Effective Online Reordering with Arc-Eager Transitions

Ryosuke Kohita **Hiroshi Noji** **Yuji Matsumoto**
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
{kohita.ryosuke.kj9, noji, matsu}@is.naist.jp

Abstract

We present a new transition system with word reordering for unrestricted non-projective dependency parsing. Our system is based on decomposed arc-eager rather than arc-standard, which allows more flexible ambiguity resolution between a local projective and non-local crossing attachment. In our experiment on Universal Dependencies 2.0, we find our parser outperforms the ordinary swap-based parser particularly on languages with a large amount of non-projectivity.

1 Introduction

A dependency tree as illustrated in Figure 1 is called a *non-projective* tree, which contains discontinuous subtrees and is informally remarked with crossing arcs (arcs from *idea*$_4$ to *talking*$_8$ and from *who*$_5$ to *to*$_9$). Comparing to the class of projective trees, which has a weak equivalence to the context-free grammars (Gaifman, 1965), parsing non-projective trees is generally involved. This is particularly the case for *transition-based* dependency parsing; contrary to the graph-based approaches, in which a simple spanning-tree algorithm is capable of handling them (McDonald et al., 2005), due to the incremental nature, transition-based parsers need some extra mechanisms to find crossing arcs.

There are several attempts to handle crossing arcs in transition-based parsers. Among them online reordering with swap (Nivre, 2009) has a number of appealing properties, of which the most notable is that it inherits the standard architecture of the transition systems using a stack and buffer while covering all types of crossing arcs. This simplicity allows us to incorporate the ideas developed for the standard projective parsers, such as neural network architectures (Chen and Manning, 2014; Dyer et al., 2015), and joint modeling with other phenomena (Hatori et al., 2011; Honnibal and Johnson, 2014), with a minimal effort. Such extensions with swap include a recent non-projective neural parser (Straka et al., 2015) and joint system with POS tagging (Bohnet and Nivre, 2012). Other approaches often employ additional data structures with non-trivial transitions (Covington, 2001; Choi and McCallum, 2013; Pitler and McDonald, 2015), which interfere with the transparency to the standard systems, or cannot handle all crossing arcs (Attardi, 2006).

Despite the popularity of the swap system, to our knowledge there is little work focusing on the swap mechanism, or the transition system itself, apart from the original proposal (Nivre, 2009; Nivre et al., 2009). In other words, we are still unsure whether the current swap mechanism is the best strategy for handling crossing arcs with word reordering.

In this work, we present a dependency parser with a new transition system that employs swap-based reordering but in a different manner from the existing one (Nivre, 2009) built on the arc-standard system. As we discuss (Section 2.2), in Nivre's transition system, choosing a correct swap transition is sometimes hard due to the parser's preference to local attachments. The proposed system (Section 3) alleviates this difficulty by allowing a swap transition for a token that is already linked on the stack. As we will see, it can be seen as an extension to the arc-eager system (Nivre, 2003) while we divide each attachment transition into two more primitive operations as in the divided formulation of Gómez-Rodríguez and Nivre (2013). The divided system is more flexible, and by operating swap on this we can deal with the issue of reordering at an appropriate step.

On this transition system we implement a parsing model with the stack LSTMs (Dyer et al., 2015) (Section 4). We extensively examine the utility of new transition system (Section 5) with the recently released Universal Dependencies (UD) 2.0 dataset, which contains more than 60 treebanks with varying degree of non-projectivity, and find that our system is superior to the ordinary swap system particularly for languages with a larger amount of non-projectivity.

2 Background

We first introduce some notations and the concept of transition systems (Section 2.1), and then describe the existing swap-based transition system of Nivre (2009) (Section 2.2).

2.1 Transition System

We focus in this paper on a standard transition system operating on a triple called a *configuration* $c = (\sigma, \beta, A)$, where σ is a stack, β is a buffer, and A is a set of labeled arcs. See Nivre (2008) for the other variants and overview. In a configuration i-th token in a sentence is denoted by its index i while 0 denotes the special root token. Following the standard notations, by $\sigma|i$ and $j|\beta$, we mean i and j are the top-most tokens of the stack and buffer, respectively. We use $i \xrightarrow{l} j$ or (i, l, j) to denote an arc from i to j with label l.

Given a sentence of length n, the system begins parsing with the initial configuration $c_0 = ([0], [1, 2, 3, ...n], \phi)$ where only the root token is on the stack, all inputs are on the buffer, and the set of arcs is empty. Parsing finishes when it reaches a *terminal configuration*, in which any transitions cannot be performed, and the set of arcs A defines a labeled dependency tree. The system continues to make a transition decision at each step under the current configuration until it reaches a terminal configuration.

2.2 Arc-Standard Swap

Arc-standard swap system (ASS) (Nivre, 2009) is the most popular transition system for non-projective dependency parsing, which can handle arbitrary crossing arcs. ASS rows in Table 1 show the set of transitions, in which LA, RA, and SH are the transitions of the arc-standard system, which can only produce a projective tree by linking two adjacent tokens on the top of the stack. Swap (SW) is the key transition to support non-projectivity, which reorders the top two tokens on the stack by moving the second top token back to the buffer. Reducing a reordered token by LA or RA means we create subtrees that are non-adjacent with each other, i.e., crossing arcs.

One potential issue in ASS is its tendency to prefer local attachments due to the mechanism of LA and RA, which at the same time reduce the dependent token. This is problematic in that because crossing arcs often involve a longer dependency arc, if two tokens on the stack are locally likely to be connected, choosing correct SW rather than LA and RA is quite difficult.

To see an example, let us consider the configuration in Figure 2 ($c = (\sigma|\mathit{who}|\mathit{talking}, \mathit{to}|\beta, A)$), which occurs when parsing the sentence in Figure 1. The correct action here is SW to create a crossing arc $\mathit{talking}_8 \xrightarrow{\mathit{obl}} \mathit{who}_5$. However, at this point LA is a more likely transition since $\mathit{talking}_8 \xrightarrow{\mathit{obl}} \mathit{who}_5$ is a typical arc in a relative clause. The problem is that since LA reduces *talking*, we will miss the arc $\mathit{who}_5 \xrightarrow{\mathit{case}} \mathit{to}_9$ if we choose LA rather than SW.

3 New System: Stay-Eager Swap

Now we describe our proposed transition system, which we call **Stay-Eager Swap** (SES). We first present the transitions and its advantage (Section 3.1), and then discuss oracles (Section 3.2) and some improvements (Section 3.3).

3.1 Transition System

SES rows in Table 1 show the set of transitions of the new system. To understand the mechanism, we first note that without SW, this system looks very much similar to the arc-eager transition system (Nivre, 2003). The main difference from the original one is in the attaching transitions, which we call STAY-LEFT (SL) and STAY-RIGHT (SR) and do not reduce a dependent token, but just establish an arc between two tokens on the stack top and buffer top. Specifically, this system is identical to the divided arc-eager transition system with the primitive operations in Gómez-Rodríguez and Nivre (2013); in the arc-eager system, LEFT-ARC first builds an arc and then reduces the top of the stack, i.e., it is a combination of SL → RD (reduce) in our system, while RIGHT-ARC builds an arc and shift the top token of the buffer, i.e., it can be seen as SR → SH.[1]

[1] To make this system without SW to fully mimic the original arc-eager system, we need a constraint to prohibit

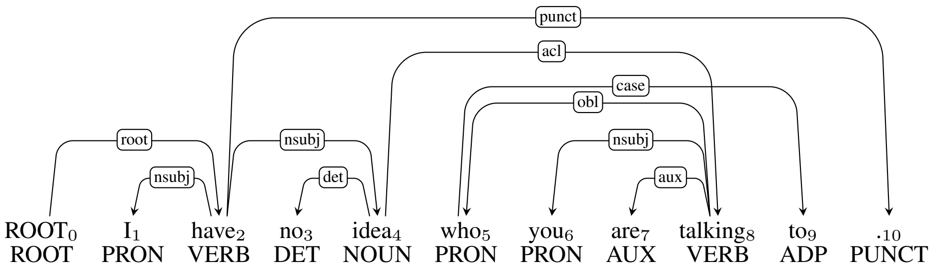

Figure 1: A non-projective sentence.

	Transition	Current configuration	⇒	Resulting configuration	Condition
ASS	LEFT-ARC$_l$ (LA)	$(\sigma\|i\|j, \quad \beta, \quad A)$	⇒	$(\sigma\|j, \quad \beta, \quad A \cup \{(j,l,i)\})$	$i \neq 0$
	RIGHT-ARC$_l$ (RA)	$(\sigma\|i\|j, \quad \beta, \quad A)$	⇒	$(\sigma\|i, \quad \beta, \quad A \cup \{(i,\cdot,j)\})$	
	SHIFT (SH)	$(\sigma, \quad i\|\beta, \quad A)$	⇒	$(\sigma\|i, \quad \beta, \quad A)$	
	SWAP (SW)	$(\sigma\|i\|j, \quad \beta, \quad A)$	⇒	$(\sigma\|j, \quad i\|\beta, \quad A)$	$0 < i < j$
SES	STAY-LEFT$_l$ (SL)	$(\sigma\|i, \quad j\|\beta, \quad A)$	⇒	$(\sigma\|i, \quad j\|\beta, \quad A \cup \{(j,l,i)\})$	$i \neq 0 \wedge (\cdot,\cdot,i) \notin A \wedge i \xrightarrow{*} j \notin A$
	STAY-RIGHT$_l$ (SR)	$(\sigma\|i, \quad j\|\beta, \quad A)$	⇒	$(\sigma\|i, \quad j\|\beta, \quad A \cup \{(i,l,j)\})$	$(\cdot,\cdot,j) \notin A \wedge j \xrightarrow{*} i \notin A$
	SHIFT (SH)	$(\sigma, \quad i\|\beta, \quad A)$	⇒	$(\sigma\|i, \quad \beta, \quad A)$	
	REDUCE (RD)	$(\sigma\|i, \quad \beta, \quad A)$	⇒	$(\sigma, \quad \beta, \quad A)$	$(\cdot,\cdot,i) \in A$
	SWAP (SW)	$(\sigma\|i, \quad j\|\beta, \quad A)$	⇒	$(\sigma, \quad j\|i\|\beta, \quad A)$	$0 < i < j$
AUX	UNSHIFT (UN)	$(\sigma\|i, \quad [], \quad A)$	⇒	$(\sigma, \quad [i], \quad A)$	$(\cdot,\cdot,i) \notin A$
	RIGHT-ROOT (RR)	$([0,i] \quad [], \quad A)$	⇒	$([0], \quad [], \quad A \cup \{(0,root,i)\})$	$(\cdot,\cdot,i) \notin A \wedge (0,root,i) \notin A$

Table 1: The set of transitions for arc-standard swap (ASS) and stay-eager swap (SES). $\cdot$ in conditions means an arbitrary value, e.g., $(\cdot,\cdot,i) \notin A$ means $\forall j.\forall h.(j,h,i) \notin A$ (i's head is unspecified). $i \xrightarrow{*} j \notin A$ means a path from i to j does not exist, which is needed to guarantee acyclicity.

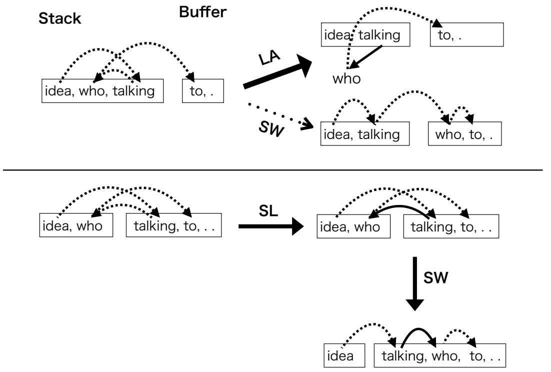

Figure 2: A configuration difficult for ASS (above), which fails when LA is selected (SW is correct). Our system avoides this difficulty by first attaching *who* to *talking* (SL) and then SW (below). Dotted arcs are correct arcs yet unattached.

We allow SW at an arbitrary point. This means we can insert SW just after SL and SR, by which we can alleviate the difficulty with an attachment vs. swap transitions discussed in Section 2.2. Figure 2 shows how our system can swap after resolving local projective attachments.

3.2 Static and Non-static Oracles

An oracle for a transition system is a function from a configuration to the action that leads to a given dependency tree. Before discussing in details, we first note that our system suffers from the spurious ambiguity as in the arc-eager system (Goldberg and Nivre, 2012), which means an oracle for some configurations is not unique.

Static oracle Table 2 shows a specific oracle, which checks for each action in descending order whether the current configuration satisfies the condition, and select the first found one. This is a *static* oracle in that it is a deterministic function given a configuration. Table 3 shows the transitions by this oracle for the sentence in Figure 3.[2]

SL → SH and SR → RD, which cause a configuration never reached by the original one. For simplicity, we do not impose such constraint. Rather, since our oracles prefer RD over SH by default it is not uncommon to explore such configurations during training.

[2] In this oracle, the condition for SW is given by *isCross*(i,j), which checks whether i and j are involved in two crossing arcs. This is apart from the condition for swap in ASS, which is based on the notion of *projective order* (Nivre, 2009). These two requisites are conceptually similar, but not identical, and they often result in different predictions for the next transition. We choose to use *isCross*(i,j) as we feel it is simpler. As a reviewer pointed out, for ASS using *isCross* generally increases the number of necessary swaps. This is essentially because ASS reduces a token when establishing

Transition	Configuration	Condition
STAY-LEFT$_l$	$(\sigma \vert i, j \vert \beta, A)$	$(j, l, i) \in A_g$
STAY-RIGHT$_l$	$(\sigma \vert i, j \vert \beta, A)$	$(i, l, j) \in A_g$
SWAP	$(\sigma \vert i, j \vert \beta, A)$	*isCross*(i, j)
REDUCE	$(\sigma \vert i, \quad \beta, A)$	$(\cdot, \cdot, i) \in A \wedge \forall h. \forall j. ((i, h, j) \in A_g \rightarrow (i, h, j) \in A)$
SHIFT	$(\sigma, \quad j \vert \beta, A)$	$\forall i. \forall l. ((j, l, i) \in A_g \wedge j < i) \rightarrow (j, l, i) \in A$

[1] *isCross*(i, j) returns true if i and j are two endpoints of two crossing arcs yet unattached. Formally: $(\exists b. b \in \beta \wedge ((b, \cdot, i) \in A_g \wedge (b, \cdot, i) \notin A \vee (i, \cdot, b) \in A_g \wedge (i, \cdot, b) \notin A)) \wedge (\exists s. s \in \sigma \wedge ((s, \cdot, j) \in A_g \wedge (s, \cdot, j) \notin A \vee (j, \cdot, s) \in A_g \wedge (j, \cdot, s) \notin A))$.

Table 2: A static oracle for our transition system. A_g is the set of gold arcs. $\cdot$ means an arbitrary value.

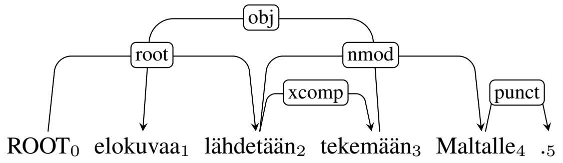

Figure 3: A non-projective sentence of Finnish.

t	Transition	Stack	Buffer	Added Arc
0		$[0]$	$[1, 2, 3, 4, 5]$	
1	SH	$[0, 1]$	$[2, 3, 4, 5]$	
2	SW	$[0]$	$[2, 1, 3, 4, 5]$	
3	SR	$[0]$	$[2, 1, 3, 4, 5]$	(0, root, 2)
4	SH	$[0, 2]$	$[1, 3, 4, 5]$	
5	SH	$[0, 2, 1]$	$[3, 4, 5]$	
6	SL	$[0, 2, 1]$	$[3, 4, 5]$	(3, obj, 1)
7	RD	$[0, 2]$	$[3, 4, 5]$	
8	SR	$[0, 2]$	$[3, 4, 5]$	(2, xcomp, 3)
9	SH	$[0, 2, 3]$	$[4, 5]$	
10	RD	$[0, 2]$	$[4, 5]$	
11	SR	$[0, 2]$	$[4, 5]$	(2, nmod, 4)
12	SH	$[0, 2, 4]$	$[5]$	
13	SR	$[0, 2, 4]$	$[5]$	(4, punct, 5)
14	RD	$[0, 2]$	$[5]$	
15	SH	$[0, 2, 5]$	$[]$	
16	RD	$[0, 2]$	$[]$	
17	RD	$[0]$	$[]$	

Table 3: Static oracle transitions by our system for the sentence in Figure 3.

Non-static oracle In addition to the static oracle, we also try a partially non-static oracle, which occasionally prefers SH over RD when both are applicable. Specifically, for this oracle when both conditions for SH and RD are satisfied we choose SH with some probability. This allows the parser to learn the transitions that postpone RD when possible, but stochastically, which we found effective in many languages in practice. This is a partially non-static oracle since it does not postpone the other transitions such as SL and SR. Designing such oracle could also be possible; for example, in Figure 2, we can also build the gold tree by SW followed by SH and SR (see also footnote 2). We leave such fully non-static oracle as well as the dynamic oracle (Goldberg and Nivre, 2012) as a future work.

3.3 Auxiliary transitions

Our system employs the following two additional transitions (AUX in Table 1), which can be applied in restricted conditions.

UNSHIFT The arc-eager system is not guaranteed to output a single rooted tree, i.e., it may keep unconnected tree fragments on the stack at a terminal configuration (Nivre, 2008). To escape from this, we employ the same hack as Nivre and Fernández-González (2014) and add a special transition UN, which pushes back the stack top node to the empty buffer. We only apply UN at decoding. It is deterministic and does not have any associated score. By adding this our system becomes both sound and complete for the class of all non-projective trees (the proof is omitted).

RIGHT-ROOT This is our new transition to improve the root attachment accuracy for arc-eager. The arc-eager system attaches the sentence root to the special ROOT *eagerly* immediately after it collects all its left dependents, but this decision is sometimes hard for some types of garden-path. The purpose of RR is to postpone the decision of this root token until a terminal configuration, as in the arc-standard system.

To be concrete, during training, we allow the parser to select SH with some probability when the gold transition is SR$_{root}$. This eventually leads to a terminal configuration where the sentence root token not attached to ROOT remains on the stack.

an arc, which necessitates to reorder the tokens on crossing arcs with swaps *before* creating arcs. Though the theoretical analysis is beyond the scope of this paper, we find our oracle in Table 2 does not suffer from such inefficiency in most cases. The key observation is that our oracle prefers SL and SR over SW, meaning that it attaches an arc regardless of *isCross* condition. If our oracle postpones SL and SR as in the oracle of Nivre (2009), the number of necessary SWs will be increased.

RR is used only on this configuration, also during decoding. Note that unlike UN, the parser explores this transition during training and learns the parameters associated with it. We hope by this the parser becomes capable of postponing the decision on the root token during decoding when it seems ambiguous locally. We use this transition only with the non-static oracle.

4 Parser Model

A model of a transition-based parser calculates the score of each transition at the current configuration. Our model is basically the stack-LSTM parser (Dyer et al., 2015)[3], which we slightly customize from the original architecture (Section 4.2). In this work we focus our attention on the incremental setting, in which the model is not able to access the full tokens in the buffer. We remark for transition-based parsers this is practically a more important scenario where the graph-based parser is not applicable.

4.1 Stack-LSTM parser

We first briefly describe the original model in Dyer et al. (2015) designed for the arc-standard system. For a configuration c_t at time t, the parser maintains the three vector representations, $\mathbf{s}_t$ that encodes the stack, $\mathbf{b}_t$ the buffer, and $\mathbf{a}_t$ the action history. Each of them is modeled with a stack LSTM, an LSTM that supports push and pop operations by keeping the representations of intermediate time steps. The stack LSTM for $\mathbf{s}_t$ is left-to-right while that for $\mathbf{b}_t$ is right-to-left. $\mathbf{a}_t$ encodes the entire action history from the initial action to the last action. Using these representations we encode the configuration into a single vector:

$$\mathbf{p}_t = \mathrm{ReLU}(\mathbf{W}[\mathbf{s}_t; \mathbf{b}_t; \mathbf{a}_t] + \mathbf{e}_p),$$

where $\mathbf{W}$ is the parameters. Here and the followings $\mathbf{e}_x$ denotes a bias vector.

Using this the probability for each valid transition z_t is obtained with restricted softmax:

$$p(z_t|\mathbf{p}_t) = \frac{\exp(\mathbf{g}_{z_t}^{\mathrm{T}}\mathbf{p}_t + q_{z_t})}{\sum_{z' \in \mathcal{A}(c_t)} \exp(\mathbf{g}_{z'}^{\mathrm{T}}\mathbf{p}_t + q_{z'})},$$

where $\mathbf{g}_z$ is the parameters and q_z is the bias term for action z. The set $\mathcal{A}(c_t)$ returns the set of valid transitions at c_t. After each transition we update $\mathbf{s}_{t+1}$, $\mathbf{b}_{t+1}$, and $\mathbf{a}_{t+1}$ accordingly following the new configuration.

For the buffer, each element of the LSTM is a token representation, which Dyer et al. (2015) obtains from the word and POS embeddings. $\mathbf{b}_t$ is then the last output of the LSTM.

For the stack, each element of the LSTM is a compositional representation of a subtree, or a token if it is just a shifted one. The subtree representation of the stack element is updated in a recursive manner. In Dyer et al. (2015) when LA or RA builds an arc $h \xrightarrow{l} d$, the representation $\mathbf{h}$ for the subtree rooted at h is updated by:

$$\mathbf{h}' = \tanh(\mathbf{U}[\mathbf{h}; \mathbf{d}; \mathbf{l}] + \mathbf{e}_h), \qquad (1)$$

in which $\mathbf{d}$ is the representation for the subtree rooted at d, $\mathbf{l}$ is the label embeddings. $\mathbf{s}_t$ is then updated by popping the top two elements of the stack LSTM, $\mathbf{h}$ and $\mathbf{d}$, and pushing $\mathbf{h}'$.

4.2 Modifications

We modify the above basic architecture in the following three ways.

Configuration encoding This is a restriction that we impose on the model. While the original model exploits the entire sentence for the buffer representation $\mathbf{b}_t$ using the LSTM, this violates our assumption of incrementality, the main advantage of the transition-based parsers. We thus avoid to use $\mathbf{b}_t$ and instead use the representations of top three nodes on the buffer: $\mathbf{b1}_t$, $\mathbf{b2}_t$, and $\mathbf{b3}_t$. We also use the representations of the top three nodes (subtrees) on the stack, $\mathbf{s1}_t$, $\mathbf{s2}_t$, and $\mathbf{s3}_t$, which we found effective. The new encoding is:

$$\mathbf{o} = \mathbf{W}[\mathbf{s}_t; \mathbf{a}_t; \mathbf{s1}_t; \mathbf{s2}_t; \mathbf{s3}_t; \mathbf{b1}_t; \mathbf{b2}_t; \mathbf{b3}_t] + \mathbf{e}_p,$$
$$\mathbf{p}_t = \mathrm{ReLU}(\mathbf{o}).$$

Token representation Many languages in UD are annotated with XPOS, fine language specific tags, as well as FEATS, the morphological features. We utilize the embeddings of these features, initialized randomly. We also add character embeddings, which we obtain from character-level bidirectional LSTMs. Our token representation is:

$$\mathbf{x} = \mathrm{ReLU}(\mathbf{V}[\mathbf{w}; \mathbf{t}; \mathbf{tx}; \mathbf{f}; \mathbf{w}_{ch}] + \mathbf{e}_x),$$

where $\mathbf{w}$, $\mathbf{t}$, $\mathbf{tx}$, and $\mathbf{f}$ are word, POS, XPOS, and FEAT embeddings, respectively. $\mathbf{w}_{ch}$ is the output of linear mapping from the concatenation of

[3] Please refer Ballesteros et al. (2015), Ballesteros et al. (2016), Ballesteros et al. (2017) for the latest version which is sophisticated in some architectures (e.g. character information, dynamic oracle).

the last hidden states of the forward and backward character-level LSTMs.

Composition This is the only modification needed to obtain the stack representation in our stay-eager transition system. The subtree representation in Dyer et al. (2015) is fully compositional in that $\mathbf{h}$ in Eq. 1 encodes the entire subtree with the recursive network. This is possible essentially because of the bottom-up nature of the arc-standard system. Unfortunately the same encoding is not straightforward in our system due to its arc-eager property, in which the right arcs are constructed top-down (Nivre, 2004). In this work, we give up the full compositionality of the original model, and simply mimic Eq. 1 with the following equation:

$$\mathbf{c}' = \tanh(\mathbf{U}[\mathbf{h}; \mathbf{d}; \mathbf{l}; \mathbf{c}] + \mathbf{e}_h). \quad (2)$$

We update the node representation of both of the stack top and the buffer top. This means that apart from the original model we also update the dependent node with composition. In the equation, $\mathbf{c}'$ is the updated representation of the head or the dependent after SR or SL, which is originally $\mathbf{c}$. For example, after SL, since the stack top becomes dependent, its representation ($\mathbf{d}$) is updated to $\tanh(\mathbf{U}[\mathbf{h}; \mathbf{d}; \mathbf{r}; \mathbf{d}] + \mathbf{e}_h)$. Note that without $\mathbf{c}$ in Eq. 2, the representations of two updated nodes are identical. The role of $\mathbf{c}$ is thus to distinguish the two updates for $\mathbf{h}$ and $\mathbf{d}$.

5 Experiment

Data We use the 63 treebanks in 45 languages in Universal Dependencies 2.0 (Nivre et al., 2017), which are distributed with the training data in the recent shared task in CoNLL 2017 (Zeman et al., 2017).[4] Following the shared task, we focus on real world parsing and assume the raw input text. For all preprocessing (sentence segmentation, tokenization, and tagging), we use UDpipe 1.1 (Straka et al., 2015). We report the official F1 LAS used in the shared task.

Baseline To make a comparison between transition systems fair, we implement the arc-standard *lazy swap* (ASS) parser (Nivre et al., 2009) with almost the same settings as our stay-eager swap (SES) parser including our network architecture (Section 4.2).[5] We also report the scores of UDpipe 1.1, the baseline system in the shared task, although the results may not be directly comparable as they tune several settings including the oracle and learning rate etc. for each language.

Settings Our network sizes are: 100 dimensional word embeddings and LSTMs, 50 dimensional POS, XPOS, and FEATS embeddings, and 20 dimensional action and label embeddings, and 32 dimensional character embeddings and bi-LSTMs. We do not use any pre-trained embeddings. We use Adam (Kingma and Ba, 2014) for the optimizer, and set the learning decay of 0.08 and the dropout ratio in LSTMs of 0.33.

In addition to the greedy search, we also try beam search for learning and decoding (beam size is 8). Note that due to swap, each transition sequence may have a different number of actions. We alleviate this inconsistency by ranking with the average scores (Honnibal and Johnson, 2014).[6]

For non-static oracles, we set both probabilities to postpone RD and SR_{root} to 0.33, which works well for the development set.

Results The main results are shown in the left columns of Table 4. Comparing to ASS, our non-static SES achieves the higher LAS on average, regardless of search method. In more detail, it is on about half treebanks (27 for greedy and 28 for beam search) that the score improvements from ASS are more than 0.5 points. Also the static SES is not stronger, suggesting that non-static exploration during training is important for our system.

Focusing on the results on only non-projective sentences (right columns), the score improvements get larger: the average LAS difference between non-static SES and SAS is 1.54 points with greedy search, and 1.18 points with beam search.

To further inspect the parser behaviors on non-projective and projective sentences, we next com-

[4] For small treebanks without the development set, we randomly divide the training data at 1:9 ratio for development and training.

[5] This baseline is competitive to or stronger than the original implementation of Dyer et al. (2015), which also implements arc-standard swap (https://github.com/clab/lstm-parser). Example UAS on development sets (with gold tags) are: Arabic: 80.83 (Dyer et al.) vs. 82.01 (ours); English: 86.71 (Dyer et al.) vs. 85.28 (ours); and German: 82.87 (Dyers et al.) vs. 84.45 (ours). Both employ greedy search. Note that our system does not use the buffer LSTM.

[6] For learning, we find the following heuristics inspired by max-violation (Huang et al., 2012) works well. Our training is basically local with cross-entropy while for each sentence we calculate the max violation point by beam search and use only the prefix until that point. Although this is simpler than global structured learning (Andor et al., 2016), it provides some improvements with much faster training time.

Language	Non-proj ratio	All sentences				Only non-projective sentences			
		ASS	SES		UDpipe	ASS	SES		UDpipe
			static	non-static			static	non-static	
grc	64.40%	49.18 (51.28)	50.00 (53.04)	**50.10** (**53.85**)	56.04	45.54 (47.54)	46.57 (49.89)	**46.73** (**50.72**)	52.83
la	47.50%	37.78 (37.82)	38.32 (42.50)	**41.22** (**43.27**)	43.77	32.64 (32.19)	32.76 (**37.60**)	**35.51** (37.31)	38.22
grc_proiel	37.92%	60.67 (63.74)	**60.92** (**64.69**)	60.66 (63.77)	65.22	54.57 (58.37)	56.15 (**60.58**)	**56.61** (60.28)	60.77
la_ittb	35.87%	72.29 (75.20)	71.98 (75.01)	**72.61** (**75.64**)	76.98	67.07 (71.10)	**67.56** (70.56)	67.31 (**72.03**)	72.14
eu	31.80%	**66.61** (67.67)	65.68 (67.66)	65.74 (**68.96**)	69.15	**58.98** (60.01)	58.15 (60.86)	58.49 (**62.21**)	61.46
en_lines	29.54%	70.37 (72.16)	71.71 (72.59)	**71.98** (**73.02**)	72.94	65.07 (66.96)	**66.86** (67.30)	66.12 (**67.69**)	67.83
la_proiel	28.30%	50.56 (54.38)	53.71 (54.72)	**53.87** (**56.18**)	57.54	45.80 (48.98)	48.52 (50.37)	**48.98** (**51.44**)	53.22
nl_lassysmall	28.13%	76.48 (76.25)	77.35 (77.78)	**77.63** (**77.83**)	78.15	71.76 (71.58)	73.90 (**74.52**)	**75.36** (73.88)	74.12
pt	23.69%	**77.12** (**78.01**)	74.89 (75.49)	75.07 (77.75)	82.11	**71.06**(**72.42**)	70.01 (70.87)	69.54 (71.37)	77.04
de	23.23%	65.11 (**66.94**)	63.35 (66.64)	**65.13** (66.83)	69.11	62.30 (64.51)	61.90 (64.53)	**62.83** (**64.99**)	66.66
gl_treegal	23.00%	62.51 (63.78)	64.09 (63.66)	**64.55** (**64.48**)	65.82	60.51 (62.51)	**63.38** (62.28)	63.23 (**62.94**)	64.30
nl	22.92%	**64.56** (65.69)	62.87 (66.86)	64.26 (**67.99**)	68.90	61.52 (63.79)	61.66 (**67.03**)	**63.07** (66.33)	68.30
got	22.67%	54.07 (57.40)	55.46 (57.71)	**56.66** (**58.25**)	59.81	47.04 (50.99)	48.73 (52.13)	**49.44** (**52.28**)	53.45
hu	21.60%	61.64 (62.72)	60.61 (62.44)	**61.94** (**63.45**)	64.30	56.79 (57.83)	56.18 (59.95)	**57.22** (**60.18**)	60.05
cu	18.93%	58.50 (60.41)	**61.17** (62.77)	60.66 (**63.62**)	62.76	51.25 (54.44)	53.23 (58.40)	**54.44** (**58.57**)	56.21
ur	18.88%	**75.91** (**76.70**)	75.46 (76.46)	75.85 (76.49)	76.69	69.50 (71.22)	69.61 (70.69)	**70.06** (**71.38**)	71.45
sv_lines	18.38%	71.48 (72.77)	71.18 (72.58)	**72.14** (**73.95**)	74.29	63.43 (65.56)	64.90 (65.77)	**65.84** (**67.05**)	67.24
et	16.87%	56.02 (56.26)	**56.33** (56.57)	56.10 (**57.64**)	58.79	48.12 (47.91)	47.88 (49.42)	**49.42** (**51.05**)	52.32
da	16.46%	70.23 (72.03)	69.63 (70.24)	**70.38** (**72.31**)	73.38	65.05 (67.83)	64.30 (66.07)	**66.11** (**69.15**)	68.27
sl	15.99%	78.58 (79.85)	**78.72** (79.49)	**78.72** (**79.95**)	81.15	**75.18** (75.59)	75.11 (76.73)	74.11 (**78.28**)	78.56
cs_cltt	15.76%	68.78 (68.66)	69.97 (70.06)	**70.52** (**71.30**)	71.64	63.95 (64.15)	65.82 (**66.12**)	**66.69** (65.85)	68.59
cs_cac	12.74%	79.92 (**81.56**)	**80.10** (81.20)	79.42 (81.18)	82.46	74.47 (75.43)	**75.16** (76.38)	74.52 (**76.76**)	77.34
hi	12.53%	**85.29** (**85.82**)	84.22 (85.56)	84.38 (85.79)	86.77	**82.05** (82.56)	81.06 (83.35)	81.98 (**83.40**)	83.54
sl_sst	12.18%	41.45 (42.67)	**42.40** (44.15)	41.69 (**44.40**)	46.45	36.06 (37.91)	**38.79** (**41.10**)	37.76 (40.71)	42.45
tr	12.10%	**52.67** (51.56)	52.00 (52.54)	**52.78** (52.49)	53.19	41.88 (41.97)	42.46 (**43.23**)	**44.68** (41.59)	42.50
cs	11.98%	77.82 (**80.43**)	**78.13** (79.62)	78.01 (80.36)	82.87	73.57 (77.03)	**74.95** (77.10)	74.54 (**77.69**)	80.47
el	11.62%	76.89 (78.31)	76.67 (77.72)	**77.73** (**78.67**)	79.26	**76.56** (75.36)	76.02 (**77.34**)	75.66 (77.22)	77.52
ar	11.62%	63.79 (64.80)	**64.97** (**65.04**)	64.27 (64.74)	65.30	55.58 (**57.69**)	**57.65** (57.57)	56.82 (57.21)	58.24
kk	11.56%	**21.83** (21.75)	18.43 (19.59)	20.08 (**22.84**)	24.51	**15.65** (**15.49**)	11.85 (12.41)	12.86 (15.44)	16.46
fr	11.54%	**78.05** (**79.52**)	77.70 (78.86)	77.97 (79.43)	80.75	**75.14** (75.59)	72.99 (76.50)	74.52 (**77.18**)	78.03
ca	10.94%	82.94 (83.86)	83.28 (84.02)	**83.40** (**84.33**)	85.39	78.01 (78.83)	78.35 (**79.40**)	**78.97** (78.76)	80.07
ga	10.13%	58.58 (60.29)	60.50 (61.30)	**61.34** (**62.03**)	61.52	56.14 (59.11)	58.89 (59.45)	**59.89** (**59.89**)	60.34
es	10.09%	79.48 (80.42)	79.64 (**80.81**)	**79.71** (80.49)	81.47	77.14 (78.49)	76.83 (**78.69**)	**78.49** (78.33)	79.57
ro	9.74%	76.66 (78.25)	77.51 (78.34)	**77.75** (**78.92**)	79.88	73.01 (**74.78**)	71.07 (74.48)	**73.96** (74.17)	76.16
es_ancora	9.65%	81.43 (83.18)	**81.99** (82.88)	81.72 (**83.21**)	83.78	75.74 (77.31)	**76.56** (77.04)	75.22 (**78.09**)	77.37
ko	9.61%	**73.70** (**74.24**)	72.11 (72.80)	72.33 (73.39)	59.09	**65.50** (**65.22**)	63.02 (65.09)	63.02 (64.12)	47.56
ru	9.48%	71.08 (73.84)	**72.49** (73.33)	72.17 (**74.85**)	74.03	58.79 (61.80)	**62.84** (62.17)	**62.84** (**64.24**)	63.31
ru_syntagrus	9.21%	83.89 (85.33)	**84.26** (84.95)	84.17 (**85.42**)	86.76	78.42 (79.55)	78.97 (80.03)	**79.16** (**80.42**)	82.24
no_nynorsk	9.20%	**78.69** (79.52)	78.26 (78.78)	78.13 (**79.56**)	81.56	72.62 (74.25)	**74.88** (73.71)	74.01 (**75.90**)	77.23
hr	9.17%	**73.74** (**76.08**)	73.47 (75.63)	73.61 (75.41)	77.18	**67.63** (70.20)	64.86 (**71.32**)	66.58 (70.73)	72.12
fr_sequoia	8.77%	77.22 (78.69)	77.09 (77.61)	**78.19** (**79.16**)	79.98	73.91 (73.91)	**74.58** (73.42)	73.29 (**76.05**)	76.35
uk	7.95%	57.85 (58.67)	58.97 (59.84)	**59.80** (**61.23**)	60.76	50.63 (52.37)	**52.42** (53.51)	52.10 (**55.04**)	54.55
no_bokmaal	7.63%	79.81 (80.69)	79.75 (80.37)	**80.36** (**81.31**)	83.27	73.00 (73.28)	73.28 (74.01)	**73.37** (**75.40**)	76.98
fa	7.17%	74.83 (**77.89**)	74.61 (77.42)	**75.94** (77.87)	79.24	67.59 (73.42)	69.42 (72.61)	**72.95** (**74.64**)	74.78
fi_ftb	7.02%	70.95 (**73.15**)	71.07 (72.03)	**71.30** (72.75)	74.03	61.66 (66.13)	**66.01** (67.56)	65.12 (**68.04**)	66.85
lv	6.64%	55.54 (57.10)	54.50 (56.76)	**56.47** (**57.35**)	59.95	46.56 (47.83)	48.62 (49.25)	**49.49** (**50.20**)	53.83
fi	6.24%	**71.55** (72.37)	70.45 (**72.62**)	71.53 (72.18)	73.75	65.95 (67.94)	68.26 (**68.74**)	**68.31** (67.56)	65.79
id	5.75%	71.48 (73.78)	72.17 (**74.16**)	**72.49** (74.04)	74.61	62.20 (67.20)	64.43 (66.49)	**65.50** (**67.38**)	66.13
it	4.98%	83.06 (84.68)	83.61 (84.30)	**83.74** (**84.92**)	85.28	73.86 (**77.20**)	74.47 (76.29)	**75.79** (75.08)	77.91
pt_br	4.90%	**83.59** (**84.39**)	83.38 (83.92)	83.35 (84.04)	85.36	74.86 (**76.47**)	73.24 (75.25)	**75.40** (75.35)	76.57
ug	4.78%	31.54 (34.02)	33.09 (33.04)	**34.40** (**34.57**)	34.18	23.49 (25.44)	24.01 (23.23)	**25.83** (**27.13**)	25.05
sk	4.34%	70.13 (70.76)	**70.76** (**70.99**)	70.71 (70.69)	72.75	62.65 (64.85)	**66.80** (64.59)	63.81 (**68.35**)	68.35
fr_partut	4.25%	75.86 (76.47)	76.81 (77.10)	**77.43** (**77.97**)	77.38	70.48 (71.01)	**71.01** (**75.07**)	69.60 (72.60)	70.13
en_partut	4.00%	71.00 (72.61)	71.04 (72.43)	**71.78** (**73.44**)	73.64	**65.95** (68.11)	66.52 (67.39)	**66.95** (**69.40**)	68.97
en	3.71%	**72.53** (73.56)	72.51 (73.81)	72.29 (**73.90**)	75.84	59.12 (61.13)	58.95 (**63.04**)	**62.66** (61.95)	63.96
vi	3.25%	36.37 (36.37)	36.11 (37.00)	**36.68** (**37.47**)	37.47	32.17 (31.98)	33.72 (**33.14**)	**35.27** (31.98)	31.59
bg	2.87%	**81.24** (**82.02**)	79.92 (81.32)	80.58 (81.79)	83.64	**80.00** (79.67)	76.49 (**80.67**)	77.66 (78.66)	80.33
sv	2.30%	74.09 (74.67)	73.65 (74.29)	**74.24** (**75.35**)	76.73	**69.94** (66.34)	68.42 (69.53)	**69.94** (**69.67**)	71.19
he	1.83%	55.02 (56.58)	54.91 (56.21)	**56.33** (**56.90**)	57.23	55.34 (59.92)	52.29 (56.11)	**61.07** (**61.45**)	61.45
zh	1.40%	**53.55** (**55.23**)	51.65 (53.94)	51.93 (53.11)	57.40	42.11 (**47.24**)	42.88 (45.70)	**45.96** (45.19)	50.58
pl	0.27%	76.80 (78.02)	**77.70** (**78.09**)	77.26 (77.75)	78.78	66.67 (68.75)	81.25 (**81.25**)	**83.33** (70.83)	66.67
gl	0.00%	76.16 (77.43)	**77.24** (77.68)	77.04 (**78.35**)	77.31	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00
ja	0.00%	**71.47** (**72.38**)	63.91 (68.59)	66.96 (71.77)	72.21	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00
Average.	13.76%	67.59 (68.93)	67.56 (68.88)	**68.05** (**69.55**)	70.34	59.51 (61.18)	60.28 (61.98)	**61.05** (**62.36**)	62.75

Table 4: LAS of all sentences and of only non-projective sentences ordered by the ratio of non-projective sentences in the test data. The scores in brackets are the results with beam search.

Sentence	Non-proj ratio	ASS	SES		UDpipe
			static	non-static	
All	HIGH	62.07 (63.79)	62.21 (64.34)	**62.96** (**65.09**)	66.42
	MID	67.27 (68.30)	67.39 (68.35)	**67.64** (**69.05**)	69.93
	LOW	73.23 (74.85)	73.25 (74.50)	**73.73** (**75.11**)	75.19
	VERYLOW	67.49 (68.61)	67.09 (68.18)	**67.65** (**68.80**)	69.68
Projective	HIGH	73.45 (73.57)	73.57 (74.71)	**73.89** (**75.25**)	76.27
	MID	69.86 (69.96)	69.96 (71.46)	**70.38** (**71.73**)	72.76
	LOW	**68.41** (67.32)	67.32 (69.43)	68.20 (**70.10**)	70.48
	VERYLOW	65.16 (65.22)	65.22 (66.13)	**65.54** (**66.58**)	67.55
Non-projective	HIGH	57.19 (59.20)	58.02 (60.61)	**58.60** (**60.98**)	62.17
	MID	62.04 (63.27)	62.41 (63.99)	**63.02** (**64.50**)	65.22
	LOW	66.21 (68.35)	67.28 (68.63)	**67.66** (**69.47**)	68.75
	VERYLOW	59.74 (61.39)	60.77 (**62.40**)	**62.56** (62.13)	62.52

[1] HIGH (20%-) : grc, la, grc_proiel, la_ittb, eu, en_lines, la_proiel, nl_lassysmall, pt, de, gl_treegal, nl, got, and hu.

[2] MID (10%-20%) : cu, ur sv_lines, et, da, sl, cs_cltt, cs_cac, hi, sl_sst, tr, cs, el, ar, kk, fr, ca, ga, and es.

[3] LOW (5%-10%) : ro, es_ancora, ko, ru, ru_syntagrus, no_nynorsk, hr, fr_sequoia, uk, no_bokmaal, fa, fi_ftb, lv, fi, and id.

[4] VERYLOW (0%-5%) : it, pt_br, ug, sk, fr_rtut, en_partut, en, vi, bg, sv, he, zh, and pl.

[5] gl and ja are excluded when calculating the scores of "non-projective only" (bottom rows), as these treebanks only contain projective sentences in their test data.

Table 5: The average LAS on all, only projective, and only non-projective sentences on the grouped treebanks according to the ratio of non-projective sentences in the test set. The scores in brackets are the results with beam search.

pare the average LAS on a subset of treebanks, which we divide into four groups according to the ratio of non-projective sentences (Table 5). When evaluating on all sentences (top rows), we can see the larger improvements by the non-static SES in HIGH, MID, and LOW groups (having non-negligible non-projectivity), which confirms the above results. Interestingly, for projective sentences only (mid rows) the scores of SES do not degrade comparing to ASS, or rather improves in most cases. This suggests our transition system also helps to recover the projective arcs.

6 Discussion

While the ordinary reordering-based transition system is built on the arc-standard system, we choose arc-eager as our basic architecture. One reason for this is that decomposing the arc-standard system is more involved than arc-eager; Gómez-Rodríguez and Nivre (2013) observe the RIGHT-ARC in arc-standard would be divided into four transitions including a nontrivial UNSHIFT operation. Otherwise, we need two different reduce operations for each direction, which complicates the system and learning.

Though we have seen the empirical advantage, in terms of the reordering strategy our approach may not be optimal. Consider the sentence in Figure 4, which we borrow from Nivre et al. (2009).

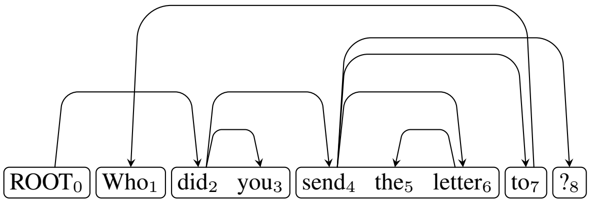

Figure 4: Another non-projective sentence.

Our system needs more swap transitions than the Nivre et al.'s swap-lazy system for this sentence. In Nivre et al.'s system, swapping *Who*$_1$ and *did*$_2$ occurs after *you*$_3$ is reduced as a dependent of *did*$_2$. In our system, due to the right top-down nature of arc-eager, we need to build *ROOT*$_0$ → *did*$_2$ before *did*$_2$ → *you*$_3$. This means we also need an additional swap between *Who*$_1$ and *you*$_3$.

Past work shows that a smaller number of swap transitions improves accuracies (Björkelund and Nivre, 2015), and thus it is an important future work to revise our system to minimize the necessary swap transitions. Another direction might be to incorporate our idea to postpone swap transitions into the arc-standard system, possibly with the divided system as we did for arc-eager.

In our system each word is once attached, shifted, and reduced, so the total number of transitions is $3n$ plus the number of swap transitions. This is greater than Nivre et al.'s system, though we expect this additional cost is not substantial comparing to the other techniques, e.g., beam search with larger beam sizes.

7 Conclusion

We have shown for incremental non-projective parsing, explicitly separating the attachment and reduce transitions alleviates the difficulty of local decisions, and leads to higher parsing accuracies for both projective and crossing arcs. Non-projectivity is prevalent in multilingual parsing beyond the popular languages in the current NLP such as English and Chinese. Also incremental parsing is essential for many online applications, in particular the speech-oriented systems. In this paper, we proposed an alternative to the popular approach for incremental non-projective parsing. There are much rooms for improvements, and this is our first step of reconsidering the optimal mechanism to handle crossing arcs incrementally.

Acknowledgements

This work was in part supported by JST CREST Grant Number JPMJCR1513, Japan, and JSPS KAKENHI Grant Number 16H06981.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 2442–2452. http://www.aclweb.org/anthology/P16-1231.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. Association for Computational Linguistics, New York City, pages 166–170. http://www.aclweb.org/anthology/W/W06/W06-2922.

Miguel Ballesteros, Chris Dyer, Yoav Goldberg, and Noah A Smith. 2017. Greedy transition-based dependency parsing with stack lstms. *Computational Linguistics* .

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 349–359. http://aclweb.org/anthology/D15-1041.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack lstm parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 2005–2010. https://aclweb.org/anthology/D16-1211.

Anders Björkelund and Joakim Nivre. 2015. Non-deterministic oracles for unrestricted non-projective transition-based dependency parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*. Association for Computational Linguistics, Bilbao, Spain, pages 76–86. http://www.aclweb.org/anthology/W15-2210.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, Jeju Island, Korea, pages 1455–1465. http://www.aclweb.org/anthology/D12-1133.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 740–750. http://www.aclweb.org/anthology/D14-1082.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Sofia, Bulgaria, pages 1052–1062. http://www.aclweb.org/anthology/P13-1104.

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

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 334–343. http://www.aclweb.org/anthology/P15-1033.

Haim Gaifman. 1965. Dependency systems and phrase-structure systems. *Information and Control* 8(3):304 – 337.

https://doi.org/http://dx.doi.org/10.1016/S0019-9958(65)90232-9.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*. The COLING 2012 Organizing Committee, Mumbai, India, pages 959–976. http://www.aclweb.org/anthology/C12-1059.

Carlos Gómez-Rodríguez and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Computational Linguistics* 39(4):799–845. http://www.aclweb.org/anthology/J/J13/J13-4002.pdf.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental joint pos tagging and dependency parsing in chinese. In *Proceedings of 5th International Joint Conference on Natural Language Processing*. Asian Federation of Natural Language Processing, Chiang Mai, Thailand, pages 1216–1224. http://www.aclweb.org/anthology/I11-1136.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics* 2:131–142. https://transacl.org/ojs/index.php/tacl/article/view/234.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Montréal, Canada, pages 142–151. http://www.aclweb.org/anthology/N12-1015.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Vancouver, British Columbia, Canada, pages 523–530. http://www.aclweb.org/anthology/H/H05/H05-1066.

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

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*. Association for Computational Linguistics, Barcelona, Spain, pages 50–57.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553. http://www.aclweb.org/anthology/J/J08/J08-4003.pdf.

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*. Association for Computational Linguistics, Suntec, Singapore, pages 351–359. http://www.aclweb.org/anthology/P/P09/P09-1040.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen Cebirolu Eryiit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Silvie Cinková, Çar Çöltekin, Miriam Connor, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökrmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà M, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Petter Hohle, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, Phng Lê Hng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Lng Nguyn Th, Huyn

Nguyn Th Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalnia, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Rudolf Rosa, Davide Rovati, Shadi Saleh, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Antonio Stella, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jonathan North Washington, Zhuoran Yu, Zdeněk Žabokrtský, Daniel Zeman, and Hanzhi Zhu. 2017. Universal dependencies 2.0 CoNLL 2017 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. http://hdl.handle.net/11234/1-2184.

Joakim Nivre and Daniel Fernández-González. 2014. Arc-eager parsing with the tree constraint. *Computational linguistics* 40(2):259–267.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*. Association for Computational Linguistics, Paris, France, pages 73–76. http://www.aclweb.org/anthology/W09-3811.

Emily Pitler and Ryan McDonald. 2015. A linear-time transition system for crossing interval trees. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Denver, Colorado, pages 662–671. http://www.aclweb.org/anthology/N15-1068.

Milan Straka, Jan Hajič, Jana Straková, and jr. Jan Hajič. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *14th International Workshop on Treebanks and Linguistic Theories (TLT 2015)*. IPIPAN, IPIPAN, Warszawa, Poland, pages 208–220.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinkova, Jan Hajic jr., Jaroslava Hlavacova, Václava Kettnerová, Zdenka Uresova, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria dePaiva, Kira Droganova, Héctor Martínez Alonso, Çağr Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonca, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. Conll 2017 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics, Vancouver, Canada, pages 1–19. http://www.aclweb.org/anthology/K/K17/K17-3001.pdf.