Neural Greedy Constituent Parsing with Dynamic Oracles

Maximin Coavoux[1,2] and **Benoît Crabbé**[1,2,3]
[1]Univ. Paris Diderot, Sorbonne Paris Cité
[2]Alpage, Inria
[3]Institut Universitaire de France
maximin.coavoux@inria.fr
benoit.crabbe@linguist.univ-paris-diderot.fr

Abstract

Dynamic oracle training has shown substantial improvements for dependency parsing in various settings, but has not been explored for constituent parsing. The present article introduces a dynamic oracle for transition-based constituent parsing. Experiments on the 9 languages of the SPMRL dataset show that a neural greedy parser with morphological features, trained with a dynamic oracle, leads to accuracies comparable with the best non-reranking and non-ensemble parsers.

1 Introduction

Constituent parsing often relies on search methods such as dynamic programming or beam search, because the search space of all possible predictions is prohibitively large. In this article, we present a greedy parsing model. Our main contribution is the design of a dynamic oracle for transition-based constituent parsing. In NLP, dynamic oracles were first proposed to improve greedy dependency parsing training without involving additional computational costs at test time (Goldberg and Nivre, 2012; Goldberg and Nivre, 2013).

The training of a transition-based parser involves an oracle, that is a function mapping a configuration to the best transition. Transition-based parsers usually rely on a static oracle, only well-defined for gold configurations, which transforms trees into sequences of gold actions. Training against a static oracle restricts the exploration of the search space to the gold sequence of actions. At test time, due to error propagation, the parser will be in a very different situation than at training time. It will have to infer good actions from noisy configurations. To alleviate error propagation, a solution is to train the parser to predict the best action given any configuration, by allowing it to explore a greater part of the search space at train time. Dynamic oracles are non-deterministic oracles well-defined for any configuration. They give the best possible transitions for any configuration. Although dynamic oracles are widely used in dependency parsing and available for most standard transition systems (Goldberg and Nivre, 2013; Goldberg et al., 2014; Gómez-Rodríguez et al., 2014; Straka et al., 2015), no dynamic oracle parsing model has yet been proposed for phrase structure grammars.

The model we present aims at parsing morphologically rich languages (MRL). Recent research has shown that morphological features are very important for MRL parsing (Björkelund et al., 2013; Crabbé, 2015). However, traditional linear models (such as the structured perceptron) need to define rather complex feature templates to capture interactions between features. Additional morphological features complicate this task (Crabbé, 2015). Instead, we propose to rely on a neural network weighting function which uses a non-linear hidden layer to automatically capture interactions between variables, and embeds morphological features in a vector space, as is usual for words and other symbols (Collobert and Weston, 2008; Chen and Manning, 2014).

The article is structured as follows. In Section 2, we present neural transition-based parsing. Section 3 motivates learning with a dynamic oracle and presents an algorithm to do so. Section 4 introduces the dynamic oracle. Finally, we present parsing experiments in Section 5 to evaluate our proposal.

2 Transition-Based Constituent Parsing

Transition-based parsers for phrase structure grammars generally derive from the work of Sagae

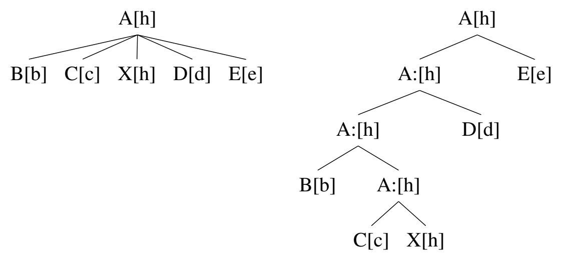

Figure 1: Order-0 head markovization.

Stack: $S\|(C,l,i)\|(B,i,k)\|(A,k,j)$	
Action	Constraints
RL(X) or RR(X), X$\in N$	A$\notin N_{tmp}$ and B $\notin N_{tmp}$
RL(X:) or RR(X:), X:$\in N_{tmp}$	C$\notin N_{tmp}$ or $j < n$
RR(X)	B$\notin N_{tmp}$
RL(X)	A$\notin N_{tmp}$

Table 1: Constraints to ensure that binary trees can be unbinarized. n is the sentence length.

$$\begin{array}{ll}
\text{Input} & w_0 w_1 \dots w_{n-1} \\
\text{Axiom} & \langle 0, \epsilon, \bot, \emptyset \rangle \\
\text{S} & \dfrac{\langle j, S, \bot, \gamma \rangle}{\langle j+1, S|(t_j, j, j+1), \top, \gamma \rangle} \\
\text{RL(X)} & \dfrac{\langle j, S|(A,i,k)|(B,k,j), \bot, \gamma \rangle}{\langle j, S|(X,i,j), \bot, \gamma \cup \{(X,i,j)\} \rangle} \\
\text{RU(X)} & \dfrac{\langle j, S|(t_{j-1}, j-1, j), \top, \gamma \rangle}{\langle j, S|(X, j-1, j), \bot, \gamma \cup \{(X, j-1, j)\} \rangle} \\
\text{GR} & \dfrac{\langle j, S, \top, \gamma \rangle}{\langle j, S, \bot, \gamma \rangle}
\end{array}$$

Figure 2: Transition system, the transition RR(X) and the lexicalization of symbols are omitted.

and Lavie (2005). In the present paper, we extend Crabbé (2015)'s transition system.

Grammar form We extract the grammar from a head-annotated preprocessed constituent treebank (cf Section 5). The preprocessing involves two steps. First, unary chains are merged, except at the preterminal level, where at most one unary production is allowed. Second, an order-0 head-markovization is performed (Figure 1). This step introduces temporary symbols in the binarized grammar, which are suffixed by ":". The resulting productions have one the following form:

$$\begin{array}{ll} X[h] \rightarrow A[a]\ B[b] & X[h] \rightarrow A[a]\ b \\ X[h] \rightarrow h & X[h] \rightarrow a\ B[b] \end{array}$$

where X, A, B are delexicalised non-terminals, a, b and $h \in \{a, b\}$ are tokens, and $X[h]$ is a lexicalized non-terminal. The purpose of lexicalization is to allow the extraction of features involving the heads of phrases together with their tags and morphological attributes.

Transition System In the transition-based framework, parsing relies on two data structures: a **buffer** containing the sequence of tokens to parse and a **stack** containing partial instantiated trees. A **configuration** $C = \langle j, S, b, \gamma \rangle$ is a tuple where j is the index of the next token in the buffer, S is the current stack, b is a boolean, and γ is the set of constituents constructed so far.[1]

Constituents are instantiated non-terminals, i.e. tuples (X, i, j) such that X is a non-terminal and (i, j) are two integers denoting its span. Although the content of γ could be retrieved from the stack, we make it explicit because it will be useful for the design of the oracle in Section 4.

From an initial configuration $C_0 = \langle 0, \epsilon, \bot, \emptyset \rangle$, the parser incrementally derives new configurations by performing actions until a final configuration is reached. S(HIFT) pops an element from the buffer and pushes it on the stack. R(EDUCE)(X) pops two elements from the stack, and pushes a new non-terminal X on the stack with the two elements as its children. There are two kinds of binary reductions, left (RL) or right (RR), depending on the position of the head. Finally, unary reductions (RU(X)) pops only one element from the stack and pushes a new non-terminal X. A **derivation** $C_{0 \Rightarrow \tau} = C_0 \overset{a_0}{\Rightarrow} \dots \overset{a_{\tau-1}}{\Rightarrow} C_\tau$ is a sequence of configurations linked by actions and leading to a final configuration. Figure 2 presents the algorithm as a deductive system. G(HOST)R(EDUCE) actions and boolean b ($\top$ or $\bot$) are used to ensure that unary reductions (RU) can only take place once after a SHIFT action.[2]

Constraints on the transitions make sure that predicted trees can be unbinarized. Figure 3 shows two examples of trees that could not have been obtained by the binarization process. In the first tree, a temporary symbol rewrites as two tempo-

[1]The introduction of γ is the main difference with Crabbé (2015)'s transition system.

[2]This transition system is similar to the extended system of Zhu et al. (2013). The main difference is the strategy used to deal with unary reductions. Our strategy ensures that derivations for a sentence all have the same number of steps, which can have an effect when using beam search. We use a GHOST-REDUCE action, whereas they use a padding strategy with an IDLE action.

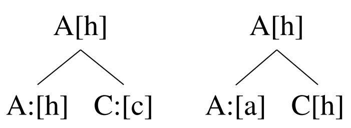

Figure 3: Examples of ill-formed binary trees

rary symbols. In the second one, the head of a temporary symbol is not the head of its direct parent. Table 1 shows a summary of the constraints used to ensure that any predicted tree is a well-formed binarized tree.[3] In this table, N is the set of non-terminals and $N_{tmp} \subset N$ is the set of temporary non-terminals.

Weighted Parsing The deductive system is inherently non-deterministic. Determinism is provided by a scoring function

$$s(C_{0 \Rightarrow \tau}) = \sum_{i=1}^{\tau} f_{\boldsymbol{\theta}}(C_{i-1}, a_i)$$

where $\boldsymbol{\theta}$ is a set of parameters. The score of a derivation decomposes as a sum of scores of actions. In practice, we used a feed-forward neural network very similar to the scoring model of Chen and Manning (2014). The input of the network is a sequence of typed symbols. We consider three main types (non-terminals, tags and terminals) plus a language-dependent set of morphological attribute types, for example, gender, number, or case (Crabbé, 2015). The first layer $\mathbf{h}^{(0)}$ is a lookup layer which concatenates the embeddings of each typed symbol extracted from a configuration. The second layer $\mathbf{h}^{(1)}$ is a non-linear layer with a rectifier activation (ReLU). Finally, the last layer $\mathbf{h}^{(2)}$ is a softmax layer giving a distribution over possible actions, given a configuration. The score of an action is its log probability.

Assuming $\mathbf{v_1}, \mathbf{v_2} \ldots, \mathbf{v}_\alpha$ are the embeddings of the sequence of symbols extracted from a configuration, the forward pass is summed up by the following equations:

$$\begin{aligned}
\mathbf{h}^{(0)} &= [\mathbf{v_1}; \mathbf{v_2}; \ldots; \mathbf{v}_\alpha] \\
\mathbf{h}^{(1)} &= \max\{0, \mathbf{W}^{(h)} \cdot \mathbf{h}^{(0)} + \mathbf{b}^{(h)}\} \\
\mathbf{h}^{(2)} &= \text{Softmax}(\mathbf{W}^{(o)} \cdot \mathbf{h}^{(1)} + \mathbf{b}^{(o)}) \\
f_{\boldsymbol{\theta}}(C, a) &= \log(\mathbf{h}^{(2)}_a)
\end{aligned}$$

[3]There are additional constraints which are not presented here. For example, SHIFT assumes that the buffer is not empty. A full description of constraints typically used in a slightly different transition system can be found in Zhang and Clark (2009)'s appendix section.

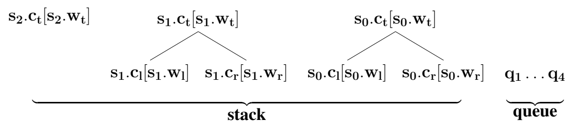

Figure 4: Schematic representation of local elements in a configuration.

Thus, $\boldsymbol{\theta}$ includes the weights and biases for each layer ($\mathbf{W}^{(h)}$, $\mathbf{W}^{(o)}$, $\mathbf{b}^{(h)}, \mathbf{b}^{(o)}$), and the embedding lookup table for each symbol type.

We perform greedy search to infer the best-scoring derivation. Note that this is not an exact inference. Most propositions in phrase structure parsing rely on dynamic programming (Durrett and Klein, 2015; Mi and Huang, 2015) or beam search (Crabbé, 2015; Watanabe and Sumita, 2015; Zhu et al., 2013). However we found that with a scoring function expressive enough and a rich feature set, greedy decoding can be surprisingly accurate (see Section 5).

Features Each terminal is a tuple containing the word form, its part-of-speech tag and an arbitrary number of language-specific morphological attributes, such as CASE, GENDER, NUMBER, ASPECT and others (Seddah et al., 2013; Crabbé, 2015). The representation of a configuration depends on symbols at the top of the two data structures, including the first tokens in the buffer, the first lexicalised non-terminals in the stack and possibly their immediate descendants (Figure 4). The full set of templates is specified in Table 6 of Annex A. The sequence of symbols that forms the input of the network is the instanciation of each position described in this table with a discrete symbol.

3 Training a Greedy Parser with an Oracle

An important component for the training of a parser is an oracle, that is a function mapping a gold tree and a configuration to an action. The oracle is used to generate local training examples from trees, and feed them to the local classifier.

A **static oracle** (Goldberg and Nivre, 2012) is an incomplete and deterministic oracle. It is only well-defined for gold configurations (the configurations derived by the gold action sequence) and returns the unique gold action. Usually, parsers use a static oracle to transform the set of binarized trees into a set $\mathcal{D} = \{C^{(i)}, a^{(i)}\}_{1 \le i \le T}$ of training examples. Training consists in minimizing the negative log likelihood of these examples. The limitation of this training method is that only gold configurations are seen during training. At test time, due to error propagation, the parser will have to predict good actions from noisy configurations, and will have much difficulty to recover after mistakes.

To alleviate this problem, a line of work (Daumé III et al., 2006; Ross et al., 2011) has cast the problem of structured prediction as a search problem and developed training algorithms aiming at exploring a greater part of the search space. These methods require an oracle well-defined for every search state, that is, for every parsing configuration.

A **dynamic oracle** is a complete and non-deterministic oracle (Goldberg and Nivre, 2012). It returns the non-empty set of the best transitions given a configuration and a gold tree. In dependency parsing, starting from Goldberg and Nivre (2012), dynamic oracle algorithms and training methods have been proposed for a variety of transition systems and led to substantial improvements in accuracy (Goldberg and Nivre, 2013; Goldberg et al., 2014; Gómez-Rodríguez et al., 2014; Straka et al., 2015; Gómez-Rodríguez and Fernández-González, 2015).

Online training An online trainer iterates several times over each sentence in the treebank, and updates its parameters until convergence. When a static oracle is used, the training examples can be pregenerated from the sentences. When we use a dynamic oracle instead, we generate training examples on the fly, by following the prediction of the parser (given the current parameters) instead of the gold action, with probability p, where p is a hyperparameter which controls the degree of exploration. The online training algorithm for a single sentence s, with an oracle function o is shown in Figure 5. It is a slightly modified version of Goldberg and Nivre (2013)'s algorithm 3, an approach they called *learning with exploration*.

In particular, as our neural network uses a cross-entropy loss, and not the perceptron loss used in Goldberg and Nivre (2013), updates are performed even when the prediction is correct. When $p = 0$, the algorithm acts identically to a static oracle trainer, as the parser always follows the gold transition. When the set of actions predicted by the oracle has more than one element, the best scoring element among them is chosen as the reference action to update the parameters of the neural network.

```
function TRAINONESENTENCE($s, \boldsymbol{\theta}, p, o$)
    $C \leftarrow$ INITIAL($s$)
    while $C$ is not a final configuration do
        $A \leftarrow o(C, s)$                ▷ set of best actions
        $\hat{a} \leftarrow \text{argmax}_a f_{\boldsymbol{\theta}}(C)_a$
        if $\hat{a} \in A$ then
            $t \leftarrow \hat{a}$                ▷ $t$: target
        else
            $t \leftarrow \text{argmax}_{a \in A} f_{\boldsymbol{\theta}}(C)_a$
        $\boldsymbol{\theta} \leftarrow$ UPDATE($\boldsymbol{\theta}, C, t$)        ▷ backprop
        if RANDOM() $< p$ then
            $C \leftarrow \hat{a}(C)$         ▷ Follow prediction
        else
            $C \leftarrow t(C)$          ▷ Follow best action
    return $\boldsymbol{\theta}$
```

Figure 5: Online training for a single annotated sentence s, using an oracle function o.

4 A Dynamic Oracle for Transition-Based Parsing

This section introduces a dynamic oracle algorithm for the parsing model presented in the previous 2 sections, that is the function o used in the algorithm in Figure 5.

The dynamic oracle must minimize a cost function $\mathcal{L}(c; t, T)$ computing the cost of applying transition t in configuration c, with respect to a gold parse T. As is shown by Goldberg and Nivre (2013), the oracle's correctness depends on the cost function. A correct dynamic oracle o will have the following general formulation:

$$o(c, T) = \{t | \mathcal{L}(c; t, T) = \min_{t'} \mathcal{L}(c; t', T)\} \quad (1)$$

The correctness of the oracle is not necessary to improve training. The oracle needs only to be good enough (Daumé et al., 2009), which is confirmed by empirical results (Straka et al., 2015). Goldberg and Nivre (2013) identified *arc-decomposability*, a powerful property of certain dependency parsing transition systems for which we can easily derive correct efficient oracles. When this property holds, we can infer whether a tree is reachable from the reachability of individual arcs. This simplifies the calculation of each transition cost. We rely on an analogue property we call **constituent decomposition**. A set of constituents is **tree-consistent** if it is a subset of a set corresponding to a well-formed tree. A phrase structure transition system is constituent-decomposable iff *for any configuration C and any tree-consistent set of constituents γ, if every constituent in γ is reachable from C, then the whole set is reachable from C* (constituent reachability will be formally defined in Section 4.1).

The following subsections are structured as follows. First of all, we present a cost function (Section 4.1). Then, we derive a correct dynamic oracle algorithm for an ideal case where we assume that there is no temporary symbols in the grammar (Section 4.2). Finally, we present some heuristics to define a dynamic oracle for the general case (Section 4.3).

4.1 Cost Function

The cost function we use ignores the lexicalization of the symbols. For the sake of simplicity, we momentarily leave apart the headedness of the binary reductions (until the last paragraph of Section 4) and assume a unique binary REDUCE action.

For the purpose of defining a cost function for transitions, we adopt a representation of trees as sets of constituents. For example, `(S (NP (D the) (N cat)) (VP (V sleeps)))` corresponds to the set $\{(\text{S}, 0, 3), (\text{NP}, 0, 2), (\text{VP}, 2, 3)\}$. As is shown in Figure 2, every reduction action (unary or binary) adds a new constituent to the set γ of already predicted constituents, which was introduced in Section 2. We define the cost of a predicted set of constituents $\hat{\gamma}$ with respect to a gold set γ^* as the number of constituents in γ^* which are not in $\hat{\gamma}$ penalized by the number of predicted unary constituents which are not in the gold set:

$$\begin{aligned}\mathcal{L}_r(\hat{\gamma}, \gamma^*) &= |\gamma^* - \hat{\gamma}| \\ &+ |\{(X, i, i+1) \in \hat{\gamma} | (X, i, i+1) \notin \gamma^*\}| \quad (2)\end{aligned}$$

The first term penalizes false negatives and the second one penalizes unary false positives. The number of binary constituents in γ^* and $\hat{\gamma}$ depends only on the sentence length n, thus binary false positives are implicitly taken into account by the fist term.

The cost of a transition and that of a configuration are based on **constituent reachability**. The relation $C \vdash C'$ holds iff C' can be deduced from C by performing a transition. Let $\vdash^*$ denote the reflexive transitive closure of $\vdash$. A set of constituents γ (possibly a singleton) is reachable from a configuration C iff there is a configuration $C' = \langle j, S, b, \gamma' \rangle$ such that $C \vdash^* C'$ and $\gamma \subseteq \gamma'$, which we write $C \leadsto \gamma$.

Then, the cost of an action t for a configuration C is the cost difference between the best tree reachable from $t(C)$ and the best tree reachable from C:

$$\mathcal{L}_r(t; C, \gamma^*) = \min_{\gamma : t(C) \leadsto \gamma} \mathcal{L}(\gamma, \gamma^*) - \min_{\gamma : C \leadsto \gamma} \mathcal{L}(\gamma, \gamma^*)$$

This cost function is easily decomposable (as a sum of costs of transitions) whereas F1 measure is not.

By definition, for each configuration, there is at least one transition with cost 0 with respect to the gold parse. Otherwise, it would entail that there is a tree reachable from C but unreachable from $t(C)$, for any t. Therefore, we reformulate equation 1:

$$o(C, \gamma^*) = \{t | \mathcal{L}_r(C; t, \gamma^*) = 0\} \qquad (3)$$

In the transition system, the grammar is left implicit: any reduction is allowed (even if the corresponding grammar rule has never been seen in the training corpus). However, due to the introduction of temporary symbols during binarization, there are constraints to ensure that any derivation corresponds to a well-formed unbinarized tree. These constraints make it difficult to test the reachability of constituents. For this reason, we instantiate two transition systems. We call SR-TMP the transition system in Figure 2 which enforces the constraints in Table 1, and SR-BIN, the same transition system without any of such constraints. SR-BIN assumes an idealized case where the grammar contains no temporary symbols, whereas SR-TMP is the actual system we use in our experiments.

4.2 A Correct Oracle for SR-BIN Transition System

SR-BIN transition system provides no guarantees that predicted trees are unbinarisable. The only condition for a binary reduction to be allowed is that the stack contains at least two symbols. If so, any non-terminal in the grammar could be used. In such a case, we can define a simple necessary and sufficient condition for constituent reachability.

Constituent reachability Let γ^* be a tree-consistent constituent set, and $C = \langle j, S, b, \gamma \rangle$ a parsing configuration, such that:

$$S = (X_1, i_0, i_1) \dots (X_p, i_{p-1}, i)|(A, i, k)|(B, k, j)$$

A binary constituent (X, m, n) is reachable iff it satisfies one of the three following properties :

1. $(X, m, n) \in \gamma$
2. $j < m < n$
3. $m \in \{i_0, \dots i_{p-1}, i, k\}, n \geq j$ and $(m, n) \neq (k, j)$

The first two cases are trivial and correspond respectively to a constituent already constructed and to a constituent spanning words which are still in the buffer.

In the third case, (X, m, n) can be constructed by performing $n - j$ times the transitions SHIFT and GHOST-REDUCE (or REDUCE-UNARY), and then a sequence of binary reductions ended by an X reduction. Note that as the index j in the configuration is non-decreasing during a derivation, the constituents whose span end is inferior to j are not reachable if they are not already constructed. For a unary constituent, the condition for reachability is straightforward: a constituent $(X, i-1, i)$ is reachable from configuration $C = \langle j, S, b, \gamma\rangle$ iff $(X, i-1, i) \in \gamma$ or $i > j$ or $i = j \wedge b = \top$.

Constituent decomposability SR-BIN is constituent decomposable. In this paragraph, we give some intuition about why this holds. Reasoning by contradiction, let's assume that every constituent of a tree-consistent set γ^* is reachable from $C = \langle j, S|(A, i, k)|(B, k, j), b, \gamma\rangle$ and that γ^* is not reachable (contraposition). This entails that at some point during a derivation, there is no possible transition which maintains reachability for all constituents of γ^*. Let's assume C is in such a case. If some constituent of γ^* is reachable from C, but not from SHIFT(C), its span must have the form (m, j), where $m \leq i$. If some constituent of γ^* is reachable from C, but not from REDUCE(X)(C), for any label X, its span must have the form (k, n), where $n > j$. If both conditions hold, γ^* contains incompatible constituents (crossing brackets), which contradicts the assumption that γ^* is tree-consistent.

Computing the cost of a transition The conditions on constituent reachability makes it easy to compute the cost of a transition t for a given configuration $C = \langle j, S|(A, i, k)|(B, k, j), b, \gamma\rangle$ and a gold set γ^*:

```
1:  function O($\langle j, S|(A,i,k)|(B,k,j), b, \gamma\rangle, \gamma^*$)
2:      if $b = \top$ then        ▷ Last action was SHIFT
3:          if $(X, j-1, j) \in \gamma^*$ then
4:              return {REDUCEUNARY(X)}
5:          else
6:              return {GHOSTREDUCE}
7:      if $\exists n > j, (X, k, n) \in \gamma^*$ then
8:          return {SHIFT}
9:      if $(X, i, j) \in \gamma^*$ then
10:         return {REDUCE(X)}
11:     if $\exists m < i, (X, m, j) \in \gamma^*$ then
12:         return {REDUCE(Y), $\forall Y$}
13:     return $\{a \in \mathcal{A} | a$ is a possible action$\}$
```

Figure 6: Oracle algorithm for SR-BIN.

- The cost of a SHIFT is the number of constituents not in γ, reachable from C and whose span ends in j.
- The cost of a binary reduction REDUCE(X) is a sum of two terms. The first one is the number of constituents of γ^* whose span has the form (k, n) with $n > j$. These are no longer compatible with (X, i, j) in a tree. The second one is one if $(Y, i, j) \in \gamma^*$ and $Y \neq X$ and zero otherwise. It is the cost of mislabelling a constituent with a gold span.
- The cost of a unary reduction or that of a ghost reduction can be computed straightforwardly by looking at the gold set of constituents.

We present in Figure 6 an oracle algorithm derived from these observations.

4.3 A Heuristic-based Dynamic Oracle for SR-TMP transition system

The conditions for constituent reachability for SR-BIN do not hold any longer for SR-TMP. In particular, constituent reachability depends crucially on the distinction between temporary and non-temporary symbols. The algorithm in Figure 6 is not correct for this transition system. In Figure 7, we give an illustration of a prototypical case in which the algorithm in Figure 6 will fail. The constituent (C:, i, j) is in the gold set of constituents and could be constructed with REDUCE(C:). The third symbol on the stack being temporary symbol D:, the reduction to a temporary symbol will jeopardize the reachability of (C, m, j) because reductions are not possible when the two symbols at the top of the stack are temporary symbols. The best course of action is then a reduction to any non-temporary symbol, so as to keep (C, m, j) reachable. Note that in this case, the cost of REDUCE(C:) cannot be smaller than that of a single mislabelled constituent.

In fact, this example shows that the constraints inherent to SR-TMP makes it non constituent-decomposable. In the example in Figure 7, both constituents in the set $\{(\text{C}, m, j), (\text{C:}, i, j)\}$, a tree-consistent constituent set, is reachable. However, the whole set is not reachable, as REDUCE(C:) would make (C, m, j) not reachable.

In dependency parsing, several exact dynamic oracles have been proposed for non arc-decomposable transition systems (Goldberg et al., 2014), including systems for non-projective parsing (Gómez-Rodríguez et al., 2014). These oracles rely on tabular methods to compute the cost of transitions and have (high-degree) polynomial worst case running time. Instead, to avoid resorting to more computationally expensive exact methods, we adapt the algorithm in Figure 6 to the constraints involving temporary symbols using the following heuristics:

- If the standard oracle predicts a reduction, make sure to choose its label so that every reachable constituent $(X, m, j) \in \gamma^*$ $(m < i)$ is still reachable after the transition. Practically, if such constituent exists and if the third symbol on the stack is a temporary symbol, then do not predict a temporary symbol.
- When reductions to both temporary symbols and non-temporary symbols have cost zero, only predict temporary symbols. This should not harm training and improve precision for the unbinarized tree, as any non temporary symbol in the binarized tree corresponds to a constituent in the n-ary tree.

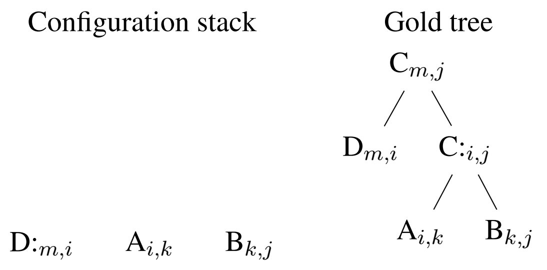

Figure 7: Problematic case. Due to the temporary symbol constraints enforced by SR-TMP, the algorithm in Figure 6 will fail on this example.

Dev F1 (EVALB)		Decoding	toks/sec
static (this work)	88.6	greedy	
dynamic (this work)	89.0	greedy	
Test F1 (EVALB)			
Hall et al. (2014)	89.2	CKY	12
Berkeley (Petrov et al., 2006)	90.1	CKY	169
Durrett and Klein (2015)†	91.1	CKY	-
Zhu et al. (2013)†	91.3	beam=16	1,290
Crabbé (2015)	90.0	beam=8	2,150
Sagae and Lavie (2006)	85.1	greedy	-
static (this work)	88.0	greedy	3,820$^\diamond$
dynamic (this work)	88.6	greedy	3,950$^\diamond$

Table 3: Results on the Penn Treebank (Marcus et al., 1993). † use clusters or word vectors learned on unannotated data. $^\diamond$ different architecture (2.3Ghz Intel), single processor.

Head choice In some cases, namely when reducing two non-temporary symbols to a new constituent (X, i, j), the oracle must determine the head position in the reduction (REDUCE-RIGHT or REDUCE-LEFT). We used the following heuristic: if (X, i, j) is in the gold set, choose the same head position, otherwise, predict both RR(X) and RL(X) to keep the non-determinism.

5 Experiments

We conducted parsing experiments to evaluate our proposal. We compare two experimental settings. In the 'static' setting, the parser is trained only on gold configurations; in the 'dynamic' setting, we use the dynamic oracle and the training method in Figure 5 to explore non-gold configurations. We used both the SPMRL dataset (Seddah et al., 2013) in the 'predicted tag' scenario, and the Penn Treebank (Marcus et al., 1993), to compare our proposal to existing systems. The tags and morphological attributes were predicted using Marmot (Mueller et al., 2013), by 10-fold jackknifing for the train and development sets. For the SPMRL dataset, the head annotation was carried out with the procedures described in Crabbé

Number of possible values	≤ 8	≤ 32	> 32
Dimensions for embedding	4	8	16

Table 4: Size of morphological attributes embeddings.

	Decoding	Arabic	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish	Avg
		Development F1 (EVALBSPMRL)									
Durrett and Klein (2015)[†]	CKY	80.68	84.37	80.65	**85.25**	89.37	**89.46**	82.35	92.10	**77.93**	84.68
Crabbé (2015)	beam=8	**81.25**	84.01	**80.87**	84.08	**90.69**	88.27	83.09	**92.78**	77.87	84.77
static (this work)	greedy	80.25	84.29	79.87	83.99	89.78	88.44	84.98	92.38	76.63	84.51
dynamic (this work)	greedy	80.94	**85.17**	80.31	84.61	90.20	88.70	**85.46**	92.57	77.87	**85.09**
		Test F1 (EVALBSPMRL)									
Björkelund et al. (2014)[†]		81.32[*]	**88.24**	**82.53**	**81.66**	**89.80**	**91.72**	83.81	90.50	**85.50**	**86.12**
Berkeley (Petrov et al., 2006)	CKY	79.19	70.50	80.38	78.30	86.96	81.62	71.42	79.23	79.18	78.53
Berkeley-Tags	CKY	78.66	74.74	79.76	78.28	85.42	85.22	78.56	86.75	80.64	80.89
Durrett and Klein (2015)[†]	CKY	80.24	85.41	**81.25**	**80.95**	88.61	**90.66**	82.23	**92.97**	**83.45**	**85.09**
Crabbé (2015)	beam=8	**81.31**	84.94	80.84	79.26	**89.65**	90.14	82.65	92.66	83.24	84.97
Fernández-González and Martins (2015)		-	85.90	78.75	78.66	88.97	88.16	79.28	91.20	82.80	(84.22)
static (this work)	greedy	79.77	85.91	79.62	79.20	88.64	90.54	84.53	92.69	81.45	84.71
dynamic (this work)	greedy	80.71	**86.24**	79.91	80.15	88.69	90.51	**85.10**	**92.96**	81.74	**85.11**
dynamic (this work)	beam=2	81.14	86.45	80.32	80.68	89.06	90.74	85.17	**93.15**	82.65	85.48
dynamic (this work)	beam=4	81.59	86.45	80.48	80.69	89.18	90.73	85.31	93.13	82.77	85.59
dynamic (this work)	beam=8	**81.80**	**86.48**	80.56	80.74	89.24	**90.76**	**85.33**	93.13	82.80	**85.64**

Table 2: Results on development and test corpora. Metrics are provided by `evalb_spmrl` with `spmrl.prm` parameters (`http://www.spmrl.org/spmrl2013-sharedtask.html`). [†] use clusters or word vectors learned on unannotated data. [*] Björkelund et al. (2013).

(2015), using the alignment between dependency treebanks and constituent treebanks. For English, we used Collins' head annotation rules (Collins, 2003). Our system is entirely supervised and uses no external data. Every embedding was initialised randomly (uniformly) in the interval $[-0.01, 0.01]$. Word embeddings have 32 dimensions, tags and non-terminal embeddings have 16 dimensions. The dimensions of the morphological attributes depend on the number of values they can have (Table 4). The hidden layer has 512 units.[4]

For the 'dynamic' setting, we trained every other k sentence with the dynamic oracle and the other sentences with the static oracle. This method, used by Straka et al. (2015), allows for high values of p, without slowing or preventing convergence. We used several hyperparameters combinations (see Table 5 of Annex A). For each language, we present the model with the combination which maximizes the developement set F-score. We used Averaged Stochastic Gradient Descent (Polyak and Juditsky, 1992) to minimize the negative log likelihood of the training examples. We shuffled the sentences in the training set before each iteration.

Results Results for English are shown in Table 3. The use of the dynamic oracle improves F-score by 0.4 on the development set and 0.6 on the test set. The resulting parser, despite using greedy decoding and no additional data, is quite accurate. For example, it compares well with Hall et al. (2014)'s span based model and is much faster.

For the SPMRL dataset, we report results on the development sets and test sets in Table 2. The metrics take punctuation and unparsed sentences into account (Seddah et al., 2013). We compare our results with the SPMRL shared task baselines (Seddah et al., 2013) and several other parsing models. The model of Björkelund et al. (2014) obtained the best results on this dataset. It is based on a product grammar and a discriminative reranker, together with morphological features and word clusters learned on unannotated data. Durrett and Klein (2015) use a neural CRF based on CKY decoding algorithm, with word embeddings pretrained on unannotated data. Fernández-González and Martins (2015) use a parsing-as-reduction approach, based on a dependency parser with a label set rich enough to reconstruct constituent trees from dependency trees. Finally, Crabbé (2015) uses a structured perceptron with rich features and beam-search decoding. Both Crabbé (2015) and Björkelund et al. (2014) use MARMOT-predicted morphological tags (Mueller et al., 2013), as is done in our experiments.

Our results show that, despite using a very simple greedy inference and being strictly supervised, our base model (static oracle training) is competitive with the best single parsers on this dataset.

[4]We did not tune these hyperparameters for each language. Instead, we chose a set of hyperparameters which achieved a tradeoff between training time and model accuracy. The effect of the morphological features and their dimensionality are left to future work.

We hypothesize that these surprising results come both from the neural scoring model and the morphological attribute embeddings (especially for Basque, Hebrew, Polish and Swedish). We did not test these hypotheses systematically and leave this investigation for future work.

Furthermore, we observe that the dynamic oracle improves training by up to 0.6 F-score (averaged over all languages). The improvement depends on the language. For example, Swedish, Arabic, Basque and German are the languages with the most important improvement. In terms of absolute score, the parser also achieves very good results on Korean and Basque, and even outperforms Björkelund et al. (2014)'s reranker on Korean.

Combined effect of beam and dynamic oracle Although initially, dynamic oracle training was designed to improve parsing without relying on more complex search methods (Goldberg and Nivre, 2012), we tested the combined effects of dynamic oracle training and beam search decoding. In Table 2, we provide results for beam decoding with the already trained local models in the 'dynamic' setting. The transition from greedy search to a beam of size two brings an improvement comparable to that of the dynamic oracle. Further increase in beam size does not seem to have any noticeable effect, except for Arabic. These results show that effects of the dynamic oracle and beam decoding are complementary and suggest that a good tradeoff between speed and accuracy is already achieved in a greedy setting or with a very small beam size

6 Conclusion

We have described a dynamic oracle for constituent parsing. Experiments show that training a parser against this oracle leads to an improvement in accuracy over a static oracle. Together with morphological features, we obtain a greedy parser as accurate as state-of-the-art (non reranking) parsers for morphologically-rich languages.

Acknowledgments

We thank the anonymous reviewers, along with Héctor Martínez Alonso and Olga Seminck for valuable suggestions to improve prior versions of this article.

References

Anders Björkelund, Özlem Çetinoğlu, Richárd Farkas, Thomas Mueller, and Wolfgang Seeker. 2013. (re)ranking meets morphosyntax: State-of-the-art results from the SPMRL 2013 shared task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145, Seattle, Washington, USA, October. Association for Computational Linguistics.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Mueller, Wolfgang Seeker, and Zsolt Szántó. 2014. Introducing the ims-wrocław-szeged-cis entry at the spmrl 2014 shared task: Reranking and morpho-syntax meet unlabeled data. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 97–102, Dublin, Ireland, August. Dublin City University.

Léon Bottou. 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186. Physica-Verlag HD.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4):589–637, December.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA. ACM.

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, September. Association for Computational Linguistics.

Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning*, 75(3):297–325.

Hal Daumé III, John Langford, and Daniel Marcu. 2006. Searn in practice.

Greg Durrett and Dan Klein. 2015. Neural crf parsing. In *Proceedings of the Association for Computational Linguistics*, Beijing, China, July. Association for Computational Linguistics.

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, July. Association for Computational Linguistics.

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

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. 2014. A tabular method for dynamic oracles in transition-based parsing. *TACL*, 2:119–130.

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, July. 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, October. 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)*, Baltimore, Maryland, June. Association for Computational Linguistics.

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

Haitao Mi and Liang Huang. 2015. Shift-reduce constituency parsing with dynamic programming and pos tag lattice. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1030–1035, Denver, Colorado, May–June. Association for Computational Linguistics.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA, October. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July. Association for Computational Linguistics.

B. T. Polyak and A. B. Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, July.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 691–698. 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, October. Association for Computational Linguistics.

Milan Straka, Jan Hajič, Jana Straková, and Jan Hajič jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *Proceedings of Fourteenth International Workshop on Treebanks and Linguistic Theories (TLT 14)*, December.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent 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 1169–1179, Beijing, China, July. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 162–171, Stroudsburg, PA, USA. Association for Computational Linguistics.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *ACL (1)*, pages 434–443. The Association for Computer Linguistics.

A Supplementary Material

'static' and 'dynamic' setting			'dynamic' setting	
learning rate	α	iterations	k	p
$\{0.01, 0.02\}$	$\{0, 10^{-6}\}$	$[1, 24]$	$\{8, 16\}$	$\{0.5, 0.9\}$

Table 5: Hyperparameters. α is the decrease constant used for the learning rate (Bottou, 2010).

$\mathbf{s_0.c_t}$	$\mathbf{s_0.w_t.tag}$	$\mathbf{s_0.w_t.form}$	$\mathbf{q_1.tag}$
$\mathbf{s_0.c_l}$	$\mathbf{s_0.w_l.tag}$	$\mathbf{s_0.w_l.form}$	$\mathbf{q_2.tag}$
$\mathbf{s_0.c_r}$	$\mathbf{s_0.w_r.tag}$	$\mathbf{s_0.w_r.form}$	$\mathbf{q_3.tag}$
$\mathbf{s_1.c_t}$	$\mathbf{s_1.w_t.tag}$	$\mathbf{s_1.w_t.form}$	$\mathbf{q_4.tag}$
$\mathbf{s_1.c_l}$	$\mathbf{s_1.w_l.tag}$	$\mathbf{s_1.w_l.form}$	$\mathbf{q_1.form}$
$\mathbf{s_1.c_r}$	$\mathbf{s_1.w_r.tag}$	$\mathbf{s_1.w_r.form}$	$\mathbf{q_2.form}$
$\mathbf{s_2.c_t}$	$\mathbf{s_2.w_t.tag}$	$\mathbf{s_2.w_t.form}$	$\mathbf{q_3.form}$
			$\mathbf{q_4.form}$
$\mathbf{s_0.w_t.m} \forall m \in M$		$\mathbf{q_0.m} \forall m \in M$	
$\mathbf{s_1.w_t.m} \forall m \in M$		$\mathbf{q_1.m} \forall m \in M$	

Table 6: These templates specify a list of addresses in a configuration. The input of the neural network is the instanciation of each address by a discrete typed symbol. Each $\mathbf{v_i}$ (Section 2) is the embedding of the i^{th} instantiated symbol of this list. M is the set of all available morphological attributes for a given language. We use the following notations (cf Figure 4): $\mathbf{s}_i$ is the i^{th} item in the stack, $\mathbf{c}$ denotes non-terminals, **t**op, **l**eft and **r**ight, indicate the position of an element in the subtree. Finally, $\mathbf{w}$ and $\mathbf{q}$ are respectively stack and buffer tokens.