Quasi-Second-Order Parsing for 1-Endpoint-Crossing, Pagenumber-2 Graphs

Junjie Cao, Sheng Huang, Weiwei Sun and **Xiaojun Wan**
Institute of Computer Science and Technology, Peking University
The MOE Key Laboratory of Computational Linguistics, Peking University
{junjie.cao,huangsheng,ws,wanxiaojun}@pku.edu.cn

Abstract

We propose a new Maximum Subgraph algorithm for first-order parsing to 1-endpoint-crossing, pagenumber-2 graphs. Our algorithm has two characteristics: (1) it separates the construction for noncrossing edges and crossing edges; (2) in a single construction step, whether to create a new arc is deterministic. These two characteristics make our algorithm relatively easy to be extended to incorporiate crossing-sensitive second-order features. We then introduce a new algorithm for quasi-second-order parsing. Experiments demonstrate that second-order features are helpful for Maximum Subgraph parsing.

1 Introduction

Previous work showed that treating semantic dependency parsing as the search for Maximum Subgraphs is not only elegant in theory but also effective in practice (Kuhlmann and Jonsson, 2015; Cao et al., 2017). In particular, our previous work showed that 1-endpoint-crossing, pagenumber-2 (1EC/P2) graphs are an appropriate graph class for modelling semantic dependency structures (Cao et al., 2017). On the one hand, it is highly expressive to cover a majority of semantic analysis. On the other hand, the corresponding Maximum Subgraph problem with an arc-factored disambiguation model can be solved in low-degree polynomial time.

Defining disambiguation models on wider contexts than individual bi-lexical dependencies improves various syntactic parsers in different architectures. This paper studies exact algorithms for second-order parsing for 1EC/P2 graphs. The existing algorithm, viz. our previous algorithm (GCHSW, hereafter), has two properties that make it hard to incorporate higher-order features in a principled way. First, GCHSW does not explicitly consider the construction of noncrossing arcs. We will show that incorporiating higher-order factors containing crossing arcs without increasing time and space complexity is extremely hard. An effective strategy is to only include higher-order factors containing only noncrossing arcs (Pitler, 2014). But this *crossing-sensitive* strategy is incompatible with GCHSW. Second, all existing higher-order parsing algorithms for projective trees, including (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010), require that which arcs are created in a construction step be deterministic. This design is also incompatible with GCHSW. In summary, it is not convenient to extend GCHSW to incorporate higher-order features while keeping the same time complexity.

In this paper, we introduce an alternative Maximum Subgraph algorithm for first-order parsing to 1EC/P2 graphs. while keeping the same time and space complexity to GCHSW, our new algorithm has two characteristics that make it relatively easy to be extended to incorporate crossing-sensitive, second-order features: (1) it separates the construction for noncrossing edges and possible crossing edges; (2) whether an edge is created is deterministic in each construction rule. We then introduce a new algorithm to perform second-order parsing. When all second-order scores are greater than or equal to 0, it exactly solves the corresponding optimization problem.

We implement a practical parser with a statistical disambiguation model and evaluate it on four data sets: those used in SemEval 2014 Task 8 (Oepen et al., 2014), and the dependency graphs extracted from CCGbank (Hockenmaier and Steedman, 2007). On all data sets, we find that our second-order parsing models are more accurate than the first-order baseline. If we do not use features derived from syntactic trees, we get an absolute unlabeled F-score improvement of 1.3 on average. When syntactic analysis is used, we get an improvement of 0.4 on average.

2 Preliminaries

2.1 Maximum Subgraph Parsing

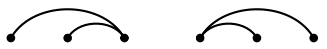
Semantic dependency parsing can be formulated as the search for Maximum Subgraph for graph class $\mathcal{G}$: Given a graph $G = (V, A)$, find a subset $A' \subseteq A$ with maximum total score such that the induced subgraph $G' = (V, A')$ belongs to $\mathcal{G}$. Formally, we have the following optimization problem:

$$\arg\max_{G^*\in\mathcal{G}(s,G)} \sum_{p \text{ in } G^*} s_{\text{part}}(s, p)$$

$\mathcal{G}(s, G)$ denotes the set of all graphs that belong to $\mathcal{G}$ and are compatible with s and G. G is usually a complete digraph. $s_{\text{part}}(s, p)$ evaluates the event that part p (from a candidate graph G^*) is good. We define the *order* of p according to the number of arcs it contains, in analogy with tree parsing in terminology. Previous work only discussed the first-order case:

$$\arg\max_{G^*\in\mathcal{G}(G)} \sum_{d\in\textsc{Arc}(G^*)} s_{\text{arc}}(d)$$

If $\mathcal{G}$ is the set of noncrossing or 1EC/P2 graphs, the above optimization problem can be solved in cubic-time (Kuhlmann and Jonsson, 2015) and quintic-time (Cao et al., 2017) respectively. Furthermore, ignoring one linguistically-rare structure in 1EC/P2 graphs descreases the complexity to $O(n^4)$. This paper is concerned with second-order parsing, with a special focus on the following factorizations:



And the objective function turns to be:

$$\sum_{d\in\textsc{Arc}(G^*)} s_{\text{arc}}(d) + \sum_{s\in\textsc{Sib}(G^*)} s_{\text{sib}}(s)$$

Sun et al. (2017) introduced a dynamic programming algorithm for second-order planar parsing. Their empirical evaluation showed that second-order features are effective to improve parsing accuracy. It is still unknown how to incorporate such features for 1EC/P2 parsing.

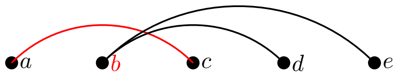


Figure 1: $e_{(a,c)}$'s crossing edges $e_{(b,d)}$ and $e_{(b,e)}$ share an endpoint b.

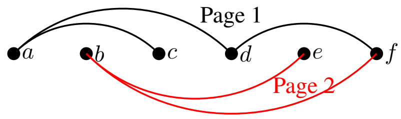


Figure 2: A pagenumber-2 graph. The upper and the lower figures represent two half-planes respectively.

2.2 1-Endpoint-Crossing, Pagenumber-2 Graphs

The formal description of the 1-endpoint-crossing property is adopted from (Pitler et al., 2013).

Definition 1. *Edges* e_1 *and* e_2 *cross if* e_1 *and* e_2 *have distinct endpoints and exactly one of the endpoints of* e_1 *lies between the endpoints of* e_2.

Definition 2. *A dependency graph is 1-Endpoint-Crossing if for any edge* e*, all edges that cross* e *share an endpoint* p *named pencil point.*

Given a sentence $s = w_0w_1 \cdots w_{n-1}$ of length n, the vertices, i.e. words, are indexed with integers, an arc from w_i to w_j as $a_{(i,j)}$, and the common endpoint, namely pencil point, of all edges crossed with $a_{(i,j)}$ or $a_{(j,i)}$ as $pt(i, j)$. We denote an edge as $e_{(i,j)}$, if we do not consider its direction. Figure 1 is an example.

Definition 3. *A pagenumber-*k *graph means it consists at most* k *half-planes, and arcs on each half-plane are noncrossing.*

These half-planes may be thought of as the pages of a book, with the vertex line corresponding to the books spine, and the embedding of a graph into such a structure is known as a book embedding. Figure 2 is an example.

(Pitler et al., 2013) proved that 1-endpoint-crossing trees are a subclass of graphs whose pagenumber is at most 2. In Cao et al. (2017), we studied graphs that are constrained to be both 1-endpoint-crossing and pagenumber-2. In this paper, we ignored a complex and linguistic-rare

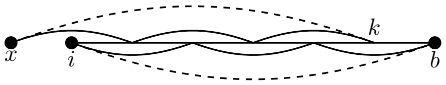


Figure 3: C structure has two crossing chains.

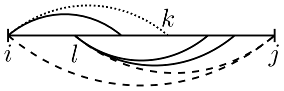


Figure 4: A prototype backbone of 1EC/P2 graphs. To decompose this structure, GCHSW focuses on $e_{(i,j)}$ and $e_{(l,j)}$, because these two edges can be optionally created without violation of both 1EC and P2 restrictions. Our algorithm focuses on the existence of $e_{(i,k)}$, and makes it the only edge that is constructed by applying a corresponding rule.

structure and studied a subset of 1EC/P2 graphs. The complex structure is named as C structures in our previous paper, and Figure 3 is the prototype of C structures. In this paper, we present new algorithms for finding optimal 1EC/P2, C-free graphs.

2.3 The GCHSWAlgorithm

Cao et al. (2017) designed a polynomial time Maximum Subgraph algorithm, viz. GCHSW, for 1EC/P2 graphs by exploring the following property: Every subgraph of a 1EC/P2 graph is also a 1EC/P2 graph. GCHSW defines a number of prototype backbones for decomposing a 1EC/P2 graph in a principled way. In each decomposition step, GCHSW focuses on the edges that can be created without violating either the 1EC nor P2 restriction. Sometimes, multiple edges can be created simultaneously in one single step. Figure 4 is an example.

There is an important difference between GCHSW and Eisner-style Maximum Spanning Tree algorithms (MST; Eisner, 1996; McDonald and Pereira, 2006; Koo and Collins, 2010). In each construction step, GCHSW allows multiple arcs to be constructed, but whether or not such arcs are added to the target graph depends on their arc-weights. If all arcs are assigned scores that are greater than 0, the output of our algorithm includes the most complicated 1EC/P2 graphs. For the higher-order MST algorithms, in a single construction step, it is clear whether adding a new arc, and which one. There is no local search. This deterministic strategy is also followed by Kuhlmann and Jonsson's Maximum Subgraph algorithm for noncrossing graphs. Higher-order MST models associate higher-order score functions with the construction of individual dependencies. Therefore the deterministic strategy is a prerequisite to incorporate higher-order features. The design of GCHSW is incompatible with this strategy.

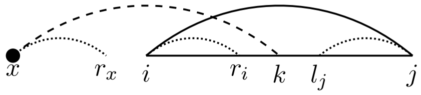


Figure 5: A typical structure of crossing arcs.

2.4 Challenge of Second-Order Decoding

It is very difficult to enumerate all high-order features for crossing arcs. Figure 5 illustrates the idea. There is a pair of corssing arcs, viz. $e_{(x,k)}$ and $e_{(i,j)}$. The key strategy to develop a dynamic programming algorithm to generate such crossing structure is to treat parts of this structures as intervals/spans together with an external vertex (Pitler et al., 2013; Cao et al., 2017). Without loss of generality, we assume $[i, j]$ makes up such an interval and x is the corresponding external vertex. When we consider $e_{(i,j)}$, its neighboring edges can be $e_{(i,r_i)}$ and $e_{(l_j,j)}$, and therefore we need to consider searching the best positions of both r_i and l_j. Because we have already taken into account three vertices, viz. x, i and j, the two new positions increase the time complexity to be at least quintic.

Now consider $e_{(x,k)}$. When we decompose the whole graph into inverval $[i, j]$ plus x and remaining part, we will factor out $e_{(x,k)}$ in a successive decomposition for resolving $[i, j]$ plus x. We cannot capture the second features associated to $e_{(x,k)}$ and $e_{(x,r_x)}$, because they are in different intervals, and when these intervals are combined, we have already hidden the position information of k. Explicitly encoding k increases the time complexity to be at least quintic too.

Pitler (2014) showed that it is still possible to build accurate tree parsers by considering only higher-order features of noncrossing arcs. This is in part because only a tiny fraction of neighboring arcs involve crossing arcs. However, this strategy is not easy to by applied to GCHSW, because GCHSW does not explicitly analyze sub-graphs of noncrossing arcs.

3 A New Maximum Subgraph Algorithm

Based on the discussion of Section 2.3 and 2.4, we can see that it is not easy to extend the existing algorithm, viz. GCHSW, to handle second-order features. In this paper, we propose an alternative first-order dynamic programming algorithm. Because ignoring one linguistically-rare structure associated with the **C** problem in GCHSW descreases the complexity, we exclude this structure in our algorithm. Formally, we introduce a new algorithm

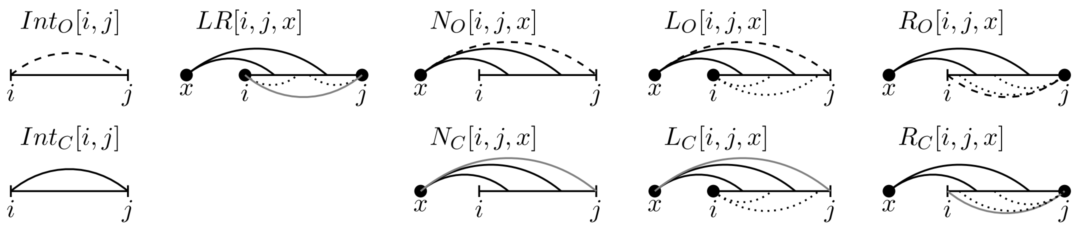


Figure 6: Graphical representations of sub-problems. Gray curves mean the corresponding edge in this sub-problem, but should be included in the final generated graph.

$$Int_O(i,j) \leftarrow \max \begin{cases} Int_O(i+1,j) \\ Int_C(i,j) \\ Int_C(i,k) + Int_O(k,j) \\ R_C(i,k,x) + Int_O(k,x) + L_O(x,j,k) + \mathrm{s}_{\mathrm{arc}}(i,k) \\ LR(i,k,x) + Int_O(k,x) + Int_O(x,j,k) + \mathrm{s}_{\mathrm{arc}}(i,k) \\ Int_O[i,x] + L_C[x,k,i] + N_O[k,j,x] + \mathrm{s}_{\mathrm{arc}}(i,k) \\ R_O[i,x,k] + Int_O[x,k] + L_O[k,j,x] + \mathrm{s}_{\mathrm{arc}}(i,k) \end{cases}$$

$$Int_C(i,j) \leftarrow \mathrm{s}_{\mathrm{arc}}(i,j) + \max \begin{cases} Int_O(i+1,j) \\ Int_C(i,k) + Int_O(k,j) \\ R_C(i,k,x) + Int_O(k,x) + L_O(x,j,k) + \mathrm{s}_{\mathrm{arc}}(i,k) \\ LR(i,k,x) + Int_O(k,x) + Int_O(x,j,k) + \mathrm{s}_{\mathrm{arc}}(i,k) \\ Int_O[i,x] + L_C[x,k,i] + N_O[k,j,x] + \mathrm{s}_{\mathrm{arc}}(i,k) \\ R_O[i,x,k] + Int_O[x,k] + L_O[k,j,x] + \mathrm{s}_{\mathrm{arc}}(i,k) \end{cases}$$

$$N_O(i,j,x) \leftarrow \max \begin{cases} Int_O(i,j) \\ N_C(i,j,x) + \mathrm{s}_{\mathrm{arc}}(x,j) \\ N_C(i,k,x) + Int_O(k,j) + \mathrm{s}_{\mathrm{arc}}(x,k) \end{cases}$$

$$L_O(i,j,x) \leftarrow \max \begin{cases} Int_O(i,j) \\ L_C(i,j,x) + \mathrm{s}_{\mathrm{arc}}(x,j) \\ L_C(i,k,x) + N_O(k,j) + \mathrm{s}_{\mathrm{arc}}(x,k) \\ Int_O(i,k,x) + L_O(k,j) + \mathrm{s}_{\mathrm{arc}}(x,k) \end{cases}$$

$$R_O(i,j,x) \leftarrow \max \begin{cases} Int_O(i,j) \\ R_C(i,j,x) + \mathrm{s}_{\mathrm{arc}}(x,j) \\ N_C(i,k,j) + R_O(k,j,x) + \mathrm{s}_{\mathrm{arc}}(x,k) \\ R_O(i,k,x) + Int_O(k,j) + \mathrm{s}_{\mathrm{arc}}(x,k) \end{cases}$$

$$N_C(i,j,x) \leftarrow \max \begin{cases} Int_O(i,j) \\ N_C(i,k,x) + Int_O(k,j) + \mathrm{s}_{\mathrm{arc}}(x,k) \end{cases}$$

$$LR(i,j,x) \leftarrow \max \begin{cases} L_O(i,k,x) + R_O(k,j,x) \end{cases}$$

$$L_C(i,j,x) \leftarrow \max \begin{cases} Int_O(i,j) \\ L_C(i,j,x) + \mathrm{s}_{\mathrm{arc}}(x,j) \\ L_C(i,k,x) + N_O(k,j,i) + \mathrm{s}_{\mathrm{arc}}(x,k) \\ Int_O(i,k) + L_O(k,j,i) + \mathrm{s}_{\mathrm{arc}}(x,k) \end{cases}$$

$$R_C(i,j,x) \leftarrow \max \begin{cases} N_C(i,k,j) + R_O(k,j,x) + \mathrm{s}_{\mathrm{arc}}(x,k) \\ R_O(i,k,x) + Int_O(k,j) + \mathrm{s}_{\mathrm{arc}}(x,k) \end{cases}$$

Figure 7: A dynamic program to find optimal 1EC/P2, **C**-free graphs with arc-factored weights.

to solve the following optimization problem:

$$\arg \max_{G^* \in \mathcal{G}(G)} \sum_{d \in \textsc{Arc}(G^*)} \mathrm{s}_{\mathrm{arc}}(d)$$

where $\mathcal{G}$ means 1EC/P2, **C**-free graphs. Our algorithm has the same time and space complexity to the degenerated version of GCHSW. We represent our algorithm using undirected graphs.

3.1 Sub-problems

Following GCHSW, we consider five sub-problems when we construct a maximum dependency graph on a given interval $[i,k]$. Though the sub-problems introduced by GCHSW and us handle similar structures, their definitions are quite different. The sub-problems are explained as follows:

Int $Int[i,j]$ represents an interval from i to j inclusively. And there is no edge $e_{(i',j')}$ such that $i' \in [i,j]$ and $j' \notin [i,j]$. We distinguish two sub-types for **Int**. $Int_O[i,j]$ may or may not contain $e_{(i,j)}$, while $Int_C[i,j]$ contains $e_{(i,j)}$.

LR $LR[i,j,x]$ represents an interval from i to j inclusively and an external vertex x. $\forall p \in [i,j], pt(x,p) = i$ or j. $LR[i,j,x]$ implies the existence of $e_{(i,j)}$, but does not contain $e_{(i,j)}$. When $LR[i,j,x]$ is combined with other DP sub-structures, $e_{(i,j)}$ is immediately created. $LR[i,j,x]$ disallows neither $e_{(x,i)}$ nor $e_{(x,j)}$.

N $N[i,j,x]$ represents an interval from i to j inclusively and an external vertex x. $\forall p \in [i,j], pt(x,p) \notin [i,j]$. $N[i,j,x]$ could contain $e_{(i,j)}$ but disallows $e_{(x,i)}$. We distinguish two sub-types. $N_O[i,j,x]$ may or may not contain $e_{(x,j)}$. $N_C[i,j,x]$ implies the existence of but does not contain $e_{(x,j)}$. When $N[i,j,x]$ is combined with others, $e_{(x,j)}$ is immediately created.

L $L[i,j,x]$ represents an interval from i to j inclusively as well as an external vertex x. $\forall p \in [i,j], pt(x,p) = i$. $L[i,j,x]$ could contain $e_{(i,j)}$ but disallows $e_{(x,i)}$. We distinguish sub-two types for **L**. $L_O[i,j,x]$ may or may not contain $e_{(x,j)}$. $L_C[i,j,x]$ implies the existence of but does not contain $e_{(x,j)}$. When it is combined with others, $e_{(x,j)}$ is immediately created.

R $R[i,j,x]$ represents an interval from i to j inclusively as well as an external vertex x. $\forall p \in [i,j], pt(x,p) = j$. $R[i,j,x]$ disallows $e_{(x,j)}$ and $e_{(x,i)}$. We distinguish two sub-types for **R**. $R_O[i,j,x]$ may or may not contain $e_{(i,j)}$. $R_C[i,j,x]$ implies the existence of but does not contain $e_{(i,j)}$. When it is combined with others, $e_{(i,j)}$ is immediately created.

3.2 Decomposing Sub-problems

Figure 7 gives a sketch of our dynamic programming algorithm. We give a detailed illustration for **Int**, a rough idea for **L** and **LR**, and omit other sub-problems. More details about the whole algorithm can be found in the supplementary note.

3.2.1 Decomposing an Int Sub-problem

Consider $Int_O[i,j]$ and $Int_C[i,j]$ sub-problem. Because the decomposition for $Int_C[i,j]$ is very similar to $Int_O[i,j]$ and needs to be modified by our second-order parsing algorithm, we only show the decomposition of $Int_C[i,j]$. Assume that $k(k \in (i,j))$ is the **farthest** vertex that is adjacent to i, and $x = pt(i,k)$. If there is no such k (i.e. there no arc from i to some other node in this interval), then we denote k as $\emptyset$. So it is to x. We illustrate different cases as following and give a graphical representation in Figure 8.

Case a: $k = \emptyset$**.** We can directly consider interval $[i+1,j]$. Because there is no edge from i to any node in $[i+1,j]$, $[i+1,j]$ is an $\textbf{Int}_\textbf{O}$.

Case b: $x = \emptyset$**.** $x = \emptyset$ means that $e_{(i,k)}$ does not cross other arcs. So $[i,k]$ and $[k,j]$ are **Int**.

Case c: $x \in (k,j]$**.** $e_{(i,k)}$ is taken as a possible crossing edge. k and x divide the interval $[i,j]$ into three parts: $[i,k]$, $[k,x]$, $[x,j]$. Because x may be j, interval $[x,j]$ may only contain j and become an empty interval. We define x' as the pencil point of all edges from (i,k) to x, and distinguish two sub-problems as follows.

c.1 Assume that there exists an edge from k to some node r in $(x,j]$, so x' can only be k and pencil point of edges from k to $(x,j]$ is x. Thus interval $[i,k,x]$ is an **R**. Due to the existence of $e_{(i,k)}$, its sub-type is $\textbf{R}_\textbf{C}$. The $e_{(i,k)}$ is created in this construction and thus not contained by $R_C[i,k,x]$. An edge from within $[k,x]$ to outside violates the 1EC restriction, so $[k,x]$ is an **Int**. Since x is endpoint of edge from k to $[x,r]$, interval $[k,j]$ is an $\boldsymbol{L_O}$ with external vertex k.

c.2 We assume no edge from k to any node in $[x,j]$, x' thus can be i or k. As a result, $[x,j]$ is an **Int** and $[i,k,x]$ is an **LR**.

Case d: $x \in (i,k)$**.**

d.1 Assume that there exist edges from i to (x,k), so the pencil point of edges from x to $(k,j]$ is i. Therefore $[k,j]$ is an **N**. Because x is pencil point of edges from i to $(x,k]$, $[x,k]$ is an **L**. Furthmore, it is an $\textbf{L}_\textbf{C}$ because we generate $e_{(i,k)}$ in this step. It is obvious that $[i,x]$ is an **Int**.

d.2 Assume that there exists edges from k to (i,x), and the pencil point of edges from x to $(k,j]$ is thus k. Similar to the above analysis, we reach $R_O[i,x,k] + Int_O[x,k] + L_O[k,j,x] + e_{(i,k)} + e_{(i,j)}$.

For $Int_O[i,j]$, because there may be $e_{(i,j)}$, we add one more rule: $Int_O[i,j] = Int_C[i,j]$. And we do not need to create $e_{(i,j)}$ in all cases.

3.2.2 Decomposing an L Sub-problem

Without loss of generality, we show the decomposition of $L_O[i,j,x]$ as follows. For $L_C[i,j,x]$, we ignore Case b but follow the others.

Case a. If there is no more edge from x to $(i,j]$, then it will degenerate to $Int_O[i,j]$.

Case b. If there exists $e_{(x,j)}$, then it will degenerate to $L_C[i,j,x] + e_{(x,j)}$.

Case c. Assume that there are edges from x to (i,j) and $e_{(x,k)}$ is the farthest one. It divides $[i,j]$ into $[i,k]$ and $[k,j]$.

c.1 If there is an edge from x to (i,k), $[i,k]$ and $[k,j]$ are $L_C[i,k,x]$ and $N_O[k,j,i]$.

c.2 If there is no edge from x to (i,k), $[i,k]$ and $[k,j]$ are $Int_O[i,k]$ and $L_O[k,j,i]$.

Figure 8 is a graphical representation.

3.2.3 Decomposing an LR Sub-problem

$LR[i,j,x]$ means i or j is the pencil point of edges from x to (i,j). We show the decomposition of $LR[i,j,x]$ as follows:

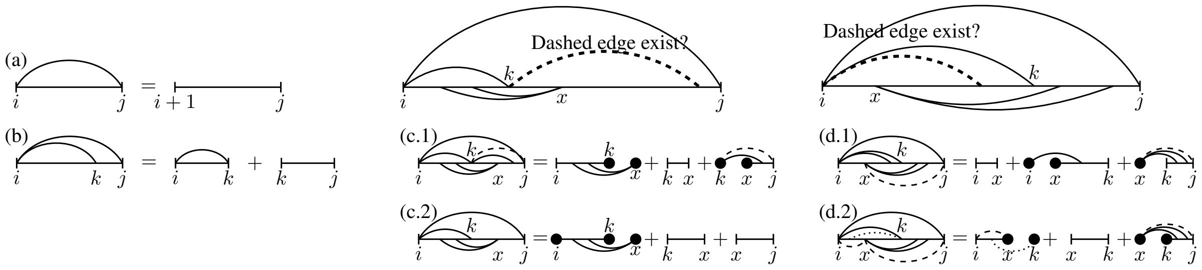


Figure 8: Decomposition for $Int_C[i, j]$ in the first-order parsing algorithm. $pt(i, k) = x$.

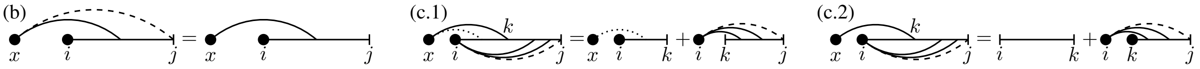


Figure 9: Decomposition for $L_O[i, j, x]$.

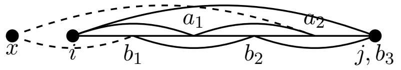


Figure 10: $b_3 = j$, Not both $e_{(x,b_1)}$ and $e_{(x,a_2)}$ exist.

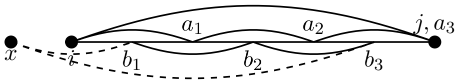


Figure 11: $a_3 = j$. Both $e_{(x,b_1)}$ and $e_{(x,b_3)}$ exist.

Case a. If there is a vertex k within (i, j), which divides $[i, j]$ into $[i, k]$ and $[k, j]$. And it guarantees no edge from $[i, k)$ to $(k, j]$. i is the pencil point of edges from x to $(i, k]$ because no edge from j to (i, k) can cross these edges. Similarly j has to be the pencil point of edges from x to (k, j). Obviously, $[i, k]$ is an L_O and $[k, j]$ is an R_O with external x. Thus the problem is decomposed as $L_O[i, k, x] + R_O[k, j, x]$.

Case b. If there is no such vertex k, there must be edges from $[i, k')$ to $(k', j]$ for every k' in (i, j) without considering $e_{(i,j)}$. For $i + 1$, we assume $e_{(i,a_1)}$ is the farthest edge that goes from i. For a_1, we assume $e_{(b_1,b_2)}$ is the farthest edge from b_1 where b_1 is in (i, a_1) and b_2 is in (a_1, j). For b_2, we assume $e_{(a_1,a_3)}$ is the farthest edge from a_1 where a_3 is in (b_2, j) and a_1 is the pencil point. We then get the series $\{a_1, a_2, a_3...a_n\}$ and $\{b_1, b_2...b_m\}$ which guarantees $b_i < a_i$, $a_i < b_{i+1}$ and $max(a_n, b_m) = j$.

If $b_m = j$, we will get a graph like Figure 10. If $e_{(x,b_1)}$ exists, this LR subproblem degenerates to an L subproblem. If $e_{(x,a_n)}$ exists, this subproblem degenerates to an R subproblem.

If $a_m = j$, we will get a graph like Figure 11. If there exists only $e_{(x,b_1)}$ or $e_{(x,b_m)}$, we can solve it like $b_m = j$. If both exist, this is a typical C-structure like Figure 3 and we cannot get it through other decompostion.

The above discussion gives the rough idea of the correctness of the following conclusion.

Theorem 1. *Our new algorithm is sound and complete with respect to* 1EC/P2*, C-free graphs.*

3.3 Spurious Ambiguity

An **LR**, **L**, **R** or **N** sub-problem allows to build crossing arcs, but does not necessarily create crossing arcs. For example, $L_C[i, j, x]$ allows $e_{(i,j)}$ to cross with $e_{(x,y)}$ $(y \in (i, j))$. Because every subgraph of a 1EC/P2 graph is also a 1EC/P2 graph, we allow an $L_C[i, j, x]$ to be directly degenerated to $I_O[i, j]$. In this way, we can make sure that all subgraphs can be constructed by our algorithm. Figure 12 shows the rough idea. To generate the same graph, we have different derivations. The spurious ambiguity in our algorithm does not affect the correctness of first-order parsing, because scores are assigned to individual dependencies, rather than derivation processes. There is no need to distinguish one special derivation here.

4 Quasi-Second-Order Extension

We propose a second-order extension of our new algorithm. We focus on factorizations introduced in Section 2.1. Especially, the two arcs in a factor should not cross other arcs. Formally, we introduce a new algorithm to solve the following optimization problem:

$$\arg\max_{G^*} \sum_{d\in\textsc{Arc}(G^*)} s_{\text{arc}}(d) + \sum_{s\in\textsc{Sib}(G^*)} \max(s_{\text{sib}}(s), 0)$$

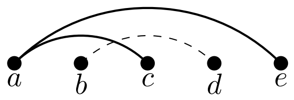


Figure 12: Illustration of spurious ambiguity. The two solid curves represent two arcs in the target graph, but not the dashed one. Excluding crossing edges leads to the first derivation: $Int_C[a,e] \Rightarrow e_{(a,e)} + Int_C[a,c] + Int_O[c,e] + e_{(a,c)}$. Assuming that a pair of crossing arcs may exist yields another derivation: $Int_C[a,e] \Rightarrow e_{(a,e)} + LR[a,c,d] + Int_O[k,d] + L_O[d,e,c] + e_{(a,c)}$; Then $LR[a,c,d] \Rightarrow L_O[a,b,d] + R_O[b,c,d] \Rightarrow Int_O[a,b] + Int_O[b,c]$.

In the first-order algorithm, all noncrossing edges can be constructed as the frontier edge of an $\mathbf{Int_C}$. So we can develop an exact decoding algorithm by modifying the composition for $\mathbf{Int_C}$ while keeping intact the decomposition for **LR**, **N**, **L**, **R**.

4.1 New Decomposition for $Int_\mathbf{C}$

In order to capture the second-order features from noncrossing neighbors, we need to find the rightmost node adjacent to i, denoted as r_i, and the leftmost node adjacent to j, denoted as l_j,while $i < r_i \leq l_j < j$. To do this, we split $Int_C[i,j]$ into at most three parts to capture the sibling factors. Denote the score of adjacent edges $e_{(i,j_1)}$ and $e_{(i,j_2)}$ as $\mathrm{s}_2(i,j_1,j_2)$. When j is the inner most node adjacent to i, we denote the score as $\mathrm{s}_2(i,\emptyset,j)$. We give a sketch of the decomposition in Figure 14 and a graphical representation in Figure 13. The following is a rough illustration.

Case a: $r_i = \emptyset$**.** We further distinguish three sub-problems:

a.1 If $l_j = \emptyset$ too, both sides are the inner most second-order factor.

a.2 There is a crossing arc from j. This case is handled in the same way as the first-order algorithm.

a.3 $l_j \neq \emptyset$. We introduce a new decomposition rule.

Case b: There is a crossing arc from i**.**

b.1 $l_j = \emptyset$. Similar case to (a.2).

b.2 There is a crossing arc from j. Similar case to (a.2).

b.3 There is a noncrossing arc from j. We introduce a new rule to calculate $\textsc{Sib}(j,l_j,i)$.

Case c: There is a noncrossing arc from i**.**

c.1 $l_j = \emptyset$. Similar to (a.3).

c.2 There is a crossing arc from j. Similar to (b.3).

c.3 There is a noncrossing arc from j too. We introduce a new rule to calculate $\textsc{Sib}(i,r_i,j)$ and $\textsc{Sib}(j,l_j,i)$.

4.2 Complexity

The complexity of both first- and second-order algorithms can be analyzed in the same way. The sub-problem **Int** is of size $O(n^2)$, with a calculating time of order $O(n^2)$ at most. For sub-problems **L**, **R**, **LR**, and **N**, each has $O(n^3)$ elements, with a unit calculating time $O(n)$. Therefore both algorithms run in time of $O(n^4)$ with a space requirement of $O(n^3)$.

4.3 Discussion

A traditional second-order model takes as the objective function $\sum_{s \in \textsc{Sib}(G^*)} \mathrm{s}_{\text{sib}}(s)$. Our model instead tries to optimize $\sum_{s \in \textsc{Sib}(G^*)} \max(\mathrm{s}_{\text{sib}}(s), 0)$. This model is somehow inadequate given that the second-order score function cannot penalize a *bad* factor. When a negative score is assigned to a second-order factor, it will be taken as 0 by our algorithm.

This inadequacy is due to the spurious ambiguity problem that is illustrated in Section 3.3. Take the two derivations in Figure 12 for example. The derivation that starts from $Int_C[a,e] \Rightarrow Int_C[a,c] + Int_O[c,e]$ incorporates the second-order score $\mathrm{s}_{\text{sib}}(a,c,e)$. This is different when we consider the derivation that starts from $Int_C[a,e] \Rightarrow LR[a,c,d] + Int_O[k,d] + L_O[d,e,c]$. Because we assume temporarily that $e_{(a,c)}$ crosses others, we do not consider $\mathrm{s}_{\text{sib}}(a,c,e)$. We can see from this example that second-order scores not only depend on the derived graphs but also sensitive to the derivation processes.

If a second-order score is greater than 0, our algorithm selects the derivation that takes it into account since it increases the total score. If a second-order score is negative, our algorithm avoids including it by selecting other paths. In other words, our algorithm treats this score as 0.

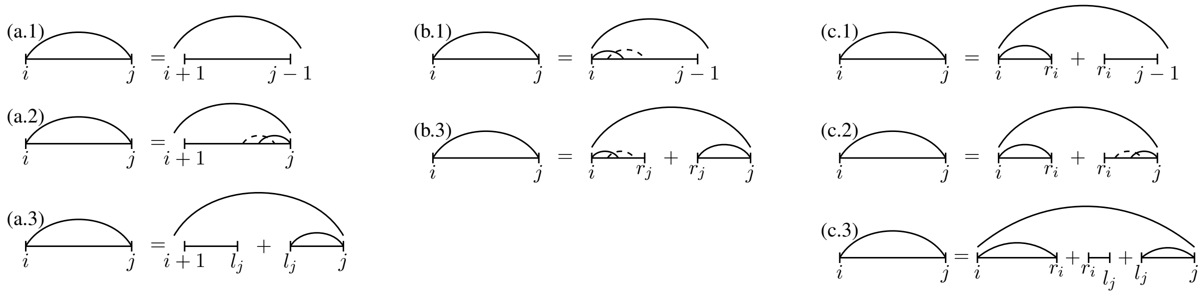


Figure 13: Decomposition for $Int_C[i, j]$ in the second-order parsing algorithm.

$$
Int_C(i,j) \leftarrow \mathrm{s}_{\text{arc}}(i,j) + \max
\begin{cases}
Int_O(i+1, j-1) + \mathrm{s}_{\text{sib}}(i, \emptyset, j) + \mathrm{s}_{\text{sib}}(j, \emptyset, i) \\
Int_O(i+1, j) + \mathrm{s}_{\text{sib}}(i, \emptyset, j) \\
Int_O(i+1, l_j) + Int_C(l_j, j) + \mathrm{s}_{\text{sib}}(i, \emptyset, j) + \\
\qquad\qquad\qquad\qquad \mathrm{s}_{\text{sib}}(j, l_j, i) \\
Int_O(i, j-1) + \mathrm{s}_{\text{sib}}(j, \emptyset, i) \\
Int_O(i, l_j) + Int_C(l_j, j) + \mathrm{s}_{\text{sib}}(j, l_j, i) \\
Int_C(i, r_i) + Int_O[r_i, j-1] + \mathrm{s}_{\text{sib}}(i, r_i, j) + \\
\qquad\qquad\qquad\qquad \mathrm{s}_{\text{sib}}(j, \emptyset, i) \\
Int_C(i, r_i) + Int_O[r_i, j] + \mathrm{s}_{\text{sib}}(i, r_i, j) \\
Int_C(i, r_i) + Int_O[r_i, l_j] + Int_C(l_j, j) + \\
\qquad\qquad \mathrm{s}_{\text{sib}}(i, r_i, j) + \mathrm{s}_{\text{sib}}(j, l_j, i) \\
R_C(i, k, x) + Int_O(k, x) + L_O(x, j, k) + e_{(i,k)} \\
LR(i, k, x) + Int_O(k, x) + Int_O(x, j, k) + e_{(i,k)} \\
Int_O[i, x] + L_C[x, k, i] + N_O[k, j, x] + e_{(i,k)} \\
R_O[i, x, k] + Int_O[x, k] + L_O[k, j, x] + e_{(i,k)}
\end{cases}
$$

Figure 14: Decomposition for $Int_C[i, j, x]$.

5 Practical Parsing

5.1 Derivation-Sensitive Training

We extend our quartic-time parsing algorithm into a practical parser. In the context of data-driven parsing, this requires an extra disambiguation model. As with many other parsers, we employ a global linear model. Following Zhang et al. (2016)'s experience, we define rich features extracted from word, POS-tags and pseudo trees. To estimate parameters, we utilize the averaged perceptron algorithm (Collins, 2002).

Our training proceudre is sensitive to derivation rather then derived graphs. For each sentence, we first apply our algorithm to find the optimal prediction derivation. The we collect all first- and second-order factors from this derivation to update parameters. To train a first-order model, because our algorithm includes all factors, viz. dependencies, there is no difference between our derivation-based method and a traditional derived structure-based method. For the second-order model, our method increases the second-order scores somehow.

5.2 Data and Preprocessing

We evaluate first- and second-order models on four representative data sets: CCGBank (Hockenmaier and Steedman, 2007), DeepBank (Flickinger et al., 2012), Enju HPSGBank (Miyao et al., 2005) and Prague Dependency TreeBank (Hajic et al., 2012). We use "standard" training, validation, and test splits to facilitate comparisons.

- Following previous experimental setup for English CCG parsing, we use section 02-21 as training data, section 00 as the development data, and section 23 for testing.
- The DeepBank, Enju HPSGBank and Prague Dependency TreeBank are from SemEval 2014 Task 8 (Oepen et al., 2014), and the data splitting policy follows the shared task.

Experiments for CCG-grounded analysis were performed using automatically assigned POS-tags that are generated by a symbol-refined HMM tagger (Huang et al., 2010). Experiments for the other three data sets used POS-tags provided by the shared task. We also use features extracted from pseudo trees. We utilize the Mate parser (Bohnet, 2010) to generate pseudo trees. All experimental results consider directed dependencies in a standard way. We report Unlabeled Precision (UP), Recall (UR) and F-score (UF), which are calculated using the official evaluation tool provided by SDP2014 shared task.

5.3 Accuracy

Table 1 lists the accuracy of our system. The output of our parser was evaluated against each dependency in the corpus. We can see that the first-order parser obtains a considerably good accuracy, with rich syntactic features. Furthermore, we can see that the introduction of higher-order features improves parsing substantially for all data sets, as

		DeepBank			EnjuBank			CCGBank			PCEDT		
Tree		UP	UR	UF	UP	UR	UF	UP	UR	UF	UP	UR	UF
No	1or	89.43	83.03	86.11	90.10	87.10	88.58	91.63	88.07	89.82	88.13	81.53	84.70
	2or	89.23	85.98	87.57	90.88	89.90	90.39	91.96	89.54	90.74	88.56	84.57	86.52
Syn	1or	91.24	87.14	89.14	92.72	90.96	91.83	94.28	91.79	93.02	91.53	86.95	89.18
	2or	90.93	88.79	89.85	92.73	92.11	92.42	93.99	92.27	93.13	91.02	88.20	89.59

Table 1: Parsing accuracy evaluated on the development sets.

		DeepBank			EnjuBank			CCGBank			PCEDT		
Tree		UP	UR	UF	UP	UR	UF	UP	UR	UF	UP	UR	UF
No	1or	88.87	82.50	85.57	90.12	86.76	88.41	91.95	88.29	90.08	86.87	80.45	83.54
	2or	88.77	85.61	87.16	91.06	89.50	90.27	92.25	89.80	91.01	87.07	83.45	85.22
Syn	1or	90.68	86.57	88.58	92.82	90.62	91.71	94.32	91.88	93.09	90.11	85.83	87.97
	2or	90.13	88.21	89.16	92.84	91.50	92.17	94.09	92.27	93.17	89.73	87.13	88.41
SJW (2or)		89.99	87.77	88.87	92.87	92.04	92.46	93.45	92.51	92.98	89.58	87.73	88.65

Table 2: Parsing accuracy evaluated on the test sets. "SJW" denotes the book embedding parser introduced in (Sun et al., 2017).

expected. When syntactic trees are utilized, the improvement is smaller but still significant on the three SemEval data sets.

Table 2 lists the parsing results on the test data together with the result obtained by Sun et al. (SJW; 2017)'s system. The building architectures of both systems are comparable.

1. Both systems have explicit control of the output structures. While Sun et al.'s system constrain the output graph to be P2 only, our system adds an additional 1EC restriction.

2. Their system's second-order features also includes both-side neighboring features.

3. Their system uses beam search and dual decomposition and therefore approximate, while ours perform exact decoding.

We can see that while our purely Maximum Subgraph parser obtains better results on DeepBank and CCGBank; while the book embedding parser is better on the other two data sets.

5.4 Analysis

Our algorithm is sensitive to the derivation process and may exclude a couple of negative second-order scores by selecting misleading derivations. Nevertheless, our algorithm works in an exact way to include all positive second-order scores. Table 3 shows the coverage of all second-order factors. On average, 99.67% second-order factors are calculated by our algorithm. This relatively satisfactory coverage suggests that our algorithm is very effective to include second-order features. Only a very small portion is dropped.

	DeepBank	EnjuBank	CCGBank	PCEDT
No	99.08	99.52	99.67	98.32
Syn	99.77	99.69	99.88	99.33

Table 3: Coverage of second-order factors on the developmenet data.

6 Conclusion

This paper proposed two exact, graph-based algorithms for 1EC/P2 parsing with first-order and quasi-second-order scores. The resulting parser has the same asymptotic run time as Cao et al. (2017)'s algorithm. An exploration of other factorizations that facilitate semantic dependency parsing may be an interesting avenue for future work. Recent work has investigated faster decoding for higher-order graph-based projective parsing e.g. vine pruning (Rush and Petrov, 2012) and cube pruning (Zhang and McDonald, 2012). It would be interesting to extend these lines of work to decrease the complexity of our quartic algorithm.

Acknowledgments

This work was supported by 863 Program of China (2015AA015403), NSFC (61331011), and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology). Weiwei Sun is the corresponding author.

References

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Coling 2010 Organizing Committee, Beijing, China, pages 89–97. http://www.aclweb.org/anthology/C10-1011.

Junjie Cao, Sheng Huang, Weiwei Sun, and Xiaojun Wan. 2017. Parsing to 1-endpoint-crossing, pagenumber-2 graphs. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *In Proc. EMNLP-CoNLL*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1–8. https://doi.org/10.3115/1118693.1118694.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: an exploration. In *Proceedings of the 16th conference on Computational linguistics - Volume 1*. Association for Computational Linguistics, Stroudsburg, PA, USA, pages 340–345.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank: A dynamically annotated treebank of the wall street journal. In *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories*. pages 85–96.

Jan Hajic, Eva Hajicová, Jarmila Panevová, Petr Sgall, Ondej Bojar, Silvie Cinková, Eva Fucíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jirí Semecký, Jana Sindlerová, Jan Stepánek, Josef Toman, Zdenka Uresová, and Zdenek Zabokrtský. 2012. Announcing prague czech-english dependency treebank 2.0. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*. Istanbul, Turkey.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics* 33(3):355–396.

Zhongqiang Huang, Mary Harper, and Slav Petrov. 2010. Self-training with products of latent variable grammars. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Cambridge, MA, pages 12–22. http://www.aclweb.org/anthology/D10-1002.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Uppsala, Sweden, pages 1–11. http://www.aclweb.org/anthology/P10-1001.

Marco Kuhlmann and Peter Jonsson. 2015. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics* 3:559–570.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006))*. volume 6, pages 81–88.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. 2005. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the penn treebank. In *IJCNLP*. pages 684–693.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Association for Computational Linguistics and Dublin City University, Dublin, Ireland, pages 63–72. http://www.aclweb.org/anthology/S14-2008.

Emily Pitler. 2014. A crossing-sensitive third-order factorization for dependency parsing. *TACL* 2:41–54. http://www.transacl.org/wp-content/uploads/2014/02/39.pdf.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *TACL* 1:13–24. http://www.transacl.org/wp-content/uploads/2013/03/paper13.pdf.

Alexander Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Montréal, Canada, pages 498–507. http://www.aclweb.org/anthology/N12-1054.

Weiwei Sun, Junjie Cao, and Xiaojun Wan. 2017. Semantic dependency parsing via book embedding. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, Jeju Island, Korea, pages 320–331. http://www.aclweb.org/anthology/D12-1030.

Xun Zhang, Yantao Du, Weiwei Sun, and Xiaojun Wan. 2016. Transition-based parsing for deep dependency structures. *Computational Linguistics* 42(3):353–389. http://aclweb.org/anthology/J16-3001.