From partial neural graph-based LTAG parsing towards full parsing

Tatiana Bladier[*] BLADIER@PHIL.HHU.DE
Jakub Waszczuk[*] WASZCZUK@PHIL.HHU.DE
Laura Kallmeyer[*] KALLMEYER@PHIL.HHU.DE
Jörg Hendrik Janke[*] JOERG.JANKE@HHU.DE

[*]*Heinrich Heine University of Düsseldorf, Germany*

Abstract

In this paper, we extend recent approaches to Lexicalized Tree Adjoining Grammar (LTAG) parsing that combine supertagging with dependency parsing. In other words, we assign supertags (= unanchored elementary trees) to lexical items and we compute substitution/adjunction arcs between them. Kasai et al. (2017, 2018) jointly predict these structures with a neural graph-based parser. Predicting 1-best supertags and dependency arcs (as in Kasai et al. (2017, 2018)) however leads only to partial parsing due to incompatibilities between elementary trees and derivation trees. We therefore extend the approach described in Kasai et al. (2017, 2018) to n-best supertags and k-best dependency arcs and combine it with a subsequent A*-parsing step that extends the TAG parser from Waszczuk (2017). We show that this architecture allows for efficient full TAG parsing while being sufficiently accurate. We test our architecture on an LTAG extracted from the French Treebank (FTB).

1. Introduction

Lexicalized Tree-Adjoining Grammar (LTAG; Joshi and Schabes (1997)) is a linguistically motivated grammar formalism which supports an *extended domain of locality* (EDL). The building blocks in LTAG – called *elementary trees* – capture a domain of locality which is large enough to cover co-occurrence relationships between a lexical item (the *anchor* of the elementary tree) and the nodes it posits syntactic or semantic constraints on (Carroll et al., 1999). LTAG elementary trees are capable of expressing linguistic generalizations (for example, the *wh*-movement or multi-word-expressions) which are not captured by typical statistical parsers based on context-free grammars or dependency parsing. The linguistically rich analyses of LTAG can be used for the downstream tasks like semantic role labeling or semantic parsing (Liu and Sarkar, 2007; Kallmeyer and Osswald, 2013).

Several parsing approaches have been proposed for LTAG, including symbolic and statistical ones (Joshi and Srinivas, 1994; Bäcker and Harbusch, 2002; Chiang, 2000; Sarkar, 2000; Kallmeyer and Satta, 2009). Recent advances in LTAG parsing include graph-based and transition-based architectures based on Recurrent Neural Networks (RNN) (Kasai et al., 2017, 2018). While many of the developed approaches for LTAG parsing are computationally costly due to the large number of elementary trees per lexical item, previous work (Bangalore and Joshi, 1999; Sarkar, 2007) has shown that LTAG parsing can be facilitated through an intermediate step of *supertagging.* Supertagging is the task of assigning a supertag, i.e., an LTAG elementary tree template, to each word in a given sentence. This step can be seen as “almost parsing”, since it performs a considerable amount of syntactic disambiguation before applying a computationally more costly algorithm for actual parsing. For example, linguistic properties of supertags in LTAG grammars proved to enhance the performance of transition-based LTAG parsers (Chung et al., 2016).

Kasai et al. (2017) proposed to extend the supertagging step of LTAG parsing by jointly predicting the supertags and arcs of the LTAG derivation trees using a deep learning architecture. The idea is to not only predict the elementary tree templates but also predict how to combine them (hence, the dependency arcs) to form a derived tree. Predicting dependency relations along with the supertags should facilitate the subsequent parsing step even further. The authors claim their approach to be sufficient for LTAG parsing. However, in this paper we show that predicting only 1-best supertags and dependency relations between the supertags is not sufficient for the full LTAG parsing step (i.e. step in which the derived trees are predicted) but only allows for partial parsing. The reason is that the system of Kasai et al. (2017) outputs only sequences of the one most probable supertag per word in a sentence along with the single most probable dependency arc for this supertag. The supertags and dependency arcs must be mutually compatible in order to build the derived tree of a sentence, which means that in many cases predicted 1-best supertags and dependency arcs cannot be combined to a complete derived tree.

In the present paper we extend the architecture proposed by Kasai et al. (2018) by combining it with the bottom-up A* parsing system ParTAGe developed by Waszczuk (2017) in order to allow for the full parsing step. Our parsing architecture uses the output from the parser by Kasai et al. (2018) which is fed into ParTAGe in the subsequent step. For this, we changed the architecture developed by Kasai et al. (2018) to predict n-best supertags and k-best dependency arcs (for some arbitrary n and k). This output is then used as the input to the A* parser which computes the most probable combination of supertags and arcs to predict a full derived tree. The original architecture of ParTAGe, based on weighted deduction rules (see e.g. Nederhof (2003)), was changed in order to take n-best supertags and k-best dependency arcs as input, since the parser no longer works with the whole extracted grammar, but has to deal with the supertags and arcs sentence-wise.

Our approach to LTAG parsing takes up on the idea of imposing constraints on the available dependencies between the LTAG elementary trees for a more efficient parsing, similar to the hypothesis stated in Carroll et al. (1999). In section 5 we show that lowering the number of potential dependencies between the LTAG supertags improves the speed for parsing of longer sentences (i.e. sentences with 40 tokens or longer) and reduces the size of the hypergraph created while processing the parsing chart items.

For our experiments, we extract an LTAG for French along the lines of Bladier et al. (2018c) from the French Treebank (FTB; Abeillé et al. (2003)). This grammar contains more than 4500 distinct supertags, half of which appear only once in the corpus. The grammar we use was extracted from 21550 sentences in the current version of the FTB (version 1.0 2016) using the top-down LTAG extraction algorithm proposed by Xia (1999). A peculiarity of this French LTAG is that it only requires *sister-adjunction* and not regular TAG adjunction (i.e., it does not contain TAG's standard auxiliary trees where one of the frontier nodes is marked as a *foot node*). The reason for this decision is the fact that the FTB trees have a flat structure and do not allow to extract regular TAG auxiliary trees.

In this paper we show that sufficiently high numbers n of supertags and k of dependency arcs allow for full parsing for every sentence in the FTB. We also show that our architecture achieves state of the art labeled EVALB F1 score results of 84.36 % on parsing with the test set of the SPMRL (2013) version of the French Treebank (Seddah et al., 2013). The approach to LTAG parsing presented in this paper can be extended to different kinds of LTAGs and other grammars consisting of sets of elementary tree templates, such as for example formalized Role and Reference Grammar (Osswald and Kallmeyer, 2018).

The paper is structured as follows: Section 2 gives a brief overview of the LTAG theory; section 3 provides a general overview of our architecture and explains the architecture proposed by Kasai et al. (2018) and our modifications to it. Section 4 describes in detail the changes we made upon the architecture of ParTAGe (Waszczuk, 2017). We present our experiments and the extracted French LTAG in section 5. Section 6 provides some error analysis, and we conclude with plans for future work in section 8.

2. Lexicalized Tree-Adjoining Grammar

Lexicalized Tree-Adjoining Grammar is a linguistically motivated grammar formalism which supports an *extended domain of locality* (Joshi and Schabes, 1997). A TAG consists of a finite set of elementary trees, which can be combined into larger trees via the operations *substitution* for filling the argument slots and *adjunction* for modifier insertion (see an example in Fig. 1a and 1b). In case of an adjunction an internal node in a tree is replaced with an *auxiliary tree*, which has a special leaf node marked with an asterisk (called *foot node*). When adjoining an auxiliary tree to a node μ, the subtree with root μ in the old tree is put below the foot node of the auxiliary tree in the resulting tree. Non-auxiliary elementary trees in LTAG are called *initial trees*.

Each elementary tree contains (at least) one leaf labeled with a lexical item, its lexical *anchor*. For a given derivation, the derivation tree (see Fig. 1c) describes the way the elementary trees are combined: It contains a node for each elementary tree that has been used and an edge for each substitution or adjunction that has been performed. Edges are labeled with Gorn addresses of the target nodes of the respective operation. Because of the property of *extended domain of locality* in LTAG, it is possible to state linguistic dependencies between nodes which are further apart in the final derived tree. For example, the relation between a topicalized constituent and its governor can be stated locally in a single elementary tree.

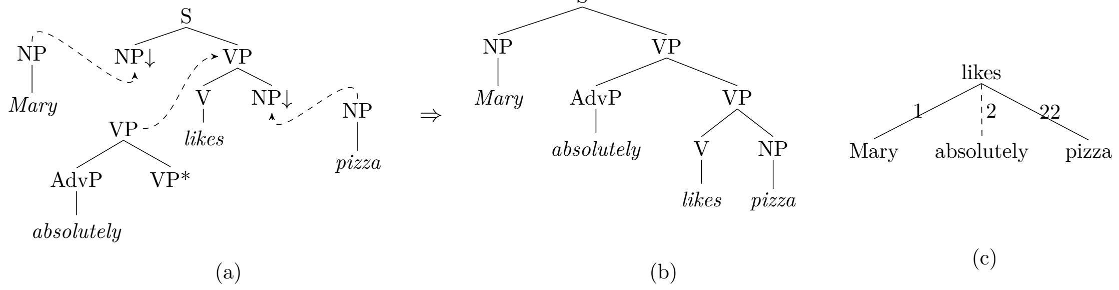


Figure 1: Elementary tree operations (a), a derived tree (b), and a derivation tree (c) in LTAG.

Besides regular adjunction as in Fig. 1a, a simpler type of adjunction for adding modifiers has also been proposed in the literature, namely *sister-adjunction* (Rambow et al., 1995), see Fig. 2 for an example. A modifier tree can be added by sister adjunction if its root category matches the category of the target node. In this case, it adds a new daughter to this node. The result of sister-adjunction are flatter trees, compared to regular adjunction, since no extra nodes are being added to the original tree. Regular adjunction is more powerful concerning generative capacity since it can add two substrings in different places (the spans to the left and the right of the foot node), while sister adjunction adds only one substring. LTAGs using only substitution and sister-adjunction are weakly equivalent to CFGs while LTAG in general can generate a larger class of string languages.

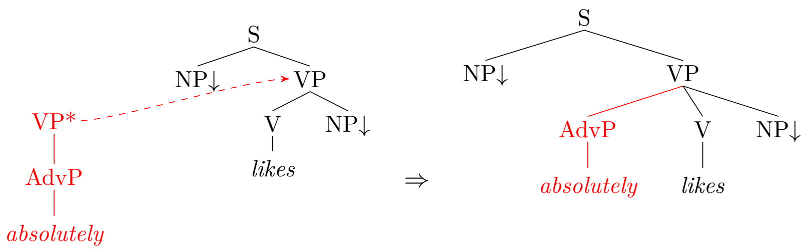


Figure 2: Sister-adjunction operation and the resulting tree. A sister-adjoining elementary tree is marked red.

LTAG grammars for natural languages can be written manually (English XTAG; XTAG Research Group (1998)), also using a metagrammar (French TAG; Crabbé (2005)) or they can be induced automatically from a treebank using top-down (Xia, 1999) or bottom-up (Chen et al., 2002) approaches. An LTAG typically includes several thousands of elementary tree templates, i.e. of unanchored elementary trees (around 4000 on average). About half of them appear only once for the used treebank. The large number of such supertags makes the task of predicting the correct supertag sequence difficult and leads to a large difference in parsing performance with gold and predicted supertags (Chung et al., 2016).

3. LTAG parsing architecture as a pipeline of supertagging, dependency parsing, and A* parsing

LTAG parsing systems based on supertagging consist of a two-step pipeline including a supertagger and a subsequent actual parser (Bangalore and Joshi, 1999; Sarkar, 2007). Supertagging is a *sequence labeling task* which takes as input a sequence of tokens $s_{input} = (w_1, \ldots, w_n)$ for each sentence and outputs a sequence of supertags $s_{output} = (t_1, \ldots, t_n)$ for this sentence. This sequence of supertags is used as input for the subsequent actual parsing step, during which the supertags are combined to a complete derived LTAG tree. In the present paper we use a similar pipeline consisting of the supertagger and dependency parser developed by Kasai et al. (2018) and an A* parser developed by Waszczuk (2017) for the actual parsing step. Fig. 3 shows the pipeline architecture of our parser. Note that we modified both the supertagging architecture provided by Kasai et al. (2018) as well as the A* parser by Waszczuk (2017).

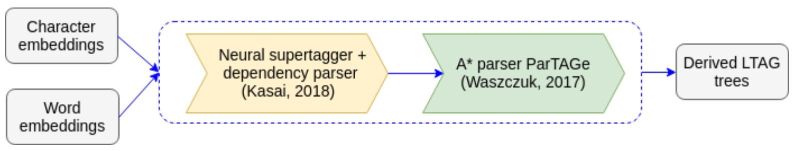

Character embeddings
Word embeddings
Neural supertagger + dependency parser (Kasai, 2018)
A* parser ParTAGe (Waszczuk, 2017)
Derived LTAG trees

Figure 3: Pipeline neural LTAG parsing architecture.

The term "dependency parser" might be misleading here. The goal of this component is to predict LTAG derivation trees, which formally are dependency structures (see the example in Fig. 1c). It means that the parser does not yield syntactic dependencies in the standard sense but edges for adjunctions and substitutions relating those words whose supertags get combined. Fig. 4 gives an example of what the output of the supertagger and dependency parser should look like.

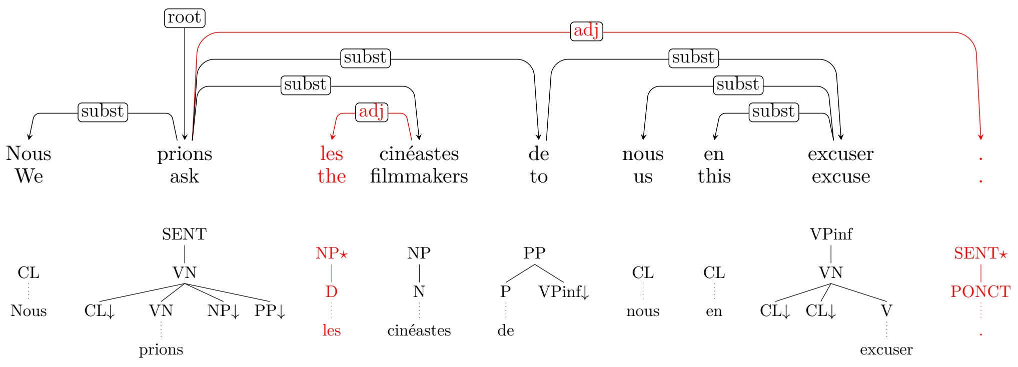


Figure 4: A sentence with supertags and labeled dependency arcs. Auxiliary trees are marked red to distinguish them from initial trees.

Since supertagging is a sequence labeling problem, a neural network is a good choice for such a task, since Recurrent Neural Networks (RNN) with their variants such as Long-Short Term Memory LSTM (Lewis et al., 2016) or Bi-LSTM plus Conditional Random Fields BiLSTM-CRF models (Le and Haralambous, 2019; Ma and Hovy, 2016) have been proven to deliver state of the art performance for sequence labeling tasks in the recent years. The graph-based supertagger and dependency parser developed by Kasai et al. (2018) uses a BiLSTM-based architecture with highway connections between the layers. The highway connections use gating units to regulate the flow of information through the neural network in order to reduce the risk of overfitting and improve the results Srivastava et al. (2015). The neural network model takes as features character embeddings and pre-trained word embeddings and jointly predicts POS tags, dependency arcs, dependency labels, and supertags. The supertagger and dependency parser is based on the graph-based parsing archictecture with deep biaffine attention proposed by Dozat and Manning (2016) (see Fig. 5).

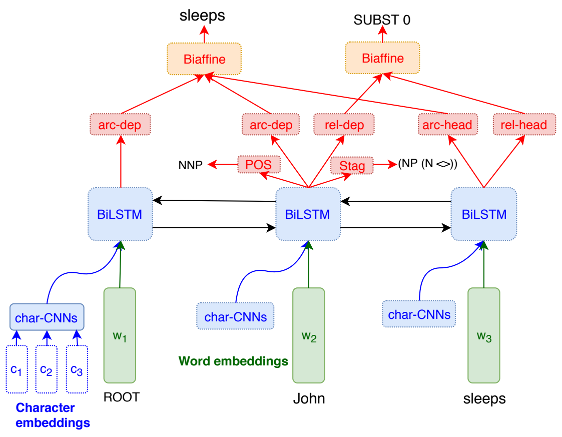


Figure 5: Neural dependency parser for jointly predicting supertags, dependency relations between supertags, POS-tags, and dependency arc labels (Kasai et al., 2018).

Kasai's (2018) architecture predicts 1-best supertag for every word in a sentence and 1-best dependency arc. This is not yet full parsing, since the supertags are not yet combined to a derived tree. The supertags and arcs have to be mutually compatible in order to produce a full derived tree, which is, however, not always the case. The left column in Fig. 6 shows an example of a sequence of predicted supertags which cannot be combined to form a full derived tree, because the supertag for the word *"prions"* (line 2) is lacking the substitution slot for the supertags in lines 5 to 9.

We used Kasai's (2018) parser for our extracted French LTAG and used pre-trained 200-dimensional word embeddings for French (Fauconnier, 2015). We use the n-best output of Kasai's architecture as the input for the ParTAGe parser. Fig. 7 shows a strongly simplified example of an output of Kasai's architecture displaying token number, tokens, dependency arcs with probabilities, and supertags with probabilities (the probabilities are given in float format after the colon). For example, the verb '*eats*' is assigned a dependency arc to the root node (id 0) with probability 1.0 and the two supertags (SENT(NP)(VP(V ◇))) and (SENT(NP)(VP(V ◇)(NP))) with probabilities 0.6 and 0.4 respectively. Among the given n-best supertags and k-best arcs in the input data ParTAGe picks the most probable combination leading to a full derived tree (as represented in the output section in the lower part of Fig. 7).

The information about the dependency arcs on the one hand facilitates the actual parsing leading to a strong decrease in possible parses – as compared to supertagging with no information about the dependency arcs. But on the other hand it can also be a source of parsing errors. For example, it can be the case that all supertags are predicted correctly, but cannot be combined due to erroneously

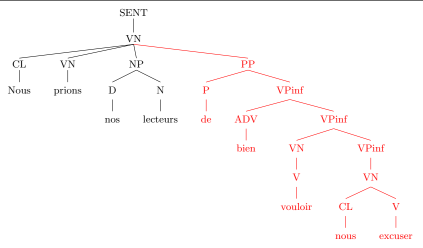
		1-best output (Kasai's (2018) parser)		A* parser output (10 best)	
line	token	arc	LTAG supertag	arc	LTAG supertag
1	Nous	2	(CL ⋄)	2	(CL ⋄)
2	prions	0	(SENT (VN (CL) (V ⋄)) (NP))	0	(SENT (VN (CL) (V ⋄)) (NP) (PP))
3	nos	4	(NP* (D ⋄))	4	(NP* (D ⋄))
4	lecteurs	2	(NP (N ⋄))	2	(NP (N ⋄))
5	de	2	(PP (P ⋄) (VPinf))	2	(PP (P ⋄) (VPinf))
6	bien	7	(VPinf* (ADV ⋄))	7	(VPinf* (ADV ⋄))
7	vouloir	5	(VPinf (VN (V ⋄)) (VPinf))	5	(VPinf (VN (V ⋄)) (VPinf))
8	nous	9	(CL ⋄)	9	(CL ⋄)
9	excuser	7	(VPinf (VN (CL) (V ⋄)))	7	(VPinf (VN (CL) (V ⋄)))
		NO PARSE		 SENT VN CL VN NP PP Nous prions D N P VPinf nos lecteurs de ADV VPinf bien VN VPinf V VN vouloir CL V nous excuser	

Figure 6: The left column shows the output from Kasai et al. (2018) architecture, and the right column provides the output from our model for the sentence *We ask our readers to kindly forgive us.* The 1-best output does not guarantee a full parse. Due to an error in supertag prediction in the highlighted line 2, the supertags in lines 5 to 9 do not have an attachment site and cannot form a full derived tree. Our parsing model solves this problem by using 10-best supertags and arcs.

predicted dependency arcs. In section 5 we show that for French LTAG predicting only 1-best supertag and 1-best arc leads to a fully derived tree in only around 50 % of sentences.

Input

```
1      John     2:1.0              (NP (N <>)):1.0
2      eats     0:1.0              (SENT (NP) (VP (V <>))):0.6
                                   (SENT (NP) (VP (V <>) (NP))):0.4
3      an       4:0.5|1:0.5        (NP* (D <>)):1.0
4      apple    2:0.5|0:0.5        (NP (N <>)):1.0
```

Output

```
1      John     2                  (NP (N <>))
2      eats     0                  (SENT (NP) (VP (V <>) (NP)))
3      an       4                  (NP* (D <>))
4      apple    2                  (NP (N <>))

(SENT (NP (N John)) (VP (V eats) (NP (D an) (N apple))))
```

Figure 7: Input and output (simplified) of the ParTAGe parser.

We changed the architecture of Kasai et al. (2018) by making it predict n-best supertags and k-best dependency arcs ($n, k \geq 1$) and modified the A* parsing architecture developed by Waszczuk (2017) to combine the supertags according to information about dependency arcs to produce a full derived tree. We experimented with several values for n and k, and it turned out that $n = 10$ and $k = 10$ were the best choices on the development data in order to produce full parses for all sentences (see a summary of our experiments in Fig. 8).

k-best stags (FTB, dev set)

n-best arcs	1	2	4	6	8	10
1	968	696	509	448	410	385
5	841	296	53	14	5	2
10	838	276	38	7	2	0

Figure 8: Number of sentences in FTB-dev with `no parse` as a function of the number of `n`-best supertags (columns) and `k`-best arcs (rows). The values `n` = 10 and `k` = 10 are sufficiently high to obtain full parses for every sentence in the dev set of the French Treebank. These values can vary depending on the used treebank and the size of the extracted LTAG grammar.

We describe the A* parsing step and the changes which have been made upon the original ParTAGe architecture as described in Waszczuk (2017) in more detail in the next section.

4. A* TAG Parser

Our point of departure is ParTAGe, an A* bottom-up, left-to-right LTAG parser introduced in Waszczuk et al. (2016b); Waszczuk (2017). The parser relies on the A* algorithm, which allows to find a most probable derivation – based on the underlying set of *weighted deduction rules* (Shieber et al., 1995; Nederhof, 2003) – without exploring the entire parsing chart/hypergraph (Klein and Manning, 2001). This is possible provided that an appropriate *heuristic* function is defined, which serves to estimate the cost of parsing the remaining part of the input sentence at any given point of the parsing process. The use of the heuristic and the A* algorithm significantly improves the parser's efficiency in comparison with the corresponding, purely symbolic parser, which requires creating the entire parsing hypergraph before a most probable derivation can be extracted.

ParTAGe provides support for weighted TAGs, i.e., TAGs where a non-negative weight is assigned to each elementary tree (ET). Such a weighting scheme is suboptimal in that it does not allow to benefit from the potential knowledge about the *bilexical dependencies* (i.e. asymmetrical syntactic relations between head tokens and their dependents). The early formalization of probabilistic TAGs admits their importance – each composition of two (lexicalized) ETs, be it via adjunction or substitution, incurs the corresponding probability cost (Resnik, 1992). In the context of CCGs, extending the supertagging-based architecture of Lewis and Steedman (2014); Lewis et al. (2016) with bilexical probabilities leads to significant improvements in terms of the parsing results (Yoshikawa et al., 2017).

In this work, we extend ParTAGe[1] so as to account for the bilexical affinities within the context of lexicalized TAGs, which allows to apply it to the output of the neural LTAG parser (Kasai et al., 2018).

This section is structured as follows. Sec. 4.1 and Sec. 4.2 describe the properties of the TAG grammar required by the revised parser. In Sec. 4.3, the link between the output of the neural parser and the A* parser's input is established. The subsequent sections describe the internals of the parser. In particular, in Sec. 4.12 the parser's deduction rules are specified, and in Sec. 4.13 the A* heuristic tailored to the adopted grammar representation is defined.

4.1 Grammar restrictions

In the new version of ParTAGe, we adopt additional restrictions on the form of the grammar:

- The grammar must be lexicalized, i.e., each ET must contain some terminals in its leaves.
- More specifically, each ET has to have exactly one terminal. Given an ET t, we denote this terminal as $term(t)$.

1. The code of the revised parser can be found at https://github.com/kawu/partage.

The point of this limitation is to adapt the parser to handle dependency relations which hold between terminals (tokens) in the input sentence. In this parsing architecture, tokens and terminals are synonymous and interchangeable.

4.2 Weighting scheme

In the extended version of ParTAGe, both ETs and dependency links are weighted. More precisely:

- A non-negative weight $\omega(t) \in \mathbb{R}_{\geq 0}$ is assigned to each ET t in the grammar.
- A non-negative weight $\omega(x, y) \in \mathbb{R}_{\geq 0}$ is assigned to each pair x, y of grammar terminals, which represents the cost of making y the head of x. This cost applies each time substitution, adjunction, or sister adjunction is performed.

The weight of a particular derivation is defined as the sum of the weights of the participating ETs, as in Waszczuk (2017), plus the sum of the weights of the arcs in the derivation tree, i.e., the weights incurred by the ET combinations. Derivations with lower weights are preferable to those with higher weights. Parsing thus boils down to finding a lowest-weight derivation among all the derivations that can be constructed based on the underlying grammar.

4.3 Interface with supertagging and dependency parsing

As described in Sec. 3, the steps preceding A* parsing include (neural) supertagging and dependency parsing. This means that, for each word in the input sentence, the probability distributions of (i) the potential supertags, and (ii) the potential dependency heads are provided. These can be easily transformed to the weighting scheme required by the parser by taking the negative logarithm of the individual probability values given on input. Figure 9 shows the weighted input grammar presented in Figure 7 after applying this transformation.

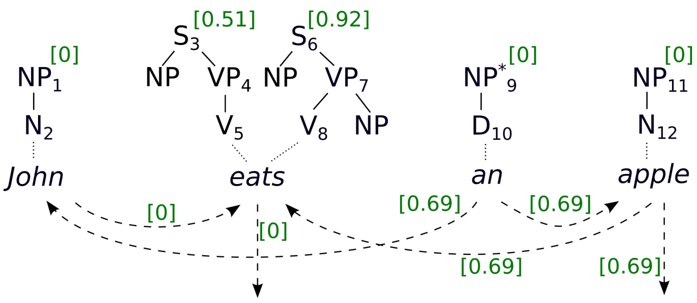


Figure 9: Input grammar from Figure 7 after converting probabilities to weights (enclosed between square brackets and marked in green). Internal nodes are additionally decorated with a unique IDs.

4.3.1 Input

We define the input sentence as a sequence $(s_i)_{i=1}^{n}$, where n is the length of the sentence and each s_i is a distinct terminal symbol (token). Each terminal must be considered as distinct because, even if the same word form is used at two positions of the input sentence, the two positions will receive two distinct distributions of supertags and dependency heads.

4.4 Grammar representation

ParTAGe adopts a two-layered grammar representation in which (a) the set of the grammar ETs is first transformed to the equivalent directed acyclic graph (DAG), and (b) an automaton-based representation is then used to compress the traversals of the individual nodes and their children in the DAG (from left to right).

In this work, we only assume the first layer of this representation, i.e., the grammar DAG. The traversals of the individual nodes and their children are represented explicitly via dotted rules. For simplicity, we assume no subtree sharing (Waszczuk et al., 2016a) in the grammar DAG in our formalization.[2]

We define the following auxiliary functions:

- $\ell(N)$ – the label (non-terminal, terminal, or ϵ) assigned to node N
- $\mathit{root}(N)$ – predicate which tells if N is a root node
- $\mathit{leaf}(N)$ – predicate which tells if N is a leaf node
- $\mathit{foot}(N)$ – predicate which tells if N is a foot node
- $\mathit{sister}(N)$ – predicate which tells if N is the root of a sister adjunction tree

4.5 Architecture

ParTAGe is based on *weighted deduction rules* (Shieber et al., 1995; Nederhof, 2003), which serve to infer *chart items*. Each chart item is a pair (x, r), where x is a *configuration* and r is a *span*. Each configuration x represents a position in the traversal of the corresponding ET t_x, and each span r represents a fragment of the input sentence. Informally, (x, r) asserts that the already traversed part of t_x can be matched against the words in r.

Additionally, to each item (x, r) a *weight* is assigned, which is also calculated via deduction rules and which represents the cumulative weight of the already traversed part of t_x. We get back to the topic of weights in Sec. 4.9.

As mentioned before, ParTAGe is an A* parser, which means that apart from the weights calculated via the deduction rules, an estimate of the weight remaining to parse the entire input sentence has to be determined for each chart item (see Sec. 4.13). This allows the parser to explore more plausible derivations first and find an optimal parse without creating the entire chart.

4.6 Configuration

A parsing configuration x represents a position in the traversal of the corresponding ET t_x. It takes the form of a *dotted rule*[3] $N \to \alpha \bullet \beta$, where N is a non-leaf node of an ET and $\alpha\beta$ is the sequence of N's children nodes (from left to right). The interpretation of the rule $N \to \alpha \bullet \beta$ is that the nodes in α (on the left of $\bullet$) are already parsed, i.e., matched against some input words, while the nodes in β still need to be matched.

We also use N to denote a parsing configuration where all the children nodes α have been matched against input. While N can be seen as syntactic sugar for $N \to \alpha\bullet$ (for some α), the parser makes a distinction between the two types of parsing configurations and it includes a rule which explicitly transforms $N \to \alpha\bullet$ to N.

4.7 Span

Let $\mathit{pos}(n) = \{0, \ldots, n\}$ be the set of positions between the words of the input sentence. A *span* is a 4-tuple (i, j, k, l) where $i, l \in \mathit{pos}(n)$ and $j, k \in \mathit{pos}(n) \cup \{-\}$. If $j, k \neq -$, $i \leq j \leq k \leq l$. Otherwise, $i \leq l$. We also write (i, l) to denote $(i, -, -, l)$. The interpretation of the span r depends on the parsing configuration x accompanying r within a chart item.

2. However, we used subtree sharing among the ETs attached to the same input position in our implementation and experiments.
3. Not to be confused with a dotted rewriting rule. A dotted rule in our architecture represents a point in the traversal of a fragment of an ET.

4.8 Chart item

The presence of the item $\eta = (x, (i, j, k, l))$ in the chart asserts that:

- If $j = k = -$, then x is matched against all the input words between positions i and l.
- Otherwise, x corresponds to an auxiliary ET t_x with a foot node. In this case, t_x's foot is matched against the span (j, k), the processed part of t_x on the left of the foot is matched against (i, j), and the processed part of t_x on the right of the foot is matched against (k, l).

If x is a dotted rule, we call η *active*. Otherwise, if x is a node of an ET, we call it *passive*.

4.9 Weight pair

To each item $\eta = (x, (i, j, k, l))$ in the chart a pair of weights (w, w') is assigned,[4] where w is the *inside weight*, i.e., the weight of the inside derivation of η, and w' is the *prediction weight*, i.e., the total weight of the partial derivations used for prediction in η's inside derivation.

Both w and w' are calculated directly via deduction rules. The inside weight w corresponds to the weight of the part of the derivation that is already determined – each full derivation based on η will have to contain it as its part. In our deduction system, the prediction weight w' corresponds to the cost of scanning the words in the gap (j, k) while providing the non-terminal necessary to match the foot of the auxiliary ET t_x. The purpose of w' is to facilitate the calculation of the A* heuristic (see Sec. 4.13), which serves to estimate the *outside weight*, i.e., the weight remaining to parse the entire input sentence. While the words in the gap are accounted for in the prediction weight w', the heuristic has to additionally account for the words on the left of i and on the right of l.

4.10 Scanning cost

To estimate the cost of parsing the remaining part of the sentence, we assume that each word outside of the current item's span will be analyzed with the lowest-weight ET and that it will get assigned the lowest-weight dependency head. This estimation strategy guarantees that the resulting A* heuristic is admissible, i.e., it never overestimates the cost of parsing the remaining words. We thus define, for a given token x, the lower-bound cost $\mathcal{C}(x)$ as:

$$\mathcal{C}(x) = \min\{\omega(t) : t \in T, \mathit{term}(t) = x\} \;+\; \min\{\omega(x, y) : y \in s\} \tag{1}$$

where T is the set of grammar ETs, restricted to supertagging results. The lower-bound cost of parsing the remaining set of tokens X can be then represented as:

$$\mathcal{C}(X) = \sum\nolimits_{x \in X} \mathcal{C}(x) \tag{2}$$

4.11 Amortized weight

Given a chart item $\eta = (x, r)$, we define the *amortized weight* $A(x)$ of x as:

$$A(x) = \omega(t_x) + \omega(t_x, \cdot) - \mathcal{C}(\mathit{sup}(x)),$$

where $\omega(t_x)$ is the weight of the ET t_x corresponding to x, $\omega(t_x, \cdot) = \min\{\omega(\mathit{term}(t_x), y) : y \in s\}$ is the minimal cost of attaching $\mathit{term}(t_x)$ as a dependent to another token in the input sentence, and $\mathcal{C}(\mathit{sup}(x))$ is the cost of scanning the terminals in t_x that remain to be matched ($\mathit{sup}(x)$). In practice, $\mathit{sup}(x)$ is either empty (if t_x's anchor is in the already traversed part of the tree) or contains the t_x's single anchor (we assume that each ET contains precisely one terminal, see Sec. 4.1).

4. We slightly diverge from the terminology used in Nederhof (2003), whose *inner* weight roughly corresponds to our *inside* weight, and *forward* weight to the sum of our *inside* and *prediction* weights.

Intuitively, $A(x)$ can be understood as the weight of the already parsed part of t_x, augmented with the minimal dependency weight of attaching t_x to another tree. Both $\omega(t_x)$ and (at least) $\omega(t_x, \cdot)$ will have to be accounted for in the total weight of any full derivation based on η. $\mathcal{C}(sup(x))$, on the other hand, gets discounted because the anchor $term(t_x)$ may still be outside of the η's item span (i.e., $sup(x) = \{term(t_x)\}$) and we assume the minimal scanning cost for each remaining token in the calculation of the heuristic (see Sec. 4.13 below).

4.12 Deduction rules

Table 1 shows the deduction rules of the extended version of ParTAGe, simplified in comparison with the rules presented previously (Waszczuk et al., 2017) in that no automaton-based grammar compression is assumed, and extended with two new rules (`SA` for handling sister adjunction and `EM` for empty terminals) as well as support for bilexical dependency weights. `AX` (axiom) posits that each subtree can be matched starting from any non-final position in the sentence, which corresponds to the bottom-up nature of the parser. `SC` and `EM` handle scanning input and empty terminals, respectively, while `DE` (deactivate) handles the situation where a full ET subtree has been matched. `PS` is responsible for combining two adjacent fragments of the same ET. `SU` models regular substitution, i.e., matching the leaf node of an ET with another, fully matched ET. `FA` and `RA` both model regular adjunction: `FA` performs predictions in order to identify the spans over which adjunction can be potentially performed, while `RA` performs the actual adjunction, i.e., attaching the auxiliary tree to an internal node of another tree. Finally, `SA` models sister adjunction, where a fully matched sister ET is attached to a non-leaf node of another tree.

AX:	$\dfrac{}{(0,0) : (N{\to}\bullet\alpha,(i,i))}$	$i{\in}\{0,\dots,n{-}1\}$ $N{\to}\alpha$ is a rule
SC:	$\dfrac{(w,w') : (N{\to}\alpha\bullet M\beta,(i,j,k,l))}{(w,w') : (N{\to}\alpha M\bullet\beta,(i,j,k,l{+}1))}$	$\ell(M){=}s_{l+1}$
EM:	$\dfrac{(w,w') : (N{\to}\alpha\bullet M\beta,(i,j,k,l))}{(w,w') : (N{\to}\alpha M\bullet\beta,(i,j,k,l))}$	$\ell(M){=}\epsilon$
DE:	$\dfrac{(w,w') : (N{\to}\alpha\bullet,(i,j,k,l))}{(w,w') : (N,(i,j,k,l))}$	
PS:	$\dfrac{(w_1,w_1') : (N{\to}\alpha\bullet M\beta,(i,j,k,l)) \quad (w_2,w_2') : (M,(l,j',k',l'))}{(w_1{+}w_2,w_1'{+}w_2') : (N{\to}\alpha M\bullet\beta,(i,j{\oplus}j',k{\oplus}k',l'))}$	
SU:	$\dfrac{(w_1,w_1') : (N{\to}\alpha\bullet M\beta,(i,j,k,l)) \quad (w_2,0) : (R,(l,l'))}{(w_1{+}w_2{+}\omega(R,N),w_1') : (N{\to}\alpha M\bullet\beta,(i,j,k,l'))}$	$leaf(M) \wedge \neg foot(M)$ $root(R) \wedge \neg sister(R)$ $\ell(M){=}\ell(R)$
FA:	$\dfrac{(w_1,0) : (N{\to}\alpha\bullet F\beta,(i,l)) \quad (w_2,w_2') : (M,(l,j',k',l'))}{(w_1,w_2{+}w_2'{+}A(M)) : (N{\to}\alpha F\bullet\beta,(i,l,l',l'))}$	$foot(F) \wedge \ell(M){=}\ell(F)$ $root(M) \implies (j',k'){=}(-,-)$ $\neg sister(M)$
RA:	$\dfrac{(w_1,w_1') : (R,(i,j,k,l)) \quad (w_2,w_2') : (M,(j,j',k',k))}{(w_1{+}w_2{+}\omega(R,M),w_2') : (M,(i,j',k',l))}$	$root(R) \wedge \ell(R){=}\ell(M)$ $root(M) \implies (j',k'){=}(-,-)$ $\neg sister(M)$
SA:	$\dfrac{(w_1,w_1') : (N{\to}\alpha\bullet\beta,(i,j,k,l)) \quad (w_2,0) : (M,(l,l'))}{(w_1{+}w_2{+}\omega(M,N),w_1') : (N{\to}\alpha\bullet\beta,(i,j,k,l'))}$	$\ell(M){=}\ell(N) \wedge sister(M)$

Table 1: Deduction rules of the revised parser, where (in `PS`) $i{\oplus}j$ is equal to i if $j = -$ and j otherwise. To simplify notation, we write $\omega(M, N)$ to denote $\omega(t_M)$ plus the weight $\omega(term(t_M), term(t_N))$ of the dependency arc combining the corresponding ETs.

Note that the weight $\omega(t_x)$ of an ET t_x is transferred to the inside weight only when t_x gets attached as dependent to another tree (via one of the `SU`, `RA`, or `SA` rules). Firstly, while it could be more intuitive to transfer $\omega(t_x)$ to the inside weight already in the axiom rule (`AX`), the current solution composes better with subtree sharing.[5] Secondly, the situation where an ET t_x becomes the root of the derivation has to be handled as a special case at the end of parsing. More precisely, we have to make sure that both $\omega(t_x)$ and the weight of t_x becoming a dependent of the dummy root position get transferred to the inside weight of the corresponding chart item. For simplicity, these details are not reflected in the deduction rules in Tab. 1.

4.13 A* heuristic

Given a chart item $\eta = (x, r)$ such that $r = (i, j, k, l)$ and the corresponding weight pair (w, w'), the heuristic h (which provides a lower-bound estimate on the cost of parsing the remaining part of the sentence) is defined as follows:

$$h(x, r) = A(x) + \mathcal{C}(\mathit{rest}(r)) + w', \tag{3}$$

where $\mathit{rest}(r)$ is the set of tokens remaining on the left and right of i and l, respectively, and $\mathcal{C}(\mathit{rest}(r))$ is the total minimal cost of scanning each remaining word in $\mathit{rest}(r)$. Note that the minimal possible cost of scanning the words in the gap (j, k) (provided that $j, k \neq -$) is accounted for in w' (see Sec. 4.9). $A(x)$, on the other hand, represents the weight of the part of t_x that has already been processed, which is not yet transferred to the inside weight (see Sec. 4.11).

The total weight of item η, i.e., the sum of its inside and (estimated) outside weights, is equal to $w + h(x, r)$. This total weight determines the order in which the created chart items are removed from the agenda.

4.14 Example

Fig. 10 shows a fragment of the hypergraph constructed when processing the sentence and grammar presented in Fig. 7 and Fig. 9. Each node represents a chart item and each (hyper)arc represents an application of a deduction rule (see Tab. 1). Additionally, to each chart item a pair of weights $[w; w_h]$ is assigned (linked to it via a dotted line), where w is the corresponding inside weight (calculated as prescribed by the deduction rules), and w_h – the corresponding value of the heuristic (see Sec. 4.13). Prediction weights are not shown because, in this example, they are all equal to 0.

4.15 Admissibility and monotonicity

The A* heuristic is *admissible* if it never overestimates the remaining parsing cost. It is *monotonic* if the total parsing cost $(w + h(\eta))$ never decreases as new chart items are created. The two properties guarantee correctness, i.e., that the first *final* chart item $\eta = (N, (0, n)) : \mathit{root}(N) \wedge \ell(N) \in S$ reached by the parser corresponds to a lowest-weight derivation, where S is the set of start symbols, and that the inside weight w accompanying η is the corresponding lowest weight.

The heuristic defined in Eq. 3 is both admissible and monotonic within the context of the parser specified in Tab. 1. Admissibility is virtually by definition – it stems from the assumption adopted in the heuristic that each remaining word will be matched with the lowest-weight ET and the lowest-weight dependency head. Monotonicity, on the other hand, can be proved by induction on the parser's deduction rules. We provide a formal monotonicity proof, written in Coq, in the ParTAGe's code repository. Additionally, the tool provides an optional runtime check, which allows to verify the monotonicity of the actual implementation each time a new chart item is created.

5. When subtree sharing is used, the parsing configuration x can correspond to several different ETs t_x and, therefore, $\omega(t_x)$ cannot be uniquely determined.

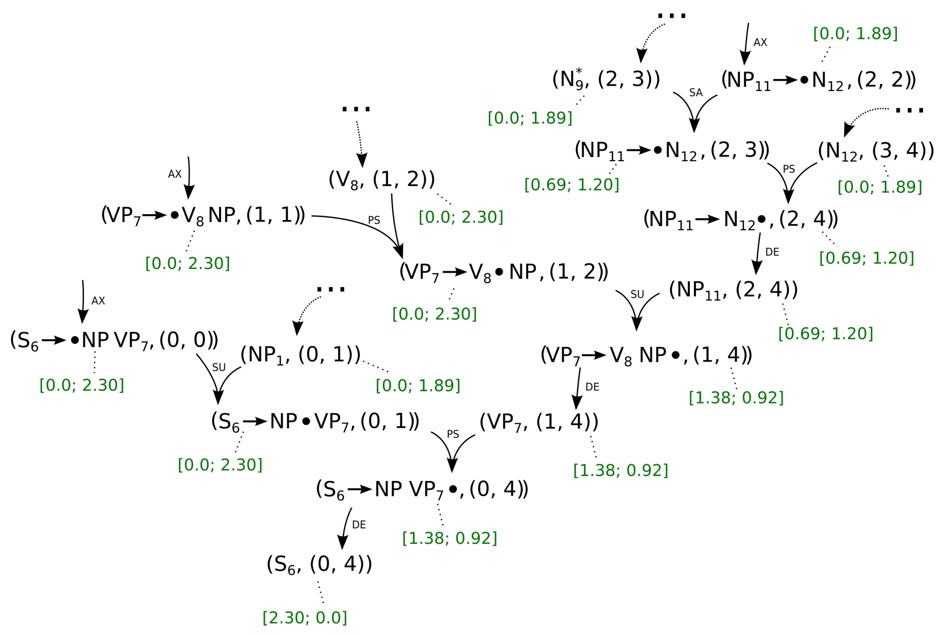


Figure 10: Fragment of the hypergraph decorated with the inside and the estimated outside (heuristic values, see Sec. 4.13) weights, generated when processing the input from Fig. 7 and Fig. 9. The prediction weights (calculated via the deduction rules along the inside weights) are not shown because, in this example, they are all equal to 0.

5. Evaluation/Experiments

5.1 Dataset: French LTAG and Word Embeddings

In our experiments we use an LTAG for French extracted from the French Treebank (FTB; Abeillé et al. (2003)). We followed the approach to treebank-based LTAG induction developed by Xia (1999) and largely adopted parameters reported in Bladier et al. (2018c) for extracting an LTAG from FTB. Inducing a grammar from a treebank means identifying a set of productions that could have produced its parse trees. In our case, this amounts to decomposing each treebank tree into a sequence of elementary trees together with a derivation tree that specifies how the elementary trees have to be combined.

We use the top-down approach to LTAG induction described in Xia (1999). The main idea of this approach is to cast the constituency trees in the treebank as derived trees in LTAG. Elementary trees are extracted in a top-down fashion using percolation tables to identify grammatically obligatory elements (i.e., *complements*), optional elements (i.e., *modifiers*), as well as a head child for each constituent. Upon marking the nodes in trees as being heads, complements or modifiers, all subtrees corresponding to modifiers and complements are extracted forming auxiliary trees and initial trees, respectively. The head child and its lexical anchor are kept in the tree. When extracted in this way, elementary trees contain the corresponding lexical anchor and the branches represent a particular syntactic context of a construction with slots for its complements (Bladier et al., 2018c).

Since several different LTAGs can be extracted from the same treebank depending on the number of POS-tag and node labels, head and modifier percolation tables as well as the research question, we adopted the parameters for the French LTAG induction with the most accurate supertagging results described in Bladier et al. (2018a). We reduced the number of POS tags to 13 and kept most of the multi-word-expressions (MWEs) in the grammar. We only transformed some of the nominal MWEs with regular syntactic patterns to regular NP constituents (for example *(MWN (A ancien)*

(N élève)) → *(NP (AP (A ancien)) (N élève)))* in order to keep the size of the grammar low. Table 2 provides some statistics on the extracted LTAG grammar (13 POS tags, including compounds and including punctuation marks).

Since the trees in FTB have a rather flat structure, the LTAG grammar we use for the experiments does not have regular adjunction, but sister-adjunction. The resulting LTAG with sister-adjunction is basically an LTIG *Lexicalized Tree Insertion Grammar*; (Schabes and Waters, 1995). Auxiliary trees in an LTIG do not allow wrapping adjunction or adjunction on the root node but permit multiple simultaneous adjunction on a single node of initial trees. However, since LTIG is a special variant of LTAG, we refer to the extracted grammar as LTAG in the remainder of the paper.

Parameters	**French LTAG**
Supertags	5103
Supertags occuring once	2611
POS tags	13
Sentences	21550
Avg. sentence length	29.81
# initial trees	1953
# auxiliary trees	3150

Table 2: Statistics on the extracted French LTAG.

For our experiments, we used the train, dev., and test sets from the French Treebank (version SPMRL 2013, described in Seddah et al. (2013)) in order to compare with previous work. We also included additional sentences to the train set from the latest version of the French Treebank (version 1.0 2016). Thus, our experiment data include 19 080, 1235, and 2541 sentences in the train, dev., and test set, respectively). We use pre-trained 200-dimensional word embeddings for French (Fauconnier, 2015) trained on 1.6 billion words in frWaC corpus to transform tokens in our corpus into numeric vectors. For out-of-vocabulary words, we assign embeddings by random vector.

	English LTAG from PTB (Kasai et al., 2018)			**French LTAG from FTB (our work, dev set)**		
Parsing Model	Stag acc.	UAS	LAS	Stag acc.	UAS	LAS
BiLSTM3	–	91.75	90.22	–	87.74	82.88
BiLSTM3-CNN	–	92.27	90.76	–	88.76	84.68
BiLSTM3-HW-CNN	–	92.29	90.71	–	88.52	84.30
BiLSTM4-CNN	–	92.11	90.66	–	88.73	84.43
BiLSTM4-HW-CNN	–	92.78	91.26	–	88.82	84.62
BiLSTM5-CNN	–	92.34	90.77	–	31.94	17.43
BiLSTM5-HW-CNN	–	92.64	91.11	–	–	–
BiLSTM4-CNN-POS	–	92.07	90.53	–	89.15	85.15
BiLSTM4-CNN-Stag	–	92.15	90.65	–	88.31	84.07
Joint (Stag)	90.51	92.97	91.48	84.05	88.97	84.70
Joint (POS + Stag)	**90.67**	**93.22**	**91.80**	**84.91**	**89.61**	**85.60**

Table 3: Supertagging and dependency parsing experiments and comparison with previous work. HW and CNN stand for the models with highway connections and the CNN-layer, while the numbers 3, 4 and 5 indicate the number of BiLSTM-layers. Both joint models use 4 BiLSTM layers, CNN, and the highway connections to predict dependencies along with supertags or POS tags + supertags, respectively. Our French results show a similar pattern to the reported results for English (Kasai et al., 2018).

5.2 Supertagging and Dependency Parsing Experiments

The pipeline of the tree parsing models based on supertagging consists of the step of choosing the sequences of supertags and dependency arcs and a subsequent actual parsing step. Supertagging is beneficial for parsing, since it disambiguates many choices before applying the costly actual parsing step. The problem of choosing the correct supertags, however, remains the bottleneck for such parsing models, meaning that the accuracy of the pipeline is strongly dependent on this step (Lewis et al., 2016). Thus, following the experiments reported in Kasai et al. (2018), we run several experiments with different parameters on the French LTAG to make sure we get the best results on predicting the supertags and dependency arcs. We used BiLSTM-models including 3, 4, and 5 hidden layers, with and without the CNN layer, and highway connections. We compare our results on French with previous work on English to prove that the parsing models show similar behaviour for both languages with the Joint (POS + Stag) model showing the best results (see Table 3).

The parsing models BiLSTM4-CNN-POS and BiLSTM4-CNN-Stag are pipeline models which use predicted POS and supertags as features for the system. The two joint models (Join Stag and Joint POS + Stag) predict either only stags or also POS tags and supertags together with dependency arcs and labels using only word and character features as input. The results of our supertagging and dependency parsing experiments are summarized in Table 3. The BiLSTM-models in this table predict only dependency arcs, while the joint models predict dependency arcs, labels of dependency arcs, supertags and also POS tags. We clustered 27 original dependency labels provided in the FTB data to 6 more general labels (*'suj'*, *'obj'*, *'oblique_arg'*, *'adj'*, *'subst'*, and *'root'*). We used the best parameters from the previous experiments (4 layers, CNN layer for character embeddings, highway connections) for the joint model, which proved to provide the most accurate predictions.

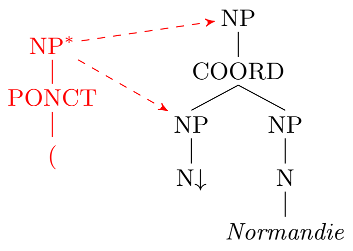


Figure 11: Attachment ambiguity: the tree on the left can be attached to both NP nodes in the tree for *Normandie*.

In addition, we also run an experiment with gold data using our modified A* parsing algorithm for LTAGs. For this experiment we used gold supertags and dependency arcs from the extracted French LTAG. The experiments show the parsing accuracy of 99,4 % (see Table 4). The accuracy is not 100 % even with the gold supertags and gold dependency arcs data. The reason for the lacking 0,6 % are the attachment ambiguities: Kasai's parser does not predict the Gorn addresses of the nodes in the elementary trees where the substitution or adjunction takes place. Lack of this information leads to an attachment ambiguity in cases where an initial tree has two different nodes with the same label on which the adjunction can take place (see an example in Fig. 11).

	gold supertags and arcs	
	dev	dev $\leq$ 25
Exactly matching sentences	88.50	95.61
POS accuracy	100.00	100.00
Labeled Recall	99.40	99.60
Labeled Precision	99.40	99.60
Labeled F1	**99.40**	**99.60**
# sentences	1235	501

Table 4: Parsing results with ParTAGe on gold data on the full set of sentences and on the sentences with less than 25 tokens.

5.3 A* LTAG Parsing Experiments

Finally, we carried out experiments with 10-best supertags and 10-best arcs on the FTB SPMRL (2013) Seddah et al. (2013) test set, which showed that ParTAGe is able to find a combination of suitable supertags and dependency arcs to produce full derived trees for every sentence (see Table 5). In comparison, taking the 1-best output allowed to produce a complete derivation only in around 50% of sentences in this dataset. We also measured the labeled evalb F1-score on the resulting constituency trees, excluding punctuation, which showed that our system achieves close to state-of-the-art results on FTB, although lower than the LSTM-based parser with self-attention developed by Kitaev and Klein (2018).

Note that the labeled F1 only evaluates the resulting derived trees with respect to the treebank trees. Our parser, however, outputs also information on which tree fragments constitute elementary trees and how they combine with each other. This additional syntactic information has been claimed to be useful for semantic tasks that require knowledge about predicate-argument relations such as semantic role labeling (Liu and Sarkar, 2009).

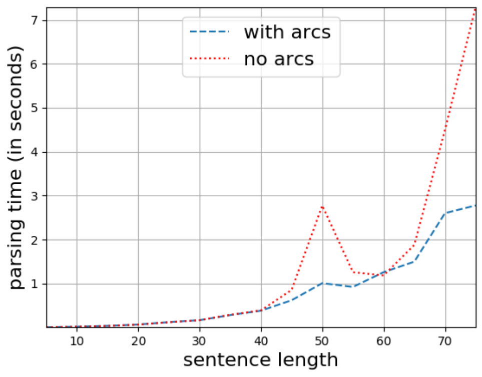
	1-best output (supertagger output)	**10-best output** (with arcs)		**10-best output** (no arcs)	
	test	test	test $\leq$ 25	test	test $\leq$ 25
Unlabeled Attachment Score (UAS)	88.54	**88.76**	**91.35**	84.17	89.07
Supertagging accuracy	81.22	**81.37**	82.93	81.35	**83.04**
Sentences 100% correct stags + arcs:	13.81	**14.44**	**30.16**	12.08	26.26
# sentences without parse	1215 (47.82%)	**0 (0%)**			
Exactly matching parses	–	**21.68**	**41.77**	17.83	36.40
Labeled F1 (our parser)	–	**84.36**	**88.02**	73.97	82.51
# sentences	2541	2541	1154	2541	1154
F1 Best Neural Parser (Cross and Huang, 2016), no punc.		83.11			
F1 Best Top-Down Parser (Stern et al., 2017), no punc.		82.23			
F1 LSTM Self-Attention (Kitaev and Klein, 2018), with punc.		**84.06**			
F1 Multiling. BiLSTM (Coavoux and Crabbé, 2017), with punc.		82.49			

Table 5: Results with ParTAGe parsing on predicted data (10-best columns) compared to the supertagger output (1-best column) and other parsing systems. We measured the results on parsing with 10-best predicted LTAG supertags including and excluding predicted dependency arcs. The results are provided for the full set of sentences and for the sentences with less than 25 tokens on the test set of SPMRL French Treebank (Seddah et al., 2013). We measured our results without counting punctuation marks. We use the coarse-grained POS tags provided in the SPMRL data and do not include function labels into our evaluations.

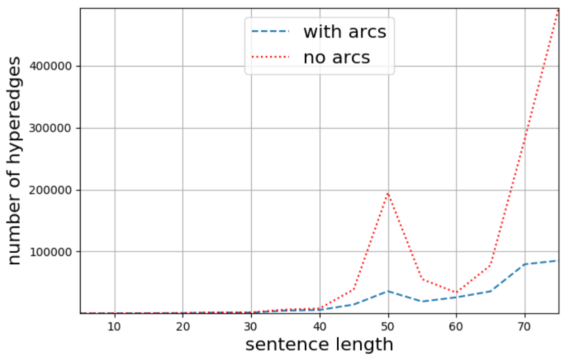
In the 10-best output (no arcs) column in Tab. 5, we additionally present the results when ParTAGe uses the distributions of supertags but ignores information about dependencies provided on input. In this case, the system achieves comparable supertagging accuracy, but at the cost of significantly lower UAS and F1 results. In particular, the almost 10% drop in F1 shows how important the dependency-related information is for the results of constituency parsing in this architecture. Besides, dependency-related information can also reduce the parsing time and the size of the resulting hypergraph, as shown in Fig. 12a and Fig. 12b, respectively.

6. Error Analysis

In the previous section we have shown that using 1-best supertags and 1-best dependency arcs is not sufficient for full LTAG parsing, while a 10-best input to the A* parsing model yields a parse for every sentence in the test and development set. Although every sentence now gets a full



(a)



(b)

Figure 12: Reduction of the parsing time and the number of hypergraph edges due to dependency information (dev set, 10-best output for sentences with length < 80.)

parse, the unlabeled attachment score (UAS) and labeled attachment score (LAS) results as well as results on supertagging improved only slightly compared to the (1-best) output of the original neural architecture of the supertagger and dependency parser by Kasai et al. (2018), while the supertagging results using dependency arcs got slightly worse. The difference in both metrics is small because the algorithm of ParTAGe searches for mutually compatible supertags and arcs to combine them to a full derived tree, and thus might pick not the correct supertags for the sentence but the compatible ones depending on their weights. More precisely, due to this compatibility requirement, an error in one place (incorrect supertag or incorrect dependency arc) oftentimes leads to further errors in order to retrieve a correct derivation tree. Thus, prediction of the supertags and arcs remains the bottleneck of the parsing architecture.

As we have seen, the results for English are better than the ones for French (see Table 3). One reason for this is probably the size of the training data for both languages (39 561 PTB sentences for English versus 19 080 FTB sentences for French). Furthermore, the average sentence length in the PTB (23.90 tokens) is lower than the one in the FTB (29.81 tokens), which makes the prediction of dependency arcs slightly harder for the French data.

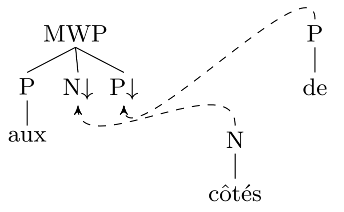


Figure 13: Flat supertags composing the multiword preposition *aux côtés de*.

A large number of supertagging errors for French are due to different possible attachment sites for punctuation marks. Punctuation marks are attached to the corresponding constituents and not to the root node of the whole sentence. However, punctuation marks help to identify possible constituents in sentences and omitting them does not substantially improve supertagging (Bladier et al., 2018c). PP attachments are another major source of errors while predicting supertags with French LTAGs. The supertagger encounters difficulties with classifying PPs as modifiers or complements, since FTB in the majority of cases does not offer additional function marks to distinguish these two. The supertagger also encounters problems with identifying the correct site for attaching the PPs to a node in the syntactic tree. Another major source of errors are the flat multi-word expressions encountered in the FTB, which lead to a high number of flat elementary trees and produce noise in the training data. See an example for the multi-word preposition *aux côtés de* ("alongside (with)") in Table 6 and the corresponding supertags in Fig. 13.

Gold supertag	**Predicted supertag**	**Example**
(NP* (PP (P ⋄) (NP↓)))	(PP* (PP (P ⋄) (NP↓)))	apprentissage **des** enfants
(NP* (PONCT ⋄))	(SENT* (PONCT ⋄))	**-LRB-** 66,7 % **-RRB-**
(N ⋄)	(NP (N ⋄))	aux **côtés** de Volvo
(NP (N ⋄))	(NP (N ⋄) (NP↓))	aux **partenaires** du montage
(SENT (NP↓) (VN (V ⋄)))	(AP (A ⋄))	des chômeurs sont **inscrits**

Table 6: Most common error classes for LTAG supertagging with French Treebank (SENT and PONCT stand for *sentence* and *punctuation*).

Table 7 shows the F1-scores for parsing of the 10 most frequent types of constituents in FTB. NP is the most frequent constituent in the FTB. Parsing NPs shows a relatively low F1 score, the reason for which is a diverse inner structure of NPs in French Treebank, which allows NPs to consist of differently structured subtrees including the problematic PP-subtrees. Thus, NPs can consist of a big variety of different supertags. This makes the prediction of the correct supertags hard, and supertagging errors are then oftentimes subsequently reflected in the resulting constituency tree.

Structural diversity of daughter nodes is also the reason of parsing errors for *Sint* and *Srel* constituents (i.e. final clauses in FTB). Note that parsing errors in NP constituents percolate to other constituents which contain them, for example *PP*, *COORD* (coordinating constituents), and *Sint* or *Ssub* (i.e. internal or subordinate clauses). Prediction of supertags for NPs can be potentially facilitated by using heuristic rules to make the inner structures of the subtrees constituting NPs more regular. In addition to this, the errors in parsing PP constituents result from the attachment to a wrong node in the tree, which is a result of errors in supertag prediction.

label	frequency	recall	precision	F1
(any)	100.00	84.82	83.90	84.36
NP	34.97	87.61	83.23	85.36
PP	20.39	88.54	79.19	83.60
VN	11.67	96.66	97.49	97.07
AP	5.74	93.60	71.81	81.27
SENT	5.02	100.00	100.00	100.00
VP	4.87	83.41	84.61	84.00
COORD	3.56	89.12	89.12	89.12
MWN	3.01	80.43	82.33	81.37
Sint	2.41	78.13	78.91	78.52
Srel	1.56	88.97	87.31	88.14

Table 7: Results of evaluating the pipeline parsing system on the test set, overall and for the 10 most frequent constituent labels. The scores are labeled EVALB scores.

7. Conclusions

We present a novel architecture for LTAG parsing based on the pipeline of a neural supertagger and dependency parser proposed by Kasai et al. (2018) and a modified A* based LTAG parsing algorithm ParTAGe implemented by Waszczuk (2017). We modified the supertagging model to produce n-best supertags and k-best arcs instead of 1-best output. This output is used as the input for the second step of the parsing pipeline. We also modified the A* based parser ParTAGe (Waszczuk, 2017) to be able to process n-best dependency arcs.

We have shown that the 1-best predicted supertags and 1-best predicted dependency relations between supertags are not sufficient do produce a full parse (i.e. full LTAG derived tree) for a large number of sentence. However, a sufficient large number of n and k has proven to be enough for obtaining full parsing trees on our data with our architecture.

We tested our architecture on an LTAG extracted automatically from the French Treebank (FTB). Our architecture shows comparable results to other parsers for French. Adding information on dependency arcs along with the information on supertags considerably reduces ambiguity for the actual parsing step. We have shown that adding information on dependency arcs to the original architecture of the A* based parser ParTAGe greatly decreases the parsing time and the number of possible parses for every tree.

8. Future Work

The parsing approach presented in this paper can be used for several kinds of LTAGs for different languages. In our follow up work we plan to extract different LTAG grammars for English, Polish and Dutch and to test our architecture on these grammars. We also intend to induce our own LTAG from the Penn Treebank (allowing both sister adjunction and regular adjunction), since the existing LTAGs (Chen et al., 2006; Xia, 1999) do not produce the original derived trees from the treebanks, but introduce extra nodes due to the binarization arising from adjoining regular auxiliary trees in LTAG, which contain a footnode. We will extract our grammars for Polish and Dutch from the existing treebanks Składnica (Woliński et al., 2011) and LASSY (Van Noord et al., 2013).

Besides using LTAG, we also plan to apply the approach presented in this paper to statistical parsing with Role and Reference Grammar (RRG; Van Valin and LaPolla (1997); Van Valin Jr (2005)). RRG is a grammar theory that has recently been formlized as a tree-rewriting formalism in the style of LTAG (Kallmeyer et al., 2013; Osswald and Kallmeyer, 2018), except that it has slightly different composition operation, namely an additional *wrapping* operation besides substitution and sister adjunction. For this work we will use RRGbank (Bladier et al., 2018b), an RRG-based version of the PTB, and we have to adapt ParTAGe in order to support wrapping.

Acknowledgments

This work was carried out as a part of the research projects TreeGraSP (treegrasp.phil.hhu.de) funded by a Consolidator Grant of the European Research Council (ERC) and the project "Parsing beyond CFG" funded by the German Research Foundation (DFG). The authors would like to thank Jungo Kasai for his help with the adaptation of the supertagger and the dependency parser, Marie Candito and Benoît Crabbé for consultations on the use of the French linguistic resources and David Arps for his comments on the paper. We also thank the anonymous reviewers for their careful reading, valuable suggestions and constructive comments.

References

Abeillé, Anne, Lionel Clément, and François Toussenel (2003), Building a treebank for French, *Treebanks*, Springer, pp. 165–187.

Bäcker, Jens and Karin Harbusch (2002), Hidden markov model-based supertagging in a user-initiative dialogue system, *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar andRelated Frameworks (TAG+ 6)*, pp. 269–278.

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

Bladier, Tatiana, Andreas van Cranenburgh, and Laura Kallmeyer (2018a), Extraction of LTAG-based supertags from the French treebank: challenges and possible solutions. Abstract presented at Grammar and Corpora 2018 (GAC 2018), November 15–17, 2018, University of Paris-Diderot, France. http://drehu.linguist.univ-paris-diderot.fr/gac-2018/abstracts/bladier_etal.pdf.

Bladier, Tatiana, Andreas van Cranenburgh, Kilian Evang, Laura Kallmeyer, Robin Möllemann, and Rainer Osswald (2018b), RRGbank: a Role and Reference Grammar Corpus of Syntactic Structures Extracted from the Penn Treebank, *Proceedings of the 17th International Workshop on Treebanks and Linguistic Theories (TLT 2018), December 13–14, 2018, Oslo University, Norway*, number 155, Linköping University Electronic Press, pp. 5–16.

Bladier, Tatiana, Andreas van Cranenburgh, Younes Samih, and Laura Kallmeyer (2018c), German and French neural supertagging experiments for LTAG parsing, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Student Research Workshop, Melbourne, Australia.* https://www.aclweb.org/anthology/P18-3009/.

Carroll, John, Nicolas Nicolov, Olga Shaumyan, Martine Smets, and David Weir (1999), Parsing with an extended domain of locality, *Ninth Conference of the European Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, Bergen, Norway. https://www.aclweb.org/anthology/E99-1029.

Chen, John, Srinivas Bangalore, and K Vijay-Shanker (2006), Automated extraction of tree-adjoining grammars from treebanks, *Natural Language Engineering* **12** (3), pp. 251–299, Cambridge University Press.

Chen, John, Srinivas Bangalore, Michael Collins, and Owen Rambow (2002), Reranking an n-gram supertagger, *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+ 6)*, pp. 259–268.

Chiang, David (2000), Statistical parsing with an automatically-extracted tree adjoining grammar, *Proceedings of the 38th annual meeting of the Association for Computational Linguistics*, pp. 456–463.

Chung, Wonchang, Siddhesh Suhas Mhatre, Alexis Nasr, Owen Rambow, and Srinivas Bangalore (2016), Revisiting supertagging and parsing: How to use supertags in transition-based parsing, *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12).* https://www.aclweb.org/anthology/W16-3309.pdf.

Coavoux, Maximin and Benoît Crabbé (2017), Multilingual lexicalized constituency parsing with word-level auxiliary tasks, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Association for Computational Linguistics, Valencia, Spain, pp. 331–336. https://www.aclweb.org/anthology/E17-2053.

Crabbé, Benoit (2005), Représentation informatique de grammaires fortement lexicalisées: Applicationa la grammaire darbres adjoints, *These de Doctorat, Université Nancy.*

Cross, James and Liang Huang (2016), Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Austin, Texas, pp. 1–11. https://www.aclweb.org/anthology/D16-1001.

Dozat, Timothy and Christopher D. Manning (2016), Deep biaffine attention for neural dependency parsing, *CoRR.* http://arxiv.org/abs/1611.01734.

Fauconnier, Jean-Philippe (2015), French word embeddings. http://fauconnier.github.io.

Joshi, Aravind K and Bangalore Srinivas (1994), Disambiguation of super parts of speech (or supertags): Almost parsing, *Proceedings of the 15th conference on Computational linguistics-Volume 1*, Association for Computational Linguistics, pp. 154–160.

Joshi, Aravind K and Yves Schabes (1997), Tree-adjoining grammars, *Handbook of formal languages*, Springer, pp. 69–123.

Kallmeyer, Laura and Giorgio Satta (2009), A polynomial-time parsing algorithm for tt-mctag, *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 994–1002. http://dl.acm.org/citation.cfm?id=1690219.1690285.

Kallmeyer, Laura and Rainer Osswald (2013), Syntax-driven semantic frame composition in lexicalized tree adjoining grammars, *Journal of Language Modelling* **1** (2), pp. 267–330.

Kallmeyer, Laura, Rainer Osswald, and Robert D. Van Valin, Jr. (2013), Tree wrapping for Role and Reference Grammar, *in* Morrill, G. and M.-J. Nederhof, editors, *Formal Grammar 2012/2013*, Vol. 8036 of *LNCS*, Springer, pp. 175–190.

Kasai, Jungo, Bob Frank, Tom McCoy, Owen Rambow, and Alexis Nasr (2017), TAG parsing with neural networks and vector representations of supertags, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1712–1722.

Kasai, Jungo, Robert Frank, Pauli Xu, William Merrill, and Owen Rambow (2018), End-to-end graph-based TAG parsing with neural networks, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Association for Computational Linguistics, New Orleans, Louisiana, pp. 1181–1194. https://www.aclweb.org/anthology/N18-1107.

Kitaev, Nikita and Dan Klein (2018), Constituency parsing with a self-attentive encoder, *arXiv preprint arXiv:1805.01052*.

Klein, Dan and Christopher D. Manning (2001), Parsing and Hypergraphs, *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001), 17-19 October 2001, Beijing, China*, Tsinghua University Press.

Le, Ngoc Luyen and Yannis Haralambous (2019), CCG Supertagging Using Morphological and Dependency Syntax Information, *International Conference on Computational Linguistics and Intelligent Text Processing*, Springer.

Lewis, Mike and Mark Steedman (2014), A* CCG Parsing with a Supertag-factored Model, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, pp. 990–1000. http://aclweb.org/anthology/D14-1107.

Lewis, Mike, Kenton Lee, and Luke Zettlemoyer (2016), LSTM CCG parsing, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 221–231.

Liu, Yudong and Anoop Sarkar (2007), Experimental evaluation of ltag-based features for semantic role labeling, *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 590–599.

Liu, Yudong and Anoop Sarkar (2009), Exploration of the ltag-spinal formalism and treebank for semantic role labeling, *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks, ACL-IJCNLP 2009*, pp. 1–9.

Ma, Xuezhe and Eduard Hovy (2016), End-to-end sequence labeling via bi-directional LSTM-CNNS-CRF, *arXiv preprint arXiv:1603.01354*.

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

Osswald, Rainer and Laura Kallmeyer (2018), Towards a formalization of role and reference grammar, *in* Kailuweit, Rolf, Lisann Knkel, and Eva Staudinger, editors, *Applying and Expanding Role and Reference Grammar.*, Albert-Ludwigs-Universitt, Universittsbibliothek. [NIHIN studies], Freiburg, pp. 355–378.

Rambow, Owen, K. Vijay-Shanker, and David Weir (1995), D-tree grammars, *33rd Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Cambridge, Massachusetts, USA, pp. 151–158. https://www.aclweb.org/anthology/P95-1021.

Resnik, Philip (1992), Probabilistic tree-adjoining grammar as a framework for statistical natural language processing, *COLING 1992 Volume 2: The 15th International Conference on Computational Linguistics*. https://www.aclweb.org/anthology/C92-2065.

Sarkar, Anoop (2000), Practical experiments in parsing using tree adjoining grammars, *Proceedings of the Fifth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+ 5)*, pp. 193–198.

Sarkar, Anoop (2007), Combining supertagging and lexicalized tree-adjoining grammar parsing, *Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach* p. 113, MIT Press Boston, MA, USA.

Schabes, Yves and Richard C Waters (1995), Tree insertion grammar: a cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced, *Computational Linguistics* **21** (4), pp. 479–513.

Seddah, Djamé, 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, *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, Association for Computational Linguistics, Seattle, Washington, USA, pp. 146–182. https://www.aclweb.org/anthology/W13-4917.

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

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015), Highway networks, *arXiv preprint arXiv:1505.00387.*

Stern, Mitchell, Jacob Andreas, and Dan Klein (2017), A minimal span-based neural constituency parser, *arXiv preprint arXiv:1705.03919.*

Van Noord, Gertjan, Gosse Bouma, Frank Van Eynde, Daniel De Kok, Jelmer Van der Linde, Ineke Schuurman, Erik Tjong Kim Sang, and Vincent Vandeghinste (2013), Large scale syntactic annotation of written Dutch: Lassy, *Essential speech and language technology for Dutch*, Springer, pp. 147–164.

Van Valin Jr, Robert D. (2005), *Exploring the syntax-semantics interface*, Cambridge University Press.

Van Valin, Robert D., Jr. and Randy LaPolla (1997), *Syntax: Structure, meaning and function*, Cambridge University Press.

Waszczuk, Jakub (2017), *Leveraging MWEs in practical TAG parsing: towards the best of the two worlds*, PhD thesis.

Waszczuk, Jakub, Agata Savary, and Yannick Parmentier (2016a), Enhancing practical TAG parsing efficiency by capturing redundancy, *21st International Conference on Implementation and Application of Automata (CIAA 2016)*, Proceedings of the 21st International Conference on Implementation and Application of Automata (CIAA 2016), Séoul, South Korea. https://hal.archives-ouvertes.fr/hal-01309598.

Waszczuk, Jakub, Agata Savary, and Yannick Parmentier (2016b), Promoting multiword expressions in A* TAG parsing, *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pp. 429–439. http://aclweb.org/anthology/C/C16/C16-1042.pdf.

Waszczuk, Jakub, Agata Savary, and Yannick Parmentier (2017), Multiword Expression-Aware A* TAG Parsing Revisited, *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, Association for Computational Linguistics, pp. 84–93. http://aclweb.org/anthology/W17-6209.

Woliński, Marcin, Katarzyna Głowińska, and Marek Świdziński (2011), A preliminary version of składnicaa treebank of polish, *Proceedings of the 5th Language & Technology Conference, Poznań*, pp. 299–303.

Xia, Fei (1999), Extracting tree adjoining grammars from bracketed corpora, *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, pp. 398–403.

XTAG Research Group (1998), A lexicalized tree adjoining grammar for english, *arXiv preprint cs/9809024*.

Yoshikawa, Masashi, Hiroshi Noji, and Yuji Matsumoto (2017), A* CCG parsing with a supertag and dependency factored model, *CoRR*. http://arxiv.org/abs/1704.06936.