Head-driven Transition-based Parsing with Top-down Prediction

Katsuhiko Hayashi†, Taro Watanabe‡, Masayuki Asahara§, Yuji Matsumoto†
†Nara Institute of Science and Technology
Ikoma, Nara, 630-0192, Japan
‡National Institute of Information and Communications Technology
Sorakugun, Kyoto, 619-0289, Japan
§National Institute for Japanese Language and Linguistics
Tachikawa, Tokyo, 190-8561, Japan
katsuhiko-h@is.naist.jp, taro.watanabe@nict.go.jp
masayu-a@ninjal.ac.jp, matsu@is.naist.jp

Abstract

This paper presents a novel top-down head-driven parsing algorithm for data-driven projective dependency analysis. This algorithm handles global structures, such as clause and coordination, better than shift-reduce or other bottom-up algorithms. Experiments on the English Penn Treebank data and the Chinese CoNLL-06 data show that the proposed algorithm achieves comparable results with other data-driven dependency parsing algorithms.

1 Introduction

Transition-based parsing algorithms, such as shift-reduce algorithms (Nivre, 2004; Zhang and Clark, 2008), are widely used for dependency analysis because of the efficiency and comparatively good performance. However, these parsers have one major problem that they can handle only local information. Isozaki et al. (2004) pointed out that the drawbacks of shift-reduce parser could be resolved by incorporating top-down information such as root finding.

This work presents an $O(n^2)$ top-down head-driven transition-based parsing algorithm which can parse complex structures that are not trivial for shift-reduce parsers. The deductive system is very similar to Earley parsing (Earley, 1970). The Earley prediction is tied to a particular grammar rule, but the proposed algorithm is data-driven, following the current trends of dependency parsing (Nivre, 2006; McDonald and Pereira, 2006; Koo et al., 2010). To do the prediction without any grammar rules, we introduce a weighted prediction that is to predict lower nodes from higher nodes with a statistical model.

To improve parsing flexibility in deterministic parsing, our top-down parser uses beam search algorithm with dynamic programming (Huang and Sagae, 2010). The complexity becomes $O(n^2+b)$ where $b$ is the beam size. To reduce prediction errors, we propose a lookahead technique based on a FIRST function, inspired by the LL(1) parser (Aho and Ullman, 1972). Experimental results show that the proposed top-down parser achieves competitive results with other data-driven parsing algorithms.

2 Definition of Dependency Graph

A dependency graph is defined as follows.

Definition 2.1 (Dependency Graph) Given an input sentence $W = n_0 \ldots n_n$ where $n_0$ is a special root node $\$,$ a directed graph is defined as $G_W = (V_W, A_W)$ where $V_W = \{0, 1, \ldots, n\}$ is a set of (indices of) nodes and $A_W \subseteq V_W \times V_W$ is a set of directed arcs. The set of arcs is a set of pairs $(x, y)$ where $x$ is a head and $y$ is a dependent of $x$. $x \rightarrow^* l$ denotes a path from $x$ to $l$. A directed graph $G_W = (V_W, A_W)$ is well-formed if and only if:

- There is no node $x$ such that $(x, 0) \in A_W$.
- If $(x, y) \in A_W$ then there is no node $x'$ such that $(x', y) \in A_W$ and $x' \neq x$.
- There is no subset of arcs $\{(x_0, x_1), (x_1, x_2), \ldots, (x_{l-1}, x_l)\} \subseteq A_W$ such that $x_0 = x_l$.

These conditions are referred to ROOT, SINGLE-HEAD, and ACYCLICITY, and we call an well-formed directed graph as a dependency graph.

Definition 2.2 (Projectivity) A dependency graph $G_W = (V_W, A_W)$ is projective if and only if,
input: \( W = n_0 \ldots n_n \)

axiom\((p_0)\): \(0 : \langle 1, 0, n + 1, n_0 \rangle : \emptyset\)

\(\text{pred}_{\wedge}:\)

\[ \ell : \langle i, h, j, s_d|...|s_0 \rangle : \pi \]

\(\text{state } \ell + 1 : \langle i, h, j, s_d|...|s_0 \rangle : \{ p \} \)

\(\forall k : i \leq k < h \)

\(\text{pred}_{\vee}:\)

\[ \ell + 1 : \langle i, h, j, s_d|...|s_0 \rangle : \pi \]

\(\forall k : i \leq k < j \wedge h < i \)

\(\text{scan}:\)

\[ \ell + 1 : \langle i + 1, h, j, s_d|...|s_0 \rangle : \pi \]

\(i = h \)

\(\text{comp}:\)

\[ \ell + 1 : \langle i, h', j', s_d'|...|s_0'| \rangle : \pi' \]

\[ \ell : \langle i, h, j, s_d|...|s_0 \rangle : \pi \]

\(q \in \pi, h < i \)

\(\text{goal}:\)

\(3n : \langle n + 1, 0, n + 1, s_0 \rangle : \emptyset\)

---

Figure 1: The non-weighted deductive system of top-down dependency parsing algorithm: \(\_\) means “take anything”.

for every arc \((x, y) \in A_W\) and node \(l\) in \(x < l < y\) or \(y < l < x\), there is a path \(x \rightarrow^* l \) or \(y \rightarrow^* l\).

The proposed algorithm in this paper is for projective dependency graphs. If a projective dependency graph is connected, we call it a dependency tree, and if not, a dependency forest.

3 Top-down Parsing Algorithm

Our proposed algorithm is a transition-based algorithm, which uses stack and queue data structures. This algorithm formally uses the following state:

\[ \ell : \langle i, h, j, S \rangle : \pi \]

where \(\ell\) is a step size, \(S\) is a stack of trees \(s_d|...|s_0\) where \(s_0\) is a top tree and \(d\) is a window size for feature extraction, \(i\) is an index of node on the top of the input node queue, \(h\) is an index of root node of \(s_0\), \(j\) is an index to indicate the right limit (\(j - 1\) inclusive) of \(\text{pred}_{\wedge}\), and \(\pi\) is a set of pointers to predictor states, which are states just before putting the node in \(h\) onto stack \(S\). In the deterministic case, \(\pi\) is a singleton set except for the initial state.

This algorithm has four actions, \(\text{pred}_{\wedge}(\text{pred}_{\vee}), \text{pred}_{\wedge}(\text{pred}_{\vee}), \text{scan}\) and \(\text{complete}(\text{comp})\). The deductive system of the top-down algorithm is shown in Figure 1. The initial state \(p_0\) is a state initialized by an artificial root node \(n_0\). This algorithm applies one action to each state selected from applicable actions in each step. Each of three kinds of actions, \(\text{pred}, \text{scan}, \text{and } \text{comp}\), occurs \(n\) times, and this system takes \(3n\) steps for a complete analysis.

Action \(\text{pred}_{\wedge}\) puts \(n_0\) onto stack \(S\) selected from the input queue in the range, \(i \leq k < h\), which is to the left of the root \(n_h\) in the stack top. Similarly, action \(\text{pred}_{\vee}\) puts a node \(n_k\) onto stack \(S\) selected from the input queue in the range, \(h < i \leq k < j\), which is to the right of the root \(n_h\) in the stack top. The node \(n_i\) on the top of the queue is scanned if it is equal to the root node \(n_h\) in the stack top. Action \(\text{comp}\) creates a directed arc \((h', h)\) from the root \(h'\) of \(s_0\) to a predictor state \(q\) to the root \(h\) of \(s_0\) on a current state \(p\) if \(h < i\).

The precondition \(i < h\) of action \(\text{pred}_{\wedge}\) means that the input nodes in \(i \leq k < h\) have not been predicted yet. \(\text{pred}_{\wedge}\), \(\text{scan}\), and \(\text{pred}_{\vee}\) do not conflict with each other since their preconditions \(i < h\), \(i = h\) and \(h < i\) do not hold at the same time. However, this algorithm faces a \(\text{pred}_{\wedge} \wedge \text{comp}\) conflict because both actions share the same precondition \(h < i\), which means that the input nodes in \(1 \leq k \leq h\) have been predicted and scanned. This

\[\text{in a single root tree, the special root symbol } S_0 \text{ has exactly one child node. Therefore, we do not apply comp action to a state if its condition satisfies } s_1.h = n_0 \wedge \ell = 3n - 1.\]
parser constructs left and right children of a head node in a left-to-right direction by scanning the head node prior to its right children. Figure 2 shows an example for parsing a sentence “I saw a girl”.

4 Correctness

To prove the correctness of the system in Figure 1 for the projective dependency graph, we use the proof strategy of (Nivre, 2008a). The correct deductive system is both sound and complete.

Theorem 4.1 The deductive system in Figure 1 is correct for the class of dependency forest.

Proof 4.1 To show soundness, we show that $G_{p_0} = (V_W, \emptyset)$, which is a directed graph defined by the axiom, is well-formed and projective, and that every transition preserves this property.

- **ROOT**: The node 0 is a root in $G_{p_0}$, and the node 0 is on the top of stack of $p_0$. The two pred actions put a word onto the top of stack, and predict an arc from root or its descendant to the child. The comp actions add the predicted arcs which include no arc of $(x, 0)$.

- **SINGLE-HEAD**: $G_{p_0}$ is single-head. A node $y$ is no longer in stack and queue when the directed path $y \rightarrow^* x$ was made by adding an arc $(l, x)$. There is no chance to add the arc $(x, y)$ on the directed path $y \rightarrow^* x$.

- **PROJECTIVITY**: $G_{p_0}$ is projective. Projectivity is violated by adding an arc $(x, y)$ when there is a node l in $x < l < y$ or $y < l < x$ with the path to or from the outside of the span $x$ and $y$. When pred creates an arc relation from $x$ to $y$, the node $y$ cannot be scanned before all nodes $l$ in $x < l < y$ are scanned and completed. When pred creates an arc relation from $x$ to $y$, the node $y$ cannot be scanned before all nodes $k$ in $k < y$ are scanned and completed, and the node $x$ cannot be scanned before all nodes $l$ in $y < l < x$ are scanned and completed. In those processes, the node $l$ in $x < l < y$ or $y < l < x$ does not make a path to or from the outside of the span $x$ and $y$, and a path $x \rightarrow^* l$ or $y \rightarrow^* l$ is created. □

To show completeness, we show that for any sentence $W$, and dependency forest $G_W = (V_W, A_W)$, there is a transition sequence $C_{0,m}$, such that $G_{p_m} = G_W$ by an inductive method.

- If $|W| = 1$, the projective dependency graph for $W$ is $G_W = (\{0\}, \emptyset)$ and $G_{p_0} = G_W$.

- Assume that the claim holds for sentences with length less or equal to $t$, and assume that $|W| = t + 1$ and $G_W = (V_W, A_W)$. The subgraph $G_{W'}$ is defined as $(V_{W'} - t, A^{-1})$ where

\[
\begin{array}{c|c|c|c|c}
\text{step} & \text{state} & \text{stack} & \text{queue} & \text{action} & \text{state information} \\
0 & p_0 & \$0 & I_1 & \text{saw}_{2} & a_3 & \text{girl}_{4} & \rightarrow & (1, 0, 5) : \emptyset \\
1 & p_1 & \$0|saw_{2} & I_1 & \text{saw}_{2} & a_3 & \text{girl}_{4} & \text{pred}_{\rightarrow} & (1, 2, 5) : \{p_0\} \\
2 & p_2 & \text{saw}_{2}|I_1 & I_1 & \text{saw}_{2} & a_3 & \text{girl}_{4} & \text{pred}_{\rightarrow} & (1, 1, 2) : \{p_1\} \\
3 & p_3 & \text{saw}_{2}|I_1 & I_1 & \text{saw}_{2} & a_3 & \text{girl}_{4} & \text{scan} & (2, 1, 2) : \{p_1\} \\
4 & p_4 & \$0|I_1 \rightarrow \text{saw}_{2} & \text{saw}_{2} & a_3 & \text{girl}_{4} & \text{comp} & (2, 2, 5) : \{p_0\} \\
5 & p_5 & \$0|I_1 \rightarrow \text{saw}_{2} & a_3 & \text{girl}_{4} & \text{scan} & (3, 2, 5) : \{p_0\} \\
6 & p_6 & I_1 \rightarrow \text{saw}_{2}|\text{girl}_{4} & a_3 & \text{girl}_{4} & \text{pred}_{\rightarrow} & (3, 4, 5) : \{p_5\} \\
7 & p_7 & \text{girl}_{4}|a_3 & a_3 & \text{girl}_{4} & \text{pred}_{\rightarrow} & (3, 3, 4) : \{p_0\} \\
8 & p_8 & \text{girl}_{4}|a_3 & \text{girl}_{4} & \text{scan} & (4, 3, 4) : \{p_0\} \\
9 & p_9 & I_1 \rightarrow \text{saw}_{2}|a_3 \rightarrow \text{girl}_{4} & a_3 & \text{girl}_{4} & \text{comp} & (4, 4, 5) : \{p_5\} \\
10 & p_{10} & I_1 \rightarrow \text{saw}_{2}|a_3 \rightarrow \text{girl}_{4} & \text{girl}_{4} & \text{scan} & (5, 4, 5) : \{p_5\} \\
11 & p_{11} & \$0|I_1 \rightarrow \text{saw}_{2}|a_3 \rightarrow \text{girl}_{4} & a_3 & \text{girl}_{4} & \text{comp} & (5, 2, 5) : \{p_0\} \\
12 & p_{12} & \$0|saw_{2} & \text{detected system in Figure 1 is correct for the class of dependency forest.}

\begin{align*}
\text{Figure 2: Stages of the top-down deterministic parsing process for a sentence “I saw a girl”. We follow a convention and write the stack with its topmost element to the right, and the queue with its first element to the left. In this example, we set the window size } d \text{ to 1, and write the descendants of trees on stack elements } s_0 \text{ and } s_1 \text{ within depth 1.}
\end{align*}
Algorithm 1 Top-down Parsing with Beam Search

1: input $W = n_0, \ldots, n_n$
2: $start \leftarrow (1, 0, n+1, n_0)$
3: $buf[0] \leftarrow \{start\}$
4: for $\ell \leftarrow 1, \ldots, 3n$ do
5: $\hypo \leftarrow \{\}$
6: for each state in $buf[\ell - 1]$ do
7: $\forall act \leftarrow \text{applicableAct(state)}$ do
8: $\text{newstates} \leftarrow \text{actor}(act, state)$
9: addAll newstates to hypo
10: add top $b$ states to $buf[\ell]$ from hypo
11: return best candidate from $buf[3n]$
Figure 4: An example of tree structure: Each \( h, l \) and \( r \) denotes head, left and right child nodes.

\[ c_p(h, \text{sib}, c) = \theta_p \cdot f_p(h, \text{sib}, c). \]  

(4)

The 1st- and sibling 2nd-order models are the same as McDonald and Pereira (2006)'s definitions, except the cost factors of the sibling 2nd-order model. The cost factors for a tree structure in Figure 4 are defined as follows:

\[ c_p(h, -, l_1) + \sum_{y=1}^{l-1} c_p(h, l_y, l_y+1) \]

\[ + c_p(h, -, r_1) + \sum_{y=1}^{m-1} c_p(h, r_y, r_y+1). \]

This is different from McDonald and Pereira (2006) in that the cost factors for left children are calculated from left to right, while those in McDonald and Pereira (2006)'s definition are calculated from right to left. This is because our top-down parser generates left children from left to right. Note that the cost of weighted prediction model in this section is incrementally calculated by using only the information on the current state, thus the condition of state merge in Equation 2 remains unchanged.

5.3 Weighted Deductive System

We extend deductive system to a weighted one, and introduce forward cost and inside cost (Stolcke, 1995; Huang and Sagae, 2010). The forward cost is the total cost of a sequence from an initial state to the end state. The inside cost is the cost of a top tree \( s_0 \) in stack \( S \). We define these costs using a combination of stack-based model and weighted prediction model. The forward and inside costs of the combination model are as follows:

\[
\begin{align*}
\{ c_{fw}^s &= c_{fw}^s + c_{fw}^p \\
\{ c_{in}^s &= c_{in}^s + c_{in}^p 
\end{align*}
\]

(5)

where \( c_{fw}^s \) and \( c_{in}^s \) are a forward cost and an inside cost for stack-based model, and \( c_{fw}^p \) and \( c_{in}^p \) are a forward cost and an inside cost for weighted prediction model. We add the following tuple of costs to a state:

\[
(c_{fw}^s, c_{in}^s, c_{fw}^p, c_{in}^p).
\]

For each action, we define how to efficiently calculate the forward and inside costs\(^3\), following Stolcke (1995) and Huang and Sagae (2010)'s works. In either case of \( \text{pred}_\lor \) or \( \text{pred}_\land \),

\[
\begin{align*}
(c_{fw}^s, c_{in}^s, c_{fw}^p, c_{in}^p) \\
(c_{fw}^s + \lambda, 0, c_{fw}^p + c_p(s_0, h, n_k), 0)
\end{align*}
\]

where

\[
\lambda = \left\{ \begin{array}{ll}
\theta_s \cdot f_{s, \text{pred}_\lor}(i, h, j, S) & \text{if } \text{pred}_\lor \\
\theta_s \cdot f_{s, \text{pred}_\land}(i, h, j, S) & \text{if } \text{pred}_\land
\end{array} \right.
\]

(6)

In the case of scan,

\[ (c_{fw}^s + \xi, c_{in}^s + \xi, c_{fw}^p, c_{in}^p) \]

(7)

In the case of comp,

\[
\begin{align*}
(c_{fw}^s, c_{in}^s, c_{fw}^p, c_{in}^p) & \rightarrow (c_{fw}^s + \mu, c_{in}^s + \mu, \\
& c_{fw}^p + c_{in}^p + c_p(s_0, h, s_0, h))
\end{align*}
\]

where

\[
\mu = \theta_s \cdot f_{s, \text{comp}}(i, h, j, S) + \theta_s \cdot f_{s, \text{pred}}(\lambda, h', j', S').
\]

(8)

\(^3\) For brevity, we present the formula not by 2nd-order model as equation 4 but a 1st-order one for weighted prediction.
Pred\_ takes either pred\_ or pred\_. Beam search is performed based on the following linear order for the two states p and p′ at the same step, which have (c\_fw, c\_in) and (c\_fw, c\_in) respectively:

\[ p \succ p' \text{ iff } c\_fw < c\_fw \text{ or } c\_fw = c\_fw \land c\_in < c\_in. \] (9)

We prioritize the forward cost over the inside cost since forward cost pertains to longer action sequence and is better suited to evaluate hypothesis states than inside cost (Nederhof, 2003).

### 5.4 FIRST Function for Lookahead

Top-down backtrack parser usually reduces backtracking by precomputing the set FIRST(·) (Aho and Ullman, 1972). We define the set FIRST(·) for our top-down dependency parser:

\[
\text{FIRST}(t') = \{ \text{ld}\_t | \text{ld} \in \text{Im descendant(Tree, t')} \} \quad (10)
\]

where t′ is a POS-tag, Tree is a correct dependency tree which exists in Corpus, a function Im descendant(Tree, t′) returns the set of the leftmost descendant node ld of each nodes in Tree whose POS-tag is t′, and ld\_t denotes a POS-tag of ld. Though our parser does not backtrack, it looks ahead when selecting possible child nodes at the prediction step by using the function FIRST. In case of pred\_:

\[
\forall k : i \leq k < h \land n\_i \_t \in \text{FIRST}(n\_k \_t)
\]

\[
\ell : (i, h, j, s\_d ... s\_0) : \_ : \\
\ell + 1 : (i, k, h, s\_d ... s\_0 n\_k) : \{ p \}
\]

where n\_t is a POS-tag of the node n, on the top of the queue, and n\_k\_t is a POS-tag in kth position of an input nodes. The case for pred\_ is the same. If there are no nodes which satisfy the condition, our top-down parser creates new states for all nodes, and pushes them into hypo in line 9 of Algorithm 1.

### 6 Time Complexity

Our proposed top-down algorithm has three kinds of actions which are scan, comp and predict. Each scan and comp actions occurs n times when parsing a sentence with the length n. Predict action also occurs n times in which a child node is selected from a node sequence in the input queue. Thus, the algorithm takes the following times for prediction:

\[ n + (n - 1) + \cdots + 1 = \sum_{i}^{n} i = \frac{n(n + 1)}{2}. \] (11)

As n^2 for prediction is the most dominant factor, the time complexity of the algorithm is O(n^2) and that of the algorithm with beam search is O(n^2 + b).

### 7 Related Work

Alshawi (1996) proposed head automaton which recognizes an input sentence top-down. Eisner and Satta (1999) showed that there is a cubic-time parsing algorithm on the formalism of the head automaton grammars, which are equivalently converted into split-head bilexical context-free grammars (SBCFGs) (McAllester, 1999; Johnson, 2007). Although our proposed algorithm does not employ the formalism of SBCFGs, it creates left children before right children, implying that it does not have spurious ambiguities as well as parsing algorithms on the SBCFGs. Head-corner parsing algorithm (Kay, 1989) creates dependency tree top-down, and in this our algorithm has similar spirit to it.

Yamada and Matsumoto (2003) applied a shift-reduce algorithm to dependency analysis, which is known as arc-standard transition-based algorithm (Nivre, 2004). Nivre (2003) proposed another transition-based algorithm, known as arc-eager algorithm. The arc-eager algorithm processes right-dependent top-down, but this does not involve the prediction of lower nodes from higher nodes. Therefore, the arc-eager algorithm is a totally bottom-up algorithm. Zhang and Clark (2008) proposed a combination approach of the transition-based algorithm with graph-based algorithm (McDonald and Pereira, 2006), which is the same as our combination model of stack-based and prediction models.

### 8 Experiments

Experiments were performed on the English Penn Treebank data and the Chinese CoNLL-06 data. For the English data, we split WSJ part of it into sections 02-21 for training, section 22 for development and section 23 for testing. We used Yamada and Matsumoto (2003)’s head rules to convert phrase structure to dependency structure. For the Chinese data,
we used the information of words and fine-grained POS-tags for features. We also implemented and experimented Huang and Sagae (2010)'s arc-standard shift-reduce parser. For the 2nd-order Eisner-Satta algorithm, we used MSTParser (McDonald, 2012).

We used an early update version of averaged perceptron algorithm (Collins and Roark, 2004) for training of shift-reduce and top-down parsers. A set of feature templates in (Huang and Sagae, 2010) were used for the stack-based model, and a set of feature templates in (McDonald and Pereira, 2006) were used for the 2nd-order prediction model. The weighted prediction and stack-based models of top-down parser were jointly trained.

8.1 Results for English Data

During training, we fixed the prediction size and beam size to 5 and 16, respectively, judged by preliminary experiments on development data. After 25 iterations of perceptron training, we achieved 92.94 unlabeled accuracy for top-down parser with the FIRST function and 93.01 unlabeled accuracy for shift-reduce parser on development data by setting the beam size to 8 for both parsers and the prediction size to 5 in top-down parser. These trained models were used for the following testing.

We compared top-down parsing algorithm with other data-driven parsing algorithms in Table 1. Top-down parser achieved comparable unlabeled accuracy with others, and outperformed them on the sentence complete rate. On the other hand, top-down parser was less accurate than shift-reduce.

Table 1: Results for test data: Time measures the parsing time per sentence in seconds. Accuracy is an unlabeled attachment score, complete is a sentence complete rate, and root is a correct root rate. ∗ indicates our experiments.

|                      | time | accuracy | complete | root |
|----------------------|------|----------|----------|------|
| McDonald05,06 (2nd)  | 0.15 | 90.9, 91.5 | 37.5, 42.1 |      |
| Koo10 (Koo and Collins, 2010) | -   | 93.04    | -        | -    |
| Hayashi11 (Hayashi et al., 2011) | 0.3  | 92.89    | -        | -    |
| 2nd-MST∗              | 0.13 | 92.3     | 43.7     | 96.0 |
| Goldberg10 (Goldberg and Elhadad, 2010) | -   | 89.7     | 37.5     | 91.5 |
| Koo10 (Koo and Collins, 2010) | -   | 93.04    | -        | -    |
| Hayashi11 (Hayashi et al., 2011) | 0.3  | 92.89    | -        | -    |
| 2nd-MST∗              | 0.13 | 92.3     | 43.7     | 96.0 |
| Goldberg10 (Goldberg and Elhadad, 2010) | -   | 89.7     | 37.5     | 91.5 |
| Koo10 (Koo and Collins, 2010) | -   | 93.04    | -        | -    |
| Hayashi11 (Hayashi et al., 2011) | 0.3  | 92.89    | -        | -    |
| 2nd-MST∗              | 0.13 | 92.3     | 43.7     | 96.0 |

Table 2: Oracle score, choosing the highest accuracy parse for each sentence on test data from results of top-down (beam 8, pred 5) and shift-reduce (beam 8) and MST(2nd) parsers in Table 1.

|                      | accuracy | complete | root |
|----------------------|----------|----------|------|
| top-down (beam:8, pred:5) | 90.9     | 80.4     | 93.0 |
| shift-reduce (beam:8) | 90.8     | 77.6     | 93.5 |
| 2nd-MST               | 91.4     | 79.3     | 94.2 |
| oracle (sh+mst)       | 94.0     | 85.1     | 95.9 |
| oracle (top+sh)       | 93.8     | 84.0     | 95.6 |
| oracle (top+mst)      | 93.6     | 84.2     | 95.3 |
| oracle (top+sh+mst)   | 94.7     | 86.5     | 96.3 |

Table 3: Results for Chinese Data (CoNLL-06)

|                      | accuracy | complete | root |
|----------------------|----------|----------|------|
| top-down (beam:8, pred:5) | 90.9     | 80.4     | 93.0 |
| shift-reduce (beam:8) | 90.8     | 77.6     | 93.5 |
| 2nd-MST               | 91.4     | 79.3     | 94.2 |
| oracle (sh+mst)       | 94.0     | 85.1     | 95.9 |
| oracle (top+sh)       | 93.8     | 84.0     | 95.6 |
| oracle (top+mst)      | 93.6     | 84.2     | 95.3 |
| oracle (top+sh+mst)   | 94.7     | 86.5     | 96.3 |
Little Lily, as Ms. Cunningham calls herself in the book, really wasn’t ordinary.

Table 4: Two examples on which top-down parser is superior to two bottom-up parsers: In correct analysis, the boxed portion is the head of the underlined portion. Bottom-up parsers often mistake to capture the relation.

8.3 Analysis of Results

Table 4 shows two interesting results, on which top-down parser is superior to either shift-reduce parser or 2nd-MST parser. The sentence No.717 contains an adverbial clause structure between the subject and the main verb. Top-down parser is able to handle the long-distance dependency while shift-reduce parser cannot correctly analyze it. The effectiveness on the clause structures implies that our head-driven parser may handle non-projective structures well, which are introduced by Johansson’s head rule (Johansson and Nuges, 2007). The sentence No.127 contains a coordination structure, which it is difficult for bottom-up parsers to handle, but, top-down parser handles it well because its top-down prediction globally captures the coordination.

9 Conclusion

This paper presents a novel head-driven parsing algorithm and empirically shows that it is as practical as other dependency parsing algorithms. Our head-driven parser has potential for handling non-projective structures better than other non-projective dependency algorithms (McDonald et al., 2005; Attardi, 2006; Nivre, 2008b; Koo et al., 2010). We are in the process of extending our head-driven parser for non-projective structures as our future work.

Acknowledgments

We would like to thank Kevin Duh for his helpful comments and to the anonymous reviewers for giving valuable comments.
References

A. V. Aho and J. D. Ullman. 1972. The Theory of Parsing, Translation and Compiling, volume 1: Parsing. Prentice-Hall.

H. Alshawi. 1996. Head automata for speech translation. In Proc. the ICSLP.

G. Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In Proc. the 10th CoNLL, pages 166–170.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In Proc. the 42nd ACL.

J. Earley. 1970. An efficient context-free parsing algorithm. Communications of the Association for Computing Machinery, 13(2):94–102.

J. M. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In Proc. the 37th ACL, pages 457–464.

Y. Goldberg and M. Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In Proc. the HLT-NAACL, pages 742–750.

K. Hayashi, T. Watanabe, M. Asahara, and Y. Matsumoto. 2011. The third-order variational reranking on packed-shared dependency forests. In Proc. EMNLP, pages 1479–1488.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In Proc. the 48th ACL, pages 1077–1086.

H. Isozaki, H. Kazawa, and T. Hirao. 2004. A deterministic word dependency analyzer enhanced with preference learning. In Proc. the 21st COLING, pages 275–281.

R. Johansson and P. Nugues. 2007. Extended constituent-to-dependency conversion for english. In Proc. NODALIDA.

M. Johnson. 2007. Transforming projective bilexical dependency grammars into efficiently-parsable CFGs with unfold-fold. In Proc. the 45th ACL, pages 168–175.

M. Kay. 1989. Head driven parsing. In Proc. the IWPT.

K. Kitagawa and K. Tanaka-Ishii. 2010. Tree-based deterministic dependency parsing — an application to nivre’s method —. In Proc. the 48th ACL 2010 Short Papers, pages 189–193, July.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In Proc. the 48th ACL, pages 1–11.

T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In Proc. EMNLP, pages 1288–1298.

D. McAllester. 1999. A reformulation of eisner and satta’s cubic time parser for split head automata grammars. http://ttic.uchicago.edu/dmcallester/.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In Proc. EACL, pages 81–88.

R. McDonald, F. Pereira, K. Ribaric, and J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In Proc. HLT-EMNLP, pages 523–530.

R. McDonald. 2012. Minimum spanning tree parser. http://www.seas.upenn.edu/strctlrn/MSTParser.

M.-J. Nederhof. 2003. Weighted deductive parsing and knuth’s algorithm. Computational Linguistics, 29:135–143.

J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In Proc. the IWPT, pages 149–160.

J. Nivre. 2004. Incrementality in deterministic dependency parsing. In Proc. the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together, pages 50–57.

J. Nivre. 2006. Inductive Dependency Parsing. Springer.

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

J. Nivre. 2008b. Sorting out dependency parsing. In Proc. the CoTAL, pages 16–27.

A. Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Computational Linguistics, 21(2):165–201.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In Proc. the IWPT, pages 195–206.

Y. Zhang and S. Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In Proc. EMNLP, pages 562–571.

Y. Zhang and J. Nivre. 2011. Transition-based dependency parsing with rich non-local features. In Proc. the 49th ACL, pages 188–193.