# Improving the Extraction of Supertags for Constituency Parsing with Linear Context-Free Rewriting Systems

**Thomas Ruprecht**
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
thomas.ruprecht@tu-dresden.de

## Abstract

In parsing phrase structures, supertagging achieves a symbiosis between the interpretability of formal grammars and the accuracy and speed of more recent neural models. The approach was only recently transferred to parsing discontinuous constituency structures with linear context-free rewriting systems (LCFRS). We reformulate and parameterize the previously fixed extraction process for LCFRS supertags with the aim to improve the overall parsing quality. These parameters are set in the context of several steps in the extraction process and are used to control the granularity of extracted grammar rules as well as the association of lexical symbols with each supertag. We evaluate the influence of the parameters on the sets of extracted supertags and the parsing quality using three treebanks in the English and German language, and we compare the best-performing configurations to recent state-of-the-art parsers in the area. Our results show that some of our configurations and the slightly modified parsing process improve the quality and speed of parsing with our supertags over the previous approach. Moreover, we achieve parsing scores that either surpass or are among the state-of-the-art in discontinuous constituent parsing.

## 1 Introduction

Discontinuous constituency parsing deals with the task to find hierarchies of – possibly non-contiguous – phrases (*constituents*) in a given sentence and assigns a label (*constituent symbol*) to each phrase. Traditional approaches use grammar formalisms such as linear context-free rewriting systems (LCFRS) to model these hierarchies (Maier and Søgaard, 2008; Kallmeyer and Maier, 2013; van Cranenburgh et al., 2016; Gebhardt, 2020). Statistical parsing with these grammars is remarkably slow and inaccurate by today's standards. But they still find some attraction as both, the grammars and parsing with them, are easily interpretable. More recent parsers use neural classifiers and either leverage the parsing process into a linear task (Coavoux, 2021; Fernández-González and Gómez-Rodríguez, 2021b,a, 2022) or score constituent labels for selected phrases (Corro, 2020; Stanojević and Steedman, 2020). Although, in the latter approaches, the occurring discontinuities are restricted to a small degree.

In supertagging-based parsing (Bangalore and Joshi, 1999), a grammar is accompanied by a classifier that selects and scores a small sample of rules. After that, these rules and their scores are interpreted as a weighted grammar and used for statistical parsing in the usual manner. This remaining statistical parsing process is significantly faster than the traditional approach, as the grammar is much smaller. The approach was investigated in combination with tree adjoining grammars (TAG; Kasai et al., 2017; Bladier et al., 2018) and combinatory categorial grammars (CCG; Clark, 2002; Kadari et al., 2018), often in the context of dependency parsing. Since the introduction of recurrent neural networks (RNN) as classifiers, the accuracy of the supertag prediction has improved by far (Vaswani et al., 2016; Kasai et al., 2017; Bladier et al., 2018; Kadari et al., 2018). A recent publication (Ruprecht and Mörbitz, 2021) showed that supertagging improves the quality and speed of parsing constituency structures with LCFRS, bringing it close to recent discontinuous parsing methods. However, their extraction process for supertags is rather convoluted and uses hard-wired strategies for, e.g., the lexicalization and binarization. In this work, we investigate if the quality of predictions and parsing with LCFRS supertags can be improved by introducing parameters that replace these hard-wired configurations in the extraction.

Section 3 presents a formulation of supertags and an extraction algorithm that is – in our eyes – easier to grasp than the previous definition, be-



*Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1466 - 1477

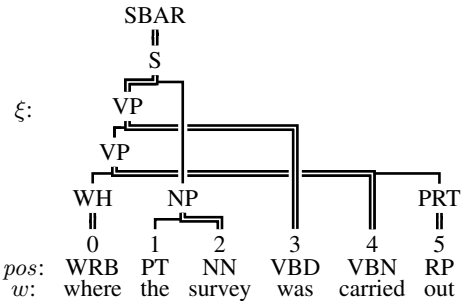



Figure 1: Discontinuous constituent tree for the phrase *where the survey was carried out*. The tree is illustrated with crossing branches, so that the leaves appear ordered. For each constituent, the path to its lexical head is double-struck.

cause it avoids cumbersome transformations of LCFRS derivations. Both tackle the following limitations of the existing approach: (i) Before the extraction of grammar rules, the constituent trees were binarized using specific fixed parameters. We will investigate the impact of varying those parameters to the processes for extraction and parsing. (ii) Constructing lexical LCFRS rules picked a sentence position for each inner node of the constituent tree according to a fixed strategy. We will investigate multiple such strategies, which we will call *guide constructors*. (iii) LCFRS rules were constructed with constituent symbols as nonterminals, which were then supplemented with annotations during the lexicalization process. The authors noted that the sets of extracted supertags are rather large compared to other approaches, and we deem that the granularity of nonterminals plays a significant role in this issue. We decouple the nonterminals from the other extraction processes and introduce multiple strategies to define them, called *nonterminal constructors*. Section 4 describes experiments with the discontinuous English Penn Treebank (DPTB, Marcus et al., 1994; Evang and Kallmeyer, 2011), and the two German treebanks NeGra (Skut et al., 1998) and Tiger (Brants et al., 2004). It explains how we found viable configurations for the introduced parameters and gives results for parsing with them. The implementation is available as free software.[1]

## 2 Notation

A *discontinuous constituent tree* is a tuple $(\xi, pos, w)$ as follows: $w$ is a sequence of terminal symbols (*phrase*), $pos$ is a sequence of part-of-speech (*pos*) symbols with the same length as $w$, and the *constituent structure* $\xi$ is a tree; its inner nodes are *constituent symbols* and its *leaves* are positions $0, \ldots, |w| - 1$ such that each occurs exactly once in $\xi$. Figure 1 shows an example.

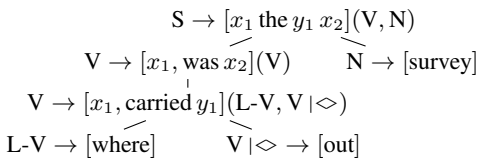



Figure 2: A binary lexical LCFRS derivation for the string *where the study was carried out*.

We use the usual notation for *(Gorn-)positions*[2] in the constituent structure, i.e. each position determines exactly one node in $\xi$. The *set of all inner node positions* in $\xi$ is denoted by $\mathrm{npos}(\xi)$. The *subtree of $\xi$ at position $\rho$* is denoted by $\xi|_\rho$. The *yield* $\mathrm{yd}(\xi)$ is the set of leaves in $\xi$. The *fanout of (a set of leaves) $L$* is the smallest number of contiguous subsets of $L$. For instance, in Fig. 1, the yield of the subtree governed by the upper node labeled by VP is the set $\{\underline{0}, \underline{3}, \underline{4}, \underline{5}\}$, its fanout is 2. $\xi(\rho)$ denotes the constituent symbol at position $\rho$.

We sometimes consider *lexical heads* for the constituent structure. For each inner node, the lexical head is the critical sentence position that determines its symbol; and we call each of the node's children that does not contain the lexical head a *modifier*. In Fig. 1, for each inner node in $\xi$, the path to its lexical head is double struck. E.g. for the inner node with symbol S, the lexical head is the position 3. The subtree starting at the NP node is its only modifier.

We briefly cover the notation for *binary lexical LCFRS* (in the following just *LCFRS*). Each rule is either of the form $A \to [w]$ (*nullary*), $A \to [u_1, \ldots, u_k](B_1)$ (*unary*) or $A \to [u_1, \ldots, u_k](B_1, B_2)$ (*binary*). $A$ is the left-hand side (*lhs*) nonterminal, $B_1, B_2$ are right-hand side (*rhs*) nonterminals, and $w$ is the rule's lexical symbol (or terminal). The non-empty strings $u_1, \ldots, u_k$ consist of exactly one lexical symbol and variables $x_1, \ldots, x_n$ (and $y_1, \ldots, y_m$ in binary rules). They denote a function composing $k$

[1] https://github.com/truprecht/lcfrs-supertagger

[2] Each position is a sequence of integers, where, starting with the root in the tree and from left to right in the sequence, each $n$ in the sequence recursively addresses the $n$-th child of the current node. We expect that the children of each node are ordered according to their leftmost leaf. E.g. $\varepsilon$ is the root node's, and 1.2 the NP node's position in Fig. 1.

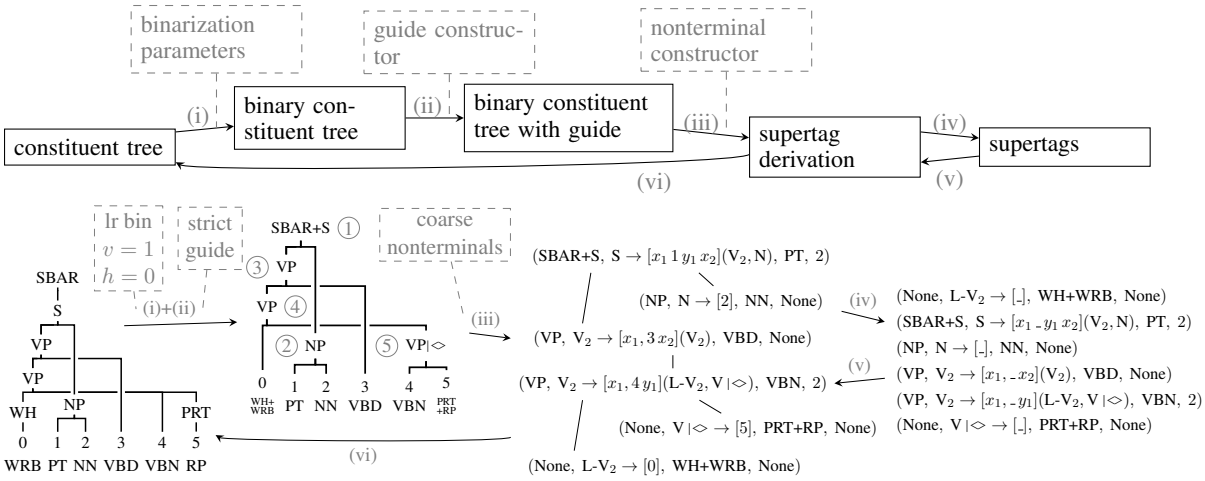



Figure 3: (Top) Visualization of the extraction (left to right) and parsing (right to left) process. (Bottom) Examples for the steps shown in the top. The results after steps (i) and (ii) are shown in tandem: gray boxes next to the inner nodes show the leaves assigned by the guide. Dashed gray boxes show parameters involved in the extraction.

strings from the lexical symbol and $n$ (respectively $n + m$ in the binary case) argument strings.

An LCFRS derivation is a tree where each node is a rule such that its number of successors matches the rule's arity and its rhs nonterminals match the successors' lhs nonterminals; Fig. 2 shows an example. The strings produced by a derivation are obtained recursively for each node from bottom to the top as follows:

- If the node is a nullary rule $A \to [w]$ and has no successors, then the produced string is $w$.
- If the node is either a unary or binary rule of the form $A \to [u_1, \ldots, u_k](\vec{B})$ with $|\vec{B}|$ successors, then it produces $k$ strings which are obtained from $u_1, \ldots, u_k$ by replacing each variable $x_i$ with the $i$-th string produced by the first and $y_j$ with the $j$-th string produced by the second successor.

## 3 Contributions

This section presents all concepts involved with the processes for the extraction from (discontinuous) constituency treebanks and parsing with LCFRS supertags. We will start in Section 3.1 with a short motivation for our notation for supertags, which deviates from the previous formulation. Section 3.2 describes our process for the extraction of supertags and highlights parts that require sets of parameters. These parameters are described in detail in the following Section 3.3. Section 3.4 concludes with an overview for the parsing process. All the described steps and concepts are illustrated along the arrows in Fig. 3, their labeling conforms with the numbering in the paragraph headings of Sections 3.2 and 3.4.

### 3.1 Supertags

Ruprecht and Mörbitz (2021) introduced LCFRS supertags as rules with certain annotations that do not fit into the usual framework but are necessary to convert derivations into constituent trees. Moreover, their notation is closely tied to the extraction and parsing pipeline, which, as explained in Section 1, assumes some fixed choices that we establish as hyperparameters. To tackle these limitations, we introduce the following notation for supertags: a *supertag* is a tuple $(r, t, c, p)$ where

- $r$ is an LCFRS rule, its terminal is a wildcard,
- $t$ is *None* or an index in $\{1, 2\}$ tracking the transformations for the extraction of lexical rules,[3]
- $c$ is either *None* or a constituent symbol[4], and
- $p$ is a pos symbol.[5]

### 3.2 Extraction Process

The process described in this section is used to extract a sequence of supertags, one for each word in a sentence, from a given constituent tree. We distinguish four steps (i–iv) which are executed consecutively. The parameters for steps (i–iii) are described in detail in the next section.

[3] $t$ targets the limitation ii in Section 1 and allows us to remove the annotation *swap* that was introduced by Ruprecht and Mörbitz (2021) in step 5 of their lexicalization procedure. This annotation does not fit into the LCFRS framework, but is considered an integral part of the extracted supertag.

[4] The symbol $c$ targets the limitation iii. It decouples the induction of nonterminals from constituent symbols.

[5] Pos symbols are usually handled separately from the constituent structure in most recent parsing approaches, including the existing LCFRS supertag parser.

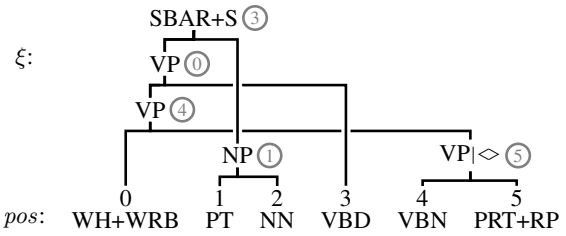



Figure 4: Constituent structure and pos symbols after binarization ($v = 1$, $h = 0$; ho and lr binarization coincide) of Fig. 1. The symbol "VP|<>" was introduced during binarization; former unary nodes were joined by "+". Gray integers next to inner nodes show the leafs assigned by a guide for the constituent structure.

**(i) Binarization.** We construct a binary constituent tree with the usual strategies in constituent parsing (Kallmeyer and Maier, 2013): Each unary node is merged with its child (or pos symbol, if the child is a leaf), and nodes with arity $n > 2$ are split into $n - 1$ binary nodes according to the parameters described in Section 3.3. After this step, the constituent tree for a sentence $w$ is equipped with $|w| - 1$ inner nodes. Figure 4 shows a binary tree resulting from binarization of the tree in Fig. 1.

**(ii) Guide.** In this step, we define a *guide* for the binary constituent structure $\xi$, i.e. a mapping $\mathtt{G}$ between inner node positions and leaves in the constituent structure. In the following step, a lexical LCFRS rule will be constructed for the constituent at each inner node and the assigned leaf. Intuitively, the guide determines which sentence position is "responsible" for the constituent at each position. Formally, a guide for $\xi$ is an injective function $\mathtt{G}\colon \mathrm{npos}(\xi) \to \mathrm{yd}(\xi)$ such that, for each $\rho \in \mathrm{npos}(\xi)$, the assigned leaf $\mathtt{G}(\rho)$ is in $\mathrm{yd}(\xi|_\rho)$. Figure 4 shows a guide for our example constituent structure, assigning a leaf (illustrated in gray circles) to each inner node. As $\mathtt{G}$ is injective and there is one less inner node than leaves in $\xi$, there is exactly one leaf that is not in the image of $\mathtt{G}$. We will investigate multiple strategies, called *guide constructors*, to define guides for a given constituent tree as discussed in Section 3.3.

**(iii) Lexical rule induction.** In this step, we will construct a lexical LCFRS rule $r'$ and the components $c$, $t$ and $p$ of the supertag for each leaf in the constituent structure. The nonterminals in the rule are determined by a chosen hyperparameter $\mathtt{NT}$, called the *nonterminal constructor*, in terms of the constituent symbol and the guide $\mathtt{G}$ as described in Section 3.3. We give examples for (the root position in) the tree in Fig. 4, the guide values shown in pentagons (shortest guide constructor) and constituent symbols as nonterminals (classic nonterminal constructor).

We start with the leaf $i'$ that is not in the image of $\mathtt{G}$ and define a new nonterminal L-$A$ using the fixed string "L-" and the nonterminal $A$ produced by $\mathtt{NT}$ for the parent of $i'$. In that case $c = t =$ None, $p = pos(i')$, and $r' = \text{L-}A \to [i']$. In Fig. 4, the leaf $i' = 2$ is not in the image of $\mathtt{G}$, it yields $r' = \text{L-NP} \to [2]$, and $p = \text{NN}$).

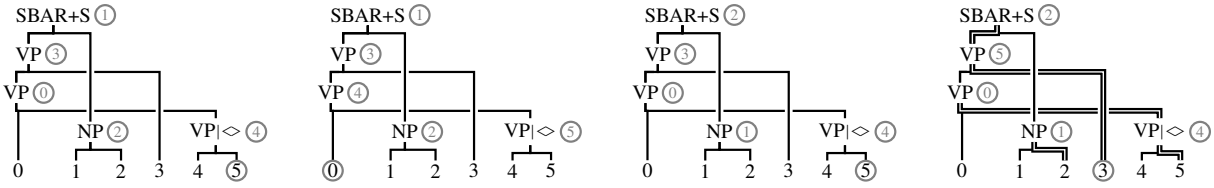
After that, we define the following for each position $\rho \in \mathrm{npos}(\xi)$ (from bottom up) and its assigned leaf $i = \mathtt{G}(\rho)$:

- The LCFRS rule $r'$ is assembled in the usual manner from $i$ as lexical symbol and variables for the spans formed by the leaves in each successor except those leaves that are assigned by $\mathtt{G}$ to $\rho$'s ancestors. $\mathtt{NT}$ produces the lhs nonterminal for $\rho$, the rhs nonterminals are the lhs nonterminals constructed for $\rho$'s children. In our example, $r$ is assembled from the lexical symbol 3, the left successor's leaves are $\{\underline{0}_{(x_1)}, \underline{4,5}_{(x_2)}\}$ and the right one's are $\{\underline{1,2}_{(y_1)}\}$; hence $r = \text{S} \to [x_1\, y_1\, 3\, x_2](\text{VP}_2, \text{NP})$.
- $t$ is *None* if the leaf $\mathtt{G}(\rho)$ is a direct child of $\rho$, otherwise it is the index among the children where $\mathtt{G}(\rho)$ is located. In our example, 3 is not a child of the root, it is in its first successor; therefore $t = 1$.
- $c$ is $\xi(\rho)$ if it was not introduced during binarization (i.e. the symbol $\xi(\rho)$ does not contain some markers $|\langle . . \rangle$), otherwise it is None. In our example, $c =$ SBAR+S.
- $p$ is the pos symbol at $\mathtt{G}(\rho)$ in $pos$. In our example $p =$ VBD.

**(iv) Supertag extraction.** The tuples constructed in the previous step closely resemble supertags as described in the previous subsection. We pull them from the constructed tree and order them according to the sentence position included in the lexical rule. Lastly, the sentence position in each rule is replaced by a wildcard symbol "_".

### 3.3 Extraction parameters

Apart from the constituent treebank, our extraction algorithm expects parameters for binarization, a *guide constructor* and a *nonterminal constructor*. The *vanilla* parameters coincide with the existing algorithm.




(a) vanilla guide (b) strict guide (c) least guide (d) modifier guide

Figure 5: Guides defined by the constructors introduced in Section 3.3. Gray integers show the leaf assigned to each inner node for the binary constituent structure. Encircled leaves are not in the image of the guide. The guide defined by the *shortest constructor* is shown in Fig. 4.

**Binarization parameters.** During binarization, we distinguish factorization (the direction in which new nodes are introduced) from left to right (*lr bin*) or head-outward (*ho bin*, mixes left and right-branching nodes such that the node's lexical head is among at the last one).[6] Both strategies are extended by markovization. The width of the horizontal markovization window is denoted by $h$, the vertical one by $v$.

**Guide constructors.** We define guides for given constituent structures using the following strategies. Figures 4 and 5 show the leaf assigned to each inner node for each guide constructor in an example constituent structure.

• *vanilla*: The guide maps each node position either to the leftmost leaf that is a direct successor, or (if not available) to the leftmost leaf in the yield of its right successor. The assignment is determined for each node from top to bottom.

• *strict*: The guide maps each node position to the leftmost leaf in the yield of its right successor.

• *modifier*: The guide maps each position to its modifier's lexical head. This guide requires that the constituents structures are binarized head-outward, which guarantees that each inner node has exactly one modifier.

• *least*: The guide aims to map many positions to leafs that are direct children. The guide is determined for each position from bottom to the top and selects the nearest (and leftmost, if ambiguous) leaf for each position.

• *shortest*: The guide aims to map many positions to leaves that are as near as possible. The guide is determined for each position from top to bottom and, similar to the least guide, selects the nearest (and leftmost, if ambiguous) leaf for each position. When searching for the nearest leaf, we exclude a subtree if a leaf in it was selected previously.

**Nonterminal constructors.** A *nonterminal constructor* computes a lhs nonterminal for the grammar rule included in each supertag. Each of the following constructors computes a nonterminal for the position $\rho$ in $\xi$ from the constituent symbol $\xi(\rho)$, the set of leaves $\mathrm{yd}(\xi|_\rho)$ below $\rho$ and the set of leaves $L$ assigned by $\mathtt{G}$ to the ancestors of $\rho$. We omit the fanout subscripts if they are 1. We give examples using the *shortest guide*, as shown in Fig. 4, for the root position $\varepsilon$ where $\xi(\varepsilon) =$ SBAR+S, $\mathrm{yd}(\xi|_\varepsilon) = \{0, \ldots, 5\}$ and $L = \emptyset$ and the position 1.1 of the bottom VP node where $\xi(1.1) =$ VP, $\mathrm{yd}(\xi|_{1.1}) = \{0, 4, 5\}$, $L = \{0, 3\}$ and $\mathrm{fo}(\mathrm{yd}(\xi|_{1.1}) \setminus L) = 1$.

• *vanilla*: The nonterminal consists of the symbol $\xi(\rho)$, the fanout $\mathrm{fo}(\mathrm{yd}(\xi|_\rho))$ as subscript, and if $L$ contains any leaf in $\mathrm{yd}(\xi|_\rho)$, then the difference in fanout $\mathrm{fo}(\mathrm{yd}(\xi|_\rho) \setminus L) - \mathrm{fo}(\mathrm{yd}(\xi|_\rho))$ as superscript. This superscript indicates the difference in fanout at $\rho$ in the original constituent tree compared to the leaves assigned to the nodes in the subtree at $\rho$ by the guide. (At most one leaf is assigned to an ancestor of $\rho$.) In our two examples, we construct the nonterminals SBAR+S and $\mathrm{VP}_2^{-1}$

• *classic*: The nonterminal consists of the first symbol[7] in $\xi(\rho)$ (including markers introduced during binarization) and the fanout $\mathrm{fo}(\mathrm{yd}(\xi|_\rho) \setminus L)$ as subscript. This constructor omits annotations that depend on the guide and is more akin to usual strategies in LCFRS extraction (Maier and Søgaard, 2008). For our examples, the nonterminals are SBAR and VP.

[6] All three treebanks that we used are biased towards right-branching structures. In this setting, parsing with right-branching grammars has been shown to be more efficient and equally accurate than with left-branching grammars. (Bodenstab, 2012) Here, we focus on comparing right-branching to the linguistically motivated head-outward binarization scheme, and drop an investigation for left-branching binarization.

[7] After merging unary nodes in step (i) of the extraction, each node may consist of multiple constituent symbols.

• *coarse*: Like the *classic* nonterminals, but we replace the constituent symbols occurring in the treebank by their first letter. This is a very rough approximation of nonterminals in coarse-to-fine parsing (Charniak et al., 2006) that does not need a specific clustering for each treebank. For our examples, the nonterminals are S and V.

### 3.4 Parsing

For parsing, a small sample of supertags is predicted for each position in the input. Each tag is equipped with a score expressing its prediction confidence among all supertags for this position.

**(v) Supertag derivations.** For each predicted tag, the wildcard in the LCFRS rule is replaced by the input position it was predicted for. The sequence of input positions is parsed using the set of all lexical LCFRS rules inside the supertags. We equip each LCFRS rule with the prediction confidence of its supertag[8] and use discodop (van Cranenburgh et al., 2016), an off-the-shelf statistical parser, to pick an LCFRS derivation which maximizes the product of prediction confidences. In this derivation, we replace each LCFRS rule by the supertag tuple it was taken from.

**(vi) Transformation into constituent trees.** The function listed in Algorithm 1 transforms the tree of supertags obtained in the previous step into a constituent structure for each node from top to bottom via recursive calls. Its first argument is the (sub-)tree to transform, the second one ($i_1$) is either *None* or a leaf that is transported from the top to a lower position. The leaf for the current supertag ($i_2$) is read from the lexical rule $r$ (line 4). The list of children $\vec{d}$ in the supertag tree and the index $t$ determine how the list of (usually) two children $\vec{s}$ is assembled (lines 6–12): each child in $\vec{d}$ will establish a child in $\vec{s}$ via a recursive call (line 9); if there is only one child in $\vec{d}$, $t$ will determine which leaf in $\{i_1, i_2\}$ is absorbed into $\vec{s}$ (line 6 and 11); if there is no child then both $i_1$ and $i_2$ are in $\vec{s}$ (line 11). Remaining leaves in $\{i_1, i_2\}$ will be transported into the children according to $t$ (line 6 and 9). If $c$ is not *None*, the function adopts the constituent symbols (lines 13–15), otherwise the children in $\vec{s}$ are merged with their siblings (lines 16 and 10). The pos symbol $p$ for the leaf $i_2$ is stored in a set $pos$ (line 5) and merged with the other pos symbols for each leaf (lines 9 and 15).

[8] If multiple supertags have the same LCFRS rule, we use the greatest prediction confidence.

**Algorithm 1** Transforming a derivation of supertags into a constituent structure and pos tags. $\text{SYMB}(r)$ is the terminal in the lexical rule $r$.

**Require:** $d = tag(\vec{d})$: tree of supertags with root $tag$ and list of children $\vec{d}$ where $|\vec{d}| \in \{0, 1, 2\}$
**Ensure:** if $d$ was extracted in steps (i–iii) from a constituent tree $(\xi, p, w)$, then $\text{CTREE}(d, \text{None}) = (\xi, \{(i, p(i)) \mid 1 \leq i \leq |p|\})$

```
1:  function CTREE(tag(d⃗), i_1)
2:      (r, t, c, p) ← tag
3:      s⃗ ← ε
4:      i_2 ← SYMB(r)
5:      pos ← {(i_2, p)}
6:      if t = 1 then i_1, i_2 ← i_2, i_1
7:      for idx in {1, 2} do
8:          if |d⃗| ≥ idx then
9:              add CTREE(d⃗[idx], i_idx) to s⃗, pos
10:         else if i_idx ≠ None then
11:             add i_idx to s⃗ as a leaf
12:     if c is not None then
13:         for c′ in reverse(split(c, +)) do
14:             s⃗ ← c′(s⃗)
15:     return s⃗, pos
```

## 4 Experiments

Our experiments are conducted with the usual train/dev/test splits[9] for the three discontinuous constituent treebanks DPTB, NeGra and Tiger. For each treebank, we select parameters for the extraction using an incomplete grid search as described in Section 4.1. For the final models, we fine-tune each a bert-base-cased (bert-base-german-cased for NeGra and Tiger) and a bert-large-cased (gbert-large, respectively; Devlin et al., 2019; Chan et al., 2020) model with the selected parameter configurations for 20 epochs and report the parsing scores and speed in Table 6. We use the selected parameters for fine-tuning the same classifiers (Devlin et al., 2019, cf. sec. A.2). Appendix A lists the details for the classifier's training parameters and the used computing infrastructure. Results for models trained without any pretrained embeddings are shown in Appendix B.

[9] We use the split for NeGra by Dubey and Keller (2003), for Tiger by Seddah et al. (2013), and the standard split for DPTB (sections 2–21 for training, 22 for development, 23 for testing). In evaluation, we use the implementation for F-scores by van Cranenburgh et al. (2016) with default parameters in `proper.prm`.

Table 1: Number of extracted core supertags, parsing scores (F1, F1-d), number of parse fails (↯) and prediction accuracy (acc.) in NeGra for varying cores.

| core | no. core tags | F1 | F1-d | ↯ | acc. core | acc. $c$ | acc. $t$ | acc. $p$ |
|---|---|---|---|---|---|---|---|---|
| r | 2078 | 88.0 | 62.1 | 7 | 86.8 | 94.4 | 95.7 | 98.8 |
| rt | 2294 | 88.8 | 68.9 | 8 | 86.7 | 94.3 | — | 98.7 |
| rc | 2151 | 89.0 | 62.8 | 2 | 86.5 | — | 95.4 | 98.8 |
| rp | 7695 | 86.9 | 62.1 | 15 | 84.5 | 94.1 | 95.5 | 96.3 |
| rtc | 2368 | 90.6 | 70.7 | 1 | 86.9 | — | — | 98.7 |
| rtp | 8207 | 87.4 | 64.1 | 18 | 84.6 | 94.3 | — | 96.1 |
| rcp | 7784 | 87.6 | 60.3 | 10 | 84.2 | — | 95.7 | 96.4 |
| rtcp | 8295 | 88.7 | 66.4 | 12 | 84.0 | — | — | 96.8 |

### 4.1 Parameter selection.

For each treebank, we conduct a parameter search in four steps to select a satisfactory configuration.

(i) First, we investigate which components of the supertag tuples to predict in tandem (*core supertags*) and which ones separately for each position in a sentence.

(ii) We investigate a set of combinations for nonterminal and guide constructors and eliminate underperforming ones.

(iii) The previously found set of combinations is investigated for each binarization configuration and one final configuration is chosen.

(iv) Finally, we assess how many tags per sentence position ($k$) shall be predicted.

Each step is implemented as a grid search. For each configuration in a grid, we fine-tune a bert-base model for 5 epochs using the supertags extracted from the training set, and evaluate using the dev set of the treebank. This process is documented for the NeGra treebank in the following paragraphs in very detail as an example.

**(i) Core supertags.** We investigate if there are advantages in predicting subsets of supertag components jointly (*core*) while the others are determined independently.[10] The core components always include the grammar rule $r$. During parsing, we use the top $k$ predictions for core components and only the best prediction for each other component. We ran this experiment with one parameter configuration (vanilla guide, classic nonterminals, lr bin with $v = 1$ and $h = 0$, $k = 10$). and propose that the results shown in Table 1 are clear enough to omit repeating this experiment with other configurations. They suggest that the prediction and parsing accuracies benefit from the absence of pos tags as well as the presence of both other components. Not including the pos tags, there are far less core supertag tuples and the quality of pos tag prediction is significantly better. We continue all following experiments with the core $(r, t, c)$.

[10] In supertags as defined by Ruprecht and Mörbitz (2021), grammar rules $r$, constituent symbols $c$ and indices $t$ were all stored in the grammar rules, the pos symbols were predicted separately.

Table 2: No. supertags extracted from NeGra by nonterminal constructors (rows) and guides (columns).

| | vanilla | strict | least | shortest | modifier |
|---|---|---|---|---|---|
| vanilla | 3265 | 2773 | 4677 | 4236 | 4528 |
| classic | 2367 | 2228 | 3544 | 3611 | 3587 |
| coarse | 1754 | 1677 | 2823 | 2837 | 2933 |

Table 3: Parsing scores (F1) and supertag prediction accuracy (acc.) in NeGra for combinations of nonterminal constructors (rows) and guides (columns).

| | vanilla | | strict | | least | | shortest | | modifier | |
|---|---|---|---|---|---|---|---|---|---|---|
| | F1 | acc. | F1 | acc. | F1 | acc. | F1 | acc. | F1 | acc. |
| vanilla | 89.7 | 85.9 | 90.9 | 88.2 | 86.9 | 78.9 | 87.4 | 78.9 | 89.1 | 87.1 |
| classic | 90.5 | 84.6 | 91.2 | 89.0 | 88.8 | 82.6 | 87.8 | 78.8 | 90.4 | 87.3 |
| coarse | 89.9 | 85.7 | 90.8 | 88.8 | 88.3 | 82.5 | 87.7 | 79.6 | 90.1 | 87.8 |

**(ii) Guide and Nonterminal Constructors.** Here, we extract supertags for each combination of nonterminal and guide constructor. Binarization is fixed to lr (except for the modifier guide which requires ho) with $h = 0$ and $v = 1$, and $k = 10$. In Table 2, we can clearly see that both parameters determine the size of the grammar; of course that behavior was intended for the nonterminal constructors. Significantly less supertags are extracted using the strict and vanilla constructors than with the three other guides. Table 3 shows that the strict guide takes a clear lead in prediction and parsing accuracy. We continue the search restricted to the strict guide and all three nonterminal constructors.

**(iii) Binarization.** We extract supertags using the following configurations for binarization: ho and lr, each with horizontal markovization context $h \in \{0, 1\}$ and vertical markovization context $v \in \{0, 1, 2\}$. Table 4 shows the parsing scores for supertags extracted using these combinations. Markovization contexts $h > 0$ and $v > 1$ do not seem to give us an advantage in this setting, it is clearly disadvantageous with vanilla and classic

Table 4: Parsing scores (F1) in NeGra using supertags extracted with different configurations for binarization (rows distinguish lr and ho, columns distinguish values for $h$ and $v$) and nonterminal constructors (rows).

| | | $h=0$ | | | $h=1$ | | |
|---|---|---|---|---|---|---|---|
| | | $v=0$ | $v=1$ | $v=2$ | $v=0$ | $v=1$ | $v=2$ |
| | | F1 | F1 | F1 | F1 | F1 | F1 |
| vanilla | rl | 90.8 | 90.9 | 89.9 | 89.3 | 87.5 | 85.0 |
| | ho | 84.6 | 89.8 | 88.6 | 88.2 | 86.2 | 81.1 |
| classic | rl | 90.9 | 91.2 | 89.9 | 89.8 | 88.5 | 86.1 |
| | ho | 84.2 | 90.9 | 88.5 | 89.2 | 87.6 | 83.9 |
| coarse | rl | 90.5 | 90.8 | 90.1 | 90.4 | 90.0 | 89.5 |
| | ho | 84.2 | 90.5 | 89.7 | 89.6 | 89.9 | 89.2 |

Table 5: Parsing scores (F1, F1-d), number of parse fails (↯) and speed (sent/s) in NeGra for varying amounts of supertags considered during parsing ($k$).

| $k$ | NeGra | | | | Tiger | | | | DPTB | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F1 | F1-d | ↯ | sent/s | F1 | F1-d | ↯ | sent/s | F1 | F1-d | ↯ | sent/s |
| 5 | 90.7 | 73.1 | 10 | 148 | 92.9 | 76.1 | 110 | 133 | 93.6 | 85.9 | 34 | 85 |
| 10 | 91.1 | 73.8 | 1 | 125 | 93.1 | 76.9 | 10 | 130 | 94.7 | 88.0 | 7 | 61 |
| 15 | 91.2 | 74.4 | 0 | 109 | 93.1 | 76.8 | 0 | 115 | 94.8 | 88.1 | 3 | 50 |
| 25 | 91.3 | 74.4 | 0 | 72 | 93.1 | 76.5 | 0 | 83 | 94.9 | 88.3 | 2 | 52 |
| 40 | 91.3 | 74.9 | 0 | 65 | 93.1 | 76.6 | 0 | 65 | 94.9 | 88.6 | 0 | 35 |

nonterminals. The impact of greater contexts is significantly less with coarse nonterminals. However, it does not benefit from higher values either. We select the final configuration for NeGra via the highest F1-score in the table: classic nonterminals and lr bin with $h = 0$ and $v = 1$.

**(iv) Predictions per Position** After training the final models with bert-base, we pick a suitable value for $k$ from the set $\{5, 10, 15, 25, 40\}$. From the results in Table 5, we suggest that there is only one case where a value $k \neq 10$ shows improvements in quality that justifies the given decrease in speed, that is $k = 15$ for parsing NeGra. For both other treebanks, we continue with $k = 10$.

### 4.2 Final configurations.

In the parameter search, we found the following configurations for our final models:

(DPTB) strict guide, classic nonterminals with lr binarization where $h = 0$ and $v = 2$,

(NeGra) strict guide, classic nonterminals with lr binarization where $h = 0$ and $v = 1$,

(Tiger) strict guide, coarse nonterminals with lr binarization where $h = 0$ and $v = 1$.

Each model is trained to predict pos symbols separately from the other supertag components. We use the top $k = 10$ (DPTB, Tiger) and $k = 15$ (NeGra) predicted supertags during parsing.

## 5 Conclusion

We generalized the extraction of supertags from treebanks by introducing parameters for previously fixed parts of the construction. The parameters allow us to control the part of the constituent tree that is associated with a terminal for each supertag (guide constructor) as well as the granularity of the grammar rules (nonterminal constructor). At the same time, the extraction process was re-ordered so that its description is less convoluted while retaining the same functionality.

Some introduced guide and nonterminal constructors performed better than the vanilla variants. Specifically, we observe the following: While the highly ambiguous grammars extracted from DPTB benefit from finer nonterminal granularity with greater markovization window, the large and more specific grammars extracted from Tiger improve with coarser granularity; the grammar for NeGra lies somewhere in between.

Compared to the previous implementation of supertagging with LCFRS, we could improve the parsing quality across all three discontinuous treebanks. The improvements close the gap between the quality of parsing with LCFRS supertagging and the most recent discontinuous parsing strategies. In case of the two German treebanks, we could even surpass them, most notably in terms of the F1-score for discontinuous constituents.

## 6 Limitations

In our experiments, we chose to heavily restrict the set of hyperparameters for the supertag extraction and training of the neural model to finish them in feasible time. (We used fixed parameters for training and a step-wise incomplete grid search for the extraction.) Therefore, some interactions between parameters might still be concealed and optimal solutions yet to be found.

While we achieve state-of the art results with the two German treebanks NeGra and Tiger, our results fall behind the competition in the English DPTB. We could not find a concluding reason for that, and it should be further investigated.

We clearly emphasize that our implementation is a prototype, especially the reported parsing speed in Table 6 should not be considered final for the approach. The two major reasons are that

- the statistical LCFRS parser that we used for step (v) in Section 3.4 is not optimized for

Table 6: Our results on test sets compared to other published parsers for discontinuous constituents. The column "Type" gives a rough classification of the parsing approach in the following concepts: G – statistical grammar-based, GS – grammar-based with supertagging, C – grammarless chart-based, T – transition-based, N – untraditional neural approaches. *bert-b* and *bert-L* are language specific bert-base and bert-large models.
† – Fernández-González and Gómez-Rodríguez (2021a) use RoBERTa-large (Liu et al., 2019) and GottBERT-base (Scheible et al., 2020) instead of bert-large and gbert-large.

| Type | Model | pretrained embeddings | NeGra F1 | NeGra F1-d | NeGra sent/s | Tiger F1 | Tiger F1-d | Tiger sent/s | DPTB F1 | DPTB F1-d | DPTB sent/s |
|---|---|---|---|---|---|---|---|---|---|---|---|
| G | van Cranenburgh et al., 2016 | – | 76.8 | – | 2 | 78.2 | – | 1 | 87.0 | – | < 1 |
| | Gebhardt, 2020 | – | 81.7 | 43.5 | – | 77.7 | 40.7 | – | – | – | – |
| | Versley, 2016 | – | – | – | – | 79.5 | – | – | – | – | – |
| GS | ours | (bert-b) | 91.8 | 74.6 | 120 | 89.7 | 72.6 | 105 | 94.4 | 82.0 | 81 |
| | ours | (bert-L) | **93.9** | **79.1** | 88 | **91.6** | **75.4** | 77 | 94.9 | 82.4 | 64 |
| | Ruprecht and Mörbitz, 2021 | (bert-b) | 90.9 | 72.6 | 68 | 88.3 | 69.0 | 60 | 93.3 | 80.5 | 57 |
| C | Corro, 2020 | (bert-b) | 91.6 | 66.1 | – | 90.0 | 62.1 | – | 94.8 | 68.9 | – |
| | Stanojević and Steedman, 2020 | – | 83.3 | 50.7 | 15 | 83.4 | 53.5 | 9 | 90.5 | 67.7 | – |
| T | Coavoux, 2021 | (bert-b) | 91.7 | 73.3 | – | 90.2 | 72.9 | – | 95.0 | 82.5 | – |
| N | Vilares, Gómez-R., 2020 | (bert-b) | 84.2 | 46.9 | 80 | 84.7 | 51.6 | 80 | 91.7 | 49.1 | 80 |
| | Vilares, Gómez-R., 2020 | (bert-L) | – | – | – | – | – | – | 92.8 | 53.9 | – |
| | Fernández-G., Gómez-R., 2021a | (bert-b) | 90.0 | 65.9 | 275 | 88.5 | 63.0 | 238 | 94.0 | 72.9 | 231 |
| | Fernández-G., Gómez-R., 2021a | (bert-L) | 92.0 | 67.9 | 216 | 90.5 | 68.1 | 207 | 95.1 | 74.1 | 193 |
| | Fernández-G., Gómez-R., 2021b | (bert-L)† | 89.1 | 67.1 | – | 88.5 | 67.8 | – | **95.5** | **82.9** | – |
| | Fernández-G., Gómez-R., 2022 | (bert-b) | 91.0 | 76.6 | – | 90.0 | 62.6 | – | – | – | – |

lexical grammar rules and could possibly be improved upon, and

- `flair`, the framework that we use for training the neural classifiers, is easy to use in development but rather slow during execution.

In this approach we extract supertags equipped with lexical grammar rules using an injective mapping between binary constituent tree nodes and sentence positions (called guide, cf. step (iii) in Section 3.2). We suggest it could be sensible to associate some sentence positions with multiple constituent tree nodes or no nodes at all. E.g. we found a guide that maps each node position to its lexical head intuitive, but multiple nodes may share the same lexical head while some leaves are not the lexical head of any node. Even though we decouple the constituent symbols from the grammar rules in our formulation, the formalisms that we use can not keep track of the child relations in multiple constituent nodes per supertag.

Lastly, our approach inherits the problem of incomplete coverages from parsing with grammars. As Table 5 shows, there are still small amounts of sentences in all three used datasets that fail to parse in certain configurations. We chose to accept these in a trade-off with parsing speed. However, in settings where parse fails are critical, the extraction and parsing parameters should be selected very carefully.

## References


Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265.

Tatiana Bladier, Andreas van Cranenburgh, Younes Samih, and Laura Kallmeyer. 2018. German and French neural supertagging experiments for LTAG parsing. In *Proceedings of ACL 2018, Student Research Workshop*, pages 59–66, Melbourne, Australia. Association for Computational Linguistics.

Nathan Matthew Bodenstab. 2012. *Prioritization and Pruning: Efficient Inference with Weighted Context-Free Grammars*. Ph.D. thesis.

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.

Branden Chan, Stefan Schweter, and Timo Möller. 2020. German's next language model. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6788–6796, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R Shrivaths, Jeremy Moore, Michael Pozar, et al. 2006. Multilevel coarse-to-fine pcfg parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 168–175.

Stephen Clark. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+ 6)*, pages 19–24.

Maximin Coavoux. 2021. BERT-proof syntactic structures: Investigating errors in discontinuous constituency parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3259–3272, Online. Association for Computational Linguistics.

Caio Corro. 2020. Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(nˆ6) down to O(nˆ3). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2753–2764, Online. Association for Computational Linguistics.

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

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo, Japan. Association for Computational Linguistics.

Kilian Evang and Laura Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin, Ireland. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2021a. Discontinuous grammar as a foreign language.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2021b. Reducing discontinuous to continuous parsing with pointer network reordering.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2022. Multitask pointer network for multi-representational parsing. *Knowledge-Based Systems*, 236:107760.

Kilian Gebhardt. 2020. Advances in using grammars with latent annotations for discontinuous parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 91–97, Online. Association for Computational Linguistics.

Rekia Kadari, Yu Zhang, Weinan Zhang, and Ting Liu. 2018. CCG supertagging via bidirectional LSTM-CRF neural architecture. *Neurocomputing*, 283:31–37.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Jungo Kasai, Bob Frank, Tom McCoy, Owen Rambow, and Alexis Nasr. 2017. TAG parsing with neural networks and vector representations of supertags. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722, Copenhagen, Denmark. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proceedings of the 13th conference on Formal Grammar*, pages 61–76, Hamburg, Germany. CSLI Publications.

Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119, Plainsboro, New Jersey. Association for Computational Linguistics.

Thomas Ruprecht and Richard Mörbitz. 2021. Supertagging-based parsing with linear context-free rewriting systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Online. Association for Computational Linguistics.

Raphael Scheible, Fabian Thomczyk, Patric Tippmann, Victor Jaravine, and Martin Boeker. 2020. Gottbert: a pure german language model.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics.

Wojciech Skut, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit. 1998. A linguistically interpreted corpus of German newspaper text. In *Proceedings of the ESSLLI Workshop on Recent Advances in Corpus Annotation*, Saarbrücken, Germany.

Miloš Stanojević and Mark Steedman. 2020. Span-based LCFRS-2 parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 111–121, Online. Association for Computational Linguistics.

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.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237, San Diego, California. Association for Computational Linguistics.

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.

David Vilares and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing as sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2771–2785, Online. Association for Computational Linguistics.

## A Neural Classifier

The following table contains the values for the neural classifier's architecture and hyperparameters during training. All experiments were run on the same compute server with an Intel Xeon Silver 4114 (40 cores 2.2GHz), 256GB RAM and an Nvidia GeForce RTX 2080.

| parameter | value |
|---|---|
| embeddings | last 4 layers of bert-base/bert-large |
| architecture | single feed forward layer per tag type (core supertags and pos tags) |
| loss | cross entropy |
| learning rate | $5 \cdot 10^{-5}$ |
| weight decay | $10^{-2}$ |
| dropout | $10^{-1}$ |
| optimizer | AdamW |
| batch size | 32 |
| epochs | 5 during grid search<br>20 for final model |

## B Supervised Classifier

We trained classifiers that only rely on the data in the training corpus (i.e. without pre-trained embeddings) using the same hyperparameters for the extraction that were found in Section 4.1. For that, we used the same supervised architecture for word encoding as Stanojević and Steedman (2020); Corro (2020): each word is embedded into the concatenation of per-word vectors and the output of a character-level bi-LSTM. These embeddings are then fed into a sentence-level bi-LSTM. Finally, the score of each tag prediction is computed using a multi-layer perceptron. The following table gives an overview of the used architecture and used hyperparameters.

| parameter | value |
|---|---|
| word embeddings | 300 |
| character-level bi-LSTM layers | 1 |
| character-level bi-LSTM dim | 100 |
| sentence-level bi-LSTM layers | 2 |
| sentence-level bi-LSTM dim | 800 |
| prediction MLP layers | 3 |
| prediction MLP dim | 800 |
| prediction MLP activation | ReLU |
| loss | cross entropy |
| learning rate | $10^{-3}$ |
| weight decay | $10^{-1}$ |
| dropout | 0.5 |
| optimizer | AdamW |
| batch size | 32 |
| epochs | 32 |

Table 7: Our results on test sets compared to other published parsers for discontinuous constituents using supervised embeddings. The column "Type" gives a rough classification of the parsing approach in the following concepts: G – statistical grammar-based, GS – grammar-based with supertagging, C – grammarless chart-based, T – transition-based, N – untraditional neural approaches.

| Type | Model | pretrained embeddings | NeGra | | | Tiger | | | DPTB | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | F1 | F1-d | sent/s | F1 | F1-d | sent/s | F1 | F1-d | sent/s |
| G | van Cranenburgh et al., 2016 | – | 76.8 | – | 2 | 78.2 | – | 1 | 87.0 | – | $< 1$ |
| | Gebhardt, 2020 | – | 81.7 | 43.5 | – | 77.7 | 40.7 | – | – | – | – |
| | Versley, 2016 | – | – | – | – | 79.5 | – | – | – | – | – |
| GS | ours | – | 83.7 | 52.9 | 132 | 83.7 | **59.5** | 104 | 91.7 | 74.2 | 82 |
| | Ruprecht and Mörbitz, 2021 | – | 82.7 | 49.0 | 136 | 82.5 | 55.9 | 101 | 90.1 | 72.9 | 95 |
| C | Corro, 2020 | – | **86.3** | **56.1** | 478 | **85.2** | 51.2 | 474 | **92.9** | 64.9 | 355 |
| | Stanojević and Steedman, 2020 | – | 83.3 | 50.7 | 15 | 83.4 | 53.5 | 9 | 90.5 | 67.7 | – |
| T | Coavoux, 2021 | – | 82.3 | 55.6 | – | 82.9 | 57.4 | – | 91.4 | **74.4** | – |
| N | Vilares, Gómez-R., 2020 | – | 77.1 | 36.5 | 715 | 79.2 | 40.1 | 568 | 89.1 | 41.8 | 611 |