An Efficient Dynamic Oracle for Unrestricted Non-Projective Parsing

Carlos Gómez-Rodríguez
Departamento de Computación
Universidade da Coruña
Campus de Elviña, s/n
15071 A Coruña, Spain
carlos.gomez@udc.es

Daniel Fernández-González
Departamento de Informática
Universidade de Vigo
Campus As Lagoas, s/n
32004 Ourense, Spain
danifg@uvigo.es

Abstract

We define a dynamic oracle for the Covington non-projective dependency parser. This is not only the first dynamic oracle that supports arbitrary non-projectivity, but also considerably more efficient ($O(n)$) than the only existing oracle with restricted non-projectivity support. Experiments show that training with the dynamic oracle significantly improves parsing accuracy over the static oracle baseline on a wide range of treebanks.

1 Introduction

Greedy transition-based dependency parsers build analyses for sentences incrementally by following a sequence of transitions defined by an automaton, using a scoring model to choose the best transition to take at each state (Nivre, 2008). While this kind of parsers have become very popular, as they achieve competitive accuracy with especially fast parsing times; their raw accuracy is still behind that of slower alternatives like transition-based parsers that use beam search (Zhang and Nivre, 2011; Choi and McCallum, 2013). For this reason, a current research challenge is to improve the accuracy of greedy transition-based parsers as much as possible without sacrificing efficiency.

A relevant recent advance in this direction is the introduction of dynamic oracles (Goldberg and Nivre, 2012), an improvement in the training procedure of greedy parsers that can boost their accuracy without any impact on parsing speed. An oracle is a training component that selects the best transition(s) to take at a given configuration, using knowledge about the gold tree. Traditionally, transition-based parsers were trained to follow a so-called static oracle, which is only defined on the configurations of a canonical computation that generates the gold tree, returning the next transition in said computation. In contrast, dynamic oracles are non-deterministic (not limited to one sequence, but supporting all the possible computations leading to the gold tree), and complete (also defined for configurations where the gold tree is unreachable, choosing the transition(s) that lead to a tree with minimum error). This extra robustness in training provides higher parsing accuracy.

However, defining a usable dynamic oracle for a given parser is non-trivial in general, due to the need of calculating the loss of each configuration, i.e., the minimum Hamming loss to the gold tree from a tree reachable from that configuration. While it is always easy to do this in exponential time by simulating all possible computations in the algorithm to obtain all reachable trees, it is not always clear how to achieve this calculation in polynomial time. At the moment, this problem has been solved for several projective parsers exploiting either arc-decomposability (Goldberg and Nivre, 2013) or tabularization of computations (Goldberg et al., 2014). However, for parsers that can handle crossing arcs, the only known dynamic oracle (Gómez-Rodríguez et al., 2014) has been defined for a variant of the parser by Attardi (2006) that supports a restricted set of non-projective trees. To our knowledge, no dynamic oracles are known for any transition-based parser that can handle unrestricted non-projectivity.

In this paper, we define such an oracle for the Covington non-projective parser (Covington, 2001; Nivre, 2008), which can handle arbitrary non-projective dependency trees. As this algorithm is not arc-decomposable and its tabularization is NP-hard (Neuhaus and Bröker, 1997), we do not use the existing techniques to define dynamic oracles, but a reasoning specific to this parser. It is worth noting that, apart from being the first dynamic oracle supporting unrestricted non-projectivity, our oracle is very efficient, solving the loss calculation in $O(n)$. In contrast, the restricted non-projective oracle of Gómez-Rodríguez et al.

(2014) has $O(n^8)$ time complexity.

The rest of the paper is organized as follows: after a quick outline of Covington's parser in Sect. 2, we present the oracle and prove its correctness in Sect. 3. Experiments are reported in Sect. 4, and Sect. 5 contains concluding remarks.

2 Preliminaries

We will define a dynamic oracle for the non-projective parser originally defined by Covington (2001), and implemented by Nivre (2008) under the transition-based parsing framework. For space reasons, we only sketch the parser very briefly, and refer to the above reference for more details.

Parser configurations are of the form $c = \langle \lambda_1, \lambda_2, B, A \rangle$, where λ_1 and λ_2 are lists of partially processed words, B is another list (called the buffer) with currently unprocessed words, and A is the set of dependencies built so far. Suppose that we parse a string $w_1 \cdots w_n$, whose word occurrences will be identified with their indices $1 \cdots n$ for simplicity. Then, the parser starts at an initial configuration $c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, and executes transitions chosen from those in Figure 1 until a terminal configuration of the form $\{\langle \lambda_1, \lambda_2, [], A \rangle \in C\}$ is reached, and the sentence's parse tree is obtained from A.[1]

The transition semantics is very simple, mirroring the double nested loop traversing word pairs in the formulation by Covington (2001). When the algorithm is in a configuration $\langle \lambda_1 | i, \lambda_2, j | B, A \rangle$, we will say that it is considering the **focus words** i and j, located at the end of the first list and at the beginning of the buffer. A decision is then made about whether these two words should be linked with a rightward arc $i \rightarrow j$ (Right-Arc transition), a leftward arc $i \leftarrow j$ (Left-Arc transition) or not linked (No-Arc transition). The first two choices will be unavailable in configurations where the newly-created arc would violate the **single-head constraint** (a node cannot have more than one incoming arc) or the **acyclicity constraint** (cycles are not allowed). In any of these three transitions, i is then moved to the second list to make $i-1$ and j the focus words for the next step. Alternatively, we can choose to read a new word from the string with a Shift transition, so that the focus words in the resulting configuration will be j and $j+1$.

The result is a parser that can generate any possible dependency tree for the input, and runs in quadratic worst-case time. Although in theory this complexity can seem like a drawback compared to linear-time transition-based parsers (e.g. (Nivre, 2003; Gómez-Rodríguez and Nivre, 2013)), it has been shown by Volokh and Neumann (2012) to actually outperform linear algorithms in practice, as it allows for relevant optimizations in feature extraction that cannot be implemented in other parsers. In fact, one of the fastest dependency parsers to date uses this algorithm (Volokh, 2013).

3 The oracle

As sketched in Sect. 1, a dynamic oracle is a training component that, given a configuration c and a gold tree t_G, provides the set of transitions that are applicable in c and lead to trees with minimum Hamming loss with respect to t_G. The Hamming loss between a tree t and t_G, written $\mathcal{L}(t, t_G)$, is the number of nodes that have a different head in t than in t_G. Following Goldberg and Nivre (2013), we say that a set of arcs A is **reachable** from configuration c, written $c \rightsquigarrow A$, if there is some (possibly empty) path of transitions from c to some configuration $c' = \langle \lambda_1, \lambda_2, B, A' \rangle$, with $A \subseteq A'$. Then, we can define the loss of a configuration as

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

and the set of transitions that must be returned by a correct dynamic oracle is then

$$o_d(c, t_G) = \{\tau \mid \ell(c) - \ell(\tau(c)) = 0\},$$

i.e., the transitions that do not increase configuration loss, and hence lead to the best parse (in terms of loss) reachable from c. Therefore, implementing a dynamic oracle reduces to computing the loss $\ell(c)$ for each configuration c.

Goldberg and Nivre (2013) show that the calculation of the loss is easy for parsers that are **arc-decomposable**, i.e., those where for every configuration c and arc set A that is **tree-compatible** (i.e. that can be a part of a well-formed parse[2]), $c \rightsquigarrow A$ is entailed by $c \rightsquigarrow (i \rightarrow j)$ for every $i \rightarrow j \in A$. That is, if each arc in a tree-compatible set is individually reachable from configuration c, then that

[1]The arcs in A form a forest, but we convert it to a tree by linking any node without a head as a dependent of an artificial node at position 0 that acts as a dummy root. From now on, when we refer to some dependency graph as a tree, we assume that this transformation is being implicitly made.

[2]In the cited paper, tree-compatibility required projectivity, as the authors were dealing with projective parsers. In our case, since the parser is non-projective, tree-compatibility only consists of the single-head and acyclicity constraints.

Shift:	$\langle \lambda_1, \lambda_2, j\vert B, A\rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\vert j, [], B, A\rangle$
No-Arc:	$\langle \lambda_1\vert i, \lambda_2, B, A\rangle \Rightarrow \langle \lambda_1, i\vert\lambda_2, B, A\rangle$
Left-Arc:	$\langle \lambda_1\vert i, \lambda_2, j\vert B, A\rangle \Rightarrow \langle \lambda_1, i\vert\lambda_2, j\vert B, A \cup \{j \rightarrow i\}\rangle$ only if $\nexists k \mid k \rightarrow i \in A$ (single-head) and $i \rightarrow^* j \notin A$ (acyclicity).
Right-Arc:	$\langle \lambda_1\vert i, \lambda_2, j\vert B, A\rangle \Rightarrow \langle \lambda_1, i\vert\lambda_2, j\vert B, A \cup \{i \rightarrow j\}\rangle$ only if $\nexists k \mid k \rightarrow j \in A$ (single-head) and $j \rightarrow^* i \notin A$ (acyclicity).

Figure 1: Transitions of the Covington non-projective dependency parser.

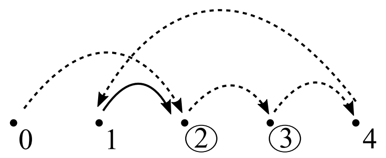

Figure 2: An example of non-arc-decomposability of the Covington parser: graphical representation of configuration $c = \langle [1,2], [], [3,4], A = \{1 \rightarrow 2\}\rangle$. The solid arc corresponds to the arc set A, and the circled indexes mark the focus words. The dashed arcs represent the gold tree t_G.

set of arcs is reachable from c. If this holds, then computing the loss of a configuration c reduces to determining and counting the gold arcs that are *not* reachable from c, which is easy in most parsers.

Unfortunately, the Covington parser is not arc-decomposable. This can be seen in the example of Figure 2: while any of the gold arcs 2→3, 3→4, 4→1 can be reachable individually from the depicted configuration, they are not jointly reachable as they form a cycle with the already-built arc 1→2. Thus, the configuration has only one individually unreachable arc (0→2), but its loss is 2.

However, it is worth noting that non-arc-decomposability in the parser is exclusively due to cycles. If a set of individually reachable arcs do not form a cycle together with already-built arcs, then we can show that the set will be reachable. This idea is the basis for an expression to compute loss based on counting individually unreachable arcs, and then correcting for the effect of cycles:

Theorem 1 *Let $c = \langle \lambda_1, \lambda_2, B, A\rangle$ be a configuration of the Covington parser, and t_G the set of arcs of a gold tree. We call $\mathcal{I}(c, t_G) = \{x \rightarrow y \in t_G \mid c \rightsquigarrow (x \rightarrow y)\}$ the set of* ***individually reachable arcs*** *of t_G; note that this set may overlap A. Conversely, we call $\mathcal{U}(c, t_G) = t_G \setminus \mathcal{I}(c, t_G)$ the set of* ***individually unreachable arcs*** *of t_G from c. Finally, let $n_c(G)$ denote the number of cycles in a graph G.*

Then $\ell(c) = |\mathcal{U}(c, t_G)| + n_c(A \cup \mathcal{I}(c, t_G))$. □

We now sketch the proof. To prove Theorem 1, it is enough to show that (1) there is at least one tree reachable from c with exactly that Hamming loss to t_G, and (2) there are no trees reachable from c with a smaller loss. To this end, we will use some properties of the graph $A \cup \mathcal{I}(c, t_G)$. First, we note that no node in this graph has in-degree greater than 1. In particular, each node except for the dummy root has exactly one head, either explicit or (if no head has been assigned in A or in the gold tree) the dummy root. No node has more than one head: a node cannot have two heads in A because the parser transitions enforce the single-head constraint, it cannot have two heads in $\mathcal{I}(c, t_G)$ because t_G must satisfy this constraint as well, and it cannot have one head in A and another in $\mathcal{I}(c, t_G)$ because the corresponding arc in $\mathcal{I}(c, t_G)$ would be unreachable due to the single-head constraint.

This, in turn, implies that the graph $A \cup \mathcal{I}(c, t_G)$ has no overlapping cycles, as overlapping cycles can only appear in graphs with in-degree greater than 1. This is the key property enabling us to exactly calculate loss using the number of cycles.

To show (1), consider the graph $A \cup \mathcal{I}(c, t_G)$. In each of its cycles, there is at least one arc that belongs to $\mathcal{I}(c, t_G)$, as A must satisfy the acyclicity constraint. We arbitrarily choose one such arc from each cycle, and remove it from the graph. Note that this results in removing exactly $n_c(A \cup \mathcal{I}(c, t_G))$ arcs, as we have shown that the cycles in $A \cup \mathcal{I}(c, t_G)$ are disjoint. We call the resulting graph $\mathcal{B}(c, t_G)$. As it has maximum in-degree 1 and it is acyclic (because we have broken all the cycles), $\mathcal{B}(c, t_G)$ is a tree, modulo our standard assumption that headless nodes are assumed to be linked to the dummy root.

This tree $\mathcal{B}(c, t_G)$ is reachable from c and has loss $\ell(c) = |\mathcal{U}(c, t_G)| + n_c(A \cup \mathcal{I}(c, t_G))$. Reachability is shown by building a sequence of transitions that will visit the pairs of words corresponding to remaining arcs in order, and intercalating the corresponding Left-Arc or Right-Arc transitions, which cannot violate the acyclicity or single-head constraints. The term $\mathcal{U}(c, t_G)$ in the loss stems from the fact that $A \cup \mathcal{I}(c, t_G)$ cannot contain arcs in $\mathcal{U}(c, t_G)$, and the term $n_c(A \cup \mathcal{I}(c, t_G))$ from not including the $n_c(A \cup \mathcal{I}(c, t_G))$ arcs that we discarded to break cycles.

Finally, from these observations, it is easy to see that $\mathcal{B}(c, t_G)$ has the best loss among reachable trees, and thus prove (2): the arcs in $\mathcal{U}(c, t_G)$ are always unreachable by definition, and for each cycle in $n_c(A \cup \mathcal{I}(c, t_G))$, the acyclicity constraint forces us to miss at least one arc. As the cycles are disjoint, this means that we necessarily miss at least $n_c(A \cup \mathcal{I}(c, t_G))$ arcs, hence $|\mathcal{U}(c, t_G)| + n_c(A \cup \mathcal{I}(c, t_G))$ is indeed the minimum loss among reachable trees. □

Thus, to calculate the loss of a configuration c, we only need to compute both of the terms in Theorem 1. For the first term, note that if c has focus words i and j (i.e., $c = \langle \lambda_1|i, \lambda_2, j|B, A\rangle$), then an arc $x \to y$ is in $\mathcal{U}(c, t_G)$ if it is not in A, and at least one of the following holds:

- $j > \max(x, y)$, as in this case we have read too far in the string and will not be able to get x and y as focus words,
- $j = \max(x, y) \wedge i < \min(x, y)$, as in this case we have $\max(x, y)$ as the right focus word but the left focus word is to the left of $\min(x, y)$, and we cannot move it back,
- there is some $z \neq 0, z \neq x$ such that $z \to y \in A$, as in this case the single-head constraint prevents us from creating $x \to y$,
- x and y are on the same weakly connected component of A, as in this case the acyclicity constraint will not let us create $x \to y$.

All of these arcs can be trivially enumerated in $O(n)$ time (in fact, they can be updated in $O(1)$ if we start from the configuration that preceded c). The second term of the loss, $n_c(A \cup \mathcal{I}(c, t_G))$, can be computed by obtaining $\mathcal{I}(c, t_G)$ as $t_G \setminus \mathcal{U}(c, t_G)$ to then apply a standard cycle-finding algorithm (Tarjan, 1972) which, for a graph with maximum in-degree 1, runs in $O(n)$ time.

Algorithm 1 presents the resulting loss calculation algorithm in pseudocode form, where CountCycles is a function that counts the number of cycles in the given graph in linear time as mentioned above. Note that the for loop runs in linear time: the condition on line 8 can be computed in constant time by recovering the head of y. The call to WeaklyConnected in line 9 finds out whether the two given nodes are weakly connected in A, and can also be resolved in $O(1)$, by querying the disjoint set data structure that implementations of the Covington algorithm commonly use for the parser's acyclicity checks (Nivre, 2008).

Algorithm 1 Computation of the loss of a configuration.

```
1:  function Loss($c = \langle \lambda_1|i, \lambda_2, j|B, A\rangle, t_G$)
2:      $U \leftarrow \emptyset$            ▷ Variable U is for $\mathcal{U}(c, t_G)$
3:      for each $x \to y \in (t_G \setminus A)$ do
4:          $left \leftarrow \min(x, y)$
5:          $right \leftarrow \max(x, y)$
6:          if $j > right \vee$
7:          $(j = right \wedge i < left) \vee$
8:          $(\exists z > 0, z \neq x : z \to y \in A) \vee$
9:          WeaklyConnected$(A, x, y)$ then
10:             $U \leftarrow u \cup \{x \to y\}$
11:     $I \leftarrow t_G \setminus U$      ▷ Variable I is for $\mathcal{I}(c, t_G)$
12:     return $|U| +$ CountCycles$(A \cup I)$
```

It is worth noting that the linear-time complexity can also be achieved by a standalone implementation of the loss calculation algorithm, without recurse to the parser's auxiliary data structures (although this is dubiously practical). To do so, we can implement WeaklyConnected so that the first call computes the connected components of A in linear time (Hopcroft and Tarjan, 1973) and subsequent calls use this information to find out if two nodes are weakly connected in constant time.

On the other hand, a more efficient implementation than the one shown in Algorithm 1 (which we chose for clarity) can be achieved by more tightly coupling the oracle to the parser, as the relevant sets of arcs associated with a configuration can be obtained incrementally from those of the previous configuration.

4 Experiments

To evaluate the performance of our approach, we conduct experiments on both static and dynamic Covington non-projective oracles. Concretely, we train an averaged perceptron model for 15 iterations on nine datasets from the CoNLL-X shared task (Buchholz and Marsi, 2006) and all data-

Unigrams
L_0w; L_0p; L_0wp; L_0l; $L_{0h}w$; $L_{0h}p$; $L_{0h}l$; $L_{0l'}w$; $L_{0l'}p$; $L_{0l'}l$; $L_{0r'}w$; $L_{0r'}p$; $L_{0r'}l$; $L_{0h2}w$; $L_{0h2}p$; $L_{0h2}l$; $L_{0l}w$; $L_{0l}p$; $L_{0l}l$; $L_{0r}w$; $L_{0r}p$; $L_{0r}l$; L_0wd; L_0pd; L_0wv_r; L_0pv_r; L_0wv_l; L_0pv_l; L_0ws_l; L_0ps_l; L_0ws_r; L_0ps_r; L_1w; L_1p; L_1wp; R_0w; R_0p; R_0wp; $R_{0l'}w$; $R_{0l'}p$; $R_{0l'}l$; $R_{0l}w$; $R_{0l}p$; $R_{0l}l$; R_0wd; R_0pd; R_0wv_l; R_0pv_l;R_0ws_l; R_0ps_l; R_1w; R_1p; R_1wp; R_2w; R_2p; R_2wp; CLw; CLp; $CLwp$; CRw; CRp; $CRwp$;
Pairs
L_0wp+R_0wp; L_0wp+R_0w; L_0w+R_0wp; L_0wp+R_0p; L_0p+R_0wp; L_0w+R_0w; L_0p+R_0p;R_0p+R_1p; L_0w+R_0wd; L_0p+R_0pd;
Triples
R_0p+R_1p+R_2p; L_0p+R_0p+R_1p; $L_{0h}p$+L_0p+R_0p; L_0p+$L_{0l'}p$+R_0p; L_0p+$L_{0r'}p$+R_0p; L_0p+R_0p+$R_{0l'}p$; L_0p+$L_{0l'}p$+$L_{0l}p$; L_0p+$L_{0r'}p$+$L_{0r}p$; L_0p+$L_{0h}p$+$L_{0h2}p$; R_0p+$R_{0l'}p$+$R_{0l}p$;

Table 1: Feature templates. L_0 and R_0 denote the left and right focus words; $L_1, L_2, \ldots$ are the words to the left of L_0 and $R_1, R_2, \ldots$ those to the right of R_0. X_{ih} means the head of X_i, X_{ih2} the grandparent, X_{il} and $X_{il'}$ the farthest and closest left dependents, and X_{ir} and $X_{ir'}$ the farthest and closest right dependents, respectively. CL and CR are the first and last words between L_0 and R_0 whose head is not in the interval $[L_0, R_0]$. Finally, w stands for word form; p for PoS tag; l for dependency label; d is the distance between L_0 and R_0; v_l, v_r are the left/right valencies (number of left/right dependents); and s_l, s_r the left/right label sets (dependency labels of left/right dependents).

sets from the CoNLL-XI shared task (Nivre et al., 2007). We use the same feature templates for all languages, which result from adapting the features described by Zhang and Nivre (2011) to the data structures of the Covington non-projective parser, and are listed in detail in Table 1.

Table 2 reports the accuracy obtained by the Covington non-projective parser with both oracles. As we can see, the dynamic oracle implemented in the Covington algorithm improves over the accuracy of the static version on all datasets except Japanese and Swedish, and most improvements are statistically significant at the .05 level.[3]

In addition, the Covington dynamic oracle achieves a greater average improvement in accuracy than the Attardi dynamic oracle (Gómez-Rodríguez et al., 2014) over their respective static versions. Concretely, the Attardi oracle accomplishes an average improvement of 0.52 percentage points in UAS and 0.71 in LAS, while our approach achieves 0.80 in UAS and 0.74 in LAS.

	s-Covington		d-Covington	
Language	UAS	LAS	UAS	LAS
Arabic	80.03	71.32	**81.47***	**72.77***
Basque	75.76	69.70	**76.49***	**70.27***
Catalan	88.66	83.92	**89.28**	**84.26**
Chinese	83.94	79.59	**84.68***	**80.16***
Czech	77.38	71.21	**78.58***	**72.59***
English	84.64	83.72	**86.14***	**84.96***
Greek	79.33	72.65	**80.52***	**73.67***
Hungarian	77.70	74.32	**78.22**	**74.61**
Italian	83.39	79.66	**83.66**	**79.91**
Turkish	82.14	76.00	**82.38**	**76.15**
Bulgarian	87.68	84.55	**88.48***	**85.32***
Danish	84.07	79.99	**84.98***	**80.85***
Dutch	80.28	77.55	**81.17***	**78.54***
German	86.12	83.93	**87.47***	**85.15***
Japanese	**93.92**	**92.51**	93.79	92.42
Portuguese	85.70	82.78	**86.23**	**83.27**
Slovene	75.31	68.97	**76.76***	**70.35***
Spanish	78.82	75.84	**79.87***	**76.97***
Swedish	**86.78**	**81.29**	86.66	81.21
Average	82.72	78.39	**83.52**	**79.13**

Table 2: Parsing accuracy (UAS and LAS, including punctuation) of Covington non-projective parser with static (s-Covington) and dynamic (d-Covington) oracles on CoNLL-XI (first block) and CoNLL-X (second block) datasets. For each language, we run five experiments with the same setup but different seeds and report the averaged accuracy. Best results for each language are shown in boldface. Statistically significant improvements ($\alpha = .05$) (Yeh, 2000) are marked with *.

5 Conclusion

We have defined the first dynamic oracle for a transition-based parser supporting unrestricted non-projectivity. The oracle is very efficient, computing loss in $O(n)$, compared to $O(n^8)$ for the only previously known dynamic oracle with support for a subset of non-projective trees (Gómez-Rodríguez et al., 2014).

Experiments on the treebanks from the CoNLL-X and CoNLL-XI shared tasks show that the dynamic oracle significantly improves accuracy on many languages over a static oracle baseline.

Acknowledgments

Research partially funded by the Spanish Ministry of Economy and Competitiveness/ERDF (grants FFI2014-51978-C2-1-R, FFI2014-51978-C2-2-R), Ministry of Education (FPU grant program) and Xunta de Galicia (grant R2014/034).

[3]Note that the loss of accuracy in Japanese and Swedish is not statistically significant.

References

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170, Morristown, NJ, USA. Association for Computational Linguistics.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia, Bulgaria.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102, New York, NY, USA. ACM.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India, December. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. 2014. A tabular method for dynamic oracles in transition-based parsing. *Transactions of the Association for Computational Linguistics*, 2:119–130.

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

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927. Association for Computational Linguistics.

John Hopcroft and Robert Endre Tarjan. 1973. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June.

Peter Neuhaus and Norbert Bröker. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL) and the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 337–343.

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

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pages 149–160. ACL/SIGPARSE.

Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553.

Robert Endre Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160.

Alexander Volokh and Günter Neumann. 2012. Dependency parsing with efficient feature extraction. In Birte Glimm and Antonio Krüger, editors, *KI*, volume 7526 of *Lecture Notes in Computer Science*, pages 253–256. Springer.

Alexander Volokh. 2013. *Performance-Oriented Dependency Parsing*. Doctoral dissertation, Saarland University, Saarbrücken, Germany.

Alexander Yeh. 2000. More accurate tests for the statistical significance of result differences. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 947–953.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, pages 188–193.