A Dynamic Oracle for Arc-Eager Dependency Parsing

Yoav Goldberg[1] *Joakim Nivre*[1,2]
(1) Google Inc.
(2) Uppsala University
yogo@google.com, joakim.nivre@lingfil.uu.se

ABSTRACT
The standard training regime for transition-based dependency parsers makes use of an oracle, which predicts an optimal transition sequence for a sentence and its gold tree. We present an improved oracle for the arc-eager transition system, which provides a set of optimal transitions for every valid parser configuration, including configurations from which the gold tree is not reachable. In such cases, the oracle provides transitions that will lead to the best reachable tree from the given configuration. The oracle is efficient to implement and provably correct. We use the oracle to train a deterministic left-to-right dependency parser that is less sensitive to error propagation, using an online training procedure that also explores parser configurations resulting from non-optimal sequences of transitions. This new parser outperforms greedy parsers trained using conventional oracles on a range of data sets, with an average improvement of over 1.2 LAS points and up to almost 3 LAS points on some data sets.

KEYWORDS: dependency parsing, transition system, oracle.

1 Introduction

The basic idea in transition-based dependency parsing is to define a nondeterministic transition system for mapping sentences to dependency trees and to perform parsing as search for the optimal transition sequence for a given sentence (Nivre, 2008). A key component in training transition-based parsers is an *oracle*, which is used to derive optimal transition sequences from gold parse trees. These sequences can then be used as training data for a classifier that approximates the oracle at parsing time in deterministic classifier-based parsing (Yamada and Matsumoto, 2003; Nivre et al., 2004), or it can be used to determine when to perform updates in online training of a beam search parser (Zhang and Clark, 2008).

Currently, such oracles work by translating a given tree to a static sequence of parser transitions which, if run in sequence, will produce the gold tree. Most transition systems, including the arc-eager and arc-standard systems described in Nivre (2003, 2004), exhibit *spurious ambiguity* and map several sequences to the same gold tree. In such cases, the oracles implicitly define a canonical derivation order. We call such oracles *static*, because they produce a single static sequence of transitions that is supposed to be followed in its entirety. Static oracles are usually specified as rules over individual parser configurations – if the configuration has properties X and the gold tree is Y, then the correct transition is Z – giving the impression that they define a function from configurations to transitions. However, these rules are only correct for configurations that are part of the canonical transition sequence defined by the oracle. Thus, static parsing oracles are only correct as functions from gold-trees to transition sequences, and not as functions from configurations to transitions.

There are at least two limitations to training parsers using static oracles. First, because of spurious ambiguity, it is not clear that the canonical transition sequence proposed by the oracle is indeed the easiest to learn. It could well be the case that a different sequence which also leads to a gold tree is preferable. Second, it happens often in greedy deterministic parsing that the parser deviates from the gold sequence, reaching configurations from which the correct tree is not derivable. The static oracle does not provide any means of dealing with such deviations. The parser's classifier is then faced with configurations which were not observed in training, and this often leads to a sequence of errors. It would be preferable for the parser to explore non-gold configurations at training time, thus mitigating the effect of error-propagation.

In this paper, we introduce the concept of a *dynamic* parsing oracle. Rather than defining a single static canonical transition sequence for producing a given gold tree, the dynamic oracle answers queries of the form: Is transition Z valid in configuration X for producing the best possible tree Y? A key difference compared to a static oracle is that the dynamic oracle no longer forces a unique transition sequence in situations where multiple sequences derive the gold tree. In this case, the dynamic oracle permits all valid transition sequences by answering "yes" on more than one transition Z in a given configuration.[1] The second crucial difference to a static oracle is that the dynamic oracle defined in this work is well-defined and correct for *all* configurations, including configurations which are not a part of a gold derivation. For configurations which are not part of a gold derivation (and from which the gold tree is not reachable), the dynamic oracle permits all transitions that can lead to a tree with minimum loss compared to the gold tree. In this paper, we provide a provably correct dynamic oracle for the arc-eager transition system of Nivre (2003, 2008).

One important use for a dynamic oracle is in training a parser that (a) is not restricted to a

[1] This is similar to the parsing oracle used in the EasyFirst parser of Goldberg and Elhadad (2010).

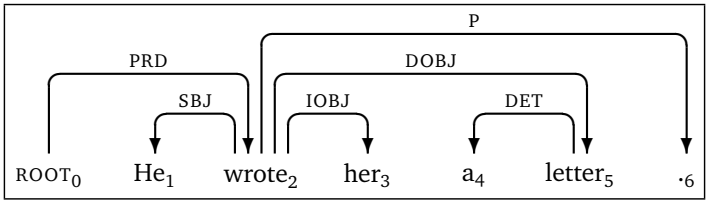

Figure 1: Projective dependency tree

particular canonical order of transitions and (b) can handle configurations that are not part of any gold sequence, thus mitigating the effect of error propagation. To this end, we provide an online training procedure based on the dynamic oracle that deals with spurious ambiguity by treating all sequences leading to a gold tree as correct, and with error-propagation by exploring transition sequences that are not optimal in the sense that they do not derive the gold tree, while training the parser to perform the optimal transitions on these non-optimal configurations. We show that both of these properties improve the accuracy of a deterministic left-to-right arc-eager parser by over 1.2 LAS points on average and up to almost 3 LAS points on some data sets, as compared to the conventional training procedure.

2 Background

2.1 Arc-Eager Transition-Based Dependency Parsing

Given a set L of dependency labels, we define a *dependency graph* for a sentence $x = w_1, \ldots, w_n$ as a labeled directed graph $G = (V_x, A)$, consisting of a set of nodes $V_x = \{0, 1, \ldots, n\}$, where each node i corresponds to the linear position of a word w_i in the sentence, plus an extra artificial root node 0, and a set of labeled arcs $A \subseteq V_x \times L \times V_x$, where each arc (i, l, j) represents a dependency with head w_i, dependent w_j, and label l. In order for a dependency graph to be well-formed, we usually require it to be a *dependency tree*, which is a directed spanning tree rooted at the node 0. In this paper, we further restrict our attention to dependency trees that are *projective*, that is, where the presence of an arc (i, l, j) entails that there is directed path from i to every node k such that $\min(i, j) < k < \max(i, j)$. Figure 1 shows a projective dependency tree for a simple English sentence.

In the arc-eager transition system of Nivre (2003), a *parser configuration* is a triple $c = (\Sigma, B, A)$ such that Σ (referred to as the *stack*) and B (the *buffer*) are disjoint sublists of the nodes V_x of some sentence x, and A is a set of dependency arcs over V_x (and some label set L); we take the initial configuration for a sentence $x = w_1, \ldots, w_n$ to be $c_s(x) = ([0], [1, \ldots, n], \{\})$; and we take a terminal configuration to be any configuration of the form $c = (\Sigma, [\,], A)$ (for any stack Σ and arc set A).[2] There are four types of *transitions*, defined formally in Figure 2, for going from one configuration to the next:

1. A LEFT-ARC$_l$ transition (for any dependency label l) adds the arc (b, l, s) to A, where s is the node on top of the stack and b is the first node in the buffer, and pops the stack. It has as a precondition that the token s is not the artificial root node 0 and does not already have a head.

[2] As is customary, we use the variables σ and β for arbitrary sublists of the stack and the buffer, respectively. For reasons of perspicuity, we will write Σ with its head (top) to the right and B with its head to the left. Thus, $c = (\sigma|s, b|\beta, A)$ is a configuration with the node s on top of the stack Σ and the node b as the first node in the buffer B.

Transition		Precondition
LEFT-ARC$_l$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma,j\|\beta,A\cup\{(j,l,i)\})$	$\neg[i=0] \wedge \neg\exists k\exists l'[(k,l',i)\in A]$
RIGHT-ARC$_l$	$(\sigma\|i,j\|\beta,A) \Rightarrow (\sigma\|i\|j,\beta,A\cup\{(i,l,j)\})$	
REDUCE	$(\sigma\|i,\beta,A) \Rightarrow (\sigma,\beta,A)$	$\exists k\exists l[(k,l,i)\in A]$
SHIFT	$(\sigma,i\|\beta,A) \Rightarrow (\sigma\|i,\beta,A)$	

Figure 2: Transitions for the arc-eager transition system

2. A RIGHT-ARC$_l$ transition (for any dependency label l) adds the arc (s,l,b) to A, where s is the node on top of the stack and b is the first node in the buffer, and pushes the node b onto the stack.

3. The REDUCE transition pops the stack and is subject to the preconditions that the top token has a head.

4. The SHIFT transition removes the first node in the buffer and pushes it onto the stack.

A *transition sequence* for a sentence x is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that c_0 is the initial configuration $c_s(x)$, c_m is a terminal configuration, and there is a legal transition t such that $c_i = t(c_{i-1})$ for every i, $1 \leq i \leq m$. The dependency graph derived by $C_{0,m}$ is $G_{c_m} = (V_x, A_{c_m})$, where A_{c_m} is the set of arcs in c_m. The arc-eager system is sound and complete for the class of projective dependency forests, meaning that every legal transition sequence derives a projective dependency forest (soundness) and that every projective dependency forest is derived by at least one transition sequence (completeness) (Nivre, 2008). A projective dependency forest is a dependency graph where every connected component is a projective tree, with the special case of a projective dependency tree in case there is only one connected component. Hence, the arc-eager transition system is also complete (but not sound) for the class of projective dependency trees.

2.2 Static Oracles for Transition-Based Parsing

In the transition-based framework, parsing is implemented as search for an optimal transition sequence, that is, a transition sequence that derives the correct parse tree for a given sentence. The simplest version of this, and also the most efficient, is to train a classifier to predict the single best transition in each configuration and use a greedy deterministic procedure to derive a single dependency graph, which results in linear-time parsing provided that each transition can be predicted and executed in constant time (Nivre, 2003, 2008). The classifier is trained on a set of configuration-transition pairs, which can be derived from a dependency treebank by finding an optimal transition sequence for each sentence x with gold tree $G_{\text{gold}} = (V_x, A_{\text{gold}})$.

Algorithm 1 defines the standard oracle function used to find the next transition t for a configuration c and gold tree $G_{\text{gold}} = (V_x, A_{\text{gold}})$. This algorithm is provably correct in the sense that, for every sentence x and projective dependency tree $G_{\text{gold}} = (V_x, A_{\text{gold}})$, if we initialize c to $c_s(x)$ and repeatedly execute the oracle transition, then we derive exactly $G_{\text{gold}} = (V_x, A_{\text{gold}})$ (Nivre, 2006). Nevertheless, it has two important limitations.

Algorithm 1 Standard oracle for arc-eager dependency parsing

if $c = (\sigma|i, j|\beta, A)$ and $(j, l, i) \in A_{\text{gold}}$ **then**
    $t \leftarrow \text{LEFT-ARC}_l$
else if $c = (\sigma|i, j|\beta, A)$ and $(i, l, j) \in A_{\text{gold}}$ **then**
    $t \leftarrow \text{RIGHT-ARC}_l$
else if $c = (\sigma|i, j|\beta, A)$ and $\exists k[k < i \wedge \exists l[(k, l, j) \in A_{\text{gold}} \vee (j, l, k) \in A_{\text{gold}}]]$ **then**
    $t \leftarrow \text{REDUCE}$
else
    $t \leftarrow \text{SHIFT}$
return t

The first is that it ignores spurious ambiguity in the transition system, that is, cases where a given dependency tree can be derived in more than one way. The dependency tree in Figure 1 is derived by two distinct transition sequences:[3]

(1) SH, LA_{SBJ}, RA_{PRD}, RA_{IOBJ}, SH, LA_{DET}, RE, RA_{DOBJ}, RE RA_{P}
(2) SH, LA_{SBJ}, RA_{PRD}, RA_{IOBJ}, RE, SH, LA_{DET}, RA_{DOBJ}, RE RA_{P}

Algorithm 1 will predict (1) but not (2). More generally, whenever there is a SH-RE ambiguity, which is the only ambiguity that exists in the arc-eager system, the oracle prediction will always be SH. In this way, the oracle implicitly defines a canonical transition sequence for every tree.

The second limitation is that we have no guarantee for what happens if we apply the oracle to a configuration that does not belong to the canonical transition sequence. In fact, it is easy to show that the oracle prediction in such cases can be suboptimal. For example, suppose that we erroneously choose the SH transition instead of RA_{IOBJ} after the first three transitions in sequence (1). This results in the following parser configuration:

$$([0, 2, 3], [4, 5, 6], \{(0, \text{PRD}, 2), (2, \text{SBJ}, 1)\})$$

Starting from this configuration, the oracle defined by Algorithm 1 will predict SH, LA_{DET}, SH, SH, which derives the dependency graph in the left-hand side of Figure 3. Using labeled attachment score to measure loss, this graph has a loss of 3 compared to the correct tree in Figure 1, since it fails to include the arcs $(2, \text{IOBJ}, 3)$, $(2, \text{DOBJ}, 5)$, $(2, \text{P}, 6)$.[4] However, if we instead apply the transitions SH, LA_{DET}, LA_{DET}, RA_{DOBJ}, RE, RA_{P}, we end up with the tree in the right-hand side of Figure 3, which only has a loss of 1.

We say that Algorithm 1 defines a *static* oracle, because it produces a single static sequence of transitions that is supposed to be followed in its entirety. The main contribution of this paper is the notion of a *dynamic* oracle, which does not presuppose a single canonical transition sequence for each dependency tree and which can dynamically adapt to arbitrary configurations that arise during parsing and still make optimal predictions.

[3] To save space, we sometimes use the following abbreviations: $\text{LA}_l = \text{LEFT-ARC}_l$, $\text{RA}_l = \text{RIGHT-ARC}_l$, RE = REDUCE, SH = SHIFT.

[4] In most practical parser implementations, this graph is converted into a tree by adding arcs from the root node to all words that lack a head. However, the loss will be exactly the same.

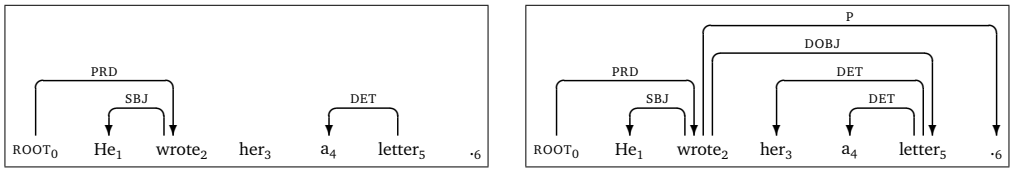

Figure 3: Dependency graphs with loss 3 (left) and loss 1 (right)

3 Dynamic Parsing Oracles

We want to define an oracle that (a) allows more than one transition sequence for a given tree and (b) makes optimal predictions in all configurations (not only configurations that are part of a globally optimal transition sequence). It follows from the first requirement that the oracle should define a *relation* from configurations to transitions, rather than a function, and we will represent it as a boolean function $o(t;c,G_{\text{gold}})$, which returns **true** just in case t is optimal in c relative to G_{gold}. But what does it mean for a transition to be optimal?

Intuitively, a transition t is optimal if it does not commit us to a parsing error, which we take to mean that the best dependency tree reachable from c is also reachable from $t(c)$. Consider the set of all dependency trees that are reachable from c. From this set, we pick the set of trees that minimize some loss function relative to G_{gold} and say that t is optimal if and only if at least one tree in this set is still reachable from $t(c)$. More precisely, we define the *cost* of t to be the difference in loss between the best tree reachable in c and the best tree reachable in $t(c)$ and say that t is optimal if it has zero cost. In the special case where the gold tree G_{gold} is reachable from c, the set of trees that minimize the loss function is the singleton set containing G_{gold}, which entails that t is optimal if and only if G_{gold} is still reachable from $t(c)$. Let us now try to make this precise.

3.1 Defining the Oracle

First, we define the *loss* $\mathscr{L}(G,G_{\text{gold}})$ of a dependency graph $G=(V_x,A)$ with respect to the gold tree $G_{\text{gold}}=(V_x,A_{\text{gold}})$ to be the number of arcs that are in G_{gold} but not in G:

$$\mathscr{L}(G,G_{\text{gold}}) = |A_{\text{gold}} \setminus A|$$

We then say that a dependency graph $G=(V_x,A)$ for a sentence x is *reachable* from a non-terminal configuration c for x, written $c \rightsquigarrow G$ if and only if there is a non-empty sequence of transitions $t_1,\ldots,t_m$ such that $[t_m \circ \cdots \circ t_1](c) = (\Sigma,[],A)$ and $G=(V_x,A)$.

Next, we define the *cost* $\mathscr{C}(t;c,G_{\text{gold}})$ of the transition t in the configuration c relative to the gold tree G_{gold} as the loss difference between the minimum loss tree reachable before and after the transition:

$$\mathscr{C}(t;c,G_{\text{gold}}) = \left[\min_{G:t(c)\rightsquigarrow G} \mathscr{L}(G,G_{\text{gold}})\right] - \left[\min_{G:c\rightsquigarrow G} \mathscr{L}(G,G_{\text{gold}})\right]$$

Note that, by definition, there must be at least one zero cost transition for every configuration c and gold tree G_{gold}. To see why, let G be some dependency graph with minimum loss reachable from c. Since G is reachable from c, there must be at least one transition t such that G is reachable from $t(c)$. And since $\mathscr{L}(G,G_{\text{gold}}) - \mathscr{L}(G,G_{\text{gold}}) = 0$, it follows that $\mathscr{C}(t;c,G_{\text{gold}}) = 0$.

Finally, we define the oracle $o(t; c, G_{\text{gold}})$ to return **true** just in case t has zero cost relative to c and G_{gold}:

$$o(t; c, G_{\text{gold}}) = \begin{cases} \textbf{true} & \text{if } \mathscr{C}(t; c, G_{\text{gold}}) = 0 \\ \textbf{false} & \text{otherwise} \end{cases}$$

3.2 A Dynamic Oracle for Arc-Eager Parsing

In order to implement the dynamic oracle in practice, we need an efficient method for computing the cost of each transition in a given configuration. A key property of the arc-eager system (stated here without proof) is that a dependency graph $G = (V_x, A)$ is reachable from a configuration c if G is a projective forest and if each individual arc in A is reachable in c. In the Arc-Eager system, an arc (i, l, j) is reachable in $c = (\Sigma, B, A')$ if either (i, l, j) is already in A' (since arcs can never be removed) or if $\min(i, j)$ is in Σ or B, $\max(i, j)$ is in B, and there is no arc in A that already assigns a head to j (since it is always possible to reach a later configuration where $\min(i, j)$ is at the top of the stack and $\max(i, j)$ is at the head of the buffer, in which case the arc can be added in a LEFT-ARC$_l$ or RIGHT-ARC$_l$ transition).

Given that our loss function (and hence our cost function) also decomposes into the loss of individual arcs, we can compute the cost of each transition by simply counting the gold arcs that are no longer reachable after that transition. We do this on a case by case basis. In all the cases, we assume a configuration c of the form $c = (\sigma|s, b|\beta, A)$.[5]

- $\mathscr{C}$(LEFT-ARC$_l$; c, G_{gold}): Adding the arc (b, l, s) and popping s from the stack means that s will not be able to acquire any head or dependents in β. The cost is therefore the number of arcs in A_{gold} of the form (k, l', s) or (s, l', k) such that $k \in \beta$. Note that the cost is 0 for the trivial case where $(b, l, s) \in A_{\text{gold}}$, but also for the case where b is not the gold head of s but the real head is not in β (due to an erroneous previous transition) and there are no gold dependents of s in β.[6]

- $\mathscr{C}$(RIGHT-ARC$_l$; c, G_{gold}): Adding the arc (s, l, b) and pushing b onto the stack means that b will not be able to acquire any head in σ or β, nor any dependents in σ. The cost is therefore the number of arcs in A_{gold} of the form (k, l', b), such that $k \in \sigma$ or $k \in \beta$, or of the form (b, l', k) such that $k \in \sigma$. Note again that the cost is 0 for the trivial case where $(s, l, b) \in A_{\text{gold}}$, but also for the case where s is not the gold head of b but the real head is not in σ or β (due to an erroneous previous transition) and there are no gold dependents of b in σ.

- $\mathscr{C}$(REDUCE; c, G_{gold}): Popping s from the stack means that s will not be able to acquire any dependents in $b|\beta$. The cost is therefore the number of arcs in A_{gold} of the form (s, l', k) such that $k \in b|\beta$. While it may seem that a gold arc of the form (k, l, s) should be accounted for as well, note that a gold arc of that form, if it exists, is already accounted for by a previous (erroneous) RIGHT-ARC$_l$ transition when s acquired its head.

- $\mathscr{C}$(SHIFT; c, G_{gold}): Pushing b onto the stack means that b will not be able to acquire any head or dependents in $s|\sigma$. The cost is therefore the number of arcs in A_{gold} of the form (k, l', b) or (b, l', k) such that $k \in s|\sigma$.

[5] This is a slight abuse of notation, since for the SHIFT transition s may not exist, and for the REDUCE transition b may not exist.

[6] One may want to associate a lower cost with cases in which the arc endpoints are correct and only the label is wrong. This extension is trivial.

$$
\begin{aligned}
&\mathscr{C}(\text{LA}_l;\ c=(\sigma|s,b|\beta,A),G_g) &&= \left|\{(k,l',s)\in A_g \mid k\in\beta\}\cup\{(s,l',k)\in A_g \mid k\in\beta\}\right| \\
&\mathscr{C}(\text{RA}_l;\ c=(\sigma|s,b|\beta,A),G_g) &&= \left|\{(k,l',b)\in A_g \mid k\in\sigma\vee k\in\beta\}\cup\{(b,l',k)\in A_g \mid k\in\sigma\}\right| \\
&\mathscr{C}(\text{RE};\ c=(\sigma|s,\beta,A),G_g) &&= \left|\{(s,l',k)\in A_g \mid k\in\beta\}\right| \\
&\mathscr{C}(\text{SH};\ c=(\sigma,b|\beta,A),G_g) &&= \left|\{(k,l',b)\in A_g \mid k\in\sigma\}\cup\{(b,l',k)\in A_g \mid k\in\sigma\}\right|
\end{aligned}
$$

Figure 4: Transition costs for the arc-eager transition system with gold tree $G_g = (V_x, A_g)$.

The computation of transition costs is summarized in Figure 4. We can now return to the example in Section 2.2 and analyze the behavior of the static oracle in the presence of erroneous transitions. In the example there, the last two SHIFT transitions predicted by the static oracle each has a cost of 1, because they each lose an arc going into the first word of the buffer. By contrast, the transition LEFT-ARC$_{\text{DET}}$ in place of the first SHIFT has a cost of 0, despite the fact that the arc $(5, \text{DET}, 3)$ is not in the gold tree, because it does not eliminate any gold arc that is still reachable – the cost of the incorrect attachment is already accounted for in the cost of the erroneous SHIFT action.

After defining the concept of a *dynamic oracle* which is correct over the entire configuration space of a transition system and providing a concrete instantiation of it for the arc-eager transition system, we now go on to present one useful application of such an oracle.

4 Training Parsers with the Dynamic Oracle

Greedy transition-based parsers trained with static oracles are known to suffer from error propagation (McDonald and Nivre, 2007). We may hope to mitigate the error propagation problem by letting the parser explore larger portions of the configuration space during training and learn how to behave optimally also after committing previous errors. While this is not possible with the usual static oracles, the dynamic oracle defined above allows us to do just that, as it returns a set of optimal transitions for each possible configuration.

Algorithm 2 is a standard online training algorithm for transition-based dependency parsers using a static oracle. Given a training sentence x with gold tree G_{gold}, it starts in the initial configuration $c_s(x)$ and repeatedly predicts a transition t_p given its current feature weights $\mathbf{w}$ and compares this to the transition t_o predicted by the static oracle. If the model prediction is different from the oracle prediction, the feature weights are updated, but the new configuration is always derived using the oracle transition (line 10), which means that only configurations in the canonical oracle-induced transition sequence are explored.[7]

Algorithm 3 is a modification of the standard algorithm which makes use of the dynamic oracle to explore a larger part of the configuration space. The first difference is in line 7, where the new algorithm, instead of getting the single prediction of the static oracle, gets the set of

[7]Some readers may be more familiar with a two-stage process in which first the oracle is used to create oracle transition sequences from the entire training set, which are then transformed to individual training examples and passed on to an external classifier. This process is equivalent to the one in Algorithm 2 in case the external classifier is a multiclass perceptron (or any other online classifier), with the only difference being that the training examples are generated on-the-fly whenever they are needed. The online formulation is used to facilitate a smooth transition to Algorithm 3.

Algorithm 2 Online training with a static oracle

```
1:  w ← 0
2:  for I = 1 → ITERATIONS do
3:      for sentence x with gold tree G_gold in corpus do
4:          c ← c_s(x)
5:          while c is not terminal do
6:              t_p ← arg max_t w · φ(c, t)
7:              t_o ← o(c, G_gold)
8:              if t_p ≠ t_o then
9:                  w ← w + φ(c, t_o) − φ(c, t_p)
10:             c ← t_o(c)
11: return w
```

transitions that have zero cost according to the dynamic oracle. The weight update is then performed only if the model prediction does not have zero cost (lines 9–10), which means that updates no longer need to reflect a single canonical transition sequence. Finally, the transition used to update the parser configuration is no longer the single transition predicted by the static oracle, but a transition that is chosen by the function CHOOSE_NEXT, which may be a transition that does not have zero-cost (lines 11-12). In our current implementation, CHOOSE_NEXT is conditioned on the predicted transition t_p, the set of zero cost transitions, and the iteration number. However, more elaborate conditioning schemes are also possible.

We propose two versions of the CHOOSE_NEXT function. In the first version, $\text{CHOOSE\_NEXT}_{\text{AMB}}$, the training algorithm only follows optimal (zero cost) transitions but permits spurious ambiguity by following the model prediction t_p if this has zero cost and a random zero cost transition otherwise. In the second version, $\text{CHOOSE\_NEXT}_{\text{EXP}}$, the training algorithm also explores non-optimal transitions. More precisely, after the first k training iterations, it follows the model prediction t_p regardless of its cost in 100(1-p)% of the cases and falls back on $\text{CHOOSE\_NEXT}_{\text{AMB}}$ in the remaining cases. It is worth noting that Algorithm 3 subsumes Algorithm 2 as a special case if we define ZERO_COST to contain only the prediction t_o of the static oracle, and define CHOOSE_NEXT to always return t_o.

The novel training algorithm presented here is based on perceptron learning.[8] Since the dynamic oracle provides a cost for every transition-configuration pair, it could be used also for cost-sensitive learning. Our preliminary attempts with cost-sensitive learning through the max-loss and prediction-based passive-aggressive algorithms of Crammer et al. (2006) show that the cost-sensitive variants of the algorithms indeed improve upon the non-cost-sensitive variants. However, the best passive-aggressive results were still significantly lower than those obtained using the averaged perceptron. We do not elaborate on cost-sensitive training in this work, and leave this direction for future investigation.

5 Experiments

We present experiments comparing greedy arc-eager transition-based parsers trained (a) using the static oracle (Algorithm 2), (b) using the dynamic oracle with spurious ambiguity (Algorithm 3 with $\text{CHOOSE\_NEXT}_{\text{AMB}}$), and (c) using the dynamic oracle with spurious ambiguity

[8] In practice, we use an averaged perceptron, although this is not reflected in the algorithm descriptions above.

Algorithm 3 Online training with a dynamic oracle

```
1:  w ← 0
2:  for I = 1 → ITERATIONS do
3:      for sentence x with gold tree G_gold in corpus do
4:          c ← c_s(x)
5:          while c is not terminal do
6:              t_p ← arg max_t w · φ(c, t)
7:              ZERO_COST ← {t | o(t; c, G_gold) = true}
8:              t_o ← arg max_{t∈ZERO_COST} w · φ(c, t)
9:              if t_p ∉ ZERO_COST then
10:                 w ← w + φ(c, t_o) − φ(c, t_p)
11:             t_n ← CHOOSE_NEXT(I, t_p, ZERO_COST)
12:             c ← t_n(c)
13: return w
```

```
1:  function CHOOSE_NEXT_AMB(I, t, ZERO_COST)
2:      if t ∈ ZERO_COST then
3:          return t
4:      else
5:          return RANDOM_ELEMENT(ZERO_COST)
```

```
1:  function CHOOSE_NEXT_EXP(I, t, ZERO_COST)
2:      if I > k and RAND() > p then
3:          return t
4:      else
5:          return CHOOSE_NEXT_AMB(I, t, ZERO_COST)
```

and non-optimal transitions (Algorithm 3 with CHOOSE_NEXT$_{\text{EXP}}$). We evaluate the models on a wide range of English data sets, as well as the data sets from the CoNLL 2007 shared task on multilingual dependency parsing (Nivre et al., 2007).

The parser is a greedy transition-based parser using the arc-eager transition system of Nivre (2003, 2008) with the feature representations defined by Zhang and Nivre (2011). As our primary goal is to compare the training methods, and not to achieve the highest possible score for each data set, we use the exact same feature representations and training parameters across all experiments. Specifically, we train an averaged perceptron model for 15 iterations. When using CHOOSE_NEXT$_{\text{EXP}}$, we set $k = 2$ and $p = 0.1$, meaning that the algorithm allows non-optimal transitions in 90% of the cases, starting from the third training iteration. Note that many of these transitions will nevertheless be correct, as the first training iterations already put the model in a good region of the parameter space.

The English model is trained on Sections 2–21 of the Penn-WSJ Treebank (Marcus et al., 1993), converted to Stanford basic dependencies (de Marneffe et al., 2006), with part-of-speech tags assigned by a structured-perceptron tagger trained on the same corpus with 4-fold jack-knifing. We use Section 22 to tune parameters, and we evaluate on the following data sets, which are also converted to the same dependency scheme, and pos-tagged using the same

	WSJ22	WSJ23	BNC	BRN	FTBL	QTB	ANS	EML	GRPS	REVS	BLGS
Unlabeled Attachment Scores											
Static	90.31	89.88	82.79	85.11	78.85	86.80	78.81	79.23	81.21	80.61	83.40
Dynamic-ambiguity	90.42	90.18	82.98	85.41	79.36	87.29	79.19	79.56	81.18	80.96	83.61
Dynamic-explore	91.24	90.96	84.17	86.22	80.04	87.50	80.21	80.04	82.08	81.81	84.72
Labeled Attachment Scores											
Static	87.88	87.69	78.47	81.15	74.69	73.69	73.60	74.95	77.15	75.87	79.66
Dynamic-ambiguity	87.95	87.83	78.69	81.47	75.05	73.91	73.90	75.29	77.16	76.19	79.91
Dynamic-explore	88.76	88.72	79.75	82.30	75.82	74.36	74.95	75.85	78.11	77.06	81.09

Table 1: Results on the English data sets

structured-perceptron tagger, trained on the entire training set.

- WSJ22: Section 22 of the Penn-WSJ Treebank (development set).
- WSJ23: Section 23 of the Penn-WSJ Treebank (test set).
- BNC: 1,000 manually annotated sentences from the British National Corpus (Foster and van Genabith, 2008).
- BRN: The entire Brown Corpus (Kucera and Francis, 1967).
- FTBL: The entire Football Corpus (Foster et al., 2011).
- QB: The entire QuestionBank (Judge et al., 2006).
- ANS, EML, GRPS, REVS, BLGS: the question-answers, emails, newsgroups, reviews and weblogs portions of the English Web Treebank (Bies et al., 2012; Petrov and McDonald, 2012).

The CoNLL models are trained on the dedicated training set for each of the ten languages and evaluated on the corresponding test set, with gold standard part-of-speech tags in both cases. Since the arc-eager parser can only handle projective dependency trees, all trees in the training set are projectivized before training, using the baseline pseudo-projective transformation in Nivre and Nilsson (2005). However, non-projective trees are kept intact in the test sets for evaluation. We include all ten languages from the CoNLL 2007 shared task:

- ARA: Arabic (Hajič et al., 2004)
- BAS: Basque (Aduriz et al., 2003)
- CAT: Catalan (Martí et al., 2007)
- CHI: Chinese (Chen et al., 2003)
- CZE: Czech (Hajič et al., 2001; Böhmová et al., 2003)
- ENG: English (Marcus et al., 1993)
- GRE: Greek (Prokopidis et al., 2005)
- HUN: Hungarian (Czendes et al., 2005)

	ARA	BAS	CAT	CHI	CZE	ENG	GRE	HUN	ITA	TUR
Unlabeled Attachment Scores										
Static	80.60	74.10	91.21	84.13	78.00	86.24	79.16	77.75	84.11	79.02
Dynamic-ambiguity	80.72	74.90	91.09	83.62	78.38	86.83	79.48	76.17	84.52	78.97
Dynamic-explore	83.06	76.10	92.01	84.65	79.54	88.81	80.66	77.10	84.77	78.84
Labeled Attachment Scores										
Static	71.04	64.42	85.96	79.75	69.49	84.90	70.94	68.10	79.93	68.80
Dynamic-ambiguity	71.06	65.18	85.73	79.24	69.39	85.56	71.88	66.99	80.63	68.58
Dynamic-explore	73.54	66.77	86.60	80.74	71.32	87.60	73.83	68.23	81.02	68.76

Table 2: Results on the CoNLL 2007 data sets

- ITA: Italian (Montemagni et al., 2003)
- TUR: Turkish (Oflazer et al., 2003)

Table 1 gives the results for the English model, while Table 2 presents the multilingual evaluation. In both cases, we present unlabeled and labeled attachment scores excluding punctuation.

For the English data sets, we see that adding spurious ambiguity (Dynamic-ambiguity) generally improves both labeled and unlabeled attachment scores by up to 0.5 percent absolute. The only exception is GRPS, where there is a small decrease in unlabeled attachment score. When the training procedure in addition explores non-optimal transitions (Dynamic-explore), the improvement is even greater, in some cases up to about 1.5 percentage points, which is quite substantial given that our baseline parser already performs at the state-of-the-art level for greedy deterministic transition-based parsers on English Stanford-dependencies.[9]

For the CoNLL data sets, results for the Dynamic-ambiguity condition are mixed causing a drop in accuracy for some data sets, but the Dynamic-explore condition makes up for it and brings substantial improvement in accuracy for all except two languages. We see very substantial improvements in both UAS and LAS for Arabic, Basque, Czech, English and Greek, as well as improvements for Catalan, Chinese and Italian. The average LAS improvement across all the CoNLL datasets is 1.5 LAS points. The only two exceptions are Hungarian and Turkish, where unlabeled attachment scores drop slightly as a result of not using the static oracle, and labeled attachment scores are practically unaffected. More analysis is needed to find out what is going on for these languages, but it is likely that results could be improved with language-specific tuning.[10]

Overall, the experimental results show a considerable improvement in the accuracy of deterministic linear-time classifier-based dependency parsing through training procedures that explore a larger part of the search space than traditional methods based on static oracles.

[9] Note that the web data sets (ANS, EML, GRPS, REVS, BLGS) are annotated according to the Ontonotes corpus guidelines, which are somewhat different than the original Penn Treebank guidelines used in the training corpora. In particular, base-NPs in the web data sets are more nested. Our scores on these data sets are thus artificially lower than they could be. We could get better scores for these data sets for all training conditions by training on the Ontonotes corpora instead, but as our main concern is not in achieving the best scores, we opted for the simpler experimental setup.

[10] Language-specific tuning is likely to improve results for the other languages as well – we did not perform any language-specific tuning, and it is well established that individual languages parsing accuracies can greatly benefit from tuning of the feature set, the transition system being used and the learning parameters (Hall et al., 2007).

6 Related Work

Deterministic classifier-based dependency parsing is an instance of independent sequential classification-based structured prediction. In sequential classification models, such as Maximum-Entropy Markov Models (McCallum et al., 2000), a structured output is produced by repeated application of a locally trained classifier, where each classification decision is conditioned on the structure created by previous decisions. Several methods have been developed to cope with error propagation in sequential classification, including stacked sequential learning (Cohen and Carvalho, 2005), LaSO (Daumé III and Marcu, 2005), Searn (Daumé III et al., 2009) and its followers SMILe (Ross and Bagnell, 2010) and DAgger (Ross et al., 2011).

While stacked learning is well suited for sequence prediction tasks such as tagging and chunking, it is not clear how to apply it to parsing.[11] Searn and the closely related DAgger algorithm are more promising for dealing with the complexity of dependency parsing, but it appears that previous attempts to apply Searn-based learning to dependency parsing have been unsuccessful. A key component in the specification of a Searn learning algorithm is an *optimal policy* mapping states in the search space (such as parser configurations) to optimal outcomes (such as transitions). Attempts to approximate the optimal policy for parsing using static oracles are unlikely to work very well, since a static oracle is only correct for a small subset of the search space. The dynamic oracle introduced in this paper, which is correct for arbitrary parser configurations, can be used to define an optimal policy for Searn-based learning. Both Searn and DAgger require several complete training rounds over the entire data set and take a relatively long time to train. We instead use a simpler online algorithm which can be viewed as a stochastic approximation of the DAgger algorithm, which is itself heavily inspired by the Searn algorithm.

Recent work on beam search and structured prediction for transition-based dependency parsing has shown that parsing accuracy can be improved considerably if models are trained to perform beam search instead of greedy one-best search, and if training is done using a global structured learning objective instead of local learning of individual decisions (Zhang and Clark, 2008; Zhang and Nivre, 2011; Bohnet and Kuhn, 2012; Huang et al., 2012). Like our method, beam search with global structured learning mitigates the effects of error propagation by exploring non-canonical configurations at training time, but the use of a beam reduces parsing speed by a factor that is roughly proportional to the size of the beam, making parsing less efficient. Our method in contrast still trains classifiers to perform local decisions, and thus incurs no efficiency penalty at parsing time, but each local decision is trained to take into account the consequences of previous, possibly erroneous, decisions. Although we may not be able to reach the accuracy level of a beam-search parser, we show that a substantial improvement in accuracy is possible also for a purely deterministic classifier-based parser, making our method suitable for training accurate parsers in situations where maximum efficiency is needed, e.g., when there is a need to process very large corpora. Integrating our dynamic oracle in the training procedure for a transition-based parser with beam search is an interesting question for future work.

The work that probably comes closest to ours is Choi and Palmer (2011), who improve the accuracy of a greedy transition-based dependency parser through an iterative training procedure that they call bootstrapping. They start by training a classifier using a standard static oracle for a hybrid transition system combining elements of the arc-eager system and the algorithm of

[11] Stacked learning has been explored to some extent in the context of parsing for integrating approximate higher order features as well as for combining the predictions of different parsers (Nivre and McDonald, 2008; Martins et al., 2008).

Covington (2001). In a second step, they then use this classifier to parse the training corpus, creating one new training instance for every configuration visited during parsing, using an adapted version of the static oracle to predict the optimal transition for each configuration. They iterate this procedure as long as parsing accuracy improves on a held-out development set and report improvements in parsing accuracy of about 0.5 percent absolute for English and almost 2 percent absolute for Czech. The main difference compared to our approach, except for the fact that they use a different transition system, is that their method for finding the optimal transition after the first training round is heuristic and does not guarantee that the best parse is still reachable.

Finally, Cohen et al. (2012) tackle the problem of spurious ambiguity for static oracles by eliminating ambiguity from the underlying transition system instead of modifying the oracle. They show how this can be achieved for the arc-standard system of Nivre (2004) as well as the non-projective extension by Attardi (2006). It is still an open question whether their technique can also be applied to the arc-eager system targeted in this paper.

7 Conclusion

We have highlighted the shortcoming of traditional static oracles used to train transition-based dependency parsers, and instead proposed the notion of a *dynamic oracle*, which allows more than one correct transition sequence in the case of spurious ambiguity, and which can predict an optimal transition also for non-optimal configurations. We have defined a concrete dynamic oracle for the arc-eager transition system and showed how it can be used in online training of greedy deterministic parsers.

Greedy deterministic transition-based dependency parsers are among the most efficient systems available for syntactic parsing of natural language. In terms of parsing accuracy, they perform near the state-of-the-art level for many languages but tend to suffer from prediction errors and subsequent error propagation. This problem can be mitigated by using our proposed training method. Experimental results for English show consistent improvements in parsing accuracy of up to almost 1.5 percent absolute on a wide range of data sets. Experimental results on the ten languages from the CoNLL 2007 shared task on dependency parsing show significant improvements of up to almost 3 LAS points for some languages, but there are also few cases where we see little or no improvement in parsing accuracy, a phenomenon that requires further investigation. Other topics for future research is the effective use of cost-sensitive learning instead of the perceptron loss used in this paper, the derivation of dynamic oracles for other transition systems, and utilizing the dynamic oracles in non-greedy settings, such as beam-search parsers.

References

Aduriz, I., Aranzabe, M. J., Arriola, J. M., Atutxa, A., Díaz de Ilarraza, A., Garmendia, A., and Oronoz, M. (2003). Construction of a Basque dependency treebank. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 201–204.

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

Bies, A., Mott, J., Warner, C., and Kulick, S. (2012). English web treebank. Linguistic Data Consortium, LDC2012T13.

Böhmová, A., Hajič, J., Hajičová, E., and Hladká, B. (2003). The Prague Dependency Treebank: A three-level annotation scenario. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 103–127. Kluwer.

Bohnet, B. and Kuhn, J. (2012). The best of bothworlds – a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87.

Chen, K., Luo, C., Chang, M., Chen, F., Chen, C., Huang, C., and Gao, Z. (2003). Sinica treebank: Design criteria, representational issues and implementation. In Abeillé, A., editor, *Treebanks*, pages 231–248. Kluwer.

Choi, J. D. and Palmer, M. (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*, pages 687–692.

Cohen, S. B., Gómez-Rodríguez, C., and Satta, G. (2012). Elimination of spurious ambiguity in transition-based dependency parsing. Technical report.

Cohen, W. W. and Carvalho, V. R. (2005). Stacked sequential learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 671–676.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Czendes, D., Csirik, J., Guimóthy, and Kocsor, A. (2005). *The Szeged Treebank*. Springer.

Daumé III, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine Learning*, 75:297–325.

Daumé III, H. and Marcu, D. (2005). Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 169–176.

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

Foster, J., ÇetinoÄŸlu, O., Wagner, J., and van Genabith, J. (2011). Comparing the use of edited and unedited text in parser self-training. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 215–219.

Foster, J. and van Genabith, J. (2008). Parser evaluation and the BNC: Evaluating 4 constituency parsers with 3 metrics. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC)*.

Goldberg, Y. and Elhadad, M. (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.

Hajič, J., Smrž, O., Zemánek, P., Šnaidauf, J., and Beška, E. (2004). Prague Arabic Dependency Treebank: Development in data and tools. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*.

Hajič, J., Vidova Hladka, B., Panevová, J., Hajičová, E., Sgall, P., and Pajas, P. (2001). Prague Dependency Treebank 1.0. LDC, 2001T10.

Hall, J., Nilsson, J., Nivre, J., Eryiğit, G., Megyesi, B., Nilsson, M., and Saers, M. (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.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of Human Language Technologies: The 2012 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*.

Judge, J., Cahill, A., and van Genabith, J. (2006). Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 497–504. Association for Computational Linguistics.

Kucera, H. and Francis, W. N. (1967). *Computational Analysis of Present-Day American English*. Brown University Press.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330.

Martí, M. A., Taule, M., Màrquez, L., and Bertran, M. (2007). CESS-ECE: A multilingual and multilevel annotated corpus. Available for download from: http://www.lsi.upc.edu/∼mbertran/cess-ece/.

Martins, A. F., Das, D., Smith, N. A., and Xing, E. P. (2008). Stacking dependency parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 157–166.

McCallum, A., Freitag, D., and Pereira, F. (2000). Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 591–598.

McDonald, R. and Nivre, J. (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

Montemagni, S., Barsotti, F., Battista, M., Calzolari, N., Corazzari, O., Lenci, A., Zampolli, A., Fanciulli, F., Massetani, M., Raffaelli, R., Basili, R., Pazienza, M. T., Saracino, D., Zanzotto, F., Nana, N., Pianesi, F., and Delmonte, R. (2003). Building the Italian Syntactic-Semantic Treebank. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 189–210. Kluwer.

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

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 50–57.

Nivre, J. (2006). *Inductive Dependency Parsing*. Springer.

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

Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007). The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

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

Nivre, J. and McDonald, R. (2008). Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 950–958.

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Oflazer, K., Say, B., Hakkani-Tür, D. Z., and Tür, G. (2003). Building a Turkish treebank. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 261–277. Kluwer.

Petrov, S. and McDonald, R. (2012). Overview of the 2012 shared task on parsing the web. In *Notes on the First Workshop on Syntatctic Analysis for Non-Canonical Language*.

Prokopidis, P., Desypri, E., Koutsombogera, M., Papageorgiou, H., and Piperidis, S. (2005). Theoretical and practical issues in the construction of a Greek dependency treebank. In *Proceedings of the 3rd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 149–160.

Ross, S. and Bagnell, J. A. (2010). Efficient reductions for imitation learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*.

Ross, S., Gordon, G. J., and Bagnell, J. A. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*.

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

Zhang, Y. and Clark, S. (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.

Zhang, Y. and Nivre, J. (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.