

# Beyond Chart Parsing: An Analytic Comparison of Dependency Chart Parsing Algorithms

Meixun Jin, Hwidong Na and Jong-Hyeok Lee

Department of Computer Science and Engineering  
Pohang University of Science and Technology (POSTECH)  
San 31 Hyoja Dong, Pohang, 790-784, Republic of Korea  
meixunj, leona, jhlee@postech.ac.kr

## Abstract

In this paper, we give a summary of various dependency chart parsing algorithms in terms of the use of parsing histories for a new dependency arc decision. Some parsing histories are closely related to the target dependency arc, and it is necessary for the parsing algorithm to take them into consideration. Each dependency treebank may have some unique characteristics, and it requires for the parser to model them by certain parsing histories. We show in experiments that proper selection of the parsing algorithm which reflect the dependency annotation of the coordinate structures improves the overall performance.

## 1 Introduction

In data-driven graph-based parsing, a chart parser is frequently combined with a learning method to derive and evaluate the parse forest and output the optimal parse tree (Chen et al., 2010; Koo and Collins, 2010). The proper selection of a parsing algorithm is important for efficiency and correctness.

Chart parsing is the realization of dynamic programming for syntactic analysis. It is suitable for ambiguous grammars, for example, the grammars of natural languages. Practically, according to the diverse implementations of dynamic programming for dependency syntactic analysis, there are a number of dependency chart parsing algorithms. In this paper, we list a number of bottom-up dependency chart parsing algorithms in terms of their use of the *parsing histories* (section 2).

Incorporating parsing histories into parse tree decoding requires changes to the parsing algorithm. For instance, when decoding a dependency by including the most recently detected *sibling* arc, a modified parsing algorithm has been used

in (McDonald et al., 2006) in contrast to the algorithm used in (McDonald et al., 2005). Parsing histories are partial results generated from previous parsing steps. In a chart parser, these histories can be used in subsequent parsing steps. Previous works have shown that the use of parsing histories helps to resolve syntactic ambiguities (Yamada and Mastumoto, 2003; Nivre et al., 2007b; McDonald et al., 2006; Carreras, 2007; Chen et al., 2010). Obviously, using more histories provides better parsing disambiguation. However, there is a trade-off between using more histories and parsing efficiently. One option is to incorporate only important histories. The selection of different histories requires changes to the parsing algorithm.

Another reason for the careful selection of parsing algorithms is from the diverse dependency annotation strategies. The dependency annotations for the same linguistic structures, i.e., coordinate structures can vary (Section 3.1). Additionally, in our opinion, some linguistic or corpus-oriented characters exist for each training data set. Different parsing algorithms are required to deal with the diversity of the corpus.

## 2 Dependency Chart Parsing Algorithms

A chart parser is the realization of dynamic programming for syntactic analysis. It parses all the substrings of the input sentence and stores the corresponding sub-parse-trees in a data structure called a chart. Dependency chart parsers can be categorized into *constituent*-based and *span*-based parsers depending on the type of substring each cell of the chart yields.

The main difference between constituent-based and span-based algorithms lies in the type of substrings they process. A constituent-based algorithm identifies and parses substrings corresponding to *constituents* and a span-based algorithm does on substrings corresponding to *spans*.

A constituent-based algorithm is a modification

of a phrase-structure (PS) chart parsing algorithm. In a constituent-based dependency parser, the head word of the substring is derived instead of the non-terminal of PS parsing, and information related to the head word is stored in corresponding cell of the chart. In a modified CYK algorithm (Younger, 1967), the cell reserves the possibility for each word of the substring to be the head. Thus,  $n$  kinds of sub-dependency-trees are reserved for processing a substring of length  $n$ . The space complexity for a cell is  $O(n)$ , and the overall space complexity is  $O(n^3)$  and time complexity is  $O(n^5)$  for parsing of a sentence of length  $n$ . For a detailed description, refer to (Nivre, 2006).

A *span* is a *half-constituent* that is formed by splitting a constituent at the position of its *head* (Eisner and Satta, 1999). The span is characterized by the head being located either on the left or right edge. In the span-based dependency chart parser proposed by (Eisner and Satta, 1999), there are two kinds of subtrees reserved in each cell of the chart, i.e., the head is the left-most word or the right-most word. Given this condition, Eisner’s algorithm can parse with a time complexity of  $O(n^3)$  and a space complexity of  $O(n^2)$ .

In bottom-up parsing, either a constituent-based or a span-based algorithm derives parse tree for a sequence by combining two (or more) subsequences with a new dependency arc. These subsequences have been parsed in earlier steps, and they are called as *parsing histories*. These histories are frequently used for a better evaluation of the new dependency arc (Chen et al., 2010). Table 1 lists a number of dependency chart parsing algorithms. Each of them adopts different method to derive a new parse tree from couple of smaller sequences. In the following, we discuss in detail how these algorithms differ in their use of the parsing histories for the new dependency arc decision.

## 2.1 Constituent-Based Algorithms

Table 1 lists the step for various dependency chart parsing algorithms to decide a new dependency arc connecting  $h$ (ead) and  $d$ (ependent).  $\triangleleft_n$  and  $\triangleleft_d$  present the constituents dominated by  $h$  and  $d$  respectively.

The derivation of the new constituent in Alg. 1 (Table 1) is processed in one-step, and the one-step processing can be defined as the function  $f$  of Alg. 1, which involves three parameters: two constituents  $\triangleleft_n$  and  $\triangleleft_d$  and the evaluation of the

dependency arc  $\widehat{h \ d}$ . The dependency between  $(h, d)$  is evaluated by  $score(h, d, (\triangleleft_n, \triangleleft_d))$  defined in Table 1. Here,  $(\triangleleft_n, \triangleleft_d)$  is the context of parsing histories, which are available for the new dependency arc detection in Alg. 1.

Algs. 2-4 listed in Table 1 are variants of Alg. 1, and the derivation of the new constituent over smaller constituents in these algorithms is processed in *two* steps. In Alg. 2, the first step combines one constituent with the detection of the dependency arc, and the process is represented by the function  $f(\widehat{h \ d}, \triangleleft_d)$ ; the combination of the second constituent represented as  $f(\triangleleft_n)$ , is processed at the second step (Table 1, Alg. 2). With such a two-step operation, parsing histories available for the decision of the arc  $\widehat{h \ d}$  in Alg. 2 is  $(h, \triangleleft_d)$ . Comparing to Alg.1 of  $(\triangleleft_n, \triangleleft_d)$ , the histories included in  $\triangleleft_n$  is not available in Alg.2. One benefit of such a two-step operation is that it reduces the time complexity  $O(n^5)$  of Alg. 1 to  $O(n^4)$ .

In Algs. 3 and 4, one of the constituents is divided into two parts (spans). The first step combines the constituent with the closer span and makes a decision about the new dependency arc. The other span is attached to the result of the first step. The available parsing histories is  $(\triangleleft_h, \triangleleft_d)$  for Alg.3, and  $(\triangleleft_n, \triangleleft_d)$  for Alg.4.

The two-step processing requires the reservation of the partial results generated at the first step:  $\widehat{\triangleleft}$  for Alg. 2,  $\widehat{\triangleleft} \triangleleft$  for Alg. 3, and  $\widehat{\triangleleft} \triangleleft \triangleleft$  for Alg. 4 (Table 1). Reserving these partial results in the chart in addition to the constituent  $\triangleleft$ , only increases the *constant factor*, the overall space complexity remains as  $O(n^3)$ .

For more information on Algs. 2 and 3 see (Eisner and Satta, 1999). and Alg. 4 see (Jin, 2011).

## 2.2 Span-Based Algorithms

Alg. 5 is the span-based algorithm proposed by (Eisner, 1996). The algorithm has been widely used in data-driven graph-based dependency parsers, because of its efficiency by parsing with a complexity of  $O(n^3)$ . When combined with a learning method, the training for a data-driven parser involves repeatedly decoding of parse trees. Parsing efficiency is an important factor in such an approach. Some extensions to Alg. 5 have been proposed (Carreras, 2007; McDonald et al., 2006) with the aim of enriching the information available for new dependency arc detection. The work

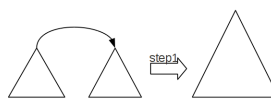
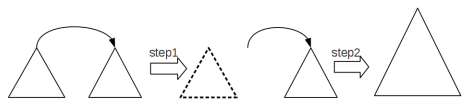
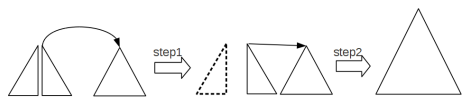
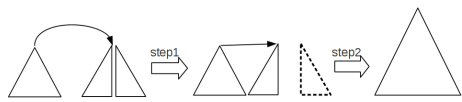
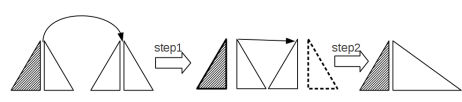
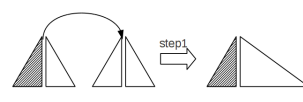
	Chart combinations	Combination function	Time / Space
		Score function for new dependency arc	Complexity
Alg. 1		$f(\triangle_h, \overset{\curvearrowright}{h} \overset{\curvearrowleft}{d}, \triangle_d)$ $score(h, d, (\triangle_h, \triangle_d))$	$O(n^5)/O(n^3)$
Alg. 2		$f(\overset{\curvearrowright}{h} \triangle_h) + f(\overset{\curvearrowright}{h} \overset{\curvearrowleft}{d}, \triangle_d)$ $score(h, d, (h, \triangle_d))$	$O(n^4)/O(n^3)$
Alg. 3		$f(\overset{\curvearrowright}{h} \triangle_h) + f(\triangle_h, \overset{\curvearrowright}{h} \overset{\curvearrowleft}{d}, \triangle_d)$ $score(h, d, (\triangle_h, \triangle_d))$	$O(n^4)/O(n^3)$
Alg. 4		$f(\triangle_h, \overset{\curvearrowright}{h} \overset{\curvearrowleft}{d}, \triangle_d) + f(\overset{\curvearrowright}{h} \triangle_h)$ $score(h, d, (\triangle_h, \triangle_d))$	$O(n^4)/O(n^3)$
Alg. 5		$f(\triangle_h, \overset{\curvearrowright}{h} \overset{\curvearrowleft}{d}, \triangle_d) + f(\overset{\curvearrowright}{h} \triangle_h)$ $score(h, d, (\triangle_h, \triangle_d))$	$O(n^3)/O(n^2)$
Alg. 6		$f(\triangle_h, \overset{\curvearrowright}{h} \overset{\curvearrowleft}{d}, \triangle_d, \triangle_d)$ $score(h, d, (\triangle_h, \triangle_d, \triangle_d))$	$O(n^4)/O(n^2)$

Table 1: Comparison of various dependency chart parsing methods. The dashed part is attached in step2. Algs. 5-6 are *span-based* algorithms, during the step of right-side span construction, the shadowed left-side span remains *unchange*.

of (Koo and Collins, 2010) is a similar propose of the method of Alg. 2 on span-based algorithms.

In terms of the use of the parsing histories, the amount of information available for a new dependency arc decision with Alg. 5 is  $(\triangle_h, \triangle_d)$ , which is about *half* of  $(\triangle_h, \triangle_d)$  of Alg.1. Some common but important relations between pair of dependents for some corpora, such as the relation between the left and right dependents of a head (the pair of dependency arcs shown in Fig. 1(a)), cannot be modeled using such an algorithm.

Alg. 6 (Table 1), is an alternative to Alg. 5. The two-step operation in Alg. 5 merges and becomes a one-step operation in Alg. 6 by direct processing over three spans  $(\triangle_h, \triangle_d, \triangle_d)$ . Such a *ternary-span* combination increases the parsing histories from  $(\triangle_h, \triangle_d)$  of Alg.5 to *three* spans as  $(\triangle_h, \triangle_d, \triangle_d)$  (see the score function of Alg. 6 in Table 1). However, the time complexity of Alg.6

increases from  $O(n^3)$  of Alg.5 to  $O(n^4)$ . Comparing to other  $O(n^4)$  algorithms, Algs.2-4, Alg. 6 is more efficient with a small constant factor. The space complexity is also modest as  $O(n^2)$  which is the same as that for Alg. 5.

The relation between the left and right dependents of a head can be modeled using Alg. 6. In span-based algorithms, the left and right spans sharing the head are treated independently, and the relations between the left and right dependents are often ignored in previous span-based algorithms. To our knowledge, Alg. 6 is the first span-based algorithm to model this. For detailed implementation of Alg.5 refer to (Eisner and Satta, 1999), and Alg.6 to (Jin, 2011).

### 3 Diverse Dependency Annotation Strategies

Since the CoNLL'06 shared tasks for *multilingual dependency parsing* (Buchholz and Marsi, 2006),

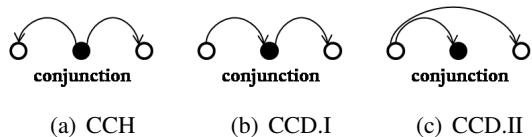


Figure 1: (a) CCH; (b) and (c) two cases of CCD, with left conjunct as head.

it has become common to construct a *universal* parser to derive parse trees of diverse languages in a *unified* way, with less consideration about the varieties among corpora.

In addition to the variations of different language, there are some variations because of the diverse strategies for dependency annotations. It is clear that a *subject* or an *object* takes a verb as its head. However, for the case of *preposition* vs. *noun*, or *complementizer* vs. *verb* in sub-clauses, there are various dependency annotation strategies. Such diversities of dependency annotation need corresponding changes for parsing algorithms. Since the same linguistic structures are presented differently with different annotation strategies. In section 3.1, we discuss for such changes required for the case of coordinate structures.

### 3.1 Diversity for Dependency Annotation of Coordinate Structures

There are various strategies for annotating dependencies of coordinate structures. These include, *coordinate conjunction as head* (CCH), and *coordinate conjunction as dependent* (CCD) strategies (McDonald and Nivre, 2007), and CCD can further be categorized into two cases illustrated in Figures 1(b) and 1(c).

Coordinate structures are characterized by symmetries between conjuncts. The symmetries are important clues for the disambiguation of coordinate structure. The different dependency annotation strategies for coordinate structures require different methods to model the symmetry sharing between conjuncts. For CCH-type coordinate structures, it is essential for the parser to model the pair of dependents shown in Figure 1(a). Existing span-based approaches (McDonald et al., 2006; Carreras, 2007; Koo and Collins, 2010) do not model such relations. That explains why the average performance for CCH-type coordinate structures is about 15% to 20% lower than that of CCD-type in the work of (McDonald et al., 2006), according to the analysis given in (McDonald and

Corpus	$2^{nd}$ -order <sup>1</sup> (McDonald et al., 2006)		Alg. 6	
	overall	coord.	overall	coord.
Chinese	88.29%	81.66%	89.41%	85.09%
Slovene	78.93%	58.79%	80.32%	74.06%

<sup>1</sup> The results of MSTParser(V0.2), which is available from <http://www.seas.upenn.edu/~strcltm/MSTParser/MSTParser.html>

Table 2: Results of experiment.

Nivre, 2007). Considering the frequent use of coordinate structures among sentences, for high performance parsing, it is crucial to select a parsing algorithm that can parse such structures well.

## 4 Experiments

We conduct our experiments on two data sets, the Chinese corpus of CoNLL’07 (Nivre et al., 2007a), and the Slovene corpus of CoNLL’06 (Buchholz and Marsi, 2006) shared task for *multilingual dependency parsing* track. Both corpora are of the CCH-type.

Table 2 lists the performance of two systems, i.e., the  $2^{nd}$ -order system of (McDonald et al., 2006) and the proposed system with Alg. 6 (Table 1) as the parse tree decoder. As the discussions given in the previous section, the  $2^{nd}$ -order gives low performance for coordinate structures compared to the overall parsing results. The proposed system gives better coordinate disambiguation by modeling the relation between dependents located on different-sides of the head.

## 5 Conclusion

In this paper, we categorize bottom-up dependency chart parsing algorithms into constituent-based and span-based algorithms according to the strings each identifies and parses. We further categorize algorithms in terms of the use of parsing histories for new dependency arc detection. We show that proper selection of the parsing algorithm helps to improve the overall parsing performance.

## Acknowledgments

This work was supported in part by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korean government (MEST No. 2011-0017960), and in part by the BK 21 Project in 2011.

## References

- Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Natural Language Learning (CoNLL-X)*, New York City, USA, June. Association for Computational Linguistics.
- Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic, June. Association for Computational Linguistics.
- Wenliang Chen, Jun’ichi Kazama, Yoshimasa Tsuruoka, and Kentaro Torisawa. 2010. Improving graph-based dependency parsing with decision history. In *Coling 2010: Posters*, pages 126–134, Beijing, China, August. Coling 2010 Organizing Committee.
- Jason M. Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.
- Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 340–345. Association for Computational Linguistics.
- Meixun Jin. 2011. Dependency chart parsing algorithms. *Technical Report, POSTECH*.
- Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July. Association for Computational Linguistics.
- Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.
- Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Conference on Natural Language Learning (CoNLL)*.
- Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June. Association for Computational Linguistics.
- Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gulsen Eryigit, Sandra Kubler, Svetoslav Marinov, and Erwin Marsi. 2007b. Malt-parser: a language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 2(13):95–135.
- Joakim Nivre. 2006. Inductive dependency parsing. *Text, Speech and Language Technology*, 34.
- Hiroyasu Yamada and Yuji Mastumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.
- Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time  $n^3$ . *Information and Control*, 12(4), 4(12):361–379.