Tree-Based Deterministic Dependency Parsing — An Application to Nivre's Method —

Kotaro Kitagawa **Kumiko Tanaka-Ishii**
Graduate School of Information Science and Technology,
The University of Tokyo
kitagawa@cl.ci.i.u-tokyo.ac.jp kumiko@i.u-tokyo.ac.jp

Abstract

Nivre's method was improved by enhancing deterministic dependency parsing through application of a tree-based model. The model considers all words necessary for selection of parsing actions by including words in the form of trees. It chooses the most probable head candidate from among the trees and uses this candidate to select a parsing action.

In an evaluation experiment using the Penn Treebank (WSJ section), the proposed model achieved higher accuracy than did previous deterministic models. Although the proposed model's worst-case time complexity is $O(n^2)$, the experimental results demonstrated an average parsing time not much slower than $O(n)$.

1 Introduction

Deterministic parsing methods achieve both effective time complexity and accuracy not far from those of the most accurate methods. One such deterministic method is Nivre's method, an incremental parsing method whose time complexity is linear in the number of words (Nivre, 2003). Still, deterministic methods can be improved. As a specific example, Nivre's model greedily decides the parsing action only from two words and their locally relational words, which can lead to errors.

In the field of Japanese dependency parsing, Iwatate et al. (2008) proposed a tournament model that takes all head candidates into account in judging dependency relations. This method assumes backward parsing because the Japanese dependency structure has a head-final constraint, so that any word's head is located to its right.

Here, we propose a tree-based model, applicable to any projective language, which can be considered as a kind of generalization of Iwatate's idea. Instead of selecting a parsing action for two words, as in Nivre's model, our tree-based model first chooses the most probable head candidate from among the trees through a tournament and then decides the parsing action between two trees.

Global-optimization parsing methods are another common approach (Eisner, 1996; McDonald et al., 2005). Koo et al. (2008) studied semi-supervised learning with this approach. Hybrid systems have improved parsing by integrating outputs obtained from different parsing models (Zhang and Clark, 2008).

Our proposal can be situated among global-optimization parsing methods as follows. The proposed tree-based model is deterministic but takes a step towards global optimization by widening the search space to include all necessary words connected by previously judged head-dependent relations, thus achieving a higher accuracy yet largely retaining the speed of deterministic parsing.

2 Deterministic Dependency Parsing

2.1 Dependency Parsing

A dependency parser receives an input sentence $x = w_1, w_2, \dots, w_n$ and computes a dependency graph $G = (W, A)$. The set of nodes $W = \{w_0, w_1, \dots, w_n\}$ corresponds to the words of a sentence, and the node w_0 is the root of G. A is the set of arcs (w_i, w_j), each of which represents a dependency relation where w_i is the *head* and w_j is the *dependent*.

In this paper, we assume that the resulting dependency graph for a sentence is well-formed and projective (Nivre, 2008). G is well-formed if and only if it satisfies the following three conditions of being **single-headed**, **acyclic**, and **rooted**.

2.2 Nivre's Method

An incremental dependency parsing algorithm was first proposed by (Covington, 2001). After

Table 1: Transitions for Nivre's method and the proposed method.

		Transition	**Precondition**
Nivre's Method	Left-Arc	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_j\|\beta, A \cup \{(w_j, w_i)\})$	$i \neq 0 \wedge \neg\exists w_k\ (w_k, w_i) \in A$
	Right-Arc	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma\|w_i\|w_j, \beta, A \cup \{(w_i, w_j)\})$	
	Reduce	$(\sigma\|w_i, \beta, A) \Rightarrow (\sigma, \beta, A)$	$\exists w_k\ (w_k, w_i) \in A$
	Shift	$(\sigma, w_j\|\beta, A) \Rightarrow (\sigma\|w_j, \beta, A)$	
Proposed Method	Left-Arc	$(\sigma\|t_i, t_j\|\beta, A) \Rightarrow (\sigma, t_j\|\beta, A \cup \{(w_j, w_i)\})$	$i \neq 0$
	Right-Arc	$(\sigma\|t_i, t_j\|\beta, A) \Rightarrow (\sigma\|t_i, \beta, A \cup \{(mphc(t_i, t_j), w_j)\})$	
	Shift	$(\sigma, t_j\|\beta, A) \Rightarrow (\sigma\|t_j, \beta, A)$	

studies taking data-driven approaches, by (Kudo and Matsumoto, 2002), (Yamada and Matsumoto, 2003), and (Nivre, 2003), the deterministic incremental parser was generalized to a state transition system in (Nivre, 2008).

Nivre's method applying an arc-eager algorithm works by using a stack of words denoted as σ, for a buffer β initially containing the sentence x. Parsing is formulated as a quadruple (S, T_s, s_{init}, S_t), where each component is defined as follows:

- S is a set of states, each of which is denoted as $(\sigma, \beta, A) \in S$.
- T_s is a set of transitions, and each element of T_s is a function $t_s : S \rightarrow S$.
- $s_{init} = ([w_0], [w_1, \ldots, w_n], \phi)$ is the initial state.
- S_t is a set of terminal states.

Syntactic analysis generates a sequence of optimal transitions t_s provided by an oracle $o : S \rightarrow T_s$, applied to a target consisting of the stack's top element w_i and the first element w_j in the buffer. The oracle is constructed as a classifier trained on treebank data. Each transition is defined in the upper block of Table 1 and explained as follows:

Left-Arc Make w_j the head of w_i and pop w_i, where w_i is located at the stack top (denoted as $\sigma|w_i$), when the buffer head is w_j (denoted as $w_j|\beta$).

Right-Arc Make w_i the head of w_j, and push w_j.

Reduce Pop w_i, located at the stack top.

Shift Push the word w_j, located at the buffer head, onto the stack top.

The method explained thus far has the following drawbacks.

Locality of Parsing Action Selection

The dependency relations are greedily determined, so when the transition Right-Arc adds a dependency arc (w_i, w_j), a more probable head of w_j located in the stack is disregarded as a candidate.

Features Used for Selecting Reduce

The features used in (Nivre and Scholz, 2004) to define a state transition are basically obtained from the two target words w_i and w_j, and their related words. These words are not sufficient to select Reduce, because this action means that w_j has no dependency relation with any word in the stack.

Preconditions

When the classifier selects a transition, the resulting graph satisfies well-formedness and projectivity only under the preconditions listed in Table 1. Even though the parsing seems to be formulated as a four-class classifier problem, it is in fact formed of two types of three-class classifiers.

Solving these problems and selecting a more suitable dependency relation requires a parser that considers more global dependency relations.

3 Tree-Based Parsing Applied to Nivre's Method

3.1 Overall Procedure

Tree-based parsing uses trees as the procedural elements instead of words. This allows enhancement of previously proposed deterministic models such as (Covington, 2001; Yamada and Matsumoto, 2003). In this paper, we show the application of tree-based parsing to Nivre's method. The parser is formulated as a state transition system (S, T_s, s_{init}, S_t), similarly to Nivre's parser, but σ and β for a state $s = (\sigma, \beta, A) \in S$ denote a stack of trees and a buffer of trees, respectively. A tree $t_i \in T$ is defined as the tree rooted by the word w_i, and the initial state is $s_{init} = ([t_0], [t_1, \ldots, t_n], \phi)$, which is formed from the input sentence x.

The state transitions T_s are decided through the following two steps.

1. **Select the most probable head candidate (MPHC)**: For the tree t_i located at the stack top, search for and select the MPHC for w_j, which is the root word of t_j located at the buffer head. This procedure is denoted as a

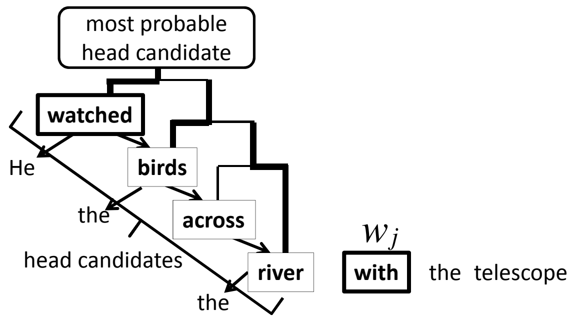

Figure 1: Example of a tournament.

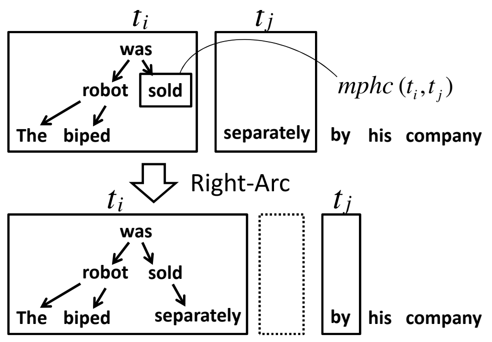

Figure 2: Example of the transition Right.

function $mphc(t_i, t_j)$, and its details are explained in §3.2.

2. **Select a transition:** Choose a transition, by using an oracle, from among the following three possibilities (explained in detail in §3.3):

Left-Arc Make w_j the head of w_i and pop t_i, where t_i is at the stack top (denoted as $\sigma|t_i$, with the tail being σ), when the buffer head is t_j (denoted as $t_j|\beta$).

Right-Arc Make the MPHC the head of w_j, and pop the MPHC.

Shift Push the tree t_j located at the buffer head onto the stack top.

These transitions correspond to three possibilities for the relation between t_i and t_j: (1) a word of t_i is a dependent of a word of t_j; (2) a word of t_j is a dependent of a word of t_i; or (3) the two trees are not related.

The formulations of these transitions in the lower block of Table 1 correspond to Nivre's transitions of the same name, except that here a transition is applied to a tree. This enhancement from words to trees allows removal of both the Reduce transition and certain preconditions.

3.2 Selection of Most Probable Head Candidate

By using $mphc(t_i, t_j)$, a word located far from w_j (the head of t_j) can be selected as the head candidate in t_i. This selection process decreases the number of errors resulting from greedy decision considering only a few candidates.

Various procedures can be considered for implementing $mphc(t_i, t_j)$. One way is to apply the tournament procedure to the words in t_i. The tournament procedure was originally introduced for parsing methods in Japanese by (Iwatate et al., 2008). Since the Japanese language has the head-final property, the tournament model itself constitutes parsing, whereas for parsing a general projective language, the tournament model can only be used as part of a parsing algorithm.

Figure 1 shows a tournament for the example of "with," where the word "watched" finally wins. Although only the words on the left-hand side of tree t_j are searched, this does not mean that the tree-based method considers only one side of a dependency relation. For example, when we apply the tree-based parsing to Yamada's method, the search problems on both sides are solved.

To implement $mphc(t_i, t_j)$, a binary classifier is built to judge which of two given words is more appropriate as the head for another input word. This classifier concerns three words, namely, the two words l (left) and r (right) in t_i, whose appropriateness as the head is compared for the dependent w_j. All word pairs of l and r in t_i are compared repeatedly in a "tournament," and the survivor is regarded as the MPHC of w_j.

The classifier is generated through learning of training examples for all t_i and w_j pairs, each of which generates examples comparing the true head and other (inappropriate) heads in t_i. Table 2 lists the features used in the classifier. Here, lex(X) and pos(X) mean the surface form and part of speech of X, respectively. X^{left} means the dependents of X located on the left-hand side of X, while X^{right} means those on the right. Also, X^{head} means the head of X. The feature design concerns three additional words occurring after w_j, as well, denoted as $w_{j+1}, w_{j+2}, w_{j+3}$.

3.3 Transition Selection

A transition is selected by a three-class classifier after deciding the MPHC, as explained in §3.1. Table 1 lists the three transitions and one precon-

Table 2: Features used for a tournament.

pos(l), lex(l)
pos(l^{head}), pos(l^{left}), pos(l^{right})

pos(r), lex(r)
pos(r^{head}), pos(r^{left}), pos(r^{right})

pos(w_j), lex(w_j), pos(w_j^{left})

pos(w_{j+1}), lex(w_{j+1}), pos(w_{j+2}), lex(w_{j+2})
pos(w_{j+3}), lex(w_{j+3})

Table 3: Features used for a state transition.

pos(w_i), lex(w_i)
pos(w_i^{left}), pos(w_i^{right}), lex(w_i^{left}), lex(w_i^{right})

pos(MPHC), lex(MPHC)
pos(MPHC^{head}), pos(MPHC^{left}), pos(MPHC^{right})
lex(MPHC^{head}), lex(MPHC^{left}), lex(MPHC^{right})

pos(w_j), lex(w_j), pos(w_j^{left}), lex(w_j^{left})
pos(w_{j+1}), lex(w_{j+1}), pos(w_{j+2}), lex(w_{j+2}), pos(w_{j+3}), lex(w_{j+3})

dition. The transition Shift indicates that the target trees t_i and t_j have no dependency relations. The transition Right-Arc indicates generation of the dependent-head relation between w_j and the result of $mphc(t_i, t_j)$, i.e., the MPHC for w_j. Figure 2 shows an example of this transition. The transition Left-Arc indicates generation of the dependency relation in which w_j is the head of w_i. While Right-Arc requires searching for the MPHC in t_i, this is not the case for Left-Arc[1].

The key to obtaining an accurate tree-based parsing model is to extend the search space while at the same time providing ways to narrow down the space and find important information, such as the MPHC, for proper judgment of transitions.

The three-class classifier is constructed as follows. The dependency relation between the target trees is represented by the three words w_i, MPHC, and w_j. Therefore, the features are designed to incorporate these words, their relational words, and the three words next to w_j. Table 3 lists the exact set of features used in this work. Since this transition selection procedure presumes selection of the MPHC, the result of $mphc(t_i, t_j)$ is also incorporated among the features.

4 Evaluation

4.1 Data and Experimental Setting

In our experimental evaluation, we used Yamada's head rule to extract unlabeled dependencies from the Wall Street Journal section of a Penn Treebank. Sections 2-21 were used as the training data, and section 23 was used as the test data. This test data was used in several other previous works, enabling mutual comparison with the methods reported in those works.

The SVM^{light} package[2] was used to build the support vector machine classifiers. The binary classifier for MPHC selection and the three-class classifier for transition selection were built using a cubic polynomial kernel. The parsing speed was evaluated on a Core2Duo (2.53 GHz) machine.

4.2 Parsing Accuracy

We measured the ratio of words assigned correct heads to all words (accuracy), and the ratio of sentences with completely correct dependency graphs to all sentences (complete match). In the evaluation, we consistently excluded punctuation marks.

Table 4 compares our results for the proposed method with those reported in some previous works using equivalent training and test data. The first column lists the four previous methods and our method, while the second through fourth columns list the accuracy, complete match accuracy, and time complexity, respectively, for each method. Here, we obtained the scores for the previous works from the corresponding articles listed in the first column. Note that every method used different features, which depend on the method.

The proposed method achieved higher accuracy than did the previous deterministic models. Although the accuracy of our method did not reach that of (McDonald and Pereira, 2006), the scores were competitive even though our method is deterministic. These results show the capability of the tree-based approach in effectively extending the search space.

4.3 Parsing Time

Such extension of the search space also concerns the speed of the method. Here, we compare its computational time with that of Nivre's method. We re-implemented Nivre's method to use SVMs with cubic polynomial kernel, similarly to our

[1]The head word of w_i can only be w_j without searching within t_j, because the relations between the other words in t_j and w_i have already been inferred from the decisions made within previous transitions. If t_j has a child w_k that could become the head of w_i under projectivity, this w_k must be located between w_i and w_j. The fact that w_k's head is w_j means that there were two phases before t_i and t_j (i.e., w_i and w_j) became the target:

- t_i and t_k became the target, and Shift was selected.
- t_k and t_j became the target, and Left-Arc was selected.

The first phase precisely indicates that w_i and w_k are unrelated.

[2]http://svmlight.joachims.org/

Table 4: Dependency parsing performance.

	Accuracy	Complete match	Time complexity	Global vs. deterministic	Learning method
McDonald & Pereira (2006)	91.5	42.1	$O(n^3)$	global	MIRA
McDonald et al. (2005)	90.9	37.5	$O(n^3)$	global	MIRA
Yamada & Matsumoto (2003)	90.4	38.4	$O(n^2)$	deterministic	support vector machine
Goldberg & Elhadad (2010)	89.7	37.5	$O(n \log n)$	deterministic	structured perceptron
Nivre (2004)	87.1	30.4	$O(n)$	deterministic	memory based learning
Proposed method	91.3	41.7	$O(n^2)$	deterministic	support vector machine

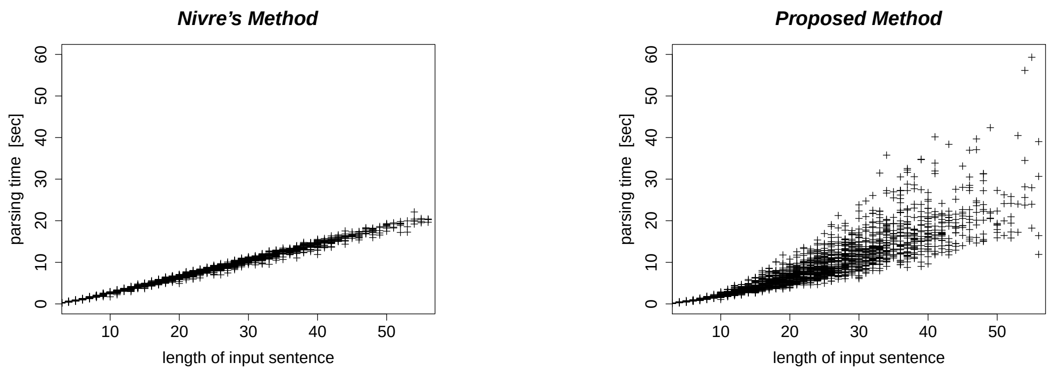

Figure 3: Parsing time for sentences.

method. Figure 3 shows plots of the parsing times for all sentences in the test data. The average parsing time for our method was 8.9 sec, whereas that for Nivre's method was 7.9 sec.

Although the worst-case time complexity for Nivre's method is $O(n)$ and that for our method is $O(n^2)$, worst-case situations (e.g., all words having heads on their left) did not appear frequently. This can be seen from the sparse appearance of the upper bound in the second figure.

5 Conclusion

We have proposed a tree-based model that decides head-dependency relations between trees instead of between words. This extends the search space to obtain the best head for a word within a deterministic model. The tree-based idea is potentially applicable to various previous parsing methods; in this paper, we have applied it to enhance Nivre's method.

Our tree-based model outperformed various deterministic parsing methods reported previously. Although the worst-case time complexity of our method is $O(n^2)$, the average parsing time is not much slower than $O(n)$.

References

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parse. *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pp. 957-961.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. *Proceedings of ACM*, pp. 95-102.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. *Proceedings of COLING*, pp. 340-345.

Yoav Goldberg and Michael Elhadad. 2010. An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. *Proceedings of NAACL*.

Masakazu Iwatate, Masayuki Asahara, and Yuji Matsumoto. 2008. Japanese dependency parsing using a tournament model. *Proceedings of COLING*, pp. 361–368.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. *Proceedings of ACL*, pp. 595–603.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking *Proceedings of CoNLL*, pp. 63–69.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. *Proceedings of ACL*, pp. 91–98.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. *Proceedings of the EACL*, pp. 81–88.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. *Proceedings of IWPT*, pp. 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, vol. 34, num. 4, pp. 513–553.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. *Proceedings of COLING*, pp. 64–70.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. *Proceedings of IWPT*, pp. 195–206.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beamsearch. *Proceedings of EMNLP*, pp. 562–571.