Faster Shift-Reduce Constituent Parsing with a Non-Binary, Bottom-Up Strategy

Daniel Fernández-González and **Carlos Gómez-Rodríguez**
Universidade da Coruña
FASTPARSE Lab, LyS Research Group, Departamento de Computación
Campus de Elviña, s/n, 15071 A Coruña, Spain
d.fgonzalez@udc.es, carlos.gomez@udc.es

Abstract

We propose a novel non-binary shift-reduce algorithm for constituent parsing. Our parser follows a classical bottom-up strategy but, unlike others, it straightforwardly creates non-binary branchings with just one Reduce transition, instead of requiring prior binarization or a sequence of binary transitions. As a result, it uses fewer transitions per sentence than existing transition-based constituent parsers, becoming the fastest such system. Using static oracle training and greedy search, the accuracy of this novel approach is on par with state-of-the-art transition-based constituent parsers and outperforms all top-down and bottom-up greedy shift-reduce systems on WSJ and CTB. Additionally, we develop a dynamic oracle for training the proposed transition-based algorithm, achieving further improvements in both benchmarks and obtaining the best accuracy to date on CTB.

1 Introduction

Sagae and Lavie (2005) initially adapted the shift-reduce transition-based framework, notably successful in dependency parsing, to efficiently build constituent representations. To achieve that, they transformed constituent trees making them closer to dependency representations. In particular, they converted the original trees into headed, binary constituent trees. In these all branchings are at most binary, and nodes are annotated with headedness information, characteristic of dependency syntax (Kahane and Mazziotta, 2015). Figure 1 shows a constituent tree and its binarized version.

Under this scenario, a transition-based dependency parser such as the *arc-standard* algorithm

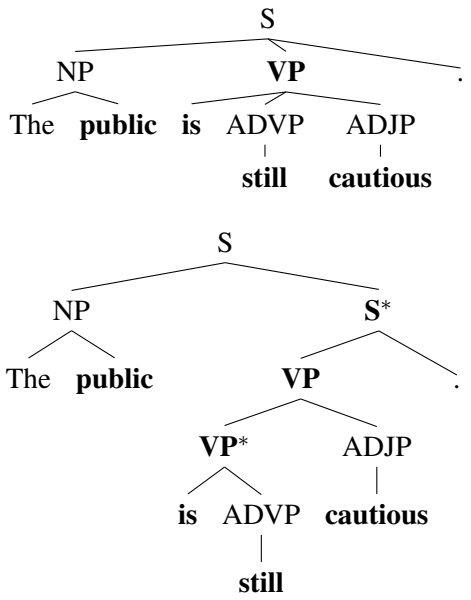

Figure 1: Simplified constituent tree (top) and its binarized equivalent (bottom), taken from English WSJ §22. Head-child nodes are in bold.

(Yamada and Matsumoto, 2003; Nivre et al., 2004) can be easily adapted to produce constituent structures. This linear-time shift-reduce algorithm applies a sequence of transitions that modify the content of two main data structures (a buffer and a stack) and create an arc (in the dependency framework) or connect two child nodes under a non-terminal node (in the constituency framework), building an analysis of the input sentence in a *bottom-up* manner. Further work on this approach (Zhang and Clark, 2009; Zhu et al., 2013; Watanabe and Sumita, 2015; Mi and Huang, 2015; Crabbé, 2015; Coavoux and Crabbé, 2016) achieved remarkable accuracy and speed, extending its popularity for constituent parsing.

Recently, some researchers have explored non-bottom-up strategies to build a constituent tree. (Dyer et al., 2016; Kuncoro et al., 2017) pro-

posed a *top-down* transition-based algorithm and, more recently, Liu and Zhang (2017a) developed a parser that builds the tree in an *in-order* traversal, seeking a compromise between the strengths of top-down and bottom-up approaches, and achieving state-of-the-art accuracy.

Liu and Zhang (2017a) also report that a traditional bottom-up strategy slightly outperforms a top-down approach when implemented under the same framework. This might be because, although a top-down approach adds a lookahead guidance to the process, it loses access to the rich features from partial trees used in bottom-up parsers.

Thus, it seems that the traditional bottom-up approach is still adequate for phrase structure parsing. However, its main drawback, compared to other strategies, is the need of binarization as preprocessing. Apart from the extra time spent, this needs a *percolation table* or a set of *head rules* of the language, which may not always be available.

To overcome this, Cross and Huang (2016a,b) presented transition systems that do not require prior binarization. However, their approach is not strictly non-binary, as transitions continue affecting only one or two nodes of the tree at a time. For instance, in these parsers, building the ternary branching $VP \rightarrow is\ ADVP\ ADJP$ in Figure 1 takes two reduce transitions: one to connect the two first children under the non-terminal VP, and another to add the last child to the subtree. Thus, binarization is not needed as pre-processing, but it is applied implicitly during parsing. Most if not all bottom-up constituent parsing algorithms (not only shift-reduce systems) use binarization explicitly or implicitly, as noted by Gómez-Rodríguez (2014) for context-free grammar parsers.

We propose a novel, bottom-up transition-based parser that is purely non-binary: for instance, it can create the previously mentioned ternary branching at once with a single reduce transition. This not only makes transition sequences shorter, achieving faster parsing, but also improves accuracy over all existing single bottom-up constituent parsers, explicitly binarized or not. The presented approach also improves over top-down parsers and is on par with state-of-the-art shift-reduce systems on the WSJ Penn Treebank (Marcus et al., 1993) and Chinese Treebank (CTB) benchmarks (Xue et al., 2005), being the fastest transition system ever created for constituent parsing.

To further improve the accuracy of our system, we also develop a *dynamic oracle* for training it. (Cross and Huang, 2016b; Coavoux and Crabbé, 2016) have already successfully applied this technique, originally designed for transition-based dependency parsing, to bottom-up phrase-structure parsing. Traditionally, transition-based systems were trained with *static oracles* that follow a gold sequence of transitions to create a model capable of analyzing new sentences at test time. This approach tends to yield parsers that are prone to suffer from error propagation (i.e. errors made in previous parser configurations that are propagated to subsequent states, causing harmful modifications in the transition sequence). To minimize the effect of error propagation, (Goldberg and Nivre, 2012) introduced the idea to train dependency parsers under closer conditions to those found at test time, where mistakes are inevitably made. This can be achieved by training the model with a dynamic oracle with exploration, which allows the parser to go through non-optimal parser configurations during learning time, teaching it how to recover from them and lose the minimum number of gold arcs.

Taking (Coavoux and Crabbé, 2016) as inspiration, we implement a dynamic oracle for our novel algorithm, achieving notable improvements in accuracy. Unlike (Coavoux and Crabbé, 2016), our dynamic oracle is optimal due to the simplicity of our algorithm and the lack of temporary symbols from binarization.

Experimental results with the dynamic oracle show further improvements in accuracy over the static oracle, and allow us to outperform the state of the art on the CTB supervised setting by 0.7 points.

2 Preliminaries

The basic bottom-up transition system of Sagae and Lavie (2005), used as a base for many other efficient constituent parsers, analyses a sentence from left to right by reading (moving) words from a buffer to a stack, where partial subtrees are built. This is done by a sequence of Shift (for reading) and Reduce (for building) transitions that will lead the parser through different states or parser configurations until a terminal one is reached.

More in detail, these parser configurations have the form $c = \langle \Sigma, i, f, \gamma \rangle$, where Σ is a *stack* of constituents, i is the index of the leftmost word in a list (called *buffer*) of unprocessed words, f is a boolean value that marks if a configuration is ter-

minal or not, and γ is the set of constituents that have already been built. Each constituent is represented as a tuple (X, l, r), where X is a non-terminal and l and r are integers defining its span (a word w_i is represented as $(w_i, i, i+1)$).

Note that the information about the set of predicted constituents γ and the spans of each constituent is not used by the parser. The transition system can be defined and work without it, and the same applies to the novel non-binary parser that will be introduced below. However, we include it explicitly in configurations because it simplifies the description of our dynamic oracle.

Given an input string $w_0 \cdots w_{n-1}$, the parsing process starts at the initial configuration $c_s(w_0 \dots w_{n-1}) = \langle [\,], 0, \mathit{false}, \{\} \rangle$ and, after applying a sequence of transitions, it ends in a terminal configuration $\langle \Sigma, n, \mathit{true}, \gamma \rangle \in C$.

Figure 2 shows the available transitions. The Shift transition moves the first (leftmost) word in the buffer to the stack; Reduce-Left-X and Reduce-Right-X pop the two topmost stack nodes and combine them into a new constituent with the non-terminal X as their parent, pushing it onto the stack (the head information provided by the direction encoded in each Reduce transition is used as a feature); Reduce-Unary pops the top node on the stack, uses it to create a unary subtree with label X, which is pushed onto the stack; and, finally, the Finish transition ends the parsing process. Note that every reduction action will add a new constituent to γ. Figure 3 shows the transition sequence that produces the binary phrase-structure tree in Figure 1.

The described transition system is determinized by training a classifier to greedily choose which transition should be applied next at each parser configuration. For this purpose, we train the parser to approximate an *oracle*, which chooses optimal transitions with respect to gold parse trees. This oracle can be static or dynamic, depending on the strategy used for training the parser. A static oracle trains the parser on only gold parser configurations, while the dynamic one can train the parser in any possible configuration, allowing the exploration of non-optimal parser states.

3 Novel Bottom-up Transition System

We propose a novel transition system for bottom-up constituent parsing that builds more-than-binary branchings at once by applying a single Reduce transition. We keep the parser configuration form defined by Sagae and Lavie (2005), but modify the transition set. In particular, we design a new non-binary Reduce transition that can create a subtree rooted at a non-terminal X and with the k topmost stack nodes as its children, with k ranging from 1 (to produce unary branchings) to the sentence length n. In that way, the ternary branch $VP \rightarrow \mathit{is}\ ADVP\ ADJP$, used previously as an example, could be created by just applying a Reduce-VP#3 transition, which will build a new constituent VP with three children at once.

The proposed transition set is formally described in Figure 4. Note that no specific transition is required for producing unary branching, as the novel Reduce transition can handle any kind of phrase-structure trees.

This approach does not need any kind of previous binarization (contrary to the one described in Figure 2) and, therefore, it lacks headedness information provided by binarized trees. In addition, since constituents of any kind can be reduced with just one transition, the parsing of a given sentence is done by consuming the shortest sequence of transitions among known transition systems for constituent parsing. Figure 5 shows the transition sequence necessary to produce the non-binary structure in Figure 1. In that simple example, we use 12 transitions, while the binary bottom-up system consumes 14 to parse the same sentence. Apart from being slower, said binary bottom-up parser needs to apply an unbinarization process to the parser output to produce a final non-binary phrase structure tree. The top-down (Dyer et al., 2016) and the in-order (Liu and Zhang, 2017a) algorithms need even more transitions to produce the tree in Figure 1: 16 and 17, respectively.

Moreover, this novel non-binary Reduce transition naturally lends itself to handling a non-terminal in a different way depending on its arity, enlarging the non-terminal dictionary. For instance, our system can distinguish between a verbal phrase with two children (VP#2) and one with three children (VP#3) and treat them differently. This is a way of encoding some information about grammatical subcategorization frames in the non-binary Reduce transition, which can be useful for the parser to learn in what circumstances it should create a VP#2 with only two elements (for instance, the verb and its direct object) or three

Shift:	$\langle \Sigma, i, false, \gamma \rangle \Rightarrow \langle \Sigma \vert (w_i, i, i+1), i+1, false, \gamma \cup \{(w_i, i, i+1)\} \rangle$
Reduce-Left/Right-X:	$\langle \Sigma \vert (Y, l, m) \vert (Z, m, r), i, false, \gamma \rangle \Rightarrow \langle \Sigma \vert (X, l, r), i, false, \gamma \cup \{(X, l, r)\} \rangle$
Reduce-Unary-X:	$\langle \Sigma \vert (Y, l, r), i, false, \gamma \rangle \Rightarrow \langle \Sigma \vert (X, l, r), i, false, \gamma \cup \{(X, l, r)\} \rangle$
Finish:	$\langle \Sigma, n, false, \gamma \rangle \Rightarrow \langle \Sigma, n, true, \gamma \rangle$

Figure 2: Transitions of a binary bottom-up constituent parser.

Transition	Σ	Buffer
	[ ]	[The, ... , .]
SH	[The]	[public, ... , .]
SH	[The, public]	[is, ... , .]
RL-NP	[NP]	[is, ... , .]
SH	[NP, is]	[still, ... , .]
SH	[NP, is, still]	[cautious , .]
U-ADVP	[NP, is, ADVP]	[cautious , .]
RR-VP*	[NP, VP*]	[cautious , .]
SH	[NP, VP*, cautious]	[.]
UN-ADJP	[NP, VP*, ADJP]	[.]
RR-VP	[NP, VP]	[.]
SH	[NP, VP, .]	[ ]
RR-S*	[NP, S*]	[ ]
RL-S	[S]	[ ]
FI	[S]	[ ]

Figure 3: Transition sequence for producing the binary constituent tree in Figure 1 using a traditional binary bottom-up parser. SH = Shift, RL-X = Reduce-Left-X, RR-X = Reduce-Right-X, U-X = Reduce-Unary-X and FI = Finish.

(for example, if that particular verb is ditransitive, and subcategorizes for both direct and indirect objects). To achieve that, we use different vector representations for non-terminals VP#2 and VP#3.

4 Dynamic Oracle

Dynamic oracles have been thoroughly studied for a large range of existing transition systems in dependency parsing. However, only a few papers show some progress in constituent parsing (Cross and Huang, 2016b; Coavoux and Crabbé, 2016).

Broadly, implementing a dynamic oracle (Goldberg and Nivre, 2012) consists of defining a *loss function* on configurations, measuring the distance from the best tree they can produce to the gold parse. In that way, we can compute the cost of the new configurations resulting from applying each permissible transition, thus obtaining which transitions will lead the parser to configurations where the minimum number of errors are made.

More formally, given a parser configuration c and a gold tree t_G, a loss function $\ell(c)$ is usually implemented as the minimum Hamming loss between t and t_G, ($\mathcal{L}(t, t_G)$), where t is the already-built tree of a configuration c' reachable from c (written as $c \rightsquigarrow t$). The Hamming loss is computed in dependency parsing as the amount of nodes in t with a different head in t_G. Therefore, the loss function defined as:

$$\ell(c) = \min_{t \vert c \rightsquigarrow t} \mathcal{L}(t, t_G)$$

will compute the cost of a configuration as the Hamming loss of the reachable tree t that differs the minimum from t_G.

A correct dynamic oracle will return the set of transitions τ that do not increase the overall loss (i.e., $\ell(\tau(c)) - \ell(c) = 0$) and, thus, will lead the system through optimal configurations, minimizing the Hamming loss with respect to t_G.

This same idea can be applied in constituent parsing, as done by (Coavoux and Crabbé, 2016), if we redefine the Hamming loss to work with constituents instead of arcs. In this case, $\mathcal{L}(t, t_G)$ is defined as the size of the symmetric difference between the constituents in the trees t and t_G.

To build a correct oracle, we need to find an easily-computable expression for this minimum Hamming loss. For this purpose, we will study constituent reachability. Namely, we will show that this parser has the constituent decomposability property (Coavoux and Crabbé, 2016), analogous to the arc-decomposability property of some dependency parsers (Goldberg and Nivre, 2013). This property implies that $\vert t_G \setminus t \vert$ can be obtained simply by counting the individually unreachable constituents from configuration c, greatly facilitating the definition of the loss function as the other term of the symmetric difference ($\vert t \setminus t_G \vert$) is easy to minimize.

4.1 Constituent Reachability

To reason about constituent reachability and decomposability, we will represent phrase structure trees as a set of constituents. For instance, the non-binary phrase-structure tree in Figure 1 can

Shift:	$\langle \Sigma, i, false, \gamma \rangle \Rightarrow \langle \Sigma \vert (w_i, i, i+1), i+1, false, \gamma \cup \{(w_i, i, i+1)\} \rangle$
Reduce-X#k:	$\langle \Sigma \vert (Y_1, m_0, m_1) \vert ... \vert (Y_k, m_{k-1}, m_k), i, false, \gamma \rangle \Rightarrow \langle \Sigma \vert (X, m_0, m_k), i, false, \gamma \cup \{(X, m_0, m_k)\} \rangle$
Finish:	$\langle \Sigma, n, false, \gamma \rangle \Rightarrow \langle \Sigma, n, true, \gamma \rangle$

Figure 4: Transitions of the novel non-binary bottom-up constituent parser.

Transition	Σ	Buffer
	[ ]	[The, ... , .]
SH	[The]	[public, ... , .]
SH	[The, public]	[is, ... , .]
RE-NP#2	[NP]	[is, ... , .]
SH	[NP, is]	[still, ... , .]
SH	[NP, is, still]	[cautious , .]
RE-ADVP#1	[NP, is, ADVP]	[cautious , .]
SH	[NP, is, ADVP, cautious]	[.]
RE-ADJP#1	[NP, is, ADVP, ADJP]	[.]
RE-VP#3	[NP, VP]	[.]
SH	[NP, VP, .]	[ ]
RE-S#3	[S]	[ ]
FI	[S]	[ ]

Figure 5: Transition sequence for producing the constituent tree in Figure 1 using the proposed non-binary bottom-up parser. SH = Shift, RE-X#k = Reduce-X#k and FI = Finish.

be decomposed as the following set of constituents: {(NP, 0, 2), (VP, 2, 5), (ADVP, 3, 4), (ADJP, 4, 5) and (S, 0, 6)}.

Let γ_G be the set of gold constituents for our current input. We say that a gold constituent $(X, l, r) \in \gamma_G$ is reachable from a configuration $c = \langle \Sigma, j, false, \gamma_c \rangle$ with $\Sigma = [(Y_p, i_p, i_{p-1}) \cdots (Y_2, i_2, i_1) \vert (Y_1, i_1, j)]$, and it is included in the set of *individually reachable constituents* $\mathcal{R}(c, \gamma_G)$, iff it satisfies one of the following conditions:

- $(X, l, r) \in \gamma_c$ (i.e. it has already been created and, therefore, it is reachable by definition).
- $j \leq l < r$ (i.e. it is still in the buffer and can be still created after shifting more words).
- $l \in \{i_k \mid 1 \leq k \leq p\} \wedge j \leq r$ (i.e. its span is completely or partially in the stack, sharing left endpoint with the kth topmost stack constituent, so it can still be built in the future by a transition reducing the stack up to and including that constituent).

The set of *individually unreachable constituents* $\mathcal{U}(c, \gamma_G)$ with respect to the set of gold constituents γ_G can be easily computed as $\gamma_G \setminus \mathcal{R}(c, \gamma_G)$ and will contain the gold constituents that can no longer be built, be it because we have read past their span or because we have created constituents that would overlap with them.

4.2 Loss function

In dependency parsing it is not necessary to count false positives as separate errors from false negatives, as a node attached to the wrong head (false positive) is directly tied to some gold arc being missed (false negative) due to the single-head constraint. In phrase-structure parsing, however, a parser can incur extra false positives by creating erroneous extra constituents, harming precision. For this reason, just like the standard F-score metric penalizes false positives, the loss function must also do so. Hence, as mentioned earlier, we base our loss function in the symmetric difference between reachable and gold constituents.

Thus, our loss function is

$$\ell(c) = \min_{\gamma \vert c \rightsquigarrow \gamma} \mathcal{L}(\gamma, \gamma_G) = \vert \mathcal{U}(c, \gamma_G) \vert + \vert \gamma_c \setminus \gamma_G \vert$$

where the first term ($\vert \mathcal{U}(c, \gamma_G) \vert$) penalizes false negatives (gold constituents that we are guaranteed to miss, as they are unreachable), and the second term ($\vert \gamma_c \setminus \gamma_G \vert$) penalizes false positives (erroneous constituents that have been built).

It is worth mentioning that we cannot directly apply the cost formulation defined by Coavoux and Crabbé (2016) as they rely on the fixed number of constituents in a binary phrase-structure tree to implicitly penalize the prediction of non-gold binary constituents (false positives), and thus only need to explicitly penalize wrong unaries with a dedicated term. In our case, the second term of our loss function is used to penalize the creation of any kind of non-gold constituents, unary or not.

In addition, unlike (Coavoux and Crabbé, 2016), our expression provides the exact loss due to the lack of dummy temporary symbols required by the binarization (they use a traditional binary bottom-up transition system).

4.3 Optimality

We now prove the correctness (or optimality) of our oracle, i.e., that the expression of $\ell(c)$ defined above indeed provides the minimum possible Hamming loss to the gold tree among the trees reachable from a configuration c:

$$\min_{\gamma|c\rightsquigarrow\gamma} \mathcal{L}(\gamma, \gamma_G) = |\mathcal{U}(c, \gamma_G)| + |\gamma_c \setminus \gamma_G|$$

First, we show that the algorithm is constituent-decomposable, i.e., that if each of a set of m constituents is individually reachable from a configuration c, then the whole set also is. We prove this by induction on m. The case for $m = 1$ is trivial. Let us now suppose that constituent-decomposability holds for sets of size up to m, and show that it also holds for an arbitrary set T of $m + 1$ tree-compatible constituents.

Let (X, l, r) be one of the constituents of T such that $l = \min\{l' \mid (X', l', r') \in T\}$ and $r = \min\{r' \mid (X', l, r') \in T\}$. Let $T' = T \setminus (X, l, r)$. T' has m constituents, so by induction hypothesis, it is a reachable set from configuration c.

By hypothesis, (X, l, r) is individually reachable, so at least one of the three conditions in the definition of the individual reachability must hold.

If the first condition holds, then (X, l, r) has already been created in c. Thus, any transition sequence that builds T' (which is a reachable set) starting from c will also include (X, l, r), so $T = T' \cup \{(X, l, r)\}$ is reachable from c.

If the second condition holds, then $j \leq l < r$ and we can create (X, l, r) with $r - j$ Shift transitions followed by one Reduce-X#$(r - l)$ transition. Constituents of T' are still individually reachable in the resulting configuration, because they either have left index l and right index greater than r (and then they meet the third reachability condition) or left index at least r (and then they meet the second). Hence, reasoning as in the previous case, T is reachable from c.

Finally, if the third condition holds, then l is the left endpoint of the kth topmost stack constituent and $r \geq j$. Then, we can create (X, l, r) with $r - j$ Shift transitions followed by one Reduce-X#$(r - j + k)$ transition. By the same reasoning as in the previous case, T is reachable from c.

With this we have shown the induction step, and thus constituent decomposability. This implies that, given a configuration c, there is always some transition sequence that builds all the individually reachable constituents from c, missing only the individually unreachable ones, i.e., $\min_{\gamma|c\rightsquigarrow\gamma} |\gamma_G \setminus \gamma| = |\mathcal{U}(c, \gamma_G)|$.

To conclude the correctness proof, we note that the set $|\gamma \setminus \gamma_G|$ (false positives) will always contain $|\gamma_c \setminus \gamma_G|$ for $c \rightsquigarrow \gamma$ (as the algorithm is monotonic and never deletes constituents); and there exists at least one transition sequence from c that generates exactly the tree $\mathcal{R}(c, \gamma_G) \cup |\gamma_c \setminus \gamma_G|$ (as the algorithm has no situations such that creation of a constituent is needed as a precondition for another). This tree has loss $|\mathcal{U}(c, \gamma_G)| + |\gamma_c \setminus \gamma_G|$, which is thus the minimum loss.

It is worth noting that the correctness of our oracle contrasts with the case of (Coavoux and Crabbé, 2016), whose oracle is only correct under the ideal case where there are no temporary symbols in the grammar, so that they have to resort to a heuristic for the general case.

5 Experiments

5.1 Data

We conduct our experiments on two widely-used benchmarks for evaluating constituent parsers: the Wall Street Journal (WSJ) sections of the English Penn Treebank (Marcus et al., 1993) (Sections 2-21 are used as training data, Section 22 for development and Section 23 for testing) and version 5.1 of the Penn Chinese Treebank (CTB) (Xue et al., 2005) (articles 001- 270 and 440-1151 are taken for training, articles 301-325 for system development, and articles 271-300 for final testing). We use the same predicted POS tags and adopt the same pre-trained word embeddings as (Dyer et al., 2016) and (Liu and Zhang, 2017a).

5.2 Neural Model

We implement our system with greedy decoding under the neural transition-based framework by Dyer et al. (2016), which follows a stack-LSTM approach to represent the stack and the buffer, as well as a vanilla LSTM to represent the action history. We also use a bidirectional LSTM as a compositional function for representing constituents in the stack. Concretely, for the new non-binary bottom-up parser we use the method originally applied in (Dyer et al., 2016) for the top-down algorithm, and adopted by (Liu and Zhang, 2017a) for the in-order parser. This consists of computing

the composition representation s_{comp} as:

$$s_{comp} = (LSTM_{fwd}[e_{nt}, s_0, ..., s_m]; \\ LSTM_{bwd}[e_{nt}, s_m, ..., s_0])$$

where e_{nt} is the vector representation of a non-terminal, and s_i, $i \in [0, m]$ is the ith child node.

Finally, we use exactly the same word representation strategy and hyper-parameter values as (Dyer et al., 2016) and (Liu and Zhang, 2017a) without further optimization.

5.3 Error exploration

Success of dynamic oracles relies on the use of a good exploration strategy. Some recent dependency parsing approaches follow Kiperwasser and Goldberg (2016), who undertake aggressive exploration to increase the impact of error exploration and avoid the early overfitting of the training data by neural networks. In particular, their implementation chooses a non-optimal action when either of these two conditions are satisfied: (1) its score is lower than the score of the optimal one by at least α (i.e. we do not have a strong optimal action and go for a non-optimal one, called aggressive exploration criterion parametrized by a constant margin α) or (2) its score is higher, with probability β (i.e. the non-optimal transition is the highest-scoring one and we follow it, but with a low probability, and we call this regular exploration criterion parametrized by a probability β).

We ran a few experiments on the development set to check which strategies and parameters were more suitable for our dynamic oracle. We started by considering the setting used by (Kiperwasser and Goldberg, 2016) (aggressive exploration with margin 1.0 together with regular exploration with probability 1.0) and, as an alternative, we studied less aggressive approaches that use only regular exploration with a small range of values of β. As shown in Table 1, aggressive exploration yields the highest F-score on the CTB, while a more conservative strategy (regular exploration with probability 0.2) achieves better results on the WSJ.

5.4 Results

Table 2 compares our system's accuracy to other state-of-the-art shift-reduce constituent parsers on the WSJ and CTB benchmarks. Our non-binary bottom-up parser, regardless the kind of oracle used for training, improves over all other bottom-up systems with greedy decoding, as well as the top-down system by Dyer et al. (2016), on both languages. Our approach is only outperformed slightly on the WSJ by the in-order parser by Liu and Zhang (2017a) and is on par with a binary bottom-up parser with beam-search decoding and enhanced with lookahead features (Liu and Zhang, 2017b). In the CTB, our system achieves the best accuracy.

It is worth mentioning that the in-order and bin-

Strategy	WSJ	CTB
aggr-1.0 ∨ reg-0.1	91.83	**89.86**
reg-0.1	91.81	89.70
reg-0.2	**91.88**	89.47
reg-0.3	91.82	89.69

Table 1: F-score comparison of different error-exploration strategies on WSJ §22 and CTB §301-325. Note that "aggr-α" stands for aggressive-exploration and,"reg-β", for regular-exploration.

Parser	Bin	Type	Strat	F1
(Cross and Huang, 2016a)	n	gs	bu	90.0
(Cross and Huang, 2016b)	n	gs	bu	91.0
(Cross and Huang, 2016b)	n	gd	bu	91.3
(Liu and Zhang, 2017a)	y	gs	bu	91.3
This work	n	gs	bu	**91.5**
This work	n	gd	bu	**91.7**
(Zhu et al., 2013)	y	b	bu	90.4
(Watanabe and Sumita, 2015)	y	b	bu	90.7
(Liu and Zhang, 2017b)	y	b	bu	91.7
(Dyer et al., 2016)	n	gs	td	91.2
(Liu and Zhang, 2017a)	n	gs	in	**91.8**

Parser	Bin	Type	Strat	F1
(Wang et al., 2015)	y	gs	bu	83.2
(Liu and Zhang, 2017a)	y	gs	bu	85.7
This work	n	gs	bu	**86.3**
This work	n	gd	bu	**86.8**
(Zhu et al., 2013)	y	b	bu	83.2
(Watanabe and Sumita, 2015)	y	b	bu	84.3
(Liu and Zhang, 2017b)	y	b	bu	85.5
(Dyer et al., 2016)	n	gs	td	84.6
(Liu and Zhang, 2017a)	n	gs	in	86.1

Table 2: Accuracy comparison of state-of-the-art shift-reduce constituent parsers on WSJ §23 (top) and CTB §271-300 (bottom). The "Bin" column marks if prior explicit binarization is required ($y$es/no). The "Type" column shows the type of parser: *gs* is a greedy parser trained with a static oracle, *gd* a greedy parser trained with a dynamic oracle, and *b* a beam search parser. Finally, the "Strat" column describes the strategy followed (bu=bottom-up, td=top-down and in=in-order).

Parser	sent./s.	tran./sent.
Binary Bottom-up	15.71	101.17
Top-down	16.75	119.87
In-order	15.24	121.94
This work	**18.31**	**85.49**

Table 3: Comparison of parsing speed (sentences per second, sent./s.) and transition sequence length per sentence (tran./sent.) for the most common transition systems on WSJ §23, excluding time spent on un-binarization for the binary bottom-up parser. All of them are implemented on the neural framework by (Dyer et al., 2016) and speeds are measured on a single core of an Intel i7-7700 CPU @3.60GHz.

Parser	#1	#2	#3	#4	#5
Bin. bottom-up	90.94	88.65	84.36	77.24	79.54
Top-down	90.98	88.76	85.01	76.63	77.35
In-order	**91.36**	**89.21**	**85.15**	77.08	79.02
This work	91.26	89.09	84.47	**77.51**	**79.77**

Table 4: F-score on constituents with a number of children ranging from one to five on WSJ §23.

ary bottom-up parsers implemented by Liu and Zhang (2017a) and the top-down system by Dyer et al. (2016) can be directly compared to our approach since all of them are implemented under the same framework and trained with the same hyper-parameters as (Dyer et al., 2016).

Regarding the effect of the dynamic oracle, our system benefits more on CTB (0.5 points) than on WSJ (0.2 points), but a wider study of error-exploration strategies might help to increase the final accuracy.

Table 3 reports parsing speeds and transition sequence length per sentence on WSJ §23 of four different transition systems implemented under the same framework by (Dyer et al., 2016). As expected, our parser is the fastest by a considerable margin, since all other transition systems need a longer transition sequence to perform the same task.

5.5 Analysis

We undertake a structure analysis to get insight into why our non-binary system is outperforming the binary version when the latter benefits from a prior binarization (which simplifies the initial problem and provides head information). In Table 4 we present the F-score obtained by each transition system on creating constituents with a number of children ranging from one (unaries) to five. We use the variant of our transition system trained by a static oracle to carry out a fair comparison. From the results, we can state that the proposed non-binary shift-reduce parser improves over the binary one not only on more-than-binary branches, but also in unary and binary structures. It is also worth mentioning that both bottom-up systems perform better on building constituents with four and five children than the state-of-the-art in-order parser, while the top-down transition system achieves the worst results on these structures.

From this simple experiment, we can also see that a binary bottom-up parser tends to be less accurate on unary and binary branches in comparison to a purely top-down strategy, while the latter suffers a drop in F-score when it comes to build constituents with four or more children. This also explains that, by combining both strategies as the in-order algorithm by (Liu and Zhang, 2017a) does, we can build a more accurate parser. It also seems that the non-binary bottom-up transition system alleviates the weaknesses of the binary version on building constituents with a short number of children, while keeping a good accuracy on larger structures.

6 Conclusion

We present, to our knowledge, the first purely non-binary bottom-up shift-reduce constituent parser and we also develop an optimal dynamic oracle for training it.

Except the in-order parser by (Liu and Zhang, 2017a) on the WSJ, it outperforms all other greedy shift-reduce parsers in terms of accuracy with just static training, and matches the second best result on the WSJ when we use a dynamic oracle for training, on par with the system developed by (Liu and Zhang, 2017b), which uses beam search and is enhanced with lookahead features. In addition, our system obtains the highest accuracy on CTB, regardless of the oracle used for training.

Finally, we note that our algorithm is the fastest transition system developed so far for constituent parsing, as it consumes the shortest sequence of transitions to produce phrase-structure trees.

References

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.

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.

James Cross and Liang Huang. 2016a. Incremental parsing with minimal features using bi-directional LSTM. In *ACL (2)*. The Association for Computer Linguistics.

James Cross and Liang Huang. 2016b. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *EMNLP*, pages 1–11. The Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *HLT-NAACL*, pages 199–209. The 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. Association for Computational Linguistics.

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. 2014. Finding the smallest binarization of a CFG is NP-hard. *J. Comput. Syst. Sci.*, 80(4):796–805.

Sylvain Kahane and Nicolas Mazziotta. 2015. Syntactic polygraphs. a formalism extending both constituency and dependency. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 152–164, Chicago, USA. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *EACL (1)*, pages 1249–1258. Association for Computational Linguistics.

Jiangming Liu and Yue Zhang. 2017a. In-order transition-based constituent parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424.

Jiangming Liu and Yue Zhang. 2017b. Shift-reduce constituent parsing with neural lookahead features. *TACL*, 5:45–58.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330.

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. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 49–56.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pages 125–132.

Zhiguo Wang, Haitao Mi, and Nianwen Xue. 2015. Feature optimization for constituent parsing via neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1138–1147.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, ACL 2015, 26-31 July 2015, Bejing, China, Volume 1: Long Papers*, pages 1169–1179.

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. *Nat. Lang. Eng.*, 11(2):207–238.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of 8th International Workshop on Parsing Technologies (IWPT 2003)*, pages 195–206. ACL/SIGPARSE.

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 *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 434–443.