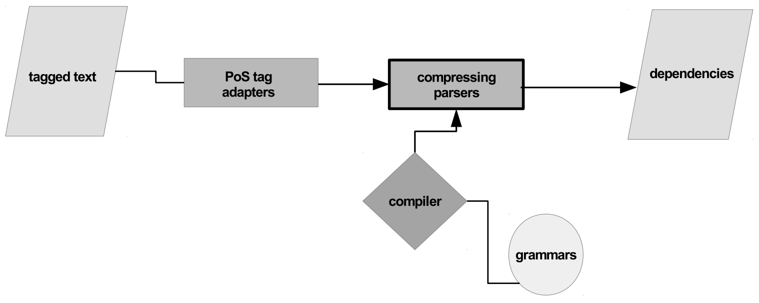


Figure 3: Architecture of DepPattern

- A grammar compiler, written in Ruby, that takes a particular grammar as input and generates a compression parser, written in Perl.

- A set of parsers in Perl, generated by the compiler from various grammars, that take as input the output of the adapters and produce dependency-based analyses.

The whole system, called DepPattern,[6] is released under the GNU General Public License (GPLv3). Six DepPattern parsers were generated. On the one hand, there are language-specific parsers for Spanish, Portuguese, English, French, and Galician. A sound evaluation of the Spanish and Portuguese parsers was reported in Gamallo (2015). On the other hand, we also used DepPattern to implement a specific parser with Universal Dependencies, called *MetaRomance*, that can be applied to any Romance language and whose performance will be analysed and discussed later in Section 6. The parsers are robust and very efficient: they are able to parse about 6,000 tokens per second on a Core i7-3770 processor at 3.4 Ghz. The system can be run on any GNU/Linux distribution.

Among all the modules of DepPattern, special attention will be paid to grammar notation.

5.2 Grammar Notation

To write compression rules, it is necessary to handle regular expressions and programming language code, too unmanageable for linguists. In order to make the task of writing rules easier, we have defined a more intuitive grammar notation. Such a notation is a higher-level language that is encoded in Perl code by means of a grammar compiler written in Ruby.

According to the high-level notation, a dependency-based rule is constituted by three elements:

[6]`https://github.com/gamallo/DepPattern`

- A pattern of tagged tokens, where each token is represented with a basic PoS tags and, if required, a list of 'attribute-values' providing more specific linguistic information. The pattern is constituted by two core elements and, if it is relevant for the guessing, an open number of contextual elements enclosed with square brackets.
- The name of the dependency relation between the two core elements. This name triggers an *Arc* operation.
- A set of optional operations applied on the core elements: Agreement, Add, Corr, Inherit, etc.

By default, each rule activates the *Reduce* operation, but it is possible to postpone it by explicitly introducing some suspension markers. Two suspension markers are the commands *NoUniq* and *NEXT*, which can be added to a rule to suspend the Reduce operation. The former introduces a permanent suspension, while the latter just applies the suspension until the next rule.

Let's take an example of a rule:

```
Modif_R :  NOUN  [ADV<type:Q>]?  ADJ
Agr: number, gender
%
```

The colon separates the pattern of PoS tags (on the right) from the name of the dependency: Modif_R. Symbol '%' means the end of the rule. This rule identifies a dependency, labeled as 'Modif_R', between a noun and an adjective situated on its right, with or without an interposed adverbial quantifier (whose type is 'Q'). 'Agr' stands for the operation of agreement, where 'number, gender' are the names of the attributes whose values must be shared by both the head and the dependent. At the end, *Reduce* removes the dependent token, ADJ, from the input sequence.

The specific names for PoS tags and dependencies must be declared in two configuration files. The linguist can modify the name of any PoS tag and can declare all those dependency labels he/she needs to write the grammar. In addition, lexical classes containing large list of lemmas can be declared in another configuration file. For instance, we can define the extensional class of verbs requiring 'human' subjects by creating a list of such verbs, and use this class as a specific lemma within a rule.

This grammar formalism as well as further linguistic properties were described in Gamallo and González (2011). For more details, there is also available a tutorial and specific documentation.[7]

5.3 The MetaRomance Grammar

We have written a small grammar with about 150 cross-lingual rules that can be applied to all Romance languages using Universal Dependencies (Nivre et al.,

[7] `http://gramatica.usc.es/pln/tools/deppattern/tutorial.html`

2016). The cost of writing the grammar is not high since its size is small and the rules are not language-specific. The strategy we followed to write the MetaRomance grammar is based on two methodological principles:

- Start with high-coverage rules.
- Otherwise, develop rules shared by as many Romance languages as possible.

The objective is to find a trade-off between high performance and low effort, i.e. we look for efficiency. Most rules satisfy these two principles, giving rise to a broad-coverage parser. We have not defined non-projective rules since, in general, they have low coverage and are language dependent.

Following these principles, MetaRomance consists of simple cross-lingual and (almost) delexicalized rules likely to be shared by most Romance languages. Rules are almost delexicalized because they are mainly applied on Universal PoS-tags, only containing few grammar words (some prepositions and conjunctions) together with a small list of verbs. The Universal Dependencies (UD) initiative (Nivre et al., 2016) provides linguistic criteria to create harmonized representations across languages, so it fits perfectly with our objective of defining cross-lingual rules. In fact, the availability of homogeniously annotated treebanks is an interesting test bench for cross-lingual dependency parsing research (McDonald et al., 2011; Mcdonald et al., 2013; Vilares et al., 2016). Most phenomena not covered by the MetaRomance grammar are related with some long distance dependencies, including subordinate clauses in non-canonical positions, or complex issues derived from coordination.

The DepPattern formalism has been adapted so as to let it interpret Universal Dependencies (UDv2). All rules of MetaRomance were written in about 12 hours by an expert linguist who has skills in the DepPattern formalism, but with no prior knowledge in UD. He took into account the syntactic structure of all Romance languages of the UDv2 treebanks except Romanian (Nivre et al., 2017a).

Finally, structural ambiguity was not taken into account. It follows that the derived parser is fully deterministic.

6 Experiments

This section presents several evaluations of MetaRomance using the data provided by the CoNLL 2017 shared task on UD parsing (Zeman et al., 2017). We will show the results of the following experiments:

- Comparison of MetaRomance with other supervised approaches on all the testing treebanks of Romance languages.
- Analysis of the performance of several parsers on different treebanks of the same language.

pos	**token**	**lemma**	**tag**	**morphological feat.**	**head**	**label**
1	They	they	PRON	Case=Nom|Num=Plur	2	nsubj
2	buy	buy	VERB	Num=Plur|Per=3|Tense=Pres	0	root
3	and	and	CONJ	_	2	cc
4	sell	sell	VERB	Num=Plur|Per=3|Tense=Pres	2	conj
5	books	book	NOUN	Num=Plur	2	dobj
6	.	.	PUNCT	_	2	punct

Table 2: CoNLL-U representation of "They buy and sell books."

- Comparison of MetaRomance with a supervised delexicalized parser for Romance languages.

Some of these experiments were previously reported in our contribution to CoNLL-2017 Shared Task Garcia and Gamallo (2017).

6.1 Datasets and Evaluation

Test data are taken from the Universal Dependencies release 2.0 (Nivre et al., 2017b) and are available for 45 languages. It is important to point out that there are more than one treebank for certain languages. Typically, the additional treebanks come from a different source whose texts are from different domains. This is crucial for the second experiment that will be reported later. Treebanks were built under the CoNLL-U data format. This format, which is used for Universal Dependencies treebanks, is deliberately similar to the CoNLL-X format that was used in the CoNLL 2007 Shared Task (Nivre et al., 2007) and has become a *de facto* standard since then. Each word has its own line and there are tab-separated columns for, at least, the following itens: token position, token, lemma, POS tag, morphological features, head position, and dependency label. For instance, Table 2 (partially) encodes the English sentence *They buy and sell books.* in CoNLL-U format.

The evaluation focuses on dependency relations, more precisely on the head node (sixth column in Table 2) and the dependency label (seventh column). The evaluation starts by aligning the columns produced by the system to the gold standard ones. Once the columns are aligned, the evaluation computes LAS as the main scoring metric. Labeled Attachment Score (LAS) is a standard evaluation metric in dependency parsing. It represents the percentage of words that are assigned both the correct syntactic head and the correct dependency label. A dependency is therefore scored as correct only if both nodes (head node and label) of the relation match existing gold-standard nodes. Precision P is the number of correct relations divided by the number of system-produced nodes; recall R is the number of correct relations divided by the number of gold-standard nodes. We then define LAS as the harmonic mean ($F1$) of P and R, where

$$P = \frac{\#correctRelations}{\#systemNodes} \tag{13}$$

$$R = \frac{\#correctRelations}{\#goldNodes} \tag{14}$$

$$LAS = \frac{2PR}{P+R} \tag{15}$$

If the evaluated parser yields a full analysis, then there is the same number of system nodes as gold nodes, and consequently: $P = R = F1$.

Besides LAS, there is another metric, Unlabeled Attachment Score (UAS), which can also be used to evaluate the performance of a depedency parser. This metric is defined in the same way as LAS but only measures the percentage of words that have the correct head, without considering dependency labels.

6.2 Results at CoNLL-2017 shared task

According to the requirements of the shared task, the participants were asked to submit results of all test treebanks. As it was expected, our system obtained low LAS and UAS results from the whole dataset. The macro-average of MetaRomance over all test treebanks was 34.98% LAS, 43.81% UAS. These poor results were expected due to the characteristics of MetaRomance: an almost delexicalized parser which does not require training data, with simple rules only based on the syntactic structure of Romance languages.

Concerning efficiency, MetaRomance needed 29 minutes and 155MB of memory to parse all the testing sets on the TIRA virtual machine provided by the shared task. It was one of the fastest systems in the shared task, namely 5th out of 32 participants.

Table 3 shows the official MetaRomance results on every treebank of a Romance language evaluated in the shared task. Evaluation was performed with the same input and the same evaluation script provided by the organizers at the shared task. On average, our system achieved F1 results of 58.9 (LAS) and 66.1 (UAS). The worst results in Romance languages were obtained in Romanian; this fact was expected because (i) Romanian is linguistically more distant than the other Romance languages (Gamallo et al., 2017), and (ii) we did not implement any dependency rule with this language in mind.

The official MetaRomance results were obtained by using as input the tokenized, lemmatized and PoS-tagged data provided by the UDPipe baseline models (Straka et al., 2016).

Even if the values in Table 3 are not comparable with most supervised systems in the competition (the best one reached 73.30% LAS (Dozat et al., 2017)), our simple parser obtained competitive results in some languages, such as Spanish (*es*), Italian (*it*), and Portuguese (*pt*). It is also interesting to point out the results obtained by MetaRomance on the parallel treebanks (*pud*), which are additional test sets extracted from parallel corpora without corresponding training data. Interestingly, MetaRomance performed better in the *pud* datasets than in the others treebanks of the same languages (with only one exception: UAS results in *pt* and *pt_pud*). By contrast most supervised systems in the shared task decreased their performance in the *pud* datasets in several points. In this

Treebank	**LAS**	**UAS**
ca	57.71	65.57
es	59.80	67.20
es_ancora	60.99	69.63
es_pud	65.49	71.68
fr	54.10	62.20
fr_partut	56.17	63.10
fr_sequoia	55.16	60.76
fr_pud	58.67	65.94
gl	54.87	62.59
gl_treegal	57.20	63.87
it	62.96	70.35
it_pud	65.49	71.82
pt	65.50	71.77
pt_br	56.19	65.81
pt_pud	66.35	71.43
ro	45.04	53.90
average	58.86	66.10

Table 3: Results on the Romance languages test sets (predicted tokens, lemmas, morphological features, and PoS-tags).

respect, MetaRomance leaded some supervised approaches in treebanks such as *pt_pud* or *gl_treegal* (this last one with small training data).

It is worth noticing that we had to modify the original MetaRomance to adapt its results to the evaluation script provided by the organizers. This script requires full parses without more than one root per sentence and without cicles. In other words, the official script requires that precision is equal to recall. As the grammar of MetaRomance is not complete, giving rise to partial parses, we implemented a post-editor script linking all tokens without head information to the corresponding sentence root. The result of this post-editor script is a full parse.

6.3 Cross-treebank performance

Our second experiment compares the cross-treebank performance of supervised models (i.e., parsing different treebanks of the same language with the same model). To carry out this experiment we trained a UDPipe model (Straka et al., 2016) in each training dataset of Spanish, Galician, and Portuguese. These models were trained using the default parameters of UDPipe 1.1, but removing the lemmas and the morphological features of the treebanks, with the aim of building parsers with more robust performance among the different test sets.[8]

[8]It is important to note that using the models provided by the shared task organization to parse test data from different treebanks than the one used for training (e.g, train on *fr-sequoia*

Target	*Source*			
Spanish	*es*		*es_ancora*	
	LAS	UAS	LAS	UAS
es	*76.85*	*81.19*	64.25	71.95
es_ancora	67.25	76.43	*79.36*	*83.42*
es_pud	74.88	82.26	67.67	76.77
Galician	*gl*		*gl_treegal*	
	LAS	UAS	LAS	UAS
gl	*73.71*	*77.17*	58.03	68.47
gl_treegal	50.98	63.37	*65.24*	*70.81*
Portuguese	*pt*		*pt_br*	
	LAS	UAS	LAS	UAS
pt	*78.74*	*82.43*	68.00	77.92
pt_br	66.85	76.19	*82.10*	*84.83*
pt_pud	71.59	77.58	67.75	77.87

Table 4: Results of UDPipe models trained in the source treebanks (columns) on the target test sets (rows).

Table 4 includes the LAS and UAS values of each model (in the columns) on the target treebanks (on each row). These numbers indicate that the results of supervised parsers show noticeably differences when parsing a different treebank to the one used for training, even if both corpora belong to the same language. These differences are much higher than those of MetaRomance, exceeding 22% in *gl* parsing *gl_treegal*, more than 15% in the analysis of *es* by *es_ancora*, or more than 14% in *pt_br* parsing *pt*.

Furthermore, these results (both the UDPipe and the MetaRomance ones) suggest that careful analyses of the different treebanks are required, aimed at knowing whether these large variations are due to different domains, annotation issues, or linguistic and syntactic differences.

6.4 Comparison with a cross-lingual delexicalized parser

In the next experiment we evaluate the original version of MetaRomance, which yields partial parses, and compare its performance with a delexicalized parser trained with a combined corpus which includes sentences from every Romance treebank.

In this experiment, the evaluation protocol is slightly different from that required by the official evaluation script provided by the organizers of CoNLL 2017 shared task. Our objective is to overcome some restrictions of the official script which could be unfair for unsupervised systems such as MetaRomance. Firstly, in the current evaluation, we do not consider punctuation marks since their dependencies have not clear linguistic criteria and, therefore, they are too dependent on the ad-hoc decisions taken by the treebank annotators. Secondly,

and test on fr) produce results with drops of more than 26% LAS.

Treebank	LAS	UAS
ca	64.86	72.42
es	68.43	76,87
es_ancora	68.48	75.10
es_pud	67.92	72.17
fr	68.11	74.51
fr_partut	69.96	77.38
fr_sequoia	71.26	75.35
fr_pud	65.03	68.59
gl	64.11	71.06
gl_treegal	72.18	77.55
it	72.70	77.22
it_pud	66.19	70.08
pt	74.93	79.49
pt_br	66.21	74.55
pt_pud	68.10	70.83
ro	58.45	65.71
average	67.93	73.56

Table 5: Results obtained by the original version of MetaRomance (partial parsing) on the Romance languages test sets (tokens, lemmas, morphological features, and PoS-tags from the gold standard), without taking into account punctuation.

the parser takes now clean texts as input instead of noisy texts which were PoS tagged by UDPipe processing modules. Clean input texts are constituted by tokens, lemmas, morphological features and PoS tags taken from the gold standard dataset. Finally, MetaRomance is evaluated without considering the post-editor module that assigns the root index to any token without head value.

In sum, MetaRomance is evaluated as a partial parser without considering elements that are not well suited for unsupervised systems. In the case of partial analysis, precision is different from recall, but LAS and UAS still stand for their corresponding F1 score. Table 5 shows the LAS and UAS scores of MetaRomance. These scores are now over 10% higher than in the official evaluation, being a little closer to state-of-the-art supervised strategies.

We compared the performance of MetaRomance with a delexicalized parser trained with a combined corpus which includes sentences from every Romance treebank. This is a competitive supervised baseline in cross-lingual transfer parsing work, which gives us an indication of how our system compares to standard cross-lingual parsers.

We trained 50 UDPipe models by randomly selecting from 1 to 50 sentences of each Romance treebank in the training data. Then, we obtained the average results on all the Romance test treebanks, and plotted them together with the MetaRomance performance in Figure 4. These scores were obtained with the (non-official) evaluation script defined above, without puntuation marks. The

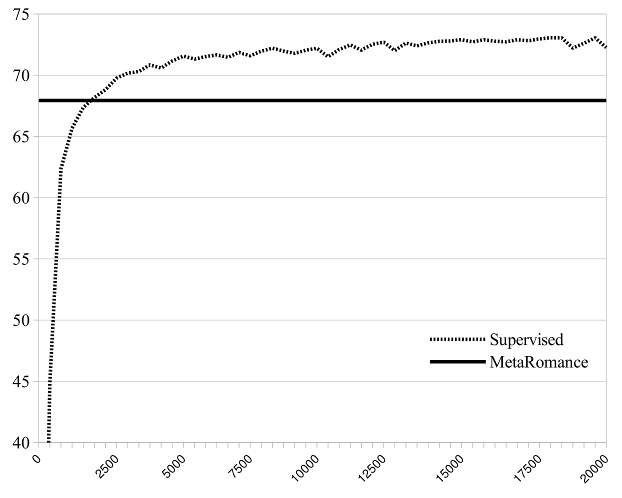


Figure 4: LAS values of MetaRomance *versus* the learning curve (0–20,000 tokens) of a delexicalized UDPipe model trained with random sentences from all Romance treebanks. Results are average F1 values of all the testing Romance treebanks.

input in also the same, consisting of lemmas and PoS tags from the gold standard test dataset.

This figure shows that MetaRomance obtains similar results ($\approx$ 68% LAS) to those achieved with about 2,000 tokens of all the Romance treebanks. The learning curve also suggest that it is difficult for cross-lingual models with no lexical features to keep increasing their cross-lingual performance on Romance languages. More precisely, UDPipe achieves 72% with about 5,000 tokens, but it cannot surpass 73% even with a training corpus of 20,000 tokens.

6.5 Discussion

Our system is a dependency parser that requires no training, and is mainly built using generic rules defined on the basis of UD tags and constraints. The experiments performed in this paper provided some interesting results that claim for further research in cross-lingual parsing.

On the one hand, there are noticeable differences when parsing different treebanks of the same language, both using a rule-based system and harmonized supervised models. Results showed that the performance of MetaRomance changes less across different treebanks than supervised methods, which turn out to be very dependent on the text domain used to train the system. In this respect, it could be interesting to analyze the source of these variations, and MetaRomance could be useful for this purpose because it uses linguistically transparent rules based on PoS-tags.

On the other hand, the learning curve of a cross-lingual delexicalized model reinforces the idea that lexical features are required to obtain high-quality parsing results. In this respect, further experiments could compare this learning curve to lexicalized cross-lingual models, which seem to obtain good results in languages from the same linguistic family. Concerning MetaRomance, the addition of new rules (both lexicalized and without lexical information) could allow the parser to better analyze different languages.

Finally, and even if this is not a fair comparison, it is worth noting that MetaRomance obtained higher results in Romance languages than those achieved by UDP and reported in Martínez Alonso et al. (2017). UDP is a training-free parser based on PageRank and a small set of head attachment rules, being more generic than MetaRomance (it can be applied to any language with more homogeneous results than our system). The differences on Romance languages vary between few decimals to more than 6% UAS, but the experiments were performed using different versions of the UD treebanks.

The analysis of results sheds light on several properties of the two evaluated systems.

7 Conclusions

The work described in the article can be seen as a contribution to improve old parsing strategies introduced at the end of the twentieth century, when most efficient techniques were based on rules/FST and constructive approaches. In particular, we described a grammar-driven parser based on FST, called compression parsing, which takes into account some elements from deterministic and incremental dependency parsing, namely *Arc* and *Reduce* transitions. This compression method, implemented in DepPattern, was compared to a data-driven, transition-based system: MaltParser. The experiments showed that the data-driven system has a clearly better performance when applied on test sentences belonging to the same domain (and same linguistic criteria) as the training set. However, the grammar-based parser seems to be more stable across domains, genres, and languages.

The cost and effort of developing compression parsers for several languages is not very high, since they can achieve reasonable performance using just very simple, cross-lingual, and general-purpose grammars. In this article, we have also introduced a simple methodology to write cross-lingual and general-purpose grammars. This methodology was applied to build MetaRomance, a generic grammar for most Romance languages. We claim that the human effort of building a generic grammar for Romance languages is clearly lower than that used to build language-specific treebanks for all these languages. This is a relevant feature for dealing with under-resourced languages.

In future work, it would be interesting to compare a variety of grammar-driven systems by measuring, not only their performance, but also the complexity of the underlying grammar: number of rules, size (in bytes) of the source files, etc. It should also be important to quantify and compare the cost and effort of both writing grammars and building treebanks. We also intend to write cross-lingual grammars for other language families (e.g. Germanic and Slavic ones). Moreover, to complement quantitative evaluation, it will be necessary to define objective protocols to compare parsers on the basis on qualitative evaluation (Lloberes et al., 2014).

Finally, we claim that FST-based parsing techniques simulate how we solve problems quickly, by taking easy decisions first which, in turn, makes it easier to solve further complex tasks. However, these parsing techniques are far from simulating two other interesting cognitive operations: first, how grammars are learnt and, second, how sentences are processed. Data-driven approaches can be seen as a good approximation to the way humans learn grammars, while incremental left-to-right parsing can be seen as a simulation of how humans process and understand input sentences. Here, a question arises: would it be possible to define a method taking advantage of all those parsing strategies? In other words, could it be possible to model a strategy that learns grammar rules from data, orders them as cascades of progressively more complex transducers, and applies them to sentences in an incremental way? A method provided with these three 'human properties' would be closer to the canonical systems in Artificial Intelligence, since the main objective would be not only to produce parse trees, but also to simulate how humans understand sentences.

Acknowledgments

This work has received financial support from a 2016 BBVA Foundation Grant for Researchers and Cultural Creators, TelePares (MINECO, ref:FFI2014-51978-C2-1-R), the Consellería de Cultura, Educación e Ordenación Universitaria (accreditation 2016-2019, ED431G/08) and the European Regional Development Fund (ERDF). Marcos Garcia has been funded by two Juan de la Cierva grants (FJCI-2014-22853 and IJCI-2016-29598) and the MINECO project with reference FFI2016-78299-P.

References

Abdi, Asad, Norisma Idris, Rasim M. Alguliev, and Ramiz M. Aliguliyev. 2015. Automatic summarization assessment through a combination of semantic and syntactic information for intelligent educational systems. *Information Processing & Management* 51(4):340–358.

Abney, Steven. 1996. Partial parsing via finite-state cascades. *Natural Language Engineering* 2(4):337–344.

Agić, Željko, Anders Johannsen, Barbara Plank, Héctor Martínez Alonso, Natalie Schluter, and Anders Søgaard. 2016. Multilingual projection for parsing truly low-resource languages. *Transactions of the Association for Computational Linguistics* 4:301–312.

Ait-Mokhtar, Salah and Jean-Pierre Chanod. 1997. Incremental Finite-State Parsing. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*. Washington, DC, USA.

Ait-Mokhtar, Salah, Jean-Pierre Chanod, and Claude Roux. 2002. Robustness beyond Shallowness: Incremental Deep Parsing. *Natural Language Engineering* 8(2/3):121–144.

Ammar, Waleed, George Mulcaire, Miguel Ballesteros, Cris Dyer, and Noah A. Smith. 2016. Many Languages, One Parser. *Transactions of the Association for Computational Linguistics* 4:431–444.

Arnola, Harri. 1998. On parsing binary dependency structures deterministically in linear time. In *Workshop on Processing of Dependency-Based Grammars (ACL-COLING)*, pages 68–77.

Carreras, Xavier. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961. Prague: Association for Computational Linguistics.

Ciravegna, Fabio and Alberto Lavelli. 2002. Full parsing approximation for information extraction via finite-state cascades. *Natural Language Engineering* 8(2/3):145–165.

Covington, Michael A. 2001. A fundamental algortithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102.

Daumé III, Hal. 2006. Domain adaptation for statistical classifiers. *Artificial Intelliegence Research* 26:101–126.

Daumé III, Hal. 2007. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263. Association for Computational Linguistics.

Derczynskia, Leon, Diana Maynard, Giuseppe Rizzo, Marieke van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. 2015. Analysis of named entity recognition and linking for tweets. *Information Processing & Management* 51(2):32–49.

Dozat, Timothy, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

Eisner, Jason and Noah A. Smith. 2010. Favor short dependencies: Parsing with soft and hard constraints on dependency length. In H. Bunt, P. Merlo, and J. Nivre, eds., *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, chap. 8, pages 121–150. Springer.

Gamallo, Pablo. 2015. Dependency parsing with compression rules. In *Proceedings of the 14th International Workshop on Parsing Technology (IWPT 2015)*, pages 107–117. Bilbao, Spain: Association for Computational Linguistics.

Gamallo, Pablo and Marcos Garcia. 2017. Linguakit: uma ferramenta multilingue para a análise linguística e a extração de informação. *Linguamática* 9(1).

Gamallo, Pablo, Marcos Garcia, and Santiago Fernández-Lanza. 2012. Dependency-based open information extraction. In *ROBUS-UNSUP 2012: Joint Workshop on Unsupervised and Semi-Supervised Learning in NLP at the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2012)*, pages 10–18. Avignon, France.

Gamallo, Pablo and Isaac González. 2011. A grammatical formalism based on patterns of part-of-speech tags. *International Journal of Corpus Linguistics* 16(1):45–71.

Gamallo, Pablo and José Ramom Pichel. 2008. Learning Spanish-Galician Translation Equivalents Using a Comparable Corpus and a Bilingual Dictionary. In *International Conference on Intelligent Text Processing and Computational Linguistics*, vol. 4919 of *Lecture Notes in Computer Science*, pages 423–433. Springer.

Gamallo, Pablo, José Ramom Pichel, and Iñaki Alegria. 2017. From language identification to language distance. *Physica A* 484:162–172.

Ganchev, Kuzman, Jennifer Gillenwater, and Ben Taskar. 2009. Dependency grammar induction via bitext projection constraints. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, vol. 1, pages 369–377. Association for Computational Linguistics.

Garcia, Marcos and Pablo Gamallo. 2010. Using morphosyntactic post-processing to improve PoS-tagging accuracy. In *Proceedings of the International Conference on Computational Processing of the Portuguese Language (PROPOR'2010). Extended Activities*. Porto Alegre.

Garcia, Marcos and Pablo Gamallo. 2011. Dependency-based text compression for semantic relation extraction. In *Workshop on Information Extraction and Knowledge Acquisition (IEKA 2011) at 8th International Conference on Recent Advances in Natural Language Processing (RANLP 2011)*, pages 21–28.

Garcia, Marcos and Pablo Gamallo. 2015. Yet another suite of multilingual NLP tools. In *Languages, Applications and Technologies*, vol. 563 of *Communications in Computer and Information Science*, pages 65–75. Switzerland: Springer. Revised Selected Papers of the Symposium on Languages, Applications and Technologies (SLATE 2015).

Garcia, Marcos and Pablo Gamallo. 2017. “A rule-based system for cross-lingual parsing of Romance languages with Universal Dependencies. In *Conference on Computational Natural Language Learning (CoNLL-2017), held at ACL 2017*. Vancouver, Canada.

Gildea, Daniel. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP 2001)*. Pittsburgh, PA: Association for Computational Linguistics.

Goldberg, Yoav and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. Stroudsburg, PA, USA.

Gómez-Rodríguez, Carlos and Daniel Fernández-González. 2012. Dependency parsing with undirected graphs. In *13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 66–76. Avignon, France.

Gómez-Rodríguez, Carlos, Daniel Fernández-González, and Víctor M. Darriba. 2015. Undirected Dependency Parsing. *Computational Intelligence* 31(2):348–384.

Grefenstette, Gregory. 1996. Light parsing as finite-state filtering. In *Workshop on Extended Finite State Models of Language ECAI'96*. Budapest, Hungary.

Grimley-Evans, Edmund. 1997. Approximating context-free grammars with a finite-state calculus. In *Proceedings of ACL-EACL'97*, pages 452–459.

Holan, Tomáš and Zdenek Zabokrtsky. 2006. Combining Czech Dependency Parsers. In *Proceedings of the 9th International Conference on Text, Speech and Dialogue*. Brno, Czech Republic.

Hwa, Rebecca, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. 2005. Bootstrapping parsers via syntactic projection across parallel texts. *Natural Language Engineering* 11(03):311–325.

Johnson, Mark. 1998. Finite state approximation of constraint-based grammars using left-corner grammar transforms. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*, pages 619–623.

Joshi, Aravind. 1996. A parser from antiquity: An early application of finite state transducers to natural language parsing. In *ECAI-96 Workshop on Extended Finite State Models of Languages*. Budapest, Hungary.

Kokkinakis, Dimitrios and Sofie Johansson Kokkinakis. 1999. A Cascaded Finite-State Parser for Syntactic Analysis of Swedish. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. Bergen, Norway.

Koskenniemi, Kimmo, Pasi Tapanainen, and Atro Voutilainen. 1992. Compiling and using finite-state syntactic rules. In *International Conference on Computational Linguistics (COLING)*, pages 156–162.

Laporte, Eric. 1996. Context-free parsing with finite-state transducers. In *South American Workshop on String Processing*, pages 171–182. McGill-Queen's University Press and Carleton University Press.

Lee, Changki, Gary Geunbae Lee, and Myunggil Jang. 2007. Dependency structure language model for topic detection and tracking. *Information Processing & Management* 43(5):1249–1259.

Lloberes, Marina, Irene Castellón, Lluís Padró, and Edgar González. 2014. Partes. test suite for parsing evaluation. *Procesamiento del Lenguaje Natural* 53:87–94.

Martínez Alonso, Héctor, Zeljko Agic, Barbara Plank, and Anders Søgaard. 2017. Parsing universal dependencies without training. In *15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017)*, pages 229–239. Valencia, Spain.

Martins, André F. T., Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2010. Turboparsers: Dependency parsing by approximate variational inference. In *Empirical Methods in Natural Language Processing (EMNLP'10)*. Boston, USA.

Mcdonald, Ryan, Joakim Nivre, Yvonne Quirmbach-brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu, and Castelló Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, pages 92–97. Sofia: Association for Computational Linguistics.

McDonald, Ryan and Fernando Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In A. for Computational Linguistics, ed., *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, pages 81–88. Trento.

McDonald, Ryan, Slav Petrov, and Keith Hall. 2011. Multi-source Transfer of Delexicalized Dependency Parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*, pages 62–72. Edinburgh, United Kingdom: Association for Computational Linguistics.

Nederhof, Mark-Jan. 2000. Practical experiments with regular approximation of context-free languages. *Computational Lingustics* 13(2):115–135.

Nivre, Joakim. 2003. An efficient algorithm for projective dependency parsing. In *8th Intenational Workshop on Parsing Technologies*, pages 149–160.

Nivre, Joakim. 2004. Incrementality in deterministic dependency parsing. In *ACL Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Barcelona: Association for Computational Linguistics.

Nivre, Joakim. 2006. *Inductive Dependency Parsing*. Netherlands: Springer.

Nivre, Joakim et al. 2017a. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, `http://hdl.handle.net/11234/1-1983`.

Nivre, Joakim, Željko Agić, Lars Ahrenberg, et al. 2017b. Universal dependencies 2.0 – CoNLL 2017 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University.

Nivre, Joakim, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666. Portorož, Slovenia: European Language Resources Association.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL-2007 shared task on dependency parsing. In *Proceedings of the Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932. Prague, Czech Republic.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2004. Proceedings of the eighth conference on computational natural language learning (conll-2004) at hlt-naacl 2004. pages 49–56.

Oflazer, Kemal. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics* 29(4):515–544.

Padró, Lluís. and Evgeny Stanilovsky. 2012. FreeLing 3.0: Towards Wider Multilinguality. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2012)*, pages 2473–2479. Istanbul, Turkey: European Language and Resources Association.

Pereira, Fernando and Rebecca N. Wright. 1997. Finite state approximation of phrase structure grammars. In E. Roche and Y. Schabes, eds., *Finite State Language Processing*. MIT Press, Cambridge, MA.

Rimell, Laura, Stephen Clark, and Mark Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Conference on Empirical Methods in Natural Language Processing*, pages 813–821. Singapore.

Roche, Emmanuel. 1997. Parsing with finite-state transducers. pages 241–281. MIT Press.

Roche, Emmanuel. 1999. Finite state transducers: parsing free and frozen sentences. In A. Kornai, ed., *Extended Finite State Models of Language*, pages 108–120. Cambridge University Press.

Rosa, Rudolf, Daniel Zeman, David Mareček, and Zdeněk Žabokrtský. 2017. Slavic forest, norwegian wood. In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial4)*, pages 210–219. Stroudsburg, PA, USA: Association for Computational Linguistics.

Saif, Hassan, Yulan He, Miriam Fernandez, and Harith Alani. 2015. Contextual semantics for sentiment analysis of twitter. *Information Processing & Management* 52(1):5–19.

Schmid, Helmut. 1995. Improvements in Part-of-Speech Tagging with an Application to German. In *ACL SIGDAT Workshop*. Dublin, Ireland.

Sekine, Satoshi. 2000. Japanese dependency analysis using a deterministic finite state transducer. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pages 761–767. Association for Computational Linguistics.

Severyn, Aliaksei, Alessandro Moschitti, Olga Uryupina, Barbara Plank, and Katja Filippova. 2015. Multi-lingual opinion mining on youtube. *Information Processing & Management* 52(1):46–60.

Sokolova, Marina and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management* 45(4):427–437.

Straka, Milan, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. Portorož, Slovenia: European Language Resources Association.

Täckström, Oscar, Ryan McDonald, and Jakob Uszkoreit. 2012. Cross-lingual word clusters for direct transfer of linguistic structure. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human language technologies (NAACL - HLT 2012)*, pages 477–487. Association for Computational Linguistics.

Tiedemann, Jörg. 2017. Cross-lingual dependency parsing for closely related languages. In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2017)*, pages 131–136. Valencia: Association for Computational Linguistics.

Tiedemann, Jörg and Željko Agić. 2016. Synthetic Treebanking for Cross-Lingual Dependency Parsing. *Journal of Artificial Intelligence Research (JAIR)* 55:209–248.

Tratz, Stephen and Eduard Hovy. 2011. A fast, accurate, non-projective, semantically-enriched parser. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1257–1268. Association for Computational Linguistics.

Tseng, Yuen-Hsien, Chi-Jen Lin, and Yu-I Lin. 2007. Text mining techniques for patent analysis. *Information Processing & Management* 43(5):1216–1247.

Uysal, Alper Kursat and Serkan Gunal. 2014. The impact of preprocessing on text classification. *Information Processing & Management* 50(1):104–112.

Versley, Yannick. 2014. Experiments with easy-first nonprojective constituent parsing. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 39–53. Dublin, Ireland.

Vilares, David, Miguel A. Alonso, and Carlos Gómez-Rodríguez. 2015. On the usefulness of lexical and syntactic processing in polarity classification of twitter messages. *Journal of the American Society for Information Science* 66(9):1799–1816.

Vilares, David, Miguel A. Alonso, and Carlos Gómez-Rodríguez. 2016. One model, two languages: training bilingual parsers with harmonized treebanks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, vol. 2, pages 425–431. Berlin, Germany.

Vilares, Jesús, Miguel A. Alonso, and Manuel Vilares. 2014. Extraction of complex index terms in non-English IR: A shallow parsing based approach. *Information Processing & Management* 44(4):1517–1537.

Yamada, Hiroyasu and Yuji Matsumoto. 2003. Statistically dependency analysis with support vector machines. In *Proceedings of 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

Yli-Jyrä, Anssi. 2005. Linguistic grammars with very low complexity. In A. Arppe, K. Lindén, J. Piitulainen, M. Suominen, M. Vainio, H. Westerlund, and A. Yli-Jyrä, eds., *Inquiries into Words, Constraints and Contexts: Festschrift in the Honor of Kimmo Koskenniemi*, pages 172–183. CSLI Studies in Computational Linguistics ONLINE, CSLI Publications.

Zampieri, Marcos, Shervin Malmasi, Nikola Ljubešić, Preslav Nakov, Ahmed Ali, Jörg Tiedemann, Yves Scherrer, and Noëmi Aepli. 2017. Findings of the VarDial Evaluation Campaign 2017. In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*. Valencia.

Zeman, Daniel, Filip Ginter, Jan Hajič, Joakim Nivre, Martin Popel, Milan Straka, and et al. 2017. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–20. Association for Computational Linguistics.

Zeman, Daniel and Philip Resnik. 2008. Cross-Language Parser Adaptation between Related Languages. In *Proceedings of the Workshop on NLP for Less Privileged Language at the 3rd International Joint Conference on Natural Language Processing (IJCNLP 2008)*, pages 35–42. Hyderabad: Asian Federation of Natural Language Processing.

Zhang, Min, GuoDong Zhou, and Aiti Aw. 2008. Exploring syntactic structured features over parse trees for relation extraction using kernel methods. *Information Processing & Management* 44(2):687–701.

Zhang, Yue and Stephen Clark. 2009. Transition-based Parsing of the Chinese Treebank Using a Global Discriminative Model. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 162–171.

Zhou, GuoDong, Junhui Li, Jianxi Fan, and Qiaoming Zhu. 2011. Tree kernel-based semantic role labeling with enriched parse tree structure. *Information Processing & Management* 47(3):349–362.