Span-Based LCFRS-2 Parsing

Miloš Stanojević
School of Informatics
University of Edinburgh
m.stanojevic@ed.ac.uk

Mark Steedman
School of Informatics
University of Edinburgh
steedman@inf.ed.ac.uk

Abstract

The earliest models for discontinuous constituency parsers used mildly context-sensitive grammars, but the fashion has changed in recent years to grammar-less transition-based parsers that use strong neural probabilistic models to greedily predict transitions.

We argue that grammar-based approaches still have something to contribute on top of what is offered by transition-based parsers. Concretely, by using a grammar formalism to restrict the space of possible trees we can use dynamic programming parsing algorithms for exact search for the most probable tree.

Previous chart-based parsers for discontinuous formalisms used probabilistically weak generative models. We instead use a span-based discriminative neural model that preserves the dynamic programming properties of the chart parsers. Our parser does not use an explicit grammar, but it does use explicit grammar formalism constraints: we generate only trees that are within the LCFRS-2 formalism. These properties allow us to construct a new parsing algorithm that runs in lower worst-case time complexity of $\mathcal{O}(l\,n^4+n^6)$, where n is the sentence length and l is the number of unique non-terminal labels. This parser is efficient in practice, provides best results among chart-based parsers, and is competitive with the best transition based parsers.

We also show that the main bottleneck for further improvement in performance is in the restriction of fan-out to degree 2. We show that well-nestedness is helpful in speeding up parsing, but lowers accuracy.

1 Introduction

Most constituency parsers are designed to predict a projective (or continuous) tree representation. This type of tree representation is not expressive enough to model (structurally) long-range dependencies that are a major concern of most syntactic theories. Take for instance the sentence in Figure 1. It contains a long range dependency between "on" and "what". This is represented differently across syntactic theories. In dependency parsing, there would be a direct arc between these two words that would cause the dependency tree to be non-projective, i.e. there would be crossed dependencies (Nivre et al., 2016). In constituency treebanks this is modelled either by using *traces* that are co-indexed with the moving element, as in English Penn treebank (Marcus et al., 1993), or by having a direct discontinuous constituent, as in German Negra and Tiger treebanks (Brants et al., 2004).

Here we adopt the discontinuous constituency approach because of its well defined formal properties, but the results are also relevant for other representations. The Penn treebank *trace* representation can be converted to a discontinuous representation (Evang and Kallmeyer, 2011)[1] and non-projective dependency trees can be interpreted as lexicalised versions of discontinuous constituency trees (Kuhlmann and Möhl, 2007).

There are two different approaches to predicting discontinuous constituency structure directly.[2] The first approach, usually grammar-based chart parsing, limits the type of trees that are acceptable (for example TAG (Joshi, 1985) or LCFRS (Vijay-Shanker et al., 1987; Seki et al., 1991)) and searches for the best tree with an exact search algorithm like CKY. The second approach, usually transition-based, does not limit the type of trees but searches for the best tree only approximately

[1]This is a lossy conversion because the discontinuous representation does not contain information about the initial location of the constituent before it was displaced, nor the attachment point in the surface tree.

[2]Indirect approaches work by conversion to some other simpler formalism, parsing in the simpler formalism, and then converting the result back. They are not the focus of this article but their results are reported in the Section 5.4.

with a beam search. Lately, with the success of neural models, transition-based parsers have been preferred to grammar-based approaches because transition-based models do not need to make any independence assumptions and strong neural models can be used to their full potential. Grammar-based methods have more difficulty incorporating rich probabilistic models due to the necessary independence assumptions needed for exact dynamic programming algorithms like CKY. Another disadvantage of grammar based models is that, even though their parsing algorithms are polynomial, they are significantly slower in practice due the high polynomial degrees and a large grammar constant.

In this work we try to improve both speed and accuracy of chart-based parsers. The accuracy is improved by using a modified version of neural span-based scoring of non-terminal nodes (Cross and Huang, 2016; Stern et al., 2017) which does not break the independence assumptions needed for efficient parsing. Speed is improved by restricting the set of acceptable trees to the ones recognizable with an LCFRS-2 grammar formalism, but no explicit grammar is used, removing the grammar constant from the worst-case complexity.[3] Additionally, the parser is implemented using an imperative approach to Viterbi CKY parsing (as opposed to deductive approach), similar to standard CFG CKY implementations with embedded loops. By avoiding the usage of standard *weighted deductive parsing* (Shieber et al., 1995; Nederhof, 2003). we avoid the need to maintain *heap property* of the agenda, further reducing the worst-case parsing complexity.

This results in a fast chart-based LCFRS-2 parser that outperforms all previous chart-based parsers for discontinuous structures, and gives performance that is on par with the best transition-based parsers.

2 LCFRS-2 Trees

LCFRS (Vijay-Shanker et al., 1987; Seki et al., 1991; Kallmeyer, 2010) is a grammar formalism that works in a similar way to CFG: it applies a series of recursive rewriting rules that eventually generate a sentence. What makes it different from CFG is that it allows each non-terminal node in the derivation tree to contain more than one continuous span of words.

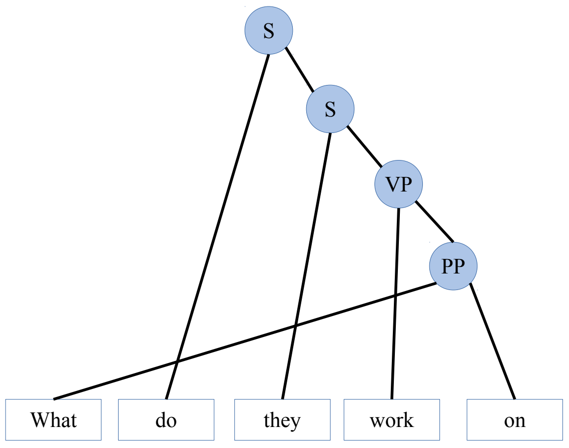

Figure 1: Discontinuous tree example.

For instance, if we look back at the example from Figure 1, we can represent the discontinuous PP as PP*(what, on)*, or in terms of spans $\mathrm{PP}\big((0,1),(4,5)\big)$. An LCFRS rule that forms this constituent can be expressed as:

$$\mathrm{PP}(X,Y) \;\rightarrow\; \mathrm{WH}(X) \;\; \mathrm{P}(Y)$$

These individual spans are often called components and the number of them per non-terminal is called the fan-out of the non-terminal. The fan-out of an LCFRS grammar is defined by the maximal fan-out of its non-terminals.

The fan-out of the grammar has significant consequences to its expressivity and the parsing complexity. For binary LCFRS the worst-case parsing complexity is $\mathcal{O}(G \cdot n^{3\phi})$ where G is the grammar constant (total number of LCFRS rules) and ϕ is the grammar's fan-out (Seki et al., 1991; Kallmeyer, 2013). If fan-out is 1 we get only the power of a standard CFG and a very efficient parser. If the fan-out is unrestricted (as big as the sentence being parsed), we could process any discontinuous structure but will get a non-polynomial parser.

Clearly, we need to restrict fan-out to some small constant number. Maier et al. (2012) suggested that restricting fan-out to 2 is sufficient to process a large portion of discontinuous structures in German treebanks. We adopt this proposal and show the consequences of it in the experiments section. We will refer to this grammar as LCFRS-2.

Another useful restriction of LCFRS is a *well-nestedness* property (Kuhlmann and Nivre, 2006; Maier and Lichte, 2009). For any LCFRS rule which contains some non-terminals A and B on its right-hand side we say that it is well-nested

[3]Note that not having explicit LCFRS-2 grammar but only explicit set of non-terminals is equivalent to having an LCFRS-2 grammar that overgenerates. This is prevented with a strong span-based probabilistic model.

if there are no components A_1 and A_2 from A, and B_1 and B_2 from B that form a linear order $A_1 < B_1 < A_2 < B_2$. This property allows for efficient parsing (Gómez-Rodríguez et al., 2010), but in our case of a binary LCFRS with fan-out 2, the effect of well-nestedness will be only proportional to some constant. Maier and Lichte (2009) state that well-nestedness holds for the majority of constituents in German treebanks. We will test it in our experiments. We will refer to this type of grammar as LCFRS-wn-2. Tree-Adjoining Grammars (TAG) (Joshi, 1985) are weakly equivalent to LCFRS-wn-2.

3 Neural Span-Based Model

We borrow and modify some ideas already popular in CFG parsing to improve LCFRS-2 chart parsing. In particular, span-based scoring is a popular approach for modelling scoring of parse trees without breaking the dynamic programming assumptions of chart parsers (Cross and Huang, 2016; Stern et al., 2017; Gaddy et al., 2018; Kitaev and Klein, 2018a,b).

In this approach words are first encoded with bi-LSTM (Hochreiter and Schmidhuber, 1997; Graves et al., 2005). These word encodings are afterwards used to score spans. For each span we take encodings of two words that are at its borders and pass them through feed-forward (Cross and Huang, 2016; Gaddy et al., 2018) or bi-affine classifier (Dozat and Manning, 2017; Stern et al., 2017) that predicts the score for each possible label (non-terminal) occupying that span. Unaries are all collapsed into a single non-terminal to simplify scoring. The score of a whole tree is defined as a sum of the scores all of its nodes. These scores are often optimised for max-margin loss (Taskar et al., 2004a) by decoding the currently best tree according to the model and minimising the margin violation in case the predicted tree is not the gold tree.

Stern et al. (2017) show that span labelling and span combination (parsing) part can be done independently for this model because the best label for each span does not depend on the span's children nodes, unlike the standard PCFG. Computing optimal labels for each span takes $\mathcal{O}(l\ n^2)$ for sentence of length n and l labels (non-terminals).

There are a couple of things that need to be addressed before this approach can be used for LCFRS-2 parsing. First is that non-terminals in LCFRS-2 can have two spans and applying the approach of Stern et al. (2017) would give labelling algorithm that runs in $\mathcal{O}(l\ n^4)$ which is prohibitively large considering the hidden constant factor of matrix multiplication done by the neural scoring layer.

To reduce the computational complexity of span scoring we introduce independence assumption that score of some discontinuous constituent with label X and spans $\big((a,b),(c,d)\big)$ is:

$$\begin{aligned}\text{score}(X\big((a,b),(c,d)\big)) = \text{score}(X_{left}(a,b))+ \\ \text{score}(X_{gap}(b{+}1,c{-}1))+ \\ \text{score}(X_{right}(c,d))\end{aligned}$$

where X_{left}, X_{gap}, X_{right} are newly created non-terminals for each X. This decomposes the discontinuous constituent scoring as scoring of three continuous constituents. The labelling complexity with l labels is still $\mathcal{O}(l\ n^4)$ but the neural matrix multiplication will be done only $\mathcal{O}(n^2)$ times just like in CFG case of Stern et al. (2017). In Section 5.3 we will show that most of the time is spent in the neural component and span combination, and that the labelling component takes a negligible proportion of time.

The second aspect of span-based models that we needed to change is the objective function. The original max-margin parsing objective proposed by Taskar et al. (2004b) maximised the margin between the gold tree and all other trees. Because that approach was too slow in practice it is usually approximated by maximising only the margin between the gold and the highest scoring tree, in case highest scoring tree is not the gold tree. This approach gave good results in CFG parsing (Stern et al., 2017), but it was very unstable in our tests. The reason for this may be in the difference between the number of possible hypotheses between CFG and LCFRS-2 which increases quadratically from the order of $\mathcal{O}(n^3)$ to $\mathcal{O}(n^6)$. In this case, optimising for just a single margin violation may be a too weak learning signal. Decreasing the scores of one bad tree alone may increase the score of another bad tree.

That is why instead of the structured max-margin training we used an alternative method where we treat each triple *(span start, span end, label)* as a binary classification task and train the model to predict the probability of that triple being part of the gold tree. For training we use not only the triples from the gold tree but all possible triples for a given sentence. We consider the probability of the tree to be the product of probabilities of the triples coming from each of its nodes. This probability model is obviously making some independence assumptions that are not correct. For instance, the probability of a constituent with a span $(1, 3)$ does not inform the probability of a constituent with a span $(2, 5)$ even though it is clear that both constituents cannot exist at the same time. This model may nevertheless give good parsing results because the optimal result of these classifications would give the optimal tree.

This method is much more stable in comparison to the max-margin approach of Stern et al. (2017) because the gradient takes into consideration the components of all possible trees at the same time instead of only the highest scoring one. In comparison to Max-Margin Markov Networks method of Taskar et al. (2004b) which also considers all trees, our approach is much faster because it does not need to build chart for each training instance.

As mentioned before, we collapse all unary chains into a single non-terminal which contains sufficient information to be unchained after parsing. Nodes that have more than 2 children are binarized with the same method as Stern et al. (2017) by labelling all new sub-nodes as $\varnothing$. Again, there are some aspects to consider before applying the method of Stern et al. (2017). First, binarization of LCFRS, unlike binarization of CFG, can increase the generative power by increasing the fan-out (Kallmeyer, 2010). If we have a tree that can be generated with LCFRS-2 and arbitrarily choose binarization method, the binarised tree may turn out not to be within the strong generative power of LCFRS-2. Hence, choosing the right binarization is important. Second, different binarizations actually correspond to different latent derivations of the tree we are modelling. These latent derivations will have different probabilities and its not easy to see which one of them should be used. The approach we will pursue is to model all of them by treating every possible triple that can be extracted from every possible binarization of a gold tree to be a positive class.

4 Direct CKY Parsing Algorithm

The algorithms for LCFRS are usually presented, and implemented, as deductive rules. These deductive rules, combined with a deduction engine of Shieber et al. (1995) can form a conceptually simple mechanism for parsing. In case of *weighted deductive rules* the modification of Nederhof (2003) can be used. It modifies the method of search to explore the most probable search space first by implementing the *agenda* as a priority queue. Almost all Probabilistic LCFRS (PLCFRS) parsers have been implemented in this way (Kallmeyer and Maier, 2010; Maier et al., 2012; van Cranenburgh et al., 2016).

However, there are many reasons not to use this approach with our span-based model. First, implementing the agenda as a priority queue adds a $\mathcal{O}(\log n)$ multiplicative term to the worst-case complexity. Second, the multiplicative grammar constant that exists in PLCFRS approaches does not exist in ours since there is no explicit grammar, and the optimal label for each span is independent of the other spans. Third, because of the difficulty of implementing optimal chart lookup under deductive approaches means that most PLCFRS parsers optimise lookup only on the non-terminal labels and not on span indices, representing a serious bottleneck.

The parsing approach we propose has instead worst-case complexity $\mathcal{O}(l\ n^4 + n^6)$ because it does not use an explicit grammar, nor priority queue, and it has very straightforward lookup based on indices. It consists of two parts. First part takes the scores from the neural model and computes the optimal score for each possible LCFRS-2 combination of spans of which there are n^4. That makes its complexity $\mathcal{O}(l\ n^4)$ where l is the number of distinct non-terminals. The second part does the actual parsing by combining these scores to form the best tree. It is a generalisation of how non-deductive CFG CKY algorithm works by having multiple embedded *for* loops and a multi-dimensional array to represent a chart. Both chart and loops have to be adapted to LCFRS.

We have two data structures involved that are both indexed by the span: a lookup table for optimal span label (and its score), and a lookup table for the optimal backpointer to children nodes (and its score). We will refer to the first one as $\mathrm{labChart}$ and to the second one as chart. Each one of them could be used for looking up continuous spans (only 2 indices) or discontinuous spans (4 indices). We can implement them with multi-dimensional arrays that provide constant lookup.

To find which loops are needed we borrow Table 1 from Maier et al. (2012) who have found all possible rule shapes for binary LCFRS-2. We augment this table with the worst-case complexity for each rule given in the fourth column. This complexity can be easily derived using the method of McAllester (2002) which states that the computational complexity of each rule depends on the number of free variables on the left-hand side of the rule, assuming rules are non-deleting. For instance, for CFG rule #3 there are three free variables: index at the start of X, index between X and Y and index at the end of Y. Therefore its complexity is $\mathcal{O}(n^3)$. Each of these indices requires an embedded *for* loop.

ID	Type	counts	$\mathcal{O}(.)$
#1	A(X) → B(X)	49	$\mathcal{O}(n^2)$
#2	A(X, Y) → B(X, Y)	1	$\mathcal{O}(n^4)$
#3	A(XY) → B(X) C(Y)	14,430	$\mathcal{O}(n^3)$
#4	A(X,Y) → B(X) C(Y)	1,644	$\mathcal{O}(n^4)$
#5	A(XYZ) → B(X,Z) C(Y)	621	$\mathcal{O}(n^4)$
#6	A(X,YZ) → B(X,Y) C(Z)	100	$\mathcal{O}(n^5)$
#7	A(X,YZ) → B(X,Z) C(Y)	142	$\mathcal{O}(n^5)$
#8	A(XY,Z) → B(X,Z) C(Y)	172	$\mathcal{O}(n^5)$
#9	A(XY,Z) → B(X) C(Y,Z)	582	$\mathcal{O}(n^5)$
#10	A(XY,ZU) → B(X,Z) C(Y,U)	7	$\mathcal{O}(n^6)$
#11	A(XY,ZU) → B(X,U) C(Y,Z)	0	$\mathcal{O}(n^6)$
#12	A(X,YZU) → B(X,Z) C(Y,U)	12	$\mathcal{O}(n^6)$
#13	A(XYZ,U) → B(X,Z) C(Y,U)	12	$\mathcal{O}(n^6)$
#14	A(XYZU) → B(X,Z) C(Y,U)	13	$\mathcal{O}(n^5)$

Table 1: LCFRS-2 rule instances from (Maier et al., 2012) with frequency counts from Tiger sentences of length 30.

This is simple when we have only one rule shape as in the case of CFG, but with LCFRS we need to make sure that all rules are tested in the right order. We know that bigger spans are always composed of smaller spans. Therefore we can have a top *for* loop that would iterate over the total span size. The *for* loop below it would split that total span size between left and right span in case of rules that produce discontinuous constituents. These top loops are needed only to ensure that constituents are built in a bottom-up topological order. Further loops are used only to compute all other needed indices for each rule. The space in this paper is not sufficient to present the implementation for all 14 rules but the example in Algorithm 1 for rule #6 should be sufficient to show how the rest of the algorithm works. The number of embedded *for* loops for this rule clearly corresponds to its computational complexity.

By designing which rules from this schema we use we can get different generative power accompanied with a different computational complexity.

Algorithm 1 Direct CKY LCFRS-2 Parsing

```
for sizeSpan in $[1 \dots n]$ do
    ...
    for $a$ in $[0 \dots n - sizeSpan - 1]$ do
        $b \leftarrow a + sizeSpan$
        // processing a continuous constituent
        // with the span (a, b)
        // using rules 3, 5 and 14
    ...
    for sizeLeft in $[1 \dots \text{sizeSpan} - 1]$ do
        sizeRight ← sizeSpan − sizeLeft
        ...
        for $a$ in $[0 \dots n - sizeSpan - 1]$ do
            $b \leftarrow a + \text{sizeLeft}$
            for $c$ in $[b + 1 \dots n - \text{sizeRight}]$ do
                $d \leftarrow c + \text{sizeRight}$
                best ← $-\infty$
                // processing a discontinuous
                // constituent with two components
                // (a, b) and (c, d)
                // finding best solution with
                // discontinuous rules 2, 4, 6,
                // 7,8,9,10,11,12,13
                ...
                // rule 6: A(X, YZ) → B(X,Y)C(Z)
                // X=(a, b), Y=(c, e), Z=(e, d)
                for $e$ in $[c + 1 \dots d - 1]$ do
                    score ← chart$[a, b, c, e]$+
                            chart$[e, d]$
                    if score > best then
                        best ← score
                ...
                chart$[a, b, c, d]$ ← best +
                            labChart$[a, b, c, d]$
```

If we use only rule #3 we get a CFG parser that can be run in $\mathcal{O}(n^3)$. If we use all of the rules we get LCFRS-2 parser with complexity $\mathcal{O}(n^6)$. However, there are interesting subsets of rules in between full LCFRS-2 and CFG. Well-nested LCFRS-2 is one of those subsets. It includes all LCFRS-2 rules except #10, #12, #13 and #14. Well-nested LCFRS-2 still has the same complexity as a full LCFRS-2 because it contains rule #11 that is $\mathcal{O}(n^6)$. If we look at its counts in the Negra treebank we can see that that rule never appears. Therefore we find it also interesting to try well-nested LCFRS-2 without the rare rule. We will refer to it as LCFRS-wn-nr-2. LCFRS-wn-nr-2 can be parsed in $\mathcal{O}(n^5)$. We will not use rule types #1 and #2 in any of the approaches because we handle unary rules in a different way as previously described.

5 Experiments

The parser is implemented in Scala using DyNet (Neubig et al., 2017) and is available on github.[4] Experiments are conducted on German and English discontinuous constituency treebanks. The reported development results are on the German Negra treebank. The test set results, in addition to German Negra, also contain German Tiger treebank (Brants et al., 2004) and English Discontinuous Penn Treebank (DPTB) (Evang and Kallmeyer, 2011). The treebanks were preprocessed using standard practice described in (Maier, 2015) by using the *treetools*[5] package. For evaluation we have used the *discodop*[6] package with the standard settings (van Cranenburgh et al., 2016).

parameter	value
word-embeddings dim.	32
char bi-lstm dim.	100
sentence-level bi-lstm layers	2
sentence-level bi-lstm dim.	200
compression MLP layers	2
compression MLP dim.	200
Adam optimiser lr.	0.001
batch size	1 sentence

Table 2: Hyper-parameters.

The architecture and hyper-parameters of the neural model are chosen to be the same as in (Coavoux and Cohen, 2019) to obtain a relatively fair comparison. That is, we use a combination of character bi-LSTM to embed each word. This embedding is concatenated with the lookup table embedding for each word and passed through a two-layer bi-LSTM that runs over the whole sentence. In case of MLP model we score labels for each span by passing two bi-LSTM vectors at borders of the spans through a two-layer MLP. In the case of the bi-affine model we compress bi-LSTM vectors with a specialised MLP for left and right index, analogous to the specialisation for head and dependent vector in Dozat and Manning (2017), and then score labels through a bi-affine layer.

5.1 What is the right objective function and classification layer?

First we test if our new objective function that *locally* optimises span labelling is better than the *max-margin* approach of Stern et al. (2017) in the context of discontinuous parsing. Table 3 shows the results in which we can see that *local* model gives much better results. This is especially true for the version of the model that as its top layer uses MLP which completely fails when trained with max-margin but gives reasonable results when trained with more stable objective that takes into consideration all possible trees.

parser		all F1	disc F1
MLP	*margin*	20.75	0.00
bi-affine	*margin*	77.85	32.85
MLP	*local*	82.49	38.62
bi-affine	*local*	**84.16**	**48.83**

Table 3: Comparison on Negra dev set of different objectives and label classifiers.

Therefore in further experiments we are going to use only the bi-affine version of the model trained with the *local* objective.

5.2 Is restriction to LCFRS-2 a good approach?

A particularly interesting point of reference is the work of Coavoux and Cohen (2019) which also uses span-based scoring, but in transition-based setting. Our model can be seen as a dynamic programming alternative to their parser.

Dynamic programming (i.e. chart parsing) provides us with an exact search mechanism, unlike the approximate greedy search used by Coavoux and Cohen (2019). However, that benefit does not come for free. The development set results shown in Table 4 show that in a comparable setting (same hyper-parameters of the encoder) there are aspects in which each of the chart-based and transition-based approaches has an advantage. Why is that?

One explanation could be that the setting in which two parsers are tested is not *fully* comparable. What we mean by that is that there are algorithmic reasons why the neural architecture cannot be exactly the same. Let us take a constituent with a gap as an example where the left component is a span (a, b) and the right component is (c, d). Coavoux and Cohen (2019) predict the probability of the next transition by encoding the gap constituent with all 4 embeddings together as (a, b, c, d). In our case we had to split the decision on the label into three independent decisions: the first one that takes (a, b), the second one for (b, c) and the final one for (c, d). This independence as-

[4]`https://github.com/stanojevic/grammarless-lcfrs`

[5]`https://github.com/wmaier/treetools`

[6]`https://github.com/andreasvc/disco-dop`

parser	all			disc		
	F1	P	R	F1	P	R
Coavoux and Cohen (2019)	84.00	—	—	**54.00**	—	—
CFG	82.86	84.50	81.29	—	—	—
LCFRS-wn-nr-2	83.99	85.21	82.80	45.04	66.24	34.12
LCFRS-wn-2	83.99	85.21	82.80	45.04	66.24	34.12
LCFRS-2	**84.16**	**85.34**	**83.02**	48.83	**67.21**	**38.34**

Table 4: Development set results on Negra treebank. The column *all* contains the results computed over all constituents, both continuous and discontinuous, while *disc* are results on the discontinuous constituents alone.

sumption is necessary because otherwise we would need to run the MLP layer $\mathcal{O}(n^4)$ times. This is not an issue for Coavoux and Cohen (2019) because they consider only a subset of spans needed in greedy search.

However, we think that the main property that distinguishes these two models is expressive power, i.e. the set of trees that they can generate. While the transition-based parser could generate any discontinuous tree, our chart-based parser can generate only trees that are within the LCFRS-2 formalism. To find evidence for the importance of this property we modified the search to explore the different levels of complexity in between CFG and LCFRS-2 while keeping the exactly same parameters of the neural scoring model. From Table 4 we can see that the higher we get on the complexity hierarchy the better are results on the development set, both for discontinuous constituents and all constituents. In comparison to Coavoux and Cohen (2019), we get better results overall but for discontinuous constituents alone the transition-based parser still has an edge.

parser with ideal scorer	all		disc	
	F1	R	F1	R
CFG	97.53	95.19	—	—
LCFRS-wn-nr-2	99.29	98.60	82.95	70.88
LCFRS-wn-2	99.35	98.71	83.04	71.01
LCFRS-2	99.69	99.39	93.32	87.48

Table 5: Oracle parsing results with an *ideal* scorer that always assigns correct probabilities.

If we look at the results for discontinuous constituents carefully we can see that precision is significantly greater than recall. The reason for this could be that the parser is good when the gold tree is within the reach of the LCFRS-2 formalism, but for discontinuous constituents that is sometimes not the case.

To test the limitations that the formalism puts on our model further we did oracle experiments that would show what would results be if we had an *ideal* scoring model that always gives perfect probability 1 to correct span labellings and probability 0 to incorrect ones. The results for the oracle experiments are shown in Table 5. If we compare results over F1 of *all* types of constituents then there is very little difference among the discontinuous formalisms. However, if we evaluate only on the discontinuous constituents, the change in coverage (recall) when we remove the well-nestedness constraint of LCFRS-wn-2 to LCFRS-2 is very large, around 16%.

The recall of 87% for our most expressive formalism LCFRS-2, seem to suggest that if we want further increases in accuracy of chart-based discontinuous constituency parsers we will need more than LCFRS-2 generative power. Furthermore, this more expressive formalism will need to be able to generate trees that are not well-nested.

This is not to be confused with requirements for well-nestedness of *dependencies*. The need for ill-nested dependencies was established in the work of Chen-Main and Joshi (2010). However, grammar formalisms like CCG can model ill-nested dependencies without having ill-nested derivations (Koller and Kuhlmann, 2009). Our statement about the need of increasing fan-out and for allowing ill-nested rules applies only to the prediction of discontinuous constituency structures of the kind found in the Negra treebank.

5.3 Parsing speed

Chart parsers have often been avoided for expressive formalisms like LCFRS because of their high worst-case complexity. Most previous work using them has either constrained sentences to those less than 30 words in length, or used length filtering in combination with heavy pruning (Evang and Kallmeyer, 2011; van Cranenburgh and Bod, 2013; van Cranenburgh et al., 2016; Ruprecht and Denkinger, 2019) it is therefore important to compare our parser with previous approaches not only in accuracy but also in speed.

	parser		DPTB		Tiger		Negra	
			all	disc	all	disc	all	disc
LCFRS	**this work**		90.5	67.1	83.4	53.5	83.6	50.7
	Evang and Kallmeyer (2011)	$\leq$ 25 gold POS	79.0		81.6			
	van Cranenburgh et al. (2016)	$\leq$ 40	87.0				74.8	
	Ruprecht and Denkinger (2019)	$\leq$ 30 gold POS					69.0	
transition-based	Coavoux and Cohen (2019)		90.9	67.3	82.5	55.9	83.2	56.3
	Coavoux et al. (2019)		**91.0**	**71.3**	82.7	55.9	83.2	54.6
	Stanojević and G. Alhama (2017)				77.0			
	Stanojević and G. Alhama (2017)	gold POS			81.6		82.9	
	Maier (2015)	gold POS			74.7	18.8	77.0	19.8
	Coavoux and Crabbé (2017)	gold POS			81.6	49.2	82.2	50.0
conversion	Corro et al. (2017)		89.2					
	Corro et al. (2017)	gold POS	90.1		81.6			
	Versley (2016)				79.5			
	Fernández-González and Martins (2015)				77.3			
	Fernández-González and Gómez-Rodríguez (2020)				**85.3**	**59.1**	**85.4**	**58.8**

Table 6: Test set results.

In theory our parser is certainly an improvement because it runs in $\mathcal{O}(l\ n^4 + n^6)$ while other parsers in the worst-case use $\mathcal{O}(G\ n^6 \log n)$. To test if the same holds in practice we ran the parser on Negra dev set sentences of different length without using any pruning techniques. The results up to length 50 are shown in Figure 2.

The neural component (mostly bi-LSTM) and labelling component (with complexity $\mathcal{O}(l\ n^4)$) are shared across all parsing approaches we have tried. The labelling component, despite its theoretical complexity, has a very small influence on the overall parsing speed even for long sentences.

Stern et al. (2017) state that in their experiments the neural component took most of the time. While in our experience that is true for CFG search, the same conclusion does not hold for LCFRS for sentences longer than 35 words.

Instead, parsing time for long sentences is dominated by the chart parsing component. The well-nested version that excludes the rare rule (LCFRS-wn-nr) is the fastest, as predicted by its complexity of $\mathcal{O}(n^5)$. As we have seen in previous sections, excluding the rare rule #11 does not affect outcome, but it does affect speed significantly.

The more powerful, and accurate, formalism of full LCFRS-2 changes the dynamics of parsing: speed quickly decreases for sentences longer than 30 words. Nevertheless, parsing time stays under 1 second for all sentences under 45 words without any need for pruning. This is significantly faster than speeds reported for all previous chart-based parsers that do not use pruning (see *ddlcfrs*, *rparse* and *GF* in Figure 4 in Ruprecht and Denkinger (2019)). The only parsers that could compare in speed are heavily pruned versions of *DiscoDOP* (van Cranenburgh and Bod, 2013) and *OP* (Ruprecht and Denkinger, 2019) that get much lower accuracy than our parser (see Table 6).

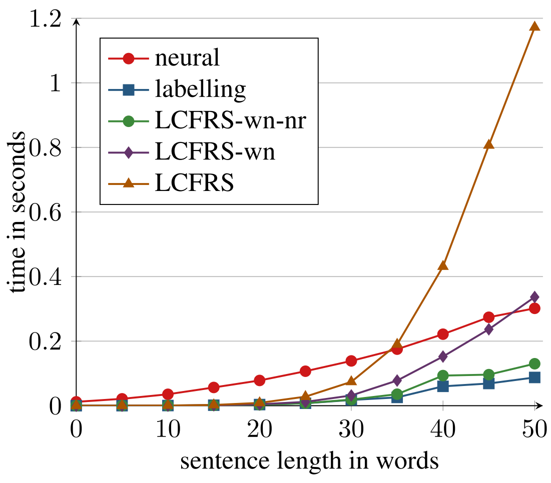

Figure 2: Parsing speed on Negra.

For sentences longer than 50 words (not visible on the plot) parsing is significantly slower, but it is still tractable. For our test set results we use no pruning up to sentence length 60: for the rare sentences above 60, we use the same model, but with only well-nested parse search.

5.4 Test set results

Test set results for English and German are shown in Table 6. Compared to previous chart-based LCFRS parsers our parser provides the best results on all measures for both English and German.

Compared to transition based parsers, it is competitive over all constituencies, but has slightly lower score on discontinuous constituents alone.

The recent parser by Fernández-González and Gómez-Rodríguez (2020) outperforms both LCFRS and transition-based parsers. It treats discontinuous constituency parsing as a diconstinuous dependency parsing with slightly enriched labels that allow conversion back to the discontinuous constituency structure. However, it is not easy to see how to compare this approach to the ones discussed above.

6 Conclusion

We have presented a span-based LCFRS-2 parser that outperforms all previous LCFRS parsers. It is in addition competitive with the best transition-based parsers, outperforming them in all-constituency evaluation for both German treebanks.

The results from this paper also indicate that the strong generative power of the grammar formalism is correlated with the accuracy. LCFRS-2 power is a great improvement over formalisms that are lower in the complexity hierarchy, but is still inadequate for complete coverage of discontinuity. Our results also show that well-nestedness significantly limiting the coverage that could be achieved even with an ideal scoring model.

Acknowledgments

We thank Maximin Coavoux for providing us access to DPTB dataset. This work was supported by ERC H2020 Advanced Fellowship GA 742137 SEMANTAX grant.

References

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. TIGER: Linguistic Interpretation of a German Corpus. *Research on Language and Computation*, 2(4):597–620.

Joan Chen-Main and Aravind K. Joshi. 2010. Unavoidable ill-nestedness in natural language and the adequacy of tree local-MCTAG induced dependency structures. In *Proceedings of the 10th International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+10)*, pages 53–60, Yale University. Linguistic Department, Yale University.

Maximin Coavoux and Shay B Cohen. 2019. Discontinuous constituency parsing with a stack-free transition system and a dynamic oracle. In *Proceedings of NAACL-HLT*, pages 204–217.

Maximin Coavoux and Benoît Crabbé. 2017. Incremental discontinuous phrase structure parsing with the GAP transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270, Valencia, Spain. Association for Computational Linguistics.

Maximin Coavoux, Benoît Crabbé, and Shay B Cohen. 2019. Unlexicalized transition-based discontinuous constituency parsing. *Transactions of the Association for Computational Linguistics*, 7:73–89.

Caio Corro, Joseph Le Roux, and Mathieu Lacroix. 2017. Efficient discontinuous phrase-structure parsing via the generalized maximum spanning arborescence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1644–1654, Copenhagen, Denmark. Association for Computational Linguistics.

Andreas van Cranenburgh and Rens Bod. 2013. Discontinuous parsing with an efficient and accurate dop model. In *Proceedings of The 13th International Conference on Parsing Technologies (IWPT 2013)*, pages 7–16.

Andreas van Cranenburgh, Remko Scha, and Rens Bod. 2016. Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57–111.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Kilian Evang and Laura Kallmeyer. 2011. PLCFRS Parsing of English Discontinuous Constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, IWPT '11, pages 104–116, Stroudsburg, PA, USA. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing with pointer networks. *arXiv preprint arXiv:2002.01824*.

Daniel Fernández-González and André F. T. Martins. 2015. Parsing as reduction. 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 1523–1533, Beijing, China. Association for 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.

Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 276–284. Association for Computational Linguistics.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2005. Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. In *Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - Volume Part II*, ICANN'05, pages 799–804, Berlin, Heidelberg. Springer-Verlag.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8).

Aravind K. Joshi. 1985. *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?*, Studies in Natural Language Processing, page 206–250. Cambridge University Press.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Springer, NY.

Laura Kallmeyer. 2013. Linear context-free rewriting systems. *Language and Linguistics Compass*, 7(1):22–38.

Laura Kallmeyer and Wolfgang Maier. 2010. Data-driven parsing with probabilistic linear context-free rewriting systems. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 537–545. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018a. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686.

Nikita Kitaev and Dan Klein. 2018b. Multilingual constituency parsing with self-attention and pre-training. *arXiv preprint arXiv:1812.11760*.

Alexander Koller and Marco Kuhlmann. 2009. Dependency Trees and the Strong Generative Capacity of CCG. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 460–468, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marco Kuhlmann and Mathias Möhl. 2007. Mildly context-sensitive dependency languages. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 160–167, Prague, Czech Republic. Association for Computational Linguistics.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly Non-projective Dependency Structures. In *Proceedings of the COLING/ACL on Main Conference Poster Sessions*, COLING-ACL '06, pages 507–514, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wolfgang Maier. 2015. Discontinuous Incremental Shift-reduce 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)*, pages 1202–1212, Beijing, China. Association for Computational Linguistics.

Wolfgang Maier, Miriam Kaeshammer, and Laura Kallmeyer. 2012. Data-driven PLCFRS parsing revisited: Restricting the fan-out to two. In *Proceedings of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, Paris, France.

Wolfgang Maier and Timm Lichte. 2009. Characterizing discontinuity in constituent treebanks. In *International Conference on Formal Grammar*, pages 167–182. Springer.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

David McAllester. 2002. On the complexity analysis of static analyses. *J. ACM*, 49(4):512–537.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

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. *arXiv preprint arXiv:1701.03980*.

Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666.

Thomas Ruprecht and Tobias Denkinger. 2019. Implementation of a chomsky-schützenberger n-best parser for weighted multiple context-free grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 178–191.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. 1995. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36.

Miloš Stanojević and Raquel G. Alhama. 2017. Neural discontinuous constituency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 818–827.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2004a. Max-margin markov networks. In *Advances in neural information processing systems*, pages 25–32.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004b. Max-margin parsing. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 1–8.

Yannick Versley. 2016. Discontinuity re^2-visited: A minimalist approach to pseudoprojective constituent parsing. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 58–69, San Diego, California. Association for Computational Linguistics.

Krishnamurti Vijay-Shanker, David J Weir, and Aravind K Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th annual meeting on Association for Computational Linguistics*, pages 104–111. Association for Computational Linguistics.