Discontinuous Constituency Parsing with a Stack-Free Transition System and a Dynamic Oracle

Discontinuous Constituency Parsing with a Stack-Free Transition System and a Dynamic Oracle

Maximin Coavoux*
Naver Labs Europe
maximin.coavoux@naverlabs.com

Shay B. Cohen
ILCC, School of Informatics
University of Edinburgh
scohen@inf.ed.ac.uk

Abstract

We introduce a novel transition system for discontinuous constituency parsing. Instead of storing subtrees in a stack –i.e. a data structure with linear-time *sequential access*– the proposed system uses a *set* of parsing items, with constant-time *random access*. This change makes it possible to construct any discontinuous constituency tree in exactly $4n - 2$ transitions for a sentence of length n, whereas existing systems need a quadratic number of transitions to derive some structures. At each parsing step, the parser considers every item in the set to be combined with a *focus* item and to construct a new constituent in a bottom-up fashion. The parsing strategy is based on the assumption that most syntactic structures can be parsed incrementally and that the set –the memory of the parser– remains reasonably small on average. Moreover, we introduce a dynamic oracle for the new transition system, and present the first experiments in discontinuous constituency parsing using a dynamic oracle. Our parser obtains state-of-the-art results on three English and German discontinuous treebanks.

1 Introduction

Discontinuous constituency trees extend standard constituency trees by allowing crossing branches to represent long distance dependencies, such as the *wh*-extraction in Figure 1. Discontinuous constituency trees can be seen as derivations of Linear Context-Free Rewriting Systems (LCFRS, Vijay-Shanker et al., 1987), a class of formal grammars more expressive than context-free grammars, which makes them much harder to parse. In particular, exact CKY-style LCFRS parsing has an $\mathcal{O}(n^{3f})$ time complexity where f is the fan-out of the grammar (Kallmeyer, 2010).

*Work done at the University of Edinburgh.

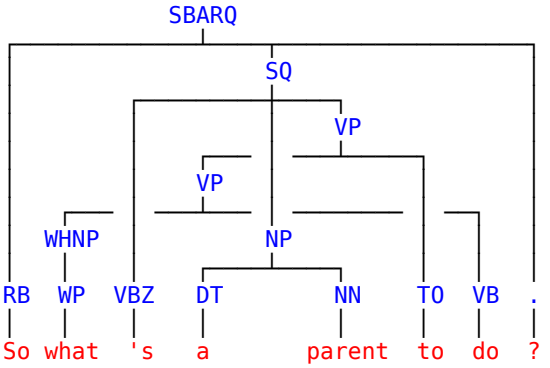

Figure 1: Discontinuous constituency tree from the Discontinuous Penn treebank.

A natural alternative to grammar-based chart parsing is transition-based parsing, that usually relies on fast approximate decoding methods such as greedy search or beam search. Transition-based discontinuous parsers construct discontinuous constituents by reordering terminals with the SWAP action (Versley, 2014a,b; Maier, 2015; Maier and Lichte, 2016; Stanojević and Garrido Alhama, 2017), or by using a split stack and the GAP action to combine two non-adjacent constituents (Coavoux and Crabbé, 2017a; Coavoux et al., 2019). These proposals represent the memory of the parser (i.e. the tree fragments being constructed) with data structures with **linear-time sequential access** (either a stack, or a stack coupled with a double-ended queue). As a result, these systems need to perform at least n actions to construct a new constituent from two subtrees separated by n intervening subtrees. Our proposal aims at avoiding this cost when constructing discontinuous constituents.

We design a novel transition system in which a discontinuous constituent is constructed in a single step, without the use of reordering actions such as SWAP. The main innovation is that the memory of the parser is not represented by a stack,

Initial configuration	$(\emptyset,\ \texttt{null},\ 0,\ \emptyset) : 0$		
Goal configuration	$(\emptyset,\ \{0, 1, \ldots, n-1\},\ n,\ C) : 4n - 2$		
Structural actions	**Input**	**Output**	**Precondition**
SHIFT	$(S,\ s_f,\ i,\ C) : j$	$\Rightarrow (S \cup \{s_f\},\ \{i\},\ i+1,\ C) : j+1$	$i < n$, j is even
COMBINE-s	$(S,\ s_f,\ i,\ C) : j$	$\Rightarrow (S - s,\ s_f \cup s,\ i,\ C) : j+1$	$s \in S$, j is even
Labelling actions			
LABEL-X	$(S,\ s_f,\ i,\ C) : j$	$\Rightarrow (S,\ s_f,\ i,\ C \cup \{(X,\ s_f)\}) : j+1$	j is odd
NO-LABEL	$(S,\ s_f,\ i,\ C) : j$	$\Rightarrow (S,\ s_f,\ i,\ C) : j+1$	$i \neq n$ or $S \neq \emptyset$, j is odd

Table 1: Set-based transition system description. Variable j is the number of steps performed since the start of the derivation.

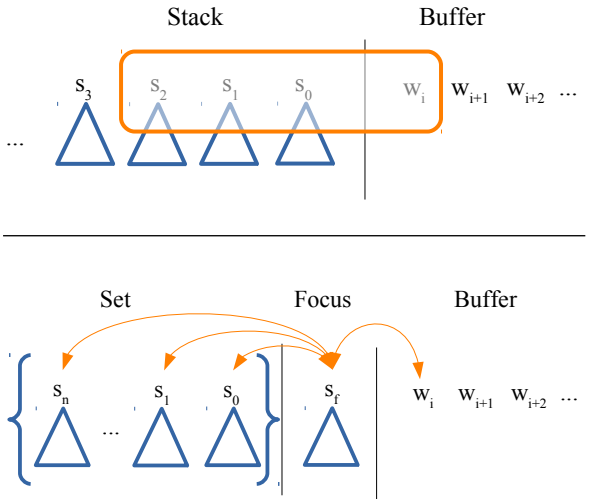

Figure 2: In a stack-based system like shift-reduce-swap (upper part), the parser extracts features from a *local* region of the configuration (orange part), to predict the next action such as: construct a new tree with label X and children s_0 and s_1 (REDUCE-X). In our proposed set-based system (lower part), we consider every item in the set to be combined bottom-up with the focus item s_f and score independently each possible transition.

as is usual in shift-reduce systems, but by an unordered **random-access** set. The parser considers every constituent in the current memory to construct a new constituent in a bottom-up fashion, and thus instantly models interactions between parsing items that are not adjacent. As such, we describe a left-to-right parsing model that deviates from the standard stack-buffer setting, a legacy from pushdown automata and classical parsing algorithms for context-free grammars.

Our contributions are summarized as follows:

- We design a novel transition system for discontinuous constituency parsing, based on a memory represented by a *set* of items, and that derives any tree in exactly $4n - 2$ steps for a sentence of length n;
- we introduce the first dynamic oracle for discontinuous constituency parsing;
- we present an empirical evaluation of the transition system and dynamic oracle on two German and one English discontinuous treebanks.

The code of our parser is released as an open-source project at https://gitlab.com/mcoavoux/discoparset.

2 Set-based Transition System

System overview We propose to represent the memory of the parser by (i) a set of parsing items and (ii) a single *focus item*. Figure 2 (lower part) illustrates a configuration in our system. The parser constructs a tree with two main actions: shift the next token to make it the new focus item (SHIFT), or combine any item in the set with the focus item to make a new constituent bottom-up (COMBINE action).

Since the memory is not an ordered data structure, the parser considers equally every pending parsing item, and thus constructs a discontinuous constituent in a single step, thereby making it able to construct any discontinuous tree in $\mathcal{O}(n)$ transitions.

The use of an unordered random-access data structure to represent the memory of the parser also leads to a major change for the scoring system (Figure 2). Stack-based systems use a *local view* of a parsing configuration to extract features and score actions: features only rely on the few topmost elements on the stack and buffer. The score of each transition depends on the totality of this local view. In constrast, we consider equally every item in the set, and therefore rely on a *global*

Even action	Set (S)	Focus (s_f)	Buffer	Odd action
	{}	none	So what 's a parent to do ?	
⇒SH⇒	{}	{So}	what 's a parent to do ?	⇒NO-LABEL
⇒SH⇒	{{So}$_0$}	{what}	's a parent to do ?	⇒LABEL-WHNP
⇒SH⇒	{{So}$_0$, {what}$_1$}	{'s}	a parent to do ?	⇒NO-LABEL
⇒SH⇒	{{So}$_0$, {what}$_1$, {'s}$_2$}	{a}	parent to do ?	⇒NO-LABEL
⇒SH⇒	{{So}$_0$, {what}$_1$, {'s}$_2$, {a}$_3$}	{parent}	to do ?	⇒NO-LABEL
⇒COMB-3⇒	{{So}$_0$, {what}$_1$, {'s}$_2$}	{a parent}	to do ?	⇒LABEL-NP
⇒COMB-2⇒	{{So}$_0$, {what}$_1$}	{'s a parent}	to do ?	⇒NO-LABEL
⇒SH⇒	{{So}$_0$, {what}$_1$, {'s a parent}$_2$}	{to}	do ?	⇒NO-LABEL
⇒SH⇒	{{So}$_0$, {what}$_1$, {'s a parent}$_2$, {to}$_5$}	{do}	?	⇒NO-LABEL
⇒COMB-1⇒	{{So}$_0$, {'s a parent}$_2$, {to}$_5$}	{what do}	?	⇒LABEL-VP
⇒COMB-5⇒	{{So}$_0$, {'s a parent}$_2$}	{what to do}	?	⇒LABEL-VP
⇒COMB-2⇒	{{So}$_0$}	{what 's a parent to do}	?	⇒LABEL-SQ
⇒COMB-0⇒	{}	{so what 's a parent to do}	?	⇒NO-LABEL
⇒SH⇒	{{So what 's a parent to do}$_0$}	{?}		⇒NO-LABEL
⇒COMB-0⇒	{}	{So what 's a parent to do ?}		⇒LABEL-SBARQ

Table 2: Full derivation for the sentence in Figure 1. As a convention, we index elements in the set with their left-index and use COMB-i to denote COMB-s_i. We also use tokens instead of their indexes for better legibility.

view of the memory (Section 3). However, we score each possible combinations independently: the score of a single combination only depends on the two constituents that are combined, regardless of the rest of the set.

2.1 System Description

Definitions We first define an *instantiated (discontinuous) constituent* (X, s) as a nonterminal label X associated with a set of token indexes s. We call $\min(s)$ the *left-index* of c and $\max(s)$ its *right-index*. For example in Figure 1, the two VPs are respectively represented by (VP, $\{1, 6\}$) and (VP, $\{1, 5, 6\}$), and they have the same right index (6) and left index (1).

A *parsing configuration* is a quadruple (S, s_f, i, C) where:

- S is a set of sets of indexes and represents the memory of the parser;
- s_f is a set of indexes called the *focus item*, and satisfies $\max(s_f) = i - 1$;
- i is the index of the next token in the buffer;
- C is a set of instantiated constituents.

Each new constituent is constructed bottom-up from the *focus* item and another item in the set S.

Transition set Our proposed transition system is based on the following types of actions:

- SHIFT constructs a singleton containing the next token in the buffer and assigns it as the new focus item. The former focus item is added to S.
- COMBINE-s computes the union between the focus item s_f and another item s from the set S, to form the new focus item $s \cup s_f$.
- LABEL-X instantiates a new constituent (X, s_f) whose yield is the set of indexes in the focus item s_f.
- NO-LABEL has no effect; its semantics is that the current focus set is not a constituent.

Following Cross and Huang (2016b), transitions are divided into **structural** actions (SHIFT, COMBINE-s) and **labelling** actions (LABEL-X, NO-LABEL). The parser may only perform a structural action on an even step and a labelling action on an odd step. For our system, this distinction has the crucial advantage of keeping the number of possible actions low at each parsing step, compared to a system that would perform a COMBINE action and a labelling action in a single REDUCE-s-X action.[1]

Table 1 presents each action as a deduction rule associated with preconditions. In Table 2, we describe how to derive the tree from Figure 1.

2.2 Oracles

Training a transition-based parser requires an oracle, i.e. a function that determines what the best action is in a specific parsing configuration to serve as a training signal. We first describe a static oracle that provides a canonical derivation for a given gold tree. We then introduce a dynamic oracle that determines what the best action is in any parsing configuration.

[1] In such a case, we would need to score $|S| \times |N| + 1$ actions, where N is the set of nonterminals, instead of $|S|+1$ actions for our system.

2.2.1 Static Oracle

Our transition system exhibits a fair amount of *spurious ambiguity*, the ambiguity exhibited by the existence of many possible derivations for a single tree. Indeed, since we use an unordered memory, an n-ary constituent (and more generally a tree) can be constructed by many different transition sequences. For example, the set $\{0, 1, 2\}$ might be constructed by combining

- $\{0\}$ and $\{1\}$ first, and the result with $\{2\}$; or
- $\{1\}$ and $\{2\}$ first, and the result with $\{0\}$; or
- $\{0\}$ and $\{2\}$ first, and the result with $\{1\}$.

Following Cohen et al. (2012), we eliminate spurious ambiguity by selecting a canonical derivation for a gold tree. In particular, we design the static oracle (i) to apply COMBINE as soon as possible in order to minimize the size of the memory (ii) to combine preferably with the most recent set in the memory when several combinations are possible. The first choice is motivated by properties of our system: when the memory is smaller, there are fewer choices, therefore decisions are simpler and less expensive to score.

2.2.2 Dynamic Oracle

Parsers are usually trained to predict the gold sequence of actions, using a static oracle. The limitation of this method is that the parser only sees a tiny portion of the search space at train time and only trains on gold input (i.e. configurations obtained after performing gold actions). At test time, it is in a different situation due to error propagation: it must predict what the best actions are in configurations from which the gold tree is probably no longer reachable.

To alleviate this limitation, Goldberg and Nivre (2012) proposed to train a parser with a *dynamic oracle*, an oracle that is defined for any parsing configuration and outputs the set of best actions to perform. In contrast, a *static oracle* is deterministic and is only defined for gold configurations.

Dynamic oracles were proposed for a wide range of dependency parsing transition systems (Goldberg and Nivre, 2013; Gómez-Rodríguez et al., 2014; Gómez-Rodríguez and Fernández-González, 2015), and later adapted to constituency parsing (Coavoux and Crabbé, 2016; Cross and Huang, 2016b; Fernández-González and Gómez-Rodríguez, 2018b,a).

In the remainder of this section, we introduce a dynamic oracle for our proposed transition system. It can be seen as an extension of the oracle of Cross and Huang (2016b) to the case of discontinuous parsing.

Preliminary definitions For a parsing configuration c, the relation $c \vdash c'$ holds iff c' can be derived from c by a single transition. We note $\vdash^*$ the reflexive and transitive closure of $\vdash$. An instantiated constituent (X, s) is **reachable** from a configuration $c = (S, s_f, i, C)$ iff there exists $c' = (S', s'_f, i', C')$ such that $(X, s) \in C'$ and $c \vdash^* c'$. Similarly, a set of constituents t (possibly a full discontinuous constituency tree) is reachable iff there exists a configuration $c' = (S', s'_f, i', C')$ such that $t \subseteq C'$ and $c \vdash^* c'$. We note $\mathsf{reach}(c, t^*)$ the set of constituents that are (i) in the gold set of constituents t^* (ii) reachable from c.

We define a total order $\preceq$ on index sets:

$$s \preceq s' \Leftrightarrow \begin{cases} \max(s) < \max(s'), \\ \textbf{or} \\ \max(s) = \max(s') \\ \text{and } s \subseteq s'. \end{cases}$$

This order naturally extends to the constituents of a tree: $(X, s) \preceq (X', s')$ iff $s \preceq s'$. If (X, s) precedes (X', s'), then (X, s) must be constructed before (X', s'). Indeed, since the right-index of the focus item is non-decreasing during a derivation (as per the transition definitions), constituents are constructed in the order of their right-index (first condition). Moreover, since the algorithm is bottom-up, a constituent must be constructed before its parent (second condition).

From a configuration $c = (S, s_f, i, C)$ at an odd step, a constituent $(X, s_g) \notin C$ is reachable iff both the following properties hold:

1. $\max(s_f) \leq \max(s_g)$;
2. $\forall s \in S \cup \{s_f\}, (s \subseteq s_g)$ or $(s \cap s_g = \emptyset)$.

Condition 1 is necessary because the parser can only construct new constituents (X, s) such that $s_f \preceq s$. Condition 2 makes sure that s_g can be constructed from a union of elements from $S \cup \{s_f\}$, potentially augmented with terminals from the bufffer: $\{i, i+1, \ldots, \max(s_g)\}$.

Following Cross and Huang (2016b), we define $\mathsf{next}(c, t^*)$ as the smallest reachable gold constituent from a configuration c. Formally:

$$\mathsf{next}(c, t^*) = \underset{\preceq}{\operatorname{argmin}}\, \mathsf{reach}(c, t^*).$$

Oracle algorithm We first define the oracle o for the odd step of a configuration $c = (S, s_f, i, C)$:

$$\mathsf{o}_{\mathsf{odd}}(c, t^*) = \begin{cases} \{\text{LABEL-X}\} & \text{if } \exists (X, s_f) \in t^*, \\ \{\text{NO-LABEL}\} & \text{otherwise.} \end{cases}$$

For even steps, assuming $\mathsf{next}(c, t^*) = (X, s_g)$, we define the oracle as follows:

$$\mathsf{o}_{\mathsf{even}}(c, t^*) = \begin{cases} \{\text{COMB-}s | (s_f \cup s) \subseteq s_g\} \\ \qquad \text{if } \max(s_g) = \max(s_f), \\ \{\text{COMB-}s | (s_f \cup s) \subseteq s_g\} \cup \{\text{SH}\} \\ \qquad \text{if } \max(s_g) > \max(s_f). \end{cases}$$

We provide a proof of the correctness of the oracle in Appendix A.

3 A Neural Network based on Constituent Boundaries

We first present an encoder that computes context-aware representations of tokens (Section 3.1). We then discuss how to compute the representation of a set of tokens (Section 3.2). We describe the action scorer (Section 3.3), the POS tagging component (Section 3.4), and the objective function (Section 3.5).

3.1 Token Representations

As in recent proposals in dependency and constituency parsing (Cross and Huang, 2016a; Kiperwasser and Goldberg, 2016), our scoring system is based on a sentence transducer that constructs a context-aware representation for each token.

Given a sequence of tokens $x_1^n = (x_1, \ldots, x_n)$, we first run a single-layer character bi-LSTM encoder $\mathbf{c}$ to obtain a character-aware embedding $\mathbf{c}(x_i)$ for each token. We represent a token x_i as the concatenation of a standard word embedding $\mathbf{e}(x_i)$ and the character-aware embedding: $\mathbf{w}_{x_i} = [\mathbf{c}(x_i); \mathbf{e}(x_i)]$.

Then, we run a 2-layer bi-LSTM transducer over the sequence of token representations:

$$(\mathbf{h}_1^{(1)}, \ldots, \mathbf{h}_n^{(1)}) = \text{bi-LSTM}(\mathbf{w}_{x_1}, \ldots, \mathbf{w}_{x_n}),$$
$$(\mathbf{h}_1^{(2)}, \ldots, \mathbf{h}_n^{(2)}) = \text{bi-LSTM}(\mathbf{h}_1^{(1)}, \ldots, \mathbf{h}_n^{(1)}).$$

The parser uses the context-aware token representations $\mathbf{h}_i^{(2)}$ to construct vector representations of sets or constituents.

3.2 Set Representations

An open issue in neural discontinuous parsing is the representation of discontinuous constituents. In projective constituency parsing, it has become standard to use the boundaries of constituents (Hall et al., 2014; Crabbé, 2015; Durrett and Klein, 2015), an approach that proved very successful with bi-LSTM token representations (Cross and Huang, 2016b; Stern et al., 2017).

Although constituent boundary features improves discontinuous parsing (Coavoux and Crabbé, 2017a), relying only on the left-index and the right-index of a constituent has the limitation of ignoring gaps inside a constituent. For example, since the two VPs in Figure 1 have the same right-index and left-index, they would have the same representations. It may also happen that constituents with identical right-index and left-index do not have the same labels.

We represent a (possibly partial) constituent with the yield s, by computing 4 indexes from s: $(\min(s), \max(s), \min(\overline{s}), \max(\overline{s}))$. The set $\overline{s}$ represents the gap in s, i.e. the tokens between $\min(s)$ and $\max(s)$ that are not in the yield of s:

$$\overline{s} = \{i | \min(s) < i < \max(s) \text{ and } i \notin s\}.$$

Finally, we extract the corresponding token representations of the 4 indexes and concatenate them to form the vector representation $\mathbf{r}(s)$ of s:

$$\mathbf{r}(s) = [\mathbf{h}^{(2)}_{\min(s)}; \mathbf{h}^{(2)}_{\max(s)}; \mathbf{h}^{(2)}_{\min(\overline{s})}; \mathbf{h}^{(2)}_{\max(\overline{s})}].$$

For an index set that does not contain a gap, we have $\overline{s} = \emptyset$. To handle this case, we use a parameter vector $\mathbf{h}_{\mathsf{nil}}$, randomly initialized and learned jointly with the network, to embed $\max(\emptyset) = \min(\emptyset) = \mathsf{nil}$.

For example, the constituents (VP, $\{1, 6\}$) and (VP, $\{1, 5, 6\}$) will be respectively vectorized as:

$$\mathbf{r}(\{1, 6\}) = [\mathbf{h}_1^{(2)}; \mathbf{h}_6^{(2)}; \mathbf{h}_2^{(2)}; \mathbf{h}_5^{(2)}],$$
$$\mathbf{r}(\{1, 5, 6\}) = [\mathbf{h}_1^{(2)}; \mathbf{h}_6^{(2)}; \mathbf{h}_2^{(2)}; \mathbf{h}_4^{(2)}].$$

This representation method makes sure that two distinct index sets have distinct representations, as long as they have at most one gap each. This property no longer holds if one index sets has more than one gap.

3.3 Action Scorer

For each type of action –structural or labelling– we use a feedforward network with two hidden layers.

Structural actions At structural steps, for a configuration $c = (S, s_f, i, C)$, we need to compute the score of $|S|$ COMBINE actions and possibly a SHIFT action. In our approach, the score of a combine-s action only depends on s and s_f and is independent of the rest of the configuration (i.e. other items in the set). We first construct input matrix M as follows:

$$M = \begin{pmatrix} \mathbf{r}(s_1) & \cdots & \mathbf{r}(s_n) & \mathbf{r}(\{i\}) \\ \mathbf{r}(s_f) & \cdots & \mathbf{r}(s_f) & \mathbf{r}(s_f) \end{pmatrix}.$$

Each of the first n columns of matrix M represents the input for a COMBINE action, whereas the last column is the input for the SHIFT action. We then compute the score of each structural action:

$$P(\cdot|c) = \text{Softmax}(\text{FF}_s(M)),$$

where FF_s is a feedforward network with two hidden layers, a $\tanh$ activation and a single output unit. In other words, it outputs a single scalar for each column vector of matrix M. This part of the network can be seen as an attention mechanism, where the focus item is the query, and the context is formed by the items in the set and the first element in the buffer.

Labelling actions We compute the probabilities of labelling actions as follows:

$$P(\cdot|s_f) = \text{Softmax}(\text{FF}_l(\mathbf{r}(s_f))),$$

where FF_l is a feedforward network with two hidden layers activated with the $\tanh$ function, and $|N| + 1$ output units, where N is the set of nonterminals.

3.4 POS Tagger

Following Coavoux and Crabbé (2017b), we use the first layer of the bi-LSTM transducer as input to a Part-of-Speech (POS) tagger that is learned jointly with the parser. For a sentence x_1^n, we compute the probability of a sequence of POS tags $t_1^n = (t_1, \ldots, t_n)$ as follows:

$$P(t_1^n|x_1^n) = \prod_{i=1}^{n} \text{Softmax}(\mathbf{W}^{(t)} \cdot \mathbf{h}_i^{(1)} + \mathbf{b}^{(t)})_{t_i},$$

where $\mathbf{W}^{(t)}$ and $\mathbf{b}^{(t)}$ are parameters.

3.5 Objective Function

In the static oracle setting, for a single sentence x_1^n, we optimize the sum of the log-likelihood of gold POS-tags t_1^n and the log-likelihood of gold parsing actions a_1^n:

$$\mathcal{L} = \mathcal{L}_t + \mathcal{L}_p,$$
$$\mathcal{L}_t = -\sum_{i=1}^{n} \log P(t_i|x_1^n),$$
$$\mathcal{L}_p = -\sum_{i=1}^{4n-2} \log P(a_i|a_1^{i-1}, x_1^n).$$

We optimize this objective by alternating a stochastic step for the tagging objective and a stochastic step for the parsing objective, as is standard in multitask learning (Caruana, 1997).

In the dynamic oracle setting, instead of optimizing the likelihood of the gold actions (assuming all previous actions were gold), we optimize the likelihood of the best actions, as computed by the dynamic oracle, from a configuration sampled from the space of all possible configurations. In practice, before each epoch, we sample each sentence from the training corpus with probability p and we use the current (non-averaged) parameters to parse the sentence and generate a sequence of configurations. Instead of selecting the highest-scoring action at each parsing step, as in a normal inference step, we sample an action using the softmax distribution computed by the parser, as done by Ballesteros et al. (2016). Then, we use the dynamic oracle to calculate the best action from each of these configurations. In case there are several best actions, we deterministically choose a single action by favoring a COMBINE over a SHIFT (to bias the model towards a small memory), and to COMBINE with the item with the highest right-index (to avoid spurious discontinuity in partial constituents). We train the parser on these sequences of potentially non-gold configuration-action pairs.

4 Experiments

We carried out experiments to assess the adequacy of our system and the effect of training with the dynamic oracle. We present the three discontinuous constituency treebanks that we used (Section 4.1), our experimental protocol (Section 4.2), then we discuss the results (Section 4.3) and the efficiency of the parser (Section 4.4).

	DPTB			Tiger			Negra		
	F1	Disc. F1	POS	F1	Disc. F1	POS	F1	Disc. F1	POS
static	91.1	68.2	97.2	87.4	61.7	98.3	83.6	51.3	97.9
dynamic	91.4	70.9	97.2	87.6	62.5	98.4	84.0	54.0	98.0

Table 3: Results on development corpora. F1 is the Fscore on all constituents, Disc. F1 is an Fscore computed only on discontinuous constituents, POS is the accuracy on part-of-speech tags. Detailed results (including precision and recall) are given in Table 7 of Appendix C.

	English (DPTB)		German (Tiger)		German (Negra)	
Model	F	Disc. F	F	Disc. F	F	Disc. F
Predicted POS tags or own tagging						
This work, dynamic oracle	**90.9**	67.3	82.5	**55.9**	**83.2**	**56.3**
Coavoux et al. (2019),* GAP, bi-LSTM	**91.0**	**71.3**	**82.7**	**55.9**	**83.2**	54.6
Stanojević and Garrido Alhama (2017),* SWAP, stack/tree-LSTM			77.0			
Coavoux and Crabbé (2017a), SR-GAP, perceptron			79.3			
Versley (2016), pseudo-projective, chart-based			79.5			
Corro et al. (2017),* bi-LSTM, Maximum Spanning Arborescence	89.2					
van Cranenburgh et al. (2016), DOP, ≤ 40	87.0				74.8	
Fernández-González and Martins (2015), dependency-based			77.3			
Gebhardt (2018), LCFRS with latent annotations			75.1			
Gold POS tags						
Stanojević and Garrido Alhama (2017),* SWAP, stack/tree-LSTM			81.6		82.9	
Coavoux and Crabbé (2017a), SR-GAP, perceptron			81.6	49.2	82.2	50.0
Maier (2015), SWAP, perceptron			74.7	18.8	77.0	19.8
Corro et al. (2017),* bi-LSTM, Maximum Spanning Arborescence	90.1		81.6			
Evang and Kallmeyer (2011), PLCFRS, < 25	79†					

Table 4: Discontinuous parsing results on the test sets.
*Neural scoring system. †Does not discount root symbols and punctuation.

4.1 Datasets

We perform experiments on three discontinuous constituency corpora. The discontinuous Penn Treebank was introduced by Evang and Kallmeyer (2011) who converted the long distance dependencies encoded by indexed traces in the original Penn treebank (Marcus et al., 1993) to discontinuous constituents. We used the standard split (sections 2-21 for training, 22 for development and 23 for test). The Tiger corpus (Brants et al., 2004) and the Negra corpus (Skut et al., 1997) are both German treebanks natively annotated with discontinuous constituents. We used the SPMRL split for the Tiger corpus (Seddah et al., 2013), and the split of Dubey and Keller (2003) for the Negra corpus.

4.2 Implementation and Protocol

We implemented our parser in Python using the Pytorch library (Paszke et al., 2017). We trained each model with the ASGD algorithm (Polyak and Juditsky, 1992) for 100 epochs. Training a single model takes approximately a week with a GPU. We evaluate a model every 4 epochs on the validation set and select the best performing model according to the validation F-score. We refer the reader to Table 5 of Appendix B for the full list of hyperparameters.

We evaluate models with the dedicated module of `discodop`[2] (van Cranenburgh et al., 2016). We use the standard evaluation parameters (`proper.prm`), that ignore punctuations and root symbols. We report two evaluation metrics: a standard Fscore (F) and an Fscore computed only on discontinuous constituents (Disc. F), which provides a more qualitative evaluation of the ability of the parser to recover long distance dependencies.

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

4.3 Results

Effect of Dynamic Oracle We present parsing results on the development sets of each corpus in Table 3. The effect of the oracle is in line with other published results in projective constituency parsing (Coavoux and Crabbé, 2016; Cross and Huang, 2016b) and dependency parsing (Goldberg and Nivre, 2012; Gómez-Rodríguez et al., 2014): the dynamic oracle improves the generalization capability of the parser.

External comparisons In Table 4, we compare our parser to other transition-based parsers (Maier, 2015; Coavoux and Crabbé, 2017a; Stanojević and Garrido Alhama, 2017; Coavoux et al., 2019), the pseudo-projective parser of Versley (2016), grammar-based chart parsers (Evang and Kallmeyer, 2011; van Cranenburgh et al., 2016; Gebhardt, 2018) and parsers based on dependency parsing (Fernández-González and Martins, 2015; Corro et al., 2017). Note that some of them only report results in a gold POS tag setting (the parser has access to gold POS tags and use them as features), a setting that is much easier than ours.

Our parser matches the state of the art of Coavoux et al. (2019). This promising result shows that it is feasible to design accurate transition systems without an ordered memory.

4.4 Efficiency

Our transition system derives a tree for a sentence of n words in exactly $4n - 2$ transitions. Indeed, there must be n SHIFT actions, and $n-1$ COMBINE actions. Each of these $2n - 1$ transitions must be followed by a single labelling action.

The statistical model responsible for choosing which action to perform at each parsing step needs to score $|S| + 1$ actions for a structural step and $|N|+1$ actions for a labelling step (where N is the set of possible nonterminals). Since in the worst case, $|S|$ contains $n - 1$ singletons, the parser has an $\mathcal{O}(n(|N| + n))$ time complexity.

In practice, the memory of the parser S remains relatively small on average. We report in Figure 3 the distribution of the size of S across configurations when parsing the development sets of three corpora. For the German treebanks, the memory contains 7 or fewer elements for more than 99 percents of configurations. For the Penn treebank, the memory is slighlty larger, with 98 percents of configuration with 11 or fewer items.

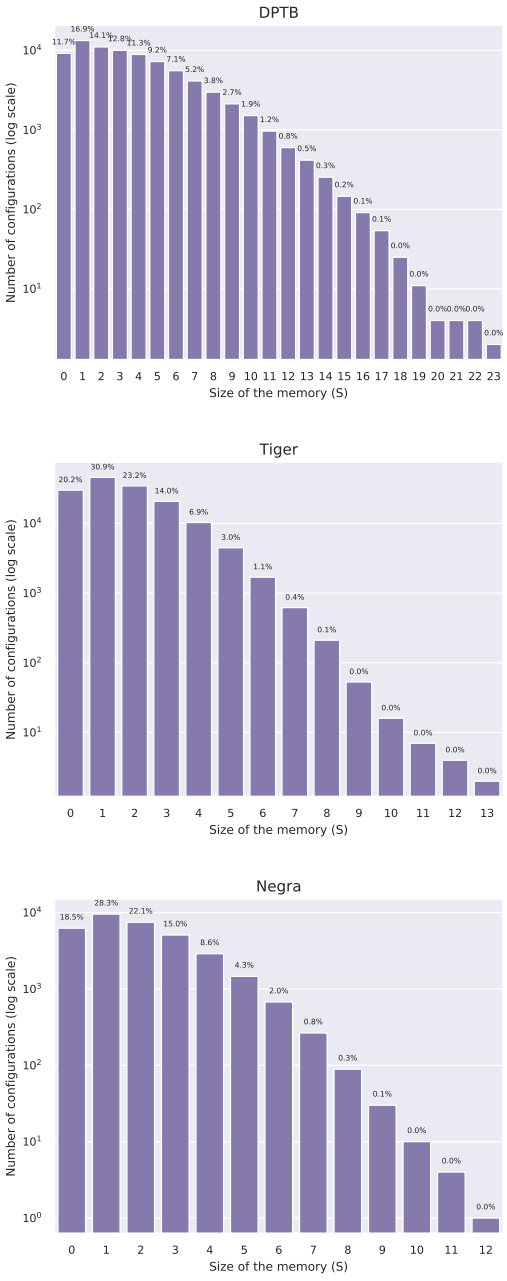

Figure 3: Distribution of the size of the memory of the parser S across configurations derived when parsing the development set of the three corpora. In practice, we observe that the memory remains small.

We report empirical runtimes in Table 6 of Appendix C. Our parser compares decently with other transition-based parsers, despite being written in Python.

5 Related Work

Existing transition systems for discontinuous constituency parsing rely on three main strategies for constructing discontinuous constituents: a swap-

based strategy, a split-stack strategy, and the use of non-local transitions.

Swap-based systems Swap-based transition systems are based on the idea that any discontinuous constituency tree can be transformed into a projective tree by reordering terminals. They reorder terminals by swapping them with a dedicated action (SWAP), commonly used in dependency parsing (Nivre, 2009). The first proposals in transition-based discontinuous constituency parsing used the SWAP action on top of an easy-first parser (Versley, 2014a,b). Subsequent proposals relied on a shift-reduce system (Maier, 2015; Maier and Lichte, 2016) or a shift-promote-adjoin system (Stanojević and Garrido Alhama, 2017).

The main limitation of swap-based system is that they tend to require a large number of transitions to derive certain trees. The choice of an oracle that minimizes derivation lengths has a substantially positive effect on parsing (Maier and Lichte, 2016; Stanojević and Garrido Alhama, 2017).

Split-stack systems The second parsing strategy constructs discontinuous constituents by allowing the parser to reduce pairs of items that are not adjacent in the stack. In practice, Coavoux and Crabbé (2017a) split the usual stack of shift-reduce parsers into two data structures (a stack and a double-ended queue), in order to give the parser access to two focus items: the respective tops of the stack and the dequeue, that may or may not be adjacent. A dedicated action, GAP, pushes the top of the stack onto the bottom of the queue to make the next item in the stack available for a reduction.

The split stack associated with the GAP action can be interpreted as a linear-access memory: it is possible to access the i^{th} element in the stack, but it requires i operations.

Non-local transitions Non-local transitions generalize standard parsing actions to non-adjacent elements in the parsing configurations. Maier and Lichte (2016) introduced a non-local transition SKIPSHIFT-i which applies SHIFT to the i^{th} element in the buffer. Non-local transitions are also widely used in non-projective dependency parsing (Attardi, 2006; Qi and Manning, 2017; Fernández-González and Gómez-Rodríguez, 2018).

The key difference between these systems and ours is that we use an unordered memory. As a result, the semantics of the COMBINE-s action we introduce in Section 2 is independent from a specific position in the stack or the buffer. A system with an action such as SKIPSHIFT-i needs to learn parameters with every possible i, and will only learn parameters with the SKIPSHIFT-i actions that are required to derive the training set. In contrast, we use the same parameters to score each possible COMBINE-s action.

6 Conclusion

We have presented a novel transition system that dispenses with the use of a stack, i.e. a memory with linear sequential access. Instead, the memory of the parser is represented by an unordered data structure with random-access: a set. We have designed a dynamic oracle for the resulting system and shown their empirical potential with state-of-the-art results on discontinuous constituency parsing of one English and two German treebanks. Finally, we plan to adapt our system to non-projective dependency parsing and semantic graph parsing.

Acknowledgments

We thank Caio Corro, Giorgio Satta, Marco Damonte, as well as NAACL anonymous reviewers for feedback and suggestions. We gratefully acknowledge the support of Huawei Technologies.

References

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170. Association for Computational Linguistics.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack lstm parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010, Austin, Texas. Association for Computational Linguistics.

Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France. Springer.

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

Rich Caruana. 1997. Multitask learning. *Machine Learning*, 28(1):41–75.

Maximin Coavoux and Benoit Crabbé. 2016. Neural greedy constituent parsing with dynamic oracles. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 172–182, Berlin, Germany. Association for Computational Linguistics.

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

Maximin Coavoux and Benoit Crabbé. 2017b. Multilingual lexicalized constituency parsing with word-level auxiliary tasks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 331–336, Valencia, Spain. Association for Computational Linguistics.

Maximin Coavoux, Benoît Crabbé, and Shay B. Cohen. 2019. Unlexicalized transition-based discontinuous constituency parsing. *CoRR*, abs/1902.08912v1.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2012. Elimination of spurious ambiguity in transition-based dependency parsing. *CoRR*, abs/1206.6735.

Caio Corro, Joseph Le Roux, and Mathieu Lacroix. 2017. Efficient discontinuous phrase-structure parsing via the generalized maximum spanning arborescence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1645–1655, Copenhagen, Denmark. Association for Computational Linguistics.

Benoit Crabbé. 2015. Multilingual discriminative lexicalized phrase structure parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1847–1856, Lisbon, Portugal. Association for Computational Linguistics.

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

James Cross and Liang Huang. 2016a. Incremental parsing with minimal features using bi-directional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37, Berlin, Germany. Association for Computational Linguistics.

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

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

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China. Association for Computational Linguistics.

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

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018. Non-projective dependency parsing with non-local transitions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 693–700. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018a. Dynamic oracles for top-down and in-order shift-reduce constituent parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1303–1313, Brussels, Belgium. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018b. Faster shift-reduce constituent parsing with a non-binary, bottom-up strategy. *CoRR*, abs/1804.07961.

Daniel Fernández-González and André F. T. Martins. 2015. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, Beijing, China. Association for Computational Linguistics.

Kilian Gebhardt. 2018. Generic refinement of expressive grammar formalisms with an application to discontinuous constituent parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3049–3063, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2015. An efficient dynamic oracle for unrestricted non-projective parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 256–261, Beijing, China. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927, Doha, Qatar. Association for Computational Linguistics.

David Hall, Greg Durrett, and Dan Klein. 2014. Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–237, Baltimore, Maryland. Association for Computational Linguistics.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*, 1st edition. Springer Publishing Company, Incorporated.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, Beijing, China. Association for Computational Linguistics.

Wolfgang Maier and Timm Lichte. 2016. Discontinuous parsing with continuous trees. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 47–57, San Diego, California. Association for Computational Linguistics.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics, Volume 19, Number 2, June 1993, Special Issue on Using Large Corpora: II*.

Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. 2015. Adding gradient noise improves learning for very deep networks. *CoRR*, abs/1511.06807.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Boris T. Polyak and Anatoli B. Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855.

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–117. Association for Computational Linguistics.

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

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 88–95, Washington, DC, USA. Association for Computational Linguistics.

Miloš Stanojević and Raquel Garrido Alhama. 2017. Neural discontinuous constituency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1667–1677, Copenhagen, Denmark. Association for Computational Linguistics.

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

Yannick Versley. 2014a. Experiments with easy-first nonprojective constituent parsing. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 39–53, Dublin, Ireland. Dublin City University.

Yannick Versley. 2014b. Incorporating semi-supervised features into discontinuous easy-first constituent parsing. *CoRR*, abs/1409.3813.

Yannick Versley. 2016. Discontinuity reˆ2-visited: A minimalist approach to pseudoprojective constituent parsing. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 58–69, San Diego, California. Association for Computational Linguistics.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics*.

A Oracle Correctness

The oracle o leads to the reachable tree with the highest F-score with respect to the gold tree. The F-score of a predicted tree $\hat{t}$ (represented as a set of instantiated constituents) with respect to a gold tree t^* is defined as:

$$\mathsf{precision}(\hat{t}, t^*) = p = \frac{|\hat{t} \cap t^*|}{|\hat{t}|},$$

$$\mathsf{recall}(\hat{t}, t^*) = r = \frac{|\hat{t} \cap t^*|}{|t^*|},$$

$$F_1(\hat{t}, t^*) = \frac{2pr}{p+r}.$$

By definition, $\mathsf{o}_{\mathsf{odd}}$ is optimal for precision because it constructs a constituent only if it is gold, and optimal for recall because it will construct a gold constituent if it is possible to do so.

Moreover, $\mathsf{o}_{\mathsf{even}}$ is optimal for recall because any gold constituent reachable from c will still be reachable after any transition in $\mathsf{o}_{\mathsf{even}}(c, t^*)$. Assuming a configuration $c = (S, s_f, i, C)$ and $\mathsf{next}(c, t^*) = s_g$, we consider separately the SHIFT case and the COMBINE-s case:

- SHIFT case ($\max(s_g) > \max(s_f)$): constituents (X, s) reachable from c and not reachable from SHIFT(c) satisfy $\max(s) = i$. If a gold constituent satisfies this property, we have $s \preceq s_g$, which contradicts the assumption that $s_g = \mathsf{next}(c, t^*)$ (see definition of oracle in Section 2.2.2).
- COMBINE-s case: Let (X, s') be a reachable gold constituent. Since it is compatible with s_g, there are three possible cases:
  - if (X, s') is an ascendant of s_g, then $(s \cup s_f) \subseteq s_g \subset s'$, therefore (X, s') is still reachable from COMBINE-s(c).
  - if (X, s') is a descendant of s_g then $s' \preceq s_g$, which contradicts the definition of s_g.
  - if s' and s_g are completely disjoint, we have $s' \cap s = s' \cap s_f = \emptyset$, therefore $s' \cap (s \cup s_f) = \emptyset$, and s' is still reachable from COMBINE-s(c).

Finally, since $\mathsf{o}_{\mathsf{even}}$ does not construct new constituents (it is the role of labelling actions), it is optimal for precision.

B Hyperparameters

The list of hyperparameters is presented in Table 5.

- We use learning rate warm-up (linear increase from 0 to t_{1000} during the first 1000 steps).
- Before the t^{th} update, we add Gaussian noise to the gradient of every parameter with mean 0 and variance $\frac{0.01}{(1+t)^{0.55}}$ (Neelakantan et al., 2015).
- All experiments use greedy search decoding (we did not experiment with beam search).
- Before each training step, we replace a word embedding by an 'UNK' pseudo-word embedding with probability 0.3. We only do this replacement for the least frequent word-types ($\frac{2}{3}$ least frequent word-types). The 'UNK' embedding is then used to represent unknown words.
- We apply dropout at the input of the tagger and the input of action scorers: each single prediction has its own dropout mask.

C Detailed Results

Architecture hyperparameters	
Dimension of word embeddings	32
Dimension of character embeddings	100
Dimension of character bi-LSTM state	50 for each direction
Dimension of sentence-level bi-LSTM	200 for each direction
Dimension of hidden layers for the action scorer	200
Activation functions	$\tanh$ for all hidden layers
Optimization hyperparameters	
Initial learning rate	$l_0 = 0.01$
Learning rate decay	$l_t = \frac{l_0}{1 + t \cdot 10^{-7}}$ for step number t
Dropout for tagger input	0.5
Dropout for parser input	0.2
Training epochs	100
Batch size	1 sentence
Optimization algorithm	Averaged SGD (Polyak and Juditsky, 1992; Bottou, 2010)
Word and character embedding initialization	$\mathcal{U}([-0.1, 0.1])$
Other parameters initialization (including LSTMs)	Xavier (Glorot and Bengio, 2010)
Gradient clipping (norm)	100
Dynamic oracle p	0.15

Table 5: Hyperparameters of the model.

Parser	Setting	Tiger		DPTB	
		tok/s	sent/s	tok/s	sent/s
This work	Python, neural, greedy, CPU	978	64	910	38
MTG (Coavoux et al., 2019)	C++, neural, greedy, CPU	1934	126	1887	80
MTG (Coavoux and Crabbé, 2017a)	C++, perceptron, beam=4, CPU	4700	260		
rparse (Maier, 2015)	Java, perceptron, beam=8, CPU		80		
rparse (Maier, 2015)	Java, perceptron, beam=1, CPU		640		
Corro et al. (2017)	C++, neural, CPU				≈ 7.3

Table 6: Parsing times on development sets in tokens per second (tok/s) and sentences per second (sent/s). The parsing times are presented as reported by authors, they are not comparable across parsers (since the experiments were run on different hardware). Our parser is run on a single core of an Intel i7 CPU.

		All const.			Disc. const.			POS
Development sets		F	P	R	F	P	R	Acc.
English (DPTB)	static	91.1	91.1	91.2	68.2	75.3	62.3	97.2
	dynamic	91.4	91.5	91.3	70.9	76.1	66.4	97.2
German (Tiger)	static	87.4	87.8	87.0	61.7	64.4	59.2	98.3
	dynamic	87.6	88.2	87.0	62.5	68.6	57.3	98.4
German (Negra)	static	83.6	83.8	83.4	51.3	53.3	49.5	97.9
	dynamic	84.0	84.7	83.4	54.0	58.1	50.5	98.0
Test sets		F	P	R	F	P	R	Acc.
English (DPTB)	dynamic	90.9	91.3	90.6	67.3	73.3	62.1	97.6
German (Tiger)	dynamic	82.5	83.5	81.5	55.9	62.4	50.6	98.0
German (Negra)	dynamic	83.2	83.8	82.6	56.3	64.9	49.8	98.0

Table 7: Detailed results. Overall, the positive effect of the dynamic oracle on Fscore is explained by its effect on precision.