

## PAPER

# Dependency Chart Parsing Algorithm Based on Ternary-Span Combination

Meixun JIN<sup>†a)</sup>, Yong-Hun LEE<sup>††b)</sup>, Nonmembers, and Jong-Hyeok LEE<sup>†c)</sup>, Member

**SUMMARY** This paper presents a new span-based dependency chart parsing algorithm that models the relations between the left and right dependents of a head. Such relations cannot be modeled in existing span-based algorithms, despite their popularity in dependency corpora. We address this problem through ternary-span combination during the subtree derivation. By modeling the relations between the left and right dependents of a head, our proposed algorithm provides a better capability of coordination disambiguation when the conjunction is annotated as the head of the left and right conjuncts. This eventually leads to state-of-the-art performance of dependency parsing on the Chinese data of the CoNLL shared task.

**key words:** *dependency chart parsing, span-based parsing, factor-based parsing, dependency parsing algorithm, averaged perceptron, syntactic analysis*

## 1. Introduction

Chart parsing is frequently used to derive the syntactic structure of a given sentence [1]–[3], and various chart parsing algorithms are available for dependency parse tree derivation (Sect. 3). Among them, Eisner’s algorithm [4], [5] is notable for its efficiency, with a time complexity of  $O(n^3)$ . In [2], Eisner’s algorithm has been successfully augmented with current learning methods in the frame of graph-based data-driven dependency parsing. Eisner’s algorithm was further extended in the works of [6], [7] and [3], and all of these parsers reported a state-of-the-art performance on the English dependency treebank that is converted from the English Penn Treebank [8]. We refer to these algorithms as *span-based* algorithms because they all perform parsing by composing *spans* (details in Sect. 2).

Our algorithm is another extension of Eisner’s algorithm, and aims to model the relations between the *left* and *right* dependents of a head. It is quite common that a head simultaneously dominates the left and right dependents, e.g., the case in which a predicate verb takes subject and object as its left and right dependents in a *Subject-Verb-Object* (SVO) language, or the case of the *coordinate structures* in which the conjunction is annotated as the head of the left- and right-side conjuncts. Despite their popularity and the

importance of modeling such relations in dependency parsing, they have not yet been modeled in existing *span-based* dependency parsing algorithms.

In the proposed parsing algorithm, we derive subtrees by augmenting *three* spans, and the relations between the left and right dependents are modeled during the span-augmentation (Sect. 2.2). Experimental results show that the proposed algorithm provides a better coordination disambiguation when the conjunctions are annotated as heads, and that this improved coordination disambiguation eventually improves the overall parsing performance (Sect. 5).

This paper is organized as follows. We describe the spans and present our algorithm in Sect. 2. In Sect. 3, we compare our algorithm with other dependency chart parsing algorithms. In Sect. 4, we briefly discuss various dependency annotation schemes and their corresponding parsing algorithms. Section 5 presents our experimental results and finally, Sect. 6 provides some concluding remarks regarding our propose.

## 2. Proposed Algorithm

As normal CYK algorithms [9], our algorithm derives a dependency parse tree by recursively combining smaller subtrees in a bottom-up manner. We use a dynamic programming table, known as a *chart*, to store the partial results for each parsing step.

There are two types of dependency chart parsing algorithms, i.e., *constituent-based* and *span-based* algorithms, depending on the type of subtrees that the algorithm processes. If the algorithm constructs subtrees of spans, it is a *span-based* algorithm, otherwise, it is a *constituent-based* algorithm. Ours is an improved version of Eisner’s span-based algorithm [4].

### 2.1 Span

A *span* is a dependency *subtree* or a sequence of words linked by dependency arcs dominated\* by the *leftmost* or *rightmost* word of the sequence. For example,  $\text{STRING}_{[s;q]}$  shown in Fig. 1 (a) is a *span* dominated by the leftmost word  $s$ , which we represent using a right triangle as shown in Fig. 1 (d). The vertical side of the triangle indicates the position of the head. If an internal word  $r$  dominates the

\*We say a word *dominates* a string, if it is possible for the word to reach any other words in the string by following the paths of dependency arcs.

Manuscript received March 6, 2012.

Manuscript revised August 19, 2012.

<sup>†</sup>The authors are with the Division of Electrical and Computer Engineering, Pohang University of Science and Technology (POSTECH), Korea.

<sup>††</sup>The author is with NHN Co. Ltd., Korea.

a) E-mail: meixunj@postech.ac.kr

b) E-mail: yhlee95@postech.ac.kr

c) E-mail: jhlee@postech.ac.kr

DOI: 10.1587/transinf.E96.D.93

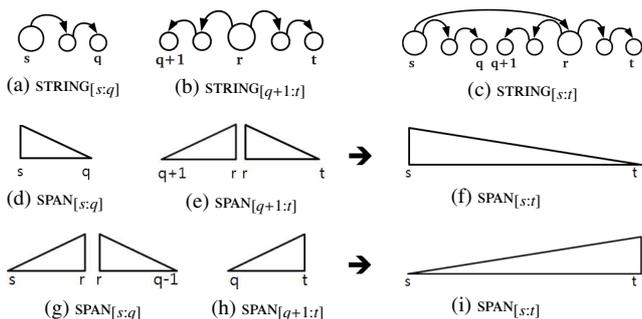


Fig. 1 Parse step represented by ternary-span combination.

string, as shown in Fig. 1 (b),  $\text{STRING}_{[q+1:t]}$  is represented by *two* spans, as shown in Fig. 1 (e).

Adding a dummy ‘ROOT’ in front of the sentence and allowing the ‘ROOT’ to dominate the sentence, the parse tree of this extended sentence becomes a *span*. This span is the ultimate one that a span-based parser aims to derive.

## 2.2 Ternary-Span Combination

A *span*-based dependency parser derives parse trees by constructing *spans* in a bottom-up fashion. A larger *span* is constructed by combining smaller spans. In this subsection, we show how our algorithm combines smaller *spans* into a larger span.

By adding a new dependency arc to connect  $s$  and  $r$ , we can combine  $\text{STRING}_{[s:q]}$  (Fig. 1 (a)) and  $\text{STRING}_{[q+1:t]}$  (Fig. 1 (b)) into  $\text{STRING}_{[s:t]}$  (Fig. 1 (c)).  $\text{STRING}_{[s:q]}$  corresponds to  $\text{SPAN}_{[s:q]}$  in Fig. 1 (d), while  $\text{STRING}_{[q+1:t]}$  corresponds to two spans, i.e.,  $\text{SPAN}_{[q+1:r]}$  and  $\text{SPAN}_{[r:t]}$  shown in Fig. 1 (e). Thus, this derivation of  $\text{STRING}_{[s:t]}$  can be represented by combining three spans into one, as shown in Figs. 1 (d)–(f). We call it *ternary-span combination*. In a similar way, the *right*-side head span construction is represented as shown in Figs. 1 (g)–(i).

The pseudocode for *ternary-span combination* is given in Fig. 2 and explained in Sect. 2.3.

Figures 1 (a)–(c) show one derivation of  $\text{STRING}_{[s:t]}$ . To derive the parse tree of a string we must select a pair of substrings that lead to an optimal parse tree. For  $\text{STRING}_{[s:t]}$  in Fig. 1 (c), we may select the optimal pair of strings, i.e.,  $\text{STRING}_{[s:q]}$  and  $\text{STRING}_{[q+1:t]}$ , by enumerating all possible values of  $q$  and  $r$  (Lines 11 and 16 of the pseudocode in Fig. 3). Here,  $q$  is the boundary of the two strings concatenated into  $\text{STRING}_{[s:t]}$ , and  $r$  is the newly selected dependent of  $s$  during the parse step.

## 2.3 The Algorithm

As a normal chart parser, we store the *max score* of a span and its related *parameters*, i.e.,  $q$  and  $r$ , for each parse step in the corresponding cells of the *chart*. For convenience, we use  $K_{[s:t][d]}$ ,  $B_{[s:t][d]}$  and  $D_{[s:t][d]}$  to represent the items in the chart.  $K_{[s:t][d]}$  represents the *max score* of the span that yields over  $\text{STRING}_{[s:t][d]}$ , beginning at  $s$  and ending at  $t$ , with

$s \leq t$ . The variable  $d$  indicates the side where the head is located:  $d = 'L'$ , when the *left* side is the head, or  $d = 'R'$ , when the *right* side is the head. We use  $B_{[s:t][d]}$  and  $D_{[s:t][d]}$  to represent the indices of two parameters  $q$  and  $r$  that give the maximum score  $K_{[s:t][d]}$ .

The pseudocodes used to calculate  $K_{[s:t][d]}$ ,  $B_{[s:t][d]}$ , and  $D_{[s:t][d]}$  are given in Fig. 3. Line 11 of Fig. 3 shows that, to find the maximum score  $K_{[s:t][L]}$  of  $\text{SPAN}_{[s:t][L]}$ , we need to find the indices  $q$  and  $r$  that lead to the maximum score through  $\text{TERNARYCOMBINATION}$  (the pseudocode in Fig. 2). After enumerating all possible values of  $q$  and  $r$ , Line 11 selects the *ternary-span combination* that gives the highest score. The maximum score is then assigned to  $K_{[s:t][L]}$  (Line 12), and the  $q$  and  $r$  ( $Q, R$ , Line 11) that give the maximum score are assigned to  $B_{[s:t][L]}$  and  $D_{[s:t][L]}$ , respectively (Lines 13 and 14). In a similar way for the *right*-side head  $\text{SPAN}_{[s:t][R]}$ , the related derivations for  $K_{[s:t][R]}$ ,  $B_{[s:t][R]}$ , and  $D_{[s:t][R]}$  are given in Lines 16–19.

The module  $\text{TERNARYCOMBINATION}$  (Fig. 2) calculates the score of the span generated by combining *three* smaller spans. Deriving the boundary indices of the *three* spans ( $s:q$ ,  $q+1:r$ , and  $r:t$ ), from its input parameters (Line 1 of Fig. 2), the *score* of the newly generated span is the accumulated scores of the *three* spans plus the score of the new dependency arc linking them together. In Line 3,  $K_{[s:q][L]}$ ,  $K_{[q+1:r][R]}$ , and  $K_{[r:t][L]}$  are the scores of the three spans combined into  $\text{SPAN}_{[s:t][L]}$ , and the  $\text{dep}(\ast)$  function evaluates the dependency arc linking them. Similarly, Line 5 calculates the score for the *right*-side head  $\text{SPAN}_{[s:t][R]}$ .

The function  $\text{dep}(\ast)$  evaluates the new dependency arc. In addition to the indices of the target head  $s$  and dependent  $r$  (Line 3 in Fig. 2),  $\text{dep}(\ast)$  uses the *parse history* (parse decisions from the previous parse steps), i.e.,  $D_{[s:q][L]}$ ,  $D_{[q+1:r][R]}$ , and  $D_{[r:t][L]}$ . This parse history is closely related to the three spans in processing:  $D_{[s:q][L]}$  is the dependent of  $s$  selected for the construction of  $\text{SPAN}_{[s:q][L]}$ , while  $D_{[q+1:r][R]}$  and  $D_{[r:t][L]}$  are the dependents of  $r$ , selected for  $\text{SPAN}_{[q+1:r][R]}$  and  $\text{SPAN}_{[r:t][L]}$ , respectively.

Here, we include the *parse history* in  $\text{dep}(\ast)$  to model the relation between the *left* and *right* dependents of  $r$ . By including the left-side dependent  $D_{[q+1:r][R]}$  and right-side dependent  $D_{[r:t][L]}$  of  $r$  in  $\text{dep}(\ast)$  at Line 3 of Fig. 2, we evaluate the relations among  $(r, D_{[q+1:r][R]}, D_{[r:t][L]})$ , simultaneously, along with the evaluation of the dependency arc connecting  $s$  and  $r$ . Regarded as *out-of-span* relations [10], the relations between the left and right dependents have been ignored in existing span-based algorithms, and to the best of our knowledge, this research may be the first attempt to model them in a span-based approach.

The parse tree with the maximum score, which is derived for the input tree sentence, is given by  $\text{SPAN}_{[1:n][L]}$  for a sentence length of  $n$ , and the parse tree can be constructed by backtracking the parameters recorded in  $B_{[s:t][d]}$  and  $D_{[s:t][d]}$ .

With a simple look through the pseudocode (Figs. 2 and 3), the overall time and space complexities of the algorithm are  $O(n^4)$  and  $O(n^2)$ , respectively.

```

1  TERNARYCOMBINATION( $s, q, r, t, d$ ) :
2  if  $d == 'L'$  then
3      score =  $K_{[s:q][L]} + K_{[q+1:r][R]} + K_{[r:t][L]} + dep(s, r, D_{[s:q][L]}, D_{[q+1:r][R]}, D_{[r:t][L]})$ 
4  else
5      score =  $K_{[s:r][R]} + K_{[r:q-1][L]} + K_{[q:t][R]} + dep(t, r, D_{[s:r][R]}, D_{[r:q-1][L]}, D_{[q:t][R]})$ 
6  end
7  return score

```

Fig. 2 Pseudo-code for ternary-span combination.

```

1  PROPOSEDALGORITHM( $x$ ) :
2  // Initialization
3   $K_{[s:s][d]} \quad \forall s \in 1..n; d \in \{'L', 'R'\};$  // the max score
4   $B_{[s:s][d]} \quad \forall s \in 1..n; d \in \{'L', 'R'\};$  // boundary  $q$  giving the max score
5   $D_{[s:s][d]} \quad \forall s \in 1..n; d \in \{'L', 'R'\};$  // dependent  $r$  giving the max score
6  for  $k = 1..n$  do
7      for  $s = 1..n$  do
8           $t = s + k;$  //  $s, t$  are start and end indices
9          if  $t > n$  then BREAK
10         // left-edge is head
11          $Q, R = \text{argmax}_{q,r}(\text{TERNARYCOMBINATION}(s, q, r, t, 'L'))$ 
12          $K_{[s:t][L]} = \text{TERNARYCOMBINATION}(s, Q, R, t, 'L')$ 
13          $B_{[s:t][L]} = Q$ 
14          $D_{[s:t][L]} = R$ 
15         // right-edge is head
16          $Q, R = \text{argmax}_{q,r}(\text{TERNARYCOMBINATION}(s, q, r, t, 'R'))$ 
17          $K_{[s:t][R]} = \text{TERNARYCOMBINATION}(s, Q, R, t, 'R')$ 
18          $B_{[s:t][R]} = Q$ 
19          $D_{[s:t][R]} = R$ 
20     end
21 end

```

Fig. 3 Pseudo-code for the proposed algorithm.

### 3. Comparison with Other Algorithms

Considering certain parse history is an attempt to capture syntactic relations beyond the explicitly-represented by dependency arcs. There has been a recent trend of considering more parse history in graph-based approaches ranging from *one-side sibling* [6] to *tri-sibling* [3]. Table 1 lists seven different dependency chart parsing algorithms. These algorithms differ in the method used for building larger subtrees from smaller subtrees. In this section, we focus on how these algorithms differ in using *parse history*, when determining a new dependency arc connecting from  $h$  (head) to  $d$  (dependent).

#### 3.1 Constituent-Based Algorithms

Algs. 1–5 in Table 1 are *constituent*-based algorithms. Table 1 shows the step(s) these algorithms use to derive *subtree*  $\hat{\leftarrow}_{h,d}$  by combining the two subtrees  $\hat{\leftarrow}_h$  and  $\hat{\leftarrow}_d$ . Here,  $\hat{\leftarrow}_h$  and  $\hat{\leftarrow}_d$  represent the *subtrees* dominated by  $h$  and  $d$ , respectively;  $\hat{\leftarrow}_{h,d}$  represents the newly constructed subtrees produced by combining  $\hat{\leftarrow}_h$  and  $\hat{\leftarrow}_d$ .

The derivation of  $\hat{\leftarrow}_{h,d}$  in Alg. 1 (Table 1) is processed in a single step by augmenting  $\hat{\leftarrow}_h$  and  $\hat{\leftarrow}_d$  on the basis of

their dependency. The notation of  $K(\hat{\leftarrow}_{h,d})$  in Table 1, represents the score of subtree  $\hat{\leftarrow}_{h,d}$ . In Alg. 1, this is calculated as:  $K(\hat{\leftarrow}_h) + K(\hat{\leftarrow}_d) + dep(*)$ . We define the dependency function  $dep(*)$  as:  $dep(h, d, (\hat{\leftarrow}_h, \hat{\leftarrow}_d))$ , which indicates that the available parse history for Alg. 1 includes  $\hat{\leftarrow}_h$  and  $\hat{\leftarrow}_d$  (represented by the dashed arcs in Fig. 4 (a)).

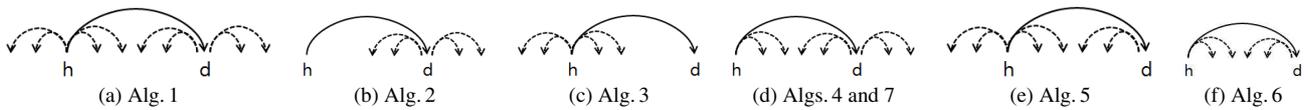
Algs. 2–5 are variants of Alg. 1, and are based on *two*-step operations. *Two*-step processing improves the computational complexity of Alg. 1 from  $O(n^5)$  to  $O(n^4)$ .

In Alg. 2, the first step generates a partial result  $\hat{\leftarrow}_d^h$ , and evaluates the possibility that word  $h$  dominates  $\hat{\leftarrow}_d$ . Score  $K(\hat{\leftarrow}_d^h)$  is defined as  $dep(*) + K(\hat{\leftarrow}_d)$ . The second step combines  $\hat{\leftarrow}_h$  and  $\hat{\leftarrow}_d^h$  into a new constituent. Evaluating the dependency between  $h$  and  $d$  in the first step, the available parse history for Alg. 2 is  $(h, \hat{\leftarrow}_d)$ , which excludes all the selected dependents of  $h$  (Fig. 4 (b)). Alg. 3 is similar to Alg. 2. Its available parse history is  $(\hat{\leftarrow}_h, d)$ , which excludes all the dependents of  $h$  (Fig. 4 (c)).

In Algs. 4 and 5, one of the constituents is divided into two parts (spans). The first step combines the constituent with the closest span by evaluating the dependency between them. During the second step, the other span is attached to the result from the first step to form a new constituent. Now, the parse history is  $(\hat{\leftarrow}_h, \hat{\leftarrow}_d)$  for Alg. 4 (Fig. 4 (d)), and  $(\hat{\leftarrow}_h,$

**Table 1** Overview of dependency chart parsing algorithms: Algs. 1–5 are *constituent*-based and Algs. 6–7 are *span*-based.

	First step	Second step	Score of the new subtree	Time/Space Complexity
			Dependency evaluation function $dep(*)$	
Alg. 1			$K(\triangleleft_h) = K(\triangleleft_h) + K(\triangleleft_d) + dep(*)$ $dep(*) = dep(h, d, (\triangleleft_h, \triangleleft_d))$	$O(n^5)/O(n^3)$
Alg. 2			$K(\triangleleft_h^{\bullet}) = dep(*) + K(\triangleleft_d)$ $K(\triangleleft_h) = K(\triangleleft_h) + K(\triangleleft_h^{\bullet})$ $dep(*) = dep(h, d, (\triangleleft_d))$	$O(n^4)/O(n^3)$
Alg. 3			$K(\triangleleft_h^{\bullet d}) = K(\triangleleft_h) + dep(*)$ $K(\triangleleft_h) = K(\triangleleft_h^{\bullet d}) + K(\triangleleft_d)$ $dep(*) = dep(h, d, (\triangleleft_h))$	$O(n^4)/O(n^3)$
Alg. 4			$K(\triangleleft_h^{\bullet d}) = K(\triangleleft_h) + K(\triangleleft_d) + dep(*)$ $K(\triangleleft_h) = K(\triangleleft_h) + K(\triangleleft_h^{\bullet d})$ $dep(*) = dep(h, d, (\triangleleft_h, \triangleleft_d))$	$O(n^4)/O(n^3)$
Alg. 5			$K(\triangleleft_h^{\bullet d}) = K(\triangleleft_h) + K(\triangleleft_d) + dep(*)$ $K(\triangleleft_h) = K(\triangleleft_h^{\bullet d}) + K(\triangleleft_d)$ $dep(*) = dep(h, d, (\triangleleft_h, \triangleleft_d))$	$O(n^4)/O(n^3)$
Alg. 6			$K(\triangleleft_h^{\bullet d}) = K(\triangleleft_h) + K(\triangleleft_d) + dep(*)$ $K(\triangleleft_h^{\bullet d}) = K(\triangleleft_h^{\bullet d}) + K(\triangleleft_d)$ $dep(*) = dep(h, d, (\triangleleft_h, \triangleleft_d))$	$O(n^3)/O(n^2)$
Alg. 7			$K(\triangleleft_h^{\bullet d}) = K(\triangleleft_h) + K(\triangleleft_d) + K(\triangleleft_d) + dep(*)$ $dep(*) = dep(h, d, (\triangleleft_h, \triangleleft_d, \triangleleft_d))$	$O(n^4)/O(n^2)$



**Fig. 4** Available parsing history (represented as dashed lines), for each parsing algorithm when determining dependency between  $h$  and  $d$ .

4) for Alg. 5 (Fig. 4 (e)).

For more information on Algs. 2 and 4, refer to [5]. Algs. 3 and 5 can be implemented in a similar way.

### 3.2 Span-Based Algorithms

In Table 1, Algs. 6–7 are span-based algorithms; they derive a new span  $\triangleleft_h^{\bullet d}$  by combining  $\triangleleft_h$ ,  $\triangleleft_d$  and  $\triangleleft_d$ . This derivation in Alg. 6 is processed with two steps. The first step

generates  $\triangleleft_h^{\bullet d}$ , which is known as an *incomplete span* in [5], by combining two spans  $\triangleleft_h$  and  $\triangleleft_d$  and evaluating the dependency between them. In the second step,  $\triangleleft_h^{\bullet d}$  is extended to  $\triangleleft_h^{\bullet d}$  by combining with  $\triangleleft_d$ . This algorithm is notable for its efficiency, with a computational complexity of  $O(n^3)$ . The parse history available for Alg. 6 include two spans  $\triangleleft_h$  and  $\triangleleft_d$ , as shown in Fig. 4 (f).

Alg. 7 is the proposed algorithm, with a parse history of  $(\triangleleft_h, \triangleleft_d, \triangleleft_d)$ , as shown in Fig. 4 (d).

**Table 2** The subtrees modeled by each span-based algorithm. The dashed arcs are the parse history used for determining the solid arcs.

	(a) Single arc	(b) One-side sib.	(c) Grandparent	(d) <i>Tri</i> -sibling	(e) <i>Grand</i> -sibling	(f) Two-side sib.
$1^{st}$ -order[4]	o					
$2^{nd}$ -order[6]	o	o				
<i>high</i> -order[7]	o	o	o			
$3^{rd}$ -order[3]	o	o	o	o	o	
Proposed.II	o	o				o

There are a couple of extensions to Alg. 6, which are known as  $2^{nd}$ -order [2], *high*-order [7] and  $3^{rd}$ -order algorithms [3], by specifying certain parse history. In the  $2^{nd}$ -order algorithm, the evaluation of a new dependent (the node connected by a solid arc in Table 2 (b)) is based on the most recently determined sibling node (the node connected by a dashed arc in Table 2 (b)). We refer to the two dependents in Table 2 (b) as *one-side siblings*, since both are located on the same side of their head.

Carreras called his approach a *high*-order algorithm [7]. In addition to the *one-side sibling*, it includes the *grandparent* node, which is the head of head  $h$ , as shown in Table 2 (c). Koo and Collins proposed a  $3^{rd}$ -order algorithm by further extending the  $2^{nd}$ -order algorithms. The  $3^{rd}$ -order algorithm primarily models *triple* arcs, shown as Tables 2 (d) and (e). For further details, see [3].

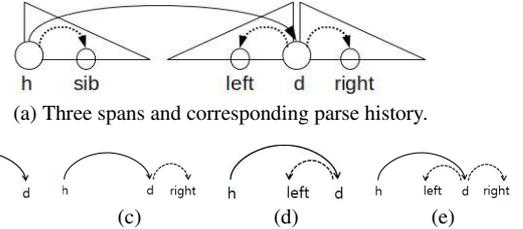
The original work with Alg. 6, such as in [2], did *not* use parse history; it is based on modeling a *single arc* as shown in Table 2 (a). Thus, Alg. 6 is also known as the  $1^{st}$ -order algorithm [3].

### 3.3 Comparison

Alg. 6 in Table 1 uses the least amount of parse history, yet, it is the most efficient; Alg. 1 uses the greatest amount of parse history, but has the highest complexity. Of the algorithms with a time complexity of  $O(n^4)$ , Algs. 4 and 5 use more parse history than Algs. 2 and 3. Our algorithm uses a similar parse history as Alg. 4 (Fig. 4 (d)), but ours obtains a better space complexity of  $O(n^2)$ , when compared with  $O(n^3)$  for Alg. 4.

The parse history used by each span-based algorithm is clearly specified in Table 2. The  $3^{rd}$ -order algorithm models more relations than the  $1^{st}$ -,  $2^{nd}$ - and *high*-order algorithms (Table 2). In addition to *one-side sibling* (Table 2 (b)) and *grandparent* relations (Table 2 (c)), the  $3^{rd}$ -order algorithm models *tri-sibling* and *grand-sibling* relations as shown in Tables 2 (d) and (e).

As shown in Fig. 5 (a), our algorithm can model parse history involved in three spans. Figures 5 (b)–(e) show some of the relations enclosed in Fig. 5 (a), which are modeled in our algorithm (Table 4). The prominent difference between our algorithm and other span-based algorithms is that existing span-based algorithms focus on modeling, or are biased in modeling dependency arcs that point toward the same direction, i.e., either from left to right (all dependency arcs in

**Fig. 5** Relations modeled in the proposed algorithm.**Table 3** Comparison of the counts/percentages of the heads dominating *triple one-side dependents* (*tri-siblings*).

Count of heads	Counts/percentages of heads that dominates triple one-side dependents.		
	(a)	(b)	
English	486,708	18,141 /3.73%	27,557/ 5.66%
Chinese	223,678	960 /0.43%	9,924/4.44%

**Table 4** Feature types modeled in the proposed algorithm.

Feature Types	Descriptions
FEA.ARC (Fig. 5 (a))	uni-gram bi-gram surrounding in-between
FEA.ONE-SIDE.SIB (Fig. 5 (b))	$h$ -POS $d$ -POS <i>sib</i> -POS $d$ -POS <i>sib</i> -POS
FEA.GRANDCHILD1 (Fig. 5 (c))	$h$ -POS $d$ -POS <i>right</i> -POS $d$ -POS <i>right</i> -POS
FEA.GRANDCHILD2 (Fig. 5 (d))	$h$ -POS $d$ -POS <i>left</i> -POS $d$ -POS <i>left</i> -POS
FEA.TWO-SIDE.SIB (Fig. 5 (e))	$h$ -POS $d$ -POS <i>left</i> -POS <i>right</i> -POS $d$ -POS <i>left</i> -POS <i>right</i> -POS $h$ -POS <i>left</i> -POS <i>right</i> -POS <i>left</i> -POS <i>right</i> -POS

Tables 2 (a)–(e) are in this direction), or from right to left. Such a *bias* is more significant for the third-order algorithm by modeling the relations of *tri*- and *grand*-siblings. The proposed algorithm is helpful to remove this bias by modeling *two-side siblings* (dashed arcs in Table 2 (f)).

## 4. Dependency Annotation and Parsing Algorithm

Previous work has shown that the use of parse history is crucial to achieve a high performance parsing. The comparison provided in Sect. 3 shows the trade-off between a decrease

in parse complexity and utilizing a greater amount of parse history. Thus, we must choose a parsing algorithm that balances the parse complexity and the use of the parse history.

Another factor related to the appropriate selection of a parse algorithm is the *structural bias* of the corpus. According to the description given in [11], the structural bias of a corpus is the tendency for a corpus to contain more specific types of structured subtrees. For example, the English dependency treebank (abbreviated as English data) converted from the English Penn Treebank [8] using Penn2Malt<sup>†</sup>, contains more of the subtree as Table 3 (a), compared with the Chinese treebank of CoNLL'07 (abbreviated as Chinese data).

In addition to the inherent characteristics of the language, structural bias of a corpus is partly due to the adopted dependency annotation method. Depending on the dependency annotation scheme, a syntactic relation may be represented differently. For example, we can annotate the *Coordinate Conjunction* as the *Head* (CCH-type), with the left and right conjuncts as its dependents; whereas other methods may set the *Coordinate Conjunction* as the *Dependent* (CCD-type) by annotating one of the conjuncts as its head [10].

As SVO languages, both English and Chinese share many syntactical similarities. Yet, the portion of the heads that dominating triple one-side dependents (*tri-siblings* as subtree of Table 3 (a)) in the English data is 3.73%, which is much higher than that in the Chinese data. The reason for this is partly due to the difference in annotating coordinations. In the English corpus, most verbal-coordinations are dominated by their left-most conjunct<sup>††</sup> [12], upon which other conjuncts and conjunction(s) are attached. Thus, it is necessary to model the subtree of Table 3 (a) in English corpus for coordination disambiguation. In this sense, among the algorithms of Table 2, the 3<sup>rd</sup>-order algorithm [3] is the better choice for the English corpus. Actually, the system of [3] includes some features, with which the 3<sup>rd</sup>-order algorithm gives better coordination disambiguations.

The Chinese corpus is one with CCH-type coordinations. Considering the importance for coordination disambiguation in parsing, an algorithm that models the subtrees shown as Table 2 (f) (the pair of arcs shown with the dashed lines) is a better choice.

## 5. Experiments

We evaluated the performance based on an *Unlabeled Attachment Score* (UAS), following the measure defined in the CoNLL shared task [13], [14]. Most data-driven dependency parsers [6], [15] adopt *two-stage* for labeled parsing by deriving dependency labels at the *second-stage* upon the *unlabeled* dependency tree of the *first-stage*. Multi-labeled classifiers are often adopted to label each dependency arc. In this sense, the proposed algorithm has little effect on dependency labelling, we did not evaluate with *Labeled Attachment Score* (LAS) in this paper.

The parsing model in this paper is conventionally de-

finied by selecting the maximum score parse tree. The score of the parse tree is the sum of the scores of the dependency arcs. The dependency function  $dep(*)$  (included in Lines 3 and 5 of Fig. 2) is defined as:

$$dep(*) = Feature(*) \times Weight(*)$$

$Feature(*)$  represents the features related to the target dependency arc, while  $Weight(*)$  is the associated weight function, which is learned through the *average perceptron* [16]. The average perceptron allows for fast learning with a respectable performance and a simple implementation, compared with other structured training algorithms, such as the max-margin model [17], the margin infused relaxed algorithm (MIRA) [6] and the log-linear model [15].

### 5.1 Features

Our algorithm can model the relations represented by the subtrees shown in Fig. 5; we define the *five* types of features accordingly, as shown in Table 4. The  $FEA.ARC$  features are directly related to the target dependency arc, including the *uni-gram*, *bi-gram*, *surrounding* and *in-between* features, as defined in [18]. Other types of features are targeted to model the *one-side sibling*, *grand-child* and *two-side sibling* relations, respectively.

In Table 4, for example, we use  $FEA.ONE.SIDE.SIB$  to represent features modeling the *one-side sibling* relation between the arc linking from  $h$  to  $d$  and arc connecting from  $h$  to  $sib$  (Fig. 5 (b)). Since the pair of arcs in *one-side sibling* indicates the relation among three nodes:  $h$ ,  $d$ , and  $sib$ , we define  $FEA.ONE.SIDE.SIB$  to include a POS tag tri-gram of  $h$ ,  $d$  and  $sib$  (represented as  $h-POS\ d-POS\ sib-POS$  in Table 4), and a POS tag bi-gram of  $d$  and  $sib$  ( $d-POS\ sib-POS$ ). The other feature types in Table 4, are defined in a similar way.

Similar features of  $FEA.ONE.SIDE.SIB$ ,  $FEA.GRAND.CHILD1$  and  $FEA.GRAND.CHILD2$  has also been defined and used in the systems of [6], [18], and [7], while,  $FEA.TWO.SIDE.SIB$  features in Table 4, are newly defined for our algorithm. The  $FEA.TWO.SIDE.SIB$  features are defined primarily to evaluate the relations between the *left* and *right* dependents of  $d$  shown as Fig. 5 (e).

### 5.2 Results

We evaluated our system on Chinese CoNLL-2007 data and the English data converted from the English Penn Treebank. For the Chinese data, we used the original included training and test sets; for the English data, we used the sections 2–21 for training and the section 23 for test as convention. For both data sets, we used the gold standard part-of-speech tags in the treebanks.

<sup>†</sup>Penn2Malt is an automatic tool for converting a Penn Treebank style phrase structure tree into dependency tree. <http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html>

<sup>††</sup>In case of a nominal-coordination, the right-most conjunct is annotated as the head in the English data with Penn2Malt.

### 5.2.1 Effect of Each Feature Type

First, we evaluated the performance of our algorithm with each type<sup>†</sup> of features described in Table 4. As shown in Table 5, adding features modeling parse history such as FEA.ONESIDE.SIB, FEA.GRANDCHILD1, FEA.GRANDCHILD2 or FEA.TWOSIDE.SIB on base feature FEA.ARC, results in better performances than using only FEA.ARC type features.

With FEA.ONESIDE.SIB, our system obtained performances of 91.20% and 88.40% for the English and Chinese data, respectively; these performances are better than those obtained from FEA.GRANDCHILD1 and FEA.GRANDCHILD2 (Table 5). These results also indicate that FEA.ONESIDE.SIB is one of the most important features for both data.

With FEA.TWOSIDE.SIB, our system obtained a performance of 88.58% for the Chinese data, and this performance is better than the performance with other types of features.

### 5.2.2 Effect of Modeling Two-Side Siblings

CCH-type coordinate are the primary case that require to model *two-side siblings*. Table 6 shows that with FEA.TWOSIDE.SIB, 83.78% of the conj.head<sup>††</sup> are correctly attached to their heads in the Chinese data, which is an improvement of 1.48%, compared to the result using FEA.ONESIDE.SIB.

Including a coordination-specialized feature COOR.FEA, defined as:

if <i>left.conj.head.POS</i> == <i>right.conj.head.POS</i>	COOR.FEA = 1
else	COOR.FEA = 0

the system provides more accurate results on coordination disambiguation than using only FEA.TWOSIDE.SIB (Table 6). Further, as shown in Table 7, COOR.FEA is more helpful for identifying remote conj.heads from their conjunctions. Here, COOR.FEA is a flag-like feature that provides symmetric information for the coordination by comparing the POS tags of its conjuncts.

The case of a verb dominating a subject at its

**Table 5** Performance comparison of the proposed algorithm on feature types defined in Table 4.

Features	UAS (English)	UAS (Chinese)
FEA.ARC	90.73%	87.34%
+ FEA.ONESIDE.SIB	<b>91.20%</b>	88.40%
+ FEA.GRANDCHILD1	90.82%	87.97%
+ FEA.GRANDCHILD2	90.97%	87.92%
+ FEA.TWOSIDE.SIB	90.99%	<b>88.58%</b>

**Table 6** Comparison between the overall parsing performance and the percentage of conj.heads that were correctly attached to their heads for the Chinese data.

FEA.ONESIDE.SIB		FEA.TWOSIDE.SIB		FEA.TWOSIDE.SIB + COOR.FEA	
overall	conj.head	overall	conj.head	overall	conj.head
88.40%	82.30%	88.58%	83.78%	88.78%	84.11%

left-side and an object at its right-side (SUB.VERB.OBJ), is a typical syntactic structure for a SVO language. With FEA.TWOSIDE.SIB, about 89.72% of SUB.VERB.OBJS are correctly identified for the English data, which was 7.70% higher than the result of 82.02% gained with FEA.ONESIDE.SIB.

However, the portion of SUB.VERB.OBJ is rather lower than expected in the English data. Among test sentences (totally 2,416), only 336 sentences are dominated by SUB.VERB.OBJ type ROOTs. The reason for such a low portion is partly due to the existence of so many SUB.VERB.VMOD ROOTs (total 1,146 cases) and annotation scheme assigning auxiliary verb as the head of main verb (614 cases) in the English data. For cases of SUB.VERB.VMOD and SUB.VERB.VC (the case including auxiliary verb), the effect of modeling *two-side siblings* is limited since the correlations between left- and right-side dependents of VERB in these cases are weak. And because of the low portion of SUB.VERB.OBJ, modeling *two-side siblings* has little effect on the overall performance in the English data.

### 5.2.3 Effect of Combined Features

We then evaluated our system empirically on the combined features. For the Chinese data, the best result was 89.41% (Table 8) with (FEA.ONESIDE.SIB + FEA.TWOSIDE.SIB + COOR.FEA).

For the English data, the best performance of our system was 91.99%, which was obtained using (FEA.ONESIDE.SIB + FEA.GRANDCHILD1 + FEA.GRANDCHILD2) (Table 8).

**Table 7** Performance comparison of conj.heads relative to their distance to the conjunction.

Distance <sup>a</sup>	0	1	2	3	>= 4
FEA.TWOSIDE.SIB	93.19%	78.00%	56.25%	55.00%	57.14%
FEA.TWOSIDE.SIB + COOR.FEA	92.88%	79.21%	64.49%	60.20%	59.22%

<sup>a</sup>The *distance* is the number of words that appeared between the conj.head and conjunction. For example, given ‘Tom and the pretty Jerry’, Tom and Jerry are 0-distance and 2-distance conj.heads, respectively.

**Table 8** Performances of the proposed algorithm on combined features.

Name	Features	UAS (English)	UAS (Chinese)
Proposed.I	+ FEA.ONESIDE.SIB + FEA.GRANDCHILD1 + FEA.GRANDCHILD2	<b>91.99%</b>	88.49%
Proposed.II	+ FEA.ONESIDE.SIB + FEA.TWOSIDE.SIB + COOR.FEA	91.03%	<b>89.41%</b>

<sup>†</sup>FEA.ARC features were included in each experiment.

<sup>††</sup>*conj.head* refers to the head word of a conjunct. In CCH-type coordination, each conj.head is attach to the conjunction.

**Table 9** Comparisons with other systems.

system	UAS (English)	UAS (Chinese)
proposed.I	91.99%	88.49%
proposed.II	91.03%	<b>89.41%</b>
2 <sup>nd</sup> -order [6]	91.5%	88.29% <sup>a</sup>
proposed.I-word	92.39%	88.65%
3 <sup>rd</sup> -order [3]	<b>93.04%</b>	-
Chen [19]	92.21%	-
Sagae [20]	-	88.94%
Nakagawa [15]	-	88.88%
high-order [7]	93.14% <sup>b</sup>	86.20%

<sup>a</sup>Evaluated with MSTParser (V0.2).

<sup>b</sup>Evaluated on development data (section 24 of Penn Treebank), based on the report given by [3].

### 5.3 Comparison with Other Parsers

#### 5.3.1 Performance

Among the systems listed in Table 9, the 2<sup>nd</sup>-order system and ours, proposed.I and proposed.II, are based on POS tagged features. The 2<sup>nd</sup>-order system was based on FEA.ONE.SIDE.SIB. In addition to FEA.ONE.SIDE.SIB, proposed.I and proposed.II model more types of features (Table 8). For the English data, proposed.I obtained a performance of 91.99%, which outperformed the 2<sup>nd</sup>-order system (Table 9). For the Chinese data, proposed.II gave the best performance by including COOR.FEA features.

In Table 9, the systems of [3], [7], and [19] include surface words in their feature vector, and proposed.I-word is the version using word features based on the same feature template defined for proposed.I. Proposed.I-word gave a performance of 92.39% for the English data, which is better than the performance of [19], and lower than the performance obtained by the 3<sup>rd</sup>-order system of [3]. In our opinion the performance gap between the 3<sup>rd</sup>-order system and ours is on modeling *tri-siblings*, especially, disambiguating CCD-type of coordination (e.g., Table 3 (a)) in the 3<sup>rd</sup>-order system. We compared the performance our system on the Chinese data with two top-ranked systems [15] and [20] for the CoNLL'07 shared task, since the systems of [3], [19] did not evaluate on the same Chinese data. Comparing the performances of [15] and [20], our systems gave the best performances on the Chinese data<sup>†</sup>.

The work of [3] also reported the performance on the development data set (the section 24 of the English Penn treebank) as 93.49%. According to report of [3], the emulated *high-order* system of [7] gave a similar (or slightly lower) performance as 93.14% on the same data set. Despite of the high performance for the English data, the *high-order* system gave a performance of 86.20% for the Chinese data (Table 9). In our opinion, lack of modeling two-side siblings is the main reason for the low performance of [7] on the Chinese data.

<sup>†</sup>The systems of [15] and [20] also reported their performances on the English data included in CoNLL'07, yet the English data of CoNLL'07 is not identical to the one used here.

**Table 10** Efficiency Comparisons on the test data of English.

system	time
proposed.I-word (Python)	489 s
3 <sup>rd</sup> -order (emulated in Python)	1233 s

#### 5.3.2 Efficiency

We implemented our systems in Python and ran on a single machine of 64-bit Intel i5 quad-core with 3.3 GHz CPUs. The proposed.I-word system spent totally 41.2 hours on training (for 10 iterations).

Both the 3<sup>rd</sup>-order system and our systems have a time complexity of  $O(n^4)$ , while, our algorithm is more efficient. When comparing the the total time spent for decoding the English test data, proposed.I-word spent 489 seconds, while the emulated 3<sup>rd</sup>-order system spent much more time than ours (Table 10).

## 6. Conclusion

This paper proposed a new span-based dependency chart parsing algorithm, that is suitable for a language or corpus where modeling the left and right dependents is essential to achieve a high parsing performance.

By modeling the relation between the left and right dependents of a head, the algorithm provides a solution for coordination disambiguation when the conjunction is annotated as the head of its left and right conjuncts. When applied to the Chinese data of the CoNLL'07 shared task, our algorithm achieved a better parse performance on the coordinate structures, which eventually improved the overall parsing performance.

We believe our algorithm can also achieve better performances for Arabic, Czech, and Slovene corpora of CoNLL 2007 [10] than existing span-based algorithms through an improved coordination disambiguation, since the coordinations in these corpora are also of CCH-type. In case of SVO languages, the proposed algorithm also provides a platform to model the relation between the subject and object with regard to their common predicate verb.

## Acknowledgments

This work was supported in part by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korean government (MEST No. 2012-0004981), in part by the BK 21 Project in 2012, and in part by IT Consilience Creative Program of MKE and NIPA (C1515-1121-0003).

## References

- [1] M. Collins, J. Hajic, L. Ramshaw, and C. Tillmann, "A statistical parser for Czech," Proc. 37th Annual Meeting of the Association for Computational Linguistics, College Park, Maryland, USA, pp.505–512, Association for Computational Linguistics, June 1999.
- [2] R. McDonald, F. Pereira, K. Ribarov, and J. Hajic, "Non-projective

- dependency parsing using spanning tree algorithms,” Proc. Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Vancouver, British Columbia, Canada, pp.523–530, Association for Computational Linguistics, Oct. 2005.
- [3] T. Koo and M. Collins, “Efficient third-order dependency parsers,” Proc. 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, pp.1–11, Association for Computational Linguistics, July 2010.
- [4] J. Eisner, “Three new probabilistic models for dependency parsing: An exploration,” Proc. 16th International Conference on Computational Linguistics (COLING-96), Copenhagen, pp.340–345, Aug. 1996.
- [5] J. Eisner and G. Satta, “Efficient parsing for bilexical context-free grammars and head automaton grammars,” Proc. 37th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 1999.
- [6] R. McDonald, K. Lerman, and F. Pereira, “Multilingual dependency analysis with a two-stage discriminative parser,” Conference on Natural Language Learning (CoNLL), 2006.
- [7] X. Carreras, “Experiments with a higher-order projective dependency parser,” Proc. CoNLL Shared Task Session of EMNLP-CoNLL 2007, Prague, Czech Republic, pp.957–961, Association for Computational Linguistics, June 2007.
- [8] M. Marcus, B. Santorini, and M. Marcinkiewicz, “Building a large annotated corpus on english: The penn treebank,” Computational Linguistics, 1993.
- [9] D.H. Younger, “Recognition and parsing of context-free languages in time  $n^3$ ,” Information and Control, vol.4, no.12, pp.361–379, 1967.
- [10] R. McDonald and J. Nivre, “Characterizing the errors of data-driven dependency parsing models,” Proc. 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp.122–131, 2007.
- [11] Y. Goldberg and M. Elhadad, “Inspecting the structural biases of dependency parsing algorithms,” Proc. Fourteenth Conference on Computational Natural Language Learning, Uppsala, Sweden, pp.234–242, Association for Computational Linguistics, July 2010.
- [12] E. Pitler, “Attacking parsing bottlenecks with unlabeled data and relevant factorizations,” Proc. 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Jeju Island, Korea, pp.768–776, Association for Computational Linguistics, July 2012.
- [13] S. Buchholz and E. Marsi, “Conll-x shared task on multilingual dependency parsing,” Proc. 10th Conference on Natural Language Learning (CoNLL-X), New York City, USA, Association for Computational Linguistics, June 2006.
- [14] J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret, “The CoNLL 2007 shared task on dependency parsing,” Proc. CoNLL Shared Task Session of EMNLP-CoNLL 2007, Prague, Czech Republic, pp.915–932, Association for Computational Linguistics, June 2007.
- [15] T. Nakagawa, “Multilingual dependency parsing using global features,” Proc. CoNLL Shared Task Session of EMNLP-CoNLL 2007, Prague, Czech Republic, pp.952–956, Association for Computational Linguistics, June 2007.
- [16] M. Collins and B. Roark, “Incremental parsing with the perceptron algorithm,” Proc. 42nd Meeting of the Association for Computational Linguistics (ACL’04), Main Volume, Barcelona, Spain, pp.111–118, July 2004.
- [17] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning, “Max-margin parsing,” Proc. EMNLP 2004, ed. D. Lin and D. Wu, Barcelona, Spain, pp.1–8, Association for Computational Linguistics, July 2004.
- [18] R. McDonald, F. Pereira, K. Crammer, and K. Lerman, “Global inference and learning algorithms for multi-lingual dependency parsing,” Unpublished manuscript, 2007.
- [19] W. Chen, J. Kazama, Y. Tsuruoka, and K. Torisawa, “Improving graph-based dependency parsing with decision history,” Coling 2010: Posters, Beijing, China, pp.126–134, Aug. 2010.
- [20] K. Sagae and J. Tsujii, “Dependency parsing and domain adaptation with LR models and parser ensembles,” Proc. CoNLL Shared Task Session of EMNLP-CoNLL 2007, Prague, Czech Republic, pp.1044–1050, Association for Computational Linguistics, June 2007.



**Meixun Jin** received a B.S. degree in Mathematics Education from YanBian University, China, in 1993. From 1995 to 2000, she was a lecturer at YanBian University of Sci. and Tech., China. She received her M.S. degree in 2004 from the Graduate School of Information Technology, Pohang University of Science and Technology (POSTECH), Korea. She is currently a Ph.D. candidate at the Department of Computer Science, POSTECH. Her main research interests include natural language processing, syntactic parsing, and machine translation.



**Yong-Hun Lee** received his B.S. degree in Computer Science and Engineering from Soongsil University, Korea, in 2002, and his M.S. and Ph.D. degrees in Computer Science and Engineering from POSTECH, in 2004 and 2011. His research interests include natural language processing, syntactic analysis and machine learning.



**Jong-Hyeok Lee** received his B.S. degree in Mathematics Education from Seoul National University in 1980, and his M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), in 1982 and 1988. From Nov. 1989 to Jan. 1991, he worked as a visiting researcher at NEC C&C Institute, Japan. He was an assistant professor at POSTECH, and since 2003 has been a full professor. He also worked as a visiting scholar for CRL/NMSU, USA and RALI/Univ. of Montreal, Canada in 1998 and 2007, respectively. His research interests include natural language processing, machine translation, and information retrieval.