Improving Transition-Based Dependency Parsing with Buffer Transitions

Daniel Fernández-González
Departamento de Informática
Universidade de Vigo
Campus As Lagoas, 32004
Ourense, Spain
danifg@uvigo.es

Carlos Gómez-Rodríguez
Departamento de Computación
Universidade da Coruña
Campus de Elviña, 15071
A Coruña, Spain
carlos.gomez@udc.es

Abstract

In this paper, we show that significant improvements in the accuracy of well-known transition-based parsers can be obtained, without sacrificing efficiency, by enriching the parsers with simple transitions that act on buffer nodes.

First, we show how adding a specific transition to create either a left or right arc of length one between the first two buffer nodes produces improvements in the accuracy of Nivre's arc-eager projective parser on a number of datasets from the CoNLL-X shared task. Then, we show that accuracy can also be improved by adding transitions involving the topmost stack node and the second buffer node (allowing a limited form of non-projectivity).

None of these transitions has a negative impact on the computational complexity of the algorithm. Although the experiments in this paper use the arc-eager parser, the approach is generic enough to be applicable to any stack-based dependency parser.

1 Introduction

Dependency parsing has become a very active research area in natural language processing in recent years. The dependency representation of syntax simplifies the syntactic parsing task, since no non-lexical nodes need to be postulated by the parsers; while being convenient in practice, since dependency representations directly show the head-modifier and head-complement relationships which form the basis of predicate-argument structure. This has led to the development of various data-driven dependency parsers, such as those by Yamada and Matsumoto (2003), Nivre et al. (2004), McDonald et al. (2005), Martins et al. (2009), Huang and Sagae (2010) or Tratz and Hovy (2011), which can be trained directly from annotated data and produce accurate analyses very efficiently.

Most current data-driven dependency parsers can be classified into two families, commonly called **graph-based** and **transition-based** parsers (McDonald and Nivre, 2011). Graph-based parsers (Eisner, 1996; McDonald et al., 2005) are based on global optimization of models that work by scoring subtrees. On the other hand, transition-based parsers (Yamada and Matsumoto, 2003; Nivre et al., 2004), which are the focus of this work, use local training to make greedy decisions that deterministically select the next parser state. Among the advantages of transition-based parsers are the linear time complexity of many of them and the possibility of using rich feature models (Zhang and Nivre, 2011).

In particular, many transition-based parsers (Nivre et al., 2004; Attardi, 2006; Sagae and Tsujii, 2008; Nivre, 2009; Huang and Sagae, 2010; Gómez-Rodríguez and Nivre, 2010) are **stack-based** (Nivre, 2008), meaning that they keep a stack of partially processed tokens and an input buffer of unread tokens. In this paper, we show how the accuracy of this kind of parsers can be improved, without compromising efficiency, by extending their set of available transitions with **buffer transitions**. These are transitions that create a dependency arc involving some node in the buffer, which would typically be considered unavailable for linking by these algo-

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 308–319, Jeju Island, Korea, 12–14 July 2012.

rithms. The rationale is that buffer transitions construct some "easy" dependency arcs in advance, before the involved nodes reach the stack, so that the classifier's job when choosing among standard transitions is simplified.

To test the approach, we use the well-known arc-eager parser by (Nivre, 2003; Nivre et al., 2004) as a baseline, showing improvements in accuracy on most datasets of the CoNLL-X shared task (Buchholz and Marsi, 2006). However, the techniques discussed in this paper are generic and can also be applied to other stack-based dependency parsers.

The rest of this paper is structured as follows: Section 2 is an introduction to transition-based parsers and the arc-eager parsing algorithm. Section 3 presents the first novel contribution of this paper, **projective buffer transitions**, and discusses their empirical results on CoNLL-X datasets. Section 4 does the same for a more complex set of transitions, **non-projective buffer transitions**. Finally, Section 5 discusses related work and Section 6 sums up the conclusions and points out avenues for future work.

2 Preliminaries

We now briefly present some basic definitions for transition-based dependency parsing; a more thorough explanation can be found in (Nivre, 2008).

2.1 Dependency graphs

Let $w = w_1 \dots w_n$ be an input string. A **dependency graph** for w is a directed graph $G = (V_w, A)$; where $V_w = \{0, 1, \dots, n\}$ is a set of nodes, and $A \subseteq V_w \times L \times V_w$ is a set of labelled arcs. Each node in V_w encodes the position of a token in w, where 0 is a dummy node used as artificial root. An arc (i, l, j) will also be called a **dependency link** labelled l from i to j. We say that i is the syntactic **head** of j and, conversely, that j is a **dependent** of i. The **length** of the arc (i, l, j) is the value $|j - i|$.

Most dependency representations of syntax do not allow arbitrary dependency graphs. Instead, they require dependency graphs to be **forests**, i.e., acyclic graphs where each node has at most one head. In this paper, we will work with parsers that assume dependency graphs $G = (V_w, A)$ to satisfy the following properties:

- Single-head: every node has at most one incoming arc (if $(i, l, j) \in A$, then for every $k \neq i$, $(k, l', j) \notin A$).
- Acyclicity: there are no directed cycles in G.
- Node 0 is a root, i.e., there are no arcs of the form $(i, l, 0)$ in A.

A dependency forest with a single root (i.e., where all the nodes but one have at least one incoming arc) is called a **tree**. Every dependency forest can trivially be represented as a tree by adding arcs from the dummy root node 0 to every other root node.

For reasons of computational efficiency, many dependency parsers are restricted to work with forests satisfying an additional restriction called projectivity. A dependency forest is said to be **projective** if the set of nodes reachable by traversing zero or more arcs from any given node k corresponds to a continuous substring of the input (i.e., is an interval $\{x \in V_w \mid i \leq x \leq j\}$). For trees with a dummy root node at position 0, this is equivalent to not allowing dependency links to cross when drawn above the nodes (planarity).

2.2 Transition systems

A **transition system** is a nondeterministic state machine that maps input strings to dependency graphs. In this paper, we will focus on **stack-based transition systems**. A stack-based transition system is a quadruple $S = (C, T, c_s, C_t)$ where

- C is a set of parser **configurations**. Each configuration is of the form $c = (\sigma, \beta, A)$ where σ is a list of nodes of V_w called the **stack**, β is a list of nodes of V_w called the **buffer**, and A is a set of dependency arcs,
- T is a finite set of **transitions**, each of which is a partial function $t : C \rightarrow C$,
- c_s is an initialization function, mapping a sentence $w_1 \dots w_n$ to an **initial configuration** $c_s = ([0], [1, \dots, n], \emptyset)$,
- C_t is the set of **terminal configurations** $C_t = (\sigma, [], A) \in C$.

Transition systems are nondeterministic devices, since several transitions may be applicable to the same configuration. To obtain a deterministic parser

from a transition system, a classifier is trained to greedily select the best transition at each state. This training is typically done by using an **oracle**, which is a function $o : C \rightarrow T$ that selects a single transition at each configuration, given a tree in the training set. The classifier is then trained to approximate this oracle when the target tree is unknown.

2.3 The arc-eager parser

Nivre's arc-eager dependency parser (Nivre, 2003; Nivre et al., 2004) is one of the most widely known and used transition-based parsers (see for example (Zhang and Clark, 2008; Zhang and Nivre, 2011)). This parser works by reading the input sentence from left to right and creating dependency links as soon as possible. This means that links are created in a strict left-to-right order, and implies that while leftward links are built in a bottom-up fashion, a rightward link $a \rightarrow b$ will be created before the node b has collected its right dependents.

The arc-eager transition system has the following four transitions (note that, for convenience, we write a stack with node i on top as $\sigma|i$, and a buffer whose first node is i as $i|\beta$):

- SHIFT : $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$.
- REDUCE : $(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)$. Precondition: $\exists k, l' \mid (k, l', i) \in A$.
- LEFT-ARC$_l$: $(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, l, i)\})$. Preconditions: $i \neq 0$ and $\nexists k, l' \mid (k, l', i) \in A$ (single-head)
- RIGHT-ARC$_l$: $(\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i, l, j)\})$.

The SHIFT transition reads an input word by removing the first node from the buffer and placing it on top of the stack. The REDUCE transition pops the stack, and it can only be executed if the topmost stack node has already been assigned a head. The LEFT-ARC transition creates an arc from the first node in the buffer to the node on top of the stack, and then pops the stack. It can only be executed if the node on top of the stack does not already have a head. Finally, the RIGHT-ARC transition creates an arc from the top of the stack to the first buffer node, and then removes the latter from the buffer and moves it to the stack.

The arc-eager parser has linear time complexity. In principle, it is restricted to projective dependency forests, but it can be used in conjunction with the pseudo-projective transformation (Nivre et al., 2006) in order to capture a restricted subset of non-projective forests. Using this setup, it scored as one of the top two systems in the CoNLL-X shared task.

3 Projective buffer transitions

In this section, we show that the accuracy of stack-based transition systems can benefit from adding one of a pair of new transitions, which we call **projective buffer transitions**, to their transition sets.

3.1 The transitions

The two projective buffer transitions are defined as follows:

- LEFT-BUFFER-ARC$_l$: $(\sigma, i|j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, l, i)\})$.
- RIGHT-BUFFER-ARC$_l$: $(\sigma, i|j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, l, j)\})$.

The LEFT-BUFFER-ARC transition creates a leftward dependency link from the second node to the first node in the buffer, and then removes the first node from the buffer. Conversely, the RIGHT-BUFFER-ARC transition creates a rightward dependency link from the first node to the second node in the buffer, and then removes the second node. We call these transitions projective buffer transitions because, since they act on contiguous buffer nodes, they can only create projective arcs.

Adding one (or both) of these transitions to a projective or non-projective stack-based transition system does not affect its correctness, as long as this starting system cannot generate configurations (σ, β, A) where a buffer node has a head in A[1]: it cannot affect completeness because we are not removing existing transitions, and therefore any dependency graph that the original system could build

[1]Most stack-based transition systems in the literature disallow such configurations. However, in parsers that allow them (such as those defined by Gómez-Rodríguez and Nivre (2010)), projective buffer transitions can still be added without affecting correctness if we impose explicit single-head and acyclicity preconditions on them. We have not included these preconditions by default for simplicity of presentation.

will still be obtainable by the augmented one; and it cannot affect soundness (be it for projective dependency forests or for any superset of them) because the new transitions can only create projective arcs and cannot violate the single-head or acyclicity constraints, given that a buffer node cannot have a head.

The idea behind projective buffer transitions is to create dependency arcs of length one (i.e., arcs involving contiguous nodes) *in advance* of the standard arc-building transitions that need at least one of the nodes to get to the stack (LEFT-ARC and RIGHT-ARC in the case of the arc-eager transition system).

Our hypothesis is that, as it is known that short-distance dependencies are easier to learn for transition-based parsers than long-distance ones (McDonald and Nivre, 2007), handling these short arcs in advance and removing their dependent nodes will make it easier for the classifier to learn how to make decisions involving the standard arc transitions.

Note that the fact that projective buffer transitions create arcs of length 1 is not explicit in the definition of the transitions. For instance, if we add the LEFT-BUFFER-ARC$_l$ transition only to the arc-eager transition system, LEFT-BUFFER-ARC$_l$ will only be able to create arcs of length 1, since it is easy to see that the first two buffer nodes are contiguous in all the accessible configurations. However, if we add RIGHT-BUFFER-ARC$_l$, this transition will have the potential to create arcs of length greater than 1: for example, if two consecutive RIGHT-BUFFER-ARC$_l$ transitions are applied starting from a configuration $(\sigma, i|i+1|i+2|\beta, A)$, the second application will create an arc $i \rightarrow i+2$ of length 2.

Although we could have added the length-1 restriction to the transition definitions, we have chosen the more generic approach of leaving it to the oracle instead. While the oracle typically used for the arc-eager system follows the simple principle of executing transitions that create an arc as soon as it has the chance to, adding projective buffer transitions opens up new possibilities: we may now have several ways of creating an arc, and we have to decide in which cases we train the parser to use one of the buffer transitions and in which cases we prefer to train it to ignore the buffer transitions and delegate to the standard ones. Following the hypothesis explained above, our policy has been to train the parser to use buffer transitions whenever possible for arcs of length one, and to not use them for arcs of length larger than one. To test this idea, we also conducted experiments with the alternative policy "use buffer transitions whenever possible, regardless of arc length": as expected, the obtained accuracies were (slightly) worse.

The chosen oracle policy is generic and can be plugged into any stack-based parser: for a given transition, first check whether it is possible to build a gold-standard arc of length 1 with a projective buffer transition.[2] If so, choose that transition, and if not, just delegate to the original parser's oracle.

3.2 Experiments

To empirically evaluate the effect of projective buffer transitions on parsing accuracy, we have conducted experiments on eight datasets of the CoNLL-X shared task (Buchholz and Marsi, 2006): Arabic (Hajič et al., 2004), Chinese (Chen et al., 2003), Czech (Hajič et al., 2006), Danish (Kromann, 2003), German (Brants et al., 2002), Portuguese (Afonso et al., 2002), Swedish (Nilsson et al., 2005) and Turkish (Oflazer et al., 2003; Atalay et al., 2003).

As our baseline parser, we use the arc-eager projective transition system by Nivre (2003). Table 1 compares the accuracy obtained by this system alone with that obtained when the LEFT-BUFFER-ARC and RIGHT-BUFFER-ARC transitions are added to it as explained in Section 3.1.

Accuracy is reported in terms of labelled (LAS) and unlabelled (UAS) attachment score. We used SVM classifiers from the LIBSVM package (Chang and Lin, 2001) for all languages except for Chinese, Czech and German. In these, we used the LIBLINEAR package (Fan et al., 2008) for classification, since it reduces training time in these larger datasets. Feature models for all parsers were specifically tuned for each language.[3]

[2] In this context, "possible" means that we can create the arc without losing the possibility of creating other gold-standard arcs. In the case of RIGHT-BUFFER-ARC, this involves checking that the candidate dependent node has no dependents in the gold-standard tree (if it has any, we cannot remove it from the stack or it would not be able to collect its dependents, so we do not use the buffer transition).

[3] All the experimental settings and feature models used are included in the supplementary material and also available at `http://www.grupolys.org/~cgomezr/exp/`.

	NE		NE+LBA		NE+RBA	
Language	LAS	UAS	LAS	UAS	LAS	UAS
Arabic	66.43	77.19	**67.78**	**78.26**	63.87	74.63
Chinese	86.46	90.18	82.47	86.14	**86.62**	**90.64**
Czech	77.24	83.40	**78.70**	**84.24**	78.28	83.94
Danish	84.91	89.80	**85.21**	**90.20**	82.53	87.35
German	86.18	88.60	84.31	86.50	**86.48**	**88.90**
Portug.	86.60	90.20	**86.92**	**90.58**	85.55	89.28
Swedish	**83.33**	**88.83**	82.81	88.03	81.66	88.03
Turkish	63.77	74.35	57.42	66.24	**64.33**	**74.73**

Table 1: Parsing accuracy (LAS and UAS, excluding punctuation) of Nivre's arc-eager parser without modification (NE), with the LEFT-BUFFER-ARC transition added (NE+LBA) and with the RIGHT-BUFFER-ARC transition added (NE+RBA). Best results for each language are shown in boldface.

As can be seen in Table 1, adding a projective buffer transition improves the performance of the parser in seven out of the eight tested languages. The improvements in LAS are statistically significant at the .01 level[4] in the Arabic and Czech treebanks.

Note that the decision of *which* buffer transition to add strongly depends on the dataset. In the majority of the treebanks, we can see that when the LEFT-BUFFER-ARC transition improves performance the RIGHT-BUFFER-ARC transition harms it, and vice versa. The exceptions are Czech, where both transitions are beneficial, and Swedish, where both are harmful. Therefore, when using projective buffer transitions in practice, the language and annotation scheme should be taken into account (or tests should be made) to decide which one to use.

Table 2 hints at the reason for this treebank-sensitiveness. By analyzing the relative frequency of leftward and rightward dependency links (and, in particular, of leftward and rightward links of length 1) in the different treebanks, we see a reasonably clear tendency: the LEFT-BUFFER-ARC transition works better in treebanks that contain a large proportion of *rightward* arcs of length 1, and the RIGHT-BUFFER-ARC transition works better in treebanks with a large proportion of *leftward* arcs of length 1. Note that, while this might seem counterintuitive at a first glance, it is coherent with the hypothesis that we formulated in Section 3.1: the advantage of projective buffer transitions is not that they build arcs more accurately than standard arc-building transitions (in fact the opposite might be expected, since they work on nodes while they are still on the buffer and we have less information about their surrounding nodes in our feature models), but that they make it easier for the classifier to decide among standard transitions. The analysis on Table 2 agrees with that explanation: LEFT-BUFFER-ARC improves performance in treebanks where it is not used too often but it can filter out leftward arcs of length 1, making it easier for the parser to be accurate on rightward arcs of length 1; and the converse happens for RIGHT-BUFFER-ARC.

Language	L%	R%	L1%	R1%	Best PBT
Arabic	12.3	87.7	6.5	55.1	LBA
Chinese	58.4	41.6	35.8	15.1	RBA
Czech	41.4	58.6	22.1	24.9	LBA*
Danish	17.1	82.9	10.9	43.0	LBA
German	39.8	60.2	20.3	19.9	RBA
Portug.	32.6	67.4	22.5	26.9	LBA
Swedish	38.2	61.8	24.1	21.8	LBA*
Turkish	77.8	22.2	47.2	10.4	RBA

Table 2: Analysis of the datasets used in the experiments in terms of: percentage of leftward and rightward links (L%, R%), percentage of leftward and rightward links of length 1 (L1%, R1%), and which projective buffer transition works better for each dataset according to the results in Table 1 (LBA = LEFT-BUFFER-ARC, RBA = RIGHT-BUFFER-ARC). Languages where both transitions are beneficial (Czech) or harmful (Swedish) are marked with an asterisk.

[4]Statistical significance was assessed using Dan Bikel's randomized parsing evaluation comparator: `http://www.cis.upenn.edu/~dbikel/software.html#comparator`

	NE		NE+LBA			NE+RBA			NE+LBA+RBA			
Language	LA	RA	LA*	RA	LBA	LA	RA*	RBA	LA*	RA*	LBA	RBA
Arabic	**58.28**	67.77	42.61	**68.65**	**77.46**	55.88	60.63	**79.70**	37.40	62.28	66.78	75.94
Chinese	85.69	**85.79**	80.92	84.19	**89.00**	**85.96**	84.77	**88.01**	81.08	79.46	87.72	86.33
Czech	85.73	76.44	80.79	**78.34**	**91.07**	**86.25**	76.62	**82.58**	79.49	75.98	90.26	81.97
Danish	89.47	83.92	88.65	**84.16**	**91.72**	86.27	78.04	**92.30**	**90.23**	77.52	88.79	92.10
German	89.15	87.11	83.75	**87.23**	**94.30**	**89.55**	84.38	**95.98**	79.26	81.60	91.66	90.73
Portuguese	**94.77**	84.91	90.83	**85.11**	**97.07**	93.84	81.86	**92.29**	88.72	79.86	96.02	89.26
Swedish	**87.75**	80.74	84.62	**81.30**	**92.83**	87.12	74.77	**90.73**	78.10	72.50	90.86	89.89
Turkish	59.68	**74.21**	53.02	74.01	**72.78**	**60.23**	69.23	**73.91**	49.34	48.48	65.57	41.94

Table 3: Labelled precision of the arcs built by each transition of Nivre's arc-eager parser without modification (NE), with a projective buffer transition added (NE+LBA, NE+RBA) and with both projective buffer transitions added (NE+LBA+RBA). We mark a standard LEFT-ARC (LA) or RIGHT-ARC (LA) transition with an asterisk (LA*, RA*) when it is acting only on a "hard" subset of leftward (rightward) arcs, and thus its precision is not directly comparable to that of (LA, RA). Best results for each language and transition are shown in boldface.

To further test this idea, we computed the labelled precision of each individual transition of the parsers with and without projective buffer transitions, as shown in Table 3. As we can see, projective buffer transitions achieve better precision than standard transitions, but this is not surprising since they act only on "easy" arcs of length 1. Therefore, this high precision does not mean that they actually build arcs more accurately than the standard transitions, since it is not measured on the same set of arcs. Similarly, adding a projective buffer transition decreases the precision of its corresponding standard transition, but this is because the standard transition is then dealing only with "harder" arcs of length greather than 1, not because it is making more errors. A more interesting insight comes from comparing transitions that are acting on the same target set of arcs: we see that, in the languages where LEFT-BUFFER-ARC is beneficial, the addition of this transition always improves the precision of the standard RIGHT-ARC transition; and the converse happens with RIGHT-BUFFER-ARC with respect to LEFT-ARC. This further backs the hypothesis that the filtering of "easy" links achieved by projective buffer transitions makes it easier for the classifier to decide among standard transitions.

We also conducted experiments adding both transitions at the same time (NE+LBA+RBA), but the results were worse than adding the suitable transition for each dataset. Table 3 hints at the reason: the precision of buffer transitions noticeably decreases when both of them are added at the same time, presumably because it is difficult for the classifier to decide between both with the restricted feature information available for buffer nodes.

	NE+LBA/RBA		NE+PP (CoNLL X)	
Language	LAS	UAS	LAS	UAS
Arabic	**67.78**	**78.26**	66.71	77.52
Chinese	86.62	**90.64**	**86.92**	90.54
Czech	**78.70**	84.24	78.42	**84.80**
Danish	**85.21**	**90.20**	84.77	89.80
German	**86.48**	**88.90**	85.82	88.76
Portug.	86.92	90.58	**87.60**	**91.22**
Swedish	82.81	88.03	**84.58**	**89.50**
Turkish	64.33	74.73	**65.68**	**75.82**

Table 4: Comparison of the parsing accuracy (LAS and UAS, excluding punctuation) of Nivre's arc-eager parser with projective buffer transitions (NE+LBA/RBA) and the parser with the pseudo-projective transformation (Nivre et al., 2006)

To further put the obtained results into context, Table 4 compares the performance of the arc-eager parser with the projective buffer transition most suitable for each dataset with the results obtained by the parser with the pseudo-projective transformation by Nivre et al. (2006) in the CoNLL-X shared task, one of the top two performing systems in that event. The reader should be aware that the purpose of this table is only to provide a broad idea of how our approach performs with respect to a well-known reference point, and not to make a detailed comparison, since the two parsers have not been tuned in homogeneous conditions: on the one hand, we had access to the CoNLL-X test sets which were unavailable

System	Arabic	Danish
Nivre et al. (2006)	66.71	84.77
McDonald et al. (2006)	66.91	84.79
Nivre (2009)	67.3	84.7
Gómez-Rodríguez and Nivre (2010)	N/A	83.81
NE+LBA/RBA	**67.78**	**85.21**

Table 5: Comparison of the Arabic and Danish LAS obtained by the arc-eager parser with projective buffer transitions in comparison to other parsers in the literature that report results on these datasets.

for the participants in the shared task; on the other hand, we did not fine-tune the classifier parameters for each dataset like Nivre et al. (2006), but used default values for all languages.

As can be seen in the table, even though the pseudo-projective parser is able to capture non-projective syntactic phenomena, the algorithm with projective buffer transitions (which is strictly projective) outperforms it in four of the eight treebanks, including non-projective treebanks such as the German one.

Furthermore, to our knowledge, our LAS results for Arabic and Danish are currently the best published results for a single-parser system on these datasets, not only outperforming the systems participating in CoNLL-X but also other parsers tested on these treebanks in more recent years (see Table 5).

Finally, it is worth noting that adding projective buffer transitions has no negative impact on efficiency, either in terms of computational complexity or of empirical runtime. Since each projective buffer transition removes a node from the buffer, no more than n such transitions can be executed for a sentence of length n, so adding these transitions cannot increase the complexity of a transition-based parser. In the particular case of the arc-eager parser, using projective buffer transitions reduces the average number of transitions needed to obtain a given dependency forest, as some nodes can be dispatched by a single transition rather than being shifted and later popped from the stack. In practice, we observed that the training and parsing times of the arc-eager parser with projective buffer transitions were slightly faster than without them on the Arabic, Chinese, Swedish and Turkish treebanks, and slightly slower than without them on the other four treebanks, so adding these transitions does not seem to noticeably degrade (or improve) practical efficiency.

4 Non-projective buffer transitions

We now present a second set of transitions that still follow the idea of early processing of some dependency arcs, as in Section 3; but which are able to create arcs skipping over a buffer node, so that they can create some non-projective arcs. For this reason, we call them **non-projective buffer transitions**.

4.1 The transitions

The two non-projective buffer transitions are defined as follows:

- LEFT-NONPROJ-BUFFER-ARC$_l$:
  $(\sigma|i, j|k|\beta, A) \Rightarrow (\sigma, j|k|\beta, A \cup \{(k, l, i)\})$.
  Preconditions: $i \neq 0$ and $\nexists m, l' \mid (m, l', i) \in A$ (single-head)

- RIGHT-NONPROJ-BUFFER-ARC$_l$:
  $(\sigma|i, j|k|\beta, A) \Rightarrow (\sigma|i, j|\beta, A \cup \{(i, l, k)\})$.

The LEFT-NONPROJ-BUFFER-ARC transition creates a leftward arc from the second buffer node to the node on top of the stack, and then pops the stack. It can only be executed if the node on top of the stack does not already have a head. The RIGHT-NONPROJ-BUFFER-ARC transition creates an arc from the top of the stack to the second node in the buffer, and then removes the latter from the buffer. Note that these transitions are analogous to projective buffer transitions, and they use the second node in the buffer in the same way, but they create arcs involving the node on top of the stack rather than the first buffer node. This change makes the precondition that checks for a head necessary for the transition LEFT-NONPROJ-BUFFER-ARC to respect the single-head constraint, since many stack-based parsers can generate configurations where the node on top of the stack has a head.

We call these transitions non-projective buffer transitions because, as they act on non-contiguous nodes in the stack and buffer, they allow the creation of a limited set of non-projective dependency arcs. This means that, when added to a projective parser, they will increase its coverage.[5] On the other hand,

[5]They may also increase the coverage of parsers allowing restricted forms of non-projectivity, but that depends on the par-

	NE		NE+LNBA		NE+RNBA	
Language	LAS	UAS	LAS	UAS	LAS	UAS
Arabic	66.43	77.19	67.13	77.90	**67.21**	**77.92**
Chinese	86.46	90.18	**87.71**	**91.39**	86.98	90.76
Czech	77.24	83.40	**78.88**	**84.72**	78.12	83.78
Danish	84.91	89.80	**85.17**	**90.10**	84.25	88.92
German	86.18	88.60	**86.96**	**88.98**	85.56	88.30
Portug.	86.60	90.20	**86.78**	**90.34**	86.07	89.92
Swedish	83.33	88.83	**83.55**	**89.30**	83.17	88.59
Turkish	63.77	74.35	63.04	73.99	**65.01**	**75.70**

Table 6: Parsing accuracy (LAS and UAS, excluding punctuation) of Nivre's arc-eager parser without modification (NE), with the LEFT-NONPROJ-BUFFER-ARC transition added (NE+LNBA) and with the RIGHT-NONPROJ-BUFFER-ARC transition added (NE+RNBA). Best results for each language are shown in boldface.

adding these transitions to a stack-based transition system does not affect soundness under the same conditions and for the same reasons explained for projective buffer transitions in Section 3.1.

Note that the fact that non-projective buffer transitions are able to create non-projective dependency arcs does not mean that *all* the arcs that they build are non-projective, since an arc on non-contiguous nodes in the stack and buffer may or may not cross other arcs. This means that non-projective buffer transitions serve a dual purpose: not only they increase coverage, but they also can create some "easy" dependency links in advance of standard transitions, just like projective buffer transitions.

Contrary to projective buffer transitions, we do not impose any arc length restrictions on non-projective buffer transitions (either as a hard constraint in the transitions themselves or as a policy in the training oracle), since we would like the increase in coverage to be as large as possible. We wish to allow the parsers to create non-projective arcs in a straightforward way and without compromising efficiency. Therefore, to train the parser with these transitions, we use an oracle that employs them whenever possible, and delegates to the original parser's oracle otherwise.

4.2 Experiments

We evaluate the impact of non-projective buffer transitions on parsing accuracy by using the same baseline parser, datasets and experimental settings as for projective buffer transitions in Section 3.2. As can be seen in Table 6, adding a non-projective buffer transition to the arc-eager parser improves its performance on all eight datasets. The improvements in LAS are statistically significant at the .01 level (Dan Bikel's comparator) for Chinese, Czech and Turkish. Note that the Chinese treebank is fully projective, this means that non-projective buffer transitions are also beneficial when creating projective arcs.

While with projective buffer transitions we observed that each of them was beneficial for about half of the treebanks, and we related this to the amount of leftward and rightward links of length 1 in each; in the case of non-projective buffer transitions we do not observe this tendency. In this case, LEFT-NONPROJ-BUFFER-ARC works better than RIGHT-NONPROJ-BUFFER-ARC in all datasets except for Turkish and Arabic.

As with the projective transitions, we gathered data about the individual precision of each of the transitions. The results were similar to those for the projective transitions, and show that adding a non-projective buffer transition improves the precision of the standard transitions. We also experimentally checked that adding both non-projective buffer transitions at the same time (NE+LNBA+RNBA) achieved worse performance than adding only the most suitable transition for each dataset.

Table 7 compares the performance of the arc-eager parser with the best non-projective buffer transition for each dataset with the results obtained by

ticular subset of non-projective structures captured by each such parser.

	NE+LNBA/RNBA		NE+PP (CoNLL X)	
Language	LAS	UAS	LAS	UAS
Arabic	**67.21**	**77.92**	66.71	77.52
Chinese	**87.71**	**91.39**	86.92	90.54
Czech	**78.88**	84.72	78.42	**84.80**
Danish	**85.09**	**89.98**	84.77	89.80
German	**86.96**	**88.98**	85.82	88.76
Portug.	86.78	90.34	**87.60**	**91.22**
Swedish	83.55	89.30	**84.58**	**89.50**
Turkish	65.01	75.70	**65.68**	**75.82**

Table 7: Comparison of the parsing accuracy (LAS and UAS, excluding punctuation) of Nivre's arc-eager parser with non-projective buffer transitions (NE+LNBA/RNBA) and the parser with the pseudo-projective transformation (Nivre et al., 2006).

System	PP	PR	NP	NR
NE	80.40	80.76	-	-
NE+LNBA/RNBA	**80.96**	**81.33**	**58.87**	15.66
NE+PP (CoNLL-X)	80.71	81.00	50.72	**29.57**

Table 8: Comparison of the precision and recall for projective (PP, PR) and non-projective (NP, NR) arcs, averaged over all datasets, obtained by Nivre's arc-eager parser with and without non-projective buffer transitions (NE+LNBA/RNBA, NE) and the parser with the pseudo-projective transformation (Nivre et al., 2006).

the parser with the pseudo-projective transformation by Nivre et al. (2006) in the CoNLL-X shared task. Note that, like the one in Table 4, this should not be interpreted as a homogeneous comparison. We can see that the algorithm with non-projective buffer transitions obtains better LAS in five out of the eight treebanks. Precision and recall data on projective and non-projective arcs (Table 8) show that, while our parser does not capture as many non-projective arcs as the pseudo-projective transformation (unsurprisingly, as it can only build non-projective arcs in one direction: that of the particular non-projective buffer transition used for each dataset); it does so with greater precision and is more accurate than that algorithm in projective arcs.

Like projective buffer transitions, non-projective transitions do not increase the computational complexity of stack-based parsers. The observed training and parsing times for the arc-eager parser with non-projective buffer transitions showed a small overhead with respect to the original arc-eager (7.1% average increase in training time, 17.0% in parsing time). For comparison, running the arc-eager parser with the pseudo-projective transformation (Nivre et al., 2006) on the same machine produced a 23.5% increase in training time and a 87.5% increase in parsing time.

5 Related work

The approach of adding an extra transition to a parser to improve its accuracy has been applied in the past by Choi and Palmer (2011). In that paper, the LEFT-ARC transition from Nivre's arc-eager transition system is added to a list-based parser. However, the goal of that transition is different from ours (selecting between projective and non-projective parsing, rather than building some arcs in advance) and the approach is specific to one algorithm while ours is generic – for example, the LEFT-ARC transition cannot be added to the arc-standard and arc-eager parsers, or to extensions of those like the ones by Attardi (2006) or Nivre (2009), because these already have it.

The idea of creating dependency arcs of length 1 in advance to help the classifier has been used by Cheng et al. (2006). However, their system creates such arcs in a separate preprocessing step rather than dynamically by adding a transition to the parser, and our approach obtains better LAS and UAS results on all the tested datasets.

The projective buffer transitions presented here bear some resemblance to the easy-first parser by Goldberg and Elhadad (2010), which allows creation of dependency arcs between any pair of contiguous nodes and is based on the idea of "easy" dependency links being created first. However, while the easy-first parser is an entirely new $O(n \log(n))$ algorithm, our approach is a generic extension for stack-based parsers that does not increase their complexity (so, for example, applying it to the arc-eager system as in the experiments in this paper yields $O(n)$ complexity).

Non-projective transitions that create dependency arcs between non-contiguous nodes have been used in the transition-based parser by Attardi (2006). However, the transitions in that parser do not use the second buffer node, since they are not intended

to create some arcs in advance. The non-projective buffer transitions presented in this paper can also be added to Attardi's parser.

6 Discussion

We have presented a set of two transitions, called *projective buffer transitions*, and showed that adding one of them to Nivre's arc-eager parser improves its accuracy in seven out of eight tested datasets from the CoNLL-X shared task. Furthermore, adding one of a set of *non-projective buffer transitions* achieves accuracy improvements in all of the eight datasets. The obtained improvements are statistically significant for several of the treebanks, and the parser with projective buffer transitions surpassed the best published single-parser LAS results on two of them. This comes at no cost either on computational complexity or (in the case of projective transitions) on empirical training and parsing times with respect to the original parser.

While we have chosen Nivre's well-known arc-eager parser as our baseline, we have shown that these transitions can be added to any stack-based dependency parser, and we are not aware of any specific property of arc-eager that would make them work better in practice on this parser than on others. Therefore, future work will include an evaluation of the impact of buffer transitions on more transition-based parsers. Other research directions involve investigating the set of non-projective arcs allowed by non-projective buffer transitions, defining different variants of buffer transitions (such as non-projective buffer transitions that work with nodes located deeper in the buffer) or using projective and non-projective buffer transitions at the same time.

Acknowledgments

This research has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (projects TIN2010-18552-C03-01 and TIN2010-18552-C03-02), Ministry of Education (FPU Grant Program) and Xunta de Galicia (Rede Galega de Recursos Lingüísticos para unha Sociedade do Coñecemento).

References

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. 2002. "Floresta sintá(c)tica": a treebank for Portuguese. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1968–1703, Paris, France. ELRA.

Nart B. Atalay, Kemal Oflazer, and Bilge Say. 2003. The annotation process in the Turkish treebank. In *Proceedings of EACL Workshop on Linguistically Interpreted Corpora (LINC-03)*, pages 243–246, Morristown, NJ, USA. Association for Computational Linguistics.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The tiger treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories, September 20-21*, Sozopol, Bulgaria.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

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

K. Chen, C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, and Z. Gao. 2003. Sinica treebank: Design criteria, representational issues and implementation. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, chapter 13, pages 231–248. Kluwer.

Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2006. Multi-lingual dependency parsing at NAIST. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 191–195, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jinho D. Choi and Martha Palmer. 2011. Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pages 687–692, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Yoav Goldberg 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 (NAACL HLT)*, pages 742–750.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1492–1501, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jan Hajič, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. 2004. Prague Arabic Dependency Treebank: Development in data and tools. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*.

Jan Hajič, Jarmila Panevová, Eva Hajičová, Jarmila Panevová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, and Marie Mikulová. 2006. Prague Dependency Treebank 2.0. CDROM CAT: LDC2006T01, ISBN 1-58563-370-4. Linguistic Data Consortium.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1077–1086, Stroudsburg, PA, USA. Association for Computational Linguistics.

Matthias T. Kromann. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 217–220, Växjö, Sweden. Växjö University Press.

Andre Martins, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. 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 (ACL-IJCNLP)*, pages 342–350.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Comput. Linguist.*, 37:197–230.

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

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 216–220.

Jens Nilsson, Johan Hall, and Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from Antiquity. In Peter Juel Henrichsen, editor, *Proceedings of the NODALIDA Special Session on Treebanks*.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 49–56, Morristown, NJ, USA. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pages 149–160. ACL/SIGPARSE.

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

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. 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 (ACL-IJCNLP)*, pages 351–359.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish treebank. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, pages 261–277. Kluwer.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 753–760.

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

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines.

In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

Yue Zhang 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.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pages 188–193, Stroudsburg, PA, USA. Association for Computational Linguistics.