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

Daniel Fernández-González[a,*], Carlos Gómez-Rodríguez[b,]

[a] *Universidade da Coruña, FASTPARSE Lab, LyS Research Group, Departamento de Computación, Campus de Elviña, s/n, 15071 A Coruña, Spain*
[b] *Universidade da Coruña, CITIC. FASTPARSE Lab, LyS Research Group, Departamento de Computación, Campus de Elviña, s/n, 15071 A Coruña, Spain*

Abstract

An increasingly wide range of artificial intelligence applications rely on syntactic information to process and extract meaning from natural language text or speech, with constituent trees being one of the most widely used syntactic formalisms. To produce these phrase-structure representations from sentences in natural language, shift-reduce constituent parsers have become one of the most efficient approaches. Increasing their accuracy and speed is still one of the main objectives pursued by the research community so that artificial intelligence applications that make use of parsing outputs, such as machine translation or voice assistant services, can improve their performance. With this goal in mind, we propose in this article 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, allowing its direct application to any language without the need of further resources such as percolation tables. As a result, it uses fewer transitions per sentence than existing transition-based constituent parsers, becoming the fastest such system and,

*Corresponding author.
Email addresses: d.fgonzalez@udc.es (Daniel Fernández-González), carlos.gomez@udc.es (Carlos Gómez-Rodríguez)

as a consequence, speeding up downstream applications. 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 the Wall Street Journal section from the English Penn Treebank and the Penn Chinese Treebank. 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 the Penn Chinese Treebank among greedy shift-reduce parsers.

Keywords: Automata, natural language processing, computational linguistics, parsing, constituent parsing
2010 MSC: 68T50

---

1. Introduction

Natural Language Processing (NLP) aims to transform unrestricted natural language text into a representation that machines can easily handle to provide applications and services widely used nowadays by our society, such as information extraction, machine translation, sentiment analysis or question answering, among others.

Syntactic parsing is one of these NLP processes that has been in the focus of the research community for the last three decades. This consists in mapping a sentence in natural language into a representation that describes its grammatical structure according to a syntactic formalism. One of the most extended syntactic formalisms is a *constituent* (or *phrase-structure*) representation [3, 8]. Basically, the sentence is decomposed into constituents or phrases and, by creating relationships between words and constituents, a phrase-structure tree is built, like those in Figure 1. A simpler alternative is to use *dependency* representations to describe the syntax of a given sentence. This formalism consists of pairs of words linked by binary and asymmetric relations called *dependencies*, denoting which word is the *head* and which is the *dependent*. Figure 2 depicts an example.

Apart from improving accuracy and coverage, the NLP community is notably interested in increasing parsing speed. Syntactic analysis is a crucial task for numerous

higher-level artificial intelligence applications, such as automatic textual knowledge extraction [4], question answering [5], machine translation [43, 61, 60], automatic summarization [1], relation extraction [64], sentiment analysis [52, 25], information retrieval [27, 17], plagiarism detection [30], name entity recognition [29], among others; and their efficiency is penalized because of the bottleneck resulting from parsers' performance. This is especially dramatic in constituent parsing, whose phrase-based structures are strongly demanded by semantics tasks that involve labeling spans such as semantic role labeling [21, 54], coreference resolution [45, 54], named entity recognition [19] and also some reading comprehension and question answering tasks [50]. Constituent parsing has also been widely used in recent works on syntax-based neural network machine translation such as [16, 2, 36, 57], that demand an efficient syntactic analysis to accomplish their task.

Initially, context-free grammar parsers with production rules derived from data [10, 6, 34, 49] became popular for generating constituent representations. However, while they provide a good accuracy, they require a significant amount of time to undertake the parsing process, becoming impractical for some downstream applications where the execution time is critical. This led the NLP community to use dependency parsers instead. Despite producing a simpler syntactic representation, their efficiency and speed made them gain popularity, especially linear-time transition-based dependency parsers [46, 47, 65, 7, 32, 18] that became the fastest alternative to accurately perform syntactic analysis.

In the past decade, one attempt to speed up constituent parsers was undertaken by using these efficient shift-reduce algorithms to produce more complex structures. Sagae and Lavie [51] 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 [31]. Figure 1 shows a constituent tree and its binarized version that describes a syntactic structure similar to that represented by the dependency formalism in Figure 2 for the same sentence.

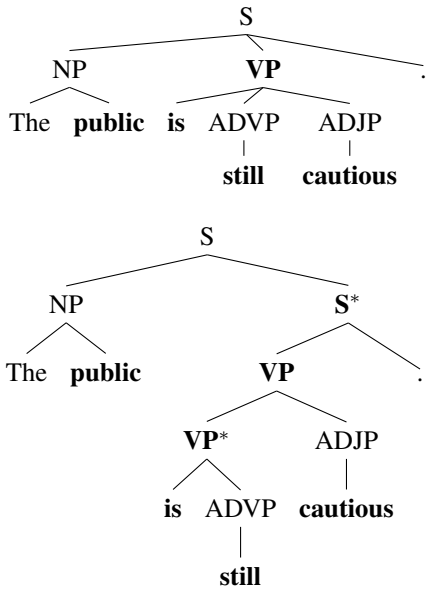

Figure 1: Simplified constituent tree (top) and its binarized equivalent (bottom), taken from English WSJ §22. Head-child nodes are in bold.

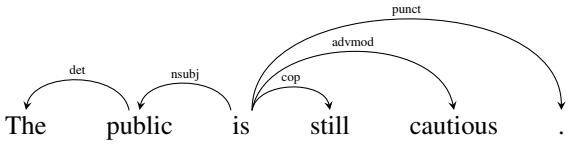

Figure 2: Dependency tree taken from English WSJ §22.

Under this scenario, a transition-based dependency parser such as the *arc-standard* algorithm [63, 48] 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 [66, 67, 59, 42, 11, 9] achieved remarkable accuracy and speed, extending its popularity for constituent parsing.

Recently, some researchers have s explored non-bottom-up strategies to build a

constituent tree. [15, 35] proposed a *top-down* transition-based algorithm and, more recently, Liu and Zhang [37] 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 [37] 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 and we will show in the following sections how we can notably improve it, increasing not only its accuracy, but also its speed, and make it more appealing for higher-level natural language processing applications.[2]

The remainder of this article is organized as follows: Section 2 presents the main weakness of traditional bottom-up parsers and how we successfully address it. In Section 3, we introduce some notation and concepts about shift-reduce parsers. Section 4 describes in detail a novel bottom-up algorithm. In Section 5, a new method for training our approach is presented and studied. Section 6 presents the neural network architecture used to implement and test the proposed system. In Section 7, we evaluate in accuracy and speed our approach and compare it against other state-of-the-art shift-reduce systems, and an analysis of the results is provided. Finally, Section 8 contains a final discussion.

2. Motivation and Proposal

Despite being among the most widely-used and efficient constituent parsers, traditional bottom-up shift-reduce algorithms have a notable drawback compared to other strategies: a binarization as preprocessing is mandatory so that the parser can be applied, and a de-transformation is also required to finally output well-formed phrase-

[2] An alternative to shift-reduce parsing, which also offers a simpler paradigm for constituent parsing with respect to traditional context-free grammar parsers, is offered by sequence-to-sequence models [56, 40, 39]. However, at the moment these parsers still lag behind shift-reduce parsers in terms of accuracy.

structure trees. Apart from the extra time spent, the binarization process needs a *percolation table* or a set of *head rules* of the language, which may not always be available. This additional resource consists of a language-specific and handcrafted set of rules that identify which one among the children in a subtree is the head-child (for instance, the head-child of the subtree $NP \rightarrow The\ public$, present in both trees of Figure 1, is $public$) and, as a result, the application of these efficient and accurate systems is restricted to only those languages with a greater amount of resources.

To overcome this, [12, 13] 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 preprocessing, 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 [24] for context-free grammar parsers.

We think that removing any kind of binarization can benefit bottom-up shift-reduce systems' performance; specially in terms of speed, where this kind of algorithms, despite being one of the fastest constituent alternatives, are notably behind dependency parsers. In this article, 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 Wall Street Journal section from the English Penn Treebank [41] and Penn Chinese Treebank (CTB) benchmarks [62], being the fastest transition system ever created for constituent parsing.

To further improve the accuracy of our system without harming parsing speed, we also develop a *dynamic oracle* for training it. [13, 9] 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, [22] 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 [9] as inspiration, we implement a dynamic oracle for our novel algorithm, achieving notable improvements in accuracy. Unlike [9], 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, while keeping the same parsing speed.

3. Preliminaries

The basic bottom-up transition system of Sagae and Lavie [51], 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 terminal or not, and γ is the set of constituents that have already been built. Each constituent

Shift: $\langle \Sigma, i, false, \gamma \rangle \Rightarrow \langle \Sigma | (w_i, i, i+1), i+1, false, \gamma \cup \{(w_i, i, i+1)\} \rangle$

Reduce-Left/Right-X: $\langle \Sigma | (Y, l, m) | (Z, m, r), i, false, \gamma \rangle \Rightarrow \langle \Sigma | (X, l, r), i, false, \gamma \cup \{(X, l, r)\} \rangle$

Reduce-Unary-X: $\langle \Sigma | (Y, l, r), i, false, \gamma \rangle \Rightarrow \langle \Sigma | (X, l, r), i, false, \gamma \cup \{(X, l, r)\} \rangle$

Finish: $\langle \Sigma, n, false, \gamma \rangle \Rightarrow \langle \Sigma, n, true, \gamma \rangle$

Figure 3: Transitions of a binary bottom-up constituent parser.

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 $(w_i, i, i+1)$). For instance, the non-binary phrase-structure tree on top in Figure 1 can decomposed as the following set of constituents: {(S, 0, 6), (NP, 0, 2), (VP, 2, 5), (ADVP, 3, 4), (ADJP, 4, 5)}.

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 $c_s(w_0 \ldots w_{n-1}) = \langle [\,], 0, false, \{\} \rangle$ and, after applying a sequence of transitions, it ends in a terminal configuration $\langle \Sigma, n, true, \gamma \rangle \in C$, where C is the set of possible terminal configurations.

Figure 3 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 4 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

Transition	Stack	Buffer	Added subtree
	[ ]	[The, ... , .]	
Shift	[The]	[public, ... , .]	
Shift	[The, public]	[is, ... , .]	
Reduce-Left-NP	[NP]	[is, ... , .]	NP→The public
Shift	[NP, is]	[still, ... , .]	
Shift	[NP, is, still]	[cautious , .]	
Reduce-Unary-ADVP	[NP, is, ADVP]	[cautious , .]	ADVP→ still
Reduce-Right-VP*	[NP, VP*]	[cautious , .]	VP*→is ADVP
Shift	[NP, VP*, cautious]	[.]	
Reduce-Unary-ADJP	[NP, VP*, ADJP]	[.]	ADJP→ cautious
Reduce-Right-VP	[NP, VP]	[.]	VP→VP* ADJP
Shift	[NP, VP, .]	[ ]	
Reduce-Right-S*	[NP, S*]	[ ]	S*→VP .
Reduce-Left-S	[S]	[ ]	S→NP S*
Finish	[ ]	[ ]	

Figure 4: Transition sequence for producing the binary constituent tree in Figure 1 using a traditional binary bottom-up parser.

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.

4. 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 [51], 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 $VP \rightarrow is\ ADVP\ ADJP$, 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 5. 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 3) 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 6 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 [15] and the in-order [37] 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

$$\textsf{Shift:} \qquad \langle \Sigma, i, false, \gamma \rangle \Rightarrow \langle \Sigma | (w_i, i, i+1), i+1, false, \gamma \cup \{(w_i, i, i+1)\} \rangle$$

$$\textsf{Reduce-X\#}k\textsf{:} \qquad \langle \Sigma | (Y_1, m_0, m_1) | ... | (Y_k, m_{k-1}, m_k), i, false, \gamma \rangle \Rightarrow$$
$$\langle \Sigma | (X, m_0, m_k), i, false, \gamma \cup \{(X, m_0, m_k)\} \rangle$$

$$\textsf{Finish:} \qquad \langle \Sigma, n, false, \gamma \rangle \Rightarrow \langle \Sigma, n, true, \gamma \rangle$$

Figure 5: Transitions of the novel non-binary bottom-up constituent parser.

circumstances it should create a VP#2 with only two elements (for instance, the verb and its direct object) or three (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.

It is worth mentioning that this straightforward non-binary algorithm could not have been successfully applied on pre-deep-learning techniques, mainly due to the considerable amount of labels that the model has to deal with.

4.1. Time Complexity

Existing shift-reduce transition systems, such as [51, 15, 37], tend to have a linear running time complexity with respect to the length of the input sentence; however, the non-binary algorithm presented here has a quadratic theoretical complexity in the worst case.

To reach this result, we observe that the non-binary transition-based algorithm can parse any sentence with length n by applying n Shift transitions to read every word from the input, plus $|N|$ Reduce transitions to build every subtree (with N being the set of non-terminal nodes in the output tree), plus one Finish transition that ends the process. Therefore, the number of transitions required for analyzing a sentence of length n is exactly $n + |N| + 1$.

To write $|N|$ as a function of n, we assume that the length of chains of unary nodes in constituent trees are bounded by a constant k (without this assumption, constituent trees can be of arbitrary size even if n is bounded, and thus complexity of this or any other constituent parser cannot be bounded). Under this assumption, $|N|$ is at most

Transition	Stack	Buffer	Added subtree
	[ ]	[The, ... , .]	
Shift	[The]	[public, ... , .]	
Shift	[The, public]	[is, ... , .]	
Reduce-NP#2	[NP]	[is, ... , .]	NP→The public
Shift	[NP, is]	[still, ... , .]	
Shift	[NP, is, still]	[cautious , .]	
Reduce-ADVP#1	[NP, is, ADVP]	[cautious , .]	ADVP→ still
Shift	[NP, is, ADVP, cautious]	[.]	
Reduce-ADJP#1	[NP, is, ADVP, ADJP]	[.]	ADJP→ cautious
Reduce-VP#3	[NP, VP]	[.]	VP→is ADVP ADJP
Shift	[NP, VP, .]	[ ]	
Reduce-S#3	[S]	[ ]	S→NP VP .
Finish	[ ]	[ ]	

Figure 6: Transition sequence for producing the constituent tree in Figure 1 using the proposed non-binary bottom-up parser.

$k(n-1)$ (the worst case is a complete binary tree, with $n-1$ nonterminal nodes, which are then expanded with the maximum allowable unary chains) and thus the total number of transitions in the worst case is $n + k(n-1) + 1$, which is $O(n)$.

In addition, let c be the maximum number of children that a non-terminal can have. Then, up to c Reduce transitions with different arity might be evaluated at each step in the worst case. Since c can never be larger than n, in the worst case we have $O(n)$ steps where $O(n)$ transitions need to be evaluated, and thus the worst-case complexity is $O(n^2)$.

In spite of that, in section 7.5, we show empirically that the non-binary algorithm behaves as a linear parser in practice, since in practice c is much smaller than n and behaves like a constant: the number of possible children of a given non-terminal (and, therefore, the number of possible parametrized Reduce transitions that will be evaluated) tends to stay low throughout the training dataset.

5. 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 [13, 9].

Broadly, implementing a dynamic oracle [22] 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:

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

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 [9], 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 [9], analogous to the arc-decomposability property of some dependency parsers [23]. This property implies that $|t_G \setminus t|$ 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 ($|t \setminus t_G|$) is easy to minimize.

5.1. Constituent Reachability

To reason about constituent reachability and decomposability, we will represent phrase structure trees as a set of constituents. As we have seen above, the non-binary phrase-structure tree in Figure 1 can be decomposed as this 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 $c = \langle \Sigma, j, \mathit{false}, \gamma_c \rangle$ with $\Sigma = [(Y_p, i_p, i_{p-1}) \cdots (Y_2, i_2, i_1)|(Y_1, i_1, j)]$, 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.

5.2. 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

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

where the first term ($|\mathcal{U}(c, \gamma_G)|$) penalizes false negatives (gold constituents that we are guaranteed to miss, as they are unreachable), and the second term ($|\gamma_c \setminus \gamma_G|$) 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 [9] 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 [9], 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).

5.3. 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 $r = \min\{r' \mid (X', l', r') \in T\}$ and $l = \max\{l' \mid (X', l', r) \in T\}$. 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, as they either have right index at least r and left index at most l (so that the third reachability condition) or right index greater than r and left index at least r (so 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 [9], 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.

6. Neural Model

We implement our system with greedy decoding under the neural transition-based framework developed by Dyer et al. [15] and reused by Liu and Zhang [37]. This provides a state-of-the-art neural network architecture that proved to be one of the best options to date for implementing transition-based constituent parsers. Another possible alternative could be a BiLSTM-based architecture [32, 12], which has also achieved good accuracy in constituent parsing, as reported by [13].

In particular, the framework introduced by Dyer et al. [15] follows a stack-LSTM [14] approach to represent the stack and the buffer, where each element is the result of a compositional representation of partial trees. In addition, a vanilla LSTM [28] is in charge of representing the action history.

This architecture is trained for greedily and sequentially making local decisions. Given a sentence $w_0, w_1, ..., w_{n-1}$ with n words, and with w_i being the ith word, for the kth parsing state $[c_j, ..., c_1, c_0, i, \mathit{false}]$ the distributional probability of the current

action p is computed as:

$$p = softmax(W\ E_{state} + b),$$

where W and b are model parameters, and E_{state} is the embedding that represents the current parsing state and is obtained by the concatenation and posterior transformation of the representations of elements that compound the transition system as follows:

$$E_{state} = relu(W_s[E_{stack}, E_{buffer}, E_{history}] + b_s)$$

Note that W_s and b_s are model parameters and E_{stack} is the vector representation obtained from the stack-LSTM that stores the constituents currently in the stack:

$$E_{stack} = stack - LSTM[c_0, c_1, ..., c_j],$$

E_{buffer} is the vector representation of the current buffer as:

$$E_{buffer} = stack - LSTM[x_i, x_{i+1}, ..., x_{n-1}],$$

where x_i is a word representation, and finally, the representation of action history is computed as:

$$E_{history} = LSTM[a_{k-1}, a_{k-2}, ..., a_0]$$

where a_t is the vector representation of action in tth parsing state. The detailed neural network architecture is depicted in Figure 7.

Following [15] and [37], the set of parameters θ in the model are trained to minimize a cross-entropy loss objective with an l_2 regularization term defined by:

$$L(\theta) = -\sum_i \sum_j log\ p_{ij} + \frac{\lambda}{2}||\theta||^2$$

where p_{ij} is the probability of the jth action in the ith training example provided by the model and λ is the regularization hyper-parameter. Finally, stochastic gradient descent is used for optimizing θ.

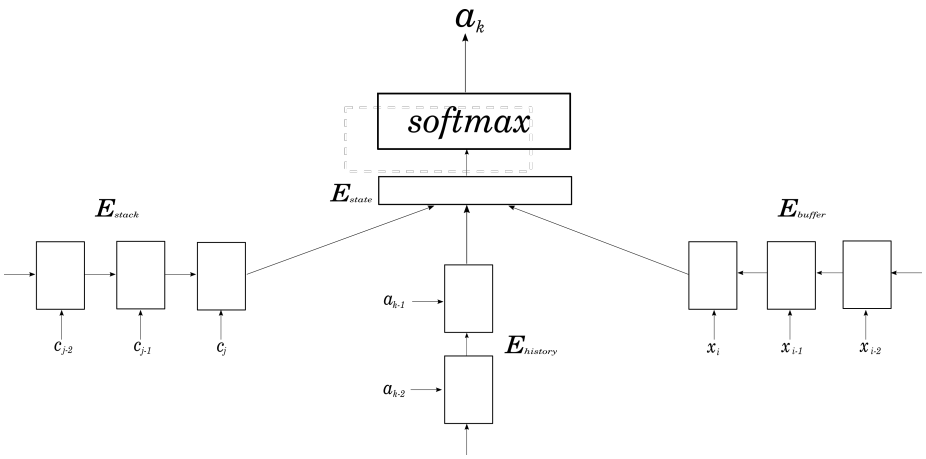

Figure 7: Neural architecture for greedy shift-reduce constituent parsing.

6.1. Word Representation Strategy

We follow the same word representation strategy as described by Dyer et al. [14]. More in detail, we combine three different embeddings: pretrained word embeddings ($e^*_{w_i}$), randomly initialized word embeddings (e_{w_i}) and randomly initialized POS tag embeddings (e_{p_i}). Those embeddings randomly initialized are fine-tuned during the training. After being concatenated the three embeddings, an affine transformation and a posterior non-linear function ReLu is used to derive the final vector representation x_i:

$$x_i = relu(W_e[e^*_{w_i}, e_{w_i}, e_{p_i}] + b_e)$$

where W_e and b_e are model parameters, and w_i and p_i represent the form and the POS tag of the ith input word.

6.2. Syntactic Compositional Function

Following Dyer et al. [15], when a Reduce transition is applied, the parser pops a sequence of completed subtrees and/or words (represented as vector embeddings) from the stack and makes them children of the selected non-terminal, building a new constituent. In order to obtain a vector representation for this new subtree, we use a

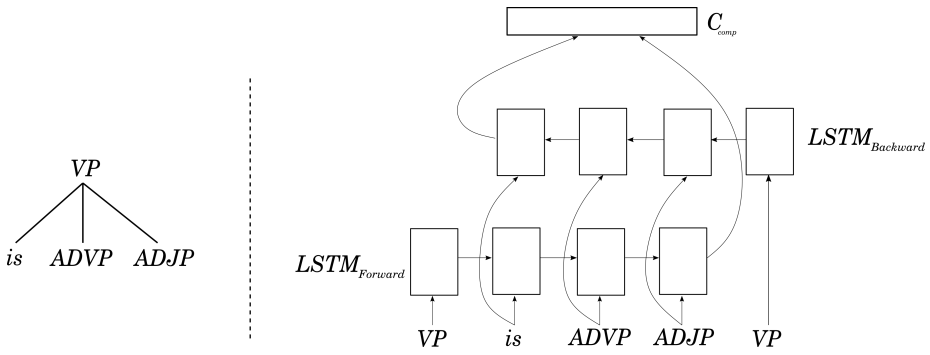

Figure 8: Non-binary syntactic compositional function that reduces a subtree from Figure 1 into a single vector representation c_{comp} by applying a BiLSTM over the tree elements' embeddings.

syntactic compositional function based on bidirectional LSTMs (BiLSTMs) [26]. For the new non-binary bottom-up parser we use the method originally applied in [15] for the top-down algorithm, and adopted by [37] for the in-order parser. More in detail, both LSTMs that compound the BiLSTM read first the selected non-terminal node and, then, the elements from the stack involved in the Reduce transition are introduced in forward direction in one of the LSTMs and, in reverse order, in the other, as shown graphically in Figure 8. After that, the final states of the forward and backward LSTMs are concatenated, and an affine transformation and a ReLu non-linearity are applied to finally compute the composition representation c_{comp}:

$$c_{comp} = relu(W_c[LSTM_{fwd}[e_{nt}, c_0, ..., c_m]; LSTM_{bwd}[e_{nt}, c_m, ..., c_0]] + b_c)$$

where W_c and b_c are model parameters, e_{nt} is the vector representation of the non-terminal on top of the new constituent, and c_i, $i \in [0, m]$ is the vector representation of the ith child node, which might be the vector embedding of a completed subtree or a word.

This differs from the compositional function used for the binary bottom-up parser implemented by [37] under the same framework. More in detail, the composition representation of the new constituent $c_{bincomp}$ for the binary case is computed as:

$$c_{bincomp} = relu(W_c[LSTM_{fwd}[e_{nt}, c_h, c_c]; LSTM_{bwd}[e_{nt}, c_c, c_h]] + b_c)$$

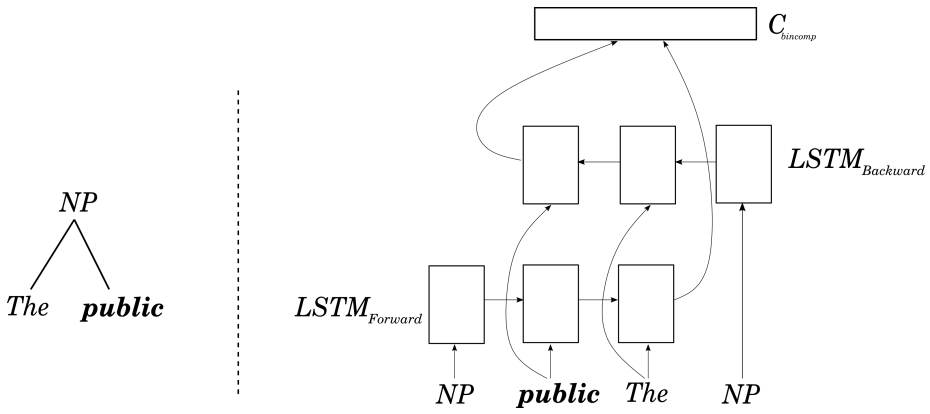

Figure 9: Binary syntactic compositional function to reduce a subtree from Figure 1 into a single vector representation $c_{bincomp}$. Note that the head-child marked in bold is read first by the BiLSTM.

where c_h and c_c denote the representations of the constituents acting as head-child and as a regular child, respectively. Unlike in the non-binary case, in this compositional function we need to use a percolation table to explicitly mark which constituent is the head-child node that must be first in the BiLSTM, adding additional information to the training. Figure 9 depicts graphically how the compositional function works in the binary scenario.

Please note that the proposed transition system and the framework used are orthogonal to approaches like beam search, re-ranking or semi-supervision that might easily improve the final accuracy; however, all these techniques would penalize parsing speed.

7. Experiments

7.1. Data and Settings

We conduct our experiments on two widely-used benchmarks for evaluating constituent parsers: the Wall Street Journal (WSJ) sections of the English Penn Treebank [41] (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) [62] (articles 001-270 and 440-1151 are taken for training, articles 301-325 for system development, and articles 271-300 for final testing).

Hyper-parameter	Value
Initialized learning rate	0.1
Learning rate decay	0.05
λ	10^{-6}
Dropout rate	0.2
Number LSTM layers	2
Stack-LTSM input dimension	128
Stack-LTSM hidden dimension	128
Word embedding dimension	32
POS tag embedding dimension	12
English pretrained word embedding dimension	100
Chinese pretrained word embedding dimension	80
Action embedding dimension	16

Table 1: Hyper-parameters used for all experiments.

We use the same POS tags as [15] and [37] automatically predicted by the Stanford tagger [55] for English and by Marmot [44] for Chinese. The same pre-trained word embeddings are also adopted: those generated on the AFP portion of English Gigaword corpus (version 5) and those extracted from the complete Chinese Gigaword corpus (version 2).

Finally, we use exactly the same hyper-parameter values as [15] and [37] without further optimization. We summarize them in Table 1.

7.2. Error exploration

Success of dynamic oracles relies on the use of a good exploration strategy. Some recent dependency parsing approaches follow Kiperwasser and Goldberg [32], 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:

Strategy	WSJ	CTB
aggr-1.0 $\vee$ 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 2: 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.

(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 [32] (aggressive exploration with margin 1.0 together with regular exploration with probability 0.1) 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 2, 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.

7.3. Results

Table 3 and Table 4 compare 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. [15], on both languages. Our approach is only outperformed slightly on the WSJ

Parser	Bin	Type	Strat	F1
Cross and Huang [12]	n	gs	bu	90.0
Cross and Huang [13]	n	gs	bu	91.0
Cross and Huang [13]	n	gd	bu	91.3
Liu and Zhang [37]	y	gs	bu	91.3
This work	n	gs	bu	**91.5**
This work	n	gd	bu	**91.7**
Zhu et al. [67]	y	b	bu	90.4
Watanabe and Sumita [59]	y	b	bu	90.7
Liu and Zhang [38]	y	b	bu	91.7
Dyer et al. [15]	n	gs	td	91.2
Liu and Zhang [37]	n	gs	in	**91.8**

Table 3: Accuracy comparison of state-of-the-art shift-reduce constituent parsers on WSJ §23. 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, *b* a beam search parser. Finally, the "Strat" column describes the strategy followed (bu=bottom-up, td=top-down and in=in-order).

Parser	Bin	Type	Strat	F1
Wang et al. [58]	y	gs	bu	83.2
Liu and Zhang [37]	y	gs	bu	85.7
This work	n	gs	bu	**86.3**
This work	n	gd	bu	**86.8**
Zhu et al. [67]	y	b	bu	83.2
Watanabe and Sumita [59]	y	b	bu	84.3
Liu and Zhang [38]	y	b	bu	85.5
Dyer et al. [15]	n	gs	td	84.6
Liu and Zhang [37]	n	gs	in	86.1

Table 4: Accuracy comparison of state-of-the-art shift-reduce constituent parsers on CTB §271-300. 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	35.87	50.57
Top-down	38.78	59.76
In-order	33.34	60.83
This work	**42.02**	**42.70**

Table 5: 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 unbinarization for the binary bottom-up parser.

by the in-order parser by Liu and Zhang [37] and is on par with a binary bottom-up parser with beam-search decoding and enhanced with lookahead features [38]. In the CTB, our system achieves the best accuracy.

It is worth mentioning that the in-order and binary bottom-up parsers implemented by Liu and Zhang [37] and the top-down system by Dyer et al. [15] 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 [15].

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 5 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. [15].[3] All of them are implemented on the same neural framework and use the same training settings as [15] and speeds are measured on a single core of an Intel i7-7700 CPU @3.60GHz. 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. Please note that times in decoding are dominated by neural network computations and a negligible fraction of total running time is consumed by the transition-based algorithm. Therefore, we can easily improve shift-reduce parsing performance by reducing the number of transitions, as this is what determines the number of times that the neural model is required for scoring, even if we undertake this by using an algorithm with quadratic worst-case time complexity.

In order to put our approach into context, we provide a comparison against four of the current state-of-the-art systems in performing constituent parsing. In particular, these parsers make use of global chart decoding to achieve the highest accuracies to date on WSJ in cubic running time, with a large cost in speed in comparison to shift-reduce systems. In table 6, we can see how our parser, without further speed optimiza-

[3] Please note that the framework developed by Dyer et al. [15] was not optimized for speed and, therefore, reported speeds are just used for comparison purposes, but they should not be taken as the best speed that a shift-reduce parser (implemented under the described neural network architecture) can potentially achieve.

Parser	F1	sent./s.
This work	91.7	**42.02**
Stern et al. [53]	91.8	22.79
Gaddy et al. [20]	92.1	19.48
Kitaev and Klein [33]	**93.6**	27.77
Kitaev and Klein [33]+ELMO	**95.1**	7.82

Table 6: Accuracy and speed comparison of our approach against current state-of-the-art chart-based constituent parsers on WSJ §23.

tion, is significantly faster than the best chart-based models under the same single core CPU conditions. This difference in speed, as well as the capability of incrementally processing the input from left to right, makes bottom-up shift-reduce parsers appealing for some downstream NLP applications, such as real-time machine translation systems, that require to produce a practically immediate output while the input is still coming.

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

7.4. Structure 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 7 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. This can be explained by the fact that constituents with a large number of children require a longer sequence of transitions to be built and, therefore, are more prone to suffer from error-propagation. Since the in-order and top-down parsers consume a greater amount of transitions than the bottom-up algorithms to produce the exactly same phrase structure, the former are being penalized in larger constituents. This also proves the advantage of the proposed non-binary approach over the binary algorithm on large constituents, since the first manages to reduce the transition sequences required by the second to build a certain tree and, therefore, alleviate the effect of error-propagation.

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 [37] 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.

7.5. Empirical Running Time Complexity

We now show that, in spite of being a worst-case quadratic running time algorithm in theory, our approach performs as a linear parser during decoding in practice. In particular, we measure the time spent by the system to analyze every sentence from WSJ §23 and depict the relation between time consumed and sentence length. We repeat this experiment for the other three transition-based algorithms implemented under the same framework. As shown in Figure 10, a linear behaviour is observed in the non-

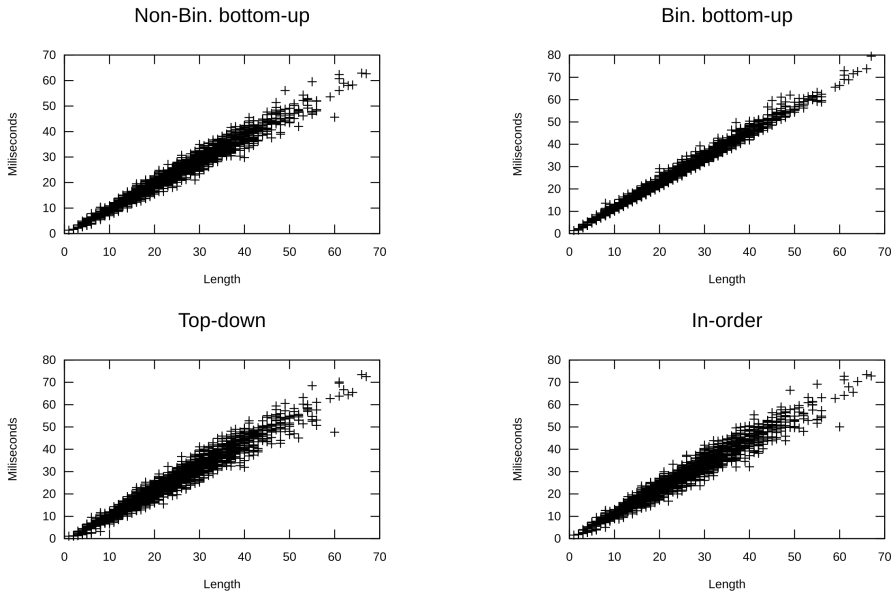

Figure 10: Running time relative to the length of the sentence, for the three mentioned lineal transition-based algorithms plus the novel non-binary shift-reduce parser on the 2,416 sentences from WSJ §23.

binary parser, similarly to the other three linear algorithms, proving that the number of parametrized Reduce transitions, evaluated by the model at each step, is considerably low (behaving practically like a constant).

8. 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. Unlike traditional bottom-up systems, this novel algorithm can be applied to any language without the need of further additional resources, required to perform prior binarization, and, despite being quadratic in theory, it performs as a linear-time parser in practice.

Except the in-order parser by Liu and Zhang [37] 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 [38], 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.

Additionally, 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. In practice, it outspeeds other approaches in a comparison under homogeneous conditions and it will certainly alleviate the bottleneck caused by parsers in NLP applications that rely on syntactic representations.

Finally, we also prove that the novel non-binary algorithm excels in building trees with a large number of children. This is probably due to the fact that our approach requires a shorter number of transitions to build a constituent and, therefore, unlike the other existing transition systems, it is less prone to suffer from the error propagation generated when a long sequence of actions are applied to build these kind of structures.

The parser's source code will be freely available after acceptance.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia (ED431B 2017/01).

References

References

[1] Abdi, A., Idris, N., Alguliev, R.M., Aliguliyev, R.M.. Automatic summarization assessment through a combination of semantic and syntactic information for intelligent educational systems. Information Processing & Management 2015;51(4):340 – 358. URL: http://www.sciencedirect.com/science/article/pii/S0306457315000254. doi:https://doi.org/10.1016/j.ipm.2015.02.001.

[2] Aharoni, R., Goldberg, Y.. Towards string-to-tree neural machine translation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Association for Computational Linguistics; 2017. p. 132–140. URL: http://aclweb.org/anthology/P17-2021. doi:10.18653/v1/P17-2021.

[3] Bloomfield, L.. Language. University of Chicago Press, 1933.

[4] Branavan, S.R.K., Silver, D., Barzilay, R.. Learning to win by reading manuals in a monte-carlo framework. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1. Stroudsburg, PA, USA: Association for Computational Linguistics; HLT '11; 2011. p. 268–277. URL: http://dl.acm.org/citation.cfm?id=2002472.2002507.

[5] Chali, Y., Hasan, S.A., Joty, S.R.. Improving graph-based random walks for complex question answering using syntactic, shallow semantic and extended string subsequence kernels. Information Processing & Management 2011;47(6):843 – 855. URL: http://www.sciencedirect.com/science/article/pii/S0306457310000877. doi:https://doi.org/10.1016/j.ipm.2010.10.002; question Answering.

[6] Charniak, E.. Tree-bank grammars. In: Proceedings of AAAI/IAAI. 1996. p. 1031–1036.

[7] Chen, D., Manning, C.D.. A fast and accurate dependency parser using neural networks. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. 2014. p. 740–750. URL: http://aclweb.org/anthology/D/D14/D14-1082.pdf.

[8] Chomsky, N.. Three models for the description of language. IRE Transactions on Information Theory 1956;IT-2:113–124.

[9] Coavoux, M., Crabbé, B.. Neural greedy constituent parsing with dynamic oracles. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Berlin, Germany: Association for Computational Linguistics; 2016. p. 172–182. URL: http://www.aclweb.org/anthology/P16-1017.

[10] Collins, M.. Three generative, lexicalised models for statistical parsing. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL) and the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL). 1997. p. 16–23.

[11] Crabbé, B.. Multilingual discriminative lexicalized phrase structure parsing. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. Lisbon, Portugal: Association for Computational Linguistics; 2015. p. 1847–1856. URL: http://aclweb.org/anthology/D15-1212.

[12] Cross, J., Huang, L.. 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). Association for Computational Linguistics; 2016. p. 32–37. URL: http://www.aclweb.org/anthology/P16-2006. doi:10.18653/v1/P16-2006.

[13] Cross, J., Huang, L.. 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. Association for Computational Linguistics; 2016. p. 1–11. URL: http://www.aclweb.org/anthology/D16-1001. doi:10.18653/v1/D16-1001.

[14] Dyer, C., Ballesteros, M., Ling, W., Matthews, A., Smith, N.A.. Transition-based dependency parsing with stack long short-term memory. 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). Association for Computational Linguistics; 2015.

p. 334–343. URL: http://www.aclweb.org/anthology/P15-1033. doi:10.3115/v1/P15-1033.

[15] Dyer, C., Kuncoro, A., Ballesteros, M., Smith, N.A.. Recurrent neural network grammars. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics; 2016. p. 199–209. URL: http://www.aclweb.org/anthology/N16-1024. doi:10.18653/v1/N16-1024.

[16] Eriguchi, A., Hashimoto, K., Tsuruoka, Y.. Tree-to-sequence attentional neural machine translation. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics; 2016. p. 823–833. URL: http://aclweb.org/anthology/P16-1078. doi:10.18653/v1/P16-1078.

[17] Fang, L., Tuan, L.A., Hui, S.C., Wu, L.. Syntactic based approach for grammar question retrieval. Information Processing & Management 2018;54(2):184 – 202. URL: http://www.sciencedirect.com/science/article/pii/S0306457317301310. doi:https://doi.org/10.1016/j.ipm.2017.11.004.

[18] Fernández-González, D., Gómez-Rodríguez, C.. 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, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers). 2018. p. 693–700. URL: https://aclanthology.info/papers/N18-2109/n18-2109.

[19] Finkel, J.R., Manning, C.D.. Joint parsing and named entity recognition. In: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics. Stroudsburg, PA, USA: Association for Computational Linguistics; NAACL

'09; 2009. p. 326–334. URL: http://dl.acm.org/citation.cfm?id=1620754.1620802.

[20] Gaddy, D., Stern, M., Klein, D.. What's going on in neural constituency parsers? an analysis. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers). 2018. p. 999–1010. URL: https://aclanthology.info/papers/N18-1091/n18-1091.

[21] Gildea, D., Jurafsky, D.. Automatic labeling of semantic roles. Comput Linguist 2002;28(3):245–288. URL: http://dx.doi.org/10.1162/089120102760275983. doi:10.1162/089120102760275983.

[22] Goldberg, Y., Nivre, J.. A dynamic oracle for arc-eager dependency parsing. In: Proceedings of COLING 2012. Mumbai, India: Association for Computational Linguistics; 2012. p. 959–976. URL: http://www.aclweb.org/anthology/C12-1059.

[23] Goldberg, Y., Nivre, J.. Training deterministic parsers with non-deterministic oracles. Transactions of the Association for Computational Linguistics 2013;1:403–414. URL: http://anthology.aclweb.org/Q/Q13/Q13-1033.pdf.

[24] Gómez-Rodríguez, C.. Finding the smallest binarization of a CFG is NP-hard. J Comput Syst Sci 2014;80(4):796–805. URL: http://dx.doi.org/10.1016/j.jcss.2013.12.003. doi:10.1016/j.jcss.2013.12.003.

[25] Gómez-Rodríguez, C., Alonso-Alonso, I., Vilares, D.. How important is syntactic parsing accuracy? an empirical evaluation on rule-based sentiment analysis. Artificial Intelligence Review 2017;URL: https://doi.org/10.1007/s10462-017-9584-0. doi:10.1007/s10462-017-9584-0.

[26] Graves, A., Schmidhuber, J.. Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks 2005;:5–6.

[27] Higashinaka, R., Kobayashi, N., Hirano, T., Miyazaki, C., Meguro, T., Makino, T., Matsuo, Y.. Syntactic filtering and content-based retrieval of twitter sentences for the generation of system utterances in dialogue systems. 2013. .

[28] Hochreiter, S., Schmidhuber, J.. Long short-term memory. Neural Comput 1997;9(8):1735–1780. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735. doi:10.1162/neco.1997.9.8.1735.

[29] Jie, Z., Muis, A., Lu, W.. Efficient dependency-guided named entity recognition. 2017. URL: https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14741.

[30] K., V., Gupta, D.. Unmasking text plagiarism using syntactic-semantic based natural language processing techniques: Comparisons, analysis and challenges. Information Processing & Management 2018;54(3):408 – 432. URL: http://www.sciencedirect.com/science/article/pii/S0306457317300547. doi:https://doi.org/10.1016/j.ipm.2018.01.008.

[31] Kahane, S., Mazziotta, N.. Syntactic polygraphs. a formalism extending both constituency and dependency. In: Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015). Chicago, USA: Association for Computational Linguistics; 2015. p. 152–164. URL: http://www.aclweb.org/anthology/W15-2313.

[32] Kiperwasser, E., Goldberg, Y.. Simple and accurate dependency parsing using bidirectional LSTM feature representations. TACL 2016;4:313–327. URL: https://transacl.org/ojs/index.php/tacl/article/view/885.

[33] Kitaev, N., Klein, D.. Constituency parsing with a self-attentive encoder. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers. 2018. p. 2675–2685. URL: https://aclanthology.info/papers/P18-1249/p18-1249.

[34] Klein, D., Manning, C.D.. Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL). 2003. p. 423–430.

[35] Kuncoro, A., Ballesteros, M., Kong, L., Dyer, C., Neubig, G., Smith, N.A.. What do recurrent neural network grammars learn about syntax? In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers. Association for Computational Linguistics; 2017. p. 1249–1258. URL: http://aclweb.org/anthology/E17-1117.

[36] Li, J., Xiong, D., Tu, Z., Zhu, M., Zhang, M., Zhou, G.. Modeling source syntax for neural machine translation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics; 2017. p. 688–697. URL: http://aclweb.org/anthology/P17-1064. doi:10.18653/v1/P17-1064.

[37] Liu, J., Zhang, Y.. In-order transition-based constituent parsing. Transactions of the Association for Computational Linguistics 2017;5:413–424. URL: https://www.transacl.org/ojs/index.php/tacl/article/view/1199.

[38] Liu, J., Zhang, Y.. Shift-reduce constituent parsing with neural lookahead features. Transactions of the Association for Computational Linguistics 2017;5:45–58.

[39] Liu, L., Zhu, M., Shi, S.. Improving sequence-to-sequence constituency parsing. 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16347.

[40] Ma, C., Liu, L., Tamura, A., Zhao, T., Sumita, E.. Deterministic attention for sequence-to-sequence constituent parsing. 2017. URL: https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14317.

[41] Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.. Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics 1993;19:313–330.

[42] Mi, H., Huang, L.. 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. Denver, Colorado: Association for Computational Linguistics; 2015. p. 1030–1035. URL: http://www.aclweb.org/anthology/N15-1108.

[43] Miceli Barone, A.V., Attardi, G.. Non-projective dependency-based pre-reordering with recurrent neural network for machine translation. 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). Association for Computational Linguistics; 2015. p. 846–856. URL: http://www.aclweb.org/anthology/P15-1082. doi:10.3115/v1/P15-1082.

[44] Mueller, T., Schmid, H., Schütze, H.. Efficient higher-order crfs for morphological tagging. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics; 2013. p. 322–332. URL: http://www.aclweb.org/anthology/D13-1032.

[45] Ng, V.. Supervised noun phrase coreference research: The first fifteen years. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA, USA: Association for Computational Linguistics; ACL '10; 2010. p. 1396–1411. URL: http://dl.acm.org/citation.cfm?id=1858681.1858823.

[46] Nivre, J.. An efficient algorithm for projective dependency parsing. In: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT). 2003. p. 149–160.

[47] Nivre, J., Hall, J., Nilsson, J.. Memory-based dependency parsing. In: Proceedings of the 8th Conference on Computational Natural Language Learning. 2004. p. 49–56.

[48] Nivre, J., Hall, J., Nilsson, J.. Memory-based dependency parsing. In: Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004). 2004. p. 49–56.

[49] Petrov, S., Klein, D.. Improved inference for unlexicalized parsing. In: Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT). 2007. p. 404–411.

[50] Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.. Squad: 100,000+ questions for machine comprehension of text. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics; 2016. p. 2383–2392. URL: http://aclweb.org/anthology/D16-1264. doi:10.18653/v1/D16-1264.

[51] Sagae, K., Lavie, A.. A classifier-based parser with linear run-time complexity. In: Proceedings of the 9th International Workshop on Parsing Technologies (IWPT). 2005. p. 125–132.

[52] Saif, H., He, Y., Fernandez, M., Alani, H.. Contextual semantics for sentiment analysis of twitter. Information Processing & Management 2016;52(1):5 – 19. URL: http://www.sciencedirect.com/science/article/pii/S0306457315000242. doi:https://doi.org/10.1016/j.ipm.2015.01.005; emotion and Sentiment in Social and Expressive Media.

[53] Stern, M., Andreas, J., Klein, D.. A minimal span-based neural constituency parser. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers. 2017. p. 818–827. URL: https://doi.org/10.18653/v1/P17-1076. doi:10.18653/v1/P17-1076.

[54] Swayamdipta, S., Thomson, S., Lee, K., Zettlemoyer, L., Dyer, C., Smith, N.A.. Syntactic scaffolds for semantic structures. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics; 2018. p. 3772–3782. URL: http://aclweb.org/anthology/D18-1412.

[55] Toutanova, K., Klein, D., Manning, C.D., Singer, Y.. Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1. Stroudsburg, PA, USA: Association for Computational Linguistics; NAACL '03; 2003. p. 173–180. URL: https://doi.org/10.3115/1073445.1073478. doi:10.3115/1073445.1073478.

[56] Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., Hinton, G.. Grammar as a foreign language. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2. Cambridge, MA, USA: MIT Press; NIPS'15; 2015. p. 2773–2781. URL: http://dl.acm.org/citation.cfm?id=2969442.2969550.

[57] Wang, X., Pham, H., Yin, P., Neubig, G.. A tree-based decoder for neural machine translation. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018. 2018. p. 4772–4777. URL: https://aclanthology.info/papers/D18-1509/d18-1509.

[58] Wang, Z., Mi, H., Xue, N.. 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. 2015. p. 1138–1147.

[59] Watanabe, T., Sumita, E.. 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. 2015. p. 1169–1179.

[60] Wu, S., Zhang, D., Yang, N., Li, M., Zhou, M.. Sequence-to-dependency neural machine translation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics; 2017. p. 698–707. URL: http://aclweb.org/anthology/P17-1065. doi:10.18653/v1/P17-1065.

[61] Xiao, T., Zhu, J., Zhang, C., Liu, T.. Syntactic skeleton-based translation. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI Press; AAAI'16; 2016. p. 2856–2862. URL: http://dl.acm.org/citation.cfm?id=3016100.3016301.

[62] Xue, N., Xia, F., Chiou, F.d., Palmer, M.. The penn chinese treebank: Phrase structure annotation of a large corpus. Nat Lang Eng 2005;11(2):207–238. URL: https://doi.org/10.1017/S135132490400364X. doi:10.1017/S135132490400364X.

[63] Yamada, H., Matsumoto, Y.. Statistical dependency analysis with support vector machines. In: Proceedings of 8th International Workshop on Parsing Technologies (IWPT 2003). ACL/SIGPARSE; 2003. p. 195–206.

[64] Yu, M., Gormley, M.R., Dredze, M.. Combining word embeddings and feature embeddings for fine-grained relation extraction. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics; 2015. p. 1374–1379. URL: http://www.aclweb.org/anthology/N15-1155. doi:10.3115/v1/N15-1155.

[65] Zhang, Y., Clark, S.. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). 2008. p. 562–571.

[66] Zhang, Y., Clark, S.. Transition-based parsing of the chinese treebank using a global discriminative model. In: Proceedings of the 11th International Conference on Parsing Technologies. Stroudsburg, PA, USA: Association for Computational Linguistics; IWPT '09; 2009. p. 162–171. URL: http://dl.acm.org/citation.cfm?id=1697236.1697267.

[67] Zhu, M., Zhang, Y., Chen, W., Zhang, M., Zhu, J.. 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. 2013. p. 434–443. URL: http://aclweb.org/anthology/P/P13/P13-1043.pdf.