Transition-based Semantic Dependency Parsing with Pointer Networks

Daniel Fernández-González and **Carlos Gómez-Rodríguez**
Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Group
Depto. de Ciencias de la Computación y Tecnologías de la Información
Campus de Elviña, s/n, 15071 A Coruña, Spain
d.fgonzalez@udc.es, carlos.gomez@udc.es

Abstract

Transition-based parsers implemented with Pointer Networks have become the new state of the art in dependency parsing, excelling in producing labelled syntactic trees and outperforming graph-based models in this task. In order to further test the capabilities of these powerful neural networks on a harder NLP problem, we propose a transition system that, thanks to Pointer Networks, can straightforwardly produce labelled directed acyclic graphs and perform semantic dependency parsing. In addition, we enhance our approach with deep contextualized word embeddings extracted from BERT. The resulting system not only outperforms all existing transition-based models, but also matches the best fully-supervised accuracy to date on the SemEval 2015 Task 18 English datasets among previous state-of-the-art graph-based parsers.

1 Introduction

In *dependency parsing*, the syntactic structure of a sentence is represented by means of a labelled tree, where each word is forced to be attached exclusively to another that acts as its head. In contrast, *semantic dependency parsing* (SDP) (Oepen et al., 2014) aims to represent binary predicate-argument relations between words of a sentence, which requires producing a labelled directed acyclic graph (DAG): not only semantic predicates can have multiple or zero arguments, but words from the sentence can be attached as arguments to more than one head word (predicate), or they can be outside the SDP graph (being neither a predicate nor an argument) as shown in the examples in Figure 1. Since existing dependency parsers cannot be directly applied, most SDP research has focused on adapting them to deal with the absence of single-head and connectedness constraints and to produce an SDP graph instead.

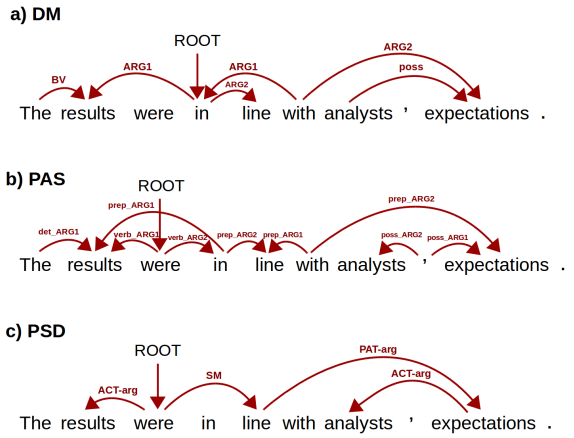

Figure 1: Sentence from the SemEval 2015 Task 18 development set parsed with semantic dependencies following the DM, PAS and PSD formalisms.

As in dependency parsing, we can find two main families of approaches to efficiently generate accurate SDP graphs. On the one hand, *graph-based* algorithms have drawn more attention since adapting them to this task is relatively straightforward. In particular, these globally optimized methods independently score arcs (or sets of them) and then search for a high-scoring graph by combining these scores. From one of the first graph-based DAG parsers proposed by McDonald and Pereira (2006) to the current state-of-the-art models (Wang et al., 2019; He and Choi, 2019), different graph-based SDP approaches have been presented, providing accuracies above their main competitors: *transition-based* DAG algorithms.

A transition-based parser generates a sequence of actions to incrementally build a valid graph (usually from left to right). This is typically done by local, greedy prediction and can efficiently parse a sentence in a linear or quadratic number of actions (transitions); however, the lack of global inference makes them more prone to suffer from error propa-

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7035–7046
July 5 - 10, 2020.

gation: *i.e.*, since transitions are sequentially and locally predicted, an erroneous action can affect future predictions, having a significant impact in long sentences and being, to date, less appealing for SDP. In fact, in recent years only a few contributions, such as the system developed by Wang et al. (2018), present a purely transition-based SDP parser. It is more common to find hybrid systems that combine transition-based approaches with graph-based techniques to alleviate the impact of error propagation in accuracy (Du et al., 2015), but this penalizes the efficiency provided by transition-based algorithms.

Away from the current mainstream, we present a purely transition-based parser that directly generates SDP graphs without the need of any additional techniques. We rely on Pointer Networks (Vinyals et al., 2015) to predict transitions that can attach multiple heads to the same word and incrementally build a labelled DAG. This kind of neural networks provide an encoder-decoder architecture that is capable of capturing information from the whole sentence and previously created arcs, alleviating the impact of error propagation and already showing remarkable results in transition-based dependency parsing (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019). We further enhance our neural network with deep contextualized word embeddings extracted from the pre-trained language model BERT (Devlin et al., 2019).

The proposed SDP parser[1] can process sentences in SDP treebanks (where structures are sparse DAGs with a low in-degree) in $O(n^2 log\ n)$ time, or $O(n^2)$ without cycle detection. This is more efficient than the current fully-supervised state-of-the-art system by Wang et al. (2019) ($O(n^3)$ without cycle detection), while matching its accuracy on the SemEval 2015 Task 18 datasets (Oepen et al., 2015). In addition, we also prove that our novel transition-based model provides promising accuracies in the semi-supervised scenario, achieving some state-of-the-art results.

2 Related Work

An early approach to DAG parsing was implemented as a modification to a graph-based parser by McDonald and Pereira (2006). This produced DAGs using approximate inference by first finding a dependency tree, and then adding extra edges that would increase the graph's overall score. A few years later, this attempt was outperformed by the first transition-based DAG parser by Sagae and Tsujii (2008). They extended the existing transition system by Nivre (2003) to allow multiple heads per token. The resulting algorithm was not able to produce DAGs with crossing dependencies, requiring the pseudo-projective transformation by Nivre and Nilsson (2005) (plus a cycle removal procedure) as a post-processing stage.

More recently, there has been a predominance of purely graph-based DAG models since the SemEval 2015 Task 18 (Oepen et al., 2015). Almeida and Martins (2015) adapted the pre-deep-learning dependency parser by Martins et al. (2013) to produce SDP graphs. This graph-based parser encodes higher-order information with hand-crafted features and employs the AD^3 algorithm (Martins et al., 2011) to find valid DAGs during decoding. This was extended by Peng et al. (2017) with BiLSTM-based feature extraction and multitask learning: the three formalisms considered in the shared task were jointly learned to improve final accuracy.

After the success of Dozat et al. (2017) in graph-based dependency parsing, Dozat and Manning (2018) proposed minor adaptations to use this biaffine neural architecture to produce SDP graphs. To that end, they removed the maximum spanning tree algorithm (Chu and Liu, 1965; Edmonds, 1967) necessary for decoding well-formed dependency trees and simply kept those edges with a positive score. In addition, they trained the unlabelled parser with a sigmoid cross-entropy (instead of the original softmax one) in order to accept multiple heads.

The parser by Dozat and Manning (2018) was recently improved by two contributions. Firstly, Wang et al. (2019) manage to add second-order information for score computation and then apply either mean field variational inference or loopy belief propagation information to decode the highest-scoring SDP graph. While significantly boosting parsing accuracy, the original $O(n^2)$ runtime complexity is modified to $O(n^3)$ in the resulting SDP system. Secondly, He and Choi (2019) significantly improve the original parser's accuracy by not only using contextualized word embeddings extracted from BERT (Devlin et al., 2019), but also introducing contextual string embeddings (called Flair) (Akbik et al., 2018), which consist in a novel type of word vector representations based on character-

[1]Source code available at https://github.com/danifg/SemanticPointer.

level language modeling. Both extensions, (Wang et al., 2019) and (He and Choi, 2019), are currently the state of the art on the SemEval 2015 Task 18 in the fully-supervised and semi-supervised scenarios, respectively.

Kurita and Søgaard (2019) have also recently proposed a complex approach that iteratively applies the syntactic dependency parser by Zhang et al. (2017), sequentially building a DAG structure. At each iteration, the graph-based parser selects the highest-scoring arcs, keeping the single-head constraint. The process ends when no arcs are added in the last iteration. The combination of partial parses results in an SDP graph. Since the graph is built in a sequential process, they use reinforcement learning to guide the model through more optimal paths. Following Peng et al. (2017), multi-task learning is also added to boost final accuracy.

On the other hand, the use of transition-based algorithms in the SDP task had been less explored until very recently. Du et al. (2015) presented a voting-based ensemble of fourteen graph- and transition-based parsers. In their work, they noticed that individual graph-based models outperform transition-based algorithms, assigning, during voting, higher weights to them. Among the transition systems used, we can find the one developed by Titov et al. (2009), which is not able to cover all SDP graphs.

We have to wait until the work by Wang et al. (2018) to see that a purely transition-based SDP parser (enhanced with a simple model ensemble technique) can achieve competitive results. They simply modified the preconditions of the complex transition system by Choi and McCallum (2013) to produce unrestricted DAG structures. In addition, their system was implemented by means of stack-LSTMs (Dyer et al., 2015), enhanced with BiLSTMs and Tree-LSTMs for feature extraction.

We are, to the best of our knowledge, first to explore DAG parsing with Pointer Networks, proposing a purely transition-based algorithm that can be a competitive alternative to graph-based SDP models.

Finally, during the reviewing process of this work, the proceedings of the CoNLL 2019 shared task (Oepen et al., 2019) were released. In that event, SDP parsers were evaluated on updated versions of SemEval 2015 Task 18 datasets, as well as on datasets in other semantic formalisms such as Abstract Meaning Representation (AMR) (Banarescu et al., 2013) and Universal Cognitive Conceptual Annotation (UCCA) (Abend and Rappoport, 2013). Although graph-based parsers achieved better accuracy in the SDP track, several BERT-enhanced transition-based approaches were proposed. Among them we can find an extension (Che et al., 2019) of the system by Wang et al. (2018), several adaptations for SDP (Hershcovich and Arviv, 2019; Bai and Zhao, 2019) of the transition-based UCCA parser by Hershcovich et al. (2017), as well as an SDP variant (Lai et al., 2019) of the constituent transition system introduced by Fernández-González and Gómez-Rodríguez (2019). Also in parallel to the development of this research, Zhang et al. (2019) proposed a transition-based parser that, while it can be applied for SDP, was specifically designed for AMR and UCCA parsing (where graph nodes do not correspond with words and must be generated during the parsing process). In particular, this approach incrementally builds a graph by predicting at each step a semantic relation composed of the target and source nodes plus the arc label. While this can be seen as an extension of our approach for those tasks where nodes must be generated, its complexity penalizes accuracy in the SDP task.

3 Multi-head Transition System

We design a novel transition system that is able to straightforwardly attach multiple heads to each word in a single pass, incrementally building, from left to right, a valid SDP graph: a labelled DAG.

To implement it, we use Pointer Networks (Vinyals et al., 2015). These neural networks are able to learn the conditional probability of a sequence of discrete numbers that correspond to positions in an input sequence and, at decoding time, perform as a pointer that selects a position from the input. In other words, we can train this neural network to, given a word, point to the position of the sentence where its head (Fernández-González and Gómez-Rodríguez, 2019) or dependent words (Ma et al., 2018) are located, depending on what interpretation we use during training. In particular, (Fernández-González and Gómez-Rodríguez, 2019) proved to be more suitable for dependency parsing than (Ma et al., 2018) since it requires half as many steps to produce the same dependency parse, being not only faster, but also more accurate (as this mitigates the impact of error propagation).

Inspired by Fernández-González and Gómez-

Rodríguez (2019), we train a Pointer Network to point to the head of a given word and propose an algorithm that does not use any kind of data structures (stack or buffer, required in classic transition-based parsers (Nivre, 2008)), but just a *focus word pointer i* for marking the word currently being processed. More in detail, given an input sentence of n words $w_1, \ldots, w_n$, the parsing process starts with i pointing at the first word w_1. At each time step, the current focus word w_i is used by the Pointer Network to return a position p from the input sentence (or 0, where the ROOT node is located). This information is used to choose between the two available transitions:

- If $p \neq i$, then the pointed word w_p is considered as a semantic head word (predicate) of w_i and an Attach-p transition is applied, creating the directed arc $w_p \rightarrow w_i$. The Attach-p transition is only permissible if the resulting predicate-argument arc neither exists nor generates a cycle in the already-built graph, in order to output a valid DAG.

- On the contrary, if $p = i$ (*i.e.*, the model points to the current focus word), then w_i is considered to have found all its head words, and a Shift transition is chosen to move i one position to the right to process the next word w_{i+1}.

The parsing ends when the last word from the sentence is shifted, meaning that the input is completely processed. As stated by Ma et al. (2018) for attaching dependent words, it is necessary to fix the order in which (in our case, head) words are assigned in order to define a deterministic decoding. As the sentence is parsed in a left-to-right manner, we adopt the same order for head assignments. For instance, the SDP graph in Figure 1(a) is produced by the transition sequence described in Table 1. We just need n Shift transitions to move the focus word pointer through the whole sentence and m Attach-p transitions to create the m arcs present in the SDP graph.

It is worth mentioning that we manage to significantly reduce the amount of transitions necessary for generating DAGs in comparison to those proposed in the complex transition systems by Choi and McCallum (2013) and Titov et al. (2009), used in the SDP systems by Wang et al. (2018) and Du et al. (2015), respectively. In addition, the described multi-head transition system is able to directly produce any DAG structure without exception, while some transition systems, such as the mentioned (Sagae and Tsujii, 2008; Titov et al., 2009), are limited to a subset of DAGs.

p	transition	focus word$_i$	added arc
		The$_1$	
1	Shift	results$_2$	
1	Attach-1	results$_2$	The$_1$ $\rightarrow$ results$_2$
4	Attach-4	results$_2$	results$_2$ $\leftarrow$ in$_4$
2	Shift	were$_3$	
3	Shift	in$_4$	
0	Attach-0	in$_4$	ROOT$_0$ $\rightarrow$ in$_4$
6	Attach-6	in$_4$	in$_4$ $\leftarrow$ with$_6$
4	Shift	line$_5$	
4	Attach-4	line$_5$	in$_4$ $\rightarrow$ line$_5$
5	Shift	with$_6$	
6	Shift	analysts$_7$	
7	Shift	'$_8$	
8	Shift	expectations$_9$	
6	Attach-6	expectations$_9$	with$_6$ $\rightarrow$ expectations$_9$
7	Attach-7	expectations$_9$	analysts$_7$ $\rightarrow$ expectations$_9$
9	Shift	.$_{10}$	
10	Shift		

Table 1: Transition sequence for generating the SDP graph in Figure 1(a).

Finally, while the outcome of the proposed transition system is a SDP graph without cycles, in other research, such as (Kurita and Søgaard, 2019) and state-of-the-art models by Dozat and Manning (2018) and Wang et al. (2019), the parser is not forced to produce well-formed DAGs, allowing the presence of cycles.

4 Neural Network Architecture

4.1 Basic Approach

Vinyals et al. (2015) introduced an encoder-decoder architecture, called *Pointer Network*, that uses a mechanism of neural attention (Bahdanau et al., 2014) to select positions from the input sequence, without requiring a fixed size of the output dictionary. This allows Pointer Networks to easily address those problems where the target classes considered at each step are variable and depend on the length of the input sequence. We prove that implementing the transition system previously defined on this neural network results in an accurate SDP system.

We follow previous work in dependency parsing (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019) to design our neural architecture:

Encoder A BiLSTM-CNN architecture (Ma and Hovy, 2016) is used to encode the input sentence

$w_1, \ldots, w_n$, word by word, into a sequence of *encoder hidden states* $\mathbf{h}_1, \ldots, \mathbf{h}_n$. CNNs with max pooling are used for extracting character-level representations of words and, then, each word w_i is represented by the concatenation of character ($\mathbf{e}_i^c$), word ($\mathbf{e}_i^w$), lemma ($\mathbf{e}_i^l$) and POS tag ($\mathbf{e}_i^p$) embeddings:

$$\mathbf{x}_i = \mathbf{e}_i^c \oplus \mathbf{e}_i^w \oplus \mathbf{e}_i^l \oplus \mathbf{e}_i^p$$

After that, the $\mathbf{x}_i$ of each word w_i is fed one-by-one into a BiLSTM that captures context information in both directions and generates a vector representation $\mathbf{h}_i$:

$$\mathbf{h}_i = \mathbf{h}_{li} \oplus \mathbf{h}_{ri} = \mathbf{BiLSTM}(\mathbf{x}_i)$$

In addition, a special vector representation $\mathbf{h}_0$, denoting the ROOT node, is prepended at the beginning of the sequence of encoder hidden states.

Decoder An LSTM is used to output, at each time step t, a *decoder hidden state* $\mathbf{s}_t$. As input of the decoder, we use the encoder hidden state $\mathbf{h}_i$ of the current focus word w_i plus extra high-order features. In particular, we take into account the hidden state of the last head word ($\mathbf{h}_h$) attached to w_i, which will be a co-parent of a future predicate assigned to w_i. Following Ma et al. (2018), we use element-wise sum to add this information without increasing the dimensionality of the input:

$$\mathbf{r}_i = \mathbf{h}_i + \mathbf{h}_h;\ \mathbf{s}_t = \mathbf{LSTM}(\mathbf{r}_i)$$

Note that feature information like this can be easily added in transition-based models without increasing the parser's runtime complexity, something that does not happen in graph-based models, where, for instance, the second-order features added by Wang et al. (2019) penalize runtime complexity.

We experimented with other high-order features such as grandparent or sibling information of the current focus word w_i, but no significant improvements were obtained from their addition, so they were discarded for simplicity. Further feature exploration might improve parser performance, but we leave this for future work.

Once $\mathbf{s}_t$ is generated, the *attention vector* $\mathbf{a}^t$, which will work as a pointer over the input, must be computed in the *pointer layer*. First, following the previously cited work, the scores between $\mathbf{s}_t$ and each encoder hidden representation $\mathbf{h}_j$ from the input sentence are computed using this *biaffine attention* scoring function (Dozat and Manning, 2017):

$$\mathbf{v}_j^t = \mathbf{score}(\mathbf{s}_t, \mathbf{h}_j) = f_1(\mathbf{s}_t)^T W f_2(\mathbf{h}_j) + \mathbf{U}^T f_1(\mathbf{s}_t) + \mathbf{V}^T f_2(\mathbf{h}_j) + \mathbf{b}$$

where parameter W is the weight matrix of the bilinear term, $\mathbf{U}$ and $\mathbf{V}$ are the weight tensors of the linear terms and $\mathbf{b}$ is the bias vector. In addition, $f_1(\cdot)$ and $f_2(\cdot)$ are two single-layer multilayer perceptrons (MLP) with ELU activation, proposed by (Dozat and Manning, 2017) for reducing dimensionality and minimizing overfitting.

Then, a softmax is applied on the resulting score vector $\mathbf{v}^t$ to compute a probability distribution over the input words:

$$\mathbf{a}^t = \mathbf{softmax}(\mathbf{v}^t)$$

The resulting attention vector $\mathbf{a}^t$ can now be used as a pointer to select the highest-scoring position p from the input. This information will be employed by the transition system to choose between the two available actions and create a predicate-argument relation between w_p and w_i (Attach-p) or move the focus word pointer to w_{i+1} (Shift). In case the chosen Attach-p is forbidden due to the acyclicity constraint, the next highest-scoring position in $\mathbf{a}^t$ is considered as output instead. Figure 2 depicts the neural architecture and the decoding procedure for the SDP structure in Figure 1(a).

Label prediction We jointly train a multi-class classifier that scores every label for each pair of words. This shares the same encoder and uses the same biaffine attention function as the pointer:

$$\mathbf{s}_{tp}^l = \mathbf{score}(\mathbf{s}_t, \mathbf{h}_p, l) = g_1(\mathbf{s}_t)^T W_l g_2(\mathbf{h}_p) + \mathbf{U}_l^T g_1(\mathbf{s}_t) + \mathbf{V}_l^T g_2(\mathbf{h}_p) + \mathbf{b}_l$$

where a distinct weight matrix W_l, weight tensors $\mathbf{U}_l$ and $\mathbf{V}_l$ and bias $\mathbf{b}_l$ are used for each label l, where $l \in \{1, 2, \ldots, L\}$ and L is the number of labels. In addition, $g_1(\cdot)$ and $g_2(\cdot)$ are two single-layer MLPs with ELU activation.

The scoring function is applied over each predicted arc between the dependent word w_i (represented by $\mathbf{s}_t$) and the pointed head word w_p in position p (represented by $\mathbf{h}_p$) to compute the score of each possible label and assign the highest-scoring one.

Training Objectives The Pointer Network is trained to minimize the negative log likelihood

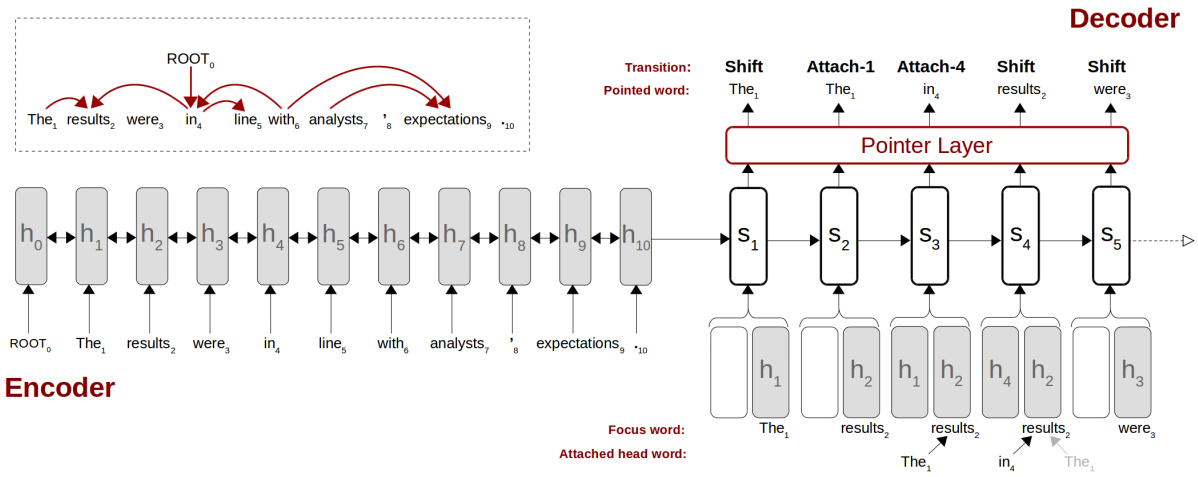

Figure 2: Neural network architecture and decoding steps to partially parse the SDP graph in Figure 1.

(implemented as cross-entropy loss) of producing the correct SDP graph y for a given sentence x: $P_\theta(y|x)$. Let y be a DAG for an input sentence x that is decomposed into a set of m directed arcs $a_1, \ldots, a_m$ following a left-to-right order. This probability can be factorized as follows:

$$P_\theta(y|x) = \prod_{k=1}^{m} P_\theta(a_k|a_{<k}, x)$$

where $a_{<k}$ denotes previous predicted arcs.

On the other hand, the labeler is trained with softmax cross-entropy to minimize the negative log likelihood of assigning the correct label l, given a dependency arc with head word w_h and dependent word w_i.

The whole neural model is jointly trained by summing the parser and labeler losses prior to computing the gradients. In that way, model parameters are learned to minimize the sum of the cross-entropy loss objectives over the whole corpus.

4.2 Deep Contextualized Word Embeddings Augmentation

In order to further improve the accuracy of our approach, we augment our model with deep contextualized word embeddings provided by the widely-used pre-trained language model BERT (Devlin et al., 2019).

Instead of including and training the whole BERT model as encoder of our system, we follow the common, greener and more cost-effective approach of leveraging the potential of BERT by extracting the weights of one or several layers as word-level embeddings. To that end, the pre-trained uncased $\text{BERT}_{\texttt{BASE}}$ model is used.

Since BERT is trained on *subwords* (*i.e.*, substrings of the original token), we take the 768-dimension vector of each subword of an input token and use the average embedding as the final representation $\mathbf{e}_i^{BERT}$. Finally, this is directly concatenated to the resulting basic word representation before feeding the BiLSTM-based encoder:

$$\mathbf{x}'_i = \mathbf{x}_i \oplus \mathbf{e}_i^{BERT};\ \mathbf{h}_i = \mathbf{BiLSTM}(\mathbf{x}'_i)$$

Higher performances can be achieved by summing or concatenating (depending on the task) several layers of BERT; however, exploring these combinations is out of the scope of this paper and we simply use embeddings extracted from the second-to-last hidden layer (since the last layer is biased to the target objectives used to train BERT's language model).

5 Experiments

5.1 Data

In order to test the proposed approach, we conduct experiments on the SemEval 2015 Task 18 English datasets (Oepen et al., 2015), where all sentences are annotated with three different formalisms: DELPH-IN MRS (DM) (Flickinger et al., 2012), Predicate-Argument Structure (PAS) (Miyao and Tsujii, 2004) and Prague Semantic Dependencies (PSD) (Hajič et al., 2012). Standard split as in previous work (Almeida and Martins, 2015; Du et al., 2015) results in 33,964 training sentences from Sections 00-19 of the Wall Street Journal corpus (Marcus et al., 1993), 1,692 development sentences from Section 20, 1,410 sentences from Section 21 as in-domain test set, and 1,849 sentences sampled from the Brown Corpus (Francis and Kucera,

Architecture hyper-parameters	
CNN window size	3
CNN number of filters	50
BiLSTM encoder layers	3
BiLSTM encoder size	512
LSTM decoder layers	1
LSTM decoder size	512
LSTM layers dropout	0.33
Word/POS/Char./Lemma embedding dimension	100
BERT embedding dimension	768
Embeddings dropout	0.33
MLP layers	1
MLP activation function	ELU
Arc MLP size	512
Label MLP size	128
UNK replacement probability	0.5
Adam optimizer hyper-parameters	
Initial learning rate	0.001
β_1, β_2	0.9
Batch size	32
Decay rate	0.75
Gradient clipping	5.0

Table 2: Model hyper-parameters.

1982) as out-of-domain test data. For the evaluation, we use the official script,[2] reporting labelled F-measure scores (LF1) (including ROOT arcs) on the in-domain (ID) and out-of-domain (OOD) test sets for each formalism as well as the macro-average over the three of them.

5.2 Settings

We use the Adam optimizer (Kingma and Ba, 2014) and follow (Ma et al., 2018; Dozat and Manning, 2017) for parameter optimization. We do not specifically perform hyper-parameter selection for SDP and just adopt those proposed by Ma et al. (2018) for syntactic dependency parsing (detailed in Table 2). For initializing word and lemma vectors, we use the pre-trained structured-skipgram embeddings developed by Ling et al. (2015). POS tag and character embeddings are randomly initialized and all embeddings (except the deep contextualized ones) are fine-tuned during training. Due to random initializations, we report average accuracy over 5 repetitions for each experiment. In addition, during a 500-epoch training, the model with the highest labelled F-score on the development set is chosen. Finally, while further beam-size exploration might improve accuracy, we use beam-search decoding with beam size 5 in all experiments.

[2] https://github.com/semantic-dependency-parsing/toolkit

5.3 Results and Discussion

Table 3 reports the accuracy obtained by state-of-the-art SDP parsers detailed in Section 2 in comparison to our approach. To perform a fair comparison, we group SDP systems in three blocks depending on the embeddings provided to the architecture: (1) just basic pre-trained word and POS tag embeddings, (2) character and pre-trained lemma embeddings augmentation and (3) pre-trained deep contextualized embeddings augmentation. As proved by these results, our approach outperforms all existing transition-based models and the widely-used approach by Dozat and Manning (2018) with or without character and lemma embeddings, and it is on par with the best graph-based SDP parser by (Wang et al., 2019) on average in the fully-supervised scenario.[3]

In addition, our model achieves the best fully-supervised accuracy to date on the PSD formalism, considered the hardest to parse. We hypothesize that this might be explained by the fact that the PSD formalism is the more tree-oriented (as pointed out by Oepen et al. (2015)) and presents a lower ratio of arcs per sentence, being more suitable for our transition-based approach.

In the semi-supervised scenario, BERT-based embeddings proved to be more beneficial for the out-of-domain data. In fact, while not being a fair comparison since we neither include contextual string embeddings (Flair) (Akbik et al., 2018) nor explore different BERT layer combinations, our new transition-based parser manages to outperform the state-of-the-art system by He and Choi (2019)[4] on average on the out-of-domain test set, obtaining a remarkable accuracy on the PSD formalism.

5.4 Complexity

Given a sentence with length n whose SDP graph has m arcs, the proposed transition system requires n Shift plus m Attach-p transitions to parse it. Therefore, since a DAG can have at most $\Theta(n^2)$ edges (as is also the case for general directed graphs), it could potentially need $O(n^2)$ transitions in the worst case. However, we prove that this does not happen in practice and real sentences can be

[3] It is common practice in the literature that systems that only use standard pre-trained word or lemma embeddings are classed as fully-supervised models, even though, strictly, they are not trained exclusively on the official training data.

[4] He and Choi (2019) do not specify in their paper the BERT layer configuration used for generating the word embeddings.

Parser	DM		PAS		PSD		Avg	
	ID	OOD	ID	OOD	ID	OOD	ID	OOD
Du et al. (2015) TbGb+Ens	89.1	81.8	91.3	87.2	75.7	73.3	85.3	80.8
Almeida and Martins (2015) Gb	88.2	81.8	90.9	86.9	76.4	74.8	85.2	81.2
Peng et al. (2017) Gb	89.4	84.5	92.2	88.3	77.6	75.3	86.4	82.7
Peng et al. (2017) Gb+MT	90.4	85.3	92.7	89.0	78.5	76.4	87.2	83.6
Wang et al. (2018) Tb	89.3	83.2	91.4	87.2	76.1	73.2	85.6	81.2
Wang et al. (2018) Tb+Ens	90.3	84.9	91.7	87.6	78.6	75.9	86.9	82.8
Dozat and Manning (2018) Gb	91.4	86.9	93.9	90.8	79.1	77.5	88.1	85.0
Kurita and Søgaard (2019) Gb	91.1	-	92.4	-	78.6	-	87.4	-
Kurita and Søgaard (2019) Gb+MT+RL	91.2	-	92.9	-	78.8	-	87.6	-
Wang et al. (2019) Gb	**93.0**	**88.4**	**94.3**	**91.5**	80.9	**78.9**	**89.4**	**86.3**
This work Tb	92.5	87.7	94.2	91.0	**81.0**	78.7	89.2	85.8
Dozat and Manning (2018) Gb**+char+lemma**	93.7	88.9	93.9	90.6	81.0	79.4	89.5	86.3
Kurita and Søgaard (2019) Gb+MT+RL**+lemma**	92.0	87.2	92.8	88.8	79.3	77.7	88.0	84.6
Wang et al. (2019) Gb**+char+lemma**	**94.0**	**89.7**	94.1	**91.3**	81.4	79.6	89.8	**86.9**
This work Tb**+char+lemma**	93.9	89.6	**94.2**	91.2	**81.8**	**79.8**	**90.0**	**86.9**
Zhang et al. (2019) Tb+char**+BERT$_{\text{LARGE}}$**	92.2	87.1	-	-	-	-	-	-
He and Choi (2019) Gb+lemma**+Flair+BERT$_{\text{BASE}}$**	**94.6**	90.8	**96.1**	**94.4**	**86.8**	79.5	**92.5**	88.2
This work Tb+char+lemma**+BERT$_{\text{BASE}}$**	94.4	**91.0**	95.1	93.4	82.6	**82.0**	90.7	**88.8**

Table 3: Accuracy comparison of state-of-the-art SDP parsers on the SemEval 2015 Task 18 datasets. Gb and Tb stand for graph- and transition-based models, $+char$ and $+lemma$ for augmentations with character-level and lemma embeddings, $+Flair$ and $+BERT_{\text{BASE|LARGE}}$ for augmentations with deep contextualized character-level and word-level embeddings, and, finally, $+MT$, $+RL$ and $+Ens$ for the application of multi-task, reinforcement learning and ensemble techniques.

parsed with $O(n)$ transitions instead.

Parsing complexity of a transition-based dependency parsing algorithm can be determined by the number of transitions performed with respect to the number of words in a sentence (Kübler et al., 2009). Therefore, we measure the transition sequence length predicted by the system to analyze every sentence from the development sets of the three available formalisms and depict the relation between them and sentence lengths. As shown in Figure 3, a linear behavior is observed in all cases, proving that the number of Attach-p transitions evaluated by the model at each step is considerably low (behaving practically like a constant).

This can be explained by the fact that, on average on the training set, the ratio of predicate-argument dependencies per word in a sentence is 0.79 in DM, 0.99 in PAS and 0.70 in PSD, meaning that the transition sequence necessary for parsing a given sentence will need no more Attach-p transitions than Shift ones (which are one per word in the sentence). It is true that one argument can be attached to more than one predicate; however, the amount of words unattached in the resulting DAG (singletons)[5] can be significant in some formalisms (as described graphically in Figure 1): on average on the training set, 23% of words per sentence in DM, 6% in PAS and 35% in PSD. In addition, edge density on non-singleton words, computed by Oepen et al. (2015) on the test sets, also backs the linear behavior shown in our experiments: 0.96 in DM, 1.02 in PAS and 1.01 in PSD for the in-domain set and 0.95 in DM, 1.02 in PAS and 0.99 in PSD for the out-of-domain data. In conclusion, we can state that, on the datasets tested, the proposed transition system executes $O(n)$ transitions.

To determine the runtime complexity of the implementation of the transition system, we need to consider the following: firstly, at each transition, the attention vector $\mathbf{a}^t$ needs to be computed, which means that each of the $O(n)$ transitions takes $O(n)$ time to run. Therefore, the overall time complexity of the parser, ignoring cycle detection, is $O(n^2)$. Note that this is in contrast to algorithms like (Wang et al., 2019), which takes cubic time even though it does not enforce acyclicity.

[5] A singleton is a word that has neither incoming nor outgoing edges.

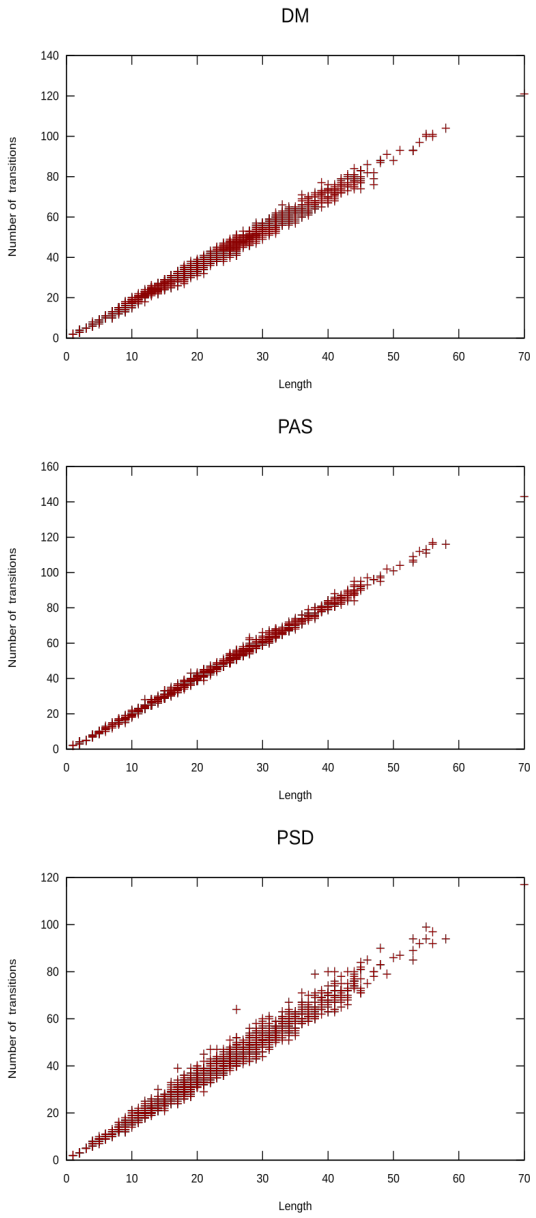

Figure 3: Number of predicted transitions relative to the length of the sentence, for the three SDP formalisms on the development set from SemEval 2015 Task 18.

If we add cycle detection, needed to forbid transitions that would create cycles and therefore to enforce that the output is a DAG, then the complexity becomes $O(n^2 log\ n)$. This is because an efficient implementation of cycle detection contributes an additive factor of $O(n^2 log\ n)$ to worst-case time complexity, which becomes the dominant factor. To achieve this efficient implementation, we incrementally keep two data structures: on the one hand, we keep track of weakly connected components using path compression and union by rank, which can be done in inverse Ackermann time, as is commonly done for cycle detection in tree and forest parsers (Covington, 2001; Gómez-Rodríguez and Nivre, 2010). On the other hand, we keep a weak topological numbering of the graph using the algorithm by Bender et al. (2015), which takes overall $O(n^2 log\ n)$ time over all edge insertions. When these two data structures are kept, cycles can be checked in constant time: an arc $a \rightarrow b$ creates a cycle if the involved nodes are in the same weakly connected component and a has a greater topological number than b.

Therefore, the overall expected worst-case running time of the proposed SDP system is $O(n^2 log\ n)$ for the range of data attested in the experiments, and can be lowered to $O(n^2)$ if we are willing to forgo enforcing acyclicity.

6 Conclusions and Future work

Our multi-head transition system can accurately parse a sentence in quadratic worst-case runtime thanks to Pointer Networks. While being more efficient, our approach outperforms the previous state-of-the-art parser by Dozat and Manning (2018) and matches the accuracy of the best model to date (Wang et al., 2019), proving that, with a state-of-the-art neural architecture, transition-based SDP parsers are a competitive alternative.

By adding BERT-based embeddings, we significantly improve our model accuracy by marginally affecting computational cost, achieving state-of-the-art F-scores in out-of-domain test sets.

Despite the promising results, the accuracy of our approach could probably be boosted further by experimenting with new feature information and specifically tuning hyper-parameters for the SDP task, as well as using different enhancements such as implementing the hierarchical decoding recently presented by Liu et al. (2019), including contextual string embeddings (Akbik et al., 2018) like He and Choi (2019), or applying multi-task learning across the three formalisms like Peng et al. (2017).

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia (ED431B 2017/01, ED431G 2019/01).

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (UCCA). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–238, Sofia, Bulgaria. Association for Computational Linguistics.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Mariana S. C. Almeida and André F. T. Martins. 2015. Lisbon: Evaluating TurboSemanticParser on multiple languages and out-of-domain data. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 970–973, Denver, Colorado. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Hongxiao Bai and Hai Zhao. 2019. SJTU at MRP 2019: A transition-based multi-task parser for cross-framework meaning representation parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 86–94, Hong Kong. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. 2015. A new approach to incremental cycle detection and related problems. *ACM Trans. Algorithms*, 12(2):14:1–14:22.

Wanxiang Che, Longxu Dou, Yang Xu, Yuxuan Wang, Yijia Liu, and Ting Liu. 2019. HIT-SCIR at MRP 2019: A unified pipeline for meaning representation parsing via efficient training and effective encoding. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 76–85, Hong Kong. Association for Computational Linguistics.

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)*, pages 1052–1062, Sofia, Bulgaria. Association for Computational Linguistics.

Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *ICLR*. OpenReview.net.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics.

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 927–931, Denver, Colorado. Association for Computational Linguistics.

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)*, pages 334–343, Beijing, China. Association for Computational Linguistics.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Faster shift-reduce constituent parsing with a non-binary, bottom-up strategy. *Artificial Intelligence*, 275:559 – 574.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Left-to-right dependency parsing with pointer networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, Volume 1 (Long and Short Papers)*, pages 710–716, Minneapolis, Minnesota. Association for Computational Linguistics.

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank : a dynamically annotated treebank of the wall street journal. In *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories (TLT11)*, pages 85–96, Lisbon. HU.

W.N. Francis and H. Kucera. 1982. *Frequency Analysis of English Usage: Lexicon and Usage*. Houghton Mifflin.

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

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing Prague Czech-English dependency treebank 2.0. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3153–3160, Istanbul, Turkey. European Language Resources Association (ELRA).

Han He and Jinho D. Choi. 2019. Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. *ArXiv*, abs/1908.04943.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1127–1138, Vancouver, Canada. Association for Computational Linguistics.

Daniel Hershcovich and Ofir Arviv. 2019. TUPA at MRP 2019: A multi-task baseline system. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 28–39, Hong Kong. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Sandra Kübler, Ryan T. McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Shuhei Kurita and Anders Søgaard. 2019. Multi-task semantic dependency parsing with policy gradient for learning easy-first strategies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2420–2430, Florence, Italy. Association for Computational Linguistics.

Sunny Lai, Chun Hei Lo, Kwong Sak Leung, and Yee Leung. 2019. CUHK at MRP 2019: Transition-based parser with cross-framework variable-arity resolve action. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 104–113, Hong Kong. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015. Two/too simple adaptations of Word2Vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1299–1304, Denver, Colorado. Association for Computational Linguistics.

Linlin Liu, Xiang Lin, Shafiq Joty, Simeng Han, and Lidong Bing. 2019. Hierarchical pointer net parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Hong Kong, China.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074. Association for Computational Linguistics.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard H. Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1403–1414.

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

André Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics.

André Martins, Noah Smith, Mário Figueiredo, and Pedro Aguiar. 2011. Dual decomposition with many overlapping components. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 238–249, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy. Association for Computational Linguistics.

Yusuke Miyao and Jun'ichi Tsujii. 2004. Deep linguistic analysis for the accurate identification of predicate-argument relations. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 1392–1398, Geneva, Switzerland. COLING.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160, Nancy, France.

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

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

Stephan Oepen, Omri Abend, Jan Hajic, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdenka Uresova. 2019. MRP 2019: Cross-framework meaning representation parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 1–27, Hong Kong. Association for Computational Linguistics.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926, Denver, Colorado. Association for Computational Linguistics.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72, Dublin, Ireland. Association for Computational Linguistics.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048, Vancouver, Canada. Association for Computational Linguistics.

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

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarization for synchronous parsing of semantic and syntactic dependencies. In *International Joint Conferences on Artificial Intelligence (IJCAI-09)*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4618, Florence, Italy. Association for Computational Linguistics.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A neural transition-based approach for semantic dependency graph parsing. In *Proc. AAAI*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. Broad-coverage semantic parsing as transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3786–3798, Hong Kong, China. Association for Computational Linguistics.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 665–676.