The (Non-)Utility of Structural Features in BiLSTM-based Dependency Parsers

Agnieszka Falenska and **Jonas Kuhn**
Institut für Maschinelle Sprachverarbeitung
University of Stuttgart
name.surname@ims.uni-stuttgart.de

Abstract

Classical non-neural dependency parsers put considerable effort on the design of feature functions. Especially, they benefit from information coming from structural features, such as features drawn from neighboring tokens in the dependency tree. In contrast, their BiLSTM-based successors achieve state-of-the-art performance without explicit information about the structural context. In this paper we aim to answer the question: How much structural context are the BiLSTM representations able to capture implicitly? We show that features drawn from partial subtrees become redundant when the BiLSTMs are used. We provide a deep insight into information flow in transition- and graph-based neural architectures to demonstrate where the implicit information comes from when the parsers make their decisions. Finally, with model ablations we demonstrate that the structural context is not only present in the models, but it significantly influences their performance.

1 Introduction

When designing a conventional non-neural parser substantial effort is required to design a powerful feature extraction function. Such a function (McDonald et al., 2005; Zhang and Nivre, 2011, among others) is constructed so that it captures as much *structural context* as possible. The context allows the parser to make well-informed decisions.[1] It is encoded in features built from partial subtrees and *explicitly* used by the models.

Recently, Kiperwasser and Goldberg (2016, K&G) showed that the conventional feature extraction functions can be replaced by modeling the left- and right-context of each word with BiLSTMs (Hochreiter and Schmidhuber, 1997; Graves and Schmidhuber, 2005). Although the proposed models *do not use any conventional structural features* they achieve state-of-the-art performance. The authors suggested that it is because the BiLSTM encoding is able to estimate the missing information from the given features and did not explore this issue further.

Since the introduction of the K&G architecture BiLSTM-based parsers have become standard in the field.[2] Yet, it is an open question how much conventional structural context the BiLSTMs representations actually are able to capture *implicitly*. Small architectures that ignore the structural context are attractive since they come with lower time complexity. But to build such architectures it is important to investigate to what extent the explicit structural information is redundant. For example, K&G also proposed an extended feature set derived from structural context, which has subsequently been re-implemented and used by others without questioning its utility.

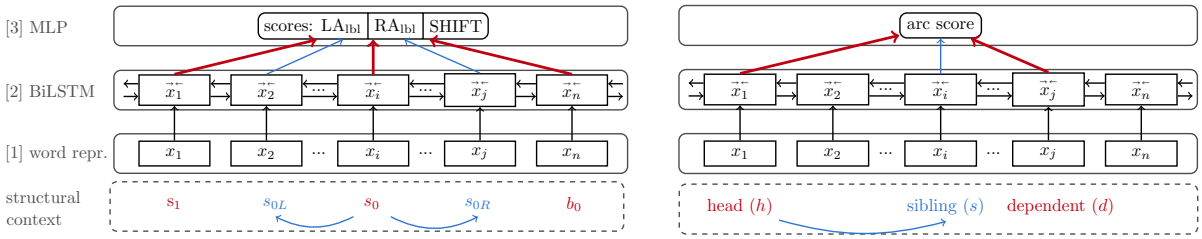
Inspired by recent work (Gaddy et al., 2018) on constituency parsing we aim at understanding what type of information is captured by the internal representations of BiLSTM-based dependency parsers and how it translates into their impressive accuracy. As our starting point we take the K&G architecture and extend it with a second-order decoder.[3] We perform systematic analyses on nine languages using two different architectures (transition-based and graph-based) across two dimensions: with and without BiLSTM representations, and with and without features drawn from structural context.

[1] See Figure 1 for the concept of structural context, details of the architectures will be described in Section 2.

[2] See results from the recent CoNLL 2018 shared task on dependency parsing (Zeman et al., 2018) for a comparison of various high-performing dependency parsers.

[3] To the best of our knowledge, this is the first BiLSTM-based second-order dependency parser. Gómez-Rodríguez et al. (2018) incorporate BiLSTM-based representations into the third-order 1-Endpoint-Crossing parser of Pitler (2014).

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 117–128
Florence, Italy, July 28 - August 2, 2019.



(a) Transition-based parser; scoring transitions for the configuration $\langle \Sigma, B, A \rangle = \langle x_1 \dots x_i, x_n, \{x_i \rightarrow x_2, x_i \rightarrow x_j, \dots\} \rangle$

(b) Graph-based parser; scoring an arc $x_1 \rightarrow x_j$

Figure 1: Schematic illustration of the K&G architecture of BiLSTM-based neural dependency parsers. Red arrows mark the basic feature sets and blue show how to extend them with features drawn from structural context.

We demonstrate that structural features are useful for neural dependency parsers but they become redundant when BiLSTMs are used (Section 4). It is because the BiLSTM representations trained together with dependency parsers capture a significant amount of complex syntactic relations (Section 5.1). We then carry out an extensive investigation of information flow in the parsing architectures and find that the implicit structural context is not only present in the BiLSTM-based parsing models, but also more diverse than when encoded in explicit structural features (Section 5.2). Finally, we present results on ablated models to demonstrate the influence of structural information implicitly encoded in BiLSTM representations on the final parsing accuracy (Section 5.3).

2 Parsing Model Architecture

Our graph- and transition-based parsers are based on the K&G architecture (see Figure 1). The architecture has subsequently been extended by, e.g., character-based embeddings (de Lhoneux et al., 2017) or attention (Dozat and Manning, 2016). To keep the experimental setup clean and simple while focusing on the information flow in the architecture, we abstain from these extensions. We use the basic K&G architecture as our starting point with a few minor changes outlined below. For further details we refer the reader to Kiperwasser and Goldberg (2016).

2.1 Word Representations

In both transition- and graph-based architectures input tokens are represented in the same way (see level [1] in Figure 1). For a given sentence with words $[w_1, \dots w_n]$ and part-of-speech (POS) tags $[t_1, \dots, t_n]$ each word representation x_i is built from concatenating the embeddings of the word and its POS tag:

$$x_i = e(w_i) \circ e(t_i)$$

The embeddings are initialized randomly at training time and trained together with the model.

The representations x_i encode words in isolation and do not contain information about their context. For that reason they are passed to the BiLSTM feature extractors (level [2] in Figure 1) and represented by a BiLSTM representation $\vec{x_i}$:

$$\vec{x_i} = \text{BiLSTM}(x_{1:n}, i)$$

2.2 Transition-Based Parser

Transition-based parsers gradually build a tree by applying a sequence of transitions. During training they learn a scoring function for transitions. While decoding they search for the best action given the current state and the parsing history.

Figure 1a illustrates the architecture of the transition-based K&G parser. For every configuration c consisting of a stack, buffer, and a set of arcs introduced so far, the parser selects a few core items from the stack and buffer (red arrows in the figure) as features. Next, it concatenates their BiLSTM vectors and passes them to a multi-layer perceptron (MLP) which assigns scores to all possible transitions. The highest scoring transition is used to proceed to the next configuration.

Our implementation (denoted **TbPars**) uses the arc-standard decoding algorithm (Nivre, 2004) extended with a SWAP transition (ASWAP, Nivre (2009)) to handle non-projective trees. The system applies arc transitions between the two topmost items of the stack (denoted s_0 and s_1). We use the lazy SWAP oracle by Nivre et al. (2009) for training. Labels are predicted together with the transitions. We experiment with two models with different feature sets:

TBMIN: is the simple architecture which does not use structural features. Since Shi et al. (2017) showed that the feature set $\{\vec{s}_0, \vec{s}_1, \vec{b}_0\}$ is minimal for the arc-standard system (i.e., it suffers almost no loss in performance in comparison to larger feature sets but significantly out-performs a feature set built from only two vectors) we apply the same feature set to ASWAP. Later we analyze if the set could be further reduced.

TBEXT: is the extended architecture. We use the original extended feature set from K&G: $\{\vec{s}_0, \vec{s}_1, \vec{s}_2, \vec{b}_0, \vec{s}_{0L}, \vec{s}_{0R}, \vec{s}_{1L}, \vec{s}_{1R}, \vec{s}_{2L}, \vec{s}_{2R}, \vec{b}_{0L}\}$, where $._L$ and $._R$ denote left- and right-most child.

2.3 Graph-Based Parser

The K&G graph-based parser follows the structured prediction paradigm: while training it learns a scoring function which scores the correct tree higher than all the other possible ones. While decoding it searches for the highest scoring tree for a given sentence. The parser employs an arc-factored approach (McDonald et al., 2005), i.e., it decomposes the score of a tree to the sum of the scores of its arcs.

Figure 1b shows the K&G graph-based architecture. At parsing time, every pair of words $\langle x_i, x_j \rangle$ yields a BiLSTM representation $\{\vec{x}_i, \vec{x}_j\}$ (red arrows in the figure) which is passed to MLP to compute the score for an arc $x_i \rightarrow x_j$. To find the highest scoring tree we apply Eisner (1996)'s algorithm. We denote this architecture **GBMIN**. We note in passing that, although this decoding algorithm is restricted to projective trees, it has the advantage that it can be extended to incorporate non-local features while still maintaining exact search in polynomial time.[4]

The above-mentioned simple architecture uses a feature set of two vectors $\{\vec{h}, \vec{d}\}$. We extend it and add information about structural context. Specifically, we incorporate information about siblings $\vec{s}$ (blue arrows in the figure). The model follows the second-order model from McDonald and Pereira (2006) and decomposes the score of the tree into the sum of adjacent edge pair scores. We use the implementation of the second-order decoder from Zhang and Zhao (2015). We denote this architecture **GBSIBL**.

[4]Replacing Eisner (1996)'s algorithm with the Chu-Liu-Edmonds's decoder (Chu and Liu, 1965; Edmonds, 1967) which can predict non-projective arcs causes significant improvements only for the Ancient Greek treebank (1.02 LAS on test set).

3 Experimental Setup

Data sets and preprocessing. We perform experiments on a selection of nine treebanks from Universal Dependencies (Nivre et al., 2016) (v2.0): Ancient Greek PROIEL (grc), Arabic (ar), Chinese (zh), English (en), Finnish (fi), Hebrew (he), Korean (ko), Russian (ru) and Swedish (sv). This selection was proposed by Smith et al. (2018) as a sample of languages varying in language family, morphological complexity, and frequencies of non-projectivity (we refer to Smith et al. (2018) for treebank statistics). To these 9, we add the English Penn Treebank (en-ptb) converted to Stanford Dependencies.[5] We use sections 2-21 for training, 24 as development set and 23 as test set.

We use automatically predicted universal POS tags in all the experiments. The tags are assigned using a CRF tagger (Mueller et al., 2013). We annotate the training sets via 5-fold jackknifing.

Evaluation. We evaluate the experiments using Labeled Attachment Score (LAS).[6] We train models for 30 epochs and select the best model based on development LAS. We follow recommendations from Reimers and Gurevych (2018) and report averages and standard deviations from six models trained with different random seeds. We test for significance using the Wilcoxon rank-sum test with p-value < 0.05.

Analysis is carried out on the development sets in order not to compromise the test sets. We present the results on the concatenation of all the development sets (one model per language). While the absolute numbers vary across languages, the general trends are consistent with the concatenation.

Implementation details. All the described parsers were implemented with the DyNet library (Neubig et al., 2017).[7] We use the same hyperparameters as Kiperwasser and Goldberg (2016) and summarize them in Table 2 in Appendix A.

[5]We use version 3.4.1 of the Stanford Parser from http://nlp.stanford.edu/software/lex-parser.shtml

[6]The ratio of tokens with a correct head and label to the total number of tokens in the test data.

[7]The code can be found on the first author's website.

	avg.	en-ptb	ar	en	fi	grc	he	ko	ru	sv	zh
TbMin	76.43	90.25	76.22	81.85†	72.51†	71.92†	79.41†	64.39	74.35†	80.11†	73.28†
TbExt	75.56	90.25	75.77	80.50	71.47	70.32	78.62	63.88	73.82	78.80	72.17
GbMin	77.74	91.40	77.25	82.53	74.37	73.48	80.83	65.47	76.43	81.22	74.47
GbSibl	77.89	91.59	77.21	82.65	74.44	73.20	81.03	65.61	76.79†	81.42	74.95†

Table 1: Average (from six runs) parsing results (LAS) on test sets. † marks statistical significance (p-value < 0.05). Corresponding standard deviations are provided in Table 3 in Appendix A.

4 Structural Features and BiLSTMs

4.1 Simple vs. Extended Architectures

We start by evaluating the performance of our four models. The purpose is to verify that the simple architectures will compensate for the lack of additional structural features and achieve comparable accuracy to the extended ones.

Table 1 shows the accuracy of all the parsers. Comparing the simple and extended architectures we see that dropping the structural features does not hurt the performance, neither for transition-based nor graph-based parsers. Figure 2 displays the accuracy relative to dependency length in terms of recall.[8] It shows that the differences between models are not restricted to arcs of particular lengths.

In the case of graph-based models (GbMin vs. GbSibl) adding the second-order features to a BiLSTM-based parser improves the average performance slightly. However, the difference between those two models is significant only for two out of ten treebanks. For the transition-based parser (TbMin vs. TbExt) a different effect can be noticed – additional features cause a significant loss in accuracy for seven out of ten treebanks. One possible explanation might be that TbExt suffers more from error propagation than TbMin. The parser is greedy and after making the first mistake it starts drawing features from configurations which were not observed during training. Since the extended architecture uses more features than the simple one the impact of the error propagation might be stronger. This effect can be noticed in Figure 2. The curves for TbMin and TbExt are almost parallel for the short arcs but the performance of TbExt deteriorates for the longer ones, which are more prone to error propagation (McDonald and Nivre, 2007).

[8] Dependency recall is defined as the percentage of correct predictions among gold standard arcs of length l (McDonald and Nivre, 2007).

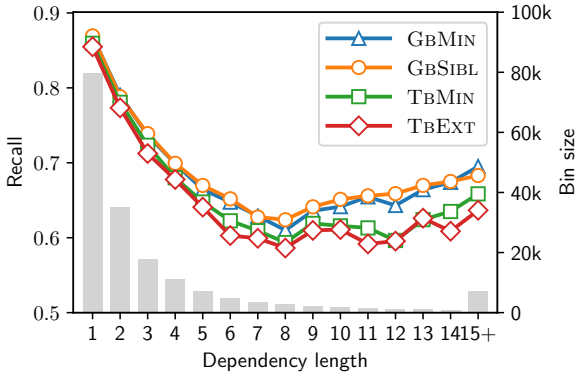


Figure 2: Dependency recall relative to arc length on development sets. The corresponding plot for precision shows similar trends (see Figure 7 in Appendix A).

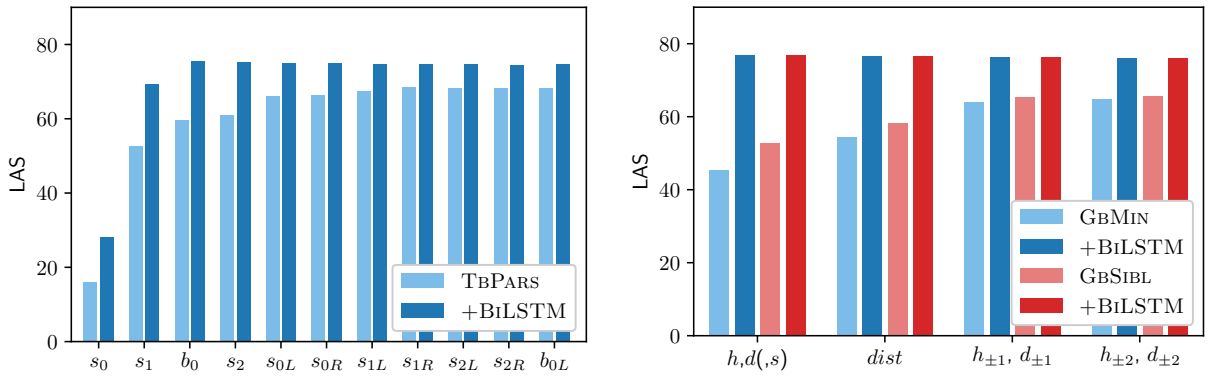
4.2 Influence of BiLSTMs

We now investigate whether BiLSTMs are the reason for models being able to compensate for lack of features drawn from partial subtrees.

Transition-based parser. We train TbPars in two settings: with and without BiLSTMs (when no BiLSTMs are used we pass vectors x_i directly to the MLP layer following Chen and Manning (2014)) and with different feature sets. We start with a feature set $\{s_0\}$ and consecutively add more until we reach the full feature model of TbExt.

Figure 3a displays the accuracy of all the trained models. First of all, we notice that the models without BiLSTMs (light bars) benefit from structural features. The biggest gains in the average performance are visible after adding vectors s_{0L} (5.15 LAS) and s_{1R} (1.12 LAS) . After adding s_{1R} the average improvements become modest.

Adding the BiLSTM representations changes the picture (dark bars). First of all, as in the case of arc-standard system (Shi et al., 2017), the feature set $\{\vec{s_0}, \vec{s_1}, \vec{b_0}\}$ is minimal for ASwap: none of the other structural features are able to improve the performance of the parser but dropping b_0 causes a big drop of almost 6 LAS on average. Secondly, the parsers which use BiLSTMs always have a big



(a) Transition-based parser

(b) Graph-based parser

Figure 3: Parsing accuracy (average LAS over ten treebanks) with incremental extensions to the feature set.

advantage over the parsers which do not, regardless of the feature model used.

Graph-based parser. We train two models: GbMin and GbSibl with and without BiLSTMs. To ensure a fairer comparison with the models without BiLSTMs we expand the basic feature sets ($\{\overrightarrow{h}, \overrightarrow{d}\}$ and $\{\overrightarrow{h}, \overrightarrow{d}, \overrightarrow{s}\}$) with additional surface features known from classic graph-based parsers, such as distance between head and dependent ($dist$), words at distance of 1 from heads and dependents ($h_{\pm1}$, $d_{\pm1}$) and at distance ±2. We follow Wang and Chang (2016) and encode distance as randomly initialized embeddings.

Figure 3b displays the accuracy of all the trained models with incremental extensions to their feature sets. First of all, we see that surface features ($dist$, $h_{\pm1}$, $d_{\pm1}$, $h_{\pm2}$, $d_{\pm2}$) are beneficial for the models without BiLSTM representations (light bars). The improvements are visible for both parsers, with the smallest gains after adding $h_{\pm2}$, $d_{\pm2}$ vectors: on average 0.35 LAS for GbSibl and 0.83 LAS for GbMin.

As expected, adding BiLSTMs changes the picture. Since the representations capture surface context, they already contain a lot of information about words around heads and dependents and adding features $h_{\pm1}$, $d_{\pm1}$ and $h_{\pm2}$, $d_{\pm2}$ does not influence the performance. Interestingly, introducing $dist$ is also redundant which suggests that either BiLSTMs are aware of the distance between tokens or they are not able to use this information in a meaningful way. Finally, even after adding all the surface features the models which do not employ BiLSTMs are considerably behind the ones which do.

Comparing GbMin (blue) with GbSibl (red) we see that adding information about structural context through second-order features is beneficial when the BiLSTM are not used (light bars): the second-order GbSibl has an advantage over GbMin of 0.81 LAS even when both of the models use all the additional surface information (last group of bars on the plot). But this advantage drops down to insignificant 0.07 LAS when the BiLSTMs are incorporated.

We conclude that, for both transition- and graph-based parsers, BiLSTMs not only compensate for absence of structural features but they also encode more information than provided by the manually designed feature sets.

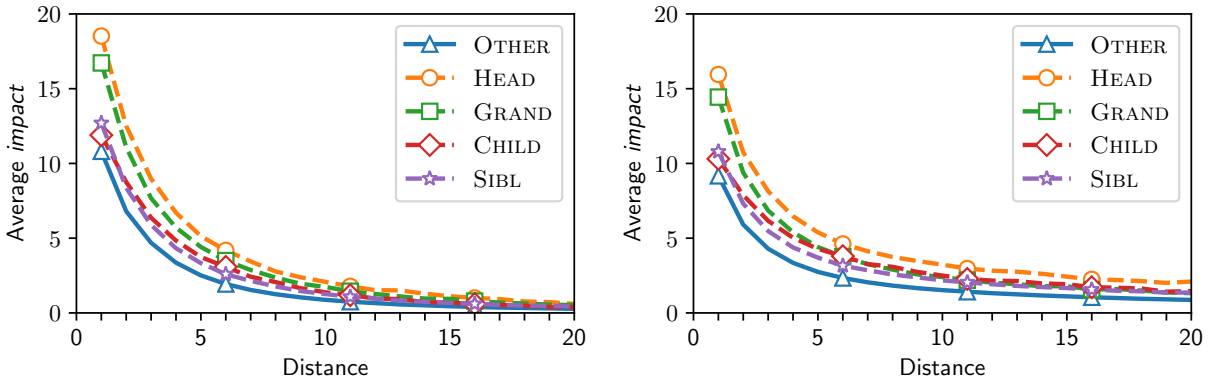
5 Implicit Structural Context

Now that we have established that structural features are indeed redundant for models which employ BiLSTMs we examine the ways in which the simple parsing models (TbMin and GbMin) implicitly encode information about partial subtrees.

5.1 Structure and BiLSTM Representations

We start by looking at the BiLSTM representations. We know that the representations are capable of capturing syntactic relations when they are trained on a syntactically related task, e.g, number prediction task (Linzen et al., 2016). We evaluate how complicated those relations can be when the representations are trained together with a dependency parser.

To do so, we follow Gaddy et al. (2018) and use derivatives to estimate how sensitive a particular part of the architecture is with respect to changes in input. Specifically, for every vector $\overrightarrow{x}$ we measure how it is influenced by every word represen-



(a) Transition-based parser (TBMIN) (b) Graph-based parser (GBMIN)

Figure 4: The average impact of tokens on BiLSTM vectors trained with dependency parser with respect to the surface distance and the structural (gold-standard) relation between them.

tation x_i from the sentence. If the derivative of $\overrightarrow{\overleftarrow{x}}$ with respect to x_i is high then the word x_i has a high influence on the vector. We compute the l_2-norm of the gradient of $\overrightarrow{\overleftarrow{x}}$ with respect to x_i and normalize it by the sum of norms of all the words from the sentence calling this measure ***impact***:

$$impact(\overrightarrow{\overleftarrow{x}}, i) = 100 \times \frac{||\frac{\partial \overrightarrow{\overleftarrow{x}}}{\partial x_i}||}{\sum_j ||\frac{\partial \overrightarrow{\overleftarrow{x}}}{\partial x_j}||}$$

For every sentence from the development set and every vector $\overrightarrow{\overleftarrow{x_i}}$ we calculate the impact of every representation x_j from the sentence on the vector $\overrightarrow{\overleftarrow{x_i}}$. We bucket those impact values according to the distance between the representation and the word. We then use the gold-standard trees to divide every bucket into five groups: correct heads of x_i, children (i.e., dependents) of x_i, grandparents (i.e., heads of heads), siblings, and other.

Figure 4 shows the average impact of tokens at particular positions. Similarly as shown by Gaddy et al. (2018) even words 15 and more positions away have a non-zero effect on the BiLSTM vector. Interestingly, the impact of words which we know to be structurally close to x_i is higher. For example, for the transition-based parser (Figure 4a) at positions ± 5 an average impact is lower than 2.5%, children and siblings of x_i have a slightly higher impact, and the heads and grandparents around 5%. For the graph-based parser (Figure 4b) the picture is similar with two noticeable differences. The impact of heads is much stronger for words 10 and more positions apart. But it is smaller than in the case of transition-based parser when the heads are next to x_i.

We conclude that the BiLSTMs are indeed influenced by the distance, but when trained with a dependency parser they also capture a significant amount of non-trivial syntactic relations.

5.2 Structure and Information Flow

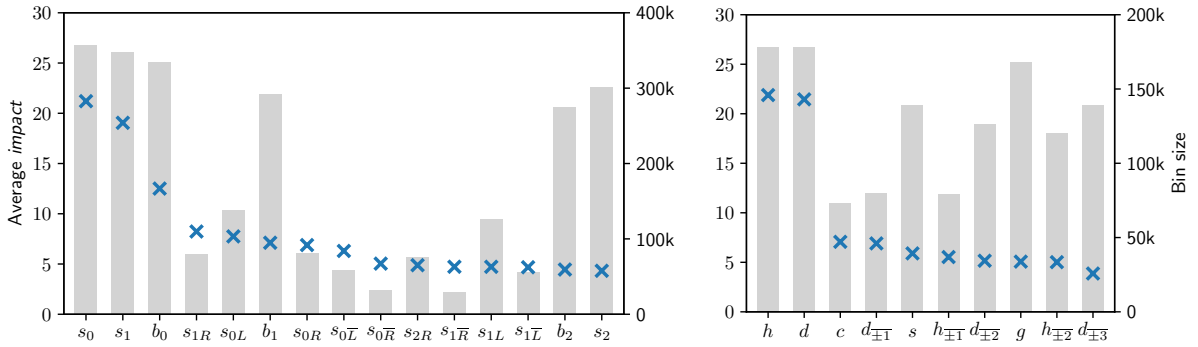
Now that we know that the representations encode structural information we ask how this information influences the decisions of the parser.

First, we investigate how much structural information flows into the final layer of the network. When we look back at the architecture in Figure 1 we see that when the final MLP scores possible transitions or arcs it uses only feature vectors $\{\overrightarrow{\overleftarrow{s_0}}, \overrightarrow{\overleftarrow{s_1}}, \overrightarrow{\overleftarrow{b_0}}\}$ or $\{\overrightarrow{\overleftarrow{h}}, \overrightarrow{\overleftarrow{d}}\}$. But thanks to the BiLSTMs the vectors encode information about other words from the sentence. We examine from which words the signal is the strongest when the parser makes the final decision.

We extend the definition of *impact* to capture how a specific word representation x_i influences the final MLP score sc (we calculate the derivative of sc with respect to x_i). We parse every development sentence. For every predicted transition/arc we calculate how much its score sc was affected by every word from the sentence. We group impacts of words depending on their positions.

Transition-based parser. For the transition-based parser we group tokens according to their positions in the configuration. For example, for the decision in Figure 1a $impact(sc, 1)$ would be grouped as s_1 and $impact(sc, j)$ as s_{0R}.

In Figure 5a we plot the 15 positions with the highest impact and the number of configurations they appear in (gray bars). As expected, s_0, s_1, and



(a) Transition-based parser (TBMIN); positions depend on the configuration; $\cdot_{\overline{L}}$ marks left children that are not the leftmost, $\cdot_{\overline{R}}$ marks right children that are not the rightmost.

(b) Graph-based parser (GBMIN); positions are: heads (h), dependents (d), children of d (c), siblings (s), grandparents (g), h,d$_{\overline{\pm i}}$ tokens at distance $\pm i$ from h or d which are none of h, d, c, s, or g.

Figure 5: Positions with the highest impact on the MLP scores (blue crosses) and their frequency (gray bars).

b_0 have the highest influence on the decision of the parser. The next two positions are s_{1R} and s_{0L}. Interestingly, those are the same positions which used as features caused the biggest gains in performance for the models which *did not* use BiLSTMs (see Figure 3a). They are much less frequent than b_1 but when they are present the model is strongly influenced by them. After b_1 we can notice positions which are not part of the manually designed extended feature set of TBEXT, such as $s_{0\overline{L}}$ (left children of s_0 that are not the leftmost).

Graph-based parser. For the graph-based parser we group tokens according to their position in the full predicted tree. We then bucket the impacts into: heads (h), dependents (d), children (i.e., dependents of dependents) (c), siblings (s), and grandparents (i.e., heads of heads) (g). Words which do not fall into any of those categories are grouped according to their surface distance from heads and dependents. For example, $h_{\overline{\pm 2}}$ are tokens two positions away from the head which do not act as dependent, child, sibling, or grandparent.

Figure 5b presents 10 positions with the highest impact and the number of arcs for which they are present (gray bars). As expected, heads and dependents have the highest impact on the scores of arcs, much higher than any of the other tokens. Interestingly, among the next three bins with the highest impact are children and siblings. Children are less frequent than structurally unrelated tokens at distance 1 ($h_{\overline{\pm 1}}$, $d_{\overline{\pm 1}}$), and much less frequent than $h_{\overline{\pm 2}}$ or $d_{\overline{\pm 2}}$ but they influence the final scores more. The interesting case is siblings – they not only have a strong average impact but they are also very frequent, suggesting that they are very important for the parsing accuracy.

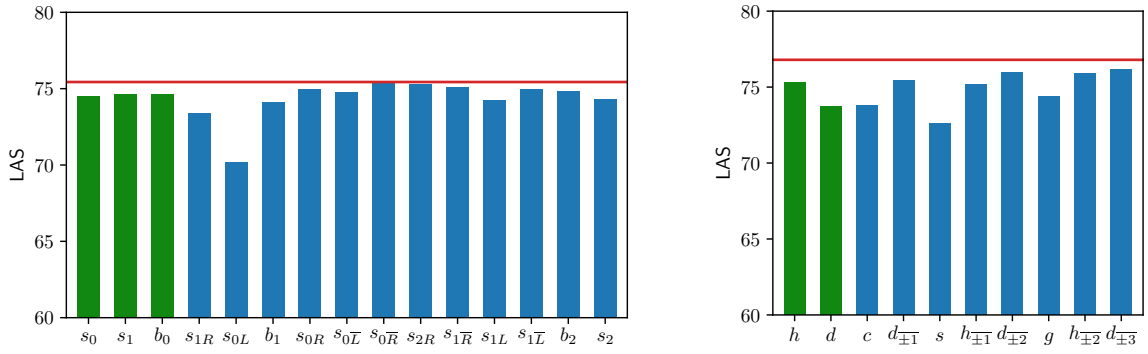
The results above show that the implicit structural context is not only present in the models, but also more diverse than when incorporated through conventional explicit structural features.

5.3 Structure and Performance

Finally, we investigate if the implicit structural context is important for the performance of the parsers. To do so, we take tokens at structural positions with the highest impact and train new ablated models in which the information about those tokens is dropped from the BiLSTM layer. For example, while training an ablated model without s_{0L}, for every configuration we re-calculate all the BiLSTM vectors as if s_{0L} was not in the sentence. When there is more than one token at a specific position, for example $s_{0\overline{L}}$ or c (i.e., children of the dependent), we pick a random one to drop. That way every ablated model looses information about at most one word.

We note that several factors can be responsible for drops in performance of the ablated models. For example, the proposed augmentation distorts distance between tokens which might have an adverse impact on the trained representations. Therefore, in the following comparative analysis we interpret the obtained drops as an approximation of how much particular tokens influence the performance of the models.

Transition-based parser. Figure 6a presents the drops in the parsing performance for the ab-



(a) Transition-based parser (TBMIN) (b) Graph-based parser (GBMIN)

Figure 6: The performance drops when tokens at particular positions are removed from the BiLSTM encoding. The red line marks average LAS of uninterrupted model. Feature sets of both models are highlighted in green.

lated models.[9] First of all, removing the vectors $\{\overrightarrow{\overleftarrow{s_0}}, \overrightarrow{\overleftarrow{s_1}}, \overrightarrow{\overleftarrow{b_0}}\}$ (marked in green on the plot) only from the BiLSTM layer (although they are still used as features) causes visible drops in performance. One explanation might be that when the vector $\overrightarrow{\overleftarrow{s_0}}$ is recalculated without knowledge of s_1 the model loses information about the distance between them. Secondly, we can notice that other drops depend on both the impact and frequency of positions. The biggest declines are visible after removing s_{0L} and s_{1R} – precisely the positions which we found to have the highest impact on the parsing decisions. Interestingly, the positions which were not a part of the TBEXT feature set, such as $s_{0\overline{L}}$ or $s_{1\overline{R}}$, although not frequent are important for the performance.

Graph-based parser. Corresponding results for the graph-based parser are presented in Figure 6b (we use gold-standard trees as the source of information about structural relations between tokens). The biggest drop can be observed for ablated models without siblings. Clearly, information coming from those tokens implicitly into MLP is very important for the final parsing accuracy. The next two biggest drops are caused by lack of children and grandparents. As we showed in Figure 5b children, although less frequent, have a stronger impact on the decision of the parser. But dropping grandparents also significantly harms the models.

We conclude that information about partial subtrees is not only present when the parser makes final decisions but also strongly influences those decisions. Additionally, the deteriorated accuracy of the ablated models shows that the implicit structural context can not be easily compensated for.

6 Related Work

Feature extraction. Kiperwasser and Goldberg (2016) and Cross and Huang (2016) first applied BiLSTMs to extract features for transition-based dependency parsers. The authors demonstrated that an architecture using only a few positional features (four for the arc-hybrid system and three for arc-standard) is sufficient to achieve state-of-the-art performance. Shi et al. (2017) showed that this number can be further reduced to two features for arc-hybrid and arc-eager systems. Decreasing the size of the feature set not only allows for construction of lighter and faster neural networks (Wang and Chang, 2016; Vilares and Gómez-Rodríguez, 2018) but also enables the use of exact search algorithms for several projective (Shi et al., 2017) and non-projective (Gómez-Rodríguez et al., 2018) transition systems. A similar trend can be observed for graph-based dependency parsers. State-of-the-art models (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2016) typically use only two features of heads and dependents, possibly also incorporating their distance (Wang and Chang, 2016). Moreover, Wang and Chang (2016) show that arc-factored BiLSTM-based parsers can compete with conventional higher-order models in terms of accuracy.

None of the above mentioned efforts address the question how dependency parsers are able to compensate for the lack of structural features. The very recent work by de Lhoneux et al. (2019) looked into this issue from a different perspec-

[9] It is worth noting that not all of the models suffer from the ablation. For example, dropping vectors s_{2R} causes almost no harm. This suggests that re-calculating the representations multiple times does not have a strong negative effect on training.

tive than ours – composition. They showed that composing the structural context with recursive networks as in Dyer et al. (2015) is redundant for the K&G transition-based architecture. The authors analyze components of the BiLSTMs to show which of them (forward v. backward LSTM) is responsible for capturing subtree information.

RNNs and syntax. Recurrent neural networks, which BiLSTMs are a variant of, have been repeatedly analyzed to understand whether they can learn syntactic relations. Such analyses differ in terms of: (1) methodology they employ to probe what type of knowledge the representations learned and (2) tasks on which the representations are trained on. Shi et al. (2016) demonstrated that sequence-to-sequence machine-translation systems capture source-language syntactic relations. Linzen et al. (2016) showed that when trained on the task of number agreement prediction the representations capture a non-trivial amount of grammatical structure (although recursive neural networks are better at this task than sequential LSTMs (Kuncoro et al., 2018)). Blevins et al. (2018) found that RNN representations trained on a variety of NLP tasks (including dependency parsing) are able to induce syntactic features (e.g., constituency labels of parent or grandparent) even without explicit supervision. Finally, Conneau et al. (2018) designed a set of tasks probing linguistic knowledge of sentence embedding methods.

Our work contributes to this line of research in two ways: (1) from the angle of methodology, we show how to employ derivatives to pinpoint what syntactic relations the representations learn; (2) from the perspective of tasks, we demonstrate how BiLSTM-based dependency parsers take advantage of structural information encoded in the representations. In the case of constituency parsing Gaddy et al. (2018) offer such an analysis. The authors show that their BiLSTM-based models implicitly learn the same information which was conventionally provided to non-neural parsers, such as grammars and lexicons.

7 Discussion and Conclusion

We examined how the application of BiLSTMs influences the modern transition- and graph-based parsing architectures. The BiLSTM-based parsers can compensate for the lack of traditional structural features. Specifically, the features drawn from partial subtrees become redundant because the parsing models encode them implicitly.

The main advantage of BiLSTMs comes with their ability to capture not only surface but also syntactic relations. When the representations are trained together with a parser they encode structurally-advanced relations such as heads, children, or even siblings and grandparents. This structural information is then passed directly (through feature vectors) and indirectly (through BiLSTMs encoding) to MLP and is used for scoring transitions and arcs. Finally, the implicit structural information is important for the final parsing decisions: dropping it in ablated models causes their performance to deteriorate.

The introduction of BiLSTMs into dependency parsers has an additional interesting consequence. The classical transition- and graph-based dependency parsers have their strengths and limitations due to the trade-off between the richness of feature functions and the inference algorithm (McDonald and Nivre, 2007). Our transition- and graph-based architectures use the same word representations. We showed that those representations trained together with the parsers capture syntactic relations in a similar way. Moreover, the transition-based parser does not incorporate structural features through the feature set. And the graph-based parser makes use of far away surface tokens but also structurally related words. Evidently, the employment of BiLSTM feature extractors blurs the difference between the two architectures. The one clear advantage of the graph-based parser is that it performs global inference (but exact search algorithms are already being applied to projective (Shi et al., 2017) and non-projective (Gómez-Rodríguez et al., 2018) transition systems). Therefore, an interesting question is if integrating those two architectures can still be beneficial for the parsing accuracy as in Nivre and McDonald (2008). We leave this question for future work.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) via the SFB 732, project D8. We would like to thank the anonymous reviewers for their comments. We also thank our colleagues Anders Björkelund, Özlem Çetinoğlu, and Xiang Yu for many conversations and comments on this work.

References

Terra Blevins, Omer Levy, and Luke Zettlemoyer. 2018. Deep RNNs Encode Soft Hierarchical Syntax. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 14–19, Melbourne, Australia. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750. Association for Computational Linguistics.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396–1400.

Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, Melbourne, Australia. Association for Computational Linguistics.

James Cross and Liang Huang. 2016. Incremental Parsing with Minimal Features Using Bi-Directional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2016. Deep Biaffine Attention for Neural Dependency Parsing. *CoRR*, abs/1611.01734.

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. Association for Computational Linguistics.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

Jason M. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's Going On in Neural Constituency Parsers? An Analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Tianze Shi, and Lillian Lee. 2018. Global Transition-based Non-projective Dependency Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2664–2675, Melbourne, Australia. Association for Computational Linguistics.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5):602–610.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. LSTMs Can Learn Syntax-Sensitive Dependencies Well, But Modeling Structure Makes Them Better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436. Association for Computational Linguistics.

Miryam de Lhoneux, Miguel Ballesteros, and Joakim Nivre. 2019. Recursive subtree composition in lstm-based dependency parsing. *arXiv preprint arXiv:1902.09781*.

Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017. From raw text to universal dependencies - look, no tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 207–217. Association for Computational Linguistics.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

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'05)*, pages 91–98. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

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*.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order crfs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332. Association for Computational Linguistics.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The Dynamic Neural Network Toolkit. *CoRR*, abs/1701.03980.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain.

Joakim Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359. Association for Computational Linguistics.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76. Association for Computational Linguistics.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).

Joakim Nivre and Ryan McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio. Association for Computational Linguistics.

Emily Pitler. 2014. A Crossing-Sensitive Third-Order Factorization for Dependency Parsing. *Transactions of the Association for Computational Linguistics*, 2:41–54.

Nils Reimers and Iryna Gurevych. 2018. Why comparing single performance scores does not allow to draw conclusions about machine learning approaches. *arXiv preprint arXiv:1803.09578*.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23. Association for Computational Linguistics.

Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does String-Based Neural MT Learn Source Syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534. Association for Computational Linguistics.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018. An Investigation of the Interactions Between Pre-Trained Word Embeddings, Character Models and POS Tags in Dependency Parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720. Association for Computational Linguistics.

David Vilares and Carlos Gómez-Rodríguez. 2018. Transition-based Parsing with Lighter Feed-Forward Networks. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 162–172. Association for Computational Linguistics.

Wenhui Wang and Baobao Chang. 2016. Graph-based Dependency Parsing with Bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315, Berlin, Germany. Association for Computational Linguistics.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193. Association for Computational Linguistics.

Zhisong Zhang and Hai Zhao. 2015. High-order Graph-based Neural Dependency Parsing. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation*, pages 114–123.

A Appendix

Word embedding dimension	100
POS tag embedding dimension	20
Hidden units in MLP	100
BiLSTM layers	2
BiLSTM dimensions	125
α for word dropout	0.25
Trainer	Adam
Non-lin function	tanh

Table 2: Hyperparameters for the parsers.

	en-ptb	ar	en	fi	grc	he	ko	ru	sv	zh
TbMin	0.237	0.323	0.207	0.163	0.382	0.391	0.740	0.282	0.295	0.398
TbExt	0.211	0.191	0.176	0.323	0.472	0.454	0.456	0.408	0.257	0.267
GbMin	0.146	0.179	0.212	0.157	0.340	0.269	0.300	0.228	0.379	0.408
GbSibl	0.103	0.186	0.149	0.219	0.372	0.229	0.163	0.169	0.195	0.441

Table 3: Standard deviation for results in Table 1.

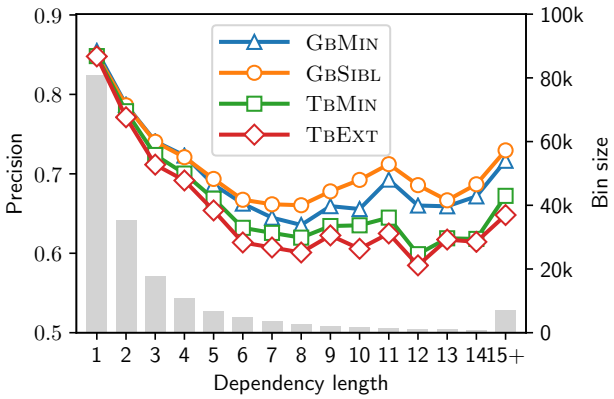


Figure 7: Dependency precision relative to arc length on development sets.