Evaluating Universal Dependency Parser Recovery of Predicate Argument Structure via CompChain Analysis

Sagar Indurkhya
LIDS/IDSS, Dept. of EECS
MIT
32 Vassar St.
Cambridge, MA 02139
`indurks@mit.edu`

Beracah Yankama
LIDS/IDSS, Dept. of EECS
MIT
32 Vassar St.
Cambridge, MA 02139
`beracah@mit.edu`

Robert C. Berwick
LIDS/IDSS, Dept. of EECS
MIT
Room 32D-728
32 Vassar St.
Cambridge, MA 02139
`berwick@csail.mit.edu`

Abstract

Accurate recovery of predicate-argument structure from a Universal Dependency (UD) parse is central to downstream tasks such as extraction of semantic roles or event representations. This study introduces *compchains*, a categorization of the hierarchy of predicate dependency relations present within a UD parse. Accuracy of compchain classification serves as a proxy for measuring accurate recovery of predicate-argument structure from sentences with embedding. We analyzed the distribution of compchains in three UD English treebanks, EWT, GUM and LinES, revealing that these treebanks are sparse with respect to sentences with predicate-argument structure that includes predicate-argument embedding. We evaluated the *CoNLL 2018 Shared Task* UDPipe (v1.2) baseline (dependency parsing) models as compchain classifiers for the EWT, GUMS and LinES UD treebanks. Our results indicate that these three baseline models exhibit poorer performance on sentences with predicate-argument structure with more than one level of embedding; we used compchains to characterize the errors made by these parsers and present examples of erroneous parses produced by the parser that were identified using compchains. We also analyzed the distribution of compchains in 58 non-English UD treebanks and then used compchains to evaluate the *CoNLL'18 Shared Task* baseline model for each of these treebanks. Our analysis shows that performance with respect to compchain classification is only weakly correlated with the official evaluation metrics (LAS, MLAS and BLEX). We identify gaps in the distribution of compchains in several of the UD treebanks, thus providing a roadmap for how these treebanks may be supplemented. We conclude by discussing how compchains provide a new perspective on the sparsity of training data for UD parsers, as well as the accuracy of the resulting UD parses.

1 Introduction

The Universal Dependencies (UD) project (De Marneffe et al., 2014; Nivre et al., 2016) is a multilingual annotation scheme for dependency grammars that has gained wide usage (Zeman et al., 2017; Kong et al., 2017; Qi et al., 2020). To this extent, automatically identifying whether a dependency parse[1] is correct or incorrect, as well as the potential source of such errors, becomes an important part of NLP pipelines. For example, such identification can prevent errors from propagating to downstream applications such as the identification of predicate-argument structure, involved in *semantic role labeling* and *sentiment analysis*.[2] Furthermore, embedding of sentences within sentences, and in particular embedding of predicate-argument structures within one another, is one of the ways in which humans have the capability to generate an infinity of different <sentences, meaning> pairings, and so it is important to evaluate whether a UD parser can accurately recover the predicate-argument structure of sentences with embedding. Thus, characterizing the limits of how accurately and consistently UD parsers assign predicate-argument structure in the context of correct UD annotation also becomes important (Nivre and Fang, 2017; Oepen et al., 2017; Fares et al., 2018; White et al., 2016; Reddy et al., 2017; Mille et al., 2018). That is the goal of this study.

In this study we introduce *compchains*, a categorization of the hierarchy of predicate dependency relations present within a Universal Dependency (UD) parse; this categorization serves as a proxy for

[1] This study only considers dependency parse trees annotated with UD. We refer to such a parse tree as a UD parse tree.

[2] Furthermore, (Surdeanu et al., 2003) has demonstrated that correct annotation of predicate-argument structure can improve the performance of information extraction systems.

predicate-argument structure. We use compchains to evaluate the accuracy of three (English) *CoNLL 2018 Shared Task* baseline models for the UDPipe dependency parser (Zeman et al., 2018). We found that the baseline model for the EWT UD treebank was more accurate than the baseline models for the LinES and GUM UD treebanks. We then use compchains to characterize the errors (relevant to predicate-argument structure) made by these models. We found that the accuracy of all three models dropped significantly when restricting the test set to samples with predicate-argument structure with embedding. Finally, we extended the analysis above to languages other than English, computing the distribution of compchains in 58 UD treebanks and evaluating the performance of the corresponding *CoNLL 2018 Shared Task* baseline models (for the UDPipe parser) as compchain classifiers. We conclude by discussing deficiencies in the distribution of predicate-argument structure with embedding present in the UD treebanks, as identified by our analysis.

2 Related Work

This section reviews prior work on the evaluation of (Universal) dependency parsers and the characterization of the errors these parsers make. The *CoNLL Shared Task* is a well established benchmark for evaluating the performance of multilingual (Universal) dependency parsers (Buchholz and Marsi, 2006; Nivre et al., 2007; Zeman et al., 2017, 2018). The task uses a number of metrics to evaluate the accuracy of the parser including: UAS (unlabeled attached score), LAS (labeled attachment score), CLAS (Content-word LAS) (Nivre and Fang, 2017), MLAS (Morphologically-aware LAS) and BLEX (BiLEXical Dependency Score). However, these metrics rely on the attachment accuracy (of dependency relations)[3] and do not take into account that errors cascade – i.e. if the parser incorrectly attaches a dependency relation, it may then be forced to make yet another incorrect attachment (Ng and Curran, 2015), thus making it difficult to identify the provenance of the error.

In light of this, efforts to further characterize the errors have proceeded in several directions. One direction involves studying whether and how the parsing errors are a result of the design of the dependency parser: (McDonald and Nivre, 2007) characterizes and compares the errors produced by graph-based dependency parsers (e.g. the MST-Parser by (McDonald and Pereira, 2006); see also (Kiperwasser and Goldberg, 2016; Cheng et al., 2016; Zhang et al., 2016)) and transition-based dependency parsers (e.g. the MaltParser by (Nivre et al., 2006)); (Zhang and Clark, 2008) shows how the two approaches to dependency parsing may be combined and documents the resulting improvement in performance.

An alternative direction involves characterizing the errors in the context of linguistic theory – e.g. (Kummerfeld et al., 2012) has introduced a method for classifying erroneous parse trees by repairing the tree with a series of tree-transformations, with each tree-transformation having a linguistic interpretation; (Mahler et al., 2017) has shown that it is possible to systematically break NLP systems for sentiment analysis by editing sentences with linguistically interpretable transformations. *In this study we pursue this latter direction, opting to characterize erroneous parse trees by classifying their predicate-argument structure using compchains.*

3 Compchains

Within a UD parse tree, predicate-argument structure[4] is encoded by *core argument* dependency relations, along with the special dependency relation *root*.[5] The *core-argument* dependency relations fall into two classes: *predicate* relations and *nominal* relations. In this study, we limit our attention to the two *predicate* dependency relations that encode embedding of clausal complements: (i) *ccomp* – a dependent, clausal complement, and (ii) *xcomp* – a clausal complement lacking a subject; the subject is determined by an argument that is external to the *xcomp*, usually the object (or otherwise subject) of the next higher clause.[6] We will focus on categorizing sequences of these two dependency relations (with POS marked as *Verb*) that originate from the root of a dependency tree (intuitively, the spine of the predicate-argument structure). This notion is formalized as follows:

Definition. A **compchain** is a finite sequence of dependency relations that traces a path starting at

[3]E.g. UAS (unlabeled attachment score) and LAS (labeled attachment score).

[4]See (Hale, 1993; Hale and Keyser, 2002) for further reference on predicate-argument structure.

[5]See `universaldependencies.org/u/dep/` for more details.

[6]*xcomp* is often used to model control/raising constructions in which an argument in the embedded clause establishes a syntactic relation with the predicate in the matrix clause.

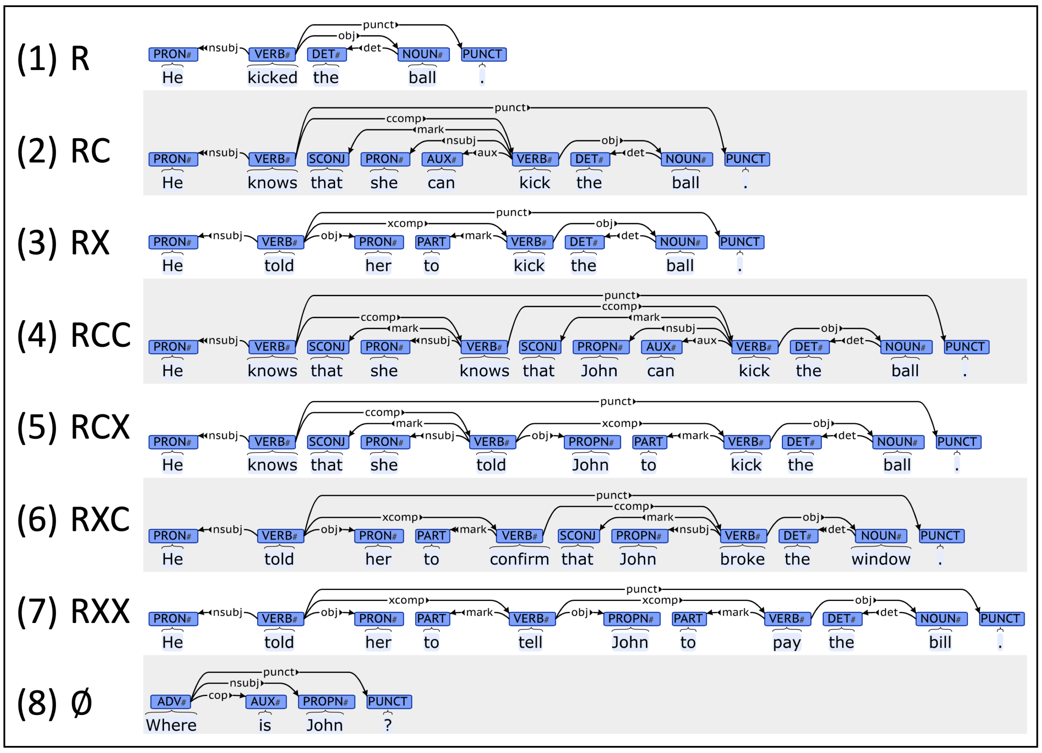
(1) R
He kicked the ball .
(2) RC
He knows that she can kick the ball .
(3) RX
He told her to kick the ball .
(4) RCC
He knows that she knows that John can kick the ball .
(5) RCX
He knows that she told John to kick the ball .
(6) RXC
He told her to confirm that John broke the window .
(7) RXX
He told her to tell John to pay the bill .
(8) ∅
Where is John ?

Figure 1: Examples of compchain classifications (left) for eight UD parses (right) produced by the UDv2.2 EWT baseline model using UDPipe 1.2. In each parse, the node with no incoming dependency relations is the *root*. Sentence 8 is classified as the ∅ compchain because the root is not marked as *VERB*.

the root node of a dependency parse tree and passing through only *xcomp* and *ccomp* dependency relations, subject to the constraints that: (i) every node in a compchain must have the POS tag of *Verb*; (ii) no node in a compchain should have a child dependency relation with POS *Verb* that is either an *xcomp* or *ccomp* and is not in the compchain as well.[7] We denote a compchain by listing the sequence of dependency relations, starting from the root of the tree, using the notation: R = *root*; X = *xcomp*; C = *ccomp*. E.g. we would denote the compchain $[root \rightarrow xcomp \rightarrow ccomp]$ as *RXC*. See Figure-1 for examples of UD parses and their compchain classifications.

One way to evaluate (indirectly) how well a UD parser can identify predicate-argument structure for sentences in a UD treebank is to evaluate whether the UD parse assigned by the parser to a sentence in the treebank has the same compchain as the compchain associated with the gold UD parse listed for that sentence (in the treebank); we refer to this task as **compchain classification**. *Performance on the compchain classification task is a proxy for performance on the task of classifying predicate-argument structure that includes predicate-argument embedding. If a UD parser performs poorly on the compchain classification task, predicate-argument structure cannot be reliably recovered from an (output) UD parse tree via top-down traversal of the sequence of dependency relations that forms the associated compchain.* See Figure-2 for examples of incorrect compchain classifications that reflect the parser recovering incorrect predicate-argument structure.

4 Experiments

4.1 Evaluation of English UD Treebanks

We evaluated the performance of the CoNLL'18 shared task baseline (parsing) models for English as *compchain classifiers* using three UD (v2.2) English treebanks: the English Web Treebank (EWT), with a total of 16,622 sentences (Silveira et al., 2014; Schuster and Manning, 2016); the English side of the English-Swedish Parallel Treebank (LinES), with a total of 4,564 sentences (Ahren-

[7]This constraint serves to ensure that if a UD parse tree has a compchain, it is unique and may be derived deterministically. This constraint also implies that some valid UD parse trees do not have a compchain – e.g. a parse in which there are two *xcomp* dependency relations that are both children of the same node. We use the symbol ∅ to denote that a UD parse tree has no compchain.

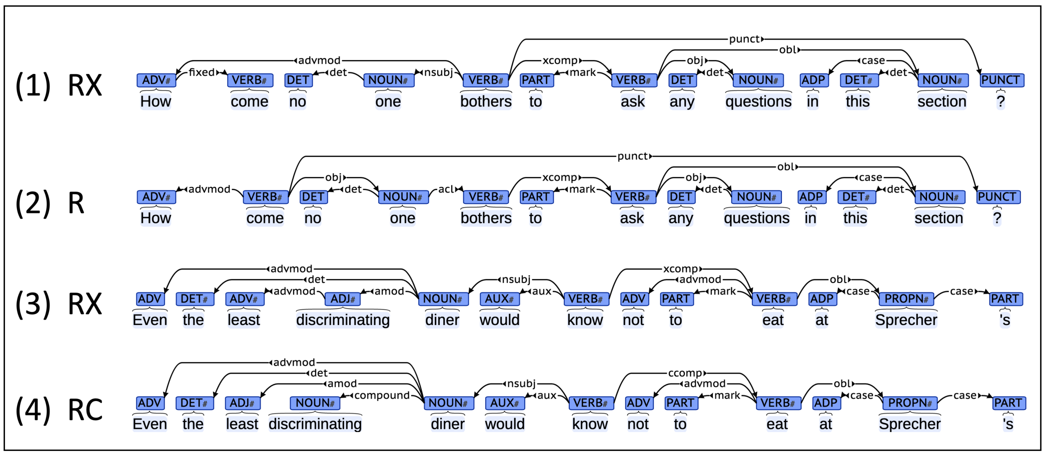
(1) RX
How come no one bothers to ask any questions in this section ?
(2) R
How come no one bothers to ask any questions in this section ?
(3) RX
Even the least discriminating diner would know not to eat at Sprecher 's
(4) RC
Even the least discriminating diner would know not to eat at Sprecher 's

Figure 2: Examples of compchain classifications (left) for four UD parses (right). The parses in (1) and (2) are for the sentence *"How come no one bothers to ask any questions in this section?"* The parses in (3) and (4) are for the sentence *"Even the least discriminating diner would know not to eat at Sprecher's."* Both sentences were taken from the UDv2.2 English Web Treebank. (1) and (3) are gold parse from the treebank whereas (2) and (4) are produced by UDPipe using the CoNLL'18 baseline language model for UDv2.2 EWT. Both (2) and (4) are incorrectly classified, reflecting that these two parses encode misinterpretations (compared to the interpretations in their respective gold parses – i.e. (1) and (3)).

berg, 2007); and the GUM treebank, with a total of 4,390 sentences (Zeldes, 2017).[8]

We began by computing the distribution of compchains in each of the sections (train, dev, test) for each of the treebanks (see Table-1). We observed that although the training section of the EWT (UD) treebank includes a non-negligible number of UD parse trees that are classified (according to their corresponding Gold UD parse) as compchains with three or more dependency relations, the test section of the EWT (UD) treebank does not. This suggests that performing well on the task of parsing the test section of the EWT (UD) treebank need not indicate competency in parsing sentences with predicate-argument embedding of degree two or more. We also observed that the LinES and GUM treebanks have a negligible number of parse trees (across all sections) that are classified as compchains with three or more dependency relations – i.e. *RCC*, *RCX*, *RXC* and *RXX*.

Next, we evaluated the CoNLL'18 shared task baseline (parsing) models[9] for the three treebanks as compchain classifiers. We used UDPipe (v1.2), a transition-based non-projective dependency parser, to parse the test section of each of the three treebanks using the corresponding baseline model (Straka and Straková, 2017). We then classified the compchain of each UD parse and compared it to the compchain associated with the corresponding gold parse. We report the F-measures for this classification task in Table 2. We observed that the baseline model for EWT had the best performance as a compchain classifier. We also computed the per-compchain F-measures and observed that for all three baseline models, their per-compchain F1-score for *RX* was notably better than for *RC*. Here we observed a steep falloff in per-compchain F1-score as the number of dependency relations in a compchain increases. This suggests that either the parsers were not trained on enough examples of sentences with predicate-argument embedding, or that they did not adequately generalize from the limited number of examples that they were trained on.

Finally, we computed and analyzed the confusion matrix (i.e. error matrix) for each of the three baseline models, evaluating each model on the test section of its associated treebank. (see Figure 3) In each confusion matrix, off-diagonal entries count instances of parses with erroneous predicate-argument structure as indicated by the predicted compchain differing from the actual compchain (if two parse trees have different compchains, then their predicate-argument structure must differ as well). On-diagonal entries count instances of parses with correctly classified com-

[8]We used the pretrained word embeddings supplied with the CoNLL Shared Task for each of the three treebanks; these embeddings were produced with *word2vec* (Mikolov et al., 2013b,a).

[9]These UDPipe models were trained on the training section of the UDv2.2 EWT/LinES/GUM respectively. We also used the tagging and tokenization pipeline provided by UDPipe.

Compchain	EWT			LinES			GUM		
	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test
∅	5230	985	1065	591	191	224	879	201	268
R	5500	815	806	1767	608	580	1661	413	419
RC	758	79	79	135	43	43	171	43	33
RX	808	100	104	202	65	50	158	43	41
RCC	47	4	6	1	0	2	8	1	2
RCX	94	7	9	17	1	6	10	2	1
RXC	48	6	3	10	2	6	6	0	2
RXX	39	2	2	12	2	3	13	3	3
Total	12543	2002	2077	2738	912	914	2914	707	769

Table 1: Distributions of compchains across the three treebanks. Counts for compchains with four or more dependency relations are not listed here because their presence in the three treebanks was negligible, although they are included in the “Total” count. Although there are very few compchains with three or more dependency relations (e.g. RCC) in the test sections of the treebanks, there are a non-negligible number of them in the training sections.

Compchain	EWT				LinES				GUM			
	F1	Prec.	Rec.	Support	F1	Prec.	Rec.	Support	F1	Prec.	Rec.	Support
∅	0.94	0.95	0.94	1065	0.74	0.72	0.75	224	0.85	0.81	0.9	268
R	0.89	0.89	0.9	806	0.85	0.87	0.83	580	0.85	0.89	0.81	419
RC	0.73	0.72	0.73	79	0.43	0.44	0.42	43	0.54	0.53	0.55	33
RX	0.79	0.8	0.79	104	0.53	0.44	0.66	50	0.64	0.57	0.73	41
RCC	0.67	0.67	0.67	6	0	0	0	2	0.4	0.33	0.5	2
RCX	0.4	0.5	0.33	9	0.25	0.5	0.17	6	0.5	0.33	1	1
RXC	0.33	0.33	0.33	3	0.55	0.6	0.5	6	0	0	0	2
RXX	1	1	1	2	0.44	0.33	0.67	3	0.4	0.5	0.33	3
W. Avg.	0.9	0.9	0.9	2077	0.78	0.78	0.77	914	0.82	0.83	0.82	769

Table 2: F-measures for the compchain classification of the parse trees in the EWT, LinES and GUM (UD) treebanks. The left most column refers to the *true* compchain from the appropriate UD treebank. Each row has the F1-score for the evaluation of the parser (as a compchain classifier) on sentences in the treebank that had the listed compchain, except for the bottom most row, which is the total (weighted) F1-score over all compchains – i.e. performance as a multi-way classifier.

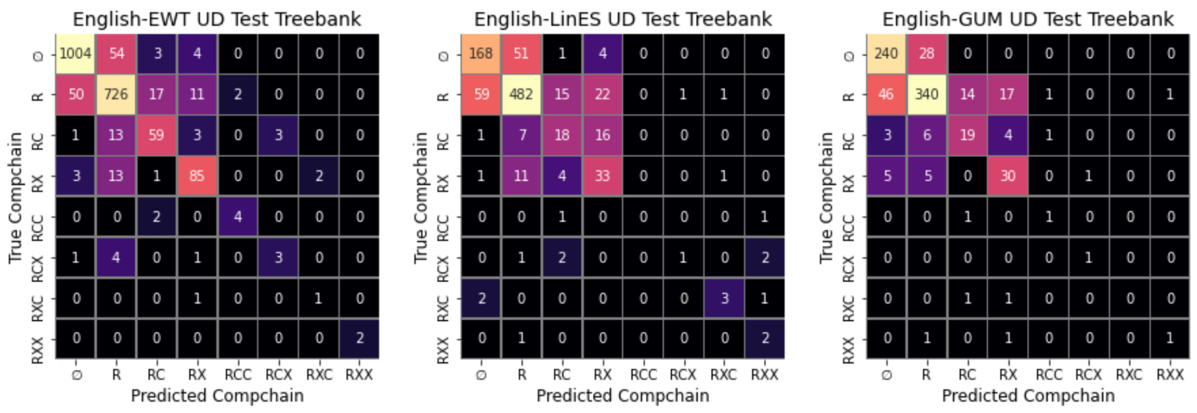

Figure 3: Confusion Matrices for Compchain Classification of the EWT, GUM and LinES UD (English) treebanks using their respective CoNLL’18 UDPipe Baseline Models.

pchains, which indicates that the parse may be correct (though it may well have errors not related to predicate-argument structure). We observed, for all three models, that compchains of length two or less were very rarely misclassified as compchains of length three or more, and that compchains of length two were often misclassified as the *R* compchain (see Figure-2 for an example of such a misclassification). We also observed that in the case of the baseline model for LinES, the compchain for *RC* is frequently misclassified as *RX*, but the compchain *RX* is rarely misclassified as *RC*; this asymmetry may reflect the difference in number of training examples in the LinES treebank – 135 in the case of *RC* and 202 in the case of *RX* (see Table-1).

4.2 Multilingual Evaluation of UD Treebanks

We also used the compchain classification task to evaluate the *CoNLL'18 shared task* baseline models (and the respective UD treebanks they were trained on) for languages other than English; this was motivated by the observation that since the UD treebanks are derived from a variety of textual sources, and thus have varying compchain distributions, we can use them collectively to evaluate and characterize the performance of the UDPipe dependency parser under various training conditions.

Figure 4 presents the distribution of compchains across 61 UD treebanks (including the three English treebanks analyzed earlier in this study).[10] Our analysis reveals that: (i) the UD treebanks for Hindi and Urdu have no instances of the compchain *RC* in either the training or test sections; (ii) the UD treebanks for Japanese, Korean, Turkish and Uyghur have no instances of the compchain *RC* in either the training or test sections; (iii) the UD treebanks for Hindi, Japanese, Turkish and Uyghur do not include *any* instances of compchains of length three or more (i.e. *RXX*, *RCC*, *RXC*, or *RCX*) in either the training or test sections.

We computed the F1-scores for the performance of each baseline model on the compchain classification task.[11] The F1-score for length-1 compchains is very weakly correlated with the F1-score for length-2 compchains, with $R^2 = 0.265$ (see Figure 5), and F1-scores for the two length-2 compchains (*RC* and *RX*) are also very weakly correlated, with $R^2 = 0.177$ (see Figure 6). This suggests that performance in recovering predicate-argument structures with differing embedding structures is largely unrelated and should be measured explicitly, just as the compchain classification task does. Additionally, we observe (as we did with the models trained on English treebanks) a rapid decline in the per-class F1-score as the length of the compchain increases, in particular for compchains of length two or more. (See Figure 7) This is revealing because, although the lack of compchains of length two or more in the UD treebanks suggests that we should not necessarily expect a dependency parser trained on the treebank to generalize out of the training domain, there is empirical evidence that humans *do* have the capacity to acquire a grammar from sentences with at most degree-1 embedding (corresponding to compchains of length 2) and then later correctly parse sentences with a degree of embedding of two or more (Wexler and Culicover, 1980; Morgan, 1986; Lightfoot, 1989); thus, the poor performance on compchains of length three or more suggests that the *CoNLL 2018 Shared Task* baseline models are not able to generalize beyond the distribution of syntactic structures they were trained upon, in contrast to human learners.

4.2.1 Impact of Word Ordering

Word ordering data (i.e. head-directionality) for each of the 61 languages in the UD treebanks was obtained from the WALS Online database (Dryer, 2013); we retrieved this information because the word ordering dictates whether a predicate precedes or succeeds its complement with respect to the linear ordering of the words in a sentence, and we wanted to understand whether this had an impact on the parser's performance on the compchain classification task. (See Table-5 in the appendix for the word-order of each language) The 47 languages with *verb-object* (VO) ordering had a median and mean weighted average F1-score of 0.85 and 0.88 respectively; the 18 languages with *object-verb* (OV) ordering had a median and mean weighted average F1-score of 0.86 and 0.85 respectively. It thus appears that the word-ordering does not appear to impact the weighted average F1-score. The F1-scores associated with compchains of length 2 (i.e. *RX* and *RC*) tell a different story: in the case of the *RC* compchain, the median F1-scores for *verb-object* and *object-verb* were 0.68 and 0.55 respectively, and in the case of the *RX* compchain, the median F1-scores for *verb-object* and *object-verb* were 0.72 and 0.42 respectively; thus for both compchains of length 2, models trained on *verb-object* ordered languages performed significantly better than models trained on *object-verb* ordered languages.[12] Given that the orderings of *verb-object* (i.e. head-initial) and *object-verb* (i.e. head-final) control whether a language will be associated with right-branching or left-branching structures respectively, our results suggest that the UDPipe

[10]See Table 4 in the appendix for a complete listing of the distribution of compchains in the Test and Training treebank for each of the 61 languages.

[11]See Table-5 for a complete listing of performance on the *compchain classification* task for each UD treebank using the associated baseline model, including a breakdown of performance per-compchain.

[12]These results also hold when comparing the mean F1-scores for compchains of length 1.

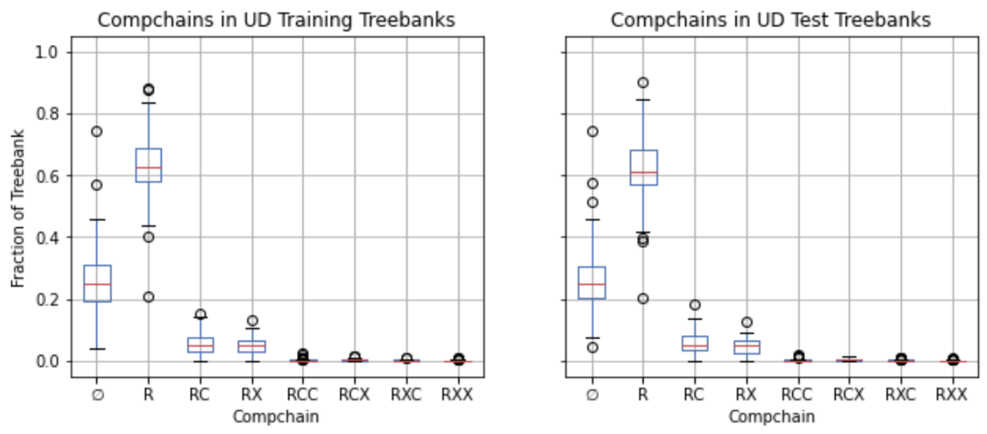

Figure 4: Distribution of Compchains in UD Training and Test Treebanks. 59 of the 61 languages had degree-2 compchains present in the test treebank; the languages with no degree-2 compchains in the test treebank were *turkish-imst* and *urdu-udtb*.

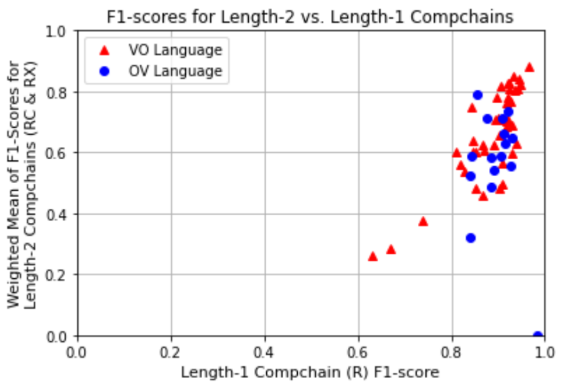

Figure 5: F1-scores for Length 2 vs. Length 1 compchains for each language in the UD treebank.

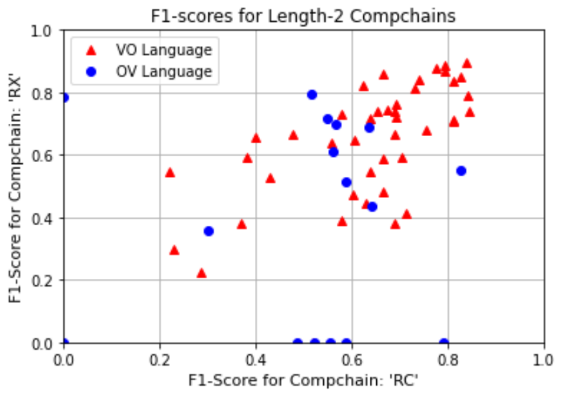

Figure 6: F1-scores for Length 2 compchains (i.e. *RC* and *RX*) for each language in the UD treebanks.

parser has difficulty dealing with left-branching structures.

4.2.2 Impact of Sentence Length

We carried out a regression analysis to investigate the relationship between the correctness of compchain classification and sentence length; this was

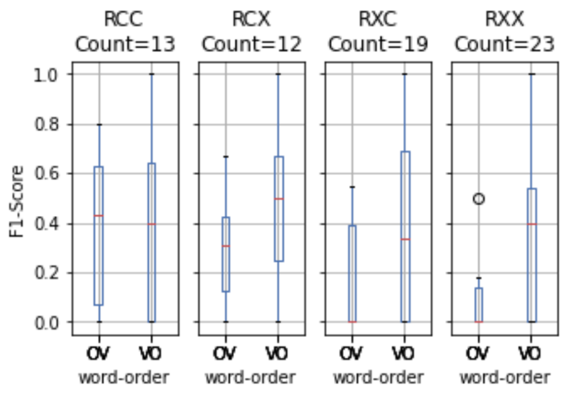

Figure 7: Distributions of F1-scores for length-3 compchains over all UD languages. For each length-3 compchain, F1-scores were reported for languages that had that compchain present in the test-treebank.

motivated by the observation that sentences with higher degrees of embedding, and thus longer compchains, tend to be longer sentences. We fitted a logistic function for each sentence in the test treebank, with the log of the sentence length (i.e. the number of tokens including punctuation) serving as the independent variable, and the (binary) dependent variable being whether the compchain associated with that sentence was correctly classified. We interpreted a good-fitting logistic function to indicate that compchain accuracy is dependent on sentence length. To evaluate the fit of the logistic function, we computed the Area Under Curve (AUC) measure of the Receiver Operator Characteristic (ROC) curve for the fitted logistic function. Figure 8 presents the distribution of AUCs for the test corpus of each of: (a) the 43 UD treebanks for languages with *verb-object* (VO) word-ordering, and (b) the 18 UD treebanks for langauges with

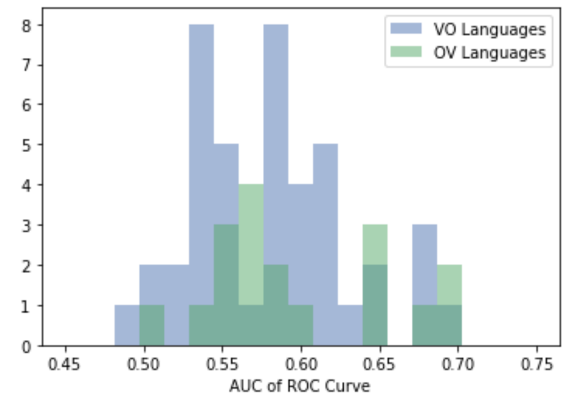

Figure 8: Histogram of Area-under-Curve (AUC) of Receiver Operator Characteristic (ROC) curve for Logistic Regression model of per-Sentence Compchain Classification Accuracy vs. log(Sentence Length). The AUC of ROC curve was computed for each UD test treebank.

object-verb (OV) word-ordering. We observe that the AUC for the majority of the treebanks falls between 0.55 and 0.65, and virtually none of the AUCs surpass 0.7, which is generally considered a minimum threshold for a binary-classifier to be considered accurate. Additionally, we observe that the OV languages tend to have a slightly higher AUC than the VO languages. We conclude that accuracy of compchain classification is weakly correlated with the log of the length of the sentence, and that this correlation is slightly higher for OV languages than for the VO languages. (Similar results were obtained when the analysis was carried out directly on the length of the sentence.)

4.2.3 Comparison with Other Eval. Metrics

In order to understand whether the compchain metric is simply a proxy for one of the three official evaluation metrics (LAS, BLEX and MLAS), we computed the pairwise linear correlation between each of the metrics for each of the 61 UD treebanks.[13] Table 3 presents the coefficient of determination for each pairing of the metrics. We observe that although LAS, MLAS and BLEX are all highly correlated with one another, they are weakly correlated with the compchain-metrics (i.e. weighted avg. of F1-score over all compchains and per-compchain F1-scores); notably, performance on compchain classification for *RX* is very weakly correlated with LAS, MLAS and BLEX ($R^2 < 0.1$).

[13]LAS, MLAS and BLEX scores for CoNLL Shared Task baseline models were obtained from https://universaldependencies.org/conll18/baseline.html#baseline-results.

	LAS	MLAS	BLEX
Compchain:W. Avg.	0.402	0.267	0.359
Compchain:R	0.329	0.180	0.277
Compchain:RC	0.399	0.305	0.388
Compchain:RX	0.074	0.089	0.096
LAS	1.000	0.780	0.918
MLAS	0.780	1.000	0.822
BLEX	0.918	0.822	1.000

Table 3: Coefficient of determination (R^2) for pairwise (linear) correlations of metric-scores over all CoNLL'18 Shared Task baseline models.

This suggests that the compchain metric is measuring an aspect of the parser's performance that is not brought to the fore by any of the three official evaluation metrics, and that *a baseline model having a good LAS, MLAS or BLEX score does not necessarily indicate that the model will correctly predict the embedding structure of a sentence with even a single level of embedding.*

5 Conclusion

In this study, we defined *compchains* and used them to evaluate how accurately a UD parser can parse sentences with predicate-argument structure that contains embedded clauses. We also used compchains to classify the errors, relevant to predicate-argument structure with embedding, made by a UD parser. Overall model performance on the compchain classification task (as measured by weighted F-measure) was found to be dominated by parse trees in the training set with no embedding (compchain *R*); closer inspection of per-compchain performance revealed that parser accuracy dropped precipitously as the degree of embedding in the predicate argument structure (i.e. length of compchain) increased. Finally, our results indicate that UD treebanks have very few parse trees with degree of embedding (i.e. length of compchain) greater than two. This presents an opportunity: if the test sets of the UD treebanks were augmented with parses with predicate-argument structure with degree of embeddings greater than two, then UD parsers can be evaluated in terms of their capacity to generalize from constructions (in the training set) with (mostly) low degree of embedding, just as a child must in some models of first language acquisition (Wexler and Culicover, 1980; Berwick, 1985; Lightfoot, 1989).

Acknowledgments

We would like to thank three anonymous reviewers for their valuable feedback and suggestions.

References

Lars Ahrenberg. 2007. Lines: An english-swedish parallel treebank. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA 2007)*, pages 270–273.

Robert C. Berwick. 1985. *The acquisition of syntactic knowledge*. MIT press.

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the tenth conference on computational natural language learning*, pages 149–164. Association for Computational Linguistics.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. *arXiv preprint arXiv:1608.02076*.

Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. 2014. Universal stanford dependencies: A cross-linguistic typology. In *LREC*, volume 14, pages 4585–4592.

Matthew S. Dryer. 2013. Order of subject, object and verb. In Matthew S. Dryer and Martin Haspelmath, editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Murhaf Fares, Stephan Oepen, Lilja Ovrelid, Jari Bjorne, and Richard Johansson. 2018. The 2018 shared task on extrinsic parser evaluation: on the downstream utility of english universal dependency parsers. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 22–33.

Kenneth Hale. 1993. On argument structure and the lexical expression of syntactic relations. In Ken Hale and Samuel J. Keyser, editors, *The view from Building 20: Essays in linguistics in honor of Sylvain Bromberger*. MIT Press.

Kenneth Locke Hale and Samuel Jay Keyser. 2002. *Prolegomenon to a theory of argument structure*, volume 39. MIT press.

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.

Lingpeng Kong, Chris Alberti, Daniel Andor, Ivan Bogatyy, and David Weiss. 2017. Dragnn: A transition-based framework for dynamically connected neural networks. *arXiv preprint arXiv:1703.04474*.

Jonathan K Kummerfeld, David Hall, James R Curran, and Dan Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059. Association for Computational Linguistics.

David Lightfoot. 1989. The child's trigger experience: Degree-0 learnability. *Behavioral and Brain Sciences*, 12(2):321–334.

Taylor Mahler, Willy Cheung, Micha Elsner, David King, Marie-Catherine de Marneffe, Cory Shain, Symon Stevens-Guille, and Michael White. 2017. Breaking NLP: Using morphosyntax, semantics, pragmatics and world knowledge to fool sentiment analysis systems. In *Proceedings of the First Workshop on Building Linguistically Generalizable NLP Systems*, pages 33–39, Copenhagen, Denmark. 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*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Simon Mille, Anja Belz, Bernd Bohnet, and Leo Wanner. 2018. Underspecified universal dependency structures as inputs for multilingual surface realisation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 199–209.

James L Morgan. 1986. *From simple input to complex grammar.* The MIT Press.

Dominick Ng and James R Curran. 2015. Identifying cascading errors using constraints in dependency parsing. 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)*, volume 1, pages 1148–1158.

Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan T McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *LREC*.

Joakim Nivre and Chiao-Ting Fang. 2017. Universal dependency evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies, 22 May, Gothenburg Sweden*, 135, pages 86–95. Linköping University Electronic Press.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The conll 2007 shared task on dependency parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Joakim Nivre, Johan Hall, Jens Nilsson, Svetoslav Marinov, et al. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225.

Stephan Oepen, L Ovrelid, Jari Bjorne, Richard Johansson, Emanuele Lapponi, Filip Ginter, and Erik Velldal. 2017. The 2017 shared task on extrinsic parser evaluation towards a reusable community infrastructure. *Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation*, pages 1–16.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Siva Reddy, Oscar Tackstrom, Slav Petrov, Mark Steedman, and Mirella Lapata. 2017. Universal semantic parsing. *arXiv preprint arXiv:1702.03196*.

Sebastian Schuster and Christopher D Manning. 2016. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *LREC*, pages 23–28. Portorož, Slovenia.

Natalia Silveira, Timothy Dozat, Marie-Catherine De Marneffe, Samuel R Bowman, Miriam Connor, John Bauer, and Christopher D Manning. 2014. A gold standard dependency corpus for english. In *LREC*, pages 2897–2904.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99.

Mihai Surdeanu, Sanda Harabagiu, John Williams, and Paul Aarseth. 2003. Using predicate-argument structures for information extraction. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*.

Kenneth. Wexler and Peter W. Culicover. 1980. *Formal principles of language acquisition*. MIT Press.

Aaron Steven White, Drew Reisinger, Keisuke Sakaguchi, Tim Vieira, Sheng Zhang, Rachel Rudinger, Kyle Rawlins, and Benjamin Van Durme. 2016. Universal decompositional semantics on universal dependencies. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1713–1723.

Amir Zeldes. 2017. The gum corpus: Creating multilayer resources in the classroom. *Lang. Resour. Eval.*, 51(3):581–612.

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.

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

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2016. Dependency parsing as head selection. *arXiv preprint arXiv:1606.01280*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference* *on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics.

A Appendix

Table-4 presents the distribution of compchains across 61 UD treebanks (including the three English treebanks analyzed earlier in this study). Table-5 presents the F1-scores for the performance of each baseline models on the compchain classification task. The rows of Table 4 and Table 5 were seriated using the *Google OR-Tools library* so that rows with similar values appear close together: Table 4 is seriated so that languages with similar compchain distributions are clustered together; Table 5 is seriated so that languages with similar F1-scores are clustered together.

Computing Infrastructure: All experiments reported in this study were performed on a MacBook Pro (Retina, 15-inch, Late 2013) with a 2.3 GHz Intel Core i7 processor, and 16 GB of 1600 MHz DDR3 RAM. We used Python v3.7.9, Pandas v1.2.1 and Matplotlib v3.2.1.

(The remainder of this page intentionally blank. Please see the next page.)

Treebank	Total	R	RC	RX	RCC	RCX	RXC	RXX
vietnamese-vtb	1400/800	48.1/48.9	11.4/10.8	10.9/8.2	1.1/1.5	1.6/1.4	0.6/1.0	0.9/0.2
chinese-gsd	3997/500	52.6/53.4	11.1/11.4	10.0/8.6	1.2/0.6	1.5/1.0	1.1/0.8	0.9/0.4
catalan-ancora	13123/1846	65.0/66.5	10.1/7.9	5.8/6.1	0.6/0.2	0.6/0.5	0.2/0.1	0.2/0.2
serbian-set	2935/491	53.2/58.0	10.4/10.0	4.9/5.5	0.7/0.4	1.7/0.8	0.3/-	-/-
spanish-ancora	14305/1721	59.5/58.8	12.2/13.5	5.3/5.5	0.8/0.5	0.9/0.8	0.1/-	0.1/0.2
greek-gdt	1662/456	62.1/59.9	13.0/9.9	5.6/5.0	1.0/0.4	1.1/0.7	0.4/0.7	-/-
galician-ctg	2272/861	58.5/57.6	14.0/13.6	1.8/1.3	2.2/1.4	0.2/0.6	0.1/-	-/0.1
persian-seraji	4798/600	46.5/46.5	15.1/18.2	0.1/-	2.4/2.2	-/-	-/-	-/-
romanian-rrt	8043/729	72.4/71.9	10.1/12.1	1.5/2.3	0.7/1.2	0.1/-	0.0/-	-/-
korean-kaist	23010/2287	61.3/61.1	8.3/6.8	0.1/-	0.3/0.2	-/-	-/-	-/-
bulgarian-btb	8907/1116	66.1/64.7	9.5/14.0	3.5/1.3	0.7/0.6	0.3/0.1	0.3/-	0.1/-
slovak-snk	8483/1061	66.1/59.8	8.2/2.1	4.6/3.3	0.3/-	1.0/0.5	0.4/0.1	0.0/-
portuguese-bosque	8329/477	58.1/59.3	7.6/8.4	4.5/2.5	0.5/0.2	0.6/0.4	0.3/-	0.3/-
latin-proiel	15906/1260	67.9/64.6	7.6/6.7	5.3/6.0	0.4/0.6	0.3/0.6	0.2/0.1	0.1/0.1
latvian-lvtb	5424/1228	62.3/60.7	7.7/7.2	6.2/4.8	0.4/0.2	0.7/0.9	0.5/0.7	0.1/-
czech-fictree	10160/1291	63.7/59.5	7.7/8.1	6.3/8.1	0.3/0.4	1.0/1.0	0.5/0.7	0.0/-
hebrew-htb	5241/491	62.5/61.7	6.5/3.7	6.8/6.1	0.2/-	0.7/1.0	0.1/-	-/-
english-ewt	12543/2077	43.8/38.8	6.0/3.8	6.4/5.0	0.4/0.3	0.7/0.4	0.4/0.1	0.3/0.1
basque-bdt	5396/1799	72.5/74.1	5.8/5.3	5.9/5.3	0.1/0.1	0.7/0.8	0.3/0.4	0.1/-
swedish-lines	2738/914	63.3/65.4	6.5/6.0	5.7/6.2	0.2/0.1	0.6/0.4	0.3/0.4	0.2/0.3
english-gum	2914/769	57.0/54.5	5.9/4.3	5.4/5.3	0.3/0.3	0.3/0.1	0.2/0.3	0.4/0.4
ancient_greek-proiel	15015/1047	72.9/72.5	5.6/4.9	5.2/6.7	0.2/0.3	0.2/0.6	0.1/0.1	0.1/0.1
slovenian-ssj	6478/788	65.5/62.2	6.0/7.4	4.8/5.7	0.2/0.4	0.6/0.8	0.4/0.6	0.0/0.1
danish-ddt	4383/565	59.5/57.0	6.6/9.9	3.7/4.1	0.2/0.4	0.3/0.5	0.0/0.2	0.0/-
finnish-ftb	14981/1867	65.0/64.9	5.4/6.4	4.1/4.0	0.1/0.4	0.3/0.4	0.2/0.1	0.0/-
norwegian-bokmaal	15696/1939	59.8/63.2	5.0/6.0	3.0/3.2	0.1/0.1	0.2/0.4	0.1/0.1	0.0/0.1
norwegian-nynorsk	14174/1511	53.4/54.5	4.7/4.0	3.0/2.9	0.2/0.1	0.2/0.1	0.1/-	0.1/-
italian-postwita	5368/674	46.0/47.2	4.3/4.0	3.3/2.4	0.1/0.1	0.2/-	0.2/0.3	0.1/0.1
swedish-talbanken	4303/1219	66.7/66.0	3.7/5.0	2.3/2.5	0.1/0.1	0.1/-	0.1/0.2	0.0/-
arabic-padt	6075/680	20.8/20.3	4.0/4.4	0.6/0.7	0.1/-	0.1/0.1	0.1/-	-/-
korean-gsd	4400/989	69.9/70.7	4.8/5.6	-/-	0.2/0.3	-/-	-/-	-/-
afrikaans-afribooms	1315/425	71.1/73.2	3.8/3.3	0.3/0.2	0.2/-	-/-	-/-	-/-
uyghur-udt	1656/900	81.0/79.6	3.1/3.7	-/-	-/-	-/-	-/-	-/-
japanese-gsd	7164/557	65.9/62.5	2.0/2.0	-/-	-/-	-/-	-/-	-/-
turkish-imst	3685/975	63.3/61.1	0.1/-	-/-	-/-	-/-	-/-	-/-
urdu-udtb	4043/535	87.9/90.3	-/-	0.0/-	-/-	-/-	-/-	-/-
hindi-hdtb	13304/1684	87.5/84.7	-/-	0.0/0.1	-/-	-/-	-/-	-/-
dutch-lassysmall	5789/876	40.1/39.8	1.1/0.8	1.8/1.5	0.0/-	0.1/-	-/-	0.0/0.1
german-gsd	13814/977	69.0/59.5	2.0/9.6	2.1/3.3	0.0/0.3	0.1/0.1	0.1/0.1	0.1/-
hungarian-szeged	910/449	75.6/75.7	0.7/0.7	4.2/6.2	-/-	0.1/-	-/-	-/-
italian-isdt	13121/482	61.0/69.9	3.5/1.9	3.9/3.1	0.2/0.6	0.2/0.2	0.1/-	0.1/-
dutch-alpino	12269/596	63.2/63.9	4.6/5.5	4.8/4.4	0.3/0.3	0.3/0.7	0.2/1.0	0.2/-
estonian-edt	20827/2737	60.4/59.8	3.4/4.2	5.1/4.6	0.1/0.1	0.4/0.4	0.2/0.1	0.1/0.1
finnish-tdt	12217/1555	60.1/58.7	3.0/4.1	5.1/5.8	0.0/0.1	0.3/0.3	0.2/0.3	0.1/0.1
ukrainian-iu	4513/783	62.6/65.1	3.1/4.1	5.9/5.1	0.0/-	0.3/0.5	0.2/-	0.1/-
russian-syntagrus	48814/6491	60.6/60.9	3.4/2.8	6.6/6.5	0.0/0.0	0.4/0.3	0.2/0.2	0.1/0.1
ancient_greek-perseus	11476/1306	83.7/68.3	4.1/12.6	7.0/7.6	0.2/0.5	0.4/1.1	0.1/0.2	0.2/0.4
latin-ittb	15808/750	52.0/51.5	5.0/4.0	6.8/9.2	0.2/-	0.2/0.3	0.4/-	0.1/-
czech-pdt	68495/10148	53.8/54.6	4.9/4.6	6.9/6.6	0.2/0.2	0.9/0.9	0.3/0.3	0.1/0.1
croatian-set	6983/1057	53.8/55.2	5.8/8.6	7.4/6.5	0.2/0.1	1.1/1.1	0.3/0.8	0.0/0.1
gothic-proiel	3387/1029	75.4/77.5	6.7/6.3	8.6/7.7	0.2/0.3	0.5/0.5	0.1/-	0.1/0.2
old_church_slavonic-proiel	4123/1141	80.4/82.6	6.0/5.2	8.3/7.2	0.1/0.1	0.8/0.4	-/0.1	0.3/0.1
english-lines	2738/914	64.5/63.5	4.9/4.7	7.4/5.5	0.0/0.2	0.6/0.7	0.4/0.7	0.4/0.3
polish-sz	6100/1100	72.5/72.9	5.0/5.3	7.6/7.2	0.0/0.2	0.6/0.6	0.4/0.4	0.1/-
old_french-srcmf	13909/1927	79.2/78.6	4.8/4.0	7.7/8.2	0.1/0.1	0.4/0.4	0.2/0.2	0.2/0.1
french-gsd	14554/416	61.0/54.6	3.1/3.8	8.2/8.7	0.2/0.7	0.5/1.0	0.2/-	0.3/0.5
polish-lfg	13774/1727	80.9/79.7	2.8/2.7	8.4/8.9	0.0/0.1	0.3/0.6	0.2/0.2	0.1/0.1
czech-cac	23478/628	59.2/61.3	2.2/2.4	6.8/6.7	0.1/-	0.4/0.3	0.3/0.2	0.1/0.2
french-spoken	1153/726	49.6/52.6	1.6/4.8	7.7/4.5	0.1/0.1	0.2/0.8	0.2/-	0.1/0.1
indonesian-gsd	4477/557	62.0/63.9	2.2/2.9	9.0/7.7	0.1/0.2	0.3/0.2	0.1/-	0.7/0.9
french-sequoia	2231/456	44.1/41.9	3.0/3.5	13.2/12.9	-/-	1.0/1.1	1.0/0.7	0.6/0.4

Table 4: Distribution of Compchains in UD 2.2 Gold Treebanks. The column *Total* presents the number of trees in the training and test sections of each treebank, and is formatted as $Count_{Training}/Count_{Test}$; the columns for each compchain present the percent of trees with that compchain in the training and test sections of the treebank respectively – e.g. with respect to the English-EWT treebank, 6% of the 12543 trees in the training section have the compchain *RC* whereas only 3.8% of the 2077 trees in the test section have the compchain *RC*. A dash ("-") indicates an absence of trees with that compchain (i.e. 0%).

Treebank	Word Order	W. Avg	R	RC	RX	RCC	RCX	RXC	RXX
polish-lfg	verb-object	0.94	0.97	0.84	0.89	1.00	0.87	0.29	1.00
french-gsd	verb-object	0.89	0.92	0.62	0.81	0.80	0.75	-	1.00
spanish-ancora	verb-object	0.89	0.92	0.81	0.78	0.55	0.60	-	0.67
english-ewt	verb-object	0.90	0.89	0.73	0.79	0.67	0.40	0.33	1.00
croatian-set	verb-object	0.90	0.92	0.79	0.87	0.67	0.75	0.71	1.00
czech-pdt	verb-object	0.92	0.93	0.79	0.89	0.72	0.80	0.64	0.67
finnish-tdt	verb-object	0.89	0.92	0.74	0.68	0.50	0.75	0.67	0.40
slovenian-ssj	verb-object	0.88	0.91	0.84	0.79	0.50	0.67	0.80	0.40
russian-syntagrus	verb-object	0.93	0.95	0.83	0.85	0.50	0.59	0.74	0.44
catalan-ancora	verb-object	0.91	0.94	0.81	0.83	0.25	0.53	1.00	0.67
czech-cac	verb-object	0.92	0.94	0.67	0.86	-	1.00	1.00	1.00
norwegian-bokmaal	verb-object	0.90	0.92	0.84	0.74	0.00	0.83	0.80	0.67
french-sequoia	verb-object	0.86	0.86	0.71	0.78	0.00	0.80	0.50	0.80
english-lines	verb-object	0.78	0.85	0.43	0.53	0.00	0.25	0.55	0.44
latin-proiel	object-verb	0.86	0.92	0.57	0.70	0.36	0.40	0.33	0.50
english-gum	verb-object	0.82	0.85	0.54	0.64	0.40	0.50	0.00	0.40
bulgarian-btb	verb-object	0.85	0.91	0.58	0.39	0.55	0.67	0.00	-
portuguese-bosque	verb-object	0.85	0.89	0.72	0.50	0.50	1.00	-	-
italian-isdt	verb-object	0.91	0.94	0.60	0.72	0.40	1.00	0.00	-
serbian-set	verb-object	0.92	0.95	0.81	0.84	0.29	0.67	0.00	-
ukrainian-iu	verb-object	0.89	0.92	0.69	0.72	-	0.60	0.00	-
old_church_slavonic-proiel	verb-object	0.89	0.94	0.61	0.65	0.00	0.67	0.00	0.00
ancient_greek-proiel	object-verb	0.87	0.93	0.55	0.72	0.00	0.50	0.00	0.00
hebrew-htb	verb-object	0.77	0.83	0.63	0.78	-	0.25	-	-
latin-ittb	object-verb	0.85	0.87	0.52	0.79	-	0.00	-	-
dutch-lassysmall	object-verb	0.91	0.89	0.59	0.52	-	-	-	0.00
arabic-padt	verb-object	0.88	0.76	0.61	0.36	-	0.00	-	-
japanese-gsd	object-verb	0.94	0.95	0.76	-	-	-	-	-
uyghur-udt	object-verb	0.87	0.93	0.56	-	-	-	-	-
afrikaans-afribooms	verb-object	0.86	0.90	0.52	0.00	-	-	-	-
korean-kaist	object-verb	0.79	0.84	0.52	-	0.29	-	-	-
korean-gsd	object-verb	0.83	0.88	0.49	-	0.50	-	-	-
persian-seraji	object-verb	0.85	0.86	0.80	-	0.72	-	-	-
romanian-rrt	verb-object	0.85	0.92	0.71	0.41	0.57	-	-	-
german-gsd	object-verb	0.80	0.85	0.68	0.49	0.50	0.00	0.00	-
swedish-talbanken	verb-object	0.88	0.92	0.69	0.67	1.00	0.00	0.00	-
greek-gdt	object-verb	0.87	0.92	0.82	0.52	0.80	0.33	0.00	0.00
danish-ddt	verb-object	0.80	0.85	0.68	0.37	0.80	0.33	0.00	-
norwegian-nynorsk	verb-object	0.88	0.90	0.70	0.59	1.00	0.50	-	0.00
finnish-ftb	verb-object	0.85	0.89	0.69	0.74	0.67	0.43	0.33	0.00
basque-bdt	object-verb	0.85	0.91	0.64	0.69	0.67	0.29	0.44	0.00
old_french-srcmf	verb-object	0.88	0.93	0.64	0.72	0.67	0.35	0.50	0.40
swedish-lines	verb-object	0.84	0.89	0.67	0.59	0.50	0.50	0.75	0.00
czech-fictree	verb-object	0.89	0.92	0.78	0.88	0.25	0.69	0.78	-
polish-sz	verb-object	0.90	0.94	0.76	0.87	0.00	0.67	0.89	-
slovak-snk	verb-object	0.92	0.93	0.74	0.84	-	0.73	0.67	0.00
dutch-alpino	object-verb	0.83	0.89	0.56	0.61	0.00	0.67	0.55	-
latvian-lvtb	verb-object	0.80	0.86	0.57	0.74	0.00	0.58	0.55	-
estonian-edt	verb-object	0.87	0.90	0.68	0.74	0.00	0.53	0.33	0.22
french-spoken	verb-object	0.80	0.83	0.60	0.47	0.00	0.50	0.00	0.40
gothic-proiel	verb-object	0.82	0.91	0.38	0.59	0.00	0.33	-	0.67
italian-postwita	verb-object	0.85	0.87	0.63	0.60	0.00	0.00	0.00	1.00
indonesian-gsd	verb-object	0.81	0.87	0.22	0.55	0.00	0.00	0.00	0.50
ancient_greek-perseus	object-verb	0.70	0.84	0.30	0.36	0.00	0.17	0.00	0.18
hindi-hdtb	object-verb	0.97	0.98	-	0.00	-	-	-	-
urdu-udtb	object-verb	0.94	0.97	-	-	-	-	-	-
turkish-imst	object-verb	0.81	0.86	-	-	-	-	-	-
vietnamese-vtb	verb-object	0.51	0.63	0.21	0.29	0.20	0.08	0.00	0.00
chinese-gsd	verb-object	0.61	0.72	0.35	0.34	0.22	0.00	0.33	0.00
galician-ctg	verb-object	0.63	0.70	0.31	0.47	0.12	0.00	0.00	0.00
hungarian-szeged	object-verb	0.85	0.91	0.00	0.79	-	-	-	-

Table 5: F1-Scores for Compchains Classifications for each UD 2.2 Gold Treebanks. The test section of each gold treebank was parsed using the corresponding pre-trained UDPipe language model; the compchain classification was computed for each pair of gold and parsed treebanks, and we report: (i) the weighted average F1-score (over all compchains); (ii) the (per-class) F1-score for each compchain. Entries for which the F1-score could not be computed due to a lack of support are marked with a dash ("-").