Transition-based Spinal Parsing

Miguel Ballesteros^{1,2} **Xavier Carreras**^{3}
^{1}NLP Group, Pompeu Fabra University ^{2}Carnegie Mellon University
^{3}Xerox Research Centre Europe
miguel.ballesteros@upf.edu xavier.carreras@xrce.xerox.com

Abstract

We present a transition-based arc-eager model to parse spinal trees, a dependency-based representation that includes phrase-structure information in the form of constituent spines assigned to tokens. As a main advantage, the arc-eager model can use a rich set of features combining dependency and constituent information, while parsing in linear time. We describe a set of conditions for the arc-eager system to produce valid spinal structures. In experiments using beam search we show that the model obtains a good trade-off between speed and accuracy, and yields state of the art performance for both dependency and constituent parsing measures.

1 Introduction

There are two main representations of the syntactic structure of sentences, namely constituent and dependency-based structures. In terms of statistical modeling, an advantage of dependency representations is that they are naturally lexicalized, and this allows the statistical model to capture a rich set of lexico-syntactic features. The recent literature has shown that such lexical features greatly favor the accuracy of statistical models for parsing (Collins, 1999; Nivre, 2003; McDonald et al., 2005). Constituent structure, on the other hand, might still provide valuable syntactic information that is not captured by standard dependencies.

In this work we investigate transition-based statistical models that produce *spinal trees*, a representation that combines dependency and constituent structures. Statistical models that use both representations jointly were pioneered by Collins (1999), who used constituent trees annotated with head-child information in order to define lexicalized PCFG models, i.e. extensions of classic constituent-based PCFG that make a central use of lexical dependencies.

An alternative approach is to view the combined representation as a dependency structure augmented with constituent information. This approach was first explored by Collins (1996), who defined a dependency-based probabilistic model that associates a triple of constituents with each dependency. In our case, we follow the representations proposed by Carreras et al. (2008), which we call spinal trees. In a spinal tree (see Figure 1 for an example), each token is associated with a *spine* of constituents, and head-modifier dependencies are attached to nodes in the spine, thus combining the two sources of information in a tight manner. Since spinal trees are inherently dependency-based, it is possible to extend dependency models for such representations, as shown by Carreras et al. (2008) using a so-called graph-based model. The main advantage of such models is that they allow a large family of rich features that include dependency features, constituent features and conjunctions of the two. However, the consequence is that the additional spinal structure greatly increases the number of dependency relations. Even though a graph-based model remains parseable in cubic time, it is impractical unless some pruning strategy is used (Carreras et al., 2008).

In this paper we propose a transition-based parser for spinal parsing, based on the arc-eager strategy by Nivre (2003). Since transition-based parsers run in linear time, our aim is to speed up spinal parsing while taking advantage of the rich representation it provides. Thus, the research question underlying this paper is whether we can accurately learn to take greedy parsing decisions for rich but complex structures such as spinal trees. To control the trade-off, we use beam search for transition-based parsing, which has been shown to be successful (Zhang and Clark, 2011b). The main contributions of this paper are

Proceedings of the 19th Conference on Computational Language Learning, pages 289–299,
Beijing, China, July 30-31, 2015. ©2015 Association for Computational Linguistics

the following:

- We define an arc-eager statistical model for spinal parsing that is based on the triplet relations by Collins (1996). Such relations, in conjunction with the partial spinal structure available in the stack of the parser, provide a very rich set of features.
- We describe a set of conditions that an arc-eager strategy must guarantee in order to produce valid spinal structures.
- In experiments using beam search we show that our method obtains a good trade-off between speed and accuracy for both dependency-based attachment scores and constituent measures.

2 Background

2.1 Spinal Trees

A spinal tree is a generalization of a dependency tree that adds constituent structure to the dependencies in the form of *spines*. In this section we describe the spinal trees used by Carreras et al. (2008). A spine is a sequence of constituent nodes associated with a word in the sentence. From a linguistic perspective, a spine corresponds to the projection of the word in the constituent tree. In other words, the spine of a word consists of the constituents whose head is the word. See Figure 1 for an example of a sentence and its constituent and spinal trees. In the example the spine of each token is the vertical sequence on top of it.

Formally a spinal tree for a sentence $x_{1:n}$ is a pair (V, E), where V is a sequence of n spinal nodes and E is a set of n spinal dependencies. The i-th node in V is a pair (x_i, σ_i), where x_i is the i-th word of the sentence and σ_i is its spine.

A spine σ is a vertical sequence of constituent nodes. We denote by $\mathcal{N}$ the set of constituent nodes, and we use $\star \notin \mathcal{N}$ to denote a special *terminal* node. We denote by $l(\sigma)$ the length of a spine. A spine σ is always non-empty, $l(\sigma) \geq 1$, its first node is always $\star$, and for any $2 \leq j \leq l(\sigma)$ the j-th node of the spine is an element of $\mathcal{N}$.

A spinal dependency is a tuple $\langle h, d, p \rangle$ that represents a directed dependency from the p-th node of σ_h to the d-th node of V. Thus, a spinal dependency is a regular dependency between a head token h and a dependent token d augmented with a position p in the head spine. It must be that $1 \leq h, d \leq n$ and that $1 < p \leq l(\sigma_h)$.

The set of spinal dependencies E satisfies the standard conditions of forming a rooted directed projected tree (Kübler et al., 2009). Plus, E satisfies that the dependencies are correctly nested with respect to the constituent structure that the spines represent. Formally, let (h, d_1, p_1) and (h, d_2, p_2) be two spinal dependencies associated with the same head h. For left dependencies, correct nesting means that if $d_1 < d_2 < h$ then $p_1 \geq p_2$. For right dependents, if $h < d_1 < d_2$ then $p_1 \leq p_2$.

In practice, it is straightforward to obtain spinal trees from a treebank of constituent trees with head-child annotations in each constituent (Carreras et al., 2008): starting from a token, its spine consists of the non-terminal labels of the constituents whose head is the token; the parent node of the top of the spine gives information about the lexical head (by following the head children of the parent) and the position where the spine attaches to. Given a spinal tree it is trivial to recover the constituent and dependency trees.

2.2 Arc-Eager Transition-Based Parsing

The arc-eager transition-based parser (Nivre, 2003) parses a sentence from left to right in linear time. It makes use of a stack that stores tokens that are already processed (partially built dependency structures) and it chooses the highest-scoring parsing action at each point. The arc-eager algorithm adds every arc at the earliest possible opportunity and it can only parse projective trees.

The training process is performed with an oracle (a set of transitions to a parse for a given sentence, (see Figure 2)) and it learns the best transition given a configuration. The SHIFT transition removes the first node from the buffer and puts it on the stack. The REDUCE transition removes the top node from the stack. The LEFT-ARC$_t$ transition introduces a labeled dependency edge between the first element of the buffer and the top element of the stack with the label t. The top element is removed from the stack (reduce transition). The RIGHT-ARC$_t$ transition introduces a labeled dependency edge between the top element of the stack and the first element in the buffer with a label d, and it performs a shift transition. Each action can have constraints (Nivre et al., 2014), Figure 2 and Section 3.2 describe the constraints of the spinal parser.

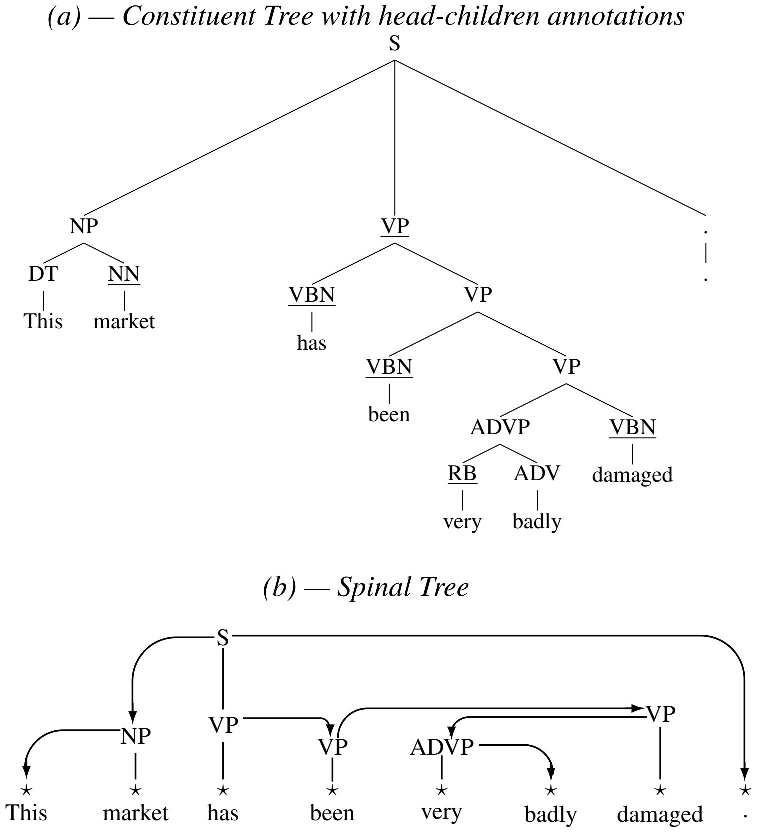

Figure 1: (a) A constituent tree for *This market has been very badly damaged.* For each constituent, the underlined child annotates the head child of the constituent. (b) The corresponding spinal tree.

In this paper, we took the already existent implementation of arc-eager from ZPar[1] (Zhang and Clark, 2009) which is a beam-search parser implemented in C++ focused on efficiency. ZPar gives competitive accuracies, yielding state-of-the-art results, and very fast parsing speeds for dependency parsing. In the case of ZPar, the parsing process starts with a root node at the top of the stack (see Figure 3) and the buffer contains the words/tokens to be parsed.

3 Transition-based Spinal Parsing

In this section we describe an arc-eager transition system that produces spinal trees. Figure 3 shows a parsing example. In essence, the strategy we propose builds the spine of a token by pieces, by adding a piece of spine each time the parser produces a dependency involving such token.

We first describe a labeling of dependencies that encodes a triplet of constituent labels, and it is the basis for defining an arc-eager statistical model. Then we describe a set of constraints that guarantees that the arc-eager derivations we produce correspond to spinal trees. Finally we discuss how to map arc-eager derivations to spinal trees.

3.1 Constituent Triplets

We follow Collins (1996) and define a labeling for dependencies based on constituent triplets.

Consider a spinal tree (V, E) for a sentence $x_{1:n}$. A *constituent triplet* of a spinal dependency $(h, d, p) \in E$ is a tuple $\langle a, b, c \rangle$ where:

- $a \in \mathcal{N}$ is the node at position p of σ_h (parent label)
- $b \in \mathcal{N} \cup \{\star\}$ is the node at position $p - 1$ of σ_h (head label)
- $c \in \mathcal{N} \cup \{\star\}$ is the top node of σ_d (dependent label)

For example, a dependency labeled with $\langle \mathrm{S}, \mathrm{VP}, \mathrm{NP} \rangle$ is a subject relation, while the triplet $\langle \mathrm{VP}, \star, \mathrm{NP} \rangle$ represents an object relation. Note that a constituent triplet, in essence, corresponds to a context-free production in a head-driven PCFG (i.e. $a \rightarrow bc$, where b is the head child of a).

[1] http://sourceforge.net/projects/zpar/

Initial configuration $C_i = \langle [\,], [x_1 \dots x_n], \emptyset, \rangle$

Terminal configuration $C_f \in \{C \mid C = \langle \Sigma, [\,], A \rangle\}$

SHIFT $\langle \Sigma, i|B, A \rangle \Rightarrow \langle \Sigma|i, B, A \rangle$

REDUCE $\langle \Sigma|i, B, A \rangle \Rightarrow \langle \Sigma, B, A \rangle$

if $\exists j, t : \{j \xrightarrow{t} i\} \in A$

LEFT-ARC($\langle a, b, c \rangle$) $\langle \Sigma|i, j|B, A \rangle \Rightarrow \langle \Sigma, j|B, A \cup \{j \xrightarrow{\langle a,b,c \rangle} i\} \rangle$

if $\neg \exists k, t : \{i \xrightarrow{t} k\} \in A$

(1) if $(c \neq \star) \ \vee \ \neg(\exists k : \{i \xrightarrow{t} k\} \in A)$

(3) if $b = \star \Rightarrow \forall \{i \xrightarrow{\langle a',b',c' \rangle} k\} \in A, \ a = a' \ \wedge \ b = b'$

RIGHT-ARC($\langle a, b, c \rangle$) $\langle \Sigma|i, j|B, A \rangle \Rightarrow \langle \Sigma|i|j, B, A \cup \{i \xrightarrow{\langle a,b,c \rangle} j\} \rangle$

(1) if $(c \neq \star) \ \vee \ \neg(\exists k : \{j \xrightarrow{t} k\} \in A)$

(2) if $\neg(\exists k, \langle a', b', c' \rangle : \{k \xrightarrow{\langle a',b',c' \rangle} i\} \in A \ \wedge \ c' = \star)$

(4) if $b = \star \Rightarrow \forall \{j \xrightarrow{\langle a',b',c' \rangle} k\} \in A$ such that $j < k, \ a = a' \ \wedge \ b = b'$

(5) if $b = \star \Rightarrow \forall \{j \xrightarrow{\langle a',b',c' \rangle} k\} \in A$ such that $k < j \ \wedge \ b' = \star, \ a = a'$

Figure 2: Arc-eager transition system with spinal constraints. Σ represents the stack, B represents the buffer, A represents the set of arcs, t represents a given triplet when its components are not relevant, $\langle a, b, c \rangle$ represents a given triplet when its components are relevant and i, j and k represent tokens of the sentence. The constraints labeled with (1) … (5) are described in Section 3.2. The constraints that are not labeled are standard constraints of the arc-eager parsing algorithm (Nivre, 2003).

In the literature, these triplets have been shown to provide very rich parameterizations of statistical models for parsing (Collins, 1996; Collins, 1999; Carreras et al., 2008).

For our purposes, we associate with each spinal dependency $(h, d, p) \in E$ a *triplet dependency* $(h, d, \langle a, b, c \rangle)$, where the triplet is defined as above. We then define a standard statistical model for arc-eager parsing that uses constituent triplets as dependency labels. An important advantage of this model is that left-arc and right-arc transitions can have feature descriptions that combine standard dependency features with phrase-structure information in the form of constituent triplets. As shown by Carreras et al. (2008), this rich set of features can obtain significant gains in parsing accuracy.

3.2 Spinal Arc-Eager Constraints

We now describe constraints that guarantee that any derivation produced by a triplet-based arc-eager model corresponds to a spinal structure.

Let us make explicit some properties that relate a derivation D with a token i, the arcs in D involving i, and its spine σ_i:

- D has at most a single arc $(h, i, \langle a, b, c \rangle)$ where i is in the dependent position. The dependent label c of this triplet defines the top of σ_i. If $c = \star$ then $\sigma_i = \star$, and i can not have dependants.
- Consider the subsequence of D of left arcs with head i, of the form $(i, j, \langle a, b, c \rangle)$. In an arc-eager derivation this subsequence follows a head-outwards order. Each of these arcs has in its triplet a pair of contiguous nodes $b{-}a$ of σ_i. We call such pairs *spinal edges*. The subsequence of spinal edges is ordered bottom-up, because arcs appear head-outwards. In addition, sibling arcs may attach to the same position in σ_i. Thus, the subsequence of left spinal edges of i in D is a subsequence with repeats of the sequence of edges of σ_i.
- Analogously, the subsequence of right spinal

Transition	**Stack**	**Buffer**	**Added Arc**
	[Root]	[This, market, has, been, very, badly, damaged, .]	
SHIFT	[Root, This]	[market, has, been, very, badly, damaged, .]	
L-A($\langle \mathrm{NP}, \star, \star \rangle$)	[Root]	[market, has, been, very, badly, damaged, .]	market $\xrightarrow{\langle \mathrm{NP},\star,\star \rangle}$ This
SHIFT	[Root, market]	[has, been, very, badly, damaged, .]	
L-A($\langle \mathrm{S}, \mathrm{VP}, \mathrm{NP} \rangle$)	[Root]	[has, been, very, badly, damaged, .]	has $\xrightarrow{\langle \mathrm{S},\mathrm{VP},\mathrm{NP} \rangle}$ market
R-A($\langle \mathrm{TOP}, \star, \mathrm{S} \rangle$)	[Root, has]	[been, very, badly, damaged, .]	Root $\xrightarrow{\langle \mathrm{TOP},\star,\mathrm{S} \rangle}$ has
R-A($\langle \mathrm{VP}, \star, \mathrm{VP} \rangle$)	[Root, has, been]	[very, badly, damaged, .]	has $\xrightarrow{\langle \mathrm{VP},\star,\mathrm{VP} \rangle}$ been
SHIFT	[Root, has,been, very]	[badly, damaged, .]	
R-A($\langle \mathrm{ADVP}, \star, \star \rangle$)	[Root, has, been, very, badly]	[damaged, .]	very $\xrightarrow{\langle \mathrm{ADVP},\star,\star \rangle}$ badly
REDUCE	[Root, has, been, very]	[damaged, .]	
L-A($\langle \mathrm{VP}, \star, \mathrm{ADVP} \rangle$)	[Root, has, been]	[damaged, .]	damaged $\xrightarrow{\langle \mathrm{VP},\star,\mathrm{ADVP} \rangle}$ very
R-A($\langle \mathrm{VP}, \star, \mathrm{VP} \rangle$)	[Root, has, been, damaged]	[.]	been $\xrightarrow{\langle \mathrm{VP},\star,\mathrm{VP} \rangle}$ damaged
REDUCE	[Root, has, been]	[.]	
REDUCE	[Root, has]	[.]	
R-A($\langle \mathrm{S}, \mathrm{VP}, \star \rangle$)	[Root, has, .]	[ ]	has $\xrightarrow{\langle \mathrm{S},\mathrm{VP},\star \rangle}$.

Figure 3: Transition sequence for *This market has been very badly damaged.*

edges of i in D is a subsequence with repeats of the edges of σ_i. In D, right spinal edges appear after left spinal edges.

We constrain the arc-eager transition process such that these properties hold. Recall that a well-formed spine starts with a terminal node $\star$, and so does the first edge of the spine and only the first. Let C be a configuration, i.e. a partial derivation. The constraints are:

(1) An arc $(h, i, \langle a, b, \star \rangle)$ is not valid if i has dependents in C.

(2) An arc $(i, j, \langle a, b, c \rangle)$ is not valid if C contains a dependency of the form $(h, i, \langle a', b', \star \rangle)$.

(3) A left arc $(i, j, \langle a, \star, c \rangle)$ is only valid if all sibling left arcs in C are of the form $(i, j', \langle a, \star, c' \rangle)$.

(4) Analogous to (3) for right arcs.

(5) If C has a left arc $(i, j, \langle a, \star, c \rangle)$, then a right arc $(i, j', \langle a', \star, c \rangle)$ is not valid if $a \neq a'$.

In essence, constraints 1-2 relate the top of a spine with the existence of descendants, while constraints 3-5 enforce that the bottom of the spine is well formed. We enforce no further constraints looking at edges in the middle of the spine. This means that left and right arc operations can add spinal edges in a free manner, without explicitly encoding how these edges relate to each other. In other words, we rely on the statistical model to correctly build a spine by adding left and right spinal edges along the transition process in a bottom-up fashion.

It is easy to see that these constraints do not prevent the transition process from ending. Specifically, even though the constraints invalidate arc operations, the arc-eager process can always finish by leaving tokens in the buffer without any head assigned, in which case the resulting derivation is a forest of several projective trees.

3.3 Mapping Derivations to Spinal Trees

The constrained arc-eager derivations correspond to spinal structures, but not necessarily to single spinal trees, for two reasons. First, from the derivation we can extract two subsequences of left and right spinal edges, but the derivation does not encode how these sequences should merge into a spine. Second, as in the basic arc-eager process, the derivation might be a forest rather than a single tree. Next we describe processes to turn a spinal arc-eager derivation into a tree.

Forming spines. For each token i we depart from the top of the spine t, a sequence L of left spinal edges, and a sequence R of right spinal edges. The goal is to form a spine σ_i, such that its top is t, and that L and R are subsequences with repeats of the edges of σ_i. We look for the shortest spine satisfying these properties. For example, consider the derivation in Figure 3 and the third token *has*:

- Top t: S
- Left edges L: VP $-$ S
- Right edges R: $\star$$-$VP, VP $-$ S

In this case the shortest spine that is consistent with the edges and the top is $\star{-}VP-S$. Our method runs in two steps:

1. Collapse. Traverse each sequence of edges and replace any contiguous subsequence of identical edges by a single occurrence. The assumption is that identical contiguous edges correspond to sibling dependencies that attach to the same node in the spine.[2]

2. Merge the left L and right R sequences of edges overlapping them as much as possible, i.e. looking for the shortest spine. We do this in $\mathcal{O}(nm)$, where n and m are the lengths of the two sequences. Whenever multiple shortest spines are compatible with the left and right edge sequences, we give preference to the spine that places left edges to the bottom.

The result of this process is a spine σ_i with left and right dependents attached to positions of the spine. Note that this strategy has some limitations: (a) it can not recover non-terminal spinal nodes that do not participate in any triplet; and (b) it flattens spinal structures that involve contiguous identical spinal edges. [3]

Rooting Forests. The arc-eager transition system is not guaranteed to generate a single root in a derivation (though see (Nivre and Fernández-González, 2014) for a solution). Thus, after mapping a derivation to a spinal structure, we might get a forest of projective spinal trees. In this case, to produce a constituent tree from the spinal forest, we promote the last tree and place the rest of trees as children of its top node.

4 Experiments

In this section we describe the performance of the transition-based spinal parser by running it with different sizes of the beam and by comparing it with the state-of-the-art. We used the ZPar implementation modified to incorporate the constraints for spinal arc-eager parsing. We used the exact same features as Zhang and Nivre (2011), which extract a rich set of features that encode higher-order interactions betwen the current action and elements of the stack. Since our dependency labels are constituent triplets, these features encode a mix of constituent and dependency structure.

4.1 Data

We use the WSJ portion of the Penn Treebank[4], augmented with head-dependant information using the rules of Yamada and Matsumoto (2003). This results in a total of 974 different constituent triplets, which we use as dependency labels in the spinal arc-eager model. We use predicted part-of-speech tags[5].

4.2 Results in the Development Set

In Table 1 we show the results of our parser for the dependency trees, the table shows unlabeled attachment score (UAS) , triplet accuracy (TA, which would be label accuracy, LA) and triplet attachment score (TAS), and spinal accuracy (SA) (the spinal accuracy is the percentage of complete spines that the parser correctly predicts). In order to be fully comparable, for the dependency-based metrics we report results including and excluding punctuation symbols for evaluation. The table also shows the speed (sentences per second) in standard hardware. We trained the parser with different beam values, we run a number of iterations until the model converges and we report the results of the best iteration.

As it can be observed the best model is the one trained with beam size 64, and greater sizes of the beam help to improve the results. Nonetheless, it also makes the parser slower. This result is expected since the number of dependency labels, i.e. triplets, is 974 so a higher size of the beam allows to test more of them when new actions are included in the agenda. This model already provides high results over 92.34% UAS and it can also predict most of the triplets that label the dependency

[2] However, this is not always the case. For example, in the Penn Treebank adjuncts create an additional constituent level in the verb-phrase structure, and this can result in a series of contiguous VP spinal nodes. The effect of flattening such structures is mild, see below.

[3] These limitations have relatively mild effects on recovering constituent trees in the style of the Penn Treebank. To measure the effect, we took the correct spinal trees of the development section and mapped them to the corresponding arc-eager derivation. Then we mapped the derivation back to a spinal tree using this process and recovered the constituent tree. This process obtained 98.4% of bracketing recall, 99.5% of bracketing precision, and 99.0 of F_1 measure.

[4] We use the standard partition: sections 02-21 for training, section 22 for development, and section 23 for testing.

[5] We use the same setting as in (Carreras et al., 2008) by training over a treebank with predicted part-of-speech tags with mxpost (Ratnaparkhi, 1996) (accuracy: 96.5) and we test on the development set and test set with predicted part-of-speech tags of Collins (1997) (accuracy: 96.8).

	Dep. (incl punct)				Dep. (excl punct)			Const			Speed
Beam-size	UAS	TA	TAS	SA	UAS	TA	TAS	LR	LP	F1	Sent/Sec
8	91.39	90.47	88.78	95.60	92.32	91.21	89.73	88.6	88.4	88.5	7.8
16	91.81	90.95	89.28	95.84	92.70	91.65	90.21	89.0	89.1	89.1	3.9
32	92.08	91.14	89.52	95.96	92.91	91.77	90.38	89.4	89.5	89.5	1.7
64	92.34	91.45	89.84	96.13	93.12	92.04	90.65	89.5	89.7	89.7	0.8

Table 1: UAS with predicted part-of-speech tags for the dev.set including and excluding punctuation symbols. Constituent results for the development set. Parsing speed in sentences per second (an estimate that varies depending on the machine). TA and TAS refer to label accuracy and labeled attachment score where the labels are the different constituent triplets described in Section 3. SA is the spinal accuracy.

arcs (91.45 TA and 89.84 TAS) (including punctuation symbols for evaluation).

Table 1 also shows the results of the parser in the development set after transforming the dependency trees by following the method described in Section 3. The result even surpasses 89.5% F1 which is a competitive accuracy. As we can see, the parser also provides a good trade-off between parsing speed and accuracy.[6]

In order to test whether the number of dependency labels is an issue for the parser, we also trained a model on dependency trees labeled with Yamada and Matsumoto (2003) rules, and the results are comparable to ours. For a beam of size 64, the best model with dependency labels provides 92.3% UAS for the development set including punctuation and 93.0% excluding punctuation, while our spinal parser for the same beam size provides 92.3% UAS including punctuation and 93.1% excluding punctuation. This means that the beam-search arc-eager parser is capable of coping with the dependency triplets, since it even provides slightly better results for unlabeled attachment scores. However, unlike (Carreras et al., 2008), the arc-eager parser does not substantially benefit of using the triplets during training.

4.3 Final Results and State-of-the-art Comparison

Our best model (obtained with beam=64) provides 92.14 UAS, 90.91 TA and 89.32 TAS in the test set including punctuation and 92.78 UAS, 91.53 TA and 90.11 TAS excluding punctuation. Table 2 compares our results with the state-of-the-art. Our model obtains comptetitive dependency accuracies when compared to other systems.

Parser	UAS
McDonald et al. (2005)	90.9
McDonald and Pereira (2006)	91.5
Huang and Sagae (2010)	92.1
Zhang and Nivre (2011)	92.9
Koo and Collins (2010)*	93.0
Bohnet and Nivre (2012)	93.0
Koo et al. (2008) †*	93.2
Martins et al. (2010)	93.3
Ballesteros and Bohnet (2014)	93.5
Carreras et al. (2008) †*	93.5
Suzuki et al. (2009) †*	93.8
this work (beam 64) †*	92.1
this work (beam 64) †	92.8

Table 2: State-of-the-art comparison for unlabeled attachment score for WSJ-PTB with Y&M rules. Results marked with † use other kind of information, and are not directly comparable. Results marked with * include punctuation for evaluation.

In terms of constituent structure, our best model (beam=64) obtains 88.74 LR, 89.21 LP and 88.97 F1. Table 3 compares our model with other constituent parsers, including shift-reduce parsers as ours. Our best model is competitive compared with the rest.

5 Related Work

Collins (1996) defined a statistical model for dependency parsing based on using constituent triplets in the labels, which forms the basis of our arc-eager model. In that work, a chart-based algorithm was used for parsing, while here we use greedy transition-based parsing.

[6] However, in absolute terms, our running times are slower than typical shift-reduce parsers. Our purpose is to show a relation between speed and accuracy, and we opted for a simple implementation rather than an engineered one. As one example, our parser considers all dependency triplets (974) in all cases, which is somehow absurd since most of these can be ruled out given the parts-of-speech of the candidate dependency. Incorporating a filtering strategy of this kind would result in a speedup factor constant to all beam sizes.

Beam-size	LR	LP	F1
Sagae and Lavie (2005)*	86.1	86.0	86.0
Ratnaparkhi (1999)	86.3	87.5	86.9
Sagae and Lavie (2006)*	87.8	88.1	87.9
Collins (1999)	88.1	88.3	88.2
Charniak (2000)	89.5	89.9	89.5
Zhang and Clark (2009)*	90.0	89.9	89.9
Petrov and Klein (2007)	90.1	90.2	90.1
Zhu et al. (2013)-1*	90.2	90.7	90.4
Carreras et al. (2008)	90.7	91.4	91.1
Zhu et al. (2013)-2†*	91.1	91.5	91.3
Huang (2008)	91.2	91.8	91.5
Charniak (2000)	91.2	91.8	91.5
Huang et al. (2010)	91.2	91.8	91.5
McClosky et al. (2006)	91.2	91.8	91.5
this work (beam 64)*	88.7	89.2	89.0

Table 3: State-of-the-art comparison in the test set for phrase structure parsing. Results marked with † use additional information, such as semi-supervised models, and are not directly comparable to the others. Results marked with * are shift-reduce parsers.

Carreras et al. (2008) was the first to use spinal representations to define an arc-factored dependency parsing model based on the Eisner algorithm, that parses in cubic time. Our work can be seen as the transition-based counterpart of that, with a greedy parsing strategy that runs in linear time. Because of the extra complexity of spinal structures, they used three probabilistic non-spinal dependency models to prune the search space of the spinal model. In our work, we show that a single arc-eager model can obtain very competitive results, even though the accuracies of our model are lower than theirs.

In terms of parsing spinal structures, Rush et al. (2010) introduced a dual decomposition method that uses constituent and dependency parsing routines to parse a combined spinal structure.

In a similar style to our method Hall et al. (2007), Hall and Nivre (2008) and Hall (2008) introduced an approach for parsing Swedish and German, in which MaltParser (Nivre et al., 2007) is used to predict dependency trees, whose dependency labels are enriched with constituency labels. They used tuples that encode dependency labels, constituent labels, head relations and the attachment. The last step is to make the inverse transformation from a dependency graph to a constituent structure.

Recently Kong et al. (2015) proposed a structured prediction model for mapping dependency trees to constituent trees, using the CKY algorithm. They assume a fixed dependency tree used as a hard constraint. Also recently, Fernández-González and Martins (2015) proposed an arc-factored dependency model for constituent parsing. In that work dependency labels encode the constituent node where the dependency arises as well as the position index of that node in the head spine. In contrast, we use constituent triplets as dependency labels.

Our method is based on constraining a shift-reduce parser using the arc-eager strategy. Nivre (2003) and Nivre (2004) establish the basis for arc-eager algorithm and arc-standard parsing algorithms, which are central to most recent transition-based parsers (Zhang and Clark, 2011b; Zhang and Nivre, 2011; Bohnet and Nivre, 2012). These parsers are very fast, because the number of parsing actions is linear in the length of the sentence, and they obtain state-of-the-art-performance, as shown in Section 4.3.

For shift-reduce constituent parsing, Sagae and Lavie (2005; 2006) presented a shift-reduce phrase structure parser. The main difference to ours is that their models do not use lexical dependencies. Zhang and Clark (2011a) presented a shift-reduce parser based on CCG, and as such is lexicalized. Both spinal and CCG representations are very expressive. One difference is that spinal trees can be directly obtained from constituent treebanks with head-child information, while CCG derivations are harder to obtain.

More recently, Zhang and Clark (2009) and the subsequent work of Zhu et al. (2013) described a beam-search shift-reduce parsers obtaining very high results. These models use dependency information via stacking, by running a dependency parser as a preprocess. In the literature, stacking is a common technique to improve accuracies by combining dependency and constituent information, in both ways (Wang and Zong, 2011; Farkas and Bohnet, 2012). Our model differs from stacking approaches in that it natively produces the two structures jointly, in such a way that a rich set of features is available.

6 Conclusions and Future Work

There are several lessons to learn from this paper. First, we show that a simple modification to the arc-eager strategy results in a competitive greedy spinal parser which is capable of predicting dependency and constituent structure jointly. In order to make it work, we introduce simple constraints to the arc-eager strategy that ensure well-formed spinal derivations. Second, by doing this, we are providing a good trade-off between speed and accuracy, while at the same time we are providing a dependency structure which can be really useful for downstream applications. Even if the dependency model needs to cope with a huge amount of dependency labels (in the form of constituent triplets), the unlabeled attachment accuracy does not drop and the labeling accuracy (for the triplets) is good enough for getting a good phrase-structure parse. Overall, our work shows that greedy strategies to dependency parsing can be successfuly augmented to include constituent structure.

In the future, we plan to explore spinal derivations in new transition-based dependency parsers (Chen and Manning, 2014; Dyer et al., 2015; Weiss et al., 2015; Zhou et al., 2015). This would allow to explore the spinal derivations in new ways and to test their potentialities.

Acknowledgements

We thank Noah A. Smith and the anonymous reviewers for their very useful comments. Miguel Ballesteros is supported by the European Commission under the contract numbers FP7-ICT-610411 (project MULTISENSOR) and H2020-RIA-645012 (project KRISTINA).

References

Miguel Ballesteros and Bernd Bohnet. 2014. Automatic feature selection for agenda-based dependency parsing. In *Proc. COLING*.

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*, pages 1455–1465, Jeju Island, Korea, July. Association for Computational Linguistics.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-Rich Parsing. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 9–16, Manchester, England, August. Coling 2008 Organizing Committee.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, NAACL 2000, pages 132–139, Stroudsburg, PA, USA. Association for Computational Linguistics.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. EMNLP*.

Michael John Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191, Santa Cruz, California, USA, June. Association for Computational Linguistics.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

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 ACL*.

Richárd Farkas and Bernd Bohnet. 2012. Stacking of dependency and phrase structure parsers. In *COLING*, pages 849–866.

Daniel Fernández-González and André F. T. Martins. 2015. Parsing as reduction. *CoRR*, abs/1503.00030.

Johan Hall and Joakim Nivre. 2008. A dependency-driven parser for german dependency and constituency representations. In *Proceedings of the Workshop on Parsing German*, PaGe '08, pages 47–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Johan Hall, Joakim Nivre, and Jens Nilsson. 2007. A Hybrid Constituency-Dependency Parser for Swedish. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA)*.

Johan Hall. 2008. Transition-based natural language parsing with dependency and constituency representations. Master's thesis, Växjö University.

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)*, pages 1077–1086.

Zhongqiang Huang, Mary Harper, and Slav Petrov. 2010. Self-training with products of latent variable grammars. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 12–22. Association for Computational Linguistics.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594.

Lingpeng Kong, Alexander M. Rush, and Noah A. Smith. 2015. Transforming dependencies into phrase structures. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies*. Association for Computational Linguistics.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 595–603.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan and Claypool.

Andre Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mario Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 34–44.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 152–159, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.

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

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135.

Joakim Nivre, Yoav Goldberg, and Ryan T. McDonald. 2014. Constrained arc-eager dependency parsing. *Computational Linguistics*, 40(2):249–527.

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 *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 50–57.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *HLT-NAACL*, volume 7, pages 404–411.

Adwait Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 125–132, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, pages 129–132, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of EMNLP*, pages 551–560.

Zhiguo Wang and Chengqing Zong. 2011. Parse reranking based on higher-order lexical dependencies. In *IJCNLP*, pages 1251–1259.

David Weiss, Christopher Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proc. ACL*.

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. 2009. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 162–171. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2011a. Shift-reduce ccg parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 683–692. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2011b. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

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 (ACL)*, pages 188–193, Portland, Oregon, USA.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A Neural Probabilistic Structured-Prediction Model for Transition-Based Dependency Parsing. In *ACL*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443. Association for Computational Linguistics.