Finding Optimal 1-Endpoint-Crossing Trees

Emily Pitler, Sampath Kannan, Mitchell Marcus
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
epitler,kannan,mitch@seas.upenn.edu

Abstract

Dependency parsing algorithms capable of producing the types of crossing dependencies seen in natural language sentences have traditionally been orders of magnitude slower than algorithms for projective trees. For 95.8-99.8% of dependency parses in various natural language treebanks, whenever an edge is crossed, the edges that cross it all have a common vertex. The optimal dependency tree that satisfies this 1-Endpoint-Crossing property can be found with an $O(n^4)$ parsing algorithm that recursively combines forests over intervals with one exterior point. 1-Endpoint-Crossing trees also have natural connections to linguistics and another class of graphs that has been studied in NLP.

1 Introduction

Dependency parsing is one of the fundamental problems in natural language processing today, with applications such as machine translation (Ding and Palmer, 2005), information extraction (Culotta and Sorensen, 2004), and question answering (Cui et al., 2005). Most high-accuracy graph-based dependency parsers (Koo and Collins, 2010; Rush and Petrov, 2012; Zhang and McDonald, 2012) find the highest-scoring *projective* trees (in which no edges cross), despite the fact that a large proportion of natural language sentences are non-projective. Projective trees can be found in $O(n^3)$ time (Eisner, 2000), but cover only 63.6% of sentences in some natural language treebanks (Table 1).

The class of directed spanning trees covers all treebank trees and can be parsed in $O(n^2)$ with edge-based features (McDonald et al., 2005), but it is NP-hard to find the maximum scoring such tree with grandparent or sibling features (McDonald and Pereira, 2006; McDonald and Satta, 2007).

There are various existing definitions of mildly non-projective trees with better empirical coverage than projective trees that do not have the hardness of extensibility that spanning trees do. However, these have had parsing algorithms that are orders of magnitude slower than the projective case or the edge-based spanning tree case. For example, well-nested dependency trees with block degree 2 (Kuhlmann, 2013) cover at least 95.4% of natural language structures, but have a parsing time of $O(n^7)$ (Gómez-Rodríguez et al., 2011).

No previously defined class of trees simultaneously has high coverage and low-degree polynomial algorithms for parsing, allowing grandparent or sibling features.

We propose *1-Endpoint-Crossing* trees, in which for any edge that is crossed, all other edges that cross that edge share an endpoint. While simple to state, this property covers 95.8% or more of dependency parses in natural language treebanks (Table 1). The optimal 1-Endpoint-Crossing tree can be found in faster asymptotic time than any previously proposed mildly non-projective dependency parsing algorithm. We show how any 1-Endpoint-Crossing tree can be decomposed into isolated sets of intervals with one exterior point (Section 3). This is the key insight that allows efficient parsing; the $O(n^4)$ parsing algorithm is presented in Section 4. 1-Endpoint-Crossing trees are a subclass of 2-planar graphs (Section 5.1), a class that has been studied in NLP. 1-Endpoint-Crossing trees also have some linguistic interpretation (pairs of cross serial verbs produce 1-Endpoint-Crossing trees, Section 5.2).

2 Definitions of Non-Projectivity

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

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

Table 1 shows the percentage of dependency parses in the CoNLL-X training sets that are 1-Endpoint-Crossing trees. Across six languages with varying amounts of non-projectivity, 95.8-99.8% of dependency parses in treebanks are 1-Endpoint-Crossing trees.[1]

We next review and compare other relevant definitions of non-projectivity from prior work: well-nested with block degree 2, gap-minding, projective, and 2-planar.

The definitions of block degree and well-nestedness are given below:

Definition 3. *For each node u in the tree, a* block *of the node is "a longest segment consisting of descendants of u." (Kuhlmann, 2013). The* block-degree *of u is "the number of distinct blocks of u". The block degree of a tree is the maximum block degree of any of its nodes. The* gap degree *is the number of gaps between these blocks, and so by definition is one less than the block degree. (Kuhlmann, 2013)*

Definition 4. *Two trees "T_1 and T_2* interleave *iff there are nodes l_1,$r_1 \in T_1$ and l_2,$r_2 \in T_2$ such that $l_1 < l_2 < r_1 < r_2$." A tree is* well-nested *if no two disjoint subtrees interleave. (Bodirsky et al., 2005)*

As can be seen in Table 1, 95.4%-99.9% of dependency parses across treebanks are both well-nested and have block degree 2. The optimal such tree can be found in $O(n^7)$ time and $O(n^5)$ space (Gómez-Rodríguez et al., 2011).

[1]Conventional edges from the artificial root node to the root(s) of the sentence reduce the empirical coverage of 1-Endpoint-Crossing trees. Excluding these artificial root edges, the empirical coverage for Dutch rises to 12949 (97.0%). These edges have no effect on the coverage of well-nested trees with block degree at most 2, gap-minding trees, or projective trees.

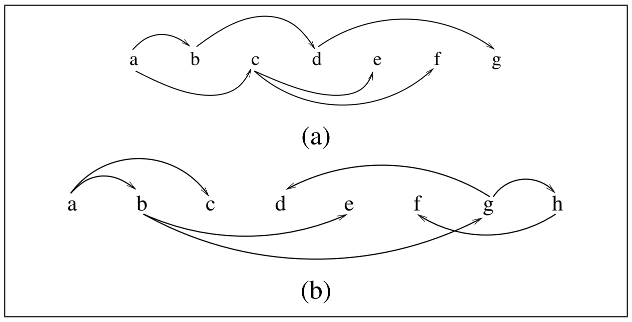


Figure 1: 1a is 1-Endpoint-Crossing, but is neither block degree 2 nor well-nested; 1b is gap-minding but not 2-planar.

Definition 5. *A tree is **gap-minding** if it is well-nested, has gap degree at most 1, and has* gap inheritance degree 0. Gap inheritance degree 0 *requires that there are no child nodes with descendants in more than one of their parent's blocks. (Pitler et al., 2012)*

Gap-minding trees can be parsed in $O(n^5)$ (Pitler et al., 2012). They have slightly less empirical coverage, however: 90.4-97.7% (Table 1).

Definition 6. *A tree is **projective** if it has block degree 1 (gap degree 0).*

This definition has the least coverage (as low as 63.6% for Dutch), but can be parsed in $O(n^3)$ (Eisner, 2000).

Definition 7. *A tree is **2-planar** if each edge can be drawn either above or below the sentence such that no edges cross (Gómez-Rodríguez and Nivre, 2010).*

Gómez-Rodríguez and Nivre (2010) presented a transition-based parser for 2-planar trees, but there is no known globally optimal parsing algorithm for 2-planar trees.

Clearly *projective $\subsetneq$ gap-minding $\subsetneq$ well-nested with block degree at most 2*. In Section 5.1, we prove the somewhat surprising fact that *1-Endpoint-Crossing $\subsetneq$ 2-planar*. These are two distinct hierarchies capturing different dimensions of non-projectivity: *1-Endpoint-Crossing $\not\subseteq$ well-nested with block degree 2* (Figure 1a), and *gap-minding $\not\subseteq$ 2-planar* (Figure 1b).

3 Edges (and their Crossing Point) Define Isolated Crossing Regions

We introduce notation to facilitate the discussion:

	Arabic	Czech	Danish	Dutch	Portuguese	Swedish	Parsing
1-Endpoint-Crossing	1457 (99.8)	71810 (98.8)	5144 (99.1)	12785 (95.8)	9007 (99.3)	10902 (98.7)	$O(n^4)$
Well-nested, block degree 2	1458 (99.9)	72321 (99.5)	5175 (99.7)	12896 (96.6)	8650 (95.4)	10955 (99.2)	$O(n^7)$
Gap-Minding	1394 (95.5)	70695 (97.2)	4985 (96.1)	12068 (90.4)	8481 (93.5)	10787 (97.7)	$O(n^5)$
Projective	1297 (88.8)	55872 (76.8)	4379 (84.4)	8484 (63.6)	7353 (81.1)	9963 (90.2)	$O(n^3)$
Sentences	1460	72703	5190	13349	9071	11042	

Table 1: Over 95% of the dependency parse trees in the CoNLL-X training sets are 1-Endpoint-Crossing trees. Coverage statistics and parsing times of previously proposed properties are shown for comparison.

Definition 8. *Within a 1-Endpoint-Crossing tree, the* **(crossing) pencil**[2] *of an edge e* ($\mathcal{P}(e)$) *is defined as the set of edges (sharing an endpoint) that cross e. The* **(crossing pencil) point** *of an edge e* ($\mathcal{P}t(e)$) *is defined as the endpoint that all edges in* $\mathcal{P}(e)$ *share.*

We will use e_{uv} to indicate an edge in either direction between u and v, i.e., either $u \rightarrow v$ or $u \leftarrow v$.

Before defining the parsing algorithm, we first give some intuition by analogy to parsing for projective trees. (This argument mirrors that of Eisner (2000, pps.38-39).) Projective trees can be produced using dynamic programming over *intervals*. Intervals are sufficient for projective trees: consider any edge e_{uv} in a projective tree.

The vertices in (u, v) must *only* have edges to vertices in $[u, v]$. If there were an edge between a vertex in (u, v) and a vertex outside $[u, v]$, such an edge would cross e_{uv}, which would contradict the assumption of projectivity. Thus every edge in a projective tree creates one interior interval isolated from the rest of the tree, allowing dynamic programming over intervals. We can analyze the case of 1-Endpoint-Crossing trees in a similar fashion:

Definition 9. *An* isolated interval $[i, j]$ *has no edges between the vertices in* (i, j) *and the vertices outside of* $[i, j]$*. An interval and one exterior vertex* $[i, j] \cup \{x\}$ *is called an* ***isolated crossing region*** *if the following two conditions are satisfied:*

1. *There are no edges between the vertices* $\in (i, j)$ *and vertices* $\notin [i, j] \cup \{x\}$
2. *None of the edges between* x *and vertices* $\in (i, j)$ *are crossed by any edges with both endpoints* $\in (i, j)$

[2]This notation comes from an analogy to geometry: "A set of distinct, coplanar, concurrent lines is a **pencil** of lines" (Ringenberg, 1967, p. 221); concurrent lines all intersect at the same single point.

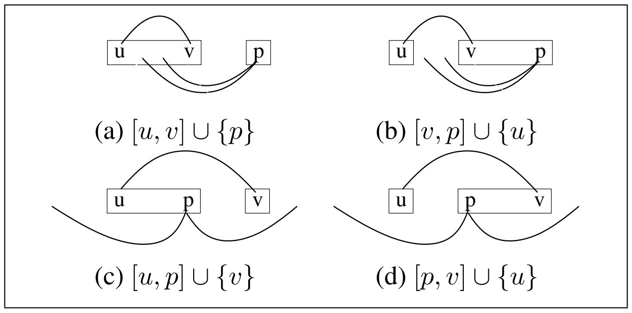


Figure 2: An edge e_{uv} and $\mathcal{P}t(e_{uv}) = p$ form two sets of isolated crossing regions (Lemma 1). 2a and 2b show $p \notin (u, v)$; 2c and 2d show $p \in (u, v)$.

Lemma 1. *Consider any edge* e_{uv} *and* $\mathcal{P}t(e_{uv}) = p$ *in a 1-Endpoint-Crossing forest F. Let l, r, and m denote the leftmost, rightmost, and middle point out of* $\{u, v, p\}$*, respectively. Then the three points u, v, and p define two isolated crossing regions: (1)* $[l, m] \cup \{r\}$*, and (2)* $[m, r] \cup \{l\}$*.*

Proof. First note that as $p = \mathcal{P}t(e_{uv})$, $\mathcal{P}(e_{uv})$ is non-empty: there must be at least one edge between vertices $\in (u, v)$ and vertices $\notin [u, v]$. p is either $\notin [u, v]$ (i.e., $p = l \vee p = r$) or $\in (u, v)$ (i.e., $p = m$):

Case 1: $\mathbf{p = l \vee p = r}$: Assume without loss of generality that $u < v < p$ (i.e., $p = r$).

(a) $[u, v] \cup \{p\}$ **is an isolated crossing region (Figure 2a)**: Condition 1: Assume for the sake of contradiction that there were an edge between a vertex $\in (u, v)$ and a vertex $\notin [u, v] \cup \{p\}$. Then such an edge would cross e_{uv} without having an endpoint at p, which contradicts the 1-Endpoint-Crossing property for e_{uv}.

Condition 2: Assume that for some e_{pa} such that $a \in (u, v)$, e_{pa} was crossed by an edge in the interior of (u, v). The interior edge would not share an endpoint with e_{uv}; since e_{uv} also crosses e_{pa}, this contradicts the 1-Endpoint-Crossing property for e_{pa}.

(b) $[v, p] \cup \{u\}$ **is an isolated crossing region (Figure 2b)**: Condition 1: Assume there were an edge e_{ab} with $a \in (v, p)$ and $b \notin [v, p] \cup \{u\}$. b cannot be in (u, v) (by above). Thus, $b \notin [u, p]$, which implies that e_{ab} crosses the edges in $\mathcal{P}(e_{uv})$; as e_{uv} does not share a vertex with e_{ab}, this contradicts the 1-Endpoint-Crossing property for all edges in $\mathcal{P}(e_{uv})$.

Condition 2: Assume that for some e_{ua} such that $a \in (v, p)$, e_{ua} was crossed by an edge in the interior of (v, p). e_{ua} would also be crossed by all the edges in $\mathcal{P}(e_{uv})$; as the interior edge would not share an endpoint with any of the edges in $\mathcal{P}(e_{uv})$, this would contradict the 1-Endpoint-Crossing property for e_{ua}.

Case 2: $\mathbf{p = m}$:

(a) $[u, p] \cup \{v\}$ **is an isolated crossing region (Figure 2c)**: Condition 1: Assume there were an edge e_{ab} with $a \in (u, p)$ and $b \notin [u, p] \cup \{v\}$ ($b \in (p, v) \vee b \notin [u, v]$). First assume $b \in (p, v)$. Then e_{ab} crosses all edges in $\mathcal{P}(e_{uv})$; as e_{ab} does not share an endpoint with e_{uv}, this contradicts the 1-Endpoint-Crossing property for the edges in $\mathcal{P}(e_{uv})$. Next assume $b \notin [u, v]$. Then e_{ab} crosses e_{uv}; since $a \neq p \wedge b \neq p$, this violates the 1-Endpoint-Crossing property for e_{uv}.

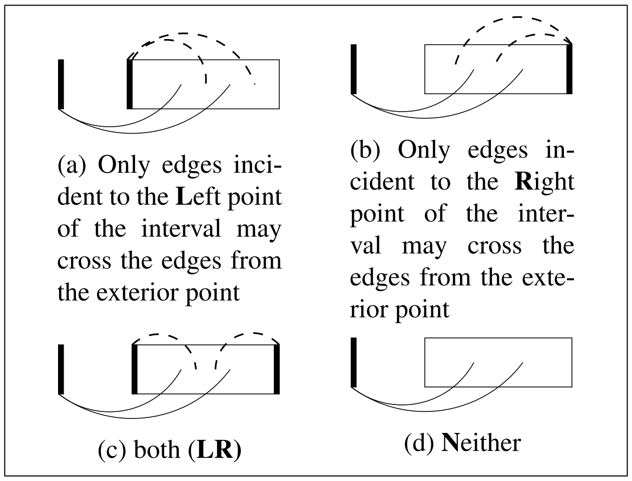
Condition 2: Assume that for some e_{va} with $a \in (u, p)$, e_{va} was crossed by an edge in the interior of (u, v). e_{va} is also crossed by all the edges in $\mathcal{P}(e_{uv})$; as the interior edge will not share an endpoint with the edges in $\mathcal{P}(e_{uv})$, this contradicts the 1-Endpoint-Crossing property for e_{va}.

(b) $[p, v] \cup \{u\}$ **is an isolated crossing region (Figure 2d)**: Symmetric to the above.

□

4 Parsing Algorithm

The optimal 1-Endpoint-Crossing tree can be found using a dynamic programming algorithm that exploits the fact that edges and their crossing points define intervals and isolated crossing regions. This section assumes an arc-factored model, in which the score of a tree is defined as the sum of the scores of its edges; scoring functions for edges are generally learned from data.


(a) Only edges incident to the **L**eft point of the interval may cross the edges from the exterior point

(b) Only edges incident to the **R**ight point of the interval may cross the edges from the exterior point

(c) both (**LR**)

(d) **N**either

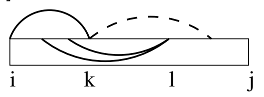
Figure 3: Isolated crossing region sub-problems.

The dynamic program uses five types of sub-problems: interval sub-problems for each interval $[i, j]$, denoted $Int[i, j]$, and four types of isolated crossing region sub-problems for each interval and exterior point $[i, j] \cup \{x\}$, which differ in whether edges from the exterior point may be crossed by edges with an endpoint at the **L**eft point of the interval, the **R**ight point, both **LR**, or **N**either (Figure 3). $L[i, j, x]$, for example, refers to an isolated crossing region over the interval $[i, j]$ with an exterior point of x, in which edges incident to i (the left boundary point) can cross edges between x and (i, j).

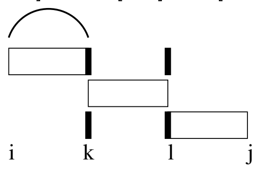
These distinctions allow the 1-Endpoint-Crossing property to be globally enforced; crossing edges in one region may constrain edges in another. For example, consider that Figure 2a allows edges with an endpoint at v to cross the edges from p, while Figure 2b allows edges from u into (v, p). Both *simultaneously* would cause a 1-Endpoint-Crossing violation for the edges in $\mathcal{P}(e_{uv})$. Figures 4 and 5 show valid combinations of the sub-problems in Figure 3.

The full dynamic program is shown in Appendix A. The final answer must be a valid dependency tree, which requires each word to have exactly one parent and prohibits cycles. We use booleans (b_i, b_j, b_x) for each sub-problem, in which the boolean is set to true if and only if the solution to the sub-problem must contain the incoming (parent) edge for the corresponding boundary point. We use the suffix *AFromB* for a sub-problem to enforce that a boundary point A must be descended from boundary point B (to avoid cycles). We will occasionally mention these issues,

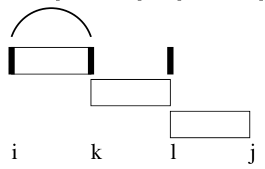
(a) If $l \in (k, j]$:



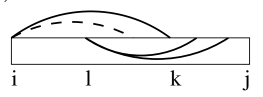
(i) **If the dashed edge exists**:
All the edges from l into (i, k) must choose k as their $\mathcal{P}t$. The interval decomposes into $S[e_{ik}] + R[i, k, l] + Int[k, l] + L[l, j, k]$:



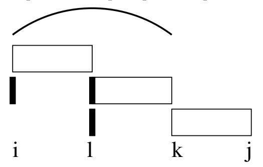
(ii) **If no edges like the dashed edge exist**:
All edges from l into (i, k) may choose either i or k as their $\mathcal{P}t$. The interval decomposes into $S[e_{ik}] + LR[i, k, l] + Int[k, l] + Int[l, j]$:



(b) If $l \in (i, k)$:



(i) **If dashed edge exists**: All the edges from l into $(k, j]$ must choose i as their $\mathcal{P}t$. The interval decomposes into $S[e_{ik}] + Int[i, l] + L[l, k, i] + N[k, j, l]$:



(ii) **If no edges like the dashed edge exist**: All edges from l may choose k as their $\mathcal{P}t$. The interval decomposes into $S[e_{ik}] + R[i, l, k] + Int[l, k] + L[k, j, l]$:

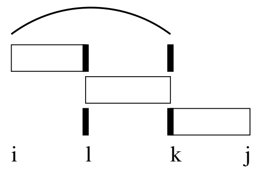


Figure 4: Decomposing an $Int[i, j]$ sub-problem, with $\mathcal{P}t(e_{ik}) = l$

but for simplicity focus the discussion on the decomposition into crossing regions and the maintenance of the 1-Endpoint-Crossing property. Edge direction does not affect these points of focus, and so we will refer simply to $S[e_{uv}]$ to mean the score of either the edge from u to v or vice-versa.

In the following subsections, we show that the optimal parse for each type of sub-problem can be decomposed into smaller valid sub-problems. If we take the maximum over all these possible combinations of smaller solutions, we can find the maximum scoring parse for that sub-problem. Note that the overall tree is a valid sub-problem (over the interval $[0, n]$), so the argument will also hold for finding the optimal overall tree. Each individual vertex and each pair of adjacent vertices (with no edges) trivially form isolated intervals (as there is no interior); this forms the base case of the dynamic program.

The overall dynamic program takes $O(n^4)$ time: there are $O(n^2)$ interval sub-problems, each of which needs two free split points to find the maximum, and $O(n^3)$ region sub-problems, each of which is a maximization over one free split point.

4.1 Decomposing an Int sub-problem

Consider an isolated interval sub-problem $Int[i, j]$. There are three cases: (1) there are no edges between i and the rest of the interval, (2) the longest edge incident to i is not crossed, (3) the longest edge incident to i is crossed. An Int sub-problem can be decomposed into smaller valid sub-problems in each of these three cases. Finding the optimal Int forest can be done by taking the maximum over these cases:

No edges between i and $[i + 1, j]$: The same set of edges is also a valid $Int[i + 1, j]$ sub-problem. b_i must be true for the $Int[i + 1, j]$ sub-problem to ensure $i + 1$ receives a parent.

Furthest edge from i is not crossed: If the furthest edge is to j, the problem can be decomposed into $S[e_{ij}] + Int[i, j]$, as that edge has no effect on the interior of the interval. Clearly, this is only applicable if the boundary point needed a parent (as indicated by the booleans) and the boolean must then be updated accordingly. If the furthest edge is to some k in (i, j), the problem is decomposed into $S[e_{ik}] + Int[i, k] + Int[k, j]$.

Furthest edge from i is crossed: This is the most interesting case, which uses two split points: the other endpoint of the edge (k), and $l = \mathcal{P}t(e_{ik})$. The dynamic program depends on the order of k and l.

$\mathbf{l} \notin (\mathbf{i}, \mathbf{k})$ **(Figure 4a)**: By Lemma 1, $[i, k] \cup \{l\}$ and $[k, l] \cup \{i\}$ form isolated regions. $(l, j]$ is the remainder of the interval, and the only vertex from $[i, l)$ that can have edges into $(l, j]$ is k: (i, k) and (k, l) are part of isolated regions, and i is ruled out because k was i's furthest neighbor.

If at least one edge from k into $(l, j]$ (the dashed line in Figure 4a) exists, the decomposition is as in Figure 4a, Case i; otherwise, it is as in Figure 4a, Case ii. In Case i, e_{ik} and the edge(s) between k and $(l, j]$ force all of the edges between l and (i, k) to have k as their $\mathcal{P}t$. Thus, the region $[i, k] \cup \{l\}$ must be a sub-problem of type R (Figure 3b), as these edges from l can only be crossed by edges with an endpoint at k (the right endpoint of $[i, k]$). All of the edges between k and $(l, j]$ have l as their $\mathcal{P}t$, as they are crossed by all the edges in $\mathcal{P}(e_{ik})$, and so the sub-problem corresponding to the region $[l, j] \cup \{k\}$ is of type L (Figure 3a). In Case ii, each of the edges in $\mathcal{P}(e_{ik})$ may choose either i or k as their $\mathcal{P}t$, so the sub-problem $[i, k] \cup \{l\}$ is of type LR (Figure 3c). Note that $l = j$ is a special case of Case ii in which the rightmost interval $Int[l, j]$ is empty.

$\mathbf{l} \in (\mathbf{i}, \mathbf{k})$ **(Figure 4b)**: $[i, l] \cup \{k\}$ and $[l, k] \cup \{i\}$ form isolated crossing regions by Lemma 1. There cannot both be edges between i and (l, k) and between k and (i, l), as this would violate 1-Endpoint-Crossing for the edges in $\mathcal{P}(e_{ik})$. If there are *any* edges between i and (l, k) (i.e., Case i in Figure 4b), then all of the edges in $\mathcal{P}(e_{ik})$ must choose i as their $\mathcal{P}t$, and so these edges cannot be crossed at all in the region $[k, j] \cup \{l\}$, and there cannot be any edges from k into (i, l). If there are no such edges (Case ii in 4b), then k must be a valid $\mathcal{P}t$ for all edges in $\mathcal{P}(e_{ik})$, and so there can both be edges from k into (i, l) and $[k, j] \cup \{l\}$ may be of type L (allowing crossings with an endpoint at k).

4.2 Decomposing an LR sub-problem

An LR sub-problem is over an isolated crossing region $[i, j] \cup \{x\}$, such that edges from x into (i, j) may be crossed by edges with an endpoint at either i or j. This sub-problem is only defined when neither i nor j get their parent from this sub-problem. From a top-down perspective, this case is only used when there will be an edge between i and j (as in one of the sub-problems in Figure 4a, Case ii).

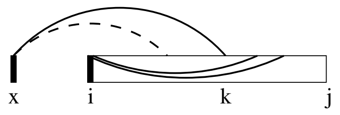
If none of the edges from x are crossed by any edges with an endpoint at i, this can be considered an R problem. Similarly, if none are crossed by any edges with an endpoint at j, this may be considered an L sub-problem. The only case which needs discussion is when both edges with an endpoint at i and also at j cross edges from x; see Figure 3c for a schematic. In that scenario, there must exist a split point such that: (1) to the left of the point, all edges crossing x-edges have an endpoint at i, and to the right of the point, all such edges have an endpoint at j, and (2) no edges in the region cross the split point.

Let r_i be i's rightmost child in (i, j); let l_j be j's leftmost child in (i, j). Every edge from x into (i, r_i) is crossed by e_{ir_i}; every edge between x and (l_j, j) is crossed by $e_{l_j j}$. e_{ir_i} cannot cross $e_{l_j j}$, as that would either violate 1-Endpoint-Crossing (because of the x-interior edges) or create a cycle (if both children are also connected by an edge to x). r_i and l_j also cannot be equal: as neither i nor j may be assigned a parent, they must both be in the direction of the child, and the child cannot have multiple parents. Thus, r_i is to the left of l_j.

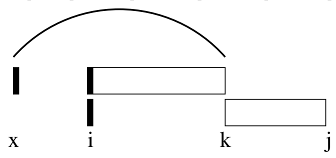
Any split point between r_i and l_j clearly satisfies (1). There is at least one point within $[r_i, l_j]$ that satisfies (2) as long as there is not a chain of crossing edges from e_{ir_i} to $e_{l_j j}$. The proof is omitted for space reasons, but such a chain can be ruled out using a counting argument similar to that in the proof in Section 5.1. The decomposition is: $L[i, k, x] + R[k, j, x]$ for some $k \in (i, j)$.

4.3 Decomposing an N sub-problem

Consider the maximum scoring forest of type N over $[i, j] \cup \{x\}$ (Figure 3d; *no* edges from x are crossed in this sub-problem). If there are no edges from x, then it is also a valid $Int[i, j]$ sub-problem. If there are edges between x and the endpoints i or j, then the forest with that edge removed is still a valid N sub-problem (with the ancestor and parent bookkeeping updated). Otherwise, if there are edges between x and (i, j), choose the neighbor of x closest to j (call it k). Since the edge e_{xk} is not crossed, there are no edges from $[i, k)$ into $(k, j]$; since k was the neighbor of x closest to j, there are no edges from x into $(k, j]$. Thus, the region decomposes into



(i) **If dashed edge exists**: All the edges from i into $(k, j]$ must choose x as their $\mathcal{P}t$. The interval decomposes into $S[e_{xk}] + L[i, k, x] + N[k, j, i]$:



(ii) **If no edges like the dashed edge exist**: Edges from i into $(k, j]$ may choose k as their $\mathcal{P}t$. The interval decomposes into $S[e_{xk}] + Int[i, k] + L[k, j, i]$:

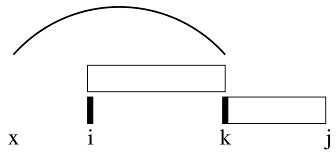


Figure 5: An L sub-problem over $[i, j] \cup \{x\}$, k is the neighbor of x furthest from i in the interval.

$S[e_{ik}] + Int[k, j] + N[i, k, x]$.

As an aside, if b_x was true (x needed a parent from this sub-problem), and k was a child of x, then x's parent must come from the $[i, k] \cup \{x\}$ sub-problem. However, it cannot be a descendant of k, as that would cause a cycle. Thus in this case, we call the sub-problem a N_XFromI problem, to indicate that x needs a parent, i and k do not, and x must be descended from i, not k.

4.4 Decomposing an L or R sub-problem

An L sub-problem over $[i, j] \cup \{x\}$ requires that any edges in this region that cross an edge with an endpoint at x have an endpoint at i (the *left* endpoint). If there are no edges between x and $[i, j]$ in an L sub-problem, then it is also a valid Int sub-problem over $[i, j]$. If there are edges between x and i or j, then the sub-problem can be decomposed into that edge plus the rest of the forest with that edge removed.

The interesting case is when there are edges between x and the interior (Figure 5). Let k be the neighbor of x within (i, j) that is furthest from i. As all edges that cross e_{xk} will have an endpoint at i, there are no edges between (i, k) and $(k, j]$. Combined with the fact that k was the neighbor of x closest to j, we have that $[i, k] \cup \{x\}$ must form an isolated crossing region, as must $[k, j] \cup \{i\}$.

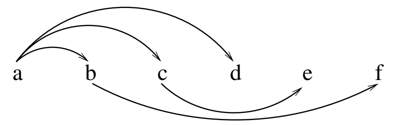


Figure 6: 2-planar but not 1-Endpoint-Crossing

If there are additional edges between x and the interior (Case i in 5), all of the edges from i into $(k, j]$ cross both the edge e_{xk} and the other edges from x into (i, k). The $\mathcal{P}t$ for all these edges must therefore be x, and as x is not in the region $[k, j] \cup \{i\}$, those edges cannot be crossed at all in that region (i.e., $[k, j] \cup \{i\}$ must be of type N). If there are no additional edges from x into (i, k) (Case ii in Figure 5), then all of the edges from i into (k, j) must choose either x or k as their $\mathcal{P}t$. As there will be no more edges from x, choosing k as their $\mathcal{P}t$ allows strictly more trees, and so $[k, j] \cup \{i\}$ can be of type L (allowing edges from i to be crossed in that region by edges with an endpoint at k).

An R sub-problem is identical, with k instead chosen to be the neighbor of x furthest from j.

5 Connections

5.1 Graph Theory: All 1-Endpoint-Crossing Trees are 2-Planar

The 2-planar characterization of dependency structures in Gómez-Rodríguez and Nivre (2010) exactly correspond to *2-page book embeddings* in graph theory: an embedding of the vertices in a graph onto a line (by analogy, along the *spine* of a book), and the edges of the graph onto one of 2 (more generally, k) half-planes (*pages* of the book) such that no edges on the same page cross (Bernhart and Kainen, 1979). The problem of finding an embedding that minimizes the number of pages required is a natural formulation of many problems arising in disparate areas of computer science, for example, sorting a sequence using the minimum number of stacks (Even and Itai, 1971), or constructing fault-tolerant layouts in VLSI design (Chung et al., 1987).

In this section we prove *1-Endpoint-Crossing* $\subseteq$ *2-planar*. These classes are not equal (Figure 6). We first prove some properties about the *crossings graphs* (Gómez-Rodríguez and Nivre, 2010) of 1-Endpoint-Crossing trees. The crossings graph of a

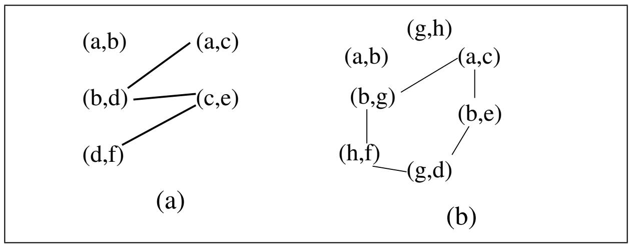


Figure 7: The crossing graphs for Figures 1a and 1b.

graph has a vertex corresponding to each edge in the original, and an edge between two vertices if the two edges they correspond to cross. The crossings graphs for the dependency trees in Figures 1a and 1b are shown in Figures 7a and 7b, respectively.

Lemma 2. *No 1-Endpoint-Crossing tree has a cycle of length* 3 *in its crossings graph.*

Proof. Assume there existed a cycle e_1, e_2, e_3. e_1 and e_3 must share an endpoint, as they both cross e_2. Since e_1 and e_3 share an endpoint, e_1 and e_3 do not cross. Contradiction. □

Lemma 3. *Any odd cycle of size* n *(*$n \geq 4$*) in a crossings graph of a 1-Endpoint-Crossing tree uses at most* n *distinct vertices in the original graph.*

Proof. Let e_1, e_2, ..., e_n be an odd cycle in a crossings graph of a 1-Endpoint-Crossing tree with $n \geq 4$. Since $n \geq 4$, e_1, e_2, e_{n-1}, and e_n are distinct edges. Let a be the vertex that e_1 and e_{n-1} share (because they both cross e_n) and let b be the vertex that e_2 and e_n share (both cross e_1). Note that e_1 and e_{n-1} cannot contain b and that e_2 and e_n cannot contain a (otherwise they would not cross an edge adjacent to them along the cycle).

We will now consider how many vertices each edge can introduce that are distinct from all vertices previously seen in the cycle. e_1 and e_2 necessarily introduce two distinct vertices each.

Let e_o be the first odd edge that contains b (we know one exists since e_n contains b). (o is at least 3, since e_1 does not contain b.) e_o's other vertex must be the one shared with e_{o-2} (e_{o-2} does not contain b, since e_o was the first odd edge to contain b). Therefore, both of e_o's vertices have already been seen along the cycle.

Similarly, let e_e be the first even edge that contains an a. By the same reasoning, e_e must not introduce any new vertices.

All other edges e_i such that $i > 2$ and $e_i \neq e_o$ and $e_i \neq e_e$ introduce at most one new vertex, since one must be shared with the edge e_{i-2}. There are $n - 4$ such edges.

Counting up all possibilities, the maximum number of distinct vertices is $4 + (n - 4) = n$. □

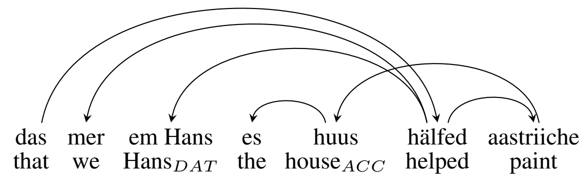
Theorem 1. *1-Endpoint-Crossing trees* $\subseteq$ *2-planar.*

Proof. Assume there existed an odd cycle in the crossings graph of a 1-Endpoint-Crossing tree. The cycle has size at least 5 (by Lemma 2). There are at least as many edges as vertices in the subgraph of the forest induced by the vertices used in the cycle (by Lemma 3). That implies the existence of a cycle in the original graph, contradicting that the original graph was a tree.

Since there are no odd cycles in the crossings graph, the crossings graph of edges is bipartite. Each side of the bipartite graph can be assigned to a page, such that no two edges on the same page cross. Therefore, the original graph was 2-planar. □

5.2 Linguistics: Cross-serial Verb Constructions and Successive Cyclicity

Cross-serial verb constructions were used to provide evidence for the "non-context-freeness" of natural language (Shieber, 1985). Cross-serial verb constructions with two verbs form 1-Endpoint-Crossing trees. Below is a cross-serial sentence from Swiss-German, from (1) in Shieber (1985):



The edges $(\textit{that}, \textit{helped})$, $(\textit{helped}, \textit{we})$, and $(\textit{helped}, \textit{Hans})$ are each only crossed by an edge with an endpoint at *paint*; the edge $(\textit{paint}, \textit{house})$ is only crossed by edges with an endpoint at *helped*. More generally, with a set of two cross serial verbs in a subordinate clause, each verb should suffice as the crossing point for all edges incident to the other verb that are crossed.

Cross-serial constructions with three or more verbs would have dependency trees that violate 1-

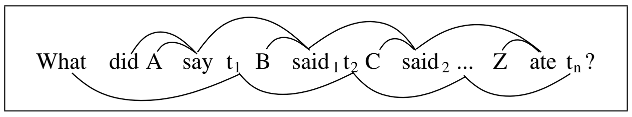


Figure 8: An example of *wh-movement* over a potentially unbounded number of clauses. The edges between the heads of each clause cross the edges from trace to trace, but all obey 1-Endpoint-Crossing.

Endpoint-Crossing. Psycholinguistically, between two and three verbs is exactly where there is a large change in the sentence processing abilities of *human* listeners (based on both grammatical judgments and scores on a comprehension task) (Bach et al., 1986).

More speculatively, there may be a connection between the form of 1-Endpoint-Crossing trees and *phases* (roughly, propositional units such as clauses) in Minimalism (Chomsky et al., 1998). Figure 8 shows an example of *wh-movement* over a potentially unbounded number of clauses. The *phase-impenetrability condition* (PIC) states that only the head of the phase and elements that have moved to its edge are accessible to the rest of the sentence (Chomsky et al., 1998, p.22). Movement is therefore required to be *successive cyclic*, with a moved element leaving a *chain* of traces at the edge of each clause on its way to its final pronounced location (Chomsky, 1981). In Figure 8, notice that the crossing edges form a repeated pattern that obeys the 1-Endpoint-Crossing property. More generally, we suspect that trees satisfying the PIC will tend to also be 1-Endpoint-Crossing. Furthermore, if the traces were *not* at the edge of each clause, and instead were positioned between a head and one of its arguments, 1-Endpoint-Crossing would be violated. For example, if t_2 in Figure 8 were between *C* and *said*$_2$, then the edge (t_1, t_2) would cross $(say, said_1)$, $(said_1, said_2)$, and $(C, said_2)$, which do not all share an endpoint. An exploration of these linguistic connections may be an interesting avenue for further research.

6 Conclusions

1-Endpoint-Crossing trees characterize over 95% of structures found in natural language treebank, and can be parsed in only a factor of n more time than projective trees. The dynamic programming algorithm for projective trees (Eisner, 2000) has been extended to handle higher order factors (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010), adding at most a factor of n to the edge-based running time; it would be interesting to extend the algorithm presented here to include higher order factors. 1-Endpoint-Crossing is a condition on *edges*, while properties such as well-nestedness or block degree are framed in terms of *subtrees*. Three edges will always suffice as a certificate of a 1-Endpoint-Crossing violation (two vertex-disjoint edges that both cross a third). In contrast, for a property like ill-nestedness, two nodes might have a least common ancestor arbitrarily far away, and so one might need the entire graph to verify whether the sub-trees rooted at those nodes are disjoint and ill-nested. We have discussed cross-serial dependencies; a further exploration of which linguistic phenomena would and would not have 1-Endpoint-Crossing dependency trees may be revealing.

Acknowledgments

We would like to thank Julie Legate for an interesting discussion. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship, NSF Award CCF 1137084, and Army Research Office MURI grant W911NF-07-1-0216.

A Dynamic Program to find the maximum scoring 1-Endpoint-Crossing Tree

Input: Matrix S: $S[i,j]$ is the score of the directed edge (i,j)
Output: Maximum score of a 1-Endpoint-Crossing tree over vertices $[0,n]$, rooted at 0
Init: $\forall i \quad Int[i,i,F,F] = Int[i,i+1,F,F] = 0$
$Int[i,i,T,F] = Int[i,i,F,T] = Int[i,i,T,T] = -\infty$
Final: $Int[0,n,F,T]$
Shorthand for booleans: $\mathcal{TF}(x,S) :=$
 if $x = T$, exactly one of the set S is true
 if $x = F$, all of the set S must be false
b_i, b_j, b_x are true iff the corresponding boundary point has its incoming edge (parent) in that sub-problem. For the LR sub-problem, b_i and b_j are always false, and so omitted. For all sub-problems with the suffix $AFromB$, the boundary point A has its parent edge in the sub-problem solution; the other two boundary points do not. For example, L_XFromI would correspond to having booleans $b_i = b_j = F$ and $b_x = T$, with the restriction that x must be a descendant of i.

$$\mathit{Int}[i,j,F,b_j] \leftarrow \max \begin{cases} \mathit{Int}[i+1,j,T,F] & \text{if } b_j = F \\ S[i,j] + \mathit{Int}[i,j,F,F] & \text{if } b_j = T \\ \max\limits_{k\in(i,j)} S[i,k] + \\ \begin{cases} \mathit{Int}[i,k,F,F] + \mathit{Int}[k,j,F,b_j] \\ \max\limits_{\mathcal{TF}(b_j,\{b_l,b_r\})} \mathit{LR}[i,k,j,b_l] + \mathit{Int}[k,j,F,b_r] \\ \max_{l\in(k,j),\mathcal{TF}(T,\{b_l,b_m,b_r\})} \\ \begin{cases} R[i,k,l,F,F,b_l] + \mathit{Int}[k,l,F,b_m] + L[l,j,k,b_r,b_j,F] \\ \mathit{LR}[i,k,l,b_l] + \mathit{Int}[k,l,F,b_m] + \mathit{Int}[l,j,b_r,b_j] \end{cases} \\ \max_{l\in(i,k),\mathcal{TF}(T,\{b_l,b_m,b_r\})} \\ \begin{cases} \mathit{Int}[i,l,F,b_l] + L[l,k,i,b_m,F,F] + N[k,j,l,F,b_j,b_r] \\ R[i,l,k,F,b_l,F] + \mathit{Int}[l,k,b_m,F] + L[k,j,l,F,b_j,b_r] \end{cases} \end{cases} \end{cases}$$

$\mathit{Int}[i,j,T,F] \leftarrow$ symmetric to $\mathit{Int}[i,j,F,T]$
$\mathit{Int}[i,j,T,T] \leftarrow -\infty$

$$\mathit{LR}[i,j,x,b_x] \leftarrow \max \begin{cases} L[i,j,x,F,F,b_x] \\ R[i,j,x,F,F,b_x] \\ \max_{k\in(i,j),\mathcal{TF}(b_x,\{b_{xl},b_{xr}\}),\mathcal{TF}(T,\{b_{kl},b_{kr}\})} \\ \quad L[i,k,x,F,b_{kl},b_{xl}] + R[k,j,x,b_{kr},F,b_{xr}] \end{cases}$$

$$N[i,j,x,b_i,b_j,F] \leftarrow \max \begin{cases} \mathit{Int}[i,j,b_i,b_j] \\ S[x,i] + N[i,j,x,F,b_j,F] & \text{if } b_i = T \\ S[x,j] + N[i,j,x,b_i,F,F] & \text{if } b_j = T \\ \max\limits_{k\in(i,j)} S[x,k] + N[i,k,x,b_i,F,F] + \mathit{Int}[k,j,F,b_j] \end{cases}$$

$$N[i,j,x,F,b_j,T] \leftarrow \max \begin{cases} S[i,x] + N[i,j,x,F,b_j,F] \\ S[x,j] + \mathit{N\_XFromI}[i,j,x] & \text{if } b_j = T \\ S[j,x] + N[i,j,x,F,F,F] & \text{if } b_j = F \\ S[j,x] + \mathit{Int}[i,j,F,T] & \text{if } b_j = T \\ \max\limits_{k\in(i,j)} S[x,k] + \mathit{N\_XFromI}[i,k,x] + \mathit{Int}[k,j,F,b_j] \\ \max\limits_{k\in(i,j)} S[k,x] + \\ \begin{cases} \mathit{Int}[i,k,F,T] + \mathit{Int}[k,j,F,b_j] \\ N[i,k,x,F,F,F] + \mathit{Int}[k,j,T,b_j] \end{cases} \end{cases}$$

$N[i,j,x,T,F,T] \leftarrow$ symmetric to $N[i,j,x,F,T,T]$
$N[i,j,x,T,T,T] \leftarrow -\infty$

$$\mathit{N\_XFromI}[i,j,x] \leftarrow \max \begin{cases} S[i,x] + N[i,j,x,F,F,F] \\ \max_{k\in(i,j)} \\ \begin{cases} S[x,k] + \mathit{N\_XFromI}[i,k,x] + \mathit{Int}[k,j,F,F] \\ S[k,x] + \mathit{Int}[i,k,F,T] + \mathit{Int}[k,j,F,F] \end{cases} \end{cases}$$

$$\mathit{N\_IFromX}[i,j,x] \leftarrow \max \begin{cases} S[x,i] + N[i,j,x,F,F,F] \\ \max\limits_{k\in(i,j)} S[x,k] + N[i,k,x,T,F,F] + \mathit{Int}[k,j,F,F] \end{cases}$$

$\mathit{N\_XFromJ}[i,j,x] \leftarrow$ symmetric to $\mathit{N\_XFromI}[i,j,x]$
$\mathit{N\_JFromX}[i,j,x] \leftarrow$ symmetric to $\mathit{N\_IFromX}[i,j,x]$

$$L[i,j,x,b_i,b_j,F] \leftarrow \max \begin{cases} \mathit{Int}[i,j,b_i,b_j] \\ S[x,i] + L[i,j,x,F,b_j,F] & \text{if } b_i = T \\ S[x,j] + L[i,j,x,b_i,F,F] & \text{if } b_j = T \\ \max\limits_{k\in(i,j),\mathcal{TF}(b_i,\{b_l,b_r\})} S[x,k] + \\ \begin{cases} L[i,k,x,b_l,F,F] + N[k,j,i,F,b_j,b_r] \\ \mathit{Int}[i,k,b_l,F] + L[k,j,i,F,b_j,b_r] \end{cases} \end{cases}$$

$$L[i,j,x,F,b_j,T] \leftarrow \max \begin{cases} S[i,x] + L[i,j,x,F,b_j,F] \\ S[x,j] + \mathit{L\_XFromI}[i,j,x] & \text{if } b_j = T \\ S[j,x] + L[i,j,x,F,F,F] & \text{if } b_j = F \\ S[j,x] + \mathit{L\_JFromI}[i,j,x] & \text{if } b_j = T \\ \max\limits_{k\in(i,j)} S[x,k] + \mathit{L\_XFromI}[i,k,x] + N[k,j,i,F,b_j,F] \\ \max\limits_{k\in(i,j)} S[k,x] + \\ \begin{cases} \mathit{L\_JFromI}[i,k,x] + N[k,j,i,F,b_j,F] \\ L[i,k,x,F,F,F] + N[k,j,i,T,b_j,F] \\ \max\limits_{\mathcal{TF}(T,\{b_l,b_r\})} \mathit{Int}[i,k,F,b_l] + L[k,j,i,b_r,b_j,F] \end{cases} \end{cases}$$

$L[i,j,x,T,b_j,T] \leftarrow$ not reachable

$$\mathit{L\_XFromI}[i,j,x] \leftarrow \max \begin{cases} S[i,x] + L[i,j,x,F,F,F] \\ \max\limits_{k\in(i,j)} S[x,k] + \mathit{L\_XFromI}[i,k,x] + N[k,j,i,F,F,F] \\ \max\limits_{k\in(i,j)} S[k,x] + \\ \begin{cases} \mathit{L\_JFromI}[i,k,x] + N[k,j,i,F,F,F] \\ L[i,k,x,F,F,F] + \mathit{N\_IFromX}[k,j,i] \\ \mathit{Int}[i,k,F,T] + L[k,j,i,F,F,F] \\ \mathit{Int}[i,k,F,F] + \mathit{L\_IFromX}[k,j,i] \end{cases} \end{cases}$$

$$\mathit{L\_IFromX}[i,j,x] \leftarrow \max \begin{cases} S[x,i] + L[i,j,x,F,F,F] \\ \max\limits_{k\in(i,j)} S[x,k] + \\ \begin{cases} L[i,k,x,T,F,F] + N[k,j,i,F,F,F] \\ L[i,k,x,F,F,F] + \mathit{N\_XFromI}[k,j,i] \\ \mathit{Int}[i,k,T,F] + L[k,j,i,F,F,F] \\ \mathit{Int}[i,k,F,F] + \mathit{L\_XFromI}[k,j,i] \end{cases} \end{cases}$$

$$\mathit{L\_JFromX}[i,j,x] \leftarrow \max \begin{cases} S[x,j] + L[i,j,x,F,F,F] \\ \max\limits_{k\in(i,j)} S[x,k] + \\ \begin{cases} L[i,k,x,F,F,F] + \mathit{Int}[k,j,F,T] \\ \mathit{Int}[i,k,F,F] + \mathit{L\_JFromI}[k,j,i] \end{cases} \end{cases}$$

$$\mathit{L\_JFromI}[i,j,x] \leftarrow \max \begin{cases} \mathit{Int}[i,j,F,T] \\ \max\limits_{k\in(i,j)} S[x,k] + \\ \begin{cases} L[i,k,x,F,F,F] + \mathit{N\_JFromX}[k,j,i] \\ \mathit{Int}[i,k,F,F] + \mathit{L\_JFromX}[k,j,i] \end{cases} \end{cases}$$

$R[i,j,x,b_i,b_j,F] \leftarrow$ symmetric to $L[i,j,x,b_i,b_j,F]$
$R[i,j,x,b_i,F,T] \leftarrow$ symmetric to $L[i,j,x,F,b_j,T]$
$R[i,j,x,b_i,T,T] \leftarrow$ not reachable
$R\_XFromJ[i,j,x] \leftarrow$ symmetric to $L\_XFromI[i,j,x]$
$R\_JFromX[i,j,x] \leftarrow$ symmetric to $L\_IFromX[i,j,x]$
$R\_IFromX[i,j,x] \leftarrow$ symmetric to $L\_JFromX[i,j,x]$
$R\_IFromJ[i,j,x] \leftarrow$ symmetric to $L\_JFromI[i,j,x]$

References

E. Bach, C. Brown, and W. Marslen-Wilson. 1986. Crossed and nested dependencies in german and dutch: A psycholinguistic study. *Language and Cognitive Processes*, 1(4):249–262.

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

M. Bodirsky, M. Kuhlmann, and M. Möhl. 2005. Well-nested drawings as models of syntactic structure. In *In Tenth Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*, pages 88–1. University Press.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, volume 7, pages 957–961.

N. Chomsky, Massachusetts Institute of Technology. Dept. of Linguistics, and Philosophy. 1998. *Minimalist inquiries: the framework*. MIT occasional papers in linguistics. Distributed by MIT Working Papers in Linguistics, MIT, Dept. of Linguistics.

N. Chomsky. 1981. *Lectures on Government and Binding*. Dordrecht: Foris.

F. Chung, F. Leighton, and A. Rosenberg. 1987. Embedding graphs in books: A layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58.

H. Cui, R. Sun, K. Li, M.Y. Kan, and T.S. Chua. 2005. Question answering passage retrieval using dependency relations. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 400–407. ACM.

A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 423. Association for Computational Linguistics.

Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 541–548. Association for Computational Linguistics.

J. Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers, October.

S. Even and A. Itai. 1971. Queues, stacks, and graphs. In *Proc. International Symp. on Theory of Machines and Computations*, pages 71–86.

C. Gómez-Rodríguez and J. Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of ACL*, pages 1492–1501.

C. Gómez-Rodríguez, J. Carroll, and D. Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37(3):541–586.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*, pages 1–11.

M. Kuhlmann. 2013. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2).

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88.

R. McDonald and G. Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 121–132.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

E. Pitler, S. Kannan, and M. Marcus. 2012. Dynamic programming for higher order parsing of gap-minding trees. In *Proceedings of EMNLP*, pages 478–488.

L.A. Ringenberg. 1967. *College geometry*. Wiley.

A. Rush and S. Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of NAACL*, pages 498–507.

S.M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343.

H. Zhang and R. McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of EMNLP*, pages 320–331.