Supertagging-based Parsing with Linear Context-free Rewriting Systems

Richard Mörlitz and Thomas Ruprecht
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
{richard.moerbitz, thomas.ruprecht}@tu-dresden.de

Abstract

We present the first supertagging-based parser for LCFRS. It utilizes neural classifiers and tremendously outperforms previous LCFRS-based parsers in both accuracy and parsing speed. Moreover, our results keep up with the best (general) discontinuous parsers, particularly the scores for discontinuous constituents are excellent. The heart of our approach is an efficient lexicalization procedure which induces a lexical LCFRS from any discontinuous treebank. It is an adaptation of previous work by Mörlitz and Ruprecht (2020). We also describe a modification to usual chart-based LCFRS parsing that accounts for supertagging and introduce a procedure for the transformation of lexical LCFRS derivations into equivalent parse trees of the original treebank. Our approach is implemented and evaluated on the English Discontinuous Penn Treebank and the German corpora NeGra and Tiger.

1 Introduction

Constituency parsing is a syntactical analysis in NLP that aims to enhance sentences with, usually tree-shaped, phrase structures (for an example cf. the left of Fig. 1). Formalisms such as context-free grammars (CFG) are used in this setting because they are conceptually simple, interpretable, and parsing is tractable (cubic in sentence length).

Discontinuous constituents span non-contiguous sets of positions in a sentence. The resulting phrase structures do not take the shape of a tree anymore, as they contain crossing branches (cf. the left of Fig. 1), and cannot be modeled by CFG. As a countermeasure, many corpora (e.g. the Penn Treebank (Marcus et al., 1994, PTB)) denote these phrase structures as trees nevertheless and introduce designated notations for discontinuity, which is then often ignored in parsing. However, discontinuity occurs in about 20% of the sentences in the PTB and to an even larger extent in German treebanks such as NeGra and Tiger. For parsing discontinuous constituents, so-called “mildly context-sensitive” grammar formalisms have been investigated, e.g. tree-adjoining grammars (TAG; Joshi et al., 1975) and linear context-free rewriting systems (LCFRS; Vijay-Shanker et al., 1987). These approaches have been empirically shown to indeed improve accuracy (cf. e.g. Evang and Kallmeyer, 2011). However, their increased expressiveness comes at the cost of a higher parsing complexity: given a sentence of length \(n\), parsing is in \(O(n^6)\) for TAG and \(O(n^3 \cdot \text{fanout}(G))\) for an LCFRS \(G\).

The fanout is grammar-specific and reflects the degree of discontinuity in the rules of \(G\). The expressiveness of TAG equals that of LCFRS with fanout 2. An LCFRS derivation of a discontinuous phrase is shown in the right of Fig. 1.

One approach for making parsing with mildly context-sensitive grammars tractable is supertagging, which was originally introduced for lexical TAG (Bangalore and Joshi, 1999). A TAG is lexical if each rule contains one word. The supertagger is a (often discriminative) classifier that selects for each position of the input sentence a subset of the rules of the TAG; these are the so-called supertags. Parsing is then performed with the much smaller grammar of supertags. Research on supertagging has also been conducted in the context of combined categorical grammars (Clark, 2002), but not yet for LCFRS. Recently, the use of recurrent neural networks (RNN) as classifiers in supertagging has improved not only the speed of such systems, but also their accuracy by far (Vaswani et al., 2016; Kasai et al., 2017; Bladier et al., 2018; Kadari et al., 2018).

In this paper, we present the first supertagging-based parser for LCFRS. We adapt the lexicaliza-


![Diagram of a discontinuous phrase structure tree](image)

Figure 1: A discontinuous phrase structure tree of the sentence *A hearing is scheduled on the issue today* (left) and a corresponding LCFRS derivation of the discontinuous noun phrase *A hearing on the issue* (right).

**2 Preliminaries**

The set of non-negative (resp. positive) integers is denoted by \( \mathbb{N} \) (resp. \( \mathbb{N}_+ \)). We abbreviate \([1, \ldots, n]\) by \([n]\) for each \( n \in \mathbb{N} \). Let \( A \) be a set; the set of (finite) strings over \( A \) is denoted by \( A^* \). An alphabet is a finite and non-empty set.

Let \( S \) be some set whose elements we call sorts. An \( S \)-sorted set is a tuple \((A, \text{sort})\) where \( A \) is a set and \( \text{sort}: A \to S \). Usually, we identify \((A, \text{sort})\) with \( A \). We denote sort by \( \text{sort}_A \) and the set \( \text{sort}^{-1}(s) \) by \( A_s \) for each \( s \in S \). We use set notation \((\in, \subseteq, \cup, \ldots)\) with sorted sets in the intuitive manner. Now let \( A \) be an \((S^* \times S)\)-sorted set. The set of trees over \( A \) is the \( S \)-sorted set \( T_A^* = \bigcup_{s \in S} T_A^s \) where \( T_A^s = \{ a(t_1, \ldots, t_k) \mid k \in \mathbb{N}, x_1, \ldots, x_k \in S, a \in A_{(s_1, \ldots, s_k)}, t_1 \in T_A^{s_1}, \ldots, t_k \in T_A^{s_k} \} \) for each \( s \in S \). A ranked set \( A \) is an \((S^* \times S)\)-sorted set where \( S = \{s\} \); the notation \( \text{rk}_A(a) = k \) abbreviates \( \text{sort}_{A_s}(a) = (s^k, s) \), and \( A_k \) abbreviates \( A_{(s^k, s)} \). If we use a usual set \( B \) in place of a ranked set, we will silently assume \( \text{rk}_B(b) = 0 \) for each \( b \in B \). Let \( X \) be a set. We let \( A(X) = \{ a(x_1, \ldots, x_k) \mid k \in \mathbb{N}, a \in A_k, x_1, \ldots, x_k \in X \} \).

**LCFRS.** Linear context-free rewriting systems extend the rule-based string rewriting mechanism of CFG to string tuples; we describe the generation process by compositions. Let \( k \in \mathbb{N} \) and \( s_1, \ldots, s_k, s \in \mathbb{N}_+ \); a \( \Sigma \)-composition is a tuple \((u_1, \ldots, u_s)\) where each \( u_i \) is a non-empty string over \( \Sigma \) and variables of the form \( x^i_j \) where \( i \in [k] \) and \( j \in [s] \). Each of these variables must occur exactly once in \( u_1 \cdots u_s \) and they are ordered such that \( x^i_1 \) occurs before \( x^i_{i+1} \) and \( x^j_i \) occurs before \( x^j_{i+1} \) for each \( i \in [k-1] \) and \( j \in [s_i-1] \). We denote the set of \( \Sigma \)-compositions by \( C^\Sigma_{s_1 \cdots s_k} \); we drop the superscript in the case \( \Sigma = \emptyset \) (then \( C_{s_1 \cdots s_k} \) is finite); we drop the subscript if we admit any configuration of \( k, s_1, \ldots, s_k \) and \( s \). We associate with each composition \((u_1, \ldots, u_s) \in C^\Sigma_{s_1 \cdots s_k}\) a function from \( k \) string tuples, where the \( i \)-th tuple is of length \( s_i \), to a string tuple of length \( s \). This function is denoted by \( [[u_1, \ldots, u_s]] \). Intuitively, it replaces each variable of the form \( x^i_j \) in \( u_1, \ldots, u_s \) by the \( j \)-th component of the \( i \)-th argument.

Let \( c \in C^\Sigma_{s_1 \cdots s_k} \) be a composition and \( \sigma \in \Sigma \). If \( s_i = 1 \) for some \( i \in [k] \), we obtain the i-parial application of \( c \) to \( \sigma \), denoted by \( [c]_{i}[\sigma] \in C^\Sigma_{s_1 \cdots s_i-1 s_{i+1} \cdots s_k} \), from \( c_2 \) by replacing \( x^i_j \) by \( \sigma \), and each \( x^i_{j'} \) by \( x^i_{j'+1} \) for \( j' > i \).

For each \( s \in \mathbb{N} \), we denote the composition \((x_1, \ldots, x_s)\) by \( \text{id}_s \).

An LCFRS is a tuple \( G = (N, \Sigma, S, R) \) where \( N \) is a finite \( \mathbb{N}_+ \)-sorted set (nonterminals), \( \cdot N \) is an alphabet (terminals), \( \cdot S \in N_1 \) (initial nonterminal), and \( \cdot R \) is a finite \((N^* \times N)\)-sorted set (rules). Each rule is of the form \( A \to c(B_1, \ldots, B_k) \), where \( k \in \mathbb{N} \), \( A, B_1, \ldots, B_k \in N \), and
c \in C^\Sigma_{\text{sort}(B_1)\cdots\text{sort}(B_k),\text{sort}(A)}}. The sort of the rule is \((B_1 \cdots B_k, A)\); we call \(A\) the left-hand side \((\text{lhs})\), \(B_1, \ldots, B_k\) the right-hand side \((\text{rhs})\) and \(c\) the rule’s \textit{composition}. We drop the parentheses around the rhs if \(k = 0\). We call rules of the form

• \(A \rightarrow c\) where \(A \neq S\) \textit{terminating}, • \(A \rightarrow c(B)\) \textit{monic}, and

\(\bullet A \rightarrow c(B_1, \ldots, B_k)\) where \(k \geq 2\) \textit{branching}. A rule is called \((\text{uni-}/\text{double-})\text{lexical}\), if its composition contains at least one terminal \((\text{resp. exactly one terminal/exactly two terminals})\). The LCFRS \(G\) is called \((\text{uni-}/\text{double-})\text{lexical}\), if each rule is \((\text{uni-}/\text{double-})\text{lexical}\) \textit{complete} derivations in \(G\) is \(D^G = T^G_S\). Let \(d = r(d_1, \ldots, d_k) \in T^R\) with \(r = A \rightarrow c(B_1, \ldots, B_k)\).

### 3 Obtaining Lexical LCFRS

We extract an uni-lexical LCFRS from a discontinuous corpus using a lexicalization scheme similar as described by Mörlitz and Ruprecht (2020).

Given a corpus, the procedure that induces an uni-lexical LCFRS from it is roughly as follows.

1. Binarize each tree in the corpus.
2. Transform each tree to an LCFRS derivation using the standard technique for induction of LCFRS (Maier and Søgaard, 2008).
3. Collapse every chain of monic rules; the non-terminals of each chain are combined to a new non-terminal.
4. Insert the lexical symbol of each non-initial terminating rule into its parent, then remove this rule.
5. Each branching rule has a distinct double-lexical terminating rule as a successor. Remove one terminal from the double-lexical rule and propagate it up the entire path to the branching rule.
6. Split all remaining double-lexical terminating rules into an uni-lexical monic rule and an uni-lexical terminating rule.
7. Read off the rules of each derivation. They are the rules of the uni-lexical LCFRS \(G_{\text{lex}}\).

Steps 3–5 correspond to a variation of the lexicalization scheme by Mörlitz and Ruprecht (2020). The approach described here differs from the previous one in that it works on individual derivations rather than on an entire grammar. As a consequence, lexical LCFRS obtained at the end may have a different language, but it is much smaller, which benefits the supertagging approach.

In the following, we describe the steps 3–6 in very detail and show examples in figs. 2–5. After step 2, we obtain derivations of a binary, terminal- and initial-separated LCFRS, i.e. each occurring rule is either of the form \(A \rightarrow (\sigma)\), where \(\sigma\) is a lexical symbol and \(A\) a part-of-speech tag, \(A \rightarrow c(B_1)\), or \(A \rightarrow c(B_1, B_2)\) where \(c\) contains no terminal symbols and none of \(B_1, B_2\) is the initial nonterminal. Furthermore, we assume the composition of each monic rule \(A \rightarrow c(B)\) to be \(c = id_{\text{lex}}(B)\). Let \(d\) be such a derivation.

**Step 3.** We repeatedly replace parts in \(d\) of the form \(A \rightarrow \text{id}_{\text{lex}}(B)(B \rightarrow c)\) by \(A + B \rightarrow c\), and \(A \rightarrow \text{id}_{\text{lex}}(B)(B \rightarrow c(C_1, C_2)(\ldots))\) by \(A + B \rightarrow c(C_1, C_2)(\ldots)\), until there is no monic rule in \(d\) left. If the rule \(A \rightarrow \text{id}_{\text{lex}}(B)\) has a parent in \(d\), then the nonterminal \(A\) in its rhs is also replaced by \(A + B\). After this step, there are only branching rules and terminating rules in \(d\). Fig. 2 shows an example for this step.

(a) A derivation for the string scheduled today. Light gray arrows show how the bottom-most composition is chained with the monic rules on top.

(b) The derivation resulting from step 3 applied to the derivation in fig. 2a.

Figure 2: Example for step 3.

**Step 4.** We replace all occurrences in \(d\) of the form

• \(A \rightarrow c(B_1, B_2)(B_1 \rightarrow (\sigma_1), B_2 \rightarrow (\sigma_2))\) by \(A \rightarrow \llbracket c(\sigma_1, \sigma_2)\rrbracket\),

• \(A \rightarrow c(B_1, B_2)(B_1 \rightarrow (\sigma_1), \ldots)\) by \(A \rightarrow \llbracket c(\sigma_1) (B_2)(\ldots)\) and

The following example illustrates the process.
• \( A \rightarrow c(B_1, B_2)(\ldots, B_2 \rightarrow (\sigma_2)) \) by \( A \rightarrow \{c\}_2(\langle \sigma_2 \rangle)(B_1)(\ldots) \).

The removed nonterminals, \( B_1 \) and/or \( B_2 \), are part-of-speech tags. We note that in each of the three cases, we construct a lexical rule, and after this step, every rule in \( d \) is either branching or lexical. Moreover, every terminal rule in \( d \) is either initial (and uni-lexical) or double-lexical. Fig. 3 shows an example for this step.

\[
\begin{align*}
\text{NP} &\rightarrow (x_1^1, x_1^2)(\text{NP}, \text{PP}) \\
\text{NP} &\rightarrow (x_1^1, x_2^1)(\text{DT}, \text{NN}) \\
\text{IN} &\rightarrow (\text{on}) \\
\text{NP}_1 &\rightarrow (x_1^2)(\text{DT}, \text{NN}) \\
\text{DT} &\rightarrow (\text{A}) \\
\text{NN} &\rightarrow (\text{hearing}) \\
\text{DT} &\rightarrow (\text{the}) \\
\text{NN} &\rightarrow (\text{issue}) \\
\end{align*}
\]

(a) A derivation for the string tuple (A hearing, on the issue). Light gray arrows show the lexical symbols that are put into binary non-lexical rules during step 4.

\[
\begin{align*}
\text{NP}_2 &\rightarrow (x_1^1, x_1^2)(\text{NP}_1, \text{PP}) \\
\text{NP}_1 &\rightarrow (\text{A hearing}) \\
\text{PP} &\rightarrow (\text{on } x_1^2)(\text{NP}_1) \\
\end{align*}
\]

(b) The derivation resulting from step 4 applied to the derivation in fig. 3a. Light gray arrows show how lexical symbols will be propagated through the derivation to lexicalize the rule in the root during step 5.

Figure 3: Example for step 4.

**Step 5.** For every occurrence \( r \) of a branching rule \( A \rightarrow c(A_1, A_2) \) in \( d \), let us consider the occurrence \( r \) of the leftmost terminating rule that is reachable via the second successor of \( r \). At each node \( s \) on the path from \( t \) to \( r \) (from bottom up):

• If \( s \) is \( t \), we remove the leftmost symbol in the rule’s composition at \( s \).

• If \( s \) is neither \( t \) nor \( r \), we insert the last removed symbol right before the variable \( x_1^1 \) and then remove the leftmost symbol in the rule’s composition at \( s \). If there is a monic rule at node \( s \), we store the information if the same symbol was inserted and removed or if the symbol in the rule was swapped. In the following, we consider this information as part of the rule, but it has no influence on its nonterminals and composition; it is solely needed for the back-transformation described in sec. 4.

• If \( s \) is \( r \), we insert the last removed symbol right before the variable \( x_1^2 \) in the rule’s composition at \( s \).

If, after removal of a symbol, the first component in the composition was empty, we annotate the lhs nonterminal (and the matching rhs nonterminal in the parent) with “−” and remove the empty component. Moreover, let this nonterminal be the 4th nonterminal in the rhs of the parent; we also remove \( x_1^1 \) in the parent’s composition and replace every other occurrence of \( x_1^1 \) by \( x_1^1 \). Otherwise, we annotate the nonterminals with “+”.

We note that the rule at \( s \) is uni-lexical and branching now, the rule at \( t \) is uni-lexical and terminating, and the number of lexical symbols in each rule between them did not change. After this step, every rule in \( d \) is lexical. Figs. 3b and 4 show an example for this step.

\[
\begin{align*}
\text{NP}_2 &\rightarrow (x_1^1, x_1^2)(\text{NP}_1, \text{PP}^+) \\
\text{NP}_1 &\rightarrow (\text{A hearing}) \\
\text{PP}^+ &\rightarrow (\text{on } x_1^2)(\text{NP}_1^+)^{\text{swapped}} \\
\text{NP}_1^+ &\rightarrow (\text{issue}) \\
\end{align*}
\]

Figure 4: The derivation resulting from step 5 applied to the derivation in fig. 3b. A light gray annotation “swapped” marks a monic rule where the terminal symbol changed.

**Step 6.** Each occurrence of the form \( A \rightarrow (\sigma_1, \sigma_2) \) is replaced by \( A \rightarrow (\sigma_1 x_1)(A^R)(A^R \rightarrow (\sigma_2)) \), and \( A \rightarrow (\sigma_1, x_1) \) is replaced by \( A \rightarrow (\sigma_1 x_1)(A^R)(A^R \rightarrow (\sigma_2)) \), where \( A^R \) is a new nonterminal. After this step, every rule in \( d \) is uni-lexical. Figs. 4 and 5 show an example for this step.

\[
\begin{align*}
\text{NP}_2 &\rightarrow (x_1^1, x_1^2)(\text{NP}_1, (\text{PP}^+)) \\
\text{NP}_1 &\rightarrow (\text{A } x_1^1)(\text{NP}_1^R) \\
\text{PP}^+ &\rightarrow (\text{on } x_1^2)(\text{NP}_1^+)^{\text{swapped}} \\
\text{NP}_1^+ &\rightarrow (\text{issue}) \\
\end{align*}
\]

Figure 5: The derivation resulting from step 6 applied to the derivation in fig. 4. Each rule in the derivation contains exactly one lexical symbol.

**4 Unlexicalizing Derivations**

The supertagging-based parser produces derivations of the LCFRS \( G_{\text{lex}} \) that was introduced in
Section 3. In this section, we describe how they are transformed into equivalent derivations of the LCFRS $G$ that is extracted from the corpus.

Given a derivation $d \in D^{\text{lex}}$, we have to apply $d$ the inverse transformation of each of the steps 6–1.

**Inverse of step 6.** Every subtree of the form $A \rightarrow (\sigma_1 x_1)(A^R)(A^R \rightarrow (\sigma_2))$ is replaced by the rule $A \rightarrow (\sigma_1 \sigma_2)$. Similarly, every subtree of the form $A \rightarrow (\sigma_1 x_1)(A^R)(A^R \rightarrow (\sigma_2))$ is replaced by the rule $A \rightarrow (\sigma_1 \sigma_2)$.

**Inverse of step 5.** This step includes a reconstruction of compositions, which is described in detail in app. A.1 as it is straight-forward, but includes many case distinctions. It is applied to each occurrence $r$ of branching rules of the form $A \rightarrow c(A_1, A_2)$ from the bottom to the top (i.e. it was already done for branching rules in the subtrees below each node). Let $t$ be the leftmost occurrence of a terminating rule that is reachable from the second child of $r$. At each node $s$ on the path from $r$ to $t$ we proceed as follows.

- If $s$ is $r$, let $\sigma$ be the lexical symbol in $c$. We replace $c$ as described in app. A.1. Then we pass $\sigma$ to the next node on the path to $t$.

- If $s$ is neither $r$ nor $t$, and there is a branching rule at $s$, we pass the passed symbol to the next node on the path to $t$. We note that the composition of $s$ was already changed before in this step.

- If there is a monic rule of the form $B \rightarrow c'(B_1)$ at $s$, we replace $c'$ as described in app. A.1. Let $\sigma_1$ be the passed symbol and $\sigma_2$ the lexical symbol in $c'$. If the terminal in this rule was swapped during step 5, we pass $\sigma_2$ to the next node on the path to $t$, otherwise we pass $\sigma_1$.

- If $s$ is $t$, it must be of the form $B \rightarrow (\sigma_2)$. Let $\sigma_1$ be the lexical symbol received from its parent. We replace it by $B \rightarrow (\sigma_1, \sigma_2)$ if $B$ is annotated with $-$ and by $B \rightarrow (\sigma_1 \sigma_2)$ otherwise.

All annotation is removed from the nonterminals after this step.

**Inverse of step 4.** We replace every occurrence of a terminating rule of the form

- $A \rightarrow (\sigma_1 \sigma_2)$ by $A \rightarrow (x_1, x_2)(A_1, A_2)(A_1 \rightarrow (\sigma_1), A_2 \rightarrow (\sigma_2))$ and

$A \rightarrow (\sigma_1, \sigma_2)$ by $A \rightarrow (x_1, x_2)(A_1, A_2)(A_1 \rightarrow (\sigma_1), A_2 \rightarrow (\sigma_2))$,

where $A_1$ and $A_2$ are the part-of-speech tags of $\sigma_1$ and $\sigma_2$, respectively.

We replace every occurrence of the form $A \rightarrow c(B)(\ldots)$, where $\sigma$ is the lexical symbol in $c$ and $A_1$ the part-of-speech tag of $\sigma$, as follows:

- If $\sigma$ is the first symbol in $c$, then $c'$ is obtained from $c$ by replacing, for each $j \in [\text{fo}(B)]$, $x^j_1$ with $x^j_1$ and $\sigma$ with $x^j_1$; the part is replaced by $A \rightarrow c'(A_1, B)(A_1 \rightarrow (\sigma), \ldots)$,

- otherwise, $c'$ is obtained from $c$ by replacing $\sigma$ with $x^j_1$; the part is replaced by $A \rightarrow c'(B, A_1)(\ldots, A_1 \rightarrow (\sigma))$.

The composition $c'$ is constructed such that $\llbracket c' \rrbracket_1(\sigma) = c$ in the first case, and $\llbracket c' \rrbracket_2(\sigma) = c$ in the second case.

We remark that the choice to use part-of-speech tags for the nonterminals $A_1$ and $A_2$ is pragmatic. It is motivated by the fact that, due to the annotation scheme of the treebank, the nonterminals $B_1$ and $B_2$ removed in step 4 are likely to be the part-of-speech tags of $\sigma_1$ and $\sigma_2$. This heuristics only fails when the part-of-speech tag is preceded by a chain, e.g. NP(NN(students)). In such a case we would collapse the chain in step 3 and thus remove the nonterminal NP + NN, losing the information about the chain. However, occurrences of this kind are so rare that we opted to ignore them.

**Inverse of step 3.** We repeatedly replace every occurrence of the form $A+B \rightarrow c(\ldots)(\ldots)$ by

$A \rightarrow \text{id}_{\text{fo}(B)}(B)(B \rightarrow c(\ldots)(\ldots))$,

until there are no nonterminals of the form $A+B$ left in $d$. If the rule $A+B \rightarrow c(\ldots)$ has a parent in $d$, then we replace the nonterminal $A+B$ in its rule’s rhs by $A$.

**Inverse of steps 2 and 1.** The derivation is transformed into a (binary) parse tree which is then unbinarized using the standard procedure.

5 Supertagging

We perform supertagging-based parsing with unlexical LCFRS. It consists of two phases.

1. Given a uni-lexical LCFRS $G_{\text{lex}}$ and sentence, a discriminative model predicts for each sentence position a sample of rules of $G_{\text{lex}}$. This
phase is called supertagging and can be considered as a sequence tagging task. The size of the sample for each position is a fixed number \( k \) which constitutes a hyperparameter of our approach.

(2) We construct a new grammar \( G'_{\text{lex}} \) from the rules predicted during supertagging. We replace the lexical symbol of each rule by the sentence position it was predicted for. Then we employ a usual chart-based parsing algorithm to parse the sequence of sentence positions with \( G'_{\text{lex}} \). As \( G'_{\text{lex}} \) has considerably fewer rules than \( G_{\text{lex}} \), this approach shifts a huge amount of work from parsing with grammars to predicting the rules.

6 Experiments

Implementation. The corpus lexicalization procedure and parsing was implemented as a part of Disco-Dop (van Cranenburgh et al., 2016), from which we could borrow the LCFRS extraction and parsing implementation. Moreover, we used the computation of evaluation scores in Disco-Dop.

The sequence tagger for the supertagging algorithm was implemented using the flair framework (Akbik et al., 2019). It features two kinds of word embeddings that we tested:

- (bert) the output of the four topmost layers of a pretrained bert model, which is fine-tuned during training,
- (bilstm) the concatenation of fasttext (Mikolov et al., 2018) (pretrained language-specific word embeddings), flair (Akbik et al., 2018) (pretrained language-specific bidirectional subword embeddings, fine-tuned during training) and learned embeddings of gold POS-tags.

On top of these embeddings, there are two layers of Bi-LSTMs (Hochreiter and Schmidhuber, 1997) and one linear layer. The sequence tagger is trained to predict the gold supertag via cross-entropy loss. More details with respect to hyperparameters for both models are shown in tbl. 1.

Data. Following (Coavoux and Cohen, 2019), we use three corpora for discontinuous constituent parsing in our evaluations: NeGra (Skut et al., 1998), Tiger (Brants et al., 2004), and a discontinuous version of the Penn treebank (Evang and Kallmeyer, 2011, DPTB). The corpora were split according to the usual standards into training, development and test sets.¹ During development, the lexicalization, tagging and parsing were mostly tested and optimized using NeGra. We binarized and markovized (with horizontal context \( h = 0 \), vertical context \( v = 1 \)) each training corpus before extracting the LCFRS and supertags. We extracted 3043 supertags from the training set of NeGra, 4108 from Tiger and 2968 from DPTB.

Baselines. We report labeled F1-scores, obtained from predicted and gold parse trees using Disco-Dop (using the usual parameters in proper.prm), for all constituents (\( F1 \)) and all discontinuous constituents (\( \text{Dis-F1} \)). Additionally to the scores, parse speed is reported in sentences per second (\( \text{sent/s} \)).

Our scores are compared to recent state-of-the-art parsers for discontinuous constituent trees in four categories:

- grammar-based parsers – that directly rely on an underlying (probabilistic) LCFRS,
- chart-based parsers – that share parsing algorithms with LCFRS, but lack an explicit set of rules,
- transition systems, and
- neural systems – all are other recent parsing approaches.

7 Results

Tbl. 2 shows some statistics of our parser on the development sets for different amounts of su-

¹We 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).

| Table 1: Hyperparameters for the sequence tagger. |
|--------------------------------------------------|
| Parameter             | Value (bilstm)            | Value (bert)            |
|-----------------------|---------------------------|-------------------------|
| embeddings            | fasttext, flair           | top 4 bert layers       |
| Bi-LSTM               | 2 layers, each 512 hidden states | no. of supertags        |
| linear layer          |                           |                         |
| dropout               | 0.05                      |                         |
| loss                  | cross entropy             |                         |
| optimizer             | Adam (\( \beta_1 = 0.9, \beta_2 = 0.999 \)) | 10⁻⁴                   |
| base lr               |                           |                         |
| lr scheduler          | reduce on plateau         |                         |
| batch size            | 32                        |                         |
| training epochs       | max. 200, or until lr < 10⁻⁷ | 10                     |
| k-best tags           |                           |                         |
pertags taken into consideration for each token (k). Specifically, we report the parsing speed (sent/s), the rate where the gold tag was among the k predicted tags per token (tag acc.) and parsing scores (labeled precision, recall and F1).

We see the parsing speed drops significantly where k > 10, but there are barely any gains in terms of parsing scores. As expected, the recall increases drastically with rising k. However, interestingly, the precision drops at first (as expected, but at a much smaller amount), but then also rises slowly. We found k = 10 to be a good parameter for the rest of our experiments.

Publications for supertaggers often omit parsing scores and only give accuracies of their predictions with respect to gold supertags. Tbl. 3 compares our supertagger to some recent publications. We include some results for CCGBank that we cannot compare to, but it shows how well supertagging performs for combinatory categorial grammars (CCG).

Tbl. 4 shows the parsing scores and speed of our final model on the test set compared to the scores reported in other recent publications for discontinuous constituent parsing. The experiments suggest that parsing using LCFRS can greatly benefit from supertagging, with respect to, both, speed and accuracy. This, however requires a strong discriminative classifier for the sequence tagger to predict useful rules.

Compared to other parsing approaches, we obtain results that on par with state-of-the-art parsing approaches, we would like to especially highlight the excellent scores for discontinuous constituents. This suggests that grammar-based approaches can keep up with other parsers, if they are appropriately combined with discriminative models. Recent publications of Corro, 2020 and (Stanojević and Steedman, 2020) obtain similar results using approaches that base on parsing with LCFRS, but lack of an underlying grammar and solely rely on neural network predictions.

8 Conclusion

We described an approach to utilize supertagging for parsing discontinuous constituent trees with LCFRS and implemented it. Compared to other parsers for the same grammar formalism, we achieve state of the art results, i.e. we are more accurate and also faster (cf. tbl. 4, Grammar-based systems). In contrast to previous parsers utilizing LCFRS, we can even keep up with other recent parsing approaches and achieve excellent results for discontinuous constituents (cf. tbl. 4, columns for Dis-F1).

Future Work. Disco-Dop currently only supports parsing sentences up to 128 words and, unfortunately, a bug was keeping us from increasing this limit. Resolving this issue should increase the scores slightly in Tiger and DPTB.

The sequence tagger’s hyperparameters need to be addressed in a proper parameter search. Now, they are mostly set by trial and error.

Acknowledgements

We thank Alex Ivliev for conducting early experiments during the development of our parser.

References

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. FLAIR: An easy-to-use framework for state-of-the-art NLP. In Proceedings of the 2019 Conference...
Table 3: Our results compared to other published supertaggers. Bladier et al., 2018 used a slightly different split of Tiger.

| Model                     | formalism | NeGra tags accuracy | Tiger tags accuracy | DPTB tags accuracy | CCGBank tags accuracy |
|---------------------------|-----------|---------------------|--------------------|--------------------|-----------------------|
| Bladier et al., 2018      | TAG       | – –                 | 3426 88.51         | – –                | – –                   |
| Kadari et al., 2018       | CCG       | – –                 | – –                | – –                | 1284 94.49            |
| Kasai et al., 2017        | TAG       | – –                 | – –                | 4727 89.71         | – –                   |
| ours (bilstm)             | LCFRS     | 3275 79.98          | 4614 84.28         | 4509 84.76         | – –                   |
| ours (bert)               | LCFRS     | 3275 86.93          | 4614 84.28         | 4509 90.78         | – –                   |
| Vaswani et al., 2016      | CCG       | – –                 | – –                | – –                | 1284 94.5             |

Table 4: Our results on test sets compared to other published constituent parsers. Van Cranenburgh et al., 2016 use another split for the Tiger corpus.

| Model                     | F1  | NeGra Dis-F1 | sent/s | Tiger Dis-F1 | sent/s | DPTB Dis-F1 | sent/s |
|---------------------------|-----|--------------|--------|--------------|--------|-------------|--------|
| **Grammar-based systems** |     |              |        |              |        |             |        |
| van Cranenburgh et al., 2016 | 76.8 | – 2 | 78.2 | – 1 | 87.0 | – < 1 |
| Gebhardt, 2020            | 81.7 | 43.5 | – | 77.7 | 40.7 | – | – |
| ours (bilstm)             | 82.79 | 52.93 | 53 | 81.82 | 54.58 | 39 | 89.07 | 63.36 | 47 |
| ours (bert)               | 89.01 | 70.09 | 30 | 86.81 | 66.00 | 24 | 92.87 | 72.73 | 36 |
| Versley, 2016             | – | – | – | 79.50 | – | – | – |
| **Chart-based systems**   |     |              |        |              |        |             |        |
| Corro, 2020 (w/o bert)    | 86.3 | 56.1 | – | 85.2 | 51.2 | – | 92.9 | 64.9 | – |
| Corro, 2020 (w/ bert)     | 91.6 | 66.1 | – | 90.0 | 62.1 | – | 94.8 | 68.9 | – |
| Stanojevic and Steedman, 2020 | 83.3 | 50.7 | – | 83.4 | 53.5 | – | 90.5 | 67.7 | – |
| **Transition systems**    |     |              |        |              |        |             |        |
| Coavoux and Cohen, 2019   | 84.0 | 54.0 | – | 87.6 | 52.5 | – | 91.4 | 70.9 | – |
| Coavoux et al., 2019      | 83.2 | 54.6 | – | 82.7 | 55.9 | 126 | 91.0 | 71.3 | 80 |
| **Neural systems**        |     |              |        |              |        |             |        |
| Fernández-G. and Gómez-R., 2020 | 86.1 | 59.9 | – | 86.3 | 60.7 | – | – | – | – |
| Vilares and Gómez-R., 2020 (bilstm) | 77.1 | 36.5 | 715 | 79.2 | 40.1 | 568 | 89.1 | 41.8 | 611 |
| Vilares and Gómez-R., 2020 (bert) | 84.2 | 46.9 | 81 | 84.7 | 51.6 | 80 | 91.7 | 49.1 | 80 |
Maximin Coavoux and Shay B. Cohen. 2019. Discontinuous constituent parsing with pointer networks. arXiv preprint arXiv:2002.01824.

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.

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

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. 10(1):136–163.

Rokia Kadari, Yu Zhang, Weinan Zhang, and Ting Liu. 2018. Ccg supertagging via bidirectional lstm-crff neural architecture. Neurocomputing, 283:31–37.

Jungo Kasaai, Robert Frank, R. Thomas McCoy, Owen Rambow, and Alexis Nasr. 2017. TAG Parsing with Neural Networks and Vector Representations of Supertags. In Conference on Empirical Methods in Natural Language Processing, pages 1712 – 1722, Copenhagen, Denmark.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In Proceedings of Formal Grammar, page 61.

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 HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advancements in pre-training distributed word representations. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018).

Richard Mörbitz and Thomas Ruprecht. 2020. Lexicalization of probabilistic linear context-free rewriting systems. In Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies, pages 98–104, Online. Association for Computational Linguistics.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richard 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 Secker, 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 10th European Summer School in Logic, Language and Information. Workshop on Recent Advances in Corpus Annotation.

Miloˇs Stanojevi ´c and Mark Steedman. 2020. Span-A. In this appendix we describe how the original advances in Corpus Annotation.

A.1 Unlexicalizing Derivations

In this appendix we describe how the original compositions of branching and monic rules are computed when applying the inverse of step 5 to some derivation $d$.

Branching rules. Let $r$ be a branching rule of the form $A \rightarrow (u_1, \ldots, u_s) (A_1, A_2)$ and $\sigma$ be the lexical symbol in $(u_1, \ldots, u_s)$.

- If $A_2$ is annotated with $\sim$ (i.e., its first component was removed during step 5), we replace $\sigma$ with $x_2^0$, and replace every occurrence of $x_2^1$ by $x_2^{i+1}$.
- Otherwise, $\sigma$ is removed from $(u_1, \ldots, u_s)$.

Moreover, if $r$ occurs as a successor of the right child of some other branching rule in $d$, then the nonterminals $A$ and $A_1$ are annotated as well.

- If $A_1$ and $A$ are annotated with $\sim$, then we replace $(u_1, \ldots, u_s)$ by $(x_1^0, u_1, \ldots, u_s)$.
- If $A_1$ is annotated with $\sim$ and $A$ with $+$, then we replace $(u_1, \ldots, u_s)$ by $(x_1^0 u_1, \ldots, u_s)$.

After that, we replace every occurrence of $x_1^i$ by $x_1^{i+1}$.

We recall that during step 5, every occurrence of a branching rule is processed up to two times: once for inserting a lexical symbol and potentially a second time for propagating a lexical symbol to its parent. Since both modifications affect the variables of different components, they can be undone independently from each other (as described in Section 4) or at the same time (as done here).

Monic rules. Let $A \rightarrow c(A_1)$ be a monic rule, $\sigma_1$ be the lexical symbol received from its parent, and $\sigma_2$ be the lexical symbol of $c$.

1. If the terminal of this rule was swapped during step 5 and $A_1$ is annotated with $\sim$, we insert $x_1^0$ before $x_1^1$ in $c$ and replace every occurrence of $x_1^i$ by $x_1^{i+1}$. Moreover, if $A$ is annotated with $\sim$, a component split is added after $x_1^i$.
2. Otherwise, we replace $\sigma_2$ in $c$ by $\sigma_1$. If $A_1$ is annotated with $\sim$, we insert $x_1^0$ after $\sigma_1$ in $c$ and replace every occurrence of $x_1^i$ by $x_1^{i+1}$. Moreover, if $A$ is annotated with $\sim$, a component split is added after $\sigma_1$.

A Supplementary details on constructions

A.1 Unlexicalizing Derivations

In this appendix we describe how the original compositions of branching and monic rules are computed when applying the inverse of step 5 to some derivation $d$.

Branching rules. Let $r$ be a branching rule of the form $A \rightarrow (u_1, \ldots, u_s) (A_1, A_2)$ and $\sigma$ be the lexical symbol in $(u_1, \ldots, u_s)$.