Generalized Transition-based Dependency Parsing via Control Parameters

Bernd Bohnet, Ryan McDonald, Emily Pitler and Ji Ma
Google Inc.
{bohnetbd,ryanmcd,epitler,maji}@google.com

Abstract

In this paper, we present a generalized transition-based parsing framework where parsers are instantiated in terms of a set of control parameters that constrain transitions between parser states. This generalization provides a unified framework to describe and compare various transition-based parsing approaches from both a theoretical and empirical perspective. This includes well-known transition systems, but also previously unstudied systems.

1 Introduction

Transition-based dependency parsing is perhaps the most successful parsing framework in use today (Nivre, 2008). This is due to the fact that it can process sentences in linear time (Nivre, 2003); is highly accurate (Zhang and Nivre, 2011; Bohnet and Nivre, 2012; Weiss et al., 2015); and has elegant mechanisms for parsing non-projective sentences (Nivre, 2009). As a result, there have been numerous studies into different transition systems, each with varying properties and complexities (Nivre, 2003; Attardi, 2006; Nivre, 2008; Nivre, 2009; Gómez-Rodríguez and Nivre, 2010; Choi and Palmer, 2011; Pitler and McDonald, 2015).

While connections between these transition systems have been noted, there has been little work on developing frameworks that generalize the phenomena parsed by these diverse systems. Such a framework would be beneficial for many reasons: It would provide a language from which we can theoretically compare known transition systems; it can give rise to new systems that could have favorable empirical properties; and an implementation of the generalization allows for comprehensive empirical studies.

In this work we provide such a generalized transition-based parsing framework. Our framework can be cast as transition-based parsing as it contains both parser states as well as transitions between these states that construct dependency trees. As in traditional transition-based parsing, the state maintains two data structures: a set of *unprocessed tokens* (normally called the *buffer*); and a set of *operative tokens* (often called the *stack*). Key to our generalization is the notion of *active* tokens, which is the set of tokens in which new arcs can be created and/or removed from consideration. A parser instantiation is defined by a set of control parameters, which dictate: the types of transitions that are permitted and their properties; the capacity of the active token set; and the maximum arc distance.

We show that a number of different transition systems can be described via this framework. Critically the two most common systems are covered – arc-eager and arc-standard (Nivre, 2008). But also Attardi's non-projective (Attardi, 2006), Kuhlmann's hybrid system (Kuhlmann et al., 2011), the directed acyclic graph (DAG) parser of Sagae and Tsujii (2008), and likely others. More interestingly, the easy-first framework of Goldberg and Elhadad (2010) can be described as an arc-standard system with an unbounded active token capacity.

We present a number of experiments with an implementation of our generalized framework. One major advantage of our generalization (and its implementation) is that it allows for easy exploration of novel systems not previously studied. In Section 5 we discuss some possibilities and provide experiments for these in Section 6.

2 Related Work

Transition-based dependency parsing can be characterized as any parsing system that maintains a state as well as a finite set of operations that move the system from one state to another (Nivre, 2008). In terms of modern statistical models that dominate the discourse today, the starting point is likely the work of Kudo and Matsumoto (2000) and Yamada and Matsumoto (2003), who adopted the idea of cascaded chunking from Abney (1991) in a greedy dependency parsing framework.

From this early work, transition-based parsing quickly grew in scope with the formalization of the arc-eager versus arc-standard paradigms (Nivre, 2003; Nivre, 2008), the latter largely being based on well-known shift-reduce principles in the phrase-structure literature (Ratnaparkhi, 1999). The speed and empirical accuracy of these systems – as evident in the widely used MaltParser software (Nivre et al., 2006a) – led to the study of a number of different transition systems.

Many of these new transition systems attempted to handle phenomena not covered by arc-eager or arc-standard transition systems, which inherently could only produce projective dependency trees. The work of Attardi (2006), Nivre (2009), Gómez-Rodríguez and Nivre (2010), Choi and Palmer (2011), and Pitler and McDonald (2015) derived transition systems that could parse non-projective trees. Each of these systems traded-off complexity for empirical coverage. Additionally, Sagae and Tsujii (2008) developed transition systems that could parse DAGs by augmentating the arc-standard and the arc-eager system. Bohnet and Nivre (2012) derived a system that could produce both labeled dependency trees as well as part-of-speech tags in a joint transition system. Taking this idea further Hatori et al. (2012) defined a transition system that performed joint segmentation, tagging and parsing.

In terms of empirical accuracy, from the early success of Nivre and colleagues (Nivre et al., 2006b; Hall et al., 2007; Nivre, 2008), there has been an succession of improvements in training and decoding, including structured training with beam search (Zhang and Clark, 2008; Zhang and Nivre, 2011), incorporating graph-based rescoring features (Bohnet and Kuhn, 2012), the aforementioned work on joint parsing and tagging (Bohnet and Nivre, 2012), and more recently the adoption of neural networks and feature embeddings (Chen and Manning, 2014; Weiss et al., 2015; Dyer et al., 2015; Alberti et al., 2015).

In terms of abstract generalizations of transition systems, the most relevant work is that of Gómez-Rodríguez and Nivre (2013) – which we abbreviate GR&N13. In that work, a generalized framework is defined by first defining a set of base transitions, and then showing that many transition-based systems can be constructed via composition of these base transitions. Like our framework, this covers common systems such as arc-eager and arc-standard, as well as easy-first parsing. In particular, arc construction in easy-first parsing can be seen as an action composed of a number of shifts, an arc action, and a number of un-shift actions. The primary conceptual difference between that work and the present study is the distinction between complex actions versus control parameters. In terms of theoretical coverage, the frameworks are not equivalent. For instance, our generalization covers the system of Attardi (2006), whereas GR&N13 cover transition systems where multiple arcs can be created in tandem. In Section 7 we compare the two generalizations.

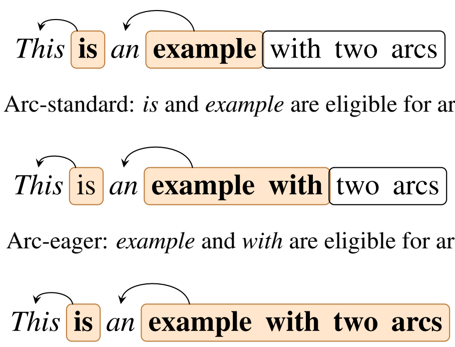
3 Generalized Transition-based Parsing

A transition system must define a parser state as well as a set of transitions that move the system from one state to the next. Correct sequences of transitions create valid parse trees. A parser state is typically a tuple of data structures and variables that represent the dependency tree constructed thus far and, implicitly, possible valid transitions to the next state.

In order to generalize across the parser states of transition-based parsing systems, we preserve their common parts and make the specific parts configurable. In order to generalize across the transitions, we divide the transitions into their basic operations. Each specific transition is then defined by composition of these basic operations. By defining the properties of the data structures and the composition of the basic operations, different transition systems can be defined and configured within one single unified system. As a consequence, we obtain a generalized parser that is capable of executing a wide range of different transition systems by setting a number of control parameters without changing the specific implementation of the generalized parser.

3.1 Basic Notation

In the following, we use a directed unlabeled dependency tree $T = \langle V, A \rangle$ for a sentence $x =$


This is an example with two arcs
This is an example with two arcs
This is an example with two arcs

(a) Arc-standard: *is* and *example* are eligible for arcs.

(b) Arc-eager: *example* and *with* are eligible for arcs.

(c) Easy-first: All unreduced tokens are active (bolded).

Figure 1: A partially processed dependency tree after having just added the arc *(example, an)* in the arc-standard, arc-eager, and easy-first systems. Tokens in the operative token set O are shaded orange, while tokens in the unordered buffer U are in an unshaded box. The bolded tokens are in ACTIVE(O) and eligible for arcs (Section 3.4).

Transitions

"Adds an arc from j to i, both $\in$ ACTIVE(O)."	
$\leftarrow_{i,j}$	$(O, U, A) \Rightarrow (O, U, A \cup (j \rightarrow i))$
"Adds an arc from i to j, both $\in$ ACTIVE(O)."	
$\rightarrow_{i,j}$	$(O, U, A) \Rightarrow (O, U, A \cup (i \rightarrow j))$
"Removes token $i \in$ ACTIVE(O) from O."	
$-_i$	$(O...i..., U, A) \Rightarrow (O, U, A)$
"Moves the top token from U to the top of O."	
$+$	$(O, i\|U, A) \Rightarrow (O\|i, U, A)$

Figure 2: Base generalized transitions over parser states.

$w_1, ..., w_n$, where $V_x = \{1, ..., n\}$ and $V_x^r = V_x \cup \{r\}$, $A_x \subset V_x^r \times V_x$. r is a placeholder for the root node and set either to 0 (root to the left of the sentence) or $n + 1$ (root to the right). The definition is mostly equivalent to that of Kübler et al. (2009) but deviates in the potential handling of the root on the right (Ballesteros and Nivre, 2013).

The set of nodes V_x index the words of a sentence x and V_x^r includes in addition the artificial root node r. Let A be the arc set, i.e., $(i, j) \in A$ iff there is a dependency from i to j. We use as alternative notation $(i \rightarrow j)$. This is the arc set the algorithm will create.

For ease of exposition, we will only address unlabeled parsing. However, for our experiments we do implement a labeled parsing variant using the standard convention of composing arc transitions with corresponding arc labels.

3.2 Generalized Parser State

Let U be an unordered set of buffered unprocessed tokens. This set is identical to the buffer from transition-based parsing. Following standard notation, we will use $i|U$ to indicate that i is the leftmost element of the set.

Let O be an ordered set of operative tokens. Specifically, O is the set of tokens that 1) have been moved out of U, and 2) are not themselves reduced. The set O is similar in nature to the traditional stack of transition-based parsing, but is not restricted to stack operations. As in transition-based parsing we will use the notation $O|i$ to indicate that i is the rightmost element of the set; $O[n]$ is the n-th rightmost element of the set. Figure 1 shows the set of tokens in O within shaded boxes and U within unshaded boxes.

3.3 Generalized Transitions

The above discussion describes what a generalized transition-based parser state looks like; it does not describe any transitions between these states, which is the core of transition-based parsing. In this section we present a set of basic transitions, which themselves can be composed to make more complex transitions (similar to Gómez-Rodríguez and Nivre (2013)).

Let $\mathcal{T} = \{\leftarrow_{i,j}, \rightarrow_{i,j}, -_i, +\}$ be the set of basic transitions (Figure 2), which have analogues to standard transition-based parsing. These transitions come with the standard preconditions, i.e., a root cannot be a modifier; each token can modify at most one word[1]; a token can only be reduced if it has a head; and a shift can only happen if the unoperative buffer is non-empty. We will often refer to these as LEFT-ARC ($\leftarrow_{i,j}$), RIGHT-ARC ($\rightarrow_{i,j}$), REDUCE ($-_i$), and SHIFT ($+$). We additionally refer to LEFT-ARC and RIGHT-ARC together as *arc-creation* actions.

3.4 Control Parameters

Instantiations of a transition system are defined via *control parameters*. We defined two sets of such parameters. The first, we call *global parameters*, dictates system wide behaviour. The second, we call *transition parameters*, dictates a specific behaviour for each transition.

[1]This precondition is not needed for DAG transition systems.

Global Control Parameters

We have two parameters for the broader behaviour of the system.

1. Active token capacity K. The active set of tokens ACTIVE(O) that can be operated on by the transitions is the set $\{O[min(|O|, K)], ..., O[1]\}$. K additionally determines the size of O at the start of parsing. E.g., if $K = 2$, then we populate O with the first two tokens. This is equivalent to making SHIFT deterministic while $|O| < K$.

2. Max arc distance D. I.e., arcs can only be created between two active tokens $O[i]$ and $O[j]$ if $|i - j| \leq D$.

Transition Control Paramters

Let $\mathcal{M}(\mathcal{T})$ be a *multiset* of transitions, such that if $t \in \mathcal{M}(\mathcal{T})$, then $t \in \mathcal{T}$. Note that $\mathcal{M}(\mathcal{T})$ is a multiset, and thus can have multiple transitions of the same type. For each $t \in \mathcal{M}(\mathcal{T})$, our generalization requires the following control parameters to be set (default in bold):

1. Bottom-up: $B \in \{[\mathbf{t}]\mathbf{rue}, [f]alse\}$. Whether creating an arc also reduces it. Specifically we will have two parameters, B_L and B_R, which specify whether LEFT/RIGHT-ARC actions are bottom up. We use the notation and say $B = true$ to mean $B_L = B_R = true$. For example $B_L = true$ indicates that $\leftarrow_{i,j}$ is immediately followed by a reduce $-_i$.

2. Arc-Shift: $S \in \{[t]rue, [\mathbf{f}]\mathbf{alse}\}$. Whether creating an arc also results in SHIFT. Specifically we will have two parameters, S_L and S_R, which specify whether LEFT/RIGHT-ARC actions are joined with a SHIFT. We use the notation and say $S = true$ to mean $S_L = S_R = true$. For example $S_L = true$ indicates that $\leftarrow_{i,j}$ is immediately followed by $+$.

3. Periphery: $P \in \{[l]eft, [r]ight, \mathbf{na}\}$. If a transition must operate on the left or right periphery of the active token set ACTIVE(O). For arc-creation transitions, this means that at least one of the head or modifier is on the specified periphery. If the value is na, that means that the action is not constrained to be on the periphery. Note, that when K $\leq$ 2, all arc-creation actions by default are on the periphery.

Each of these control parameters has a default value, which will be assumed if unspecified. Note that the relevance of these parameters is transition dependent. E.g., a SHIFT requires no such control parameters and a REDUCE needs neither B nor S.

These control parameters allow *limited* compositionality of the basic transitions in Figure 2. Unlike Gómez-Rodríguez and Nivre (2013), each transition includes at most one SHIFT, at most one REDUCE, and at most one LEFT-ARC or RIGHT-ARC. I.e., the most compositional transition is a LEFT/RIGHT-ARC with a REDUCE and/or a SHIFT. Even with this restriction, all of the transition systems covered by Gómez-Rodríguez and Nivre (2013) can still be expressed in our generalization.

3.5 Generalized Transition System

To summarize, a generalized transition is defined as follows:

1. A parser state: $\Gamma = \langle O, U, A \rangle$.

2. A set of basic transitions: $\mathcal{T} = \{\leftarrow_{i,j}, \rightarrow_{i,j}, -_i, +\}$.

And each transition system instantiation must further define:

1. Values for global control parameters: K and D.

2. A multiset of valid transitions $\mathcal{M}(\mathcal{T})$, where $\forall t \in \mathcal{M}(\mathcal{T})$, then $t \in \mathcal{T}$.

3. For each $t \in \mathcal{M}(\mathcal{T})$ values for B, S, and P.

4 Transition System Instantiations

The instantiation of the transition system consists of setting the capacity K and distance D as well as the transition control parameters. In order to make the comparison clearer, we will define typical transition systems using the notation of Kuhlmann et al. (2011). Here, a parser state is a triple (σ, β, A), where $\sigma|i$ is a stack with top element i, $j|\beta$ is a buffer whose next element is j, and A the set of created arcs. To make notation cleaner, we will drop indexes whenever the context makes it clear. E.g., if the active token capacity is 2 ($K = 2$), then necessarily for $\leftarrow_{i,j}$, $i = 2$ and $j = 1$ and we can write $\leftarrow$. When $K > 2$ or $D > 1$, a base arc-creation action can be instantiated into multiple transitions that only differ by the indexes. E.g., when $K = 3$ and $D = 2$, $\leftarrow$ with $P = na$ can have three instantiations: $\leftarrow_{3,2}$, $\leftarrow_{2,1}$ and

$\leftarrow_{3,1}$. To keep exposition compact, in such circumstances we use $\star$ to denote the set of index-pairs allowed by the given K, D and P values.

4.1 Arc-standard

Arc-standard parsing is a form of shift-reduce parsing where arc-creations actions happen between the top two elements on the stack:

LEFT-ARC:	$(\sigma\|i\|j, \beta, A) \Rightarrow$
	$(\sigma\|j, \beta, A \cup (j \rightarrow i))$
RIGHT-ARC:	$(\sigma\|i\|j, \beta, A) \Rightarrow$
	$(\sigma\|i, \beta, A \cup (i \rightarrow j))$
SHIFT:	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, A)$

This is easily handled in our generalization by having only two active tokens. $\leftarrow$ and $\rightarrow$ actions with default parameter values are used to simulate LEFT-ARC and RIGHT-ARC respectively, and $+$ is used to shift tokens from U to O.

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow$	$\{$*default*$\}$
RIGHT-ARC	$\rightarrow$	$\{$*default*$\}$
SHIFT	$+$	
capacity K		2
arc distance D		1

Note that when $K = 2$, arc-creations are by definition always on the periphery.

4.2 Arc-eager

Arc-eager transition systems have been described in various ways. Kuhlmann et al. (2011) defines it as operations between tokens at the top of the stack and front of the buffer.

LEFT-ARC:	$(\sigma\|i, j\|\beta, A) \Rightarrow$
	$(\sigma, j\|\beta, A \cup (j \rightarrow i))$
RIGHT-ARC:	$(\sigma\|i, j\|\beta, A) \Rightarrow$
	$(\sigma\|i\|j, \beta, A \cup (i \rightarrow j))$
REDUCE:	$(\sigma\|i, \beta, A) \Rightarrow (\sigma, \beta, A)$
SHIFT:	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, A)$

In our generalization, we can simulate this by having two active tokens on the operative set, representing the top of the stack and front of the buffer. SHIFT and LEFT-ARC are handled in the same way as in the arc-standard system. The RIGHT-ARC action is simulated by $\rightarrow$ with modified parameters to account for the fact that the action keeps the modifier in O and also shifts a token from U to O. The REDUCE action is handled by $-$ with the periphery set to left so as to remove the second rightmost token from O.

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow$	$\{$*default*$\}$
RIGHT-ARC	$\rightarrow$	$\{B = f, S = t\}$
REDUCE	$-$	$\{P = l\}$
SHIFT	$+$	
K		2
D		1

Note that the artificial root must be on the right of the sentence to permit the reduce to operate at the left periphery of the active token set.

4.3 Easy-first

For easy-first parsing (Goldberg and Elhadad, 2010), the number of active tokens is infinite or, more precisely, equals to the number of tokens in the input sentence, and arc-creation actions can happen between any two adjacent tokens.

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow_\star$	$\{$*default*$\}$
RIGHT-ARC	$\rightarrow_\star$	$\{$*default*$\}$
K		∞
D		1

Note here $\star$ denotes the set of indexes $\{(i,j) | i \leq |O|, j \geq 1, i = j+1\}$. Thus, at each time step there are $2 * |O|$ actions that need to be considered. Additionally, reduce is always composed with arc-creation and since $K = \infty$, then there is never a SHIFT operation as O is immediately populated with all tokens on the start of parsing.

4.4 Kuhlmann et al. (2011)

Kuhlmann et al. present a 'hybrid' transition system where the RIGHT-ARC action is arc-standard in nature, but LEFT-ARC actions is arc-eager in nature, which is equivalent to the system of Yamada and Matsumoto (2003). We can get the same effect as their system by allowing three active tokens, representing the top two tokens of the stack and the front of the buffer. Transitions can be handled similarly as the arc-standard system where only the periphery parameter need to be changed accordingly. This change also requires the root to be on the right of the sentence.

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow$	$\{P = r\}$
RIGHT-ARC	$\rightarrow$	$\{P = l\}$
SHIFT	$+$	
K		3
D		1

4.5 Sagae and Tsujii (2008)

Sagae and Tsuji present a model for projective DAG parsing by modifying the LEFT-ARC and RIGHT-ARC transitions of the arc-eager system. In their system, the LEFT-ARC transition does not remove the dependent, and RIGHT-ARC transition does not shift any token from the input to the stack.

LEFT-ARC:	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma\|i, j\|\beta, A \cup (j \to i))$
RIGHT-ARC:	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma\|i, j\|\beta, A \cup (i \to j))$
REDUCE:	$(\sigma\|i, \beta, A) \Rightarrow (\sigma, \beta, A)$
SHIFT:	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, A)$

This can be easily simulated by modifying arc-eager system mentioned in Sec. 4.2 such that both the LEFT-ARC and RIGHT-ARC transition keep the stack/buffer untouched.

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow$	$\{B = f\}$
RIGHT-ARC	$\rightarrow$	$\{B = f\}$
REDUCE	$-$	$\{P = l\}$
SHIFT	$+$	
K		2
D		1

4.6 Attardi (2006)

Now we show how our framework can extend to non-projective systems. This is primarily controlled by the arc-distance parameter D.

The base of the Attardi non-projective transition system is the arc-standard system. Attardi modifies RIGHT/LEFT-ARC actions that operate over larger contexts in the stack. For simplicity of exposition below we model the variant of the Attardi system described in Cohen et al. (2011).

RIGHT-ARCN:	$(\sigma\|i_{1+N}\|\ldots\|i_2\|i_1, \beta, A) \Rightarrow (\sigma\|i_N\|\ldots\|i_2\|i_1, \beta, A \cup (i_1 \to i_{1+N}))$
LEFT-ARCN:	$(\sigma\|i_{1+N}\|\ldots\|i_2\|i_1, \beta, A) \Rightarrow (\sigma\|i_{1+N}\|\ldots\|i_2, \beta, A \cup (i_{1+N} \to i_1))$
SHIFT:	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, A)$

Conceptually the Attardi system can be generalized to any value of N[2], and Attardi specifically allows N=1,2,3. Actions that create arcs between non-adjacent tokens permit limited non-projectivity.

Thus, for Attardi, we set the number of active tokens to N+1, to simulate the top N+1 tokens of the stack, and set the max distance D to N to indicate that tokens up to N positions below the top of the stack can add arcs with the top of the stack.

[2]Attardi (2006) also introduces Extract/Insert actions to a temporary buffer that he argues generalizes to all values of N. We don't account for that specific generalization here.

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow_\star$	$\{P = r\}$
RIGHT-ARC	$\rightarrow_\star$	$\{P = r\}$
K		$N + 1$
D		N (3 for Attardi (2006))

The critical parameter for each arc action is that $P = r$. This means that the right peripheral active token always must participate in the action, as does the right-most token of the stack for the original Attardi. Here $\star$ denotes $\{(i, j) | j = 1, i - j \leq D\}$.

5 Novel Transition Systems

Any valid setting of the control parameters could theoretically define a new transition system, however not all such combinations will be empirically reasonable. We outline two potential novel transition systems suggested by our framework, which we will experiment with in Section 6. This is a key advantage of our framework (and implementation) – it provides an easy experimental solution to explore novel transition systems.

5.1 Bounded Capacity Easy-first

Easy-first and arc-standard are similar since they are both bottom-up and both create arcs between adjacent nodes. The main difference lies in the capacity K, which is 2 for arc-standard and ∞ for easy-first. In addition, the shift action is needed by the arc-standard system. Each system has some advantages: arc-standard is faster, and somewhat easier to train, while easy-first can be more accurate (under identical learning settings). Seen this way, it is natural to ask: what happens if the active token range K is set to k, with $2 < k < \infty$? We explore various values in the region between arc-standard and easy-first in Section 6.

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow_\star$	{*default*}
RIGHT-ARC	$\rightarrow_\star$	{*default*}
SHIFT	$+$	
K		k
D		1

5.2 Non-projective Easy-first

A simple observation is that by allowing $D > 1$ makes any transition system naturally non-projective. One example would be a non-projective variant of easy-first parsing:

$\mathcal{M}(\mathcal{T})$	base	parameter values
LEFT-ARC	$\leftarrow_\star$	$\{default\}$
RIGHT-ARC	$\rightarrow_\star$	$\{default\}$
K		∞
D		any value of N

Here N denotes the maximum arc-creation distance.

We also note that one could potentially vary both K and D simultaneously, giving a non-projective limited capacity easy-first system.

6 Implementation and Experiments

Our implementation uses a linear model $\sum_{y_i \in y} \mathbf{w} \cdot \mathbf{f}(x, y_1, \ldots, y_i)$ to assign a score to a sequence $y = y_1, y_2, \ldots y_m$ of parser transitions, given sentence x. Model parameters are trained using the structured perceptron with "early update" (Collins and Roark, 2004) and features follow that of Zhang and Nivre (2011).

For the arc-standard and arc-eager transition systems, we use the static oracle to derive a single gold sequence for a given sentence and its gold tree. For systems where there is no such static oracle, for example the easy-first system, we use the method proposed by Ma et al. (2013) to select a gold sequence such that, for each update, the condition $\mathbf{w} \cdot \mathbf{f}(x, \hat{y}_k) < \mathbf{w} \cdot \mathbf{f}(x, \overline{y}_k)$ always holds, which is required for perceptron convergence. Here $\hat{y}_k$ denotes the length k prefix of a correct sequence and $\overline{y}_k$ denotes the highest scoring sequence in the beam.

We carry out the experiments on the Wall Street Journal using the standard splits for the training set (section 2-21), development set (section 22) and test set (section 23). We converted the constituency trees to Stanford dependencies version 3.3.0 (de Marneffe et al., 2006). We used a CRF-based Part-of-Speech tagger to generate 5-fold jack-knifed Part-of-Speech tag annotation of the training set and used predicted tags on the development and test set. The tagger reaches accuracy scores similar to the Stanford tagger (Toutanova et al., 2003) with 97.44% on the test set. The unlabeled and labeled accuracy scores exclude punctuation marks.

Obviously, there are many interesting instantiations for the generalized transition system. In particular, it would be interesting to investigate parsing performance of systems with different active

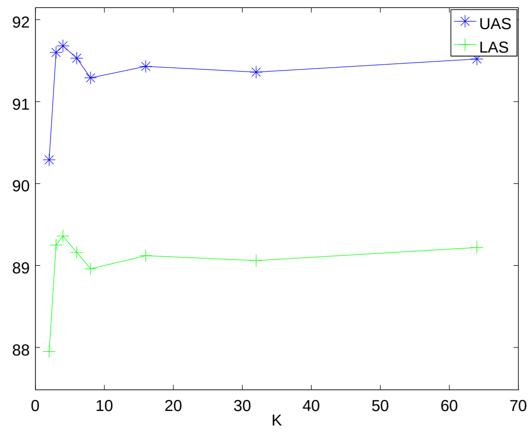


Figure 3: Labeled/unlabeled attachment scores with respect to the active capacity K.

token size and arc-distance. Before we investigate these system in the next subsections, we present the performance on standard systems.

6.1 Common Systems

The results for arc-standard, arc-eager and easy-first (bottom of Table 1) show how standard systems perform within our framework. Easy-first's labeled attachment score (LAS) is 0.46 higher than the LAS of arc-eager when using the same feature set. These results are competitive with the state-of-the-art linear parsers, but below recent work on neural network parsers. A future line of work is to adopt such training into our generalization.

System	UAS	LAS	b
Dyer et al. (2015)	93.10	90.90	–
Weiss et al. (2015)	93.99	92.05	–
Alberti et al. (2015)	94.23	92.23	8
Zhang and Nivre (2011)⋆	92.92	90.88	32
Zhang and McDonald (2014)	93.22	91.02	–
Arc-standard (gen.)	92.81	90.68	32
Arc-eager (gen.)	92.88	90.73	32
Easy-first (gen.)	93.31	91.19	32

Table 1: State-of-the-art comparison. ⋆ denotes our own re-implementation. The systems in the first block on the top use neural networks.

6.2 Bounded Capacity Easy-first

Table 1 shows that easy-first is more accurate than arc-standard. However, it is also more computationally expensive. By varying the number of active tokens, we can investigate whether there is a sweet spot in the accuracy vs. speed trade-off.

Figure 3 shows labeled/unlabeled accuracy scores on the development set for active token sizes ranging from 2 to 32, all with beam size 1. Different from the original easy-first system where all tokens are initialized as active tokens, in the bounded capacity system, a token can be active only after it has been shifted from U to O. We observe that the accuracy score increases remarkably by over a point when active token capacity gets increased from 2 to 3, and peaks at a active token capacity of 4. Generally, more active tokens allows the parser to delay "difficult" decisions to later steps and choose the "easy" ones at early steps. Such behavior has the effect of limiting the extent of error propagation. The result also suggests that a modification as simple as adding one more active token to the arc-standard system can yield significant improvement.

With a larger active token capacity, we see a slight drop of accuracy. This is likely related to the parser having to predict when to perform a shift transition. In comparison, the vanilla easy-first parser does not need to model this.

6.3 Non-projective Easy-first

For the experiments with non-projective easy-first, we use the Dutch and Danish CoNLL 2006 corpora. To assess the performance, we applied the evaluation rules of the 2006 shared task. In order to make the non-projective systems perform well, we added to all feature templates the arc-distance D. In these experiments, we included in the training an artificial root node on the right since Dutch as well a few sentences of the Danish corpus have more than one root node.

In the experiments, we use the easy-first setting with infinite set of active tokens K and increase stepwise the arc-distance D. For training, we filter out the sentences which contain non-projective arcs not parseable with the selected setting. Table 2 provides an overview of the performance with increasing arc-distance. At the bottom of the table, we added accuracy scores for the bounded capacity non-projective easy-first parser since we think these settings provide attractive trade-offs between accuracy and complexity.

The original easy-first performs $O(n)$ feature extractions and has a runtime of $O(n \log(n))$ assuming a heap is used to extract the argmax at each step and feature extraction is done over local contexts only (Goldberg and Elhadad, 2010). For the non-projective variant of easy-first with $D = d$, $O(dn)$ feature extractions are required. Thus, for the unrestricted variant where $K = D = \infty$, $O(n^2)$ feature extractions are required. Table 2 explored more practical settings: when $K = \infty$, $D \leq 7$, the number of feature extractions is back to $O(n)$ with a runtime of $O(n \log(n))$, matching the original easy-first complexity. When both K and D are small constants as in the lower portion of Table 2, the runtime is $O(n)$.

		Danish				Dutch			
D	K	UAS	LAS	CUR	NPS	UAS	LAS	CUR	NPS
1	∞	90.22	85.51	41.30	84.37	79.11	75.41	32.45	63.55
2	∞	90.28	85.85	59.78	96.91	84.73	81.01	70.59	92.44
3	∞	90.68	86.07	65.22	98.82	85.03	81.65	**77.99**	99.01
4	∞	90.58	85.53	69.57	99.69	85.99	82.73	76.85	99.89
5	∞	90.84	86.11	65.22	99.88	85.21	81.93	76.09	99.96
6	∞	90.78	**86.31**	68.48	99.94	84.57	81.13	75.90	**100.0**
7	∞	90.64	85.91	63.04	**100.0**	85.07	82.01	77.04	**100.0**
4	5	90.74	85.87	66.30	99.69	**86.51**	**82.91**	76.66	99.89
5	6	**91.00**	86.21	**72.83**	99.88	86.03	82.73	76.09	99.96

Table 2: Experiments with non-projective easy-first and bounded capacity easy-first with D the arc-distance, K the active token capacity (∞ = all tokens of a sentence), UAS and LAS are the unlabeled and labeled accuracy scores, CUR is the recall of crossing edges and NPS shows the percentage of sentences covered in the training set where 100% means all non-projective (and projective) sentences in the training can be parsed and are included in training.

7 Comparison with GR&N13

The work most similar to ours is that of Gómez-Rodríguez and Nivre (2013) (GR&N13). They define a *divisible transition system* with the principle to divide the transitions into elementary transitions and then to compose from these elementary transitions complex transitions. GR&N13 identified the following five elementary transitions:

SHIFT:	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, A)$
UNSHIFT:	$(\sigma\|i, \beta, A) \Rightarrow (\sigma, i\|\beta, A)$
REDUCE:	$(\sigma\|i, \beta, A) \Rightarrow (\sigma, \beta, A)$
LEFT-ARC:	$(\sigma\|i, j\|\beta, A) \Rightarrow$ $(\sigma\|i, j\|\beta, A \cup (j \rightarrow i))$
RIGHT-ARC:	$(\sigma\|i, j\|\beta, A) \Rightarrow$ $(\sigma\|i, j\|\beta, A \cup (i \rightarrow j))$

The notion of function composition is used to combine the elementary transitions to the complex ones. For instance, arc-standard would have three actions: SHIFT; RIGHT-ARC $\oplus$REDUCE; LEFT-ARC $\oplus$REDUCE.

The first difference we can note is the GR&N13 cannot instantiate transition systems that produce non-projective trees. This is surely a superficial difference, as GR&N13 could easily add transitions with larger arc-distance or even SWAP actions (Nivre, 2009).

However, a less superficial difference is that our generalization uses control parameters to construct instantiations of transition systems, instead of solely via transition composition like GR&N13. Through this, our generalization results in a minimal departure from 'standard' representations of these systems. While this may seem like a notational difference, this is particularly a benefit with respect to implementation, as previous techniques for classification and feature extraction can largely be reused.

For example, in GR&N13, the definition of the easy-first transition system (Goldberg and Elhadad, 2010) is complex, e.g., a RIGHT-ARC at position i requires a compositional transition of i SHIFT actions, a RIGHT-ARC, a SHIFT, a REDUCE, then i UNSHIFT actions. Note, that this means in any implementation of this generalization, the output space for a classifier will be very large. Furthermore, the feature space would ultimately need to take the entire sentence into consideration, considering that all compositional actions are centered on the same state.

In our transition system, on the other hand, easy-first operates almost as it does in its native form, where n LEFT-ARC and n RIGHT-ARC actions are ranked relative to each other. There are only two actions, each instantiated for every location in the state. Thus the output space and feature extraction are quite natural.

This leads to straight-forward implementations allowing for easy experimentation and discovery. Unlike GR&N13, we present empirical results for both known transition systems as well as some novel systems (Section 6).

8 Conclusion

We presented a generalized transition system that is capable of representing and executing a wide range of transition systems within one single implementation. These transition systems include systems such as arc-standard, arc-eager, easy-first.

Transitions can be freely composed of elementary operations. The transition system shows perfect alignment between the elementary operations on one hand and their preconditions and the oracle on the other hand. We adjust the transition system to work on a stack in a uniform way starting at a node on the stack and ending with the top node of the stack. The results produced by this system are more comparable as they can be executed with the same classifier and feature extraction system.

Finally, we would like to highlight two insights that the experiments provide. First, a few more active tokens than two can boost the accuracy level of an arc-standard transition system towards the level of an easy-first transition system. These parsing systems maintain very nicely the linear complexity of the arc-standard transition system while they provide a higher accuracy similar to those of easy-first. Second, non-projective trees can be parsed by allowing a larger arc-distance which is a simple way to allow for non-projective edges.

We think that the transition systems with more active tokens or the combination with edges that span over more words provide very attractive transition systems for possible future parsers.

Acknowledgments

We would like to thank the anonymous reviewers, Slav Petrov and Michael Collins for valuable comments on earlier versions of this manuscript.

References

Steven Abney. 1991. Parsing by chunks. In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-Based Parsing*, pages 257–278. Kluwer.

Chris Alberti, David Weiss, Greg Coppola, and Slav Petrov. 2015. Improved transition-based parsing and tagging with neural networks. In *Proceedings of EMNLP 2015*.

Giuseppe Attardi. 2006. Experiments with a multi-language non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.

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

Bernd Bohnet and Jonas Kuhn. 2012. The best of both worlds – a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chpater of the Association for Computational Linguistics (EACL)*, pages 77–87.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical*

Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 1455–1465.

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

Jinho D Choi and Martha Palmer. 2011. Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*.

Shay B Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1234–1245.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 112–119.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of ACL 2015*, pages 334–343. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 742–750.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1492–1501.

Carlos Gómez-Rodríguez and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Computational Linguistics*, 39(4):799–845.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 933–939.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2012. Incremental joint approach to word segmentation, pos tagging, and dependency parsing in chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1045–1053.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan and Claypool.

Taku Kudo and Yuji Matsumoto. 2000. Japanese dependency structure analysis based on support vector machines. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in NLP and Very Large Corpora*, pages 18–25.

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 (ACL)*, pages 673–682.

Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. 2013. Easy-first pos tagging and dependency parsing with beam search. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–114, Sofia, Bulgaria, August. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006a. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006b. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

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. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

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 (ACL-IJCNLP)*, pages 351–359.

Emily Pitler and Ryan McDonald. 2015. A linear-time transition system for crossing interval trees. In *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 662–671.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 753–760.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. 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*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of ACL 2015*, pages 323–333.

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

Yue Zhang and Stephen Clark. 2008. 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)*, pages 562–571.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 656–661, Baltimore, Maryland, June. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*.