A Three-Step Transition-Based System for Non-Projective Dependency Parsing

Ophélie Lacroix and **Denis Béchet**
LINA - University of Nantes
2 Rue de la Houssinière
44322 Nantes Cedex 3
{ophelie.lacroix,denis.bechet}@univ-nantes.fr

Abstract

This paper presents a non-projective dependency parsing system that is transition-based and operates in three steps. The three steps include one classical method for projective dependency parsing and two inverse methods predicting separately the right and left non-projective dependencies. Splitting the parsing allows to increase the scores on both projective and non-projective dependencies compared to state-of-the-art non-projective dependency parsing. Moreover, each step is performed in linear time.

1 Introduction

Dependency parsing is a particularly studied task and could be a significant step in various natural language processes. That is why dependency parsers should tend to get speed and precision. In recent years, various methods for dependency parsing were proposed (Kübler et al., 2009). Among these methods, transition-based systems are particularly suitable.

The first methods developed for transition-based parsers proposed to produce projective dependency structures (including no crossing dependencies). Then, extended methods were developed to handle the non-projective cases. The non-projective dependency structures admit non-projective dependencies (a dependency is non-projective if at least one word located between the head and the dependent of the dependency does not depend directly or inderectly on the head, see Figure 1 for example). Handling the non-projective cases has been the foundation of the first work concerning the dependency representations (Tesnière, 1959; Melcuk, 1988). Moreover, it is important to successfully parse the non-projective sentences which can be very helpful in processes such as question-answering.

The transition-based parsers achieve interesting overall results for both projective and non-projective analyses. But, in practice, the non-projective methods achieve far lower and variable scores on non-projective dependencies than on projective dependencies. Finding these dependencies is more difficult because the non-projective dependencies are often distant ones. It is then essential to achieve descent scores on non-projective dependencies as well as on projective ones because some languages contain a high rate of non-projective dependencies.

Here we propose to predict separately the projective dependencies from the non-projective ones. Using a mixed dependency representation including both projective and non-projective dependency annotations in one representation, we aim at predicting the projective dependencies in a first step. Taking advantage of the good results of projective dependency parsing, we aim at predicting the non-projective dependencies in a second step.

The formal dependency representation on which we base our work results from the formalism of categorial dependency grammars (CDG) (Dekhtyar and Dikovsky, 2008). It allows to handle the discontinuities of the natural languages. The dependency representation induced is mixed: it associates projective and non-projective dependencies to represent complementary syntactic information in one dependency

This work is licensed under a Creative Commons Attribution 4.0 International Licence. Page numbers and proceedings footer are added by the organisers. Licence details: http://creativecommons.org/licenses/by/4.0/

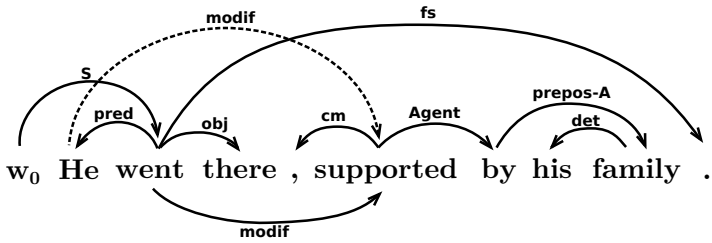

Figure 1: Dependency structure of the sentence "He went there, supported by his family." Anchors are shown below the sentence. Non-projective dependencies appear using a dash line. The other dependencies are plain projective dependencies.

structure. In this representation, each non-projective dependency is paired with a projective one called an anchor. From any dependency structure a projective tree[1] can be extracted.

Our approach is to predict the projective dependency trees first, using a standard and efficient method for projective dependency parsing. In a second step, we use the information (the projective/anchor labelled dependencies) given by the projective parsing to predict the non-projective dependencies. This second step is split into two inverse methods which predict independently the right and left non-projective dependencies. The advantage of the splitting is to perform the parsing in linear time and achieve better scores on non-projective dependencies.

Finally, in order to evaluate the efficiency of our method, we apply it on data annotated according to the formalism of the categorial dependency grammar. The data consists on a treebank containing both projective and non-projective trees associated with sentences of French.

2 Related Work

Our approach is similar to a post-processing method for retrieving the non-projective dependencies. In a way, our work is then analogous to the work of Hall and Novák (2005) who apply a post-processing method after converting constituency trees into dependency ones since the conversion can not automatically recover the non-projective relations.

Moreover, taking advantage of the efficiency of projective dependency methods to predict the non-projective dependencies is a technique used by Nivre and Nilsson (2005) in their pseudo-projective method. They projectivize the dependency trees before parsing in order to apply a projective method first and apply an inverse transformation to retrieve the non-projective dependencies. For our method, we do not need to projectivize the trees since the dependency representation we use includes both projective and non-projective annotations in one representation. But we can employ the projectivization method to build such data adding the generated projective dependencies to the non-projective structure as if they were artificial anchors. Consequently, our approach can be applied on treebank containing standard non-projective trees.

The advantage of our method is that the information that is useful for retrieving the non-projective dependencies is not predicted during the projective parsing which makes the projective and non-projective steps completely independent from each other. Moreover, the non-projective steps are data-driven and remain linear.

3 Representation and Formalism

Our work is based on dependency structures combining projective and non-projective annotations in one representation. In such a representation the projective dependencies bring both local and syntactic information while the non-projective dependencies bring only syntactic information (i.e. the relation shared by the dependents). Thus, each non-projective dependency is paired with a projective relation (called anchor) determining the position of the dependent in the sentence. Figure 1 presents a non-projective dependency structure of a sentence which illustrates the use of a projective relation (anchor)

[1] Composed of projective dependencies and anchors of non-projective dependencies, see Section 3.

Draft

and a non-projective dependency to represent a discontinuous relation: "supported" is a modifier for the pronoun "he".

The dependency representation is induced by a particular formalism: the class of the categorial dependency grammars (CDG). The categories of the grammars correspond to the dependency labels. The rules $\mathbf{L}^1$, $\mathbf{I}^1$ and $\mathbf{\Omega}^1$, presented in Table 1, are the classical left elimination rules of categorial grammars. Only the left rules are shown but there are symmetrical right rules. These rules allow to define the projective dependencies and anchors. Moreover, CDGs are classical categorial grammars in which the notion of polarized valencies was added. Each of the three first rules includes the concatenation of potentials (such as P, P_1, P_2) which are lists of polarized valencies. The polarized valencies are label names associated with a polarity (south-west $\swarrow$, north-west $\nwarrow$, north-east $\nearrow$ and south-east $\searrow$). They represent the ends of the non-projective dependencies. The south polarities indicate an incoming non-projective dependency and the north valencies indicate an outgoing non-projective dependency. The rule $\mathbf{D}^1$ allows the elimination of dual pairs of polarized valencies, following the **FA** principle.

First Available (FA) principle: the closest dual polarized valencies with the same name are paired.

Thus, the elimination of the dual pairs $(\swarrow C)$ $(\nwarrow C)$ and $(\nearrow C)$ $(\searrow C)$ defines respectively left and right non-projective dependencies labelled by C.

$\mathbf{L}^1$	$C^{P_1}\,[C\backslash\beta]^{P_2} \vdash [\beta]^{P_1P_2}$
$\mathbf{I}^1$	$C^{P_1}\,[C^*\backslash\beta]^{P_2} \vdash [C^*\backslash\beta]^{P_1P_2}$
$\mathbf{\Omega^1}$	$[C^*\backslash\beta]^{P} \vdash [\beta]^{P}$
$\mathbf{D}^1$	$\alpha^{P_1(\swarrow C)P(\nwarrow C)P_2} \vdash \alpha^{P_1PP_2}$, if $(\swarrow C)(\nwarrow C)$ satisfies the **FA** principle

Table 1: (Left) Rules of the categorial dependency grammars.

4 Method

We conduct a three-step transition-based parsing. We choose the arc-eager method of Nivre (2008) to perform the first step. Note that any projective method for dependency parsing would also be appropriate to perform this step. The second and third steps are methods which go through the sentence (respectively from left to right and from right to left) in order to find the non-projective dependencies.

4.1 Projective Dependency Parsing

The arc-eager method is an efficient transition-based method for projective dependency parsing. A transition system is composed of a set of configurations (states), a set of transitions (operations on the configurations), an initial configuration and a set of terminal configurations. The transition-based parsing consists in applying a sequence of transitions to configurations in order to build a dependency structure. For the arc-eager method, a configuration is a triplet $\langle \sigma, \beta, A \rangle$ where:

- σ is a stack of partially treated words;
- β is a buffer of non-treated words;
- A is a set of dependencies (the partially built dependency structure).

The dependencies are described by triplets such as (k, l, i) where k is the position of the head, l is the label of the dependency and i is the position of the dependent. The set of transitions includes three transitions which are evolutions of the standard transitions of the system of Yamada and Matsumoto (2003) plus the Reduce transition which allows to delete the first word of the stack when this one shares no dependency with the first word of the buffer. The standard Right-Arc and Left-Arc are renamed respectively as Local-Right and Local-Left since these transitions only add local dependencies (whitout distinction between projective ones and anchors). The Shift transition pops the first word from the buffer

Draft

Transition	Application	Condition
Local-Left(l)	$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_j \mid \beta, A \cup \{(j, l, i)\})$	$i \neq 0 \wedge \neg\exists k \exists l'(k, l', i) \in A$
Local-Right(l)	$(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma \mid w_i w_j, \beta, A \cup \{(i, l, j)\})$	$\neg\exists k \exists l'(k, l', j) \in A$
Reduce	$(\sigma \mid w_i, \beta, A) \Rightarrow (\sigma, \beta, A)$	$\exists k \exists l(k, l, i) \in A$
Shift	$(\sigma, w_i \mid \beta, A) \Rightarrow (\sigma \mid w_i, \beta, A)$	

Table 2: Transitions of the arc-eager method.

and pushes it into the stack. The Reduce transition pops the first word from the stack. The effects of the transitions on configurations are detailed in Table 2.

For a given sentence $W = w_1...w_n$, the initial configuration of the transition-based system is defined as follows: $([w_0], [w_1, ..., w_n], \emptyset)$ where w_0 is the root of the structure. And any terminal configuration is of the form: $([w_0], [], A')$ where A' contains the fully projective dependency/anchor structure for the sentence W[2].

This step should produce the projective dependency structure of Figure 2 for the sentence "Il y est allé, soutenu par sa famille" (french equivalent of the sentence seen in Figure 1).

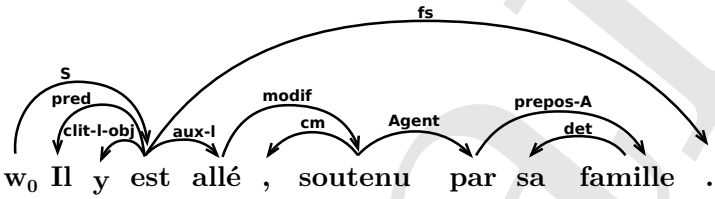

Figure 2: Projective dependency structure of the sentence "Il y est allé, soutenu par sa famille".

4.2 Adding Non-Projective Dependencies

With the aim of retrieving non-projective dependencies we propose two inverse methods also inspired by transition-based systems. For these methods, the configuration is a quadruplet $\langle \sigma, \beta, \theta, A \rangle$ where σ, β and A are the same stack, buffer and set of arcs as those defined for projective parsing in the previous subsection and θ is a list of polarized valencies. The valencies have the same role here as in the formalism of the categorial dependency grammars (detailed in section 3). They define the ends of the non-projective dependencies. Therefore, our idea is to go through the sentence in order to predict, for each word, whether a non-projective dependency could end on the word (by adding valency $\swarrow l$ or $\searrow l$ in the list θ) or should start from it (by adding valency $\nwarrow l$ or $\nearrow l$ in the list θ). As soon as dual valencies are collected in θ, they are removed from it (according to the **FA** principle) and the corresponding non-projective dependency is added to the set of dependencies.

In the second step, the valencies associated with the left dependencies are computed, i.e. the valencies of the form $\swarrow l$ and $\nwarrow l$. The sentence is linearly covered from left to right, as in the previous projective step. Details of the transitions are presented in Table 3. The Shift transition is the same as during the previous step and allows to cover the sentence classically from left to right. The PutValency transition makes possible to predict, for the first word of the buffer, exactly one southwest valency $\swarrow l$, which means that a left dependency labelled l can end on this word. In addition, the valency is concatenated at the end

Transition	Application	Condition
PutValency($\swarrow l$)	$(\sigma, w_i \mid \beta, \theta, A) \Rightarrow (\sigma \mid w_i, \beta, \theta \swarrow l^i, A)$	$\swarrow l^i \notin \theta$
Dist-Left($\nwarrow l$)	$(\sigma, w_j \mid \beta, \theta_1 \swarrow l^i \theta_2, A) \Rightarrow (\sigma, w_j \mid \beta, \theta'_1\theta'_2, A \cup \{(j, l, i)\})$	$\swarrow l \notin \theta_2 \wedge \forall k \swarrow k^i \notin \theta'_1\theta'_2$
Shift	$(\sigma, w_i \mid \beta, \theta, A) \Rightarrow (\sigma \mid w_i, \beta, \theta, A)$	

Table 3: Transitions of the left non-projective method.

[2]The words which were not attached during the parsing are automatically attached to the root node w_0.

of θ. The transition Dist-Left is applied when the first word of the buffer receives the dual valency (i.e. a valency of the form $\nwarrow l$). If at least one valency $\swarrow l$ belongs to θ then the last one is removed from θ and the non-projective dependency corresponding to the pair of dual valencies $\swarrow l \nwarrow l$ (left non-projective labelled l) is added to A.

Therefore, for a given sentence, the initial configuration of this system is $([w_0], [w_1, ..., w_n], (), A')$ where A' is the projective dependency structure predicted by the arc-eager method. And the terminal configuration is a quadruplet of the form $([w_0, ..., w_n], [], \theta', A'')$ where θ' could contain southwest valencies which did not match with their dual and A'' is a partially non-projective dependency structure.

The third step uses the inverse method of the previous step and allows to predict right non-projective dependencies. In this method, the sentence is linearly covered from right to left. The initial configuration $([w_0, .., w_{n-1}], [w_n], (), A'')$ contains the partial dependency structure A'' produced by the last method and the terminal configuration $([w_0], [w_1, ..., w_n], \theta'', A''')$ contains the fully non-projective dependency structure A'''. The transitions used here are presented in Table 4. This time, the PutValency transition adds only southeast valencies ($\searrow l$) at the beginning of θ and pops the first word of σ to push it into β. The Dist-Right transition adds a right non-projective dependency in the set of arcs by predicting a dual valency of the form $\nearrow l$. Finally, the RShift transition pops the first word of σ to push it in β.

Transition	**Application**	**Condition**
PutValency($\searrow l$)	$(\sigma \mid w_i, \beta, \theta, A) \Rightarrow (\sigma, w_i \mid \beta, \searrow l^i \theta, A)$	$\searrow l^i \notin \theta$
Dist-Right($\nearrow l$)	$(\sigma \mid w_j, \beta, \theta_1 \searrow l^i \theta_2, A) \Rightarrow (\sigma \mid w_j, \beta, \theta'_1 \theta'_2, A \cup \{(j, l, i)\})$	$\searrow l \notin \theta_1 \wedge \forall k \searrow k^i \notin \theta'_1 \theta'_2$
RShift	$(\sigma \mid w_i, \beta, \theta, A) \Rightarrow (\sigma, w_i \mid \beta, \theta, A)$	

Table 4: Transitions of the right non-projective method.

The splitting of the non-projective dependencies prediction on two different methods is essential to find the right non-projective dependencies as well as the left ones. Practically, finding the head (i.e. the $\nearrow l$ and $\nwarrow l$ valencies) of a non-projective dependency is easier once the dependent (i.e. the $\searrow l$ and $\swarrow l$ valencies) has been previously predicted. Indeed, the prediction system benefits of information about the presence of the head valency in θ to predict the dual valency. Moreover, the heads are predicted more efficiently whether the projective dependency associated with the word was predicted with the right label during the first parsing step. The next section presents the prediction system and the features needed to proceed good transition predictions.

The application of these two steps on the sentence seen in Figure 2 are shown on Table 5. The

Transition	**Configuration**			
	([w$_0$],	[Il,...,.],	(),	A)
Shift	$\Rightarrow$ ([w$_0$,Il],	[y,...,.],	(),	A)
PutValency($\swarrow$clit-l-obj)	$\Rightarrow$ ([w$_0$,Il],	[y,...,.],	($\swarrow$clit-l-obj),	A)
Shift	$\Rightarrow$ ([w$_0$,...,y],	[est,...,.],	($\swarrow$clit-l-obj),	A)
Shift	$\Rightarrow$ ([w$_0$,...,est],	[alle,...,.],	($\swarrow$clit-l-obj),	A)
DistLeft($\swarrow$clit-l-obj)	$\Rightarrow$ ([w$_0$,...,est],	[alle,...,.],	(),	$A_1 = A \cup \{(4,\text{clit-l-obj}, 2)\}$)
Shift **(x6)**	$\Rightarrow$ ([w$_0$,...,.],	[],	(),	A_1)
	([w$_0$,... ,famille],	[.],	(),	A_1)
RShift	$\Rightarrow$ ([w$_0$,...,],	[famille,.],	(),	A_1)
RShift **(x3)**	$\Rightarrow$ ([w$_0$,...,,],	[soutenu,...,.],	(),	A_1)
PutValency($\swarrow$modif)	$\Rightarrow$ ([w$_0$,...,,],	[soutenu,...,.],	($\swarrow$modif),	A_1)
RShift **(x5)**	$\Rightarrow$ ([w$_0$],	[il,...,.],	($\swarrow$modif),	A_1)
DistLeft($\swarrow$modif)	$\Rightarrow$ ([w$_0$],	[il,...,.],	(),	$A_2 = A_1 \cup \{(1,\text{modif}, 6)\}$)

Table 5: Transition sequences of the left and right non-projective steps on the sentence in Figure 2.

projective structure built during the first step (Figure 2) is substituted to the set of arcs A in the initial configuration of the left non-projective step. The non-projective dependency structure A_2 provided at the end of the right (final) non-projective step is presented in Figure 3.

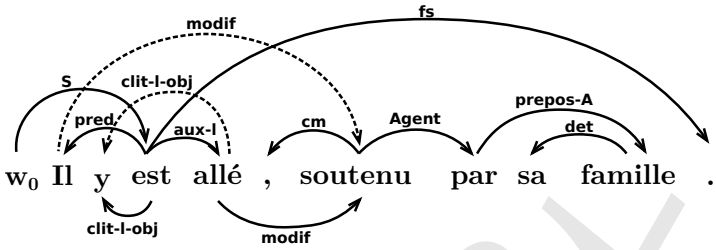

Figure 3: Non-projective dependency structure of the sentence in Figure 2.

4.3 Oracle

The transition-based systems are particularly interesting for deterministic data-driven parsing. Associated with a statistical method, such as a probabilistic graphical model or a linear classifier, and suitable features, the prediction of the transitions is very efficient. It ensures a deterministic parsing in linear time for both the projective arc-eager method and our two non-projective post-processing methods.

Previous work such as (Yamada and Matsumoto, 2003) shows that support vector machines (SVM) allow to achieve good scores on dependency parsing when associated with a transition-based system. Therefore, we chose to use this classifier to predict the transitions of our two post-processing methods. Moreover, the arc-eager method (i.e. *nivreeager*) being already successfully implemented and optimized, we decided to use the MaltParser (Nivre et al., 2007) to perform the projective dependency parsing.

For this projective step, the features are composed of classical features such as the word forms, POS-tags and dependency labels of the current words (the first elements of the stack and the buffer), their neighbors and their attached dependents. For the two non-projective steps the feature pattern includes in addition some features on the projective head of the first word of the buffer and the list of the valencies remaining in θ. The feature pattern is presented in Table 6. Nevertheless, the SVM model bears only numerical features. And each feature must be converted into a binary feature determining its absence or presence. For the valencies, the features denotes the absence or presence of each possible valency label in θ.

Feature Pattern	
• Word forms:	• POS-tags:
$w_{\{i-1,i+1\}}$	$t_{\{i-2,i+2\}}$
w_j	t_j
• Labels:	• Valencies:
l_j (projective dependency label)	$(v_0, ..., v_k)$ (the list of valencies in θ)
$(l_{j_1}, ..., l_{j_n})$ (the list of dependency labels)	

Table 6: Features for the prediction of transition in the two inverse methods. i is the position of the first word in β, j is the position of the head of w_i, the list of dependency labels is the list of labels of the right or left dependents of the head (depending on the right or left method).

5 Evaluation

In order to evaluate the efficiency of our approach, we decided to experiment on a dependency treebank for which the data were annotated following the formalism of the categorial dependency grammars[3]. We call this treebank the CDG Treebank 1. Moreover, in order to evaluate the adaptation of our method

[3]The treebank is not yet publicly available. But the authors have made it available to us.

Draft

to standard treebanks we would like to perform the method on data for which the anchors would have been artificially created. Therefore, we build a second treebank from the first one, which we call the CDG Treebank 2, in which the original anchors are replaced by artificial anchors generated by the projectivization step of the pseudo-projective method of Nivre and Nilsson (2005).

5.1 Non-Projective Dependency Treebank

The CDG Treebank 1 contains 3030 sentences of French, each paired with a dependency structure. The dependency structures are composed of both projective and non-projective dependencies. Out of the 37580 dependencies (excluding the anchor ones), 3.8% are non-projective. Hence, 41% of the dependency structures of the treebank contain at least one non-projective dependency.

The data were annotated semi-automatically using the CDG Lab (Alfared et al., 2011), a development environment dedicated to large scale grammar and treebank development. Thus, the annotations followed the formalism proposed by the categorial dependency grammar of French (Dikovsky, 2011). The labels of the dependencies are the 117 categories used by the grammar. Most of the dependency labels (89) are exclusively associated with projective dependencies. 23 labels can be associated both with projective and non-projective dependencies. Among these ones the most frequent are clitics, negatives, objects, reflexives and copredicates. In most of the cases, clitics, negatives and reflexives are associated with short dependencies (generally, one or two words separate the head from the dependent) whereas copredicates or apposition are often associated with distant dependencies (the heads and dependents can be located at the opposite ends of the sentence). Four dependency labels are exclusively associated with non-projective dependencies, they are particular cases of aggregation, copula, comparison and negation.

The grammar and the treebank were developed simultaneously. Consequently, a large part of the sentences were used to develop the grammar and were chosen to cover as much as possible the syntactic phenomenon of French. The treebank contains sentences from newspaper, 19^{th} and 20^{th} century literary works and plain language.

To build the CDG Treebank 2, we removed the anchors of the dependency structures of the CDG Treebank 1 and added the projective dependencies generated by projectivization[4]. Note that, 90.9% of the anchors are the same between the two CDG treebanks.

5.2 Experimental Settings

We evaluate our method through a 10-fold cross-validation on the non-projective dependency treebank. First, we train the prediction models (the MaltParser training model and the SVM model) on each training set containing 90% sentences of the treebank. Second, each fold of our testing data sets is tagged with Part-Of-Speech tags using Melt (Denis and Sagot, 2009), a POS-tagger that achieves high score on French. Then the sentences are parsed.

In order to estimate the benefit of our method, our results are compared with those obtained by the methods proposed by the MaltParser. The table shows the results of the methods that give the best results among the non-projective ones and the best results among the projective ones (associated with the pseudo-projective method (Nivre and Nilsson, 2005)):

- the *covnonproj* (non-projective) method inspired by Covington (2001);
- the *nivreeager* (projective) method associated with the pseudo-projective method.

For a fair comparison, the scores are computed on the same data for each experiments, i.e. on the non-projective structures minus the anchors and the dependencies combined with punctuations.

Moreover, in order to demonstrate that our method can be applied successfully on standard treebanks, the experiments are performed on the CDG Treebank 1 an 2. The comparison scores that are used in these experiments are:

[4]The labels of the artificial anchors do not contain additional encoded information. They are identical to the labels of the non-projective dependencies.

Draft

- the label accuracy (LA), i.e. the percentage of words for which the correct label is assigned;
- the unlabelled attachment score (UAS), i.e. the percentage of words for which the correct dependency is assigned;
- the labelled attachment score (LAS), i.e. the percentage of words for which the correct labelled dependency is assigned.

5.3 Experimental Results

The results of the experiments are presented in Table 7. First, we notice that the scores relating to projective dependencies of our method, both for CDG Treebank 1 (3) and CDG Treebank 2 (4), are better than those obtained by the covnonproj method (1) and equivalent to the pseudo-projective method (2). We assume that finding non-projective dependencies at the same time as the projective ones is more difficult than finding projective dependencies only. Moreover, the scores on non-projective dependencies

	All dependencies			**Projective Dep.**			**Non-projective Dep.**		
	LA	UAS	LAS	LA	UAS	LAS	LA	UAS	LAS
(1) covnonproj	82.2	85.5	78.0	82.8	86.2	78.7	68.7	68.7	62.7
(2) pseudoproj [5]	83.6	85.9	78.7	**84.1**	**87.0**	**79.7**	73.5	56.9	53.5
(3) non-projLR (CDGTbk1)	**83.7**	**86.3**	**79.1**	**84.1**	86.9	79.6	**75.5**	70.2	66.3
(4) non-projLR (CDGTbk2)	**83.7**	86.2	79.0	**84.1**	86.9	79.5	**75.5**	**70.5**	**66.7**

Table 7: Results of the non-projective dependency parsing comparing the MaltParser methods (1) and (2) with ours (3).

are particularly interesting. Our method achieves far better scores on non-projective dependencies than the other two. The label accuracy (LA) achieves significantly better scores (+6.8) than the covnonproj method. Indeed, the projective step allows to find the anchors which are a kind of projective dependencies, so there are easier to predict than the non-projective dependencies. Thus, the label accuracy of the non-projective dependencies takes advantage of the good results of the anchors which were not paired with a non-projective dependency during the second and third parsing steps. Concerning the attachment scores, our method still outperforms the two others. Globally, our method allows to recover the head of the non-projective dependencies more successfully.

The non-projective dependencies can be also compared depending on their direction. The left non-projective dependencies achieve far better scores (75.0% LAS) than the right non-projective dependencies (42.7% LAS). We know that the non-projective step performed from right to left is essential to recover the right non-projective dependencies. In fact, finding the right non-projective dependencies by performing the non-projective step from left to right seems almost infeasible because it is essential to find the dependent first. Therefore, the problem comes essentially from the bad prediction of the anchors during the projective step. Indeed, only 51.4% of the words associated with a right non-projective dependency receive the correct label (LA), compared with 84.2% for those associated with left non-projective dependencies. The under-representation of the right non-projective dependencies (25% of the non-projective dependencies) in the treebank is a first explanation. But, even the more frequent labels (associated with right non-projective dependencies) achieve low scores. Moreover, we noticed that even the right projective dependencies always achieve lower scores than the left projective dependencies. This problem may suggest that the use of a left-to-right projective method is not appropriate to predict the right dependencies.

Furthermore, we note that our method achieve equivalent scores on CDG Treebank 1 and CDG Treebank 2, and even slightly better for non-projective dependencies with the use of artificial anchors. This suggest that our method could be succesfully applied to standard treebanks in which artificial anchors would have been added.

[5] The pseudo-projective method were applied with the option "path" for projectivization and deprojectivization.

6 Conclusion

We propose a three-step method retrieving separately the projective dependencies and anchors, the left non-projective dependencies and the right non-projective dependencies through the use of a mixed dependency representation. The projective step and the two non-projective steps are performed in linear time and allow to outperform state-of-the-art transition-based scores on non-projective dependencies. The method needs a learning corpus that associate to each non-projective dependency a projective anchor. Thus the method is well adapted to CDG treebanks. But we showed that the method can be applied to standard treebanks by adding artificial anchors with the use of a method of projectivization.

One of the advantages of our method is a significant improvement on the label accuracy for the non-projective dependencies. The efficiency of the two non-projective methods depends on the good results of the projective parsing. Moreover, performing the non-projective parsing from left-to-right and from right-to-left raises interesting questions on how to recover the right and left dependencies for both projective and non-projective methods.

Acknowledgement

We want to thank Danièle Beauquier and Alexander Dikovsky for giving us the CDG Treebank on which we experimented our system. Moreover, we want to thank all our reviewers : the anonymous reviewers of Coling for their accurate reviews, and the members of the team TALN of the University of Nantes (Colin de la Higuera, Florian Boudin and the master students) who reviewed our work with a fresh eye.

References

Ramadan Alfared, Denis Béchet, and Alexander Dikovsky. 2011. CDG Lab: a Toolbox for Dependency Grammars and Dependency Treebanks Development. In *Proceedings of the International Conference on Dependency Linguistics*, DEPLING 2011, pages 272–281, Barcelona, Spain, September.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Michael Dekhtyar and Alexander Dikovsky. 2008. Generalized categorial dependency grammars. In *Trakhtenbrot/Festschrift*, LNCS 4800, pages 230–255. Springer.

Pascal Denis and Benoît Sagot. 2009. Coupling an Annotated Corpus and a Morphosyntactic Lexicon for State-of-the-Art POS Tagging with Less Human Effort. In *Proceedings of the Pacific Asia Conference on Language, Information and Computation*, PACLIC 2009, Hong Kong, China.

Alexander Dikovsky. 2011. Categorial Dependency Grammars: from Theory to Large Scale Grammars. In *Proceedings of the International Conference on Dependency Linguistics*, DEPLING 2011, September.

Keith Hall and Václav Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, IWPT 2005, pages 42–52.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. Dependency Parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

Igor Melcuk. 1988. *Dependency syntax : Theory and Practice*. State University of New York Press.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 99–106, Ann Arbor, Michigan.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Glsen Eryigit, Sandra Kbler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. *Natural Language Engineering*, 13:95–135, 6.

Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Comput. Linguist.*, 34(4):513–553, December.

Lucien Tesnière. 1959. *Éléments de syntaxe structurale*. Klincksieck.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the International Conference on Parsing Technologies*, IWPT 2003, pages 195–206.

Draft