Semantic Dependency Parsing via Book Embedding

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

Abstract

We model a dependency graph as a book, a particular kind of topological space, for semantic dependency parsing. The spine of the book is made up of a sequence of words, and each page contains a subset of noncrossing arcs. To build a semantic graph for a given sentence, we design new Maximum Subgraph algorithms to generate noncrossing graphs on each page, and a Lagrangian Relaxation-based algorithm to combine pages into a book. Experiments demonstrate the effectiveness of the book embedding framework across a wide range of conditions. Our parser obtains comparable results with a state-of-the-art transition-based parser.

1 Introduction

Dependency analysis provides a lightweight and effective way to encode syntactic and semantic information of natural language sentences. One of its branches, syntactic dependency parsing (Kübler et al., 2009) has been an extremely active research area, with high-performance parsers being built and applied for practical use of NLP. Semantic dependency parsing, however, has only been addressed in the literature recently (Oepen et al., 2014, 2015; Du et al., 2015; Zhang et al., 2016; Cao et al., 2017).

Semantic dependency parsing employs a graph-structured semantic representation. On the one hand, it is flexible enough to provide analysis for various semantic phenomena (Ivanova et al., 2012). This very flexibility, on the other hand, brings along new challenges for designing parsing algorithms. For graph-based parsing, no previously defined Maximum Subgraph algorithm has simultaneously a high coverage and a polynomial complexity to low degrees. For transition-based parsing, no principled decoding algorithms, e.g. dynamic programming (DP), has been developed for existing transition systems.

In this paper, we borrow the idea of book embedding from graph theory, and propose a novel framework to build parsers for flexible dependency representations. In graph theory, a *book* is a kind of topological space that consists of a spine and a collection of one or more half-planes. In our "book model" of semantic dependency graph, the spine is made up of a sequence of words, and each half-plane contains a subset of dependency arcs. In particular, the arcs on each page compose a noncrossing dependency graph, a.k.a. planar graph. Though a dependency graph in general is very flexible, its subgraph on each page is rather regular. Under the new perspective, semantic dependency parsing can be cast as a two-step task: Each page is first analyzed separately, and then all the pages are bound coherently.

Our work is motivated by the extant low-degree polynomial time algorithm for first-order Maximum Subgraph parsing for noncrossing dependency graphs (Kuhlmann and Jonsson, 2015). We enhance existing work with new exact second- and approximate higher-order algorithms. Our algorithms facilitate building with high accuracy the partial semantic dependency graphs on each page. To produce a full semantic analysis, we also need to integrate partial graphs on all pages into one coherent book. To this end, we formulate the problem as a combinatorial optimization problem, and propose a Lagrangian Relaxation-based algorithm for solutions.

We implement a practical parser in the new framework with a statistical disambiguation model. We evaluate this parser on four data sets: those used in SemEval 2014 Task 8 (Oepen et al., 2014), and the dependency graphs extracted from

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, pages 828–838
Vancouver, Canada, July 30 - August 4, 2017. ©2017 Association for Computational Linguistics
https://doi.org/10.18653/v1/P17-1077

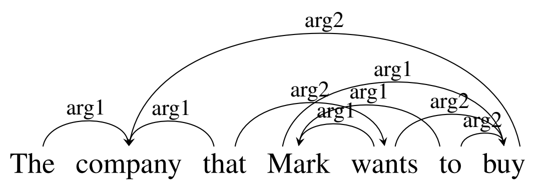


Figure 1: A fragment of a semantic dependency graph.

CCGbank (Hockenmaier and Steedman, 2007). On all data sets, we find that our higher-order parsing models are more accurate than the first-order baseline. Experiments also demonstrate the effectiveness of our page binding algorithm. Our new parser can be taken as a graph-based parser extended for more general dependency graphs. It parallels the state-of-the-art transition-based system of Zhang et al. (2016) in performance.

The implementation of our parser is available at `http://www.icst.pku.edu.cn/lcwm/grass`.

2 Background

2.1 Semantic Dependency Graphs

A dependency graph $G = (V, A)$ is a labeled directed graph for a sentence $s = w_1, \ldots, w_n$. The vertex set V consists of n vertices, each of which corresponds to a word and is indexed by an integer. The arc set A represents the labeled dependency relations of the particular analysis G. Specifically, an arc, viz. $a_{(i,j,l)}$, represents a dependency relation l from head w_i to dependent w_j.

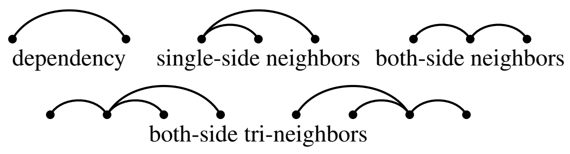
Semantic dependency parsing is the task of mapping a natural language sentence into a formal meaning representation in the form of a dependency graph. Figure 1 shows a graph fragment of a noun phrase. This semantic graph is grounded on Combinatory Categorial Grammar (CCG; Steedman, 2000), and can be taken as a proxy for predicate–argument structure. The graph includes most semantically relevant non-anaphoric local (e.g. from "wants" to "Mark") and long-distance (e.g. from "buy" to "company") dependencies.

2.2 Maximum Subgraph Parsing

Usually, syntactic dependency analysis employs *tree*-shaped representations. Dependency parsing, thus, can be formulated as the search for a maximum spanning tree (MST) of an arc-weighted graph. For semantic dependency parsing, where the target representations are not necessarily trees, Kuhlmann and Jonsson (2015) proposed to generalize the MST model to other types of subgraphs. In general, dependency parsing is 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 weight such that the induced subgraph $G' = (V, A')$ belongs to $\mathcal{G}$. Formally, we have the following optimization problem:

$$G'(s) = \arg \max_{H \in \mathcal{G}(s,G)} \sum_{p \in H} \textsc{ScorePart}(s, p)$$

Here, $\mathcal{G}(s, G)$ is the set of all graphs that belong to $\mathcal{G}$ and are compatible with s and G. For parsing, G is usually a complete graph. $\textsc{ScorePart}(s, p)$ evaluates the event that a small subgraph p of a candidate graph H is good. We define the *order* of a part according to the number of dependencies it contains, in analogy with tree parsing in terminology. Previous work only discussed the first-order case for Maximum Subgraph parsing (Kuhlmann and Jonsson, 2015). In this paper, we are also interested in higher-order parsing, with a special focus on factorizations utilizing the following parts:



If $\mathcal{G}$ is the set of projective trees or non-crossing graphs the first-order Maximum Subgraph problem can be solved in cubic-time (Eisner, 1996; Kuhlmann and Jonsson, 2015). Unfortunately, these two graph classes are not expressive enough to encode semantic dependency graphs. Moreover, this problem for several well-motivated graph classes, including acyclic or 2-planar graphs, is NP-hard, even if one only considers first-order factorization. The lack of appropriate decoding algorithms results in one major challenge for semantic dependency parsing.

2.3 Book Embedding

This section introduces the basic idea about book embedding from a graph theoretical point of view.

Definition 1. *A book is a kind of topological space that consists of a line, called the spine, together with a collection of one or more half-planes, called the pages, each having the spine as its boundary.*

Definition 2. *A book embedding of a finite graph G onto a book B satisfies the following conditions.*

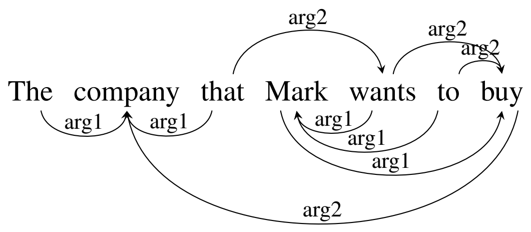


Figure 2: Book embedding for the graph in Figure 1. Arcs are assigned to two pages.

1. *Every vertex of G is depicted as a point on the spine of B.*
2. *Every edge of G is depicted as a curve that lies within a single page of B.*
3. *Every page of B does not have any edge crossings.*

A book embedding separates a graph into several subgraphs, each of which contains all vertices, but only a subset of arcs that are not crossed with each other. This kind of graph is named noncrossing dependency graph by Kuhlmann and Jonsson (2015) and planar by Titov et al. (2009), Gómez-Rodríguez and Nivre (2010) and many others.

We can formalize a semantic dependency graph as a *book*. Take the graph in Figure 1 for example. We can separate the edges into two sets and take each set as a single page, as shown in Figure 2.

Empirically, a semantic dependency graph is sparse enough that it can be that it can be usually embedded onto a very *thin* book. To measure the thickness, we can use pagenumber that is defined as follows.

Definition 3. *The book pagenumber of G is the minimum number of pages required for a book embedding of G.*

We look into the pagenumber of graphs on four linguistic graph banks (as defined in Section 5). These corpora are also used for training and evaluating our data-driven parsers. The pagenumbers are calculated using sentences in the training sets. Table 1 lists the percentages of complete graphs that can be accounted with books of different thickness. The percentages of noncrossing graphs, i.e. graphs that have pagenumber 1, vary between 48.23% and 78.26%. The practical usefulness of the algorithms for computing maximum noncrossing graphs will be limited by the relatively low coverage.

The class of graphs with pagenumber no more than two has a considerably satisfactory coverage.

PN	DM	PAS	CCD	PSD
1	69.83%	60.07%	48.23%	78.26%
2	29.85%	39.46%	49.86%	20.12%
3	0.31%	0.46%	1.71%	1.39%
4	0	0.02%	0.18%	0.21%
5	0	0	0.02%	0.02%
6	0	0	0	0.01%

Table 1: Coverage in terms of complete graphs with respect to different pagenumbers ("PN" for short). "DM," "PAS," "CCD" and "PSD" are short for DeepBank, Enju HPSGBank, CCGBank and Prague Dependency Treebank.

It can account for more than 98% of the graphs and sometimes close to 100% in each data set. Unfortunately, the power of Maximum Subgraph parsing is limited given that finding the maximum acyclic subgraph when pagenumber is at most k is NP-hard for $k \geq 2$ (Kuhlmann and Jonsson, 2015). As an alternative, we propose to model a semantic graph as a book, in which the spine is made up of a sequence of words, and each half-plane contains a subset of dependency arcs. To build a semantic graph for a given sentence, we design new parsing algorithms to generate noncrossing graphs on each page (Section 3), and a Lagrangian Relaxation-based algorithm to integrate pages into a book (Section 4).

3 Maximum Subgraph for Noncrossing Graphs

We introduce several DP algorithms for calculating the maximum noncrossing dependency graphs. Each algorithm visits all the spans from bottom to top, finding the best combination of smaller structures to form a new structure, according to the scores of first- or higher-order features. For sake of conciseness, we focus on undirected graphs and treat direction of linguistic dependencies as edge labels[1]. We will use $e_{(i,j,l)}(i < j)$ or simply $e_{(i,j)}$ to indicate an edge in either direction

[1] The *single-head* property does not hold. We currently do not consider other constraints of directions. So prediction of the direction of one edge does not affect prediction of other edges as well as their directions. The directions can be assigned *locally*, and our parser builds directed rather than undirected graphs in this way. Undirected graphs are only used to conveniently illustrate our algorithms. All experimental results in Section 5 consider directed dependencies in a standard way. We use the official evaluation tool provided by SDP2014 shared task. The numberic results reported in this paper are directly comparable to results in other papers.

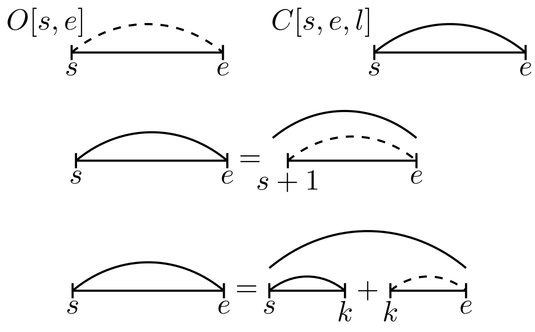


Figure 3: The sub-problems of first-order factorization and the decomposition for $C[s,e,l]$.

between i and j.

For sake of formal concision, we introduce the algorithm of which the goal is to calculate the maximum score of a subgraph. Extracting corresponding optimal graphs can be done in a number of ways. For example, we can maintain an auxiliary arc table which is populated parallel to the procedure of obtaining maximum scores. We define two score functions: (1) $\mathrm{s}_{\mathrm{fst}}(s,e,l)$ assigns a score to an individual edge $e_{(s,e,l)}$ and (2) $\mathrm{s}_{\mathrm{scd}}(s,e_1,e_2,l_1,l_2)$ assigns a score to a pair of neighboring edges $e_{(s,e_1,l_1)}$ and $e_{(s,e_2,l_2)}$.

3.1 First-Order Factorization

Given a sentence, we define two DP tables, namely $O[s,e]$ and $C[s,e,l]$ which represents the value of the highest scoring noncrossing graphs that spans sequences of words of a sentence. The two tables are related to two sub-problems, as graphically shown in Figure 3. The following is their explaination.

Open $O[s,e]$ is intended to represent the highest weighted subgraph spanning w_s to w_e. The subgraphs related to $O[s,e]$ may or may not contain $e_{(s,e)}$.

Closed $C[s,e,l]$ represents the highest weighted subgraph spanning w_s to w_e too. But the subgraphs related to $C[s,e,l]$ must contain $e_{(s,e,l)}$.

$O[s,e]$ can be obtained by one of the following combinations:

- $C[s,e,l](l \in L)$, if there is an edge between s and e with label l.
- $C[s,k,l] + O[k,e](l \in L, s < k < e)$, if $e_{(s,e)}$ does not exist and there is an edge with label l between s and some node in this span. k is the farthest node linked to s.
- $O[s+1,e]$, if $e_{(s,e)}$ does not exist and there is no edge to its right in this span.

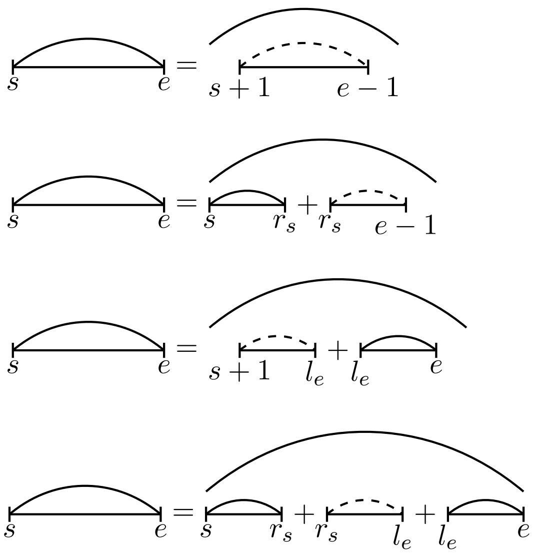


Figure 4: The decomposition for $C[s,e,l]$ in exact single-side second-order factorization.

$C[s,e,l]$ can be obtained by one of the following combinations:

- $O[s+1,e] + \mathrm{s}_{\mathrm{fst}}(s,e,l)$, if s has no edge to its right;
- $C[s,k,l'] + O[k,e] + \mathrm{s}_{\mathrm{fst}}(s,e,l)(l' \in L, s < k < e)$, if there is an edge from s to some node in the span.

For each edge, there are two directions for the edge, we encode the directions into the label l, and treat it as undirected edge. We need to search for a best split and a best label for every span, so the time complexity of the algorithm is $O(n^3|L|)$ where n is the length of the sentence and L is the set of labels.

3.2 Second-Order Single Side Factorization

We propose a new algorithm concerning single-side second-order factorization. The DP tables, as well as the decomposition for the open problem, are the same as in the first order factorization. The decomposition of $C[s,e,l]$ is very different. In order to score second-order features from adjacent edges in the same side, which is similar to *sibling* features for tree parsing (McDonald and Pereira, 2006), we need to find the rightmost node adjacent to s, denoted as r_s, and the leftmost node adjacent to e, denoted as l_e, and here we have $s < r_s \leq l_e < e$. And, sometimes, we split $C[s, e, l]$ into three parts to capture the neighbor factors on both endpoints. In summary, $C[s, e, l]$ can be obtained by one of the following combination (as graphically shown in Figure 4):

- $O[s + 1, e - 1] + s_{\text{fst}}(s, e, l) + s_{\text{scd}}(s, nil, e, nil, l) + s_{\text{scd}}(e, nil, s, nil, l)$, if there is no edge from *s/e* to any node in the span.
- $C[s, r_s, l'] + O[r_s, e - 1] + s_{\text{fst}}(s, e, l) + s_{\text{scd}}(s, r_s, e, l', l) + s_{\text{scd}}(e, nil, s, nil, l)$ $(s < r_s < e)$, if there is no edge from e to any node in the span.
- $O[s + 1, l_e] + C[l_e, e, l'] + s_{\text{fst}}(s, e, l) + s_{\text{scd}}(e, l_e, s, l', l) + s_{\text{scd}}(s, nil, e, nil, l)$ $(s < l_e < e)$, if there is no edge from s to any node in the span.
- $C[s, r_s, l'] + O[r_s, l_e] + C[l_e, e, l''] + s_{\text{fst}}(s, e, l) + s_{\text{scd}}(s, r_s, e, l', l) + s_{\text{scd}}(e, l_e, s, l'', l)$ $(s < r_s \leq l_e < e)$, otherwise.

For the last combination, we need to search for two best separating words, namely s_r and l_e, and two best labels, namely l' and l', so the time complexity of this second-order algorithm is $O(n^4|L|^2)$.

3.3 Generalized Higher-Order Parsing

Both of the above two algorithms are exact decoding algorithms. Solutions allow for exact decoding with higher-order features typically at a high cost in terms of efficiency. A trade-off between rich features and exact decoding benefit tree parsing (McDonald and Nivre, 2011). In particular, Zhang and McDonald (2012) proposed a generalized higher-order model that abandons exact search in graph-based parsing in favor of freedom in feature scope. They kept intact Eisner's algorithm for first-order parsing problems, while enhanced the scoring function in an approximate way by introducing higher-order features.

We borrow Zhang and McDonald's idea and develop a generalized parsing model for noncrossing dependency representations. The sub-problems and their decomposition are much like the first-order algorithm. The difference is that we expand the signature of each structure to include all the larger context required to compute higher-order features. For example, we can record the leftmost and the rightmost edges in the open structure to get the tri-neighbor features. The time complexity is thus always $O(n^3B^2)$, no matter how complicatedly higher-order features are incorporated.

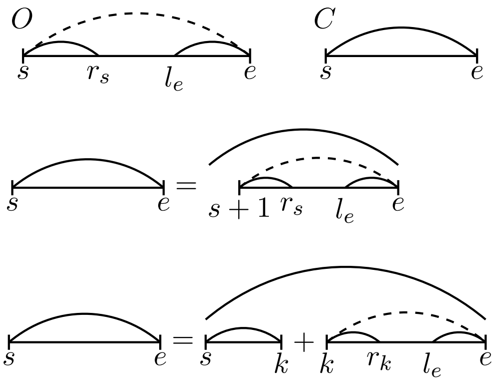


Figure 5: Sub-problems of generalized higher-order factorization and some of the combinations.

We focus on five factors introduced in Section 2.2. Still consider single-side second-order factorization. We keep the closed structure the same but modify the open one to $O[s, e; r_s, l_e, l_{s,r_s}, l_{l_e,e}]$. During parsing, we only record the top-B combinations of label concerning $e_{(s,e)}$ and related r_s, l_e, l_{s,r_s} and $l_{l_e,e}$. The split of a structure is similar to the first-order algorithm, shown in Figure 5. Note that r_s may be e and l_e may be s. In this way, we know exactly whether or not there is an edge from s to e in a refined open structure. This is different from the intuition of the design of the open structure when we consider first-order factorization.

4 Finding and Binding Pages

Statistics presented in Table 1 indicate that the coverage of noncrossing dependency graphs is relatively low. If we treat semantic dependency parsing as Maximum Subgraph parsing, the practical usefulness of the algorithms introduced above is rather limited accordingly. To deal with this problem, we model a semantic graph as a book, and view semantic dependency parsing as finding a book with coherent optimal pages. Given the considerably high coverage of pagenumber at most 2, we only consider 2-page books.

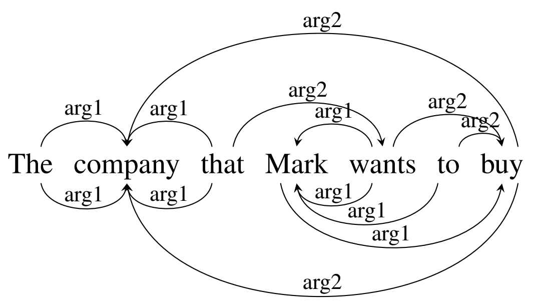


Figure 6: Every non-crossing arc is repeatedly assigned to every page.

4.1 Finding Pages via Coloring

In general, finding the pagenumber of a graph is NP-hard (Gómez-Rodríguez and Nivre, 2010). However, it is easy to figure out that the problem is solvable if the pagenumber is at most 2. Fortunately, a semantic dependency graph is not so dense that it can be usually embedded onto a very *thin* book with only 2 pages. For a structured prediction problem, the structural information of the output produced by a parser is very important. The density of semantic dependency graphs therefore results in a defect: The output's structural information is limited because only a half of arcs on average are included in one page. To enrich the structural information, we put into each page the arcs that do not cross with any other arcs. See Figure 6 for example.

We utilize an algorithm based on coloring to decompose a graph $G = (V, A)$ into two noncrossing subgraphs $G_A = (V, A_B)$ and $G_B = (V, A_B)$. A detailed description is included in the supplementary note. The key idea of our algorithm is to color each crossing arc in two colors using depth-first search. When we color an arc e_x, we examine all arcs crossing with e_x. If one of them, say e_y, has not been examined and can be colored in the other color (no crossing arc of e_y has the same color with e_y), we color e_y and then recursively process e_y. Otherwise, e_y is marked as a bad arc and dropped from both A_A and A_B. After coloring all the crossing arcs, we add every arc in different color to different subgraphs. Specially, all noncrossing arcs are assigned to both A_A and A_B.

4.2 Binding Pages via Lagrangian Relaxation

Applying the above algorithm, we can obtain two corpora to train two noncrossing dependency parsing models. In other words, we can learn two score functions f_A and f_B to score noncrossing dependency graphs. Given the trained models and a sentence, we can find two optimal noncrossing graphs, i.e. find the solutions for $\arg\max_{\boldsymbol{g}} f_A(\boldsymbol{g})$ and $\arg\max_{\boldsymbol{g}} f_B(\boldsymbol{g})$, respectively.

We can put all the arcs contained in $\boldsymbol{g}_A = \arg\max_{\boldsymbol{g}} f_A(\boldsymbol{g})$ and $\boldsymbol{g}_B = \arg\max_{\boldsymbol{g}} f_B(\boldsymbol{g})$ together as our parse for the sentence. This naive combination always gives a graph with a recall much higher than the precision. The problem is that a naive combination does not take the agreements of the graphs on the two pages into consideration, and thus loses some information. To combine the two pages in a principled way, we must do *joint* decoding to find two graphs $\boldsymbol{g}_A$ and $\boldsymbol{g}_B$ to maximize the score $f_A(\boldsymbol{g}_A) + f_B(\boldsymbol{g}_B)$, under the following constraints.

$$\begin{aligned} \boldsymbol{g}_A(i,j) &\le \sum_{\text{cross}((i,j),(i',j'))} \boldsymbol{g}_B(i',j') + \boldsymbol{g}_B(i,j) \\ \boldsymbol{g}_B(i,j) &\le \sum_{\text{cross}((i,j),(i',j'))} \boldsymbol{g}_A(i',j') + \boldsymbol{g}_A(i,j) \\ &\qquad \forall i,j \end{aligned}$$

The functionality of *cross* is to figure out whether $e_{(i,j)}$ and $e_{(i',j')}$ cross. The meaning of the first constraint is: When there is an arc $e_{(i,j)}$ in the first graph, $e_{(i,j)}$ is also in the second graph, or there is an arc $e_{(i',j')}$ in the second graph which cross with $e_{(i,j)}$. So is the second one. All constraints are linear and can be written in a simplified way as,

$$A\boldsymbol{g}_A + B\boldsymbol{g}_B \le \mathbf{0}$$

where A and B are matrices that can be constructed by checking all possible crossing arc pairs. In summary, we have the following constrained optimization problem,

$$\begin{aligned} \text{min.} \quad & -f_A(\boldsymbol{g}_A) - f_B(\boldsymbol{g}_B) \\ \text{s.t.} \quad & \boldsymbol{g}_A, \boldsymbol{g}_B \text{ are noncrossing graphs} \\ & A\boldsymbol{g}_A + B\boldsymbol{g}_B \le \mathbf{0} \end{aligned}$$

The Lagrangian of the optimization problem is

$$\begin{aligned} &\mathcal{L}(\boldsymbol{g}_A, \boldsymbol{g}_B; u) \\ &\quad = -f_g(\boldsymbol{g}_A) - f_t(\boldsymbol{g}_B) + u^\top(A\boldsymbol{g}_A + B\boldsymbol{g}_B) \end{aligned}$$

where u is the Lagrangian multiplier. Then the dual is

$$\begin{aligned} \mathcal{L}(u) &= \min_{\boldsymbol{g}_A, \boldsymbol{g}_B} \mathcal{L}(\boldsymbol{g}_A, \boldsymbol{g}_B; u) \\ &= \max_{\boldsymbol{g}_A}(f_g(\boldsymbol{g}_A) - u^\top A\boldsymbol{g}_A) \\ &\quad + \max_{\boldsymbol{g}_B}(f_y(\boldsymbol{g}_B) - u^\top B\boldsymbol{g}_B) \end{aligned}$$

```
BindTwoPages($g_A, g_B$)
1  $u^{(0)} \leftarrow 0$
2  for $k \leftarrow 0..T$ do
3      $g_A \leftarrow \arg\max_g f_A(g) - u^{(k)\top} Ag$
4      $g_B \leftarrow \arg\max_g f_B(g) - u^{(k)\top} Bg$
5      if $Ag_A + Bg_B \leq 0$ then
6          return $g_A, g_B$
7      else
8          $u^{(k+1)} \leftarrow u^{(k)} + \alpha^{(k)}(Ag_A + Bg_B)$
9  return $g_A, g_B$
```

Figure 7: The page binding algorithm.

We instead try to find the solution for $\max_u \mathcal{L}(u)$. By using a subgradient method to calculate $\max_u \mathcal{L}(u)$, we have an algorithm for joint decoding (see Figure 7). $\mathcal{L}(u)$ is divided into two optimization problems which can be decoded easily. Each sub-problem is still a parsing problem for noncrossing graphs. Only the scores of factors are modified (see Line 3 and 4). Specifically, to modify the first order weights of edges, we take a subtraction of $u^\top A$ in the first model and a substraction of $u^\top B$ in the second one. In each iteration, after obtaining two new parsing results, we check whether the constraints are satisfied. If the answer is "yes," we stop and return the merged graph. Otherwise, we update u in a way to increase $\mathcal{L}(u)$ (see Line 8).

5 Experiments

5.1 Data Sets

To evaluate the effectiveness of book embedding in practice, we conduct experiments on unlabeled parsing using four corpora: CCGBank (Hockenmaier and Steedman, 2007), DeepBank (Flickinger et al., 2012), Enju HPSGBank (EnjuBank; Miyao et al., 2004) and Prague Dependency TreeBank (PCEDT; Hajic et al., 2012), We use "standard" training, validation, and test splits to facilitate comparisons. Following previous experimental setup for CCG parsing, we use section 02-21 as training data, section 00 as the development data, and section 23 for testing. The other three data sets are from SemEval 2014 Task 8 (Oepen et al., 2014), and the data splitting policy follows the shared task. All the four data sets are publicly available from LDC (Oepen et al., 2016).

Experiments for CCG analysis were performed using automatically assigned POS-tags generated by a symbol-refined HMM tagger (Huang et al., 2010). For the other three data sets we use POS-tags provided by the shared task. We also use features extracted from trees. We consider two types of trees: (1) syntactic trees provided as a companion analysis by the shared task and CCGBank, (2) pseudo trees (Zhang et al., 2016) automatically extracted from semantic dependency annotations. We utilize the Mate parser (Bohnet, 2010) to generate pseudo trees for all data sets and also syntactic trees for CCG analysis, and use the companion syntactic analysis provided by the shared task for the other three data sets.

5.2 Statistical Disambiguation

Our parsing algorithms can be applied to scores originated from any source, but in our experiments we chose to use the framework of global linear models, deriving our scores as:

$$\textsc{ScorePart}(s, p) = \mathbf{w}^\top \phi(s, p)$$

ϕ is a feature-vector mapping and $\mathbf{w}$ is a parameter vector. p may refer to a single arc, a pair of neighboring arcs, or a general tuple of arcs, according to the definition of a parsing model. For details we refer to the source code. We chose the averaged structured perceptron (Collins, 2002) for parameter estimation.

5.3 Results of Practical Parsing

We evaluate five decoding algorithms:

M1 first-order exact algorithm,

M2 second-order exact algorithm with single-side factorization,

M3 second-order approximate algorithm[2] with single-side factorization,

M4 second-order approximate algorithm with single- and both-side factorization,

M5 third-order approximate algorithm with single- and both-side factorization.

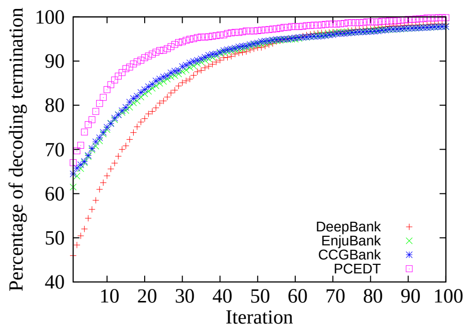
5.3.1 Effectiveness of Higher-Order Features

Table 2 lists the accuracy of Maximum Subgraph parsing. The output of our parser was evaluated against each dependency in the corpus. We report unlabeled precision (UP), recall (UR) and f-score (UF). We can see that the first-order model obtains a considerably good precision, with rich features.

[2]The beam size is set to 4 for all approximate algorithms.

			DeepBank			EnjuBank			CCGBank			PCEDT		
			UP	UR	UF	UP	UR	UF	UP	UR	UF	UP	UR	UF
Syntax Tree	M1	MS	90.97	86.11	88.47	92.92	89.71	91.29	94.21	88.70	91.37	91.49	86.39	88.87
	M2		91.04	87.47	89.22	93.03	90.48	91.74	93.95	88.96	91.39	91.11	87.56	89.30
	M3		90.94	87.65	89.27	93.27	90.62	91.93	93.93	89.11	91.46	91.25	87.66	89.42
	M4		91.02	87.78	89.37	93.18	90.65	91.90	94.02	89.14	91.51	91.43	87.98	89.67
	M5		90.91	87.51	89.18	93.15	90.57	91.84	93.91	89.19	91.49	91.29	87.96	89.59
	M4	NC	88.17	90.46	89.30	91.42	93.42	92.41	92.36	93.10	92.73	89.25	90.34	89.79
		LR	90.72	88.80	89.75	92.75	92.49	92.62	93.50	92.48	92.98	90.98	89.04	90.00
Pseudo Tree	M1	MS	90.75	86.13	88.38	93.38	90.20	91.76	94.21	88.55	91.29	90.62	85.69	88.08
	M2		90.13	87.01	88.54	93.18	90.63	91.89	93.96	88.54	91.17	89.92	86.55	88.20
	M3		90.39	87.20	88.77	93.20	90.64	91.90	93.90	88.98	91.37	90.07	86.69	88.35
	M4		90.31	87.25	88.76	93.18	90.67	91.91	94.01	89.04	91.46	90.03	86.84	88.40
	M5		90.17	87.11	88.61	93.13	90.62	91.86	93.87	89.00	91.37	90.21	86.93	88.54
	M4	NC	88.39	89.85	89.11	91.63	93.24	92.43	92.83	92.97	92.90	88.51	88.97	88.74
		LR	90.01	88.55	89.27	92.79	92.59	92.69	93.78	92.28	93.02	90.04	87.92	88.97

Table 2: Parsing accuracy evaluated on the development sets. "MS" is short for Maximum Subgraph parsing. "NC" and "LR" are short for naive combination and Lagrangian Relaxation.



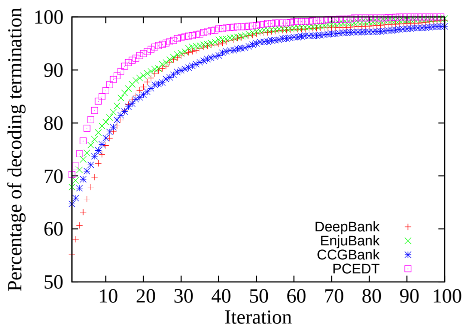


Figure 8: The termination rate of page binding. The left and right diagrams show the results obtained when applying syntactic and pseudo tree features respectively.

But due to the low coverage of the noncrossing dependency graphs, a set of dependencies can not be built. This property has a great impact on recall. Furthermore, we can see that the introduction of higher-order features improves parsing substantially for all data sets, as expected. When pseudo trees are utilized, the improvement is marginal. We think the reason is that we have already included many higher-order features at the stage of pseudo tree parsing.

5.3.2 Effectiveness of Approximate Parsing

Perhaps surprisingly approximate parsing with single-side second order features and cube pruning is even slightly better than exact parsing. This result demonstrates the effectiveness of generalized dependency parsing. Further including third-order features does not improve parsing accuracy.

5.3.3 Effectiveness of Page Binding

When arcs are assigned to two sets, we can separately train two parsers for producing two types of noncrossing dependency graphs. These two parsers can be integrated using a naive merger or a LR-based merger. Table 2 also shows the accuracy obtained by the second-order model M4. The effectivenss of the Lagrangian Relaxation-based algorithm for binding pages is confirmed.

5.3.4 Termination Rate of Page Binding

Figure 8 presents the termination rate with respective to the number of iterations. Here we apply M4 with syntax and pseudo tree features. In practice the Lagrangian Relaxation-based algorithm finds solutions in a few iterations for a majority of sentences. This suggests that even though the joint decoding is an iterative procedure, satisfactory efficiency is still available.

		DeepBank			EnjuBank			CCGBank			PCEDT		
		UP	UR	UF	UP	UR	UF	UP	UR	UF	UP	UR	UF
M4-LR	Syn	89.99	87.77	88.87	92.87	92.04	92.46	93.45	92.51	92.98	89.58	87.73	88.65
	Pse	90.01	88.16	89.08	93.17	92.48	92.83	93.66	92.06	92.85	89.27	87.37	88.31
ZDSW	Pse	89.04	88.85	88.95	92.92	92.83	92.87	92.49	92.30	92.40	- -	- -	- -
Peking		91.72	89.92	90.81	94.46	91.61	93.02	- -	- -	- -	91.79	86.02	88.81

Table 3: Parsing accuracy evaluated on the test sets.

5.4 Comparison with Other Parsers

We show the parsing results on the test data together with some relevant results from related work. We compare our parser with two other systems: (1) ZDSW (Zhang et al., 2016) is a transition-based system that obtains state-of-the-art accuracy; we present the results of their best single parsing model; (2) Peking (Du et al., 2014) is the best-performing system in the shared task; it is a hybrid system that integrate more than ten sub-models to achieve high accuracy. Our parser can be taken as a graph-based parser. It reaches state-of-the-art performance produced by the transition-based system. On DeepBank and EnjuBank, the accuracy of our parser is equivalent to ZDSW, while on CCGBank, our parser is significantly better.

There is still a gap between our single parsing model and Peking hybrid model. For a majority of NLP tasks, e.g. parsing (Surdeanu and Manning, 2010), semantic role labeling (Koomen et al., 2005), hybrid systems that combines complementary strength of heterogeneous models perform better. But good individual system is the cornerstone of hybrid systems. Better design of single system almost always benefits system ensemble.

6 Conclusion

We propose a new data-driven parsing framework, namely book embedding, for semantic dependency analysis, viz. mapping from natural language sentences to bilexical semantic dependency graphs. Our work includes two contributions:

1. new algorithms for maximum noncrossing dependency parsing.
2. a Lagrangian Relaxation based algorithm to combine noncrossing dependency subgraphs.

Experiments demonstrate the effectiveness of the book embedding framework across a wide range of conditions. Our graph-based parser obtains state-of-the-art accuracy.

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). Xiaojun Wan 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.

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.

Yantao Du, Weiwei Sun, and Xiaojun Wan. 2015. A data-driven, factorization parser for CCG dependency structures. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 1545–1555. http://www.aclweb.org/anthology/P15-1149.

Yantao Du, Fan Zhang, Weiwei Sun, and Xiaojun Wan. 2014. Peking: Profiling syntactic tree parsing techniques for semantic graph parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Association for Computational Linguistics and Dublin City University, Dublin, Ireland, pages 459–464. http://www.aclweb.org/anthology/S14-2080.

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.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Uppsala, Sweden, pages 1492–1501. http://www.aclweb.org/anthology/P10-1151.

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.

Angelina Ivanova, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who did what to whom? A contrastive study of syntacto-semantic dependencies. In *Proceedings of the Sixth Linguistic Annotation Workshop*. Jeju, Republic of Korea, pages 2–11.

Peter Koomen, Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2005. Generalized inference with multiple semantic role labeling systems. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*. Association for Computational Linguistics, Ann Arbor, Michigan, pages 181–184.

Sandra Kübler, Ryan T. McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool.

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.

Ryan T. McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics* 37(1):197–230.

Yusuke Miyao, Takashi Ninomiya, and Jun ichi Tsujii. 2004. 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, Silvie Cinková, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Zdeňka Urešová. 2016. Semantic Dependency Parsing (SDP) graph banks release 1.0 LDC2016T10. Web Download.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresová. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*.

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.

Mark Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble models for dependency parsing: Cheap and good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Los Angeles, California, pages 649–652. http://www.aclweb.org/anthology/N10-1091.

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *Proceedings of the 21st international jont conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pages 1562–1567. http://dl.acm.org/citation.cfm?id=1661445.1661696.

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.