Northern European Journal of Language Technology, 2010, Vol. 2, Article 1, pp 1–19
DOI 10.3384/nejlt.2000-1533.10211

Transition-Based Techniques for Non-Projective Dependency Parsing

Marco Kuhlmann Joakim Nivre

Uppsala University
Department of Linguistics and Philology
{marco.kuhlmann|joakim.nivre}@lingfil.uu.se

Abstract

We present an empirical evaluation of three methods for the treatment of non-projective structures in transition-based dependency parsing: pseudo-projective parsing, non-adjacent arc transitions, and online reordering. We compare both the theoretical coverage and the empirical performance of these methods using data from Czech, English and German. The results show that although online reordering is the only method with complete theoretical coverage, all three techniques exhibit high precision but somewhat lower recall on non-projective dependencies and can all improve overall parsing accuracy provided that non-projective dependencies are frequent enough. We also find that the use of non-adjacent arc transitions may lead to a drop in accuracy on projective dependencies in the presence of long-distance non-projective dependencies, an effect that is not found for the two other techniques.

1 Introduction

Transition-based dependency parsing is a method for natural language parsing based on transition systems for deriving dependency trees together with treebank-induced classifiers for predicting the next transition. The method was pioneered by Kudo and Matsumoto (2002) and Yamada and Matsumoto (2003) and has since been developed by a large number of researchers (Nivre et al., 2004; Attardi, 2006; Sagae and Tsujii, 2008; Titov and Henderson, 2007; Zhang and Clark, 2008). Similar techniques had previously been explored in other parsing frameworks by Briscoe and Carroll (1993) and Ratnaparkhi (1997), among others.

Using greedy deterministic search, it is possible to parse natural language in linear time with high accuracy as long as the transition system is restricted to projective dependency trees (Nivre, 2008). While strictly projective dependency trees are sufficient to represent the majority of syntactic structures in natural language, the evidence from existing dependency treebanks for a wide range of languages strongly suggests that certain linguistic constructions require non-projective structures – unless other representational devices such as empty nodes and coindexation are used instead (Buchholz and Marsi, 2006; Nivre et al., 2007). This poses an important challenge for transition-based dependency parsing, namely, to find methods for handling non-projective dependencies without a significant loss in accuracy and efficiency.

Nivre and Nilsson (2005) proposed a technique called *pseudo-projective parsing*, which consists in projectivizing the training data while encoding information about the transformation in augmented arc labels and applying an approximate inverse transformation to the output of the parser. In this way, the linear time complexity of the base parser can be maintained at the expense of an increase in the number of labels and a lack of complete coverage of non-projective structures. Attardi (2006) instead introduced transitions that add dependency arcs between the roots of non-adjacent subtrees – what we will call *non-adjacent arc transitions* – again maintaining linear time complexity but with incomplete coverage of non-projective structures. More recently, Nivre (2009) introduced a different extension that enables *online reordering* of the input words, leading to complete coverage of non-projective structures at the expense of an increase in worst-case complexity from linear to quadratic, although the expected running time is still linear for the range of inputs found in natural language.

All three techniques have been reported to improve overall parsing accuracy for languages with a non-negligible proportion of non-projective structures, but they have never been systematically compared on the same data sets. Such a comparison could potentially reveal strengths and weaknesses of different techniques and pave the way for further improvements, by combining existing techniques or by developing alternative methods. Moreover, since properties of the techniques may interact with properties of the language being analysed, we need to evaluate them on more than one language.

In this article, we perform a comparative evaluation of the three techniques on data from three different languages – Czech, English, German – presenting different challenges with respect to the complexity and frequency of non-projective structures. The methods are compared with respect to *theoretical coverage*, the proportion of structures in a language that the method can handle, and *empirical performance*, measured by overall accuracy and by precision and recall on projective and non-projective dependencies, respectively. The aim of the study is to compare alternative ways of extending the standard transition-based parsing method to non-projective dependencies, not to evaluate techniques for non-projective dependency parsing in general, which would be a much larger endeavour.

The remainder of the article is structured as follows. In Section 2, we define the background notions of transition-based dependency parsing. In Section 3, we review the three techniques for non-projective parsing that are evaluated in this article – pseudo-projective parsing, non-adjacent arc transitions, and online reordering. In Section 4, we describe the data and the training regime that we used in our experiments, as well as the metrics used to evaluate the three techniques. The results of our experiments are presented and discussed in Section 5. Section 6 concludes the article.

2 Preliminaries

We start by defining the syntactic representations used in dependency parsing, and introduce the framework of transition-based dependency parsing.

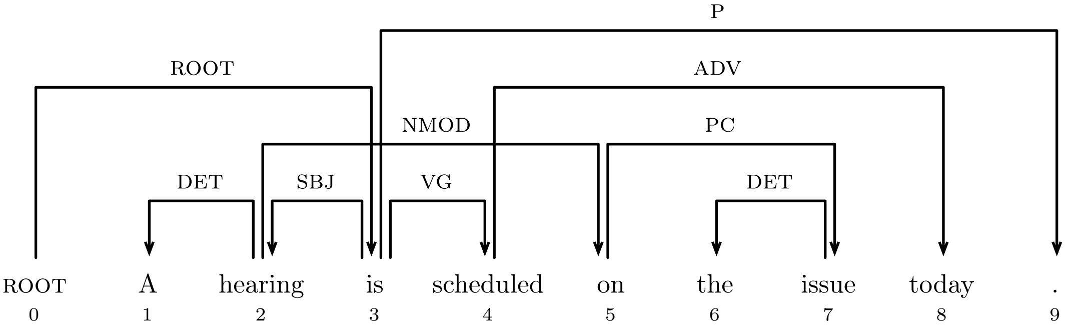


Figure 1: Dependency graph for an English sentence.

2.1 Dependency Graphs

Given a set L of labels, a *dependency graph* for a sentence $x = w_1, \dots, w_n$ is a labelled directed graph $G = (V_x, A)$, where $V_x = \{0, 1, \dots, n\}$ is the set of *nodes* of G, and $A \subseteq V_x \times L \times V_x$ is a set of *arcs*. An example dependency graph for the English sentence

> A hearing is scheduled on the issue today.

is shown in Figure 1. With the exception of the special node 0, each node represents (the position of) a word in x, while an arc $(i, l, j) \in A$ represents the information that the dependency relation denoted by l holds between w_i and w_j. As an example, the arc $(3, \textsc{sbj}, 2)$ in Figure 1 asserts that *hearing* acts as the grammatical subject (SBJ) of the verb *is*, and the arc $(7, \textsc{det}, 6)$ states that *the* is the determiner (DET) of *issue*. In an arc (i, l, j), the node i is called the *head* of the node j, and the node j is called a *dependent* of i. A dependency graph is a *dependency tree* if each node has at most one head and only the special node 0 has no head.

Given a dependency tree $G = (V_x, A)$, an arc $(i, l, j) \in A$ is *projective*, if each node k in the interval between i and j is a descendant of i, meaning that either $k = i$, or k is a descendant of some dependent of i. As an example, the arcs $(2, \textsc{nmod}, 5)$ and $(4, \textsc{adv}, 8)$ in Figure 1 are non-projective, while all the other arcs are projective. A dependency tree is *projective* if all of its arcs are projective.

2.2 Transition Systems

Each of the parsing techniques investigated in this article can be understood by means of a *transition system* in the sense of Nivre (2008). Such a system is a quadruple

$$S = (C, T, c_s, C_t),$$

where C is a set of *configurations*, T is a set of *transitions*, each of which is a partial function $t : C \to C$ from configurations to configurations, c_s is an *initialization function*, mapping sentences to configurations in C, and $C_t \subseteq C$ is a set of *terminal configurations*.

The transition systems that we investigate in this article differ only with respect to their sets of transitions, and are identical in all other aspects. In all of them, a *configuration* for a sentence $x = w_1, \dots, w_n$ is a triple $c = (\sigma, \beta, A)$, where σ and β are disjoint lists of nodes in V_x, and A is a set of arcs. We will refer to the list σ as the *stack*, and the list β as the *buffer* of the configuration, and write σ with its topmost element to the right, and β with its first element to the left. The *dependency graph associated with* c is the graph $G_c = (V_x, A)$. The *initialization function* maps a sentence $x = w_1, \dots, w_n$ to the configuration $c = ([0], [1, \dots, n], \emptyset)$. In this configuration, the special node 0 is the only node on the stack, while all other nodes are in the buffer, and the set of arcs is empty. The set of *terminal configurations* is the set of all configurations of the form $c = ([0], [], A)$, for any set of arcs A. In these configurations, the special node 0 is the only node on the stack, and the buffer is empty.

2.3 Transition-Based Dependency Parsing

A *guide* for a transition system $S = (C, T, c_s, C_T)$ is a function g that maps a configuration c to a transition t that is defined on c. Given a transition system S and a guide g for S, the following is an algorithm for deterministic transition-based dependency parsing:

```
Parse($x$)
  $c \leftarrow c_s(x)$
  while $c \notin C_T$
      do $t \leftarrow g(c)$; $c \leftarrow t(c)$
  return $G_c$
```

This algorithm repeatedly calls the guide, constructing a sequence of configurations that ends with a terminal configuration, and returns the dependency graph associated with that configuration. Guides can be constructed in many ways, but the standard approach in data-driven dependency parsing is to use a classifier trained on treebank data (Yamada and Matsumoto, 2003; Nivre et al., 2004).

Based on this algorithm, we define a deterministic transition-based *dependency parser* as a pair $P = (S, g)$, where S is a transition system, and g is a guide for S. Given a sentence x, the *parse* assigned to x by P is the dependency graph Parse(x).

2.4 Oracles

To train a transition-based dependency parser, one can use an *oracle* o, a guide that has access to the gold-standard graph G for a sentence x and based on this knowledge predicts gold-standard transitions for x and G. Each pair $(c, o(c))$ of a current configuration c and the transition $o(c)$ predicted by the oracle defines an instance of a classification problem. A classifier for this problem can then be turned into a guide: This guide returns the transition predicted by the classifier if it is admissible, and otherwise returns a default transition, which depends on the transition system.

The *coverage* of an oracle parser $P = (S, o)$, denoted by $\mathcal{D}_P$, is the set of all parses it can assign, to any sentence x and gold-standard graph G for x. Given a class $\mathcal{D}$ of dependency graphs, a parser P is *sound for* $\mathcal{D}$, if $\mathcal{D}_P \subseteq \mathcal{D}$, and *complete for* $\mathcal{D}$, if $\mathcal{D} \subseteq \mathcal{D}_P$. The coverage of an oracle parser $P = (S, o)$ can be used to compute upper bounds on the empirical performance of a parser $P' = (S, g)$ that uses a treebank-induced guide g trained on data generated by o. The upper bounds are reached when the predictions of g coincide with the predictions of o for all sentences. This is what we will call the *theoretical coverage of* P'.

Transition	Source configuration	Target configuration	Condition
Shift	$(\sigma, i\|\beta, A)$	$(\sigma\|i, \beta, A)$	
Left-Arc$_l$	$(\sigma\|i\|j, \beta, A)$	$(\sigma\|j, \beta, A \cup \{(j, l, i)\})$	$i \neq 0$
Right-Arc$_l$	$(\sigma\|i\|j, \beta, A)$	$(\sigma\|i, \beta, A \cup \{(i, l, j)\})$	
Left-Arc-2$_l$	$(\sigma\|i\|j\|k, \beta, A)$	$(\sigma\|j\|k, \beta, A \cup \{(k, l, i)\})$	$i \neq 0$
Right-Arc-2$_l$	$(\sigma\|i\|j\|k, \beta, A)$	$(\sigma\|i, j\|\beta, A \cup \{(i, l, k)\})$	
Swap	$(\sigma\|i\|j, \beta, A)$	$(\sigma\|j, i\|\beta, A)$	$0 < i < j$

Figure 2: Transitions for dependency parsing.

3 Techniques for Non-Projective Parsing

With the formal framework of transition-based dependency parsing in place, we now define the concrete parsing techniques evaluated in this article.

3.1 Parser 0: Projective Parsing

The baseline for our experiments is the projective dependency parser presented in Nivre (2009). This parser, which we will refer to by the name P_0, is based on a transition system with three transitions, which are specified in Figure 2:

(i) Shift dequeues the first node in the buffer and pushes it to the stack.

(ii) Left-Arc$_l$ adds a new arc with label l from the topmost node on the stack to the second-topmost node and removes the second-topmost node.

(iii) Right-Arc$_l$ adds a new arc with label l from the second-topmost node on the stack to the topmost node and removes the topmost node.

For training this parser, we use an oracle that predicts the first transition returned by the tests $1, 2, 6$ in Figure 3 in that order. The resulting parser is sound and complete with respect to the class of projective dependency trees (Nivre, 2008), meaning that each terminal configuration defines a projective dependency tree, and each projective dependency tree can be constructed using the parser. Its runtime is linear in the length of the input sentence.

3.2 Parser 1: Pseudo-Projective Parsing

The first non-projective parser that we use in our experiments is based on *pseudo-projective parsing* (Nivre and Nilsson, 2005), a general technique for turning a projective dependency parser into a non-projective one. In a pre-processing step, one transforms the potentially non-projective gold-standard trees into projective trees by replacing each non-projective arc (i, l, j) with a projective arc (k, l, j) whose head k is the closest transitive head of i. The training data is enriched with information about how to undo these ‘lifting’ operations. Taking the projectivized trees as the new gold-standard, one then

Test	Configuration	Conditions		Prediction
1	$(\sigma\|i\|j, \beta, A)$	$(j, l, i) \in A_G$	$A_G^i \subseteq A$	Left-Arc$_l$
2	$(\sigma\|i\|j, \beta, A)$	$(i, l, j) \in A_G$	$A_G^j \subseteq A$	Right-Arc$_l$
3	$(\sigma\|i\|j\|k, \beta, A)$	$(k, l, i) \in A_G$	$A_G^i \subseteq A$	Left-Arc-2$_l$
4	$(\sigma\|i\|j\|k, \beta, A)$	$(i, l, k) \in A_G$	$A_G^k \subseteq A$	Right-Arc-2$_l$
5	$(\sigma\|i\|j, \beta, A)$	$j <_G i$		Swap
6	$(\sigma, i\|\beta, A)$			Shift

Figure 3: Oracle predictions used during training with a gold-standard tree $G = (V_x, A_G)$. We write A_G^i for the set of those arcs in A_G that have i as the head. We write $j <_G i$ to say that j precedes i with respect to the canonical projective ordering in the gold-standard tree.

trains a standard projective parser in the usual way. In a post-processing step, the encoded information is used to deprojectivize the output of the projective dependency parser into a non-projective tree. Since the information required to undo arbitrary liftings can be very complex, deprojectivization is usually implemented as an approximate transformation.

The coverage of a pseudo-projective parser depends on the specific encoding of how to undo the projectivization. The more information one uses here, the more accurate the deprojectivization will be, but the more burden one puts onto the projective parser, which has to cope with a more complex label set. In this article, we focus on the *Head* encoding scheme, which gave the best performance in Nivre and Nilsson (2005). In this scheme, the dependency label of each lifted arc (i, l, j) is replaced by the label $l \uparrow l'$, where l' is the dependency relation between i and its own head. During deprojectivization, this information is used to search for the first possible landing site for the 'unlifting' of the arc, using top-down, left-to-right, breadth-first search.

To train the pseudo-projective parser, which we will refer to as P_1, we use the same oracle as for the projective P_0.

3.3 Parser 2: Non-Adjacent Arc Transitions

The projective baseline parser P_0 adds dependency arcs only between nodes that are adjacent on the stack. A natural idea is to allow arcs to be added also between non-adjacent nodes. Here, we evaluate a parser P_2 based on a transition system that extends the system for the baseline parser by two transitions originally introduced by Attardi (2006) (see Figure 2):[1]

(i) Left-Arc-2$_l$ adds an arc from the topmost node on the stack to the third-topmost node, and removes the third-topmost node.

(ii) Right-Arc-2$_l$ adds an arc from the third-topmost node on the stack to the topmost node, and removes the topmost node.

[1] In Attardi's paper, these transitions are called *Right2* and *Left2*, respectively.

Because of the non-adjacent transitions, the parser P_2 has larger coverage than the projective system P_0; however, it cannot derive *every* non-projective dependency tree, even though Attardi (2006) notes that LEFT-ARC-2 and RIGHT-ARC-2 are sufficient to handle almost all cases of non-projectivity in the training data. The full system considered by Attardi (2006) also includes more complex transitions.

To train P_2, we use an oracle that predicts the first transition returned by the tests $1, 2, 3, 4, 6$ in Figure 3 in that order. Such an oracle prefers to add arcs between nodes adjacent on the stack, but can resort to predicting a non-adjacent transition if adjacent transitions are impossible. The runtime of P_2 is still linear in sentence length.

3.4 Parser 3: Online Reordering

The third approach to non-projective dependency parsing that we explore in this article is to only add arcs between adjacent nodes as in projective parsing, but to provide the parser with a way to reorder the nodes during parsing. We refer to this approach as *online reordering*. The specific parser that we will evaluate here, which we will refer to as P_3, was proposed by Nivre (2009). It extends the projective parser by using a transition SWAP that switches the position of the two topmost nodes on the stack, moving the second-topmost node back into the buffer (see Figure 2).

For training, P_3 uses an oracle that predicts the first transition returned by the tests $1, 2, 5, 6$ in that order. The canonical projective ordering referred to in test 5 is defined by an inorder traversal of the gold standard dependency tree that respects the local ordering of a node and its children (Nivre, 2009). In our experiments, we actually employ an improved version of this oracle that tries to delay the prediction of the SWAP transition for as long as possible (Nivre et al., 2009); in previous experiments, this improved oracle has almost consistently produced better results than the oracle originally proposed by Nivre (2009). The obtained oracle parser is sound and complete with respect to the class of all dependency trees. Its worst-case runtime is quadratic rather than linear. However, Nivre (2009) observes that the expected runtime is still linear for the range of data attested in dependency treebanks.

4 Methodology

In this section we describe the methodological setup of the evaluation, including data sets, parser preparation and evaluation metrics.

4.1 Data Sets

Our experiments are based on training and development sets for three languages in the CoNLL 2009 Shared Task: Czech, English, German (Hajič et al., 2009), with data taken from the Prague Dependency Treebank (Hajič et al., 2001, 2006), the Penn Treebank (Marcus et al., 1993), and the Tiger Treebank (Brants et al., 2002). Since we wanted to be able to analyse the output of the parsers in detail, it was important to use development sets rather than final test sets, which is one reason why we chose not to work with the more well-known data sets from the tasks on dependency parsing in 2006 and 2007 (Buchholz and Marsi, 2006; Nivre et al., 2007), for which no separate development sets are available.

Table 1: Statistics for the data sets used in the experiments. # = number, % NP = percentage of sentences with non-projective analyses, L = average arc length in words.

	sentences		arcs overall		proj. arcs		non-proj. arcs	
	#	% NP	#	L	%	L	%	L
Czech								
training data	38,727	22.42%	652,544	3.64	98.1%	3.62	1.9%	4.49
development data	5,228	23.13%	87,988	3.65	98.1%	3.63	1.9%	4.45
English								
training data	39,279	7.63%	958,167	3.38	99.6%	3.36	0.4%	8.19
development data	1,334	6.30%	33,368	3.42	99.7%	3.40	0.3%	7.86
German								
training data	36,020	28.10%	648,677	4.08	97.7%	3.92	2.3%	10.58
development data	2,000	26.90%	32,033	3.93	97.7%	3.77	2.3%	10.29

Another reason is that the newer data sets contain automatically predicted part-of-speech tags (rather than gold standard annotation from treebanks), which makes the conditions of the evaluation more realistic. Although the study of parsing algorithm performance with gold standard annotation as input may be interesting to obtain upper bounds on performance, research has also shown that performance under realistic conditions may be drastically different (see, e.g., Eryigit et al., 2008). Since we specifically want to contrast theoretical upper bounds – as measured by our theoretical coverage – with empirical performance under realistic conditions, we therefore prefer to use the more recent data sets without gold standard annotation. From the data sets in the CoNLL 2009 Shared Task, the languages Czech, English and German were chosen because they represent different language types with respect to the frequency and complexity of non-projective structures, as seen below.

Table 1 gives an overview of the training and development sets used, listing number of trees (sentences) and arcs (words), percentage of projective and non-projective trees/arcs, and average length for projective and non-projective arcs. We see that non-projective structures are more frequent in Czech and German, where about 25% of all trees and about 2% of all arcs are non-projective, than in English, where the corresponding figures are below 10% and 0.5%, respectively. Nevertheless, English is similar to German in that non-projective arcs are on average much longer than projective arcs, while in Czech there is only a marginal difference in arc length. This seems to indicate that the majority of non-projective constructions in Czech are relatively local, while non-projective long-distance dependencies are relatively more frequent in English and German. However, it is worth emphasizing that the observed differences are dependent not only on structural properties of the languages but also on the annotation scheme used, which unfortunately is not constant across languages.

4.2 Parser Preparation

For each of the four parsers described in Section 3, we ran oracles on the training set to derive training instances for multi-class SVMs with polynomial kernels, using the LIBSVM package (Chang and Lin, 2001), which represents the state of the art in transition-based parsing (Yamada and Matsumoto, 2003; Nivre et al., 2006). Since the purpose of the experiments was to systematically compare different techniques for non-projective parsing, rather than estimate their best performance, we did not perform extensive feature selection or parameter optimization. Instead, we optimized a feature model only for the projective parser P_0. For the pseudo-projective parser P_1, we simply left the feature model as it was. For the parser P_2 with non-adjacent arc transitions, we extended the lookahead into the stack by one node, based on the intuition that this parser should be able to inspect the stack one level deeper than the projective parser to make use of non-adjacent transitions. For the parser P_3 with online reordering, finally, we added a new feature that allows the parser to inspect the part-of-speech tag of the last node swapped back into the buffer. In this way, we managed to keep the information available to the parser guides relatively constant while adapting to the special properties of each technique. The fact that we have not optimized all feature models separately is another reason for only presenting results on the development sets, saving the final test sets for future experiments. In the interest of replicability, complete information about features, parameters and experimental settings is published on the web.[2]

4.3 Evaluation Metrics

We have used four different evaluation metrics, which are all common in the dependency parsing literature and which complement each other by targeting different aspects of parser performance:

- **Attachment score:**
  Percentage of correct arcs (with or without labels).

- **Exact match:**
  Percentage of correct complete trees (with or without labels).

- **Precision:**
  Percentage of correct arcs out of *predicted* projective/non-projective arcs.

- **Recall:**
  Percentage of correct arcs out of *gold standard* projective/non-projective arcs.

Applying these metrics to the output of the oracle parsers on the development sets gives us upper bounds for each combination of parser and metric, what we call the *theoretical coverage* of each technique. Applying the same metrics to the output of the classifier-based guide parsers provides us with estimates of their *empirical performance.*

Note that the precision and recall measures used here are slightly unusual in that the number of correct arcs may be different in the two cases, because the projectivity status of an arc depends on the other arcs of the tree and may therefore be different in the

[2] `http://stp.lingfil.uu.se/~kuhlmann/nejlt2010.html`

parser output and in the gold standard. For this reason, it is not possible to compute the standard measure F_1 as the harmonic mean of precision and recall.

5 Results

In this section, we discuss the experimental results for the three non-projective parsers P_1, P_2 and P_3 in comparison to the projective baseline P_0.

5.1 Theoretical Coverage

Table 2 shows upper bounds for the performance of the parsers on the development data in terms of (labelled and unlabelled) attachment score and (labelled and unlabelled) exact match. This is complemented by Table 3, which shows the precision and recall for projective and non-projective dependencies, both over all sentences and separately for sentences with projective and non-projective trees, respectively. As expected, the strictly projective parser (P_0) has the lowest coverage on almost all metrics, and online reordering (P_3) is the only method with perfect coverage across the board. The difference is especially pronounced for the exact match scores (LEM/UEM), where P_0 only achieves about 75% on Czech and German (but close to 95% on English). Unsurprisingly, P_0 has very close to 100% recall on projective dependencies in all languages, but precision suffers because words that have a non-projective relation to their heads will by necessity be given an incorrect projective analysis. The reason that recall on projective dependencies does not always reach 100% in sentences with a non-projective analysis is that the projective oracle is not guaranteed to be correct even for projective dependencies when the overall structure is non-projective.

Turning to the two remaining techniques for non-projective parsing, we see that the pseudo-projective method (P_1) performs slightly better than non-adjacent arc transitions (P_2) on the overall metrics with close to perfect coverage on the attachment score metrics (LAS/UAS), but the differences are generally small and P_2 in fact has higher exact match scores for Czech. Nevertheless, there is an interesting difference in the performance on non-projective dependencies. With the use of non-adjacent transitions, P_2 has perfect precision – because it never constructs a non-projective arc that is not correct – but relatively low recall – because some non-projective dependencies are simply beyond its reach. With the use of pseudo-projective transformations, P_1 is always able to predict a non-projective analysis where this is relevant but sometimes fails to find the correct head because of post-processing errors – which affects precision as well as recall. The trend is therefore that P_1 has higher recall but lower precision than P_2, a pattern that is broken only for Czech, where P_2 has a much higher recall than usual. This result is undoubtedly due to the fact that non-projective dependencies are on average much shorter in Czech than in English and German, which also explains the unexpectedly high exact match scores for P_2 on Czech noted earlier.

Finally, it is worth noting that whereas P_1 has almost perfect precision and recall on projective dependencies, also in sentences with a non-projective analysis, P_2 in fact errs in both directions here. Precision errors can be explained in the same way as for P_0, that is, because some non-projective dependencies are out of reach, the parser is forced

Table 2: Theoretical coverage on the development data. LAS/UAS = labelled/unlabelled attachment score, LEM/UEM = labelled/unlabelled exact match

	LAS	UAS	LEM	UEM
Czech				
P_0	98.01%	98.09%	76.87%	76.87%
P_1	99.77%	99.85%	97.86%	97.86%
P_2	99.63%	99.74%	98.85%	98.97%
P_3	100%	100%	100%	100%
English				
P_0	99.72%	99.72%	93.70%	93.70%
P_1	99.98%	99.98%	99.55%	99.55%
P_2	99.61%	99.61%	98.35%	98.35%
P_3	100%	100%	100%	100%
German				
P_0	97.65%	97.65%	73.10%	73.10%
P_1	99.84%	99.84%	97.90%	97.90%
P_2	98.45%	98.45%	95.95%	95.95%
P_3	100%	100%	100%	100%

to over-predict projective dependencies. Recall errors, on the other hand, arise because of the bottom-up parsing strategy, where projective relations higher in the tree may be blocked if some non-projective subtree cannot be completed. This is especially noticeable for English and German, where non-projective dependencies tend to be longer and where both precision and recall for projective dependencies drops to about 95% in sentences with a non-projective analysis.

5.2 Empirical Performance

Table 4 reports the attachment and exact match scores of the classifier-based parsers, and Figure 4 shows which of the differences are statistically significant at the 0.01 and 0.05 level according to McNemar's test for proportions. Table 5 gives the breakdown into projective and non-projective arcs, both overall and in projective and non-projective sentences, respectively. The overall impression is that adding techniques for handling non-projective dependencies generally improves parsing accuracy, although there are a few exceptions that we will return to. Quantitatively, we see the greatest improvement in the exact match scores, which makes sense since these are the metrics for which the theoretical upper bounds improve the most, as seen in the previous section.

However, we also see that there are interesting differences between the three languages. For Czech, all three techniques perform significantly better than the projective baseline (P_0) with almost no significant differences between them. For English, only the pseudo-projective parser (P_1) actually improves over the baseline, and the use of non-adjacent arc transitions (P_2) is significantly worse than the other two techniques. For German, finally,

Table 3: Theoretical coverage on the development data broken down into projective and non-projective arcs, for all sentences and separately for sentences with a projective/non-projective analysis. P/R = unlabelled precision/recall

	all sentences				projective analyses				non-projective analyses			
	proj. arcs		non-proj. arcs		proj. arcs		non-proj. arcs		proj. arcs		non-proj. arcs	
	P	R	P	R	P	R	P	R	P	R	P	R
Czech												
P_0	98.09%	99.99%		0.00%	100%	100%			93.73%	99.96%		0.00%
P_1	99.91%	99.99%	96.57%	92.47%	100%	100%			99.68%	99.96%	96.57%	92.47%
P_2	99.73%	99.80%	100%	96.54%	100%	100%			99.09%	99.32%	100%	96.54%
P_3	100%	100%	100%	100%	100%	100%			100%	100%	100%	100%
English												
P_0	99.72%	100%		0.00%	100%	100%			96.01%	100%		0.00%
P_1	100%	100%	94.68%	93.68%	100%	100%			99.96%	100%	94.68%	93.68%
P_2	99.61%	99.68%	100%	77.89%	100%	100%			94.40%	95.27%	100%	77.89%
P_3	100%	100%	100%	100%	100%	100%			100%	100%	100%	100%
German												
P_0	97.65%	100%		0.00%	100%	100%			93.62%	99.99%		0.00%
P_1	99.99%	100%	93.60%	93.35%	100%	100%			99.98%	100%	93.60%	93.35%
P_2	98.41%	98.78%	100%	84.84%	100%	100%			95.52%	96.53%	100%	84.84%
P_3	100%	100%	100%	100%	100%	100%			100%	100%	100%	100%

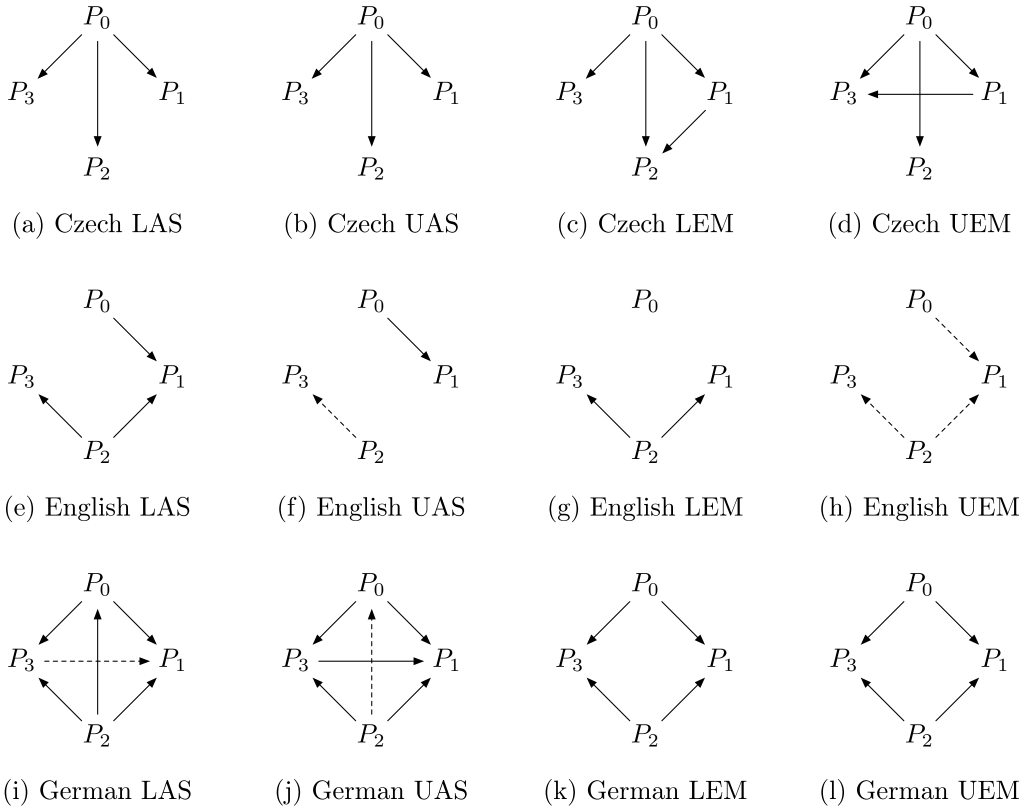
Table 4: Empirical performance on the development data. LAS/UAS = labelled/unlabelled attachment score, LEM/UEM = labelled/unlabelled exact match

	LAS	UAS	LEM	UEM
Czech				
P_0	79.65%	85.40%	27.87%	37.03%
P_1	80.58%	86.28%	30.74%	40.74%
P_2	80.64%	86.24%	31.87%	41.45%
P_3	80.71%	86.34%	31.33%	41.83%
English				
P_0	84.87%	88.47%	17.84%	31.48%
P_1	85.01%	88.55%	18.44%	32.23%
P_2	84.64%	88.37%	16.64%	30.06%
P_3	85.00%	88.63%	18.59%	32.16%
German				
P_0	83.27%	86.19%	31.90%	39.45%
P_1	84.07%	87.01%	34.65%	42.90%
P_2	82.76%	85.84%	32.50%	40.20%
P_3	83.75%	86.63%	34.60%	42.55%

online reordering (P_3) and P_1 are both significantly better than P_0 and P_2, with significantly better attachment scores for P_3. Quantitatively, we find the greatest improvement for Czech with up to 1 percentage point for LAS/UAS and 3–4 percentage points for LEM/UEM, followed by German with slightly smaller improvements. For English, the improvements are very marginal and mostly non-significant, which is probably due to the much lower frequency of non-projective structures in the English data set, which means that the theoretical upper bounds on performance are much closer to those of the projective parser (cf. Table 2). The fact that performance improves more for Czech than for German, despite a slightly higher frequency of non-projective structures in the latter language, probably has to do with the much higher average arc length for non-projective dependencies in German.

The picture that seems to emerge from these results is that both pseudo-projective parsing and online reordering improve overall parsing accuracy for languages where non-projective dependencies have a non-negligible frequency, exemplified by Czech and German, and do not hurt performance if such dependencies are rare, as in the case of English. The use of non-adjacent arc transitions, by contrast, improves parsing accuracy only if non-projective dependencies are both frequent and short, as in Czech, but can otherwise hurt performance significantly, as in English and German.

Zooming in on precision and recall for projective and non-projective dependencies, it is interesting to see that the inferior performance of P_2 on English and German is primarily due to a drop in recall (for English also in precision) on *projective* dependencies and especially in sentences with a non-projective analysis. This is related to the observation in Section 5.1 that recall errors on projective dependencies may arise because of the



(a) Czech LAS

(b) Czech UAS

(c) Czech LEM

(d) Czech UEM

(e) English LAS

(f) English UAS

(g) English LEM

(h) English UEM

(i) German LAS

(j) German UAS

(k) German LEM

(l) German UEM

Figure 4: Statistical significance of the results in Table 4 (McNemar's test). An arrow $P \to P'$ indicates that the respective score for the parser P' is better than the score for P with a difference statistically significant beyond the 0.01 level (solid line) or 0.05 level (dashed line).

Table 5: Empirical performance on the development data broken down into projective and non-projective arcs. P/R = unlabelled precision/recall

	all analyses				projective analyses				non-projective analyses			
	proj. arcs		non-proj. arcs		proj. arcs		non-proj. arcs		proj. arcs		non-proj. arcs	
	P	R	P	R	P	R	P	R	P	R	P	R
Czech												
P_0	85.40%	86.96%		5.02%	87.44%	87.44%			80.75%	85.79%		5.02%
P_1	86.41%	86.92%	76.10%	53,52%	87.54%	87.42%	23.48%		83.75%	85.70%	82.14%	53.52%
P_2	86.34%	86.66%	79.73%	64.99%	87.37%	87.25%	34.33%		83.91%	85.21%	84.87%	64.99%
P_3	86.47%	86.75%	77.89%	65.53%	87.49%	87.37%	27.42%		84.05%	85.23%	83.08%	65.53%
English												
P_0	88.47%	88.70%		8.42%	88.75%	88.75%			84.79%	87.96%		8.42%
P_1	88.60%	88.66%	63.24%	47.37%	88.75%	88.71%	27.78%		86.57%	87.96%	76.00%	47.37%
P_2	88.42%	88.49%	62.50%	46.32%	88.74%	88.71%	22.22%		84.10%	85.56%	78.26%	46.32%
P_3	88.71%	88.74%	55.84%	49.47%	88.87%	88.81%	12.50%		86.55%	87.83%	75.47%	49.47%
German												
P_0	86.19%	88.16%		4.26%	88.60%	88.60%			82.06%	87.35%		4.26%
P_1	87.23%	88.06%	71.09%	43.62%	88.65%	88.49%	11.36%		84.70%	87.27%	78.04%	43.62%
P_2	86.11%	86.71%	69.08%	49.73%	87.68%	87.42%	7.69%		83.32%	85.40%	78.03%	49.73%
P_3	87.04%	87.54%	61.98%	48.94%	88.62%	88.32%	2.82%		84.24%	86.10%	71.21%	48.94%

bottom-up parsing strategy, where projective relations higher in the tree are blocked if some non-projective subtree cannot be completed. This effect, which was visible in the theoretical coverage results, apparently seems to carry over to the empirical performance of classifier-base parsers. By contrast, for P_1 and P_3, we observe an increase in precision on projective dependencies in non-projective sentences – because the parser is no longer forced to predict projective approximations to non-projective structures – but without any corresponding drop in recall.

Turning to the performance specifically on non-projective dependencies, we see that all three techniques seem to result in relatively high precision – about 70–85% for all three languages – but substantially lower recall – slightly below 50% for English and German and up to 65% for Czech. The fact that both precision and recall is generally higher for Czech than for the other languages can probably again be explained by the fact that non-projective dependencies tend to be shorter there.

Comparing the three approaches, we see that the use of non-adjacent arc transitions (P_2) generally gives the highest precision, which is understandable given that the parser is restricted to consider non-projective dependencies with only one intervening subtree, dependencies that tend to be short and therefore easier to analyse in the context of a parser configuration. The use of online reordering (P_3), on the other hand, generally gives the highest recall (although P_2 is marginally better for German), which is natural since it is the only method that has perfect theoretical coverage in this respect. For Czech, P_3 also has very high precision, but for English and especially German it lags behind the two other systems. We hypothesize that this pattern can be explained by the greater average length of non-projective dependencies in the latter two languages, since a longer distance between the two non-adjacent structures requires the parser to perform a more complex sequence of Swap transitions in order to correctly recover the dependency, which increases the probability of error somewhere in the sequence.

The pseudo-projective approach (P_1), finally, exhibits relatively high precision on non-projective dependencies but sometimes suffers from low recall, which can probably partly be explained by the way its post-processing works. If the heuristic search fails to find a suitable 'landing site' for an arc with an augmented arc label, the arc is simply relabelled with an ordinary label and left in place. As a consequence, the pseudo-projective technique tends to underpredict non-projective dependencies – but for a different reason than the parser with non-adjacent arc transitions – which benefits precision at the expense of recall. On the other hand, pseudo-projective parsing is the technique that generally has the highest accuracy on projective dependencies, also in non-projective sentences, which is probably the reason why it is the only technique that significantly outperforms the baseline on English, where non-projective dependencies are rare and high precision and recall on projective dependencies therefore especially important for a net improvement.

6 Conclusion

The first conclusion of our study is that all three techniques for handling non-projective dependencies can improve accuracy in transition-based parsing, provided that these dependencies have a non-negligible frequency in the language at hand. The net effect on overall performance metrics like attachment score is quantitatively small, because of the low frequency of non-projective dependencies, but the probability of getting a completely correct parse clearly increases, as evidenced by the improved exact match scores. In this respect, our results corroborate earlier findings reported for the different techniques separately (Nivre and Nilsson, 2005; Attardi, 2006; Nivre, 2009).

The second conclusion is that all three techniques have very similar performance on non-projective dependencies, with relatively high precision, ranging from 70 to 85%, but lower recall, ranging from below 50% to at most 65%, but that there are significant differences in their performance on projective dependencies. These differences are mainly found in sentences for which the overall analysis is non-projective. In such sentences, all three techniques improve precision to varying degrees – because they are not forced to substitute projective dependencies for truly non-projective relations –, but the use of non-adjacent arc transitions may lead to a significant drop in recall if non-projective dependencies are too long to be handled by the transitions, in which case neighbouring projective dependencies may be blocked as well. This is a result that has not been reported in the literature before, and which emerges only as a result of a comparative study using data from languages with different characteristics.

Although the experiments presented in this article have already revealed significant differences both between languages and between techniques, it would be interesting to look in more detail at the different linguistic constructions that give rise to non-projective dependencies. Ideally, however, this should be done using annotation guidelines that are standardized across languages to ensure that we are not comparing apples and oranges, which probably calls for a community effort. Another direction for future research is to extend the analysis beyond transition-based techniques and compare the performance of other types of dependency parsers, in particular graph-based data-driven parsers (McDonald et al., 2005; McDonald and Pereira, 2006; Nakagawa, 2007; Martins et al., 2009), but also grammar-driven approaches like those of Foth et al. (2004) and Schneider (2008).

References

Attardi, Giuseppe. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. TIGER treebank. In *Proceeedings of the First Workshop on Treebanks and Linguistic Theories*, pages 24–42.

Briscoe, Edward and John Carroll. 1993. Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics* 19:25–59.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Chang, Chih-Chung and Chih-Jen Lin. 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Eryigit, Gülsen, Joakim Nivre, and Kemal Oflazer. 2008. Dependency parsing of Turkish. *Computational Linguistics* 34.

Foth, Kilian, Michael Daum, and Wolfgang Menzel. 2004. A broad-coverage parser for German based on defeasible constraints. In *Proceedings of KONVENS 2004*, pages 45–52.

Hajič, Jan, Eva Hajičová, Petr Pajas, Jarmila Panevová, and Petr Sgall. 2001. Prague Dependency Treebank 1.0. Linguistic Data Consortium, 2001T10.

Hajič, Jan, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, and Marie Mikulová. 2006. Prague Dependency Treebank 2.0. Linguistic Data Consortium, 2006T01.

Hajič, Jan, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18.

Kudo, Taku and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of the 6th Conference on Computational Language Learning (CoNLL)*, pages 63–69.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics* 19(2):313–330.

Martins, Andre, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the 47th Annual Meeting of the ACL and the Fourth International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350.

McDonald, Ryan and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88. Trento, Italy.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference (HLT) and the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–530. Vancouver, Canada.

Nakagawa, Tetsuji. 2007. Multilingual dependency parsing using global features. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 952–956.

Nivre, Joakim. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553.

Nivre, Joakim. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the 47th Annual Meeting of the ACL and the Fourth International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359.

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

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the Eighth Conference on Computational Natural Language Learning*, pages 49–56.

Nivre, Joakim, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

Nivre, Joakim, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the Eleventh International Conference on Parsing Technologies*, pages 73–76. Paris, France.

Nivre, Joakim and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106. Ann Arbor, USA.

Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–10.

Sagae, Kenji and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 753–760.

Schneider, Gerold. 2008. *Hybrid Long-Distance Functional Dependency Parsing*. Ph.D. thesis, Universität Zürich, Zürich, Switzerland.

Titov, Ivan and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 144–155.

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

Zhang, Yue and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562–571.