Parsing to Noncrossing Dependency Graphs

Marco Kuhlmann and **Peter Jonsson**
Department of Computer and Information Science
Linköping University, Sweden
marco.kuhlmann@liu.se and peter.jonsson@liu.se

Abstract

We study the generalization of maximum spanning tree dependency parsing to maximum acyclic subgraphs. Because the underlying optimization problem is intractable even under an arc-factored model, we consider the restriction to *noncrossing* dependency graphs. Our main contribution is a cubic-time exact inference algorithm for this class. We extend this algorithm into a practical parser and evaluate its performance on four linguistic data sets used in semantic dependency parsing. We also explore a generalization of our parsing framework to dependency graphs with *pagenumber at most k* and show that the resulting optimization problem is NP-hard for $k \geq 2$.

1 Introduction

Dependency parsers provide lightweight representations for the syntactic and the semantic structure of natural language. Syntactic dependency parsing (Kübler et al., 2009) has been an extremely active research area for the last decade or so, resulting in accurate and efficient parsers for a wide range of languages. Semantic dependency parsing has only recently been addressed in the literature (Oepen et al., 2014; Oepen et al., 2015; Du et al., 2015a).

Syntactic dependency parsing has been formalized as the search for maximum spanning trees in weighted digraphs (McDonald et al., 2005b). For semantic dependency parsing, where target representations are not necessarily tree-shaped, it is natural to generalize this view to maximum acyclic subgraphs, with or without the additional requirement of weak connectivity (Schluter, 2014).

While a maximum spanning tree of a weighted digraph can be found in polynomial time (Tarjan, 1977), computing a maximum acyclic subgraph is intractable, and even good approximate solutions are hard to find (Guruswami et al., 2011). In this paper we therefore address maximum acyclic subgraph parsing under the restriction that the subgraph should be *noncrossing*, which informally means that its arcs can be drawn on the half-plane above the sentence in such a way that no two arcs cross (and without changing the order of the words). The main contribution of this paper is an algorithm that finds a maximum noncrossing acyclic subgraph of a (vertex-ordered) weighted digraph on n vertices in time $O(n^3)$.

After giving some background (Section 2) we introduce the noncrossing condition, compare it to other structural constraints from the literature, and study its empirical coverage (Section 3). We then present our parsing algorithm (Section 4). To turn this algorithm into a practical parser, we combine it with an off-the-shelf feature model and online training (Section 5); the source code of our system is released with this paper.[1] We evaluate the performance of our parser on four linguistic data sets: those used in the recent SemEval task on semantic dependency parsing (Oepen et al., 2015), and the dependency graphs extracted from CCGbank (Hockenmaier and Steedman, 2007). Finally, we explore the limits of our approach by showing that finding the maximum acyclic subgraph under a natural generalization of the noncrossing condition, *pagenumber at most k*, is NP-hard for $k \geq 2$ (Section 6). We conclude the paper by discussing related and future work (Section 7).

[1] https://github.com/liu-nlp/gamma

Transactions of the Association for Computational Linguistics, vol. 3, pp. 559–570, 2015. Action Editor: Joakim Nivre.
Submission batch: 9/2015;Published 11/2015.

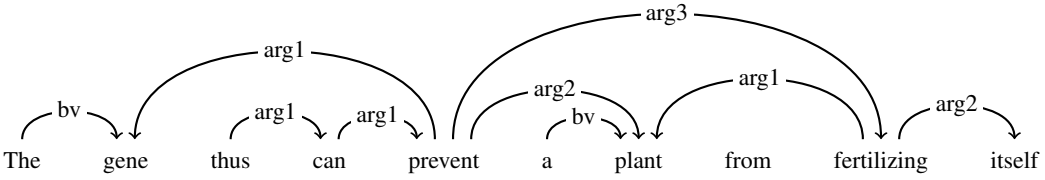


Figure 1: A noncrossing dependency graph (Oepen, 2014, DM #20209003). Using the terminology of Sagae and Tsujii (2008), the vertices *the*, *thus*, *a* and *from* are roots; of these, *thus*, *a* and *from* are covered by arcs.

2 Background

Dependency parsing is the task of mapping a natural language sentence into a formal representation of its syntactic or semantic structure in the form of a dependency graph.

2.1 Dependency Graphs

A *directed graph* or *digraph* is a pair $G = (V, A)$ where V is a set of *vertices* and $A \subseteq V \times V$ is a set of *arcs*. We consider an arc (u, v) to be directed from u to v and write it as $u \to v$. A (directed) *path* from u to v is a sequence of arcs of the form

$$v_0 \to v_1, v_1 \to v_2, \ldots, v_{m-1} \to v_m\,, \qquad m \geq 0\,,$$

where $v_0 = u$ and $v_m = v$. Note that m is allowed to be zero; in this case the path is called *empty*. A digraph is *acyclic* if there is no vertex u with a nonempty path from u to itself. A digraph is a *tree* if there exists a vertex r, the *root*, such that for every vertex u there is exactly one path from r to u. Every tree is acyclic, but not every acyclic graph is a tree.

Throughout this paper we write x for the generic sentence with n words. A *dependency graph* for x is an acyclic digraph $G = (V, A)$ whose vertices correspond one-to-one to the words in x. This correspondence imposes a total order on the vertices; we represent this order by equating V with the set of positions in x, $V = \{1, \ldots, n\}$. An example dependency graph is shown in Figure 1.[2] A *dependency tree* is a dependency graph that forms a tree. The example graph is not a tree.

2.2 Maximum Spanning Tree Parsing

McDonald et al. (2005b) cast dependency parsing as the search for a maximum spanning tree of an arc-weighted digraph. They start from the complete graph on n vertices where each arc $i \to j$ carries a real-valued weight w_{ij}, defined as a dot product

$$w_{ij} = \boldsymbol{w} \cdot \Phi(x, i \to j)$$

where Φ is a function that maps the sentence–arc pair into a *feature vector* and $\boldsymbol{w}$ is a global *weight vector* that is learned from training data. Taking the feature representation and the weight vector to be fixed, parsing under this model amounts to finding a spanning tree of maximum total weight.

2.3 Maximum Subgraph Parsing

For semantic dependency parsing, where the target representations are not necessarily trees (viz. Figure 1), we generalize the model of McDonald et al. (2005b) to other types of subgraphs. In general we are interested in the following optimization problem:

MAXIMUM SUBGRAPH FOR GRAPH CLASS $\mathcal{G}$

Given an arc-weighted digraph $G = (V, A)$, find a subset $A' \subseteq A$ with maximum total weight such that the induced subgraph $G' = (V, A')$ belongs to $\mathcal{G}$.

The computational complexity of this problem varies with the choice of $\mathcal{G}$. If $\mathcal{G}$ is the set of all dependency trees, then the problem can be solved in time $O(|V|^2)$ (Tarjan, 1977). If $\mathcal{G}$ is the unrestricted set of all dependency graphs, then the MAXIMUM SUBGRAPH problem is equivalent to the following:

MAXIMUM ACYCLIC SUBGRAPH

Given an arc-weighted digraph $G = (V, A)$, find a subset $A' \subseteq A$ with maximum total weight such that the induced subgraph $G' = (V, A')$ is acyclic.

This problem is known to be NP-hard, and also hard to approximate (Guruswami et al., 2011).

[2] Note that the arcs of the example graph are labeled. To simplify the presentation we mostly ignore this aspect in this paper; but our implementation supports parsing to labeled arcs.

3 Noncrossing Dependency Graphs

Because of the NP-hardness of MAXIMUM ACYCLIC SUBGRAPH we cannot expect to find a polynomial-time parsing algorithm for general dependency graphs. In this section we therefore introduce the restricted class of *noncrossing* dependency graphs.

3.1 The Noncrossing Condition

Let $G = (V, A)$ be a dependency graph. Recall that G is vertex-ordered by the correspondence of vertices to positions in x. Two arcs $a_1, a_2 \in A$ *cross* if either

$$\min(a_1) < \min(a_2) < \max(a_1) < \max(a_2) \quad \text{or}$$
$$\min(a_2) < \min(a_1) < \max(a_2) < \max(a_1)$$

where $\min(a)$ and $\max(a)$ denote the left and right vertex of the arc a, respectively. The graph G is called *noncrossing* if there are no two arcs that cross. For example, the graph in Figure 1 is noncrossing. Its picture is an *arc diagram*, a graph layout where one places the vertices along a line and draws each arc as a smooth curve in one of the two half-planes bounded by that line. Noncrossing dependency graphs can be characterized as exactly those graphs that permit arc diagrams where

1. the vertices are lined up in their total order,
2. all arcs are drawn on the upper half-plane only,
3. two curves intersect at most at their endpoints.

Our noncrossing condition is often referred to as *planarity*; see e.g. Titov et al. (2009). We propose to reserve the term "planar" for its standard use in graph theory.[3]

The noncrossing condition is well-known in the area of enumerative combinatorics; see for example the overview article by Flajolet and Noy (1999). In this context, one typically thinks of the vertices of a graph as being laid out on a circle, say in counterclockwise order. Then the noncrossing condition requires that the arcs of the graph can be drawn inside the circle without two arcs crossing.

3.2 Related Properties

Projectivity In syntactic dependency parsing, where the target representations are trees, the noncrossing condition is closely related to *projectivity*. More specifically, a dependency tree is projective if and only if it is noncrossing and its root is not "covered", meaning that there is no arc $i \to j$ such that the root lies properly between i and j. Sagae and Tsujii (2008) propose a generalization of this two-part characterization of projectivity to graphs. We find that the noncrossing condition alone is more practical. For example, the dependency graph in Figure 1 is not projective in the sense of Sagae and Tsujii (2008); but it is noncrossing.

Schluter (2015) uses the term "projective" with the same meaning as our term "noncrossing."

Pagenumber A natural generalization of the noncrossing condition is to relax property 2 of the arc diagram characterization and allow arcs to be drawn also in the lower half-plane bounded by the vertex line, or in any of some fixed number k of half-planes. These half-planes may be thought of as the pages of a book, with the vertex line corresponding to the book's spine, and the embedding of a graph into such a structure is known as a *book embedding* (Bernhart and Kainen, 1979). A graph that permits a crossing-free book embedding with k half-planes is said to have *pagenumber at most* k. Note that we here consider book embeddings where the order of the vertices along the boundary line is known in advance; it is given by the order of the words in the sentence.

3.3 Coverage on Linguistic Data

To estimate the practical value of a parser for noncrossing dependency graphs, we look into the coverage of these graphs on relevant linguistic data.

3.3.1 Data Sets

We choose four data sets that are generally available (through the Linguistic Data Consortium) and have already been used to build data-driven parsers.

SDP Dependencies These data sets (Oepen, 2014) consist of aligned sets of bi-lexical dependency graphs over the same Wall Street Journal text in three different representation types:

1. DM: DELPH-IN MRS-Derived Dependencies
2. PAS: Enju Predicate–Argument Structures
3. PSD: Prague Semantic Dependencies

[3] A graph is called planar if it can be drawn on the plane (not half-plane!) in such a way that no arcs cross each other.

property		DM	PAS	PSD	CCG
projective	G	2.95	1.67	42.04	1.88
	A	57.79	51.05	89.70	56.75
noncrossing	G	69.21	59.66	64.61	48.23
	A	97.66	97.26	96.01	95.76
pagenumber at most two	G	99.61	99.73	98.13	98.62
	A	99.98	99.99	99.87	99.93

Table 1: Coverage in terms of complete graphs (G) and individual arcs (A) under various structural restrictions.

These were used in the SemEval 2015 Task on Semantic Dependency Parsing (SDP; Oepen et al., 2015).[4]

CCG Dependencies This data set (Hockenmaier and Steedman, 2005) consists of the bi-lexical semantic dependency triples released with CCGbank, which also is based on the Wall Street Journal text. These triples were designed to reflect the underlying predicate–argument structure of the corresponding CCG derivations (Hockenmaier and Steedman, 2007). Recent work views the set of all triples for a sentence as a dependency graph and parses directly into these target representations (Du et al., 2015a).

3.3.2 Results

For the four data sets, Table 1 shows the percentages of complete graphs (G) and individual arcs (A) that can be accounted for under the restriction to noncrossing dependency graphs. These percentages provide upper bounds for the performance of a parser for these graphs under two standard evaluation metrics, exact match and arc-based recall. For comparison, the table also shows results for projective dependency graphs and graphs with pagenumber at most two.

The percentages of noncrossing graphs, and hence the maximal possible scores with respect to exact match, vary between 48.23% for CCG and 64.61% for PSD. On all data sets, they are considerably higher than those for projective dependency graphs. We take this as further evidence that the noncrossing property is a more practical restriction than projectivity (as discussed in Section 3.2) when it comes to semantic dependency parsing.

[4]The task organizers also provided data for other languages than English, but the English data were the only ones that were available in all three representation types.

The percentages of arcs that can be accounted for under the noncrossing condition are computed by maximizing, for every graph in the data, the cardinality of a subset of arcs that can be selected such that the subgraph induced by the selected arcs is noncrossing. These percentages, and hence the maximal possible scores with respect to arc-based recall, are close to 96% for PSD and CCG, and exceed 97% for DM and PAS. This suggests that, while a parser restricted to noncrossing dependency graphs would necessarily score low in terms of exact match, it could in principle still obtain relatively high scores in terms of arc-based recall.

The class of graphs with pagenumber at most two has the highest coverage, both with respect to exact match and arc recall. It can account for more than 98% of the graphs and more than 99% of the arcs in each of the four data sets. However, we will show in Section 6 that there is no polynomial-time parsing algorithm for this class of graphs (unless P = NP).

4 Parsing Algorithm

This section contains the main contribution of this paper: a cubic-time exact algorithm for solving the MAXIMUM SUBGRAPH problem for the class of noncrossing dependency graphs.

Theorem 1 *For noncrossing dependency graphs,* MAXIMUM SUBGRAPH *can be solved in time* $O(|V|^3)$.

4.1 Deduction System

We specify the parsing algorithm by means of a weighted deduction system in the sense of Nederhof (2003). The heart of such a system is a finite set of *inference rules* that specify how to derive solutions to subproblems of the overall problem from solutions to "simpler" subproblems. These partial solutions are represented by weighted formulas called *items*. Derivations start from a finite set of initial items called *axioms*, and the objective is to find the derivation of a *goal item* with maximal weight. This search can be carried out using a variant of Viterbi's algorithm (Viterbi, 1967).

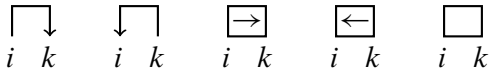
For the following, we assume that we are given a digraph $G = (V, A)$ with vertices $V = \{1, \ldots, n\}$ and arc weights as described in Section 2. We use i, j, k as metavariables for vertices (positions of the input sentence) where $i \leq k$ and $i < j < k$.

4.1.1 Items

We consider subproblems where we construct a maximum noncrossing dependency graph on a given (closed) interval $[i, k] \subseteq V$ of vertices, using only arcs in A. Based on the structure of the subgraph, we distinguish five different types of subproblems:

1. Construct a graph that contains the arc $i \rightarrow k$. We say that such a graph is *min–max-covered*.
2. Construct a graph that contains the arc $k \rightarrow i$. We say that such a graph is *max–min-covered*. Note that type 1 and type 2 are mutually exclusive, as the arcs $i \rightarrow k$ and $k \rightarrow i$ together would form a cycle. If a graph is either min–max-covered or max–min-covered, we say that it is *arc-covered*.
3. Construct a graph that is not arc-covered but contains a nonempty directed path from i to k. We say that such a graph is *min–max-connected*.
4. Construct a graph that is not arc-covered but contains a nonempty directed path from k to i. We say that such a graph is *max–min-connected*. Type 3 and type 4 are mutually exclusive, as the two paths together would form a cycle.
5. Construct a graph that is not arc-covered and does not contain a nonempty directed path between i and k. We say that such a graph is *bland*.

We represent these different types of subproblems by the following items:



We shall set up the weight of an item in such a way that it corresponds to the sum of the arc weights of the constructed subgraph.

4.1.2 Axioms

For each vertex i in G, the graph on the one-vertex interval $[i, i]$ is a bland graph. Therefore, the items of type $\square$ with $i = k$ are sound axioms of the deduction system. Because one-vertex dependency graphs have no arcs, we assign zero weight to these.

4.1.3 Goal Items

Our objective is to construct a maximum noncrossing dependency graph on the full vertex set. Therefore, the goal items of the deduction system are all items over the full interval $[1, n]$.

4.1.4 Inference Rules

The inference rules of the deduction system are shown in Figure 2. For each rule, we let the weight of the consequent be the sum of the weights of the antecedents, plus (for R16–R19) the weight of the arc required by the side condition. Thus, rule R01 for example states that whenever we have constructed a min–max-covered graph ($\sqcap\!\!\downarrow$) on the interval $[i, j]$ with some weight w_1 and another min–max-covered graph on the interval $[j, k]$ with some weight w_2, it is sound to conclude that we can construct a min–max-connected graph ($\boxed{\rightarrow}$) on the interval $[i, k]$ with weight $w_1 + w_2$. Similarly, rule R19 states that whenever we have constructed a bland graph ($\square$) on the interval $[i, k]$ with some weight w, then we may add the arc $k \rightarrow i$ (if it exists in G) and thus construct a max–min-covered graph ($\downarrow\!\!\sqcap$) whose weight is the sum of w and the weight of the new arc.

4.2 Correctness

We have already argued informally that the axioms and the inference rules of the deduction system are sound. In order to show completeness, we prove the following lemma: For each of the five types listed in Section 4.1.1, whenever the corresponding maximum noncrossing dependency graph on the interval $[i, k]$ has weight w, the appropriate item is derived.

We only sketch the (straightforward) proof of this lemma. We define the *size* of a graph as the total number of its vertices and arcs. The proof is by induction on the size. If the graph has size one (that is, consists of a single vertex), then it is bland and has weight zero; this case is covered by the axioms. For the inductive step, we consider a graph H with size $m > 1$ and assume that the lemma holds true for all graphs of smaller size. We then show how to derive the relevant item for H from the previously derived items for subgraphs of H using the inference rules.

4.3 Implementation and Runtime

In a Viterbi-style search for optimal derivations in the deduction system, we enumerate items by size and derive the weights of items with larger sizes from the weights of items with smaller sizes. Inspecting the maximal number of variables per rule (which is three, for R01–R08), we see that such a search can be implemented to run in time $O(n^3)$.

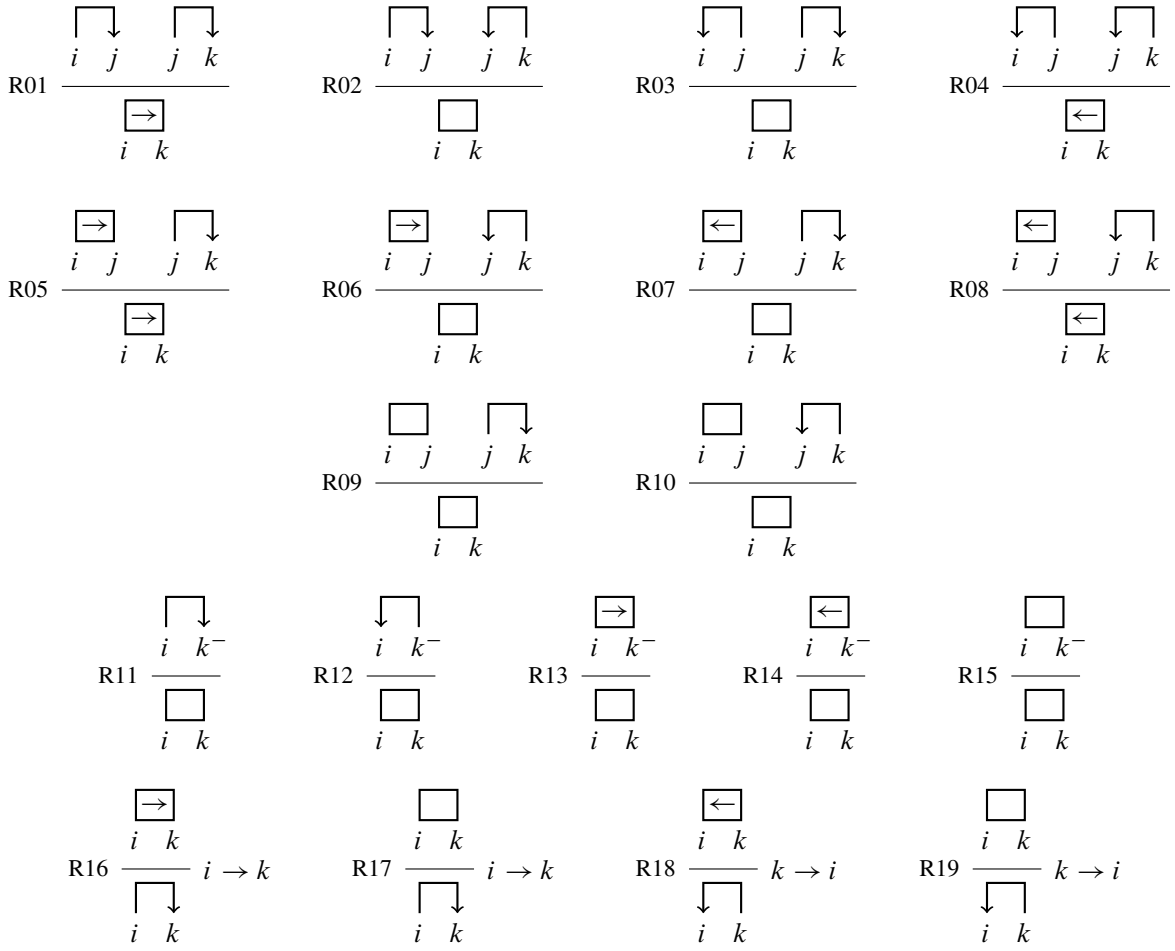


Figure 2: Deduction system for noncrossing dependency graphs ($1 \leq i < k \leq n, i < j < k$). We write k^- as a shorthand for $k - 1$. For each inference rule, the weight of the consequent is the sum of the weights of the antecedents, plus (for rules R16–R19) the weight of the arc required by the side condition.

		DM			PAS			PSD		
	$\overline{\text{LF}}$	LP	LR	LF	LP	LR	LF	LP	LR	LF
Peking	85.33	90.93	87.32	89.09	92.90	89.67	91.26	78.60	72.93	75.66
This work	79.63	86.43	79.17	82.64	88.68	83.94	86.24	74.58	65.96	70.01

	CCG		
	UP	UR	UF
Auli and Lopez			93.98
This work	93.06	86.65	89.74

Table 2: Parsing results on the SDP data (Oepen, 2014) and on CCG dependencies (Hockenmaier and Steedman, 2005). The metrics reported are precision, recall and F-score on labeled (LP, LR, LF) and unlabeled (UP, UR, UF) dependencies. $\overline{\text{LF}}$ is the averaged LF score that was used to rank systems in the SemEval-2015 Task on Broad-Coverage Semantic Dependency Parsing. References are to Du et al. (2015b) (Peking) and Auli and Lopez (2011).

4.4 Uniqueness of Derivations

Besides being sound and complete, the deduction system also has the property that it assigns a unique derivation to every noncrossing dependency graph. The proof, again, is by induction on the size. To illustrate the argument, suppose that we want to construct a min–max-connected graph H on some interval $[i, k]$. The only rules that can be used to derive the corresponding item (of type $\boxed{\rightarrow}$) are R01 and R05. In both rules, the graph corresponding to the second antecedent must be a min–max-covered graph ($\sqcap\!\!\downarrow$) on some interval $[j, k]$. This means that the vertex j is uniquely determined; it must be the left endpoint of the longest arc $j \rightarrow k$ in H. With this, the graph on the remaining interval $[i, j]$ is uniquely determined as well, and both graphs are smaller than H; hence we may assume that they have unique derivations. Note that j would *not* be uniquely determined if the deduction system included (say) an inference rule that derives an item $\boxed{\rightarrow}$ from two other such items: Such a rule would be sound, but it would create derivational ambiguity.

An immediate consequence of the uniqueness of derivations is that our parsing algorithm can be used for counting noncrossing dependency graphs, which is useful for, among other things, testing the correctness of implementations. To count, we change the system such that the weight of an item gives the number of its derivations.[5] The modified system yields sequence A246756 in the On-Line Encyclopedia of Integer Sequences (OEIS Foundation Inc., 2011).

4.5 Enforcing Weak Connectivity

Our deduction system can be modified to parse into (and count) various other classes of noncrossing graphs. For example, we can adapt our system to find a maximum noncrossing acyclic subgraph under the additional restriction that this subgraph should be weakly connected (Schluter, 2015). To do so we distinguish between two types of bland subgraphs, (a) weakly connected graphs and (b) graphs with exactly two weakly connected components, and adapt the inference rules and goal items. The change can be implemented without affecting the asymptotic runtime of the algorithm.

[5] Using the parlance of semiring parsing (Goodman, 1999), we switch from the max–plus semiring to the counting semiring.

Note that when we take the modified deduction system and consider undirected edges instead of directed arcs, we obtain an algorithm for finding maximum connected noncrossing graphs, which are counted by sequence A007297 in the OEIS. These graphs are the target representations of Link Grammar (Sleator and Temperley, 1993).

5 Practical Parsing

While the main focus of this paper is theoretical, in this section we extend our parsing algorithm into a practical parser. In the context of our general model (Section 2), this requires two additional components: a feature representation and a training algorithm.

5.1 Features

We use the arc-based features of TurboParser (Martins et al., 2009), which descend from several other feature models from the literature on syntactic dependency parsing (McDonald et al., 2005a; Carreras et al., 2006; Koo and Collins, 2010). In these models, the feature vector for an arc $i \rightarrow j$ represents information about various combinations of the exact forms, lemmas and part-of-speech tags of the words at positions i and j; the tags of the immediately surrounding words and the words between i and j; as well as the length of the arc and its direction. To support parsing to labeled dependency graphs, we additionally conjoin some of these features with the arc label. For details we refer to the source code.

5.2 Training

To learn the feature weights in the weight vector from data we use online passive–aggressive training as described by Crammer et al. (2006). For each gold instance (x, G) in the training data we let the parsing algorithm find the maximum noncrossing dependency graph $\hat{G}$ given the current weight vector $\boldsymbol{w}$ and update the weight vector as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \tau(\Phi(x, G) - \Phi(x, \hat{G}))$$

where $\Phi(x, G)$ and $\Phi(x, \hat{G})$ are the sum vectors of the arc-specific feature vectors for the dependency graphs G and $\hat{G}$, and the scalar τ is computed as

$$\min\left(C, \frac{\boldsymbol{w} \cdot (\Phi(x, \hat{G}) - \Phi(x, G)) + \sqrt{\ell(G, \hat{G})}}{\left\|\Phi(x, G) - \Phi(x, \hat{G})\right\|^2}\right).$$

In this formula, $\ell(G, \hat{G})$ is a user-defined loss function and $C > 0$ is a parameter that controls the tradeoff between optimizing the current loss and being close to the old weight vector. We use Hamming loss, defined as the number of (labeled) arcs that are exclusive to either G or $\hat{G}$. Following custom practice, we apply weight vector averaging (Freund and Schapire, 1999; Collins, 2002).

5.3 Parsing Experiments

We report parsing experiments on the four data sets described in Section 3.3.1 and discuss their results. We use gold-standard lemmas and part-of-speech tags, train each parser for 10 epochs, and report results for the final model on the test data. We use the splits recommended for the respective data sets. Following Carreras (2007), prior to training we transform each dependency graph in the training data to a closest noncrossing dependency graph.[6] In a prestudy using the DM development data we found the best value for the tradeoff parameter C to be 0.01.

5.3.1 Results and Discussion

The experimental results are shown in Table 2. For the SDP data, we report standard metrics used in the SemEval task (Oepen et al., 2015): precision, recall, and F1 on labeled dependencies. For the CCG data, we report the same metrics for unlabeled dependencies; these take into account only the two dependent words but not the lexical category containing the dependency relation or the argument slot.[7]

Given the low coverage of noncrossing dependency graphs on the four data sets (recall Section 3.3.2) and the use of a simple arc-factored model with its off-the-shelf features originally developed for syntactic parsing, it is not surprising that the parser does not achieve state-of-the art results. We still consider our results to be a useful reference for the emerging field of semantic dependency parsing. On the SDP data, the averaged labeled F1 of the parser is 79.63, which is 5.70 points below the corresponding score for the best-performing system in the task, Peking (Du et al., 2015b). Labeled F1 is highest on PAS (86.24) and lowest on PSD (70.01). On the CCG data, the parser achieves an unlabeled F1 of 89.74, 4.24 points below the best reported result for parsing with gold-standard part-of-speech tags (Auli and Lopez, 2011). On all four data sets, precision exceeds recall by a significant margin, more than one might expect given the relatively high upper bounds on arc-based recall that we observed in Section 3.3.2.

The speed of the parser is about 0.01 seconds per sentence or 110 sentences per second on each of the four test sets. Training for 10 epochs takes around 40 minutes per training set. Experiments were performed on an iMac 3.4 GHz Intel Core i5 CPU with 4 cores and Java 1.8.0.

6 Intractability for Higher Pagenumbers

The parsing algorithm presented in Section 4 has many attractive theoretical properties, but its practical usefulness is limited by the relatively low coverage of noncrossing dependency graphs on the linguistic data. It would be desirable to generalize the algorithm to more expressive classes of graphs. A natural candidate is the class of dependency graphs with pagenumber at most k, whose coverage is excellent already for $k = 2$, as we saw in Section 3.3.2. However, we shall now prove the following:

Theorem 2 *For dependency graphs with pagenumber at most k,* Maximum Subgraph *is NP-hard whenever $k \geq 2$.*

For the proof of this theorem we do not consider the maximization problem defined in Section 2.3 but the following decision version:

Maximum Subgraph for Graph Class $\mathcal{G}$, Decision Version

Given a digraph $G = (V, A)$ and an integer $m \geq 0$, is there a subset $A' \subseteq A$ with $|A'| \geq m$ such that the induced subgraph $G' = (V, A')$ belongs to $\mathcal{G}$?

Note that any polynomial-time algorithm for the maximization problem can be turned into a polynomial-time algorithm for the decision problem: Given an instance for the decision problem, assign each arc weight 1 and solve the maximization problem; then, test whether the solution contains at least m arcs. To show the NP-hardness of the maximization problem it therefore suffices to show it for the decision version of the problem.

[6] This is done by running the parser with an oracle model that assigns a score of $+1$ to correct and -1 to incorrect arcs.

[7] The high number of different arc labels in the CCG data exceeds what can be handled by the current version of our parser.

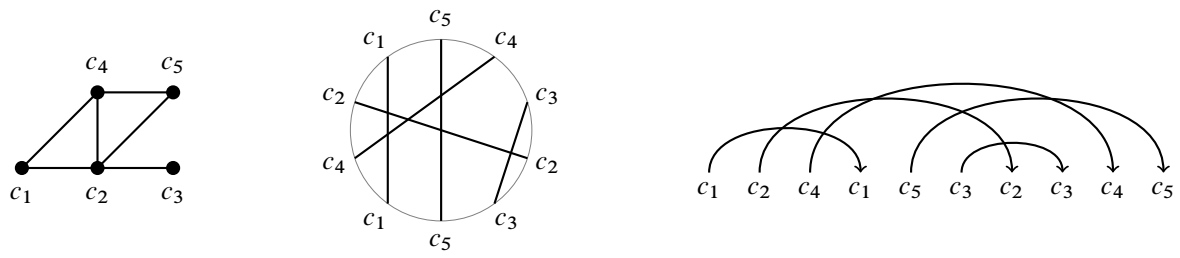


Figure 3: A circle graph (left), a corresponding chord drawing (middle), and a dependency graph as constructed by the algorithm specified in Section 6.2 (right). The example is adapted from Unger (1992).

6.1 Circle Graphs

To show that (the decision version of) MAXIMUM SUBGRAPH for dependency graphs with pagenumber at most k is NP-hard we present a polynomial reduction from a decision problem on *circle graphs*.

A circle graph is an undirected graph that represents the intersection graph of a set of chords of a circle. That is, for every circle graph G we can draw a circle and a set of chords of that circle such that the chords are in one-to-one correspondence with the vertices of G and two chords cross each other if and only if the corresponding vertices are adjacent in G. An example of a circle graph and a corresponding chord drawing is given in Figure 3. Note that one and the same circle graph may have many different chord drawings. Here, without loss of generality, we restrict our attention to drawings in which no two chords have a common endpoint. In these drawings, there are exactly twice as many points on the circle as there are vertices in the circle graph G.

6.2 Reduction

The relevant decision problem on circle graphs is given below. For a graph $G = (V, E)$ and a subset $V' \subseteq V$, we let $G|V'$ denote the subgraph of G induced by V', that is, $G|V'$ has vertex set V' and contains each edge in E between vertices in V'.

k-COLORABLE INDUCED SUBGRAPH FOR CIRCLE GRAPHS (k-CIG)

Given a circle graph $G = (V, E)$ and an integer $m \geq 0$, is there a subset $V' \subseteq V$ with $|V'| \geq m$ such that the induced graph $G|V'$ is k-colorable?

Cong and Liu (1991) show that this problem is NP-complete if $k \geq 2$. For $k = 2$ and $k \geq 4$ their proof is based on earlier results by Sarrafzadeh and Lee (1989) and Unger (1988), respectively.

The following procedure transforms an arbitrary circle graph G into a dependency graph H. For an illustration, see Figure 3.[8]

1. Construct a chord drawing for G. Recall that for this drawing there is a one-to-one correspondence between the chords and the vertices of G.
2. Cut the circle of the chord drawing and straighten it out into a line. This yields a total order on the endpoints of the chords. We identify each endpoint with its position in that order and denote the left and the right endpoint of a chord c by c_L and c_R, respectively. Because we assume that no two chords have a common endpoint, there will be exactly twice as many points on the line as there are chords and therefore vertices in G.
3. Direct each chord c from c_L to c_R. This defines a dependency graph whose arcs are the directed chords and whose vertices are (the positions of) the chords' endpoints.

The crucial property of this construction is that establishes a one-to-one correspondence between the vertices of G and the arcs of H. More formally, let $G = (V_G, E)$ and $H = (V_H, A)$. The construction establishes a bijection

$$a: V_G \to A \qquad \text{by} \qquad a(v) = c(v)_L \to c(v)_R$$

where c is the unique chord corresponding to the vertex v in the chord drawing of G.

The dependency graph H can be computed in polynomial time in the size of G. The non-obvious part is step 1; this can be carried out in time quadratic in the number of vertices of G using the algorithm of Spinrad (1994).

[8] The dependency graph obtained by this construction is essentially what Unger (1992) calls the *overlap graph model* of a circle graph, except that its edges are directed.

We now claim that an instance (G, m) of k-CIG yields a "yes" answer if and only if the corresponding instance (H, m) of MAXIMUM SUBGRAPH for dependency graphs with pagenumber at most k does.

Assume that (G, m) yields a positive answer. This means that there exists a subset of vertices $V'_G \subseteq V_G$ with $|V'_G| \geq m$ and such that the vertices in this set can be colored with at most k colors in such a way that no two adjacent vertices have the same color. Without loss of generality we assume that the colors are numbers between 1 and k. Let $f(v)$ denote the color assigned to v. Consider the set of arcs corresponding to V'_G, $A' = \{ a(v) \mid v \in V'_G \}$. We claim that A' is a solution set for (H, m). Clearly $|A'| \geq m$ and $H' = (V_H, A')$ is a dependency graph. We show that H' can be embedded into a k-book. Place every arc $a(v) \in A'$ on page $f(v)$. Assume now that two arcs $a_1 = a(v_1)$ and $a_2 = a(v_2)$ are placed on the same page and cross each other. This implies that $f(v_1) = f(v_2)$ and that there is an edge between v_1 and v_2 in G. Because $v_1, v_2 \in V'_G$, we see that this edge also appears in $G|V'_G$. This contradicts the assumption that the set V'_G is k-colorable.

Assume that (H, m) yields a positive answer. This means that there exists a set of arcs A' with $|A'| \geq m$ and such that there is an assignment of page numbers to the arcs in this set such that no two arcs with the same page number cross each other. Let $f(a)$ denote the page number assigned to a. Consider the set of vertices corresponding to A', $V'_G = \{ v \in V_G \mid a(v) \in A' \}$. We claim that V'_G is a solution set for (G, m). Clearly $|V'| \geq m$. To show that $G|V'_G$ is k-colorable, we interpret the page assignment f as a k-coloring of the vertices in V'_G in the obvious way: We color each vertex $v \in V'_G$ with the page number of the corresponding arc $a(v) \in A'$. Now consider two arbitrary vertices v_1 and v_2 that are adjacent in $G|V'$. By construction, the arcs $a(v_1)$ and $a(v_2)$ cross in H; therefore $f(a(v_1)) \neq f(a(v_1))$ and v_1 and v_2 are assigned different colors in $G|V'$.

This completes the proof of Theorem 2.

7 Conclusion

The goal of this paper was to generalize maximum spanning tree dependency parsing to target structures that are not necessarily tree-shaped. Because parsing to unrestricted dependency graphs is intractable, we studied the problem under the restriction to noncrossing graphs, which generalize projective dependency trees as they are known from syntactic parsing. We presented a cubic-time parsing algorithm for this class of graphs, extended the algorithm into a practical parser that we evaluated on four linguistic data sets, and finally proved that the (natural) step beyond the noncrossing condition to dependency graphs with pagenumber at most k renders parsing intractable.

The main contributions of this paper are theoretical. To the best of our knowledge, ours is the first exact-inference algorithm for the full class of noncrossing dependency graphs. The similar algorithm by Schluter (2015), which was developed contemporaneously but independently of ours, is restricted to (weakly) connected graphs. This is a severe practical limitation when semantically vacuous tokens are analyzed as unconnected nodes, as such an analysis renders most graphs unconnected.[9] Another difference between our algorithm and the algorithm of Schluter (2015) is that the latter does not have the uniqueness property discussed in Section 4.4.

An interesting follow-up question to our result in Section 6.2 is whether MAXIMUM SUBGRAPH remains NP-hard when the candidate space is restricted to *trees* with pagenumber at most k. This question was raised by Gómez-Rodríguez and Nivre (2013), who present a greedy parser for the case $k = 2$.

We view our algorithm as a canonical generalization of the Eisner and Satta (1999) parsing algorithm for projective dependency trees, and expect it to serve as a similar point of departure for future extensions of the paradigm. On the one hand, it seems interesting to explore the use of more expressive feature models, including the generalization of our arc-factored model to the grandparent and sibling features of Carreras (2007) and Koo and Collins (2010). On the other hand, given the low coverage of noncrossing dependency graphs, it seems necessary to explore the generalization to new, "mildly" crossing classes of graphs, such as a graph version of the 1-endpoint crossing trees of Pitler et al. (2013). Pitler (2014) shows that the two directions may also be combined, which we hope will lead to new, more accurate algorithms for semantic dependency parsing.

[9] For each of the data sets introduced in Section 3.3.1, less than 1% of the graphs are both noncrossing and connected.

Acknowledgments

We are grateful to: Jordan Tirrell, whose answer to a question on MathOverflow provided the seed for the parsing algorithm in Section 4; Emily Pitler, who proposed a presentational simplification of the algorithm; the participants of the Dagstuhl Seminar 15122 "Formal Models of Graph Transformation in Natural Language Processing", for discussions that eventually led to the hardness result in Section 6; and the anonymous reviewers and the action editor for constructive feedback. The research of Marco Kuhlmann is funded by the Linköping Institute of Technology under the CENIIT program.

References

Michael Auli and Adam Lopez. 2011. Training a log-linear parser with loss functions via softmax-margin. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 333–343, Edinburgh, UK.

Frank Bernhart and Paul C. Kainen. 1979. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331.

Xavier Carreras, Mihai Surdeanu, and Lluís Màrquez. 2006. Projective dependency parsing with perceptron. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 181–185, New York, USA.

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.

Michael Collins. 2002. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8, Philadelphia, USA.

Jason Cong and Chung Laung Liu. 1991. On the k-layer planar subset and topological via minimization problems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 10(8):972–981.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Yantao Du, Weiwei Sun, and Xiaojun Wan. 2015a. 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 of Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 1545–1555, Beijing, China.

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015b. Peking: Building semantic dependency graphs with a hybrid parser. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 927–931, Denver, CO, USA.

Jason 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 (ACL)*, pages 457–464, College Park, MD, USA.

Philippe Flajolet and Marc Noy. 1999. Analytic combinatorics of non-crossing configurations. *Discrete Mathematics*, 204(1–3):203–229.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Carlos Gómez-Rodríguez and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Computational Linguistics*, 39(4):799–845.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. 2011. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914.

Julia Hockenmaier and Mark Steedman. 2005. CCGbank LDC2005T13. Web Download.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank. A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33:355–396.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11, Uppsala, Sweden.

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

André F. T. Martins, Noah A. Smith, and Eric P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics (ACL) and the Fourth International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (IJCNLP)*, pages 342–350, Singapore.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98, Ann Arbor, USA.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Human Language Technology Conference (HLT) and Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–530, Vancouver, Canada.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

OEIS Foundation Inc. 2011. The on-line encyclopedia of integer sequences. `http://oeis.org`.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, 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)*, pages 63–72, Dublin, Republic of Ireland.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 Task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926, Denver, CO, USA.

Stephan Oepen. 2014. SemEval 2015 Task 18 evaluation LDC2014E100. Web Download. Dataset only available to task participants.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *Transactions of the Association for Computational Linguistics*, 1:13–24.

Emily Pitler. 2014. A crossing-sensitive third-order factorization for dependency parsing. *Transactions of the Association for Computational Linguistics*, 2:41–54.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 753–760, Manchester, UK.

Majid Sarrafzadeh and Der-Tsai Lee. 1989. A new approach to topological via minimization. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 8(8):890–900.

Natalie Schluter. 2014. On maximum spanning DAG algorithms for semantic DAG parsing. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 61–65.

Natalie Schluter. 2015. The complexity of finding the maximum spanning DAG and other restrictions for DAG parsing of natural language. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 259–268, Denver, CO, USA.

Daniel Sleator and Davy Temperley. 1993. Parsing English with a Link Grammar. In *Proceedings of the Third International Workshop on Parsing Technologies (IWPT)*, pages 277–292, Tilburg, The Netherlands, and Durbuy, Belgium.

Jeremy P. Spinrad. 1994. Recognition of circle graphs. *J. Algorithms*, 16(2):264–282.

Robert E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7:25–35.

Ivan Titov, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *International Joint Conferences on Artificial Intelligence*, pages 1562–1567, Pasadena, CA, USA.

Walter Unger. 1988. On the k-colouring of circle-graphs. In Robert Cori and Martin Wirsing, editors, *STACS 88, 5th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 11–13, 1988, Proceedings*, volume 294 of *Lecture Notes in Computer Science*, pages 61–72. Springer.

Walter Unger. 1992. The complexity of colouring circle graphs (extended abstract). In Alain Finkel and Matthias Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 389–400. Springer.

Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.