Global Greedy Dependency Parsing

Zuchao Li,1,2,3 Hai Zhao,1,2,3,* Kevin Parnow1,2,3

1Department of Computer Science and Engineering, Shanghai Jiao Tong University
2Key Laboratory of Shanghai Education Commission for Intelligent Interaction and Cognitive Engineering, Shanghai Jiao Tong University, Shanghai, China
3MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
{charlee, parnow}@sjtu.edu.cn, zhaohai@cs.sjtu.edu.cn

Abstract
Most syntactic dependency parsing models may fall into one of two categories: transition- and graph-based models. The former models enjoy high inference efficiency with linear time complexity, but they rely on the stacking or re-ranking of partially-built parse trees to build a complete parse tree and are stuck with slower training for the necessity of dynamic oracle training. The latter, graph-based models, may boast better performance but are unfortunately marred by polynomial time inference. In this paper, we propose a novel parsing order objective, resulting in a novel dependency parsing model capable of both global (in sentence scope) feature extraction as in graph models and linear time inference as in transitional models. The proposed global greedy parser only uses two arc-building actions, left and right arcs, for projective parsing. When equipped with two extra non-projective arc-building actions, the proposed parser may also smoothly support non-projective parsing. Using multiple benchmark treebanks, including the Penn Treebank (PTB), the CoNLL-X treebanks, and the Universal Dependency Treebanks, we evaluate our parser and demonstrate that the proposed novel parser achieves good performance with faster training and decoding.

1 Introduction
Dependency parsing predicts the existence and type of linguistic dependency relations between words (as shown in Figure 1), which is a critical step in accomplishing deep natural language processing. Dependency parsing has been well developed (Tratz and Hovy 2011; Li et al. 2018a), and it generally relies on two types of parsing models: transition-based models and graph-based models. The former (Yamada and Matsumoto 2003; Nivre and Scholz 2004; Zhang and Nivre 2011; Zhao et al. 2009; Zhang and Nivre 2011) traditionally apply local and greedy transition-based algorithms, while the latter (Eisner 1996; McDonald, Crammer, and Pereira 2005; McDonald et al. 2005; Ma and Zhao 2012; Zhang, Zhao, and Qin 2016; Li et al. 2018b; Li, Zhou, and Zhao 2019) apply globally optimized graph-based algorithms.

A transition-based dependency parser processes the sentence word-by-word, commonly from left to right, and forms a dependency tree incrementally from the operations predicted. This method is advantageous in that inference on the projective dependency tree is linear in time complexity with respect to sentence length; however, it has several obvious disadvantages. Because the decision-making of each step is based on partially-built parse trees, special training methods are required, which results in slow training and error propagation, as well as weak long-distance dependency processing (McDonald and Nivre 2011).

Graph-based parsers learn scoring functions in one-shot and then perform an exhaustive search over the entire tree space for the highest-scoring tree. This improves the performances of the parsers, particularly the long-distance dependency processing, but these models usually have slow inference speed to encourage higher accuracy.

The easy-first parsing approach (Kiperwasser and Goldberg 2016a; Li, Cai, and Zhao 2019) was designed to integrate the advantages of graph-based parsers’ better-performing trees and transition-based parsers’ linear decoding complexity. By processing the input tokens in a stepwise easy-to-hard order, the algorithm makes use of structured information on partially-built parse trees. Because of the presence of rich, structured information, exhaustive inference is not an optimal solution - we can leverage this information to conduct inference much more quickly. As an alternative to exhaustive inference,
easy-first chooses to use an approximated greedy search
that only explores a tiny fraction of the search space.
Compared to graph-based parsers, however, easy-first
parsers have two apparent weaknesses: slower training and
worse performance. According to our preliminary studies,
with the current state-of-the-art systems, we must either
sacrifice training complexity for decoding speed, or sacrifice
decoding speed for higher accuracy.

In this paper, we propose a novel Global (featuring)
Greedy (inference) parsing architecture\(^1\) that achieves fast
training, high decoding speed and good performance. With
our approach, we use the one-shot arc scoring scheme
as in the graph-based parser instead of the stepwise
local scoring in transition-based. This is essential for
achieving competitive performance, efficient training, and
fast decoding. Since, to preserve linear time decoding,
we chose a greedy algorithm, we introduce a parsing order
scoring scheme to retain the decoding order in inference
to achieve the highest accuracy possible. Just as with one-
shot scoring in graph-based parsers, our proposed parser will
perform arc-attachment scoring, parsing order scoring, and
decoding simultaneously in an incremental, deterministic
fashion just as transition-based parsers do.

We evaluated our models on the common benchmark
treebanks PTB and CTB, as well as on the multilingual
CoNLL and the Universal Dependency treebanks. From the
evaluation results on the benchmark treebanks, our proposed
model gives significant improvements when compared to the
baseline parser. In summary, our contributions are thus:

- We integrate the arc scoring mechanism of graph-based
  parsers and the linear time complexity inference approach
  of transition parsing models, which, by replacing stepwise
  local feature scoring, significantly alleviates the drawbacks
  of these models, improving their moderate performance
casted by error propagation and increasing their training
  speeds resulting from their lack of parallelism.
- Empirical evaluations on benchmark and multilingual
treebanks show that our method achieves state-of-the-art or
  comparable performance, indicating that our novel neural
  network architecture for dependency parsing is simple,
effective, and efficient.
- Our work shows that using neural networks’ excellent
  learning ability, we can simultaneously achieve both
  improved accuracy and speed.

3 Global Greedy Parsing Model
Our proposed global greedy model contains three compo-
nents: (1) an encoder that processes the input sentence and
maps it into hidden states that lie in a low dimensional vector
space \(h_i\), and feeds it into a specific representation layer
to strip away irrelevant information, (2) a modified scorer with
a parsing order objective, and (3) a greedy inference module
that generates the dependency tree.

3.1 Encoder
We employ a bi-directional LSTM-CNN architecture
(BiLSTM-CNN) to encode the context in which convolu-
tional neural networks (CNNs) learn character-level
information \(e_{\text{char}}\) to better handle out-of-vocabulary words.
We then combine these words’ character level embeddings
with their word embedding \(e_{\text{word}}\) and POS embedding
\(e_{\text{pos}}\) to create a context-independent representation, which
we then feed into the BiLSTM to create word-level
context-dependent representations. To further enhance
the word-level representation, we leverage an external fixed
representation \(e_{\text{lm}}\) from pre-trained ELMo (Peters et al.
2018) or BERT (Devlin et al. 2019) layer features.
Finally, the encoder outputs a sequence of contextualized
representations \(h_i\).

\[
h_i = \text{BiLSTM}M([e^i_{\text{word}}; e^i_{\text{pos}}; e^i_{\text{char}}; e^i_{\text{lm}}])
\]

Because the contextualized representations will be used
for several different purposes in the following scorers, it is
necessary to specify a representation for each purpose.
As shown in (Dozat and Manning 2017), applying a multi-
layer perceptron (MLP) to the recurrent output states before
the classifier strips away irrelevant information for the

\(^1\)Our code is available at https://github.com/bcmi220/ggdp.
current decision, reducing both the dimensionality and the risk of model overfitting. Therefore, in order to distinguish the biaffine scorer’s head and dependent representations and the parsing order scorer’s representations, we add a separate contextualized representation layer with ReLU as its activation function for each syntax head \( h_{i}^{\text{head}} \in H_{\text{head}} \) specific representations, dependent \( h_{i}^{\text{dep}} \in H_{\text{dep}} \) specific representations, and parsing order \( h_{i}^{\text{order}} \in H_{\text{order}} \):

\[
h_{i}^{m} = \text{ReLU}(\text{MLP}_{m}(h_{i})), \quad m \in \{\text{head}, \text{dep}, \text{order}\}
\]

### 3.2 Scorers

The traditional easy-first model relies on an incremental tree scoring process with stepwise loss backpropagation and sub-tree removal facilitated by local scoring, relying on the scorer and loss backpropagation to hopefully obtain the parsing order. Communicating the information from the scorer and the loss requires training a dynamic oracle, which exposes the model to the configurations resulting from erroneous decisions. This training process is done at the token level, not the sentence level, which unfortunately means incremental scoring prevents parallelized training and causes error propagation. We thus forego incremental local scoring, and, inspired by the design of graph-based parsing models, we instead choose to score all of the syntactic arc candidates in one-shot, which allows for global featuring at a sentence level; however, the introduction of one-shot scoring brings new problems. Since the graph-based method relies on a tree space search algorithm to find the tree with the highest score, the parsing order is not important at all. If we apply one-shot scoring to greedy parsing, we need a mechanism like a stack (as is used in transition-based parsing) to preserve the parsing order.

Both transition-based and easy-first parsers build parse trees in an incremental style, which forces tree formation to follow an order starting from either the root and working towards the leaf nodes or vice versa. When a parser builds an arc that skips any layer, certain errors will exist that it will be impossible for the parent node to find. We thus implement a parsing order prediction module to learn a parsing order objective that outputs a parsing order score addition to the arc score to ensure that each pending node is attached to its parent only after all (or at least as many as possible) of its children have been collected.

Our scorer consists of two parts: a biaffine scorer for one-shot scoring and a parsing order scorer for parsing order guiding. For the biaffine scorer, we adopt the biaffine attention mechanism (Dozat and Manning 2017) to score all possible head-dependent pairs:

\[
s_{\text{arc}} = H_{\text{head}}W_{\text{arc}}H_{\text{dep}} + U_{\text{arc}}^{T}H_{\text{head}} + V_{\text{arc}}^{T}H_{\text{dep}} + b_{\text{arc}}
\]

where \( W_{\text{arc}}, U_{\text{arc}}, V_{\text{arc}}, b_{\text{arc}} \) are the weight matrix of the bi-linear term, the two weight vectors of the linear terms, and the bias vector, respectively.

If we perform greedy inference only on the \( s_{\text{arc}} \) directly, as in Figure 3, at step \( i \), the decoder tests every pair in the pending list, and although the current score fits the correct tree structure for this example, because backtracking is not allowed in the deterministic greedy inference, according to the maximum score \( s_{\text{arc}} \), the edge selected in step \( i+1 \) is “root”→“come”. This prevents the child nodes (“today” and “.”) from finding the correct parent node in the subsequent step. Thus, the decoder is stuck with this error. This problem can be solved or mitigated by using a max spanning tree (MST) decoder or by adding beam search method to the inference, but neither guarantees maintaining linear time decoding. Therefore, we propose a new scorer for parsing order \( s_{\text{order}} \). In the scoring stage, the parsing order score is passed to the decoder to guide it and prevent (as much as possible) resorting to erroneous choices.

---

**Figure 2:** Overview for our global greedy parser.
3.3 Training Objectives

To parse the syntax tree $y$ for a sentence $x$ with length $l$, the easy-first model relies on an action-by-action process performed on $pending$. In the training stage, the loss is accumulated once per step (action), and the model is updated by gradient backpropagation according to a preset frequency. This prohibits parallelism during model training lack between and within sentences. Therefore, the traditional easy-first model was trained to maximize following probability:

$$P_{\theta}(y|x) = \prod_{i=1}^{l-1} P_{\theta}(y_i^{arc}|pending_i),$$

where $pending_i$ is the pending list state at step $i$.

While for our proposed model, it uses the training method similar to that of graph-based models, in which the arc scores are all obtained in one-shot. Consequently, it does not rely on the pending list in the training phase and only uses the pending list to promote the process of linear parsing in the inference stage. Our model is trained to optimize the probability of the dependency tree $y$ when given a sentence $x$: $P_{\theta}(y|x)$, which can be factorized as:

$$P_{\theta}(y|x) = \prod_{i=1}^{l} P_{\theta}(y_i^{arc}, y_i^{rel}, y_i^{order}|x_i)$$

where $\theta$ represents learnable parameters, $l$ denotes the length of the processing sentence, and $y_i^{arc}, y_i^{rel}$ denote the highest-scoring head and dependency relation for node $x_i$. Thus, our model factors the distribution according to a bottom-up tree structure.

Corresponding to multiple objectives, several parts compose the loss of our model. The overall training loss is the sum of three objectives:

$$\mathcal{L} = \mathcal{L}^{arc} + \mathcal{L}^{rel} + \mathcal{L}^{order},$$

where the loss for arc prediction $\mathcal{L}^{arc}$ is the negative log-likelihood loss of the golden structure $y^{arc}$:

$$\mathcal{L}^{arc}(x) = -\log P_{\theta}(y^{arc}|x),$$

the loss for relation prediction $\mathcal{L}^{rel}$ is implemented as the negative log-likelihood loss of the golden relation $y^{rel}$ with the golden structure $y^{arc}$:

$$\mathcal{L}^{rel}(x, y^{arc}) = -\log P_{\theta}(y^{rel}|x, y^{arc}),$$

and the loss for parsing order prediction $\mathcal{L}^{order}$:

$$\mathcal{L}^{order}(x) = -\log P_{\theta}(y^{order}|x),$$

Because the parsing order score of each layer in the tree increases by 1, we frame it as a classification problem and therefore add a multi-class classifier module as the order scorer.

3.4 Non-Projective Inference

For non-projective inference, we introduce two additional arc-building actions as follows.

---

*Footnote: A node is a well-formed tree via the greedy parsing algorithm.*
NP-ATTACHLEFT(i): attaches \( p_j \) to \( p_i \) where \( j > i \), which builds an arc \( (p_i, p_j) \) headed by \( p_i \), and removes \( p_j \) from \textit{pending}.

NP-ATTACHRIGHT(i): attaches \( p_j \) to \( p_i \) where \( j < i \) which builds an arc \( (p_i, p_j) \) headed by \( p_i \), and removes \( p_j \) from \textit{pending}.

If we use the two arc-building actions for non-projective dependency trees directly on \( s_{final} \), the time complexity will become \( O(n^3) \), so we need to modify this algorithm to accommodate the non-projective dependency trees. Specifically, we no longer use \( s_{final} \) directly for greedy search but instead divide each decision into two steps. The first step is to use the order score \( s_{order} \) to sort the pending list in descending order. Then, the second step is to find the edge with the largest arc score \( s_{arc} \) for this node in the first position of the pending list.

### 3.5 Time Complexity

The number of decoding steps to build a parse tree for a sentence is the same as its length, \( n \). Combining this with the searching in the \textit{pending} list (at each step, we need to find the highest-scoring pair in the pending list to attach. This has a runtime of \( O(n^2) \). The time complexity of a full decoding is \( O(n^3) \), which is equal to \( 1^{st} \)-order non-projective graph-based parsing but more efficient than \( 1^{st} \)-order projective parsing with \( O(n^2) \) and other higher order graph parsing models. Compared with the current state-of-the-art transition-based parser STACKPTR (Ma et al. 2018), with the same decoding time complexity as ours, since our number of decoding takes \( n \) steps while STACKPTR takes \( 2n - 1 \) steps for decoding and needs to compute the attention vector at each step, our model actually would be much faster than STACKPTR in decoding.

For the non-projective inference in our model, the complexity is still \( O(n^2) \). Since the order score and the arc score are two parts that do not affect each other, we can sort the order scores with time complexity of \( O(n \log n) \) and then iterate in this descending order. The iteration time complexity is \( O(n) \) and determining the arc is also \( O(n) \), so the overall time complexity is \( O(n \log n) + O(n^2) \), simplifying to \( O(n^2) \).

### 4 Experiments

We evaluate our parsing model on the English Penn Treebank (PTB), the Chinese Penn Treebank (CTB), treebanks from two CoNLL shared tasks and the Universal Dependency (UD) Treebanks, using unlabeled attachment scores (UAS) and labeled attachment scores (LAS) as the metrics. Punctuation is ignored as in previous work (Dozat and Manning 2017). For English and Chinese, we use the projective inference, while for other languages, we use the non-projective one.

#### 4.1 Treebanks

For English, we use the Stanford Dependency (SD 3.3.0) (De Marneffe and Manning 2008) conversion of the Penn
Treebanks (Marcus, Marcinkiewicz, and Santorini 1993), and follow the standard splitting convention for PTB, using sections 2-21 for training, section 22 as a development set and section 23 as a test set. We use the Stanford POS tagger (Toutanova et al. 2003) to generate predicted POS tags.

For Chinese, we adopt the splitting convention for CTB (Xue et al. 2005) described in (Dyer et al. 2015). The dependencies are converted with the Penn2Malt converter.

For the CoNLL Treebanks, we use the English treebank from the CoNLL-2008 shared task (Surdeanu et al. 2008) and all 13 treebanks from the CoNLL-X shared task (Buchholz and Marsi 2006). The experimental settings are the same as (Ma and Hovy 2015).

For UD Treebanks, following the selection of (Ma et al. 2018), we take 12 treebanks from UD version 2.1 (Nivre et al. 2017): Bulgarian (bg), Catalan (ca), Czech (cs), Dutch (nl), English (en), French (fr), German (de), Italian (it), Norwegian (no), Romanian (ro), Russian (ru) and Spanish (es). We adopt the standard training/dev/test splits and use the universal POS tags provided in each treebank for all the languages.

### 4.2 Implementation Details

**Pre-trained Embeddings** We use the GloVe (Pennington, Socher, and Manning 2014) trained on Wikipedia and Gigaword as external embeddings for English parsing. For other languages, we use the word vectors from 157 languages trained on Wikipedia and Crawl using fastText (Grave et al. 2018). We use the extracted BERT layer features to enhance the performance on CoNLL-X and UD treebanks.

**Hyperparameters** The character embeddings are 8-dimensional and randomly initialized. In the character CNN, the convolutions have a window size of 3 and consist of 50 filters. We use 3 stacked bidirectional LSTMs with 512-dimensional hidden states each. The outputs of the BiLSTM employ a 512-dimensional MLP layer for the arc scorer, a 128-dimensional MLP layer for the relation scorer, and a 128-dimensional MLP layer for the parsing order scorer, with all using ReLU as the activation function. Additionally, for parsing the order score, since considering it a classification problem over parse tree layers, we set its range to $[0, 1, ..., 32]$.

**Training** Parameter optimization is performed with the Adam optimizer with $\beta_1 = \beta_2 = 0.9$. We choose an initial learning rate of $\eta_0 = 0.001$. The learning rate $\eta$ is annealed by multiplying a fixed decay rate $\rho = 0.75$ when parsing performance stops increasing on validation sets. To reduce the effects of an exploding gradient, we use a gradient clipping of 5.0. For the BiLSTM, we use recurrent dropout with a drop rate of 0.33 between hidden states and 0.33 between layers. Following (Dozat and Manning 2017), we also use embedding dropout with a rate of 0.33 on all word, character, and POS tag embeddings.

### 4.3 Main Results

We now compare our model with several other recently proposed parsers as shown in Table 1. Our global greedy parser significantly outperforms the easy-first parser in (Kiperwasser and Goldberg 2016a) (HT-LSTM) on both PTB and CTB. Compared with other graph- and transition-based parsers, our model is also competitive with the best results in recent years among (Koo et al. 2010), (Martins et al. 2011), (Martins, Almeida, and Smith 2013), (Lei et al. 2014), (Zhang et al. 2014), (Zhang and McDonald 2014), (Pitler and McDonald 2015), and (Cheng et al. 2016).

![Table 2: UAS and LAS on 14 treebanks from CoNLL shared tasks, together with several state-of-the-art parsers. Bi-Att is the bi-directional attention based parser (Cheng et al. 2016), and NeuroMST is the neural MST parser (Ma and Hovy 2017). “Best Published” includes the best results in recent years among (Koo et al. 2010), (Martins et al. 2011), (Martins, Almeida, and Smith 2013), (Lei et al. 2014), (Zhang et al. 2014), (Zhang and McDonald 2014), (Pitler and McDonald 2015), and (Cheng et al. 2016) in addition to the ones we listed above.](image-url)

**Table 2**

<table>
<thead>
<tr>
<th></th>
<th>Bi-Att</th>
<th>NeuroMST</th>
<th>BIAF</th>
<th>STACKPTR</th>
<th>Ours</th>
<th>Best Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>80.34 [68.58]</td>
<td>80.80 [69.40]</td>
<td>82.15 [71.32]</td>
<td>83.04 [72.94]</td>
<td>84.47 [74.12]</td>
<td>81.12 [-]</td>
</tr>
<tr>
<td>bg</td>
<td>93.96 [89.55]</td>
<td>94.28 [90.60]</td>
<td>94.62 [91.56]</td>
<td>94.66 [91.40]</td>
<td>95.63 [92.47]</td>
<td>94.02 [-]</td>
</tr>
<tr>
<td>cs</td>
<td>91.16 [85.14]</td>
<td>91.18 [85.92]</td>
<td>92.24 [87.85]</td>
<td>92.83 [88.75]</td>
<td>93.71 [89.90]</td>
<td>91.16 [85.14]</td>
</tr>
<tr>
<td>da</td>
<td>91.56 [85.53]</td>
<td>91.86 [87.07]</td>
<td>92.80 [88.36]</td>
<td>92.08 [87.29]</td>
<td>93.06 [88.45]</td>
<td>92.00 [-]</td>
</tr>
<tr>
<td>de</td>
<td>92.71 [89.80]</td>
<td>93.62 [91.90]</td>
<td>94.52 [93.06]</td>
<td>94.77 [93.21]</td>
<td>95.34 [94.17]</td>
<td>92.71 [89.80]</td>
</tr>
<tr>
<td>es</td>
<td>88.74 [84.03]</td>
<td>89.20 [85.77]</td>
<td>90.43 [87.08]</td>
<td>90.87 [87.80]</td>
<td>91.92 [88.74]</td>
<td>88.75 [84.03]</td>
</tr>
<tr>
<td>ja</td>
<td>93.44 [90.67]</td>
<td>94.02 [92.60]</td>
<td>93.95 [92.46]</td>
<td>93.38 [91.92]</td>
<td>93.70 [92.05]</td>
<td>93.80 [-]</td>
</tr>
<tr>
<td>nl</td>
<td>87.15 [82.41]</td>
<td>87.85 [84.82]</td>
<td>90.07 [87.24]</td>
<td>90.10 [87.05]</td>
<td>91.84 [89.30]</td>
<td>87.39 [-]</td>
</tr>
<tr>
<td>pt</td>
<td>92.77 [88.44]</td>
<td>92.71 [88.92]</td>
<td>93.41 [89.96]</td>
<td>93.57 [90.07]</td>
<td>93.85 [90.56]</td>
<td>93.03 [-]</td>
</tr>
<tr>
<td>sl</td>
<td>86.01 [75.90]</td>
<td>86.73 [77.56]</td>
<td>87.55 [78.52]</td>
<td>87.59 [78.85]</td>
<td>88.91 [79.82]</td>
<td>87.06 [-]</td>
</tr>
<tr>
<td>sv</td>
<td>90.50 [84.05]</td>
<td>91.22 [86.92]</td>
<td>92.22 [88.44]</td>
<td>92.49 [89.01]</td>
<td>93.90 [89.91]</td>
<td>91.85 [85.26]</td>
</tr>
<tr>
<td>tr</td>
<td>78.43 [66.16]</td>
<td>77.71 [65.81]</td>
<td>79.84 [68.63]</td>
<td>79.56 [68.03]</td>
<td>82.06 [71.50]</td>
<td>78.43 [66.16]</td>
</tr>
<tr>
<td>zh</td>
<td>-</td>
<td>93.40 [90.10]</td>
<td>94.05 [90.89]</td>
<td>93.88 [90.81]</td>
<td>94.87 [91.52]</td>
<td>93.04 [-]</td>
</tr>
<tr>
<td>avg</td>
<td>88.89 [82.52]</td>
<td>89.94 [84.99]</td>
<td>90.93 [86.32]</td>
<td>90.86 [86.45]</td>
<td>92.05 [87.58]</td>
<td>-</td>
</tr>
</tbody>
</table>

Scores above 32 will be truncated to 32.
Table 3: UAS and LAS on test datasets of 12 treebanks from UD Treebanks, together with BIAF and STACKPTR for comparison.

<table>
<thead>
<tr>
<th></th>
<th>BIAF</th>
<th>STACKPTR</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
<td>UAS</td>
</tr>
<tr>
<td>bg</td>
<td>94.30</td>
<td>[90.04]</td>
<td>94.31</td>
</tr>
<tr>
<td>ca</td>
<td>94.36</td>
<td>[90.25]</td>
<td>94.47</td>
</tr>
<tr>
<td>cs</td>
<td>94.06</td>
<td>[90.60]</td>
<td>94.21</td>
</tr>
<tr>
<td>de</td>
<td>90.26</td>
<td>[86.11]</td>
<td>90.26</td>
</tr>
<tr>
<td>en</td>
<td>91.91</td>
<td>[89.82]</td>
<td>91.93</td>
</tr>
<tr>
<td>es</td>
<td>93.72</td>
<td>[91.33]</td>
<td>93.77</td>
</tr>
<tr>
<td>fr</td>
<td>92.62</td>
<td>[89.51]</td>
<td>92.90</td>
</tr>
<tr>
<td>it</td>
<td>94.75</td>
<td>[92.72]</td>
<td>94.70</td>
</tr>
<tr>
<td>nl</td>
<td>93.44</td>
<td>[91.04]</td>
<td>93.98</td>
</tr>
<tr>
<td>no</td>
<td>95.28</td>
<td>[93.58]</td>
<td>95.33</td>
</tr>
<tr>
<td>ru</td>
<td>91.94</td>
<td>[85.61]</td>
<td>91.80</td>
</tr>
<tr>
<td>ru</td>
<td>94.40</td>
<td>[92.68]</td>
<td>94.69</td>
</tr>
<tr>
<td>avg</td>
<td>93.42</td>
<td>[90.42]</td>
<td>93.52</td>
</tr>
</tbody>
</table>

Table 4: Training time and decoding speed. The experimental environment is on the same machine with Intel i9 9900k CPU and NVIDIA 1080Ti GPU.

<table>
<thead>
<tr>
<th>Systems</th>
<th>Training T (Hours)</th>
<th>Decoding S (tokens/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin Easy-First</td>
<td>≈16</td>
<td>8532</td>
</tr>
<tr>
<td>BIAF</td>
<td>≈8</td>
<td>2568</td>
</tr>
<tr>
<td>STACKPTR</td>
<td>≈12</td>
<td>1508</td>
</tr>
<tr>
<td>Ours</td>
<td>≈8</td>
<td>8541</td>
</tr>
</tbody>
</table>

Overall, our model reports more than 1.0% higher average UAS than STACKPTR and 0.3% higher than BIAF.

4.6 Runtime Analysis

In order to verify the time complexity analysis of our model, we measured the running time and speed of BIAF, STACKPTR and our model on PTB training and development set using the projective algorithm. The comparison in Table 4 shows that in terms of convergence time, our model is basically the same speed as BIAF, while STACKPTR is much slower. For decoding, our model is the fastest, followed by BIAF. STACKPTR is unexpectedly the slowest. This is because the time cost of attention scoring in decoding is not negligible when compared with the processing speed and actually even accounts for a significant portion of the runtime.

5 Conclusion

This paper presents a new global greedy parser in which we enable greedy parsing inference compatible with the global arc scoring of graph-based parsing models instead of the local feature scoring of transitional parsing models. The proposed parser can perform projective parsing when only using two arc-building actions, and it also supports non-projective parsing when introducing two extra non-projective arc-building actions. Compared to graph-based and transition-based parsers, our parser achieves a better tradeoff between parsing accuracy and efficiency by taking advantages of both graph-based models’ training methods and transition-based models’ linear time decoding strategies. Experimental results on 28 treebanks show the effectiveness of our parser by achieving good performance on 27 treebanks, including the PTB and CTB benchmarks.

References


Cheng, H.; Fang, H.; He, X.; Gao, J.; and Deng, L. 2016. Bi-directional attention with agreement for dependency parsing. In EMNLP.


Grave, E.; Bojanowski, P.; Gupta, P.; Joulin, A.; and Mikolov, T. 2018. Learning word vectors for 157 languages. In LREC.

Kiperwasser, E., and Goldberg, Y. 2016a. Easy-first dependency parsing with hierarchical tree LSTMs. TACL.

Kiperwasser, E., and Goldberg, Y. 2016b. Simple and accurate dependency parsing using bidirectional LSTM feature representations. TACL.


Lei, T.; Xin, Y.; Zhang, Y.; Burzilay, R.; and Jaakkola, T. 2014. Low-rank tensors for scoring dependency structures. In ACL.

Li, Z.; Cai, J.; He, S.; and Zhao, H. 2018a. Seq2seq dependency parsing. In COLING.

Li, Z.; He, S.; Zhang, Z.; and Zhao, H. 2018b. Joint learning of POS and dependencies for multilingual universal dependency parsing. In CoNLL.


Ma, X., and Zhao, H. 2012. Fourth-order dependency parsing. In COLING.


Marcus, M. P.; Marcinkiewicz, M. A.; and Santorini, B. 1993. Building a large annotated corpus of English: The penn treebank. CL.


Martins, A. F.; Smith, N. A.; Aguilar, P. M.; and Figueiredo, M. A. 2011. Dual decomposition with many overlapping components. In EMNLP.

McDonald, R., and Nivre, J. 2011. Analyzing and integrating dependency parsers. CL.


McDonald, R.; Crammer, K.; and Pereira, F. 2005. Online large-margin training of dependency parsers. In ACL.


Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In EMNLP.


Wang, W., and Chang, B. 2016. Graph-based dependency parsing with bidirectional LSTM. In ACL.

Wang, W.; Chang, B.; and Mansur, M. 2018. Improved dependency parsing using implicit word connections learned from unlabeled data. In EMNLP.


Zhang, Y., and Nivre, J. 2011. Transition-based dependency parsing with rich non-local features. In ACL-HLT.

Zhang, Y.; Lei, T.; Burzilay, R.; and Jaakkola, T. 2014. Greed is good if randomized: New inference for dependency parsing. In EMNLP.

Zhang, Z.; Zhao, H.; and Qin, L. 2016. Probabilistic graph-based dependency parsing with convolutional neural network. In ACL.

Zhao, H.; Song, Y.; Kit, C.; and Zhou, G. 2009. Cross language dependency parsing using a bilingual lexicon. In ACL.
