Left-corner Transitions on Dependency Parsing

Hiroshi Noji and **Yusuke Miyao**
Department of Informatics
The Graduate University for Advanced Studies
National Institute of Informatics, Tokyo, Japan
{noji,yusuke}@nii.ac.jp

Abstract

We propose a transition system for dependency parsing with a left-corner parsing strategy. Unlike parsers with conventional transition systems, such as arc-standard or arc-eager, a parser with our system correctly predicts the processing difficulties people have, such as of center-embedding. We characterize our transition system by comparing its oracle behaviors with those of other transition systems on treebanks of 18 typologically diverse languages. A crosslinguistical analysis confirms the universality of the claim that a parser with our system requires less memory for parsing naturally occurring sentences.

1 Introduction

It is sometimes argued that transition-based dependency parsing is appealing not only from an engineering perspective due to its efficiency, but also from a scientific perspective: These parsers process a sentence incrementally similar to a human parser, which have motivated several studies concerning their cognitive plausibility (Nivre, 2004; Boston and Hale, 2007; Boston et al., 2008). A cognitively plausible dependency parser is attractive for many reasons, one of the most important being that dependency treebanks are available in *many languages*, so it is suitable for crosslinguistical studies of human language processing (Keller, 2010). However, current transition systems based on shift-reduce actions fully or partially employ a bottom-up strategy[1], which is problematic from a psycholinguistical point of view: Bottom-up or top-down strategies are known to fail in predicting the difficulty for certain sentences, such as center-embedding, which people have troubles in comprehending (Abney and Johnson, 1991).

We propose a transition system for dependency parsing with a left-corner strategy. For constituency parsing, unlike other strategies, the arc-eager left-corner strategy is known to correctly predict processing difficulties people have (Abney and Johnson, 1991). To the best of our knowledge, however, the idea of left-corner strategy has not been introduced in the dependency parsing literature. We define the memory cost for a transition system as the number of unconnected subtrees on a stack. Under this condition, the proposed system incurs non-constant memory cost only when encountering center-embedded structures.

After developing the transition system, we characterize it by looking into the following question: Is it true that naturally occurring sentences can be parsed on this system with a lower memory overhead? This should be true under the assumptions that 1) people avoid generating a sentence that causes difficulty for them, and 2) center-embedding is a kind of such structure. Specifically, we focus on analyzing the oracle transitions of the system, i.e., parser actions to recover the gold dependency tree for a sentence. In English, it is known that left-corner transformed treebank sentences can be parsed with less memory (Schuler et al., 2010), but our focus in this paper is on the language universality of the claim in a crosslingual setting. Two different but relevant motivations exist for this analysis. The first is to answer the following scientific question: Is the claim that people tend to avoid generating center-embedded sentences language universal? This is unclear since the observation that a center-embedded sentence is

[1]The top-down parser of Hayashi et al. (2012) is an exception, but its processing is not incremental.

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 2140–2150, Dublin, Ireland, August 23-29 2014.

difficult to comprehend is from psycholinguistic studies mainly on English. The second motivation is to verify whether a parser with the developed system can be viable for crosslinguistical study of human language processing. There is evidence that a human parser cannot store elements of a small constant number, such as three or four (Cowan, 2001). If our system confirms to such a severe constraint, we may claim its cognitive plausibility across languages. We will pursue these questions using the multilingual dependency treebanks from the CoNLL-X and 2007 shared tasks (Buchholz and Marsi, 2006; Nivre et al., 2007).

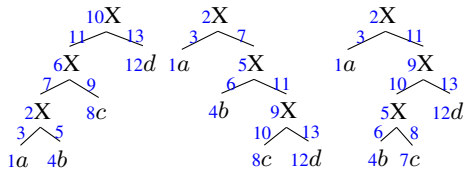
In short, our contributions of this paper can be sketched as follows:

1. We formulate a transition system for dependency parsing with a left-corner strategy.
2. We characterize our transition system with its memory cost by simulating oracle transitions along with other transition systems on the CoNLL multilingual treebanks. This is the first empirical study of required memory for left-corner parsing in a *crosslinguistical* setting.

2 Memory cost of Parsing Algorithms

In this work, we focus on the memory cost for dependency parsing transition systems. While there have been many studies concerning the memory cost for an algorithm in constituency parsing (Abney and Johnson, 1991; Resnik, 1992), the same kind of study is rare in dependency parsing. This section discusses the memory cost for the current dependency parsing transition systems. Before that, we first review the known results in constituency parsing regarding memory cost.

2.1 Center-Embedding and the Left-Corner Strategy



The structures on the right side are called left-branching, right-branching, and center-embedding, respectively. People have difficulty when parsing center-embedded structures, while no difficulty with right-branching or left-branching structures. An example of a center-embedded sentence is *the rat [the cat [the dog chased] bit] ate the cheese*, which is difficult, but if we rewrite it as *the cheese was eaten [by the rat [that bit the cat [that chased the dog]]]*, which is a kind of right-branching structure, the parse becomes easier.

Abney and Johnson (1991) showed that top-down or bottom-up strategies[2] fail to predict this result. For example, for the right-branching structure, a bottom-up strategy requires $O(n)$ memory, since it must first construct a subtree of c and d, but the center-embedded structure requires less memory. The arc-eager left-corner strategy correctly predicts the difficulty of a center-embedded structure, which is characterized by the following order of recognitions of nodes and arcs:

1. A node is enumerated when the subtree of its first child has been enumerated.
2. An arc is enumerated when two nodes it connects have been enumerated.

The numbers on the trees above indicate the order of recognition for this strategy. We can see that it requires a constant memory for both right-branching and left-branching structures. For example, for the right-branching structure, it reaches 7 after reading b, which means that a and b are connected by a subtree. On the other hand, for the center-embedded structure, it reaches 6 after reading b, but a and b cannot be connected at this point, requiring extra memory.

2.2 Transition-based Dependency Parsing

Next, we summarize the issues with current transition systems for dependency parsing with regards to their memory cost. A transition-based dependency parser processes a sentence on a transition system, which is defined as a set of configurations and a set of transitions between configurations (Nivre, 2008). Each configuration has a stack preserving constructed subtrees on which we define the memory cost as a function for each system.

[2]We should distinguish between two types of characterizations of parsing: *strategy* and *algorithm*. A parsing strategy is an abstract notion that defines "a way of enumerating the nodes and arcs of parse trees" (Abney and Johnson, 1991), while a parsing algorithm defines the implementation of that strategy, typically with push-down automata (Johnson-Laird, 1983; Resnik, 1992). A parsing strategy is useful for characterizing the properties of each parser, and we concentrate on the *strategy* for exposition of constituency parsing. For dependency parsing, we mainly discuss the algorithm, i.e., the transition system.

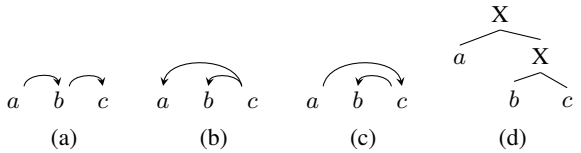


Figure 1: (a)–(c): Right-branching dependency trees for three words; (d): the corresponding CNF.

	Arc-standard	Arc-eager	Left-corner
left-branching	$O(1)$	$O(1)$	$O(1)$
right-branching	$O(n)$	$O(1 \sim n)$	$O(1)$
center-embedding	$O(n)$	$O(1 \sim n)$	$O(n)$

Table 1: Order of the memory cost for each structure for each transition system. $O(1 \sim n)$ means that it processes some structures with a constant cost but some with a non-constant cost.

Arc-Standard We define the memory cost as the number of elements on the stack since all stack elements are disjoint. In this setting, we can see that the arc-standard system has a problem for a right-branching structure, such as $a \curvearrowright b \curvearrowright c \curvearrowright \cdots$, in which the system first pushes all words on the stack before connecting each pair of words, requiring $O(n)$ memory. Nivre (2004) discussed the problem with this system in greater detail, observing that its stack size grows when processing structures that become right-branching when converted to the Chomsky normal form (CNF) of a context-free grammar (CFG). Figure 1 lists those dependency structures for three words, for which the system must construct a subtree of b and c before connecting a to either, requiring extra memory. This is because the system builds a tree bottom-up: each token collects all dependents before being attached to its head. In fact, the arc-standard system is essentially equivalent to the push-down automaton of a CFG in the CNF with a bottom-up strategy (Nivre, 2004), so it has the same property as the bottom-up parser for a CFG.

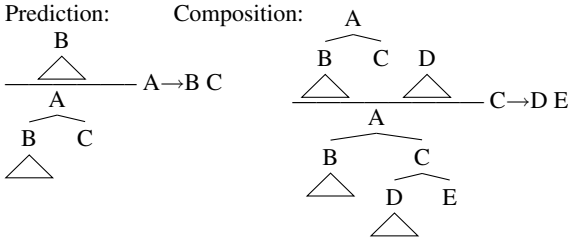
Arc-Eager In the arc-eager system, the stack contains sequences of tokens comprising connected components, so we can define the memory cost as the number of connected components on the stack. With this definition, we can partially resolve the problem with the arc-standard system. The arc-eager system does not incur any cost for processing the structure in Figure1(a) and $a \curvearrowright b \curvearrowright c \curvearrowright \cdots$ since it can connect all tokens on the stack (Nivre, 2004). Because its construction is no longer pure bottom-up, it is difficult to formally characterize the cost based on the type of tree structure. However, this transition system cannot correctly predict difficulties with center-embedding because the cost never increases as long as all dependency arcs are left-to-right, e.g., a sentence $a \curvearrowright b \overset{\frown}{c\ d}$ becomes center-embedding when converted to a CNF, but it does not incur any cost for it. Note that this system still incurs cost for some right-branching structures, such as in Figures 1(b–c), and some center-embedded structures. Therefore, for the arc-eager system, it is complicated to discuss the required order of memory cost. We summarize these results in Table 1. Our goal is to develop an algorithm with the properties of the last column, requiring non-constant memory for only center-embedded structures.

Other systems All systems where stack elements cannot be connected have the same problem as the arc-standard system because of their bottom-up constructions, including the hybrid system of Kuhlmann et al. (2011). Kitagawa and Tanaka-Ishii (2010) and Sartorio et al. (2013) present an interesting variant, which attaches a node to another node that may not be the head of a subtree on the stack. We can use the same reasoning for the arc-eager system for these systems: they sometimes do not incur costs for center-embedded structures, while they incur a non-constant cost for some right-branching structures.

3 Left-corner Dependency Parsing

We now discuss the construction of our transition system with the left-corner *strategy*. Resnik (1992) proposed a push-down recognizer for a CFG. In the following, we instead characterize his algorithm by inference rules, which are more intuitive and helpful to adapt the idea for dependency parsing.

Prediction and Composition There are two characteristic operations in the push-down recognizer of Resnik (1992): **prediction** and **composition**. We show inference rules of these operations on the right side:



Prediction is used to predict the parent node and the sibling of a recognized subtree when the sub-

SHIFT	$(\sigma, j\|\beta, A) \mapsto (\sigma\|\langle j\rangle, \beta, A)$
INSERT	$(\sigma\|\langle\sigma_1'\|i\|x(\lambda)\rangle), j\|\beta, A) \mapsto (\sigma\|\langle\sigma_1'\|i\|j\rangle, \beta, A \cup \{(i,j)\} \cup \{\cup_{k\in\lambda}(j,k)\})$
LEFT-PRED	$(\sigma\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle x(\sigma_{11})\rangle, \beta, A)$
RIGHT-PRED	$(\sigma\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle\sigma_{11}, x(\emptyset)\rangle, \beta, A)$
LEFT-COMP	$(\sigma\|\langle\sigma_2'\|x(\lambda)\rangle\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle\sigma_2'\|x(\lambda \cup \{\sigma_{11}\})\rangle, \beta, A)$
RIGHT-COMP	$(\sigma\|\langle\sigma_2'\|x(\lambda)\rangle\|\langle\sigma_{11}, \cdots\rangle, \beta, A) \mapsto (\sigma\|\langle\sigma_2'\|\sigma_{11}\|x(\emptyset)\rangle, \beta, A \cup \{\cup_{k\in\lambda}(\sigma_{11}, k)\})$

Figure 2: Actions of the left-corner transition system.

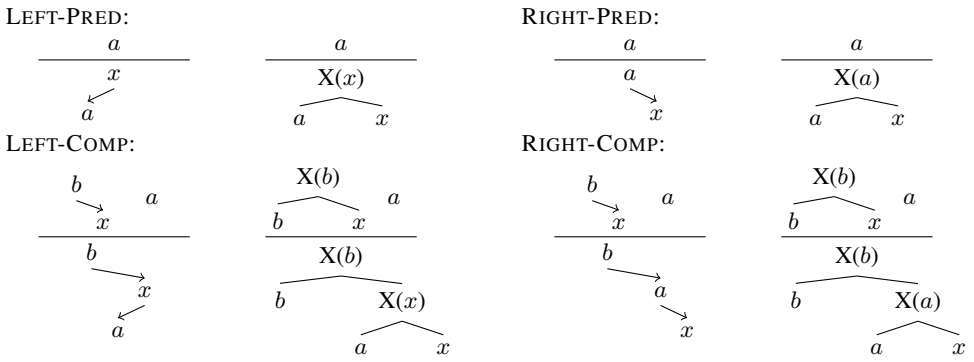
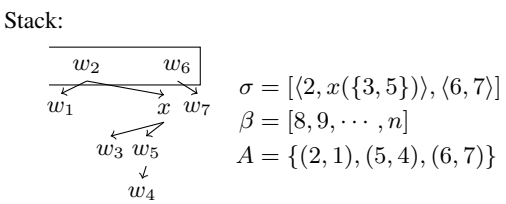


Figure 3: Correspondences of reduce actions between dependency and CFG. Nonterminal X(t) means that its lexical head is t. We only show minimal example subtrees for simplicity. However, a can have an arbitrary number of children, so can b or x, as long as x is on a right spine and has no right children.

tree is complete and its parent node is not yet recognized. **Composition** composes two subtrees by first predicting the parent and the sibling of a recognized subtree then immediately connecting trees by identifying the same node on two trees (C, in this case). This is used when the parent node of a completed subtree has already been predicted as a part of another tree in a top-down fashion.

Dummy Node We now turn to the discussion of dependency parsing. The key characteristic of our transition system is the introduction of a dummy node on a subtree, which is needed to represent a subtree containing some predicted structures as in constituency subtrees for Resnik's recognizer. To get an intuition of the parser actions, we present a simulation of transitions for the sentence in Figure 1(b), of which current systems fail to predict its difficulty. Our system first shifts a then conducts a kind of **prediction** operation, resulting in a subtree $a \swarrow x$, where x is a dummy node. This means that we predict that a will become a left dependent of an incoming word. Next, it shifts b to the stack then conducts a **composition** operation to obtain a tree $a \swarrow b \swarrow x$. It finally inserts c to the position of x, recovering the tree.

Transition system As in many other transition systems, a configuration for our system is a tuple $c = (\sigma, \beta, A)$, where σ is a stack, and we use a vertical bar to signify an append operation, e.g., $\sigma = \sigma'|\sigma_1$ denoting σ_1 is the top most element of the stack σ, and β is an input buffer consisting of token indexes not processed yet. $\beta = j|\beta'$ means j is a first element of β, and $A \subseteq V_w \times V_w$ is a set of arcs given V_w, a set of token indexes for a sentence w.



Each element of a stack is a list representing a **right spine** of a subtree, as in Kitagawa and Tanaka-Ishii (2010) and Sartorio et al. (2013). A right spine $\sigma_i = \langle\sigma_{i1}, \sigma_{i2}, \cdots, \sigma_{ik}\rangle$ consists of all nodes in a descending path from the head of σ_i, i.e., σ_{i1}, taking the rightmost child at each step. We also write $\sigma_i = \sigma_i'|\sigma_{ik}$ meaning that σ_{ik} is the right most node of spine σ_i. Each element of σ_i is an index of a token in a sentence, or a dummy node $x(\lambda)$, where λ is a set of the left dependents of x. The figure above depicts an example of the configuration, where the i-th word in a sentence is written as w_i on the stack.

In the following, we say a right spine σ_i is *complete* if it does not contain any dummy nodes, while σ_i containing a dummy node is referred to as *incomplete*. Our transition system uses six actions, two of

which are **shift** actions and four are **reduce** actions. All actions are defined in Figure 2.

Shift Actions There are two kinds of shift actions: SHIFT and INSERT[3]. SHIFT moves a token from the top of the buffer to the stack. INSERT replaces a dummy node on the top of the stack with a token from the top of the buffer. This adds arcs from/to tokens connected to the dummy node. Note that this action can be performed for a configuration where $x(\lambda)$ is the top of σ_1 or λ is empty, in which case arcs (i, j) or $\cup_{k\in\lambda}(j, k)$ are not added. Resnik (1992) does not define this action, but instead uses a verification operation (Rule 9). One can view our INSERT action as a composition of two actions: SHIFT and a verification. We note that after these shift actions, the top element of the stack must be complete.

Reduce Actions Reduce actions create new arcs for subtrees on the stack. LEFT-PRED and RIGHT-PRED correspond to the predictions of the CFG counterpart. Figure 3 describes these transitions for minimal subtrees. LEFT-PRED assigns a dummy node x as the head of a (this corresponds to σ_{11}), while RIGHT-PRED creates x as a new right dependent. When we convert the resulting tree into a CNF, we can see that the difference between these two operations lies in the predicted parent node of a: LEFT-PRED predicts a nonterminal X(x), i.e., it predicts that the head of this subtree is the head of the predicting sibling node, while RIGHT-PRED predicts that the head is a. Note that different from CFG rules, we do not have to predict the actual sibling node; rather, we can abstract this predicted node as a dummy node x. A similar correspondence holds between the composition actions: RIGHT-COMP and LEFT-COMP.

We note that to obtain a valid tree, shift and reduce actions must be performed alternatively. We can prove this as follows: Let $c = (\sigma|\sigma_2|\sigma_1, \beta, A)$. Since reduce actions turn an incomplete σ_1 into a complete subtree, we cannot perform two consecutive reduce actions. Shift actions make σ_1 complete. After a shift action, we cannot perform INSERT since it requires σ_i to be incomplete; if we perform SHIFT, the top two elements on the stack become complete, but we cannot connect these two trees since the only way to connect two trees on the stack is composition, but this requires σ_2 to be incomplete.

Defining Oracle The oracle for a transition system is a function that returns a correct action given the current configuration and a set of gold arcs. It is typically used for training a parser (Nivre, 2008), but we define it to analyze the behavior of our system on treebank sentences.

First, we show that our system has the **spurious ambiguity**, and discuss its implications. Consider a sentence $a^\frown b^\frown c$, which can be parsed with two different action sequences as follows:

1. SHIFT → LEFT-PRED → INSERT → RIGHT-PRED → INSERT
2. SHIFT → LEFT-PRED → SHIFT → RIGHT-COMP → INSERT

The former INSERTs b at step 3, then RIGHT-PREDs to wait for a right dependent (c). The latter, on the other hand, SHIFTs b at step 3, then RIGHT-COMPs to combine two subtrees ($a^\frown x$ and b) to obtain a tree $a^\frown b^\frown x$. These ambiguities between action sequences and the resulting tree are referred to as the spurious ambiguity. Next, we analyze the underlying differences between these two operations. We argue that the difference lies in the form of the recognized constituency tree: The former RIGHT-PREDs at step 4, which means that it recognizes a constituency of the form $((a\ b)\ c)$, while the latter recognizes $(a\ (b\ c))$ due to its RIGHT-COMP operation. Therefore, the spurious ambiguity of our system is caused by the ambiguity of converting a dependency tree to a constituency tree. Recently, some transition systems have exploited similar ambiguities using *dynamic oracles* (Goldberg and Nivre, 2013; Sartorio et al., 2013; Honnibal et al., 2013). The same type of analysis might be possible for our system, but we leave it for future work; here we only present a *static oracle* and discuss its properties.

Since our system performs shift and reduce actions interchangeably, we need two functions to define the oracle. Let $c = (\sigma|\sigma_2|\sigma_1, \beta, A)$. The next shift action is determined as follows:

- INSERT: if $\sigma_1 = \langle\sigma_1'|i|x(\lambda)\rangle$ and $(i, j) \in A_g$ and j has no dependents in β (if i exists) or $\exists k \in \lambda; (j, k) \in A_g$ (otherwise).
- SHIFT: otherwise.

The next reduce action is determined as follows:

[3]We use small caps to refer to a specific action, e.g., SHIFT, while "shift" refers to an action type.

- LEFT-COMP: if $\sigma_2 = \langle \sigma_2' | i | x(\lambda) \rangle, \sigma_1 = \langle \sigma_{11}, \cdots \rangle$, σ_{11} has no dependents in β, and σ_{11} can be a left dependent of x: i's next dependent is the head of σ_{11} (if i exists) or $k \in \lambda$ and σ_{11} share the same head (otherwise).
- RIGHT-COMP: if $\sigma_2 = \langle \sigma_2' | i | x(\lambda) \rangle, \sigma_1 = \langle \sigma_{11}, \cdots \rangle$, σ_{11} has one more dependent in β, and σ_{11} can be insertable at the position of x: $(i, \sigma_{11}) \in A_g$ or $\exists k \in \lambda; (\sigma_{11}, k) \in A_g$.
- RIGHT-PRED: if $\sigma_1 = \langle \sigma_{11}, \cdots \rangle$ and σ_{11} has one more dependent in β.
- LEFT-PRED: otherwise.

Each condition checks whether we can obtain the gold dependency arcs after the transition. This oracle follows the strategy of "compose or insert when possible". As we saw in the example, sometimes INSERT and SHIFT both can be valid to recover the gold arcs; however, we always select INSERT. Sometimes the same ambiguity occurs between LEFT-COMP and LEFT-PRED or RIGHT-COMP and RIGHT-PRED, but we always prefer composition.

As we saw above, the spurious ambiguity of our system occurs when the conversion from a dependency tree to a constituency tree is not deterministic. This oracle has the property that its recognized constituency tree corresponds to the one that can be obtained by constructing all left-arcs first given a dependency tree. For example, in the above example for $a \overset{\curvearrowleft}{b} \overset{\curvearrowright}{c}$, we select action sequences 1 to recognize a constituency of $((a\ b)\ c)$. We can prove this property by showing that the algorithm always collects all left-arcs for a head before any right-arcs; Not doing INSERT or composition when possible means that we create a right-arc for a head when the left-arcs are not yet completed. We can also verify that this algorithm can parse all no-center-embedded sentences in a CNF converted in this manner with the stack depth never exceeding three, requiring non-constant memory for only center-embedded structures.

4 Memory Cost Analysis

To characterize our transition system, we compare it to other systems by observing the incurred memory cost during running oracle transitions for sentences on a set of typologically diverse languages. For this analysis, we aim to verify the language universality of the claim: naturally occurring sentences should be parsed with a left-corner parser with less required memory.

Settings We collect 18 treebanks from the CoNLL-X and 2007 shared tasks (Buchholz and Marsi, 2006; Nivre et al., 2007). Some languages were covered by both shared tasks; we use only 2007 data. We remove sentences with non-projective arcs (Nivre, 2008) or without any root nodes. We follow the common practice adding a dummy root token to each sentence. This token is placed at the end of each sentence, as in Ballesteros and Nivre (2013), since it does not change the cost on sentences with one root token on all systems.

We compare three transition systems: arc-standard, arc-eager, and left-corner. For each system, we perform oracle transitions for all sentences and languages, measuring the memory cost for each configuration defined as follows. For the arc-standard and left-corner systems, we use the number of elements on the stack. This arc-standard system uses the original formulation of Nivre (2003), connecting two items on the stack at the reduce action. For the arc-eager system, we use the number of connected components. The system can create a subtree at the beginning of a buffer, in which case we add 1 to the cost.

We run a static oracle for each system. For the left-corner system, we implemented the algorithm presented in Section 3. For the arc-standard and arc-eager systems, we implemented an oracle preferring reduce actions over shift, which can minimizes the memory cost.

Memory costs for general sentences For each language, we count the number of configurations for each memory cost during performing oracles on all sentences. In Figure 4, we show the cumulative frequencies of configurations having each memory cost (see solid lines in the figure). These lines can answer the question: What memory cost is required to cover X% of configurations when recovering all gold trees? Note that comparing absolute values are not meaningful since the minimal cost to construct an arc is different for each system, e.g., the arc-standard system requires at least two items on the stack,

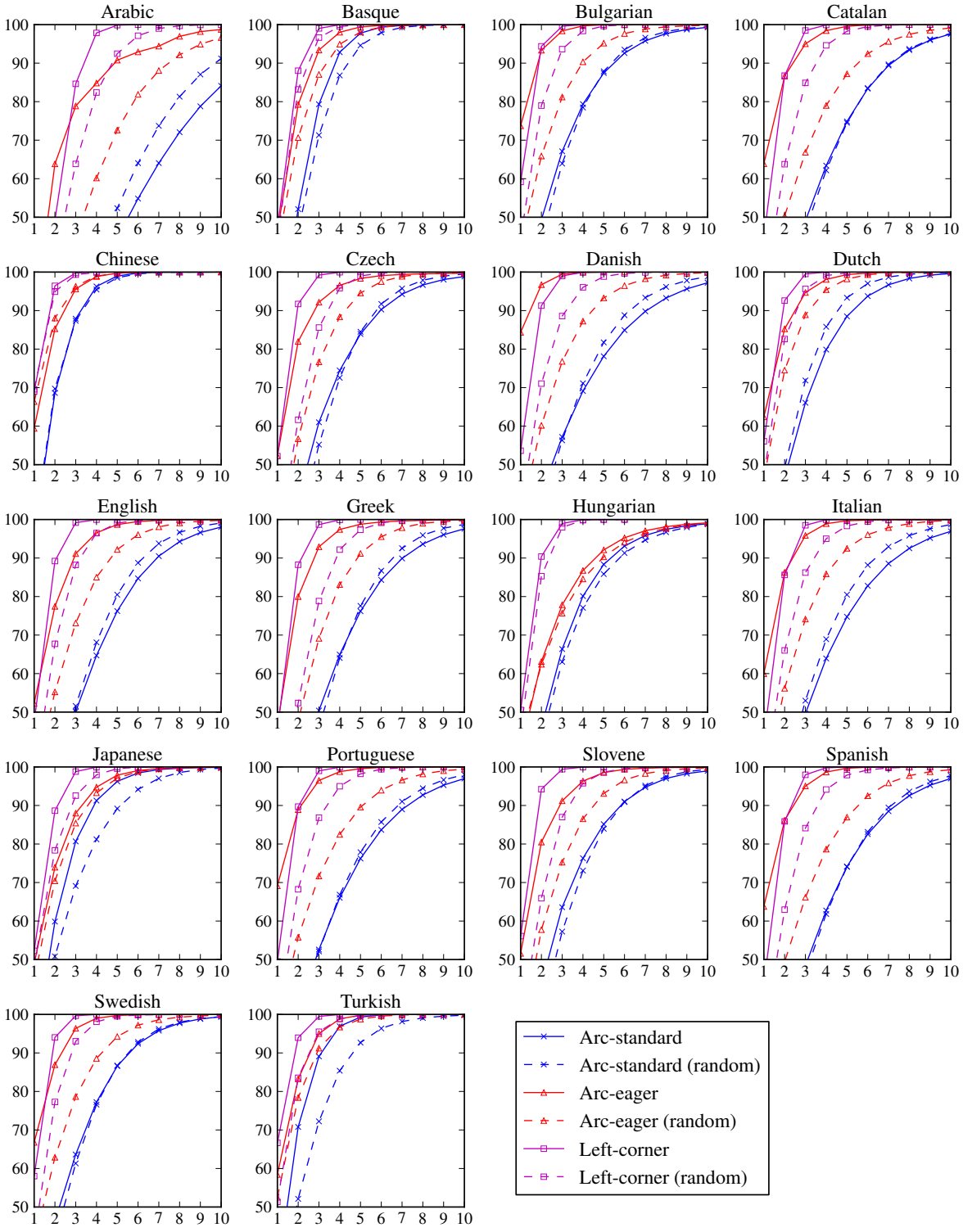


Figure 4: Crosslinguistic comparison of cumulative frequency of memory cost when processing all sentences for each system. For example, in Arabic, for the arc-eager system, around 90% of all configurations incurred memory cost of ≤ 5. Dotted lines (random) are results on the sentences, which are randomly reordered while preserving the graph structure and projectivity.

while the arc-eager system can create a right arc if the stack contains one element. Instead, we focus on the universality of each system's behavior for different languages.

As we discussed in section 2.2, the arc-standard system can process only left-branching structures with a constant memory, which are typical in head-final languages such as Japanese or Turkish, and we can

see this tendency. The system behaves poorly in many other languages.

The arc-eager and left-corner systems behave similarly for many languages, but we can see that there are some languages for which the left-corner system behaves similarly to other languages, while the arc-eager system requires larger cost; Arabic, Hungarian, or Japanese, for example. In fact, except Arabic, the left-corner system reaches 98% of configurations with a memory cost of ≤ 3, which indicates that the property of the left-corner system requiring less memory is more universal than that of other systems.

Comparison to randomized sentences One might wonder that the results above come from the nature of left-corner parsing reducing the stack size, not from the bias in language avoiding center-embedded structures. To partially answer this question, we conduct another experiment comparing oracle transitions on original treebank sentences and on wrong sentences. We create these wrong sentences by using the method from Gildea and Temperley (2007). We reorder words in each sentence by first extracting a directed graph then randomly reordering the children of each node while preserving projectivity. The dotted lines in Figure 4 denotes the results of randomized sentences for each system.

There are notable differences in required memory between original and random sentences for many languages. This result indicates that our system can parse with less memory for only naturally occurring sentences. For Chinese and Hungarian, the differences are subtle. However, the differences are also small for the other systems, which implies that these corpora have some biases on graphs reducing the differences.

5 Related Work and Discussion

To the best of our knowledge, parsing with a left-corner strategy has only been studied for constituency. Roark (2001) proposed a top-down parser for a CFG with a left-corner grammar transform (Johnson, 1998), which is essentially the same as left-corner parsing but enables several extensions in a unified framework. Roark et al. (2009) studied the psychological plausibility of Roark's parser, observing that it fits well to human reading time data. Another model with a left-corner strategy is Schuler et al. (2010): they observed that the transformed grammar of English requires only limited memory, proposing a finite state approximation with a hierarchical hidden Markov model. This parser was later extended by van Schijndel and Schuler (2013), which defined a left-corner parser for constituency with shift and reduce actions. In fact, they used the same kind of actions as our transition system: shift, insert, predict, and composition. Though they did not mentioned explicitly, we showed how to construct a left-corner parsing algorithm with these actions by decomposing the push-down recognizer of Resnik (1992). These are examples of broad-coverage parsing models with cognitively plausibility, which has recently received considerable attention in interdisciplinary research on psycholinguistics and computational linguistics (Schuler et al., 2010; Keller, 2010; Demberg et al., 2013).

Differently from previous models, our target is dependency. A dependency-based cognitively plausible model is attractive, especially from a crosslinguistical viewpoint. Keller (2010) argued that current models only work for English, or German in few exceptions, and the importance of crosslinguistically valid models of human language processing. There has been some attempts to use a transition system for studying human language processing (Boston and Hale, 2007; Boston et al., 2008), so it is interesting to compare automatic parsing behaviors with various transition systems to human processing.

We introduced a dummy node for representing a subtree with an unknown head or dependent. Recently, Menzel and colleagues (Beuck and Menzel, 2013; Kohn and Menzel, 2014) have also studied dependency parsing with a dummy node. While conceptually similar, the aim of introducing a dummy node is different between our approach and theirs: We need a dummy node to represent a subtree corresponding to that in Resnik's algorithm, while they introduced it to confirm that every dependency tree on a sentence prefix is fully connected. This difference leads to a technical difference; a subtree of their parser can contain more than one dummy node, while we restrict each subtree to containing only one dummy node on a right spine.

Our experiments in section 4 can be considered as a study on functional biases existing in language or language evolution (Jaeger and Tily, 2011). In computational linguistics, Gildea and Temperley (2007; 2010) examined the bias on general sentences called *dependency length minimization* (DLM), which

argues that grammar should favor the dependency structures that reduce the sum of dependency arc lengths. They reordered English and German treebank sentences with various criteria: original, random with projectivity, and optimal that minimizes the sum of dependency lengths. They observed that the word order of English fits very well to the optimal ordering, while German does not. We examined the universality of the bias to reduce memory cost for left-corner parsing. Although we cannot compare the incurred cost with the *optimal* reordered sentences, our results on original sentences, in which there are few configurations requiring the stack depth ≥ 4, suggest the bias to avoid center-embedded structures is language universal. It will be interesting to analyze in more detail the relationships between DLM and the bias of our system since the two biases are not independent, e.g., center-embed structures typically appear with longer dependencies. Are there languages that do not hold DLM while requiring less memory, or vice versa? For these analyses, we might have to take care of the grammar construction, e.g., there are several definitions for coordination structures for dependency grammars (Popel et al., 2013). The functional views discussed above might shed some light on the desired construction for these cases.

6 Conclusion

We have pointed out that the memory cost on current transition systems for dependency parsing do not coincide with observations in people, proposing a system with a left-corner strategy. Our crosslinguistical analysis confirms the universality of the claim that people avoid generating center-embedded sentences, which also suggests that it is worthy for crosslinguistical studies of human language processing.

As a next stage, we are seeking to train a parser model as in other transition systems with a discriminative framework such as a structured perceptron (Zhang and Nivre, 2011; Huang and Sagae, 2010). A parser with our transition system might also be attractive for the problem of grammar induction, where recovering dependency trees are a central problem (Klein and Manning, 2004), and where some linguistic biases have been exploited, such as reducibility (Mareček and Žabokrtský, 2012) or acoustic cues (Pate and Goldwater, 2013). Recently, Cohen et al. (2011) showed how to interpret shift-reduce actions as a generative model; combining their idea and our transition system might enable the model to exploit memory biases that exist in natural sentences.

Finally, dependency grammars are suitable for treating non-projective structures. Extensions for transition systems have been proposed to handle non-projective structures with additional actions (Attardi, 2006; Nivre, 2009). Although our system cannot handle non-projective structures, a similar extension might be possible, which would enable a left-corner analysis for non-projective structures.

Acknowledgements

We thank Pontus Stenetorp and anonymous reviewers for their valuable feedbacks on a preliminary version of this paper.

References

Steven Abney and Mark Johnson. 1991. Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250.

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

Miguel Ballesteros and Joakim Nivre. 2013. Going to the roots of dependency parsing. *Computational Linguistics*, 39(1):5–13.

Niels Beuck and Wolfgang Menzel. 2013. Structural prediction in incremental dependency parsing. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 7816 of *Lecture Notes in Computer Science*, pages 245–257. Springer Berlin Heidelberg.

Marisa Ferrara Boston and John T. Hale. 2007. Garden-pathing in a statistical dependency parser. In *Proceedings of the Midwest Computational Linguistics Colloquium*, West Lafayette, IN. Midwest Computational Linguistics Colloquium.

Marisa Ferrara Boston, John T. Hale, Umesh Patil, Reinhold Kliegl, and Shravan Vasishth. 2008. Parsing costs as predictors of reading difficulty: An evaluation using the Potsdam Sentence Corpus. *Journal of Eye Movement Research*, 2(1):1–12.

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June. Association for Computational Linguistics.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1245, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

Nelson Cowan. 2001. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114.

Vera Demberg, Frank Keller, and Alexander Koller. 2013. Incremental, predictive parsing with psycholinguistically motivated tree-adjoining grammar. *Computational Linguistics*, 39(4):1025–1066.

Daniel Gildea and David Temperley. 2007. Optimizing grammars for minimum dependency length. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 184–191, Prague, Czech Republic, June. Association for Computational Linguistics.

Daniel Gildea and David Temperley. 2010. Do grammars minimize dependency length? *Cognitive Science*, 34(2):286–310.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *TACL*, 1:403–414.

Katsuhiko Hayashi, Taro Watanabe, Masayuki Asahara, and Yuji Matsumoto. 2012. Head-driven transition-based parsing with top-down prediction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 657–665, Jeju Island, Korea, July. Association for Computational Linguistics.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 163–172, Sofia, Bulgaria, August. Association for Computational Linguistics.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden, July. Association for Computational Linguistics.

T.Florian Jaeger and Harry Tily. 2011. On language utility: Processing complexity and communicative efficiency. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(3):323–335.

P. N. Johnson-Laird. 1983. *Mental models: towards a cognitive science of language, inference, and consciousness*. Harvard University Press, Cambridge, MA, USA.

Mark Johnson. 1998. Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In Christian Boitet and Pete Whitelock, editors, *COLING-ACL*, pages 619–623. Morgan Kaufmann Publishers / ACL.

Frank Keller. 2010. Cognitively plausible models of human language processing. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 60–67, Uppsala, Sweden, July. Association for Computational Linguistics.

Kotaro Kitagawa and Kumiko Tanaka-Ishii. 2010. Tree-based deterministic dependency parsing — an application to nivre's method —. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 189–193, Uppsala, Sweden, July. Association for Computational Linguistics.

Dan Klein and Christopher Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 478–485, Barcelona, Spain, July.

Arne Kohn and Wolfgang Menzel. 2014. Incremental predictive parsing with turboparser. In *Proceedings of the ACL 2014 Conference Short Papers*, Baltimore, USA, June. Association for Computational Linguistics.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682, Portland, Oregon, USA, June. Association for Computational Linguistics.

David Mareček and Zdeněk Žabokrtský. 2012. Exploiting reducibility in unsupervised dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 297–307, Jeju Island, Korea, July. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain, July. Association for Computational Linguistics.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

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

John K. Pate and Sharon Goldwater. 2013. Unsupervised dependency parsing with acoustic cues. *TACL*, 1:63–74.

Martin Popel, David Mareček, Jan Štěpánek, Daniel Zeman, and Zdeněk Žabokrtský. 2013. Coordination structures in dependency treebanks. In *ACL*, pages 517–527.

Philip Resnik. 1992. Left-corner parsing and psychological plausibility. In *COLING*, pages 191–197.

Brian Roark, Asaf Bachrach, Carlos Cardenas, and Christophe Pallier. 2009. Deriving lexical and syntactic expectation-based measures for psycholinguistic modeling via incremental top-down parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 324–333, Singapore, August. Association for Computational Linguistics.

Brian Edward Roark. 2001. *Robust Probabilistic Predictive Syntactic Processing: Motivations, Models, and Applications*. Ph.D. thesis, Providence, RI, USA. AAI3006783.

Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013. A transition-based dependency parser using a dynamic parsing strategy. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 135–144, Sofia, Bulgaria, August. Association for Computational Linguistics.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-coverage parsing using human-like memory constraints. *Computational Linguistics*, 36(1):1–30.

Marten van Schijndel and William Schuler. 2013. An analysis of frequency- and memory-based processing costs. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 95–105, Atlanta, Georgia, June. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.