MODELS FOR IMPROVED TRACTABILITY AND ACCURACY IN DEPENDENCY PARSING

Emily Pitler

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2013

Supervisor of Dissertation

Mitchell P. Marcus
Professor, Computer and Information Science

Co-Supervisor of Dissertation

Sampath Kannan
Professor, Computer and Information Science

Graduate Group Chairperson

Val Tannen, Professor, Computer and Information Science

Dissertation Committee:

Michael Collins, Professor, Computer Science, Columbia University

Chris Callison-Burch, Assistant Professor, Computer and Information Science

Mark Liberman, Professor, Linguistics

Ben Taskar, Associate Professor, Computer and Information Science

MODELS FOR IMPROVED TRACTABILITY AND ACCURACY IN DEPENDENCY PARSING

Acknowledgements

Mitch Marcus and Sampath Kannan were wonderful advisors and I am truly grateful to both of them for taking me on as a student. Mitch's patience and unwavering support allowed me the freedom to find this topic for this thesis. His wide-ranging knowledge of both linguistics and computer science was incredibly valuable. Sampath inspired me to strive for clean solutions to problems. His clarity helped me to identify the crucial pieces of any result and greatly improved the material presented here.

I am grateful to my thesis committee: Chris Callison-Burch, Mike Collins, Mark Liberman, and Ben Taskar. I have benefited from Chris's advice throughout graduate school, at both Penn and JHU. I have tried to emulate his clear presentation style in both my talks and my writing. Mike's work in parsing served as inspiration for me for much of the work in this thesis. His detailed questions have both improved this document and pointed the way towards future directions. Mark's questions at CLunch over the years always brought up subtle but important details. Discussions of treebank representations in this document are largely due to Mark's influence. Ben helped me identify interesting directions at a formative stage in this work. Throughout the process, Ben encouraged me to simultaneously investigate both theoretical and empirical issues.

A large number of people contributed to my undertaking this dissertation and following through. Thanks especially to Jerry Berry and Stephen Rose at Thomas Jefferson High School for Science and Technology for introducing me to programming and computer science; Charles Yang, at Yale and Penn, for introducing me to computational linguistics; Dana Angluin and Brian Scassellati at Yale for encouraging me to pursue graduate school;

and Ken Church, Dekang Lin, and Ani Nenkova for guiding me as I started research. I benefited tremendously from two summers with Ken: he patiently taught me how to write papers and how to approach problems from multiple perspectives. Dekang has been very supportive of me and my research ever since working together at a JHU CLSP Summer Workshop in 2009. Dekang introduced me to research in parsing and applications of parsing. Ani Nenkova and Annie Louis were wonderful collaborators. Annie Louis was my constant partner throughout graduate school; it was a pleasure to go through courses, research, internships, and other milestones together. Thanks to Katerina Fragkiadaki, Jenny Gillenwater, Arun Raghavan, and David Weiss for providing feedback on drafts and practice talks; Shane Bergsma, Jason Eisner, Aravind Joshi, Julie Legate, Ryan McDonald, Slav Petrov, Giorgio Satta, and David Yarowsky for productive conversations that impacted this thesis; Terry Koo for his assistance with the dpo3 parser; Daniel Zeman for his assistance with the HamleDT treebank conversion software; Mike Felker for removing administrative hurdles; Cheryl Hickey for all her support; the students, staff, and volunteers at White-Williams Scholars/Philadelphia Futures for a rewarding time and memories; Drew Hilton, Santosh Nagarakatte, and Arun Raghavan for long games of bridge; Katie Gibson for baking delicious cupcakes and fudge; and other students I had the pleasure of overlapping with at Penn, including Adam Aviv, Alexis Baird, John Blitzer, Chris Casinghino, Erwin Chan, Sanjian Chen, Nikhil Dinesh, Mark Dredze, Ryan Gabbard, Kuzman Ganchev, João Graça, Michael Greenberg, Kai Hong, Liang Huang, Marie Jacob, Sarah Johnstone, Andrew King, Alex Kulesza, Junyi Li, Constantine Lignos, Xi Lin, Ellie Pavlick, Alex Roderer, Sudeepa Roy, Ben Sapp, Partha Pratim Talukdar, Andrew West, and Qiuye Zhao.

I am deeply grateful to my college friends: Alison, Betsy, Caroline, Charlie, Guy, Illana, Jackie, Joel, Julia, Lekshmi, Michael, and Samarth for their visits and encouragement. Samarth and I were classmates from middle school through undergrad and we wrote our first paper together. I would like to thank my brother Will, my grandparents, and Sudipto for their steadfast belief in me. Most importantly, I am thankful to my parents for nurturing my curiosity and for their love and support.

ABSTRACT

MODELS FOR IMPROVED TRACTABILITY AND ACCURACY IN DEPENDENCY PARSING

Emily Pitler

Mitchell P. Marcus

Sampath Kannan

Automatic syntactic analysis of natural language is one of the fundamental problems in natural language processing. Dependency parses (directed trees in which edges represent the syntactic relationships between the words in a sentence) have been found to be particularly useful for machine translation, question answering, and other practical applications.

For English dependency parsing, we show that models and features compatible with how conjunctions are represented in treebanks yield a parser with state-of-the-art overall accuracy and substantial improvements in the accuracy of conjunctions.

For languages other than English, dependency parsing has often been formulated as either searching over trees without any crossing dependencies (projective trees) or searching over all directed spanning trees. The former sacrifices the ability to produce many natural language structures; the latter is NP-hard in the presence of features with scopes over siblings or grandparents in the tree.

This thesis explores alternative ways to simultaneously produce crossing dependencies in the output and use models that parametrize over multiple edges.

Gap inheritance is introduced in this thesis and quantifies the nesting of subtrees over intervals. The thesis provides $O(n^6)$ and $O(n^5)$ edge-factored parsing algorithms for two new classes of trees based on this property, and extends the latter to include grandparent factors.

This thesis then defines 1-Endpoint-Crossing trees, in which for any edge that is crossed, all other edges that cross that edge share an endpoint. This property covers 95.8% or more of dependency parses across a variety of languages. A crossing-sensitive factorization introduced in this thesis generalizes a commonly used third-order factorization (capable of

scoring triples of edges simultaneously).

This thesis provides exact dynamic programming algorithms that find the optimal 1-Endpoint-Crossing tree under either an edge-factored model or this crossing-sensitive third-order model in $O(n^4)$ time, orders of magnitude faster than other mildly non-projective parsing algorithms and identical to the parsing time for projective trees under the third-order model. The implemented parser is significantly more accurate than the third-order projective parser under many experimental settings and significantly less accurate on none.

Contents

List of Tables

List of Illustrations

Chapter 1

Introduction

Automatic syntactic analysis of natural language has been one of the fundamental problems in natural language processing research. *Dependency parses*, directed trees that represents the syntactic structure of natural language sentences, have proven useful for a variety of practical applications, including machine translation (Ding and Palmer, 2005), question answering (Cui, Sun, Li, Kan, and Chua, 2005), and information extraction (Culotta and Sorensen, 2004). See Figure 1.1 for an example of a dependency parse.

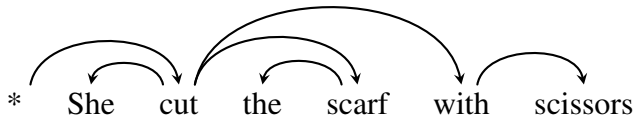


Figure 1.1: A dependency parse tree

There are some theoretical and practical issues, however, that impact the current usefulness of today's dependency parsers:

1. **Low Accuracies on Consequential Parsing Decisions**: Prepositions and conjunctions are two cases that present ambiguities when parsing English, and in fact have been treated as stand-alone tasks (Hindle and Rooth, 1993; Ratnaparkhi, Reynar, and Roukos, 1994; Collins and Brooks, 1995; Goldberg, 1999; Resnik, 1999; Bergsma, Yarowsky, and Church, 2011). Different attachment decisions of prepositions and

conjunctions in an English parse may correspond to different translations in another language, and so correctly attaching these are particularly important for a dependency parser's usefulness. Within English dependency parsing, however, the accuracies for attaching prepositions and conjunctions are well below the overall attachment accuracies.

2. **Intractability of Current Formulations**: Dependency parsing is often cast dichotomously as either searching over trees without any crossing dependencies (Eisner, 2000) or searching over all directed spanning trees (in which any pattern of crossing edges is allowed) (McDonald, Pereira, Ribarov, and Hajič, 2005b). The first approach sacrifices coverage of many natural language structures, especially in languages other than English. The second approach has efficient algorithms (Chu and Liu, 1965; Edmonds, 1967) with very simple statistical models that parametrize over only individual edges, but this problem becomes NP-hard in the presence of factors over siblings or grandparents in the tree (McDonald and Pereira, 2006; McDonald and Satta, 2007), which have been found to greatly improve accuracy in the English case (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010).

In this thesis, we will show how characterizations of dependency tree structures can be used to improve the tractability and accuracy of parsing. In particular, we will show that:

1. Models and features compatible with how linguistic constructions of interest are represented in natural language treebanks lead to state-of-the-art accuracies for English dependency parsing and substantial improvements in the accuracy of conjunctions.

2. Novel definitions of the dependency parsing output space include the vast majority of structures seen in treebanks for a variety of natural languages, tractably allow richer features, and have efficient exact parsing algorithms.

3. A generalization of the grandparent-sibling model that accounts for crossing edges allows a parser to search over the output space mentioned above without any increase in the asymptotic complexity compared with the non-crossing case.

1.1 Background: Framework

A *dependency tree* is a rooted, directed spanning tree that represents a set of dependencies between words in a sentence. The tree has one artificial root node and vertices that correspond to the words in an input sentence w_1, w_2,...,w_n. There is an edge from h to m if m depends on (or modifies) h.

The goal of a dependency parser is to output the "best" dependency parse analysis given an input of a natural language sentence. Note that "outputting the best" requires:

1. **Scoring**: How is "best" defined? For a given tree, what is its score?

2. **Searching**: Given a scoring procedure, how does the parser find the best?

These two questions are coupled: different scoring functions affect the ease of searching, and different search spaces affect what scoring functions can be easily used. Scoring and searching become even more intertwined with a data-driven discriminative approach, in which the scoring function is not given a priori, but is learned from data by repeatedly parsing sentences from the training set, comparing the tree the search procedure found to the gold-standard tree, and then updating the scoring function accordingly.

There are exponentially many dependency trees for a sentence, making it intractable to find the best tree in the presence of arbitrary features that scope over the entire tree. Therefore, one common approach that allows efficient searching is to assume the score of a tree decomposes into the sum of scores of local parts (such as edges or constant-sized local sets of edges). This thesis and a large portion of related work characterize the parsing problem according to the following framework of structured linear models:

$$y^* = argmax_{y \in Y} \sum_{p \in P(y)} w \cdot \phi(p, x) \tag{1.1}$$

where the input x is a sequence of words, each y is a valid dependency tree over the words in x, Y defines the set of all possible dependency tree structures, $P(y)$ defines the set of parts that a given y can be decomposed into, each p is one such part (for example, an edge

of the tree), $\phi(p, x)$ defines a feature vector over a local part p and the input sentence x[1], and w is a weight vector. Related work and our own work here can be characterized by the choices they make for each of these variables.

A parsing algorithm computes the $argmax$ tree under such a model. Various combinatorial algorithms are used to efficiently find the maximum scoring tree y^* within various choices of the search space Y.

We focus on two design decisions here: (i) the choice of the factorization function P (Section 1.2), (ii) the choice of the search space Y (Section 1.3), how these choices can complement each other (Section 1.4) and the implications of these choices for the tractability of the parsing problem and the accuracy of a trained dependency parser.

1.2 Scoring

Scoring a tree requires a factorization function P that defines a set of local tree parts, and a score for each of these parts, based on a feature function ϕ and weight vector w.

1.2.1 Factorizations

Factorizations that have been used in dependency parsing include decompositions of trees into sets of: edges (McDonald, Crammer, and Pereira, 2005a), pairs of edges representing siblings (McDonald and Pereira, 2006), pairs of edges representing siblings and pairs of edges representing outermost grandchildren (Carreras, 2007), and triples of edges representing a grandparent, a parent, and two siblings (Koo and Collins, 2010).

Figures 1.2 and 1.3 show two examples of prepositional phrase attachment (the phrases *with scissors* and *with stripes*), which correctly attach to the verb *cut* and the noun *scarf*, respectively. For each factorization mentioned above, we consider which parts appear in nominal versus verbal attachments in the two examples.

[1] Assuming a discriminatively trained model in which the features can freely condition *globally* over the *input*, but are *locally* constrained over the *output*.

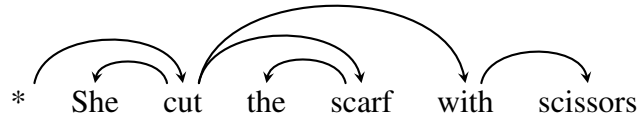


Figure 1.2: The prepositional phrase attaches to the verb.

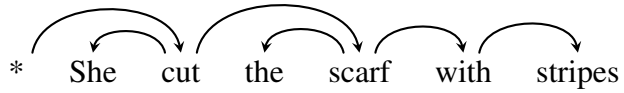


Figure 1.3: The prepositional phrase attaches to the noun.

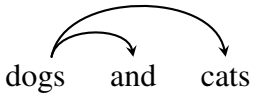
One possibility is that P decomposes any tree y into independent *edges*. In that case, the set of parts relevant to identifying the parent of *with* would be identical in the two sentences: in both cases we would have one potential part $\mathrm{Edge}(\mathit{cut}, \mathit{with})$ and one competing potential part $\mathrm{Edge}(\mathit{scarf}, \mathit{with})$. These parts appear as options in both sentences and therefore the parts alone do not distinguish between the two cases.

Another possibility is that P corresponds to a *sibling* factorization. In this case, the verbal attachment uses the part $\mathrm{Sib}(\mathit{cut}, \mathit{with}, \mathit{scarf})$ (indicating *scarf* is the adjacent inner sibling to *with* and that both modify *cut*), while the noun attachment uses the part $\mathrm{Sib}(\mathit{scarf}, \mathit{with}, \mathit{NULL})$ (indicating *with* is the innermost modifier to *scarf*). Again, both of these parts would appear as options for both of the sentences.

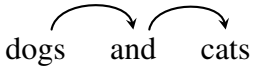
Under a *grandparent* factorization, the set of parts relevant to attaching *with* in the two sentences finally differ. In Figure 1.2, the two potential parts are $\mathrm{Grand}(\mathit{cut}, \mathit{with}, \mathit{scissors})$ versus $\mathrm{Grand}(\mathit{scarf}, \mathit{with}, \mathit{scissors})$, while in Figure 1.3, two relevant potential parts are $\mathrm{Grand}(\mathit{cut}, \mathit{with}, \mathit{stripes})$ and $\mathrm{Grand}(\mathit{scarf}, \mathit{with}, \mathit{stripes})$. If we have features capable of capturing this distinction, we should be able to learn an appropriate weight vector so that $\mathit{Score}(\mathrm{Grand}(\mathit{cut}, \mathit{with}, \mathit{scissors})) > \mathit{Score}(\mathrm{Grand}(\mathit{scarf}, \mathit{with}, \mathit{scissors}))$, and conversely, $\mathit{Score}(\mathrm{Grand}(\mathit{scarf}, \mathit{with}, \mathit{stripes})) > \mathit{Score}(\mathrm{Grand}(\mathit{cut}, \mathit{with}, \mathit{stripes}))$.

The above example motivated why we might want a factorization that includes grandparent substructures to improve *preposition* accuracy. *Conjunctions* have been represented

in a variety of ways in different dependency treebanks. Two possible representations are in Figure 1.4. In Figure 1.4a, only factorizations which include sibling features would ever have the conjunction and both conjuncts in the same scope; in Figure 1.4b, only factorizations which include grandparent features would ever have the conjunction and both conjuncts in the same scope. Chapter 2 will further investigate this relationship between treebank representations of conjunctions and the accuracy of dependency parsers using various factorizations.



(a) One conjunct is the sibling of the conjunction



(b) One conjunct is the grandparent of the other conjunct

Figure 1.4: Conjunction representations

1.2.2 Features

Besides a factorization that scopes over the relevant substructures, one also needs appropriate *features*. Common features for dependency parsers include the words and part-of-speech tags of the parent, child, and their surrounding words (McDonald et al., 2005a). Learning to prefer $\mathrm{Grand}(cut, with, scissors)$ over $\mathrm{Grand}(scarf, with, scissors)$, and also $\mathrm{Grand}(scarf, with, stripes)$ over $\mathrm{Grand}(cut, with, stripes)$ requires features capable of capturing the relevant differences. The words *stripes* and *scissors* have the same part-of-speech, so features based on part-of-speech tags should not differentiate between these two cases. The words themselves are different, however features based on the words themselves are unlikely to have appeared many times in the training set; generally, lexical statistics based on the training set only are typically sparse and have only a small effect on overall parsing performance (Gildea, 2001).

Features derived from unlabeled data, such as clusters (Koo, Carreras, and Collins,

2008) and web counts (Bansal and Klein, 2011) may help, but might not be fully effective if a) the phenomenon are represented inconsistently in the data, or b) none of the features scope over the relevant words involved.

1.2.3 Parsing with Unlabeled Data and Relevant Factorizations

In **Chapter 2**, we show the practicality of considering the compatibility between data representations and factorizations by modifying the parsing system of Koo and Collins (2010) to incorporate features from web-scale association statistics, and perform experiments showing the accuracies overall and on prepositions and conjunctions in particular for each type of factorization and each type of data representation. Table 1.1 shows how the accuracy of attaching conjunctions varies widely under different combinations of factorizations and data representations. This work achieves a new state-of-the-art for English dependencies with 93.55% correct attachments on the current standard. Furthermore, conjunctions are attached with an accuracy of 90.8% and prepositions with an accuracy of 87.4%. This chapter contains material previously published in Pitler (2012).

	Conjunction Accuracy	
Scoring	Conversion 1 (deprecated)	Conversion 2
Edge	86.3	85.3
Sib	87.8	85.5
Grand	87.2	90.6
GrandSib	88.3	90.8

Table 1.1: Unlabeled attachment accuracy for conjunctions under different factorizations and dependency representations.

1.3 Searching

Dependency parsers vary in what *space of possible tree structures* they search over when parsing a sentence. Existing options include projective trees, all arborescences, or existing definitions of mildly non-projective trees.

1.3.1 Projective Trees

Many high-accuracy dependency parsers today (Koo and Collins, 2010; Rush and Petrov, 2012; Zhang and McDonald, 2012) search only over trees without crossing edges (*projective trees*). In a projective tree, each subtree (i.e., each word and its descendants) form a consecutive sequence in the input sentence. Figure 1.5 shows an example of an English sentence that is *not* projective: note that the subtree rooted at *scarf* does not form a single interval in the sentence, and that the edges *(scarf,with)* and *(cut,yesterday)* cross when both are drawn above the sentence.

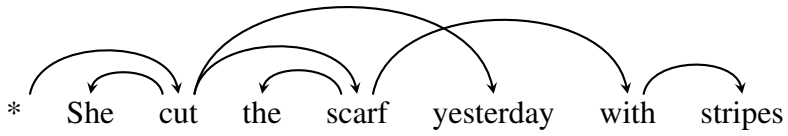


Figure 1.5: A dependency tree with crossing edges

Finding the optimal tree in the set of projective trees can be done efficiently (Eisner, 2000), even when the score of a tree depends on higher order factors (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010). However, the projectivity assumption excludes many natural language dependency trees, especially in languages with freer word orders; for example, only 63.6% of Dutch sentences from the CoNLL-X training set are projective (Table 1.2).

[2]Coverage is the range of empirical coverage of sentences in the training sets of CoNLL-X for Arabic, Czech, Danish, Dutch, Portuguese, and Swedish; Parsing is the asymptotic parsing time for an edge-factored model; Extensible indicates whether it is tractable to extend the model to grandparent and/or sibling factors.

Set of Trees	Coverage	Parsing	Extensible
Projective	**63.6-90.2%**	$O(n^3)$	Yes
Arborescences	100%	$O(n^2)$	**No**
Well-nested and block degree 2	95.4-99.9%	$\mathbf{O(n^7)}$	Yes

Table 1.2: Existing search spaces for dependency parsers[2]

1.3.2 Arborescences

At the other end of the spectrum, some parsers search over all *arborescences* (directed spanning trees), a class of structures much larger than the set of plausible linguistic structures. McDonald et al. (2005b) proposed casting the dependency parsing problem as that of finding the maximum scoring directed spanning tree, which can be found in $O(n^2)$ time (Tarjan, 1977) with a variant of the Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) algorithm when scores are over edges only.

Unfortunately, finding the maximum scoring arborescence is NP-hard with features over siblings (McDonald and Pereira, 2006) or with features over grandparents (McDonald and Satta, 2007). *After learning*, some parsers are able to find the optimal arborescence, at least in the majority of cases (Riedel and Clarke, 2006; Martins, Smith, and Xing, 2009; Koo, Rush, Collins, Jaakkola, and Sontag, 2010). However, many discriminative machine learning methods for structured prediction, such as structured perceptron (Collins, 2002), structural SVMs (Tsochantaridis, Joachims, Hofmann, and Altun, 2006), or max-margin Markov networks (Taskar, Guestrin, and Koller, 2003) rely on an inference step during learning, and no MST parser with features over grandparents and/or siblings has used *exact inference* during learning. Kulesza and Pereira (2007) and Finley and Joachims (2008) show theoretical and empirical results on the effects of approximate inference during learning.

1.3.3 Existing Definitions of Mildly Non-projective Trees

A third alternative is to consider existing definitions of *mildly non-projective trees* that are strictly larger than the set of projective trees and strictly smaller than the set of all arborescences. See Kuhlmann and Nivre (2006) for a nice overview of various constraints that have been proposed and their respective coverages of natural language treebank structures. However, few of these existing definitions have corresponding exact parsing algorithms; moreover, the known parsing algorithms are orders of magnitude slower than algorithms for parsing projective trees.

For example, one definition of mildly non-projective trees is the set of well-nested dependency trees for which the words in each subtree form at most two maximal intervals (i.e., block degree 2/gap degree 1) (Kuhlmann, 2013). This definition also has a connection to a type of mildly context-sensitive grammar: all Lexicalized Tree Adjoining Grammar (LTAG) (Joshi and Schabes, 1997) derivation trees are well-nested and have gap degree at most one (Bodirsky, Kuhlmann, and Möhl, 2005). This set of trees is extensible and has higher coverage of treebank structures than projective trees do (95-4%-99.9%, Table 1.2), but its parsing algorithm takes $O(n^7)$ time (Gómez-Rodríguez, Carroll, and Weir, 2011), which is prohibitive for practical purposes.

1.3.4 Classes of Trees Proposed in this Thesis

Each of the classes of trees discussed so far has had different tradeoffs between coverage, parsing time, and extensibility:

- Projective trees have fast parsing and extensibility, but low coverage
- Arborescences have high coverage and fast parsing, but are not extensible
- Well-nested and block degree 2 trees have high coverage and extensibility, but slow parsing time.

Are there other well-defined classes of trees that allow rich models, cover a large proportion of naturally occurring treebank structures, and can be parsed efficiently? Such

Set of Trees	Coverage	Parsing	Extensible
Projective	**63.6-90.2%**	$O(n^3)$	Yes
Arborescences	100%	$O(n^2)$	**No**
Well-nested and block degree 2	95.4-99.9%	$\mathbf{O(n^7)}$	Yes
and Inherit-1 (Chapter 3)	95.4-99.9%	$O(n^6)$	Yes
and Inherit-0 (Chapter 3)	90.4-97.7%	$O(n^5)$	Yes
1-Endpoint-Crossing (Chapter 4)	95.8-99.8%	$O(n^4)$	Yes

Table 1.3: Classes of trees and parsing algorithms proposed in this thesis, compared with existing tree classes.

classes and parsing algorithms would increase the applicability of parsers to non-English languages. This thesis defines such tree classes, summarized in Table 1.3.

In **Chapter 3**, we introduce *gap inheritance*: a child node inherits a gap of its parent if the child has descendants in more than one of its parent's intervals of descendants. A corpus analysis shows that *none* of the examples of well-nested trees with block degree at most two contain more than one gap inheriting child per node. Adding this *1-Inherit* restriction to the class of well-nested and block degree at most two trees therefore causes no drop in coverage, yet the optimal scoring tree can be found in $O(n^6)$. We also show that *0-Inherit* trees (in which no node has any gap-inheriting child) cover 90.4% or more of treebank structures and can be parsed with an $O(n^5)$ algorithm. This chapter contains material published in Pitler, Kannan, and Marcus (2012). This chapter also includes a previously unpublished result showing how the 0-Inherit property allows an extension to an arbitrary number of gaps without any increase in the complexity of the parsing algorithm.

Chapter 4 proposes *1-Endpoint-Crossing* trees: trees in which whenever an edge is crossed, the edges that cross it all have a common vertex. While simple to state, this class of trees has both better coverage *and* faster parsing. We prove that any such 1-Endpoint-Crossing tree can be decomposed into sets of intervals with one exterior point. This insight allows efficient parsing and we present an $O(n^4)$ dynamic programming parsing algorithm

that recursively combines forests over intervals with one exterior point that finds the optimal 1-Endpoint-Crossing tree. We situate the 1-Endpoint-Crossing tree class in relation to other graph theoretic descriptions, proving that 1-Endpoint-Crossing trees are a subclass of 2-page graphs (Bernhart and Kainen, 1979), or alternatively, 2-planar graphs as have been described in the transition-based parsing literature (Gómez-Rodríguez and Nivre, 2010). In contrast, we show that 1-Endpoint-Crossing and 2-planarity are orthogonal to other established properties such as gap degree and well-nestedness. The work in this chapter appeared in Pitler, Kannan, and Marcus (2013).

1.4 Factorizations that Facilitate Search

Factorizations developed for projective dependency parsing include independence assumptions that allow more efficient search over the set of projective trees. For example, the parsing algorithm of Eisner (2000) derives efficiency from assuming that left and right modifiers of a head word are independent of each other and so can be parsed independently. The sibling factorization of McDonald and Pereira (2006) continues this assumption by only conditioning on siblings on the same side of the parent. Higher order models such as the parser of Carreras (2007) and the tri-sibling (Model 2) parser of Koo and Collins (2010) have avoided increases in the asymptotic parsing time of their algorithms by defining grandparent features for only the outermost children of a parent.

In a similar spirit, we define a grandparent-sibling factorization tailored to allow efficient search over 1-Endpoint-Crossing trees (a superset of projective trees). **Chapter 5** proposes a *crossing-sensitive* third-order factorization. The decomposition of the tree depends on the pattern of crossing edges, using full grandparent and sibling parts in the locally projective portions of the tree and less surrounding context in the presence of crossings. When applied to a projective tree, the crossing-sensitive factorization simplifies exactly to the grand-sibling factorization of Koo and Collins (2010). We show an algorithm that finds the optimal 1-Endpoint-Crossing tree under this model in $O(n^4)$ time, matching the time of both the third-order projective parser (Koo and Collins, 2010) and

that of the edge-factored 1-Endpoint-Crossing parser (Chapter 4). In experiments with a variety of languages and treebank representations, the implemented crossing-sensitive third-order 1-Endpoint-Crossing parser is significantly more accurate than the projective third-order parser in nine out of the sixteen set-ups and significantly less accurate on none. When evaluated on normalized treebanks (Zeman, Mareček, Popel, Ramasamy, Štěpánek, Žabokrtský, and Hajič, 2012) with Stanford-style conjunction representations (De Marneffe and Manning, 2008), the 1-Endpoint-Crossing parser has an unlabeled attachment accuracy 0.38-3.51% higher than the third-order projective parser (Table 1.4).

Model		Dutch	Czech	Portuguese	Danish	Swedish
Proj	GSib	81.16	86.83	88.80	**88.84**	87.27
1-EC	CS-GSib	**84.67**	**88.34**	**90.20**	**89.22**	**88.15**

Table 1.4: Overall Unlabeled Attachment Scores (UAS) for all words. Proj GSib is a third order projective parser (Koo and Collins, 2010); 1-EC CS-GSib is a crossing-sensitive third-order 1-Endpoint-Crossing parser (Chapter 5). Data sources: CoNLL-2006 shared task (Buchholz and Marsi, 2006) (Danish, Dutch, Portuguese, Swedish); CoNLL-2007 shared task (Nivre et al., 2007a) (Czech), normalized and then transformed to use the Stanford-style conjunction representation using HamleDT (Zeman et al., 2012). Bold indicates the more accurate model and models not significantly different from the most accurate (sign test, $p < .05$). Languages are sorted in increasing order of projectivity (Table A.1). For more details see Table 5.5 in Chapter 5.

1.5 Thesis Contributions

This thesis shows that non-projective dependency parsing is tractable even in the presence of higher order factors under new formulations that cover 90% or more of the structures found in dependency treebanks. We provide new definitions of classes of trees and algorithms for efficient optimal search within these classes. We also show the effect of the compatibility between the scope of features in the parsing model and the representations

of difficult constructions in treebanks on the accuracy of trained parsers. In particular, this thesis contributes:

- An overview of the differences in representations between two different constituency-to-dependency conversion procedures (Section 2.2)
- An empirical demonstration of the effect of varying model factorizations on the accuracy of overall parsing, conjunctions, and prepositions (Section 2.6)
- A demonstration that unlabeled data features lead to a statistically significant improvement over the prior state-of-the-art in unlabeled attachment accuracy (Section 2.6)
- A definition of *gap inheritance*, and a demonstration that 1-Inheritance reduces complexity by a factor of n without *any* loss in empirical coverage over prior work (Sections 3.3-3.4)
- Exact $O(n^5)$ algorithms for finding the maximum well-nested, 0-inherit tree either with gap degree 1 or with unbounded gap degree (Sections 3.5 and 3.8))
- A definition of *1-Endpoint-Crossing*, a property over graphs with linearly ordered vertices novel to both linguistics and to graph theory (Sections 4.2 and 4.6)
- An $O(n^4)$ exact parsing algorithm for finding the optimal 1-Endpoint-Crossing tree under an edge-factored model (Section 4.4)
- A proof that 1-Endpoint-Crossing trees are a subclass of 2-planar graphs (Section 4.6)
- Examples that prove these are two distinct hierarchies capturing different dimensions of non-projectivity: *1-Endpoint-Crossing* $\not\subseteq$ *well-nested with block degree 2* and *gap-minding* $\not\subseteq$ *2-planar* (Figure 4.1)
- A novel crossing-sensitive grandparent-sibling factorization that generalizes the third-order projective case (Section 5.3)

- A parsing algorithm that finds the optimal 1-Endpoint-Crossing tree according to this crossing-sensitive grandparent-sibling factorization in $O(n^4)$ time (Section 5.4)
- An empirical demonstration that the third-order 1-Endpoint-Crossing parser is more accurate than the third-order projective parser on several different languages and treebank representations (Section 5.5)

Chapter 2

Attacking Parsing Bottlenecks with Unlabeled Data and Relevant Factorizations

Much of this chapter was originally published in Pitler (2012).

2.1 Introduction

Prepositions and *conjunctions* are two large remaining bottlenecks in parsing. Across various existing parsers, these two categories have the lowest accuracies, and mistakes made on these have consequences for downstream applications. Machine translation is sensitive to parsing errors involving prepositions and conjunctions, because in some languages different attachment decisions in the parse of the source language sentence produce different translations. Preposition attachment mistakes are particularly bad when translating into Japanese (Schwartz, Aikawa, and Quirk, 2003) which uses a different postposition for different attachments; conjunction mistakes can cause word ordering mistakes when translating into Chinese (Huang, 1983).

Prepositions and conjunctions are often assumed to depend on *lexical dependencies* for

correct resolution (Jurafsky and Martin, 2008). However, lexical statistics based on the training set only are typically sparse and have only a small effect on overall parsing performance (Gildea, 2001). *Unlabeled data* can help ameliorate this sparsity problem. Backing off to cluster membership features (Koo et al., 2008) or by using association statistics from a larger corpus, such as the web (Bansal and Klein, 2011; Zhou, Zhao, Liu, and Cai, 2011), have both improved parsing.

Unlabeled data has been shown to improve the accuracy of conjunctions within complex noun phrases (Pitler, Bergsma, Lin, and Church, 2010; Bergsma et al., 2011). However, it has so far been less effective within full parsing — while first-order web-scale counts noticeably improved overall parsing in Bansal and Klein (2011), the accuracy on conjunctions actually decreased when the web-scale features were added (Table 4 in that paper).

In this paper we show that unlabeled data *can* help prepositions and conjunctions, *provided that the dependency representation is compatible with how the parsing problem is decomposed for learning and inference*. By incorporating unlabeled data into factorizations which capture the relevant dependencies for prepositions and conjunctions, we produce a parser for English which has an unlabeled attachment accuracy of 93.5%, over an 18% reduction in error over the best previously published parser (Bansal and Klein, 2011) on the current standard for dependency parsing. The best model for conjunctions attaches them with 90.8% accuracy (42.5% reduction in error over MSTParser), and the best model for prepositions with 87.4% accuracy (18.2% reduction in error over MSTParser).

We describe the dependency representations of prepositions and conjunctions in Section 2.2. We discuss the implications of these representations for how learning and inference for parsing are decomposed (Section 2.3) and how unlabeled data may be used (Section 2.4). We then present experiments exploring the connection between representation, factorization, and unlabeled data in Sections 2.5 and 2.6.

2.2 Conversion to Dependency Representations

The Wall Street Journal Penn Treebank (PTB) (Marcus, Marcinkiewicz, and Santorini, 1993) contains parsed constituency trees (where each sentence is represented as a context-free-grammar derivation). Dependency parsing requires a conversion from these constituency trees to dependency trees. The Treebank constituency trees left noun phrases (NPs) flat, although there have been subsequent projects which annotate the internal structure of noun phrases (Vadas and Curran, 2007; Weischedel, Palmer, Marcus, Hovy, Pradhan, Ramshaw, Xue, Taylor, Kaufman, Franchini, et al., 2011). The presence or absence of these noun phrase internal annotations interacts with constituency-to-dependency conversion program in ways which have effects on conjunctions and prepositions.

We consider two such mapping regimes here:

1. PTB trees $\rightarrow$ *Penn2Malt*[1] $\rightarrow$ Dependencies

2. PTB trees patched with NP-internal annotations (Vadas and Curran, 2007) $\rightarrow$ *pennconverter*[2] $\rightarrow$ Dependencies

Regime (1) is very commonly done in papers which report dependency parsing experiments (e.g., McDonald and Pereira (2006); Nivre, Hall, Nilsson, Chanev, Eryigit, Kübler, Marinov, and Marsi (2007b); Zhang and Clark (2008); Huang and Sagae (2010); Koo and Collins (2010)). *Penn2Malt* uses the head finding table from Yamada and Matsumoto (2003).

Regime (2) is based on the recommendations of the two converter tools; as of the date of this writing, the *Penn2Malt* website says: "Penn2Malt has been superseded by the more sophisticated pennconverter, which we strongly recommend". The *pennconverter* website "strongly recommends" patching the Treebank with the NP annotations of Vadas and Curran (2007). A version of *pennconverter* was used to prepare the data for the CoNLL Shared

[1] `http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html`

[2] Johansson and Nugues (2007) `http://nlp.cs.lth.se/software/treebank_converter/`

Tasks of 2007-2009, so the trees produced by Regime 2 are similar (but not identical)[3] to these shared tasks. As far as we are aware, Bansal and Klein (2011) is the only published work which uses both steps in Regime (2).

The dependency representations produced by Regime 2 are designed to be more useful for extracting semantics (Johansson and Nugues, 2007). The parsing attachment accuracy of MALTPARSER (Nivre et al., 2007b) was lower using *pennconverter* than *Penn2Malt*, but using the output of MALTPARSER under the new format parses produces a much better semantic role labeler than using its output with *Penn2Malt* (Johansson and Nugues, 2007).

Figures 2.1 and 2.2 show how conjunctions and prepositions, respectively, are represented after the two different conversion processes. *These differences are not rare*–70.7% of conjunctions and 5.2% of prepositions in the development set have a different parent under the two conversion types. These representational differences have serious implications for how well various factorizations will be able to capture these two phenomena.

[3]The CoNLL data does not include the NP annotations; it does include annotations of named entities (Weischedel and Brunstein, 2005) so had some internal NP edges.

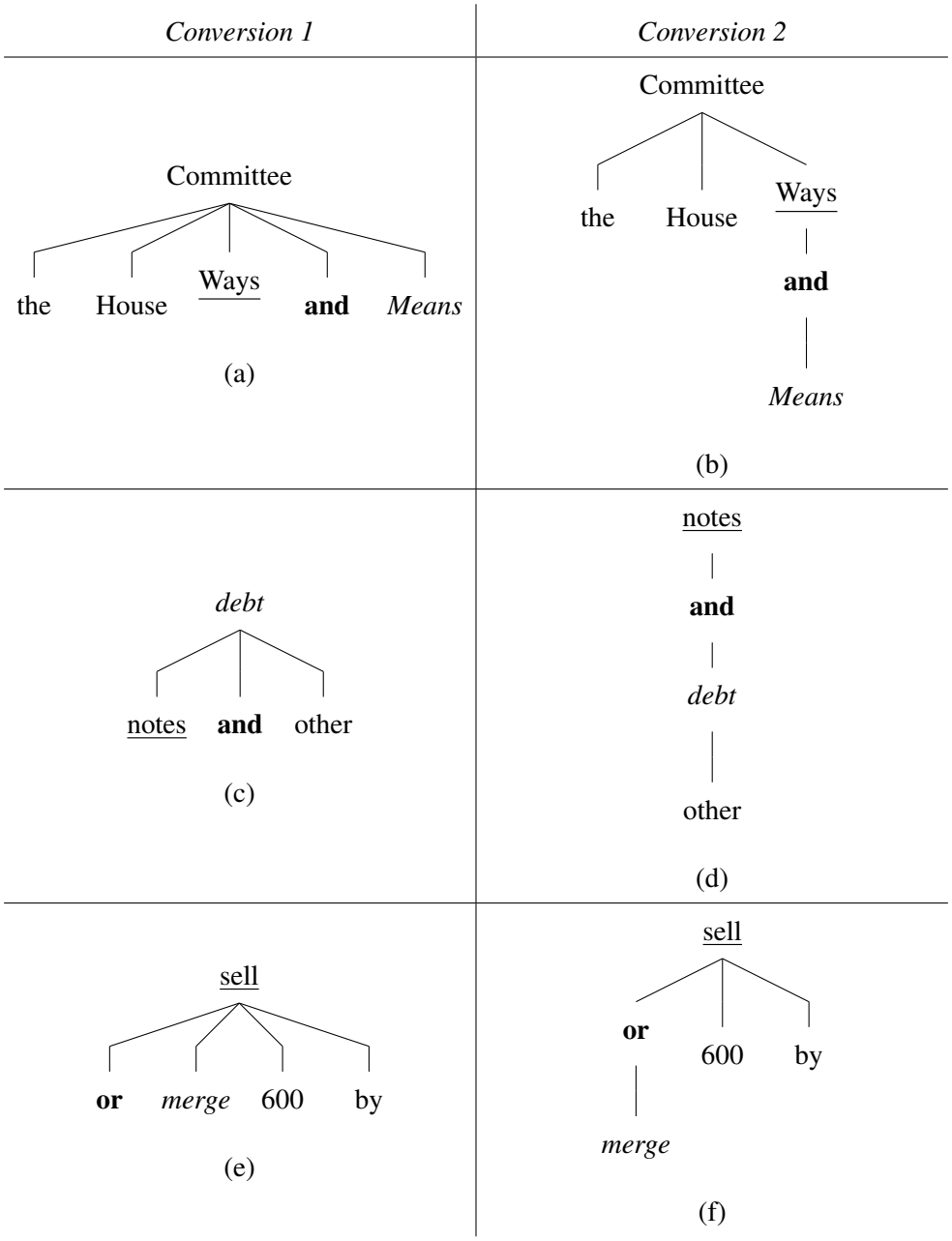


Figure 2.1: Examples of conjunctions: *the House Ways and Means Committee*, *notes and other debt*, and *sell or merge 600 by*. The conjunction is bolded, the left conjunct (in the linear order of the sentence) is underlined, and the right conjunct is italicized.

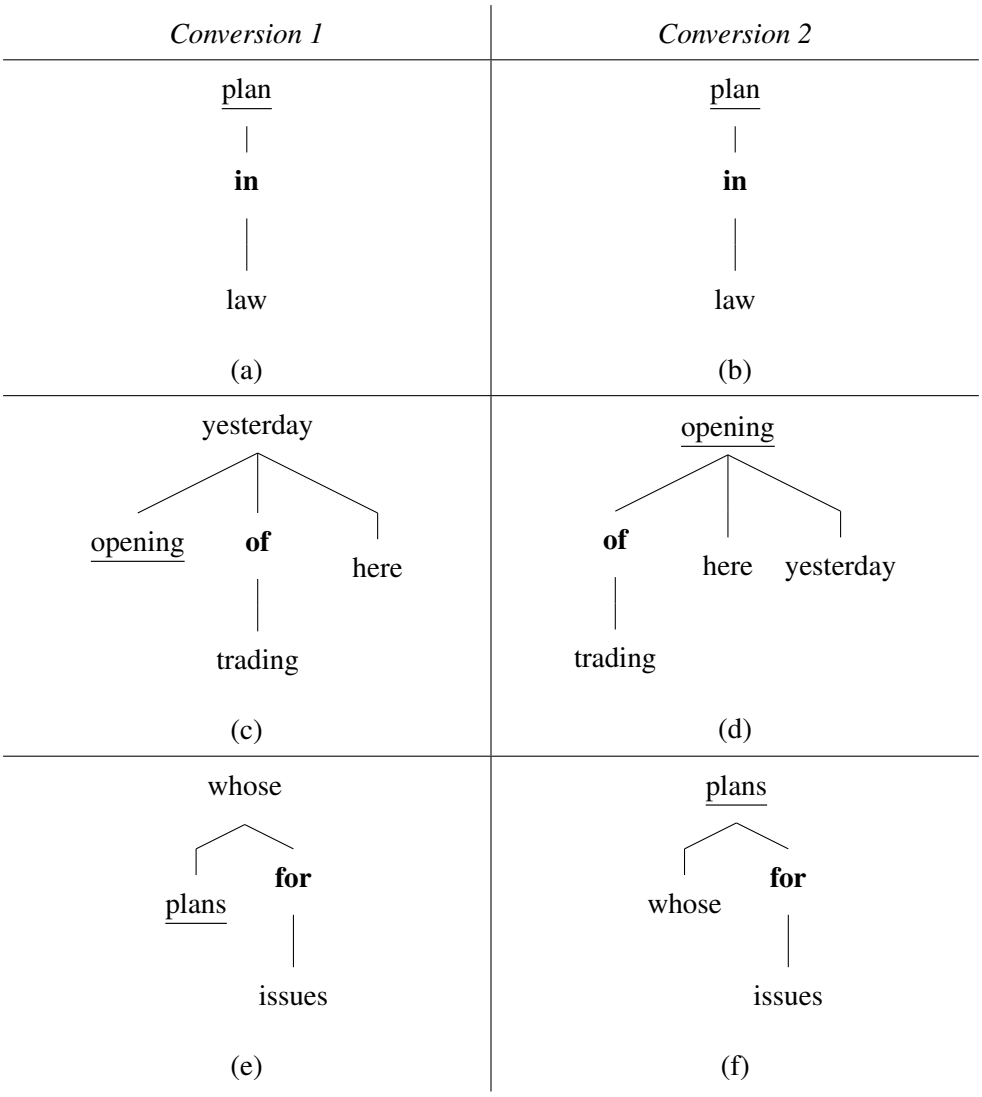


Figure 2.2: Examples of prepositions: *plan in the S&L bailout law*, *opening of trading here yesterday*, and *whose plans for major rights issues*. The preposition is bolded and the (semantic) head is underlined.

2.3 Implications of Representations on the Scope of Factorization

Parsing requires a) learning to score potential parse trees, and b) given a particular scoring function, finding the highest scoring tree according to that function. The number of potential trees for a sentence is exponential, so parsing is made tractable by decomposing the problem into a set of local substructures which can be combined using dynamic programming. Four possible factorizations are: single edges (edge-based), pairs of edges which share a parent (siblings), pairs of edges where the child of one is the parent of the other (grandparents), and triples of edges where the child of one is the parent of two others (grandparent+sibling). In this section, we discuss these factorizations and their relevance to conjunction and preposition representations.

2.3.1 Edge-based Scoring

One possible factorization corresponds to first-order parsing, in which the score of a parse tree y decomposes completely across the edges in the tree:

$$Score(y) = \sum_{(h,m) \in y} Score(\text{Edge}(h, m)) \tag{2.1}$$

Conjunctions: Under Conversion 1, we can see three different representations of conjunctions in Figures 2.1a, 2.1c, and 2.1e. Under edge-based scoring, the conjunction would be scored along with *neither* of its conjuncts in 2.1a. In Figure 2.1c, the conjunction is scored along with its right conjunct only; in figure 2.1e along with its left conjunct only. The inconsistency here is likely to make learning more difficult, as what is learned is split across these three cases. Furthermore, the conjunction is connected with an edge to either zero or one of its two arguments; at least one of the arguments is completely ignored in terms of scoring the conjunction.

In Figures 2.1c and 2.1e, the words being conjoined are connected to *each other* by an edge. This overloads the meaning of an edge; an edge indicates both a head-modifier

relationship and a conjunction relationship. For example, compare the two natural phrases *dogs and cats* and *really nice*. *dogs* and *cats* are a good pair to conjoin, but *cats* is not a good modifier for *dogs*, so there is a tension when scoring an edge like $(dogs, cats)$: it should get a high score when actually indicating a conjunction and low otherwise. $(nice, really)$ shows the opposite pattern–*really* is a good modifier for *nice*, but *nice* and *really* are not two words which should be conjoined. This may be partially compensated for by including features about the surrounding words (McDonald et al., 2005a), but any feature templates which would be identical across the two contexts will be in tension.

In Figures 2.1b, 2.1d and 2.1f, the conjunction participates in a directed edge with each of the conjuncts. Thus, in edge-based scoring, at least under Conversion 2 neither of the conjuncts is being ignored; however, the factorization scores each edge independently, so how compatible these two conjuncts are with each other cannot be included in the scoring of a tree.

Prepositions: For all of the examples in Figure 2.2, there is a directed edge from the head of the phrase that the preposition modifies to the preposition. Differences in head finding rules account for the differences in preposition representations. In the second example, the first conversion scheme chooses *yesterday* as the head of the overall NP, resulting in the edge *yesterday*$\rightarrow$ *of*, while the second conversion scheme ignores temporal phrases when finding the head, resulting in the more semantically meaningful *opening*$\rightarrow$*of*. Similarly, in the third example, the preposition *for* attaches to the pronoun *whose* in the first conversion scheme, while it attaches to the noun *plans* in the second.

With edge-based scoring, the object is not accessible when scoring where the preposition should attach, and PP-attachment is known to depend on the object of the preposition (Hindle and Rooth, 1993).

2.3.2 Sibling Scoring

Another alternative factorization is to score *siblings* as well as parent-child edges (McDonald and Pereira, 2006). Scores decompose as:

$$Score(y) = \sum_{\left\{ (h,m,s) \;\middle|\; \begin{array}{l} (h,m) \in y, (h,s) \in y, \\ (s,m) \in Siblings(y) \end{array} \right\}} Score(\mathrm{Sib}(h,m,s)) \tag{2.2}$$

where $Siblings(y)$ is the set containing *ordered* and *adjacent* sibling pairs in y: if $(s,m) \in Siblings(y)$, there must exist a shared parent h such that $(h,m) \in y$ and $(h,s) \in y$, m and s must be on the same side of h, s must be closer to h than m in the linear order of the sentence, and there must not exist any other children of h in between m and s.

Under this factorization, two of the three examples in Conversion 1 (and none of the examples in Conversion 2) in Figure 2.1 now include the conjunction and both conjuncts in the same score (Figures 2.1c and 2.1e). The scoring for head-modifier dependencies and conjunction dependencies are again being overloaded: $(debt, notes, and)$ and $(debt, and, other)$ are both sibling parts in Figure 2.1c, yet only one of them represents a conjunction. The position of the conjunction in the sibling is not enough to determine whether one is scoring a true conjunction relation or just the conjunction and a different sibling; in 2.1c the conjunction is on the right of its sibling argument, while in 2.1e the conjunction is on the left.

For none of the other preposition or conjunction examples does a sibling factorization bring more of the arguments into the scope of what is scored along with the preposition/conjunction. Sibling scoring may have some benefit in that prepositions/conjunctions should have only one argument, so for prepositions (under both conversions) and conjunctions (under Conversion 2), the model can learn to disprefer the existence of any siblings and thus enforce choosing a single child.

2.3.3 Grandparent Scoring

Another alternative over pairs of edges scores grandparents instead of siblings, with factorization:

$$Score(y) = \sum_{\left\{ (h,m,c) \;\middle|\; (h,m) \in y, (m,c) \in y \right\}} Score(\text{Grand}(h,m,c)) \tag{2.3}$$

Under Conversion 2, we would expect this factorization to perform much better on conjunctions and prepositions than edge-based or sibling-based factorizations. Both conjunctions and prepositions are consistently represented by exactly one grandparent relation (with one relevant argument as the grandparent, the preposition/conjunction as the parent, and the other argument as the child), so this is the first factorization that has allowed the compatibility of the two arguments to affect the attachment of the preposition/conjunction.

Under Conversion 1, this factorization is particularly appropriate for prepositions, but would be unlikely to help conjunctions, which have no children.

2.3.4 Grandparent-Sibling Scoring

A further widening of the factorization takes grandparents and siblings simultaneously:

$$Score(y) = \sum_{\left\{ \begin{array}{l|l} (g,h,m,s) & (g,h) \in y, (h,m) \in y, \\ & (h,s) \in y, (s,m) \in Sib(y) \end{array} \right\}} Score(\text{GrandSib}(g,h,m,s)) \tag{2.4}$$

For projective parsing, dynamic programming for this factorization was derived in Koo and Collins (2010) (Model 1 in that paper), and for non-projective parsing, dual decomposition was used for this factorization in Koo et al. (2010).

This factorization should combine all the benefits of the sibling and grandparent factorizations described above–for Conversion 1, sibling scoring may help conjunctions and grandparent scoring may help prepositions, and for Conversion 2, grandparent scoring should help both, while sibling scoring may or may not add some additional gains.

2.4 Using Unlabeled Data Effectively

Associations from unlabeled data have the potential to improve both conjunctions and prepositions. We predict that web counts which include both conjuncts (for conjunctions), or which include both the attachment site and the object of a preposition (for prepositions) will lead to the largest improvements.

For the phrase *dogs and cats*, edge-based counts would measure the associations between *dogs* and *and*, and *and* and *cats*, but never any web counts that include both *dogs* and *cats*. For the phrase *ate spaghetti with a fork*, edge-based scoring would not use any web counts involving both *ate* and *fork*.

We use *associations* rather than raw counts. The phrases *trading and transacting* versus *trading and what* provide an example of the difference between associations and counts. The phrase *trading and what* has a higher count than the phrase *trading and transacting*, but *trading and transacting* are more highly associated. In this paper, we use point-wise mutual information (PMI) to measure the strength of associations of words participating in potential conjunctions or prepositions.[4] For three words h, m, c, this is calculated with:

$$PMI(h, m, c) = \log \frac{P(h\ .*\ m\ .*\ c)}{P(h)P(m)P(c)} \tag{2.5}$$

The probabilities are estimated using web-scale n-gram counts, which are looked up using the tools and web-scale n-grams described in Lin, Church, Ji, Sekine, Yarowsky, Bergsma, Patil, Pitler, Lathbury, Rao, Dalwani, and Narsale (2010). Defining the joint probability using wildcards (rather than the exact sequence *h m c*) is crucially important, as determiners, adjectives, and other words may naturally intervene between the words of interest.

Approaches which cluster words (i.e., (Koo et al., 2008)) are also designed to identify words which are semantically related. As manually labeled parsed data is sparse, this may help generalize across similar words. However, if edges are not connected to the semantic head, cluster-based methods may be less effective. For example, the choice of *yesterday* as the head of *opening of trading here yesterday* in Figure 2.2c or *whose* in 2.2e may make

[4]PMI can be unreliable when frequency counts are small (Church and Hanks, 1990), however the data used was thresholded, so all counts used are at least 10.

cluster-based features less useful than if the semantic heads were chosen (*opening* and *plans*, respectively).

2.5 Experiments

The previous section motivated the use of unlabeled data for attaching prepositions and conjunctions. We have also hypothesized that these features will be most effective when the *data representation* and the *learning representation* both capture relevant properties of prepositions and conjunctions. We predict that Conversion 2 and a factorization which includes grand-parent scoring will achieve the highest performance. In this section, we investigate the impact of unlabeled data on parsing accuracy using the two conversions and using each of the factorizations described in Section 2.3.1-2.3.4.

2.5.1 Unlabeled Data Feature Set

Clusters: We replicate the cluster-based features from (Koo et al., 2008), which includes features over *all* edges (h, m), grand-parent triples (h, m, c), and parent sibling triples (h, m, s). The features were all derived from the publicly available clusters produced by running the Brown clustering algorithm (Brown, Desouza, Mercer, Pietra, and Lai, 1992) over the BLLIP corpus (Charniak, Blaheta, Ge, Hall, Hale, and Johnson, 2000, about 30 million words of Wall Street Journal text) with the Penn Treebank sentences excluded.[5]

Preposition and conjunction-inspired features (motivated by Section 2.4) are described below:

Web Counts: The web counts data (Lin et al., 2010) is derived from 1 trillion tokens of Web text. The source text is identical to the source text used in the data of the Google N-gram Corpus (Brants and Franz, 2006), but additional filtering of duplicate sentences and other noise was done prior to computing the counts (Lin et al., 2010). Search tools[6] allow

[5] `people.csail.mit.edu/maestro/papers/bllip-clusters.gz`

[6] `https://code.google.com/p/ngramtools/`

look-ups with wildcard queries. For each set of words of interest, we compute the PMI between the words, and then include binary features for whether the mutual information is undefined, if it is negative, and whether it is greater than each positive integer.

For conjunctions, we only do this for triples of both conjunct and the conjunction (and if the conjunction is *and* or *or* and the two potential conjuncts are the same coarse grained part-of-speech). For prepositions, we consider only cases in which the parent is a noun or a verb and the child is a noun (this corresponds to the cases considered by (Hindle and Rooth, 1993) and others). Prepositions use association features to score both the triple (parent, preposition, child) and all pairs within that triple. The counts features are not used if all the words involved are stopwords. For the scope of this paper we use *only* the above counts related to prepositions and conjunctions.

2.5.2 Parser

We use the Model 1 version of $dpo3$, a state-of-the-art third-order dependency parser (Koo and Collins, 2010))[7]. We augment the feature set used with the web-counts-based features relevant to prepositions and conjunctions and the cluster-based features. The only other change to the parser's existing feature set was the addition of binary features for the part-of-speech tag of the child of the root node, alone and conjoined with the tags of its children. For further details about the parser, see Koo and Collins (2010).

2.5.3 Experimental Set-up

Training was done on Section 2-21 of the Penn Treebank (39,832 sentences). Section 22 was used for development (1700 sentences), and Section 23 for test (2416 sentences). We use automatic part-of-speech tags for both training and testing (Ratnaparkhi, 1996). The set of potential edges was pruned using the marginals produced by a first-order parser trained using exponentiated gradient descent (Collins, Globerson, Koo, Carreras, and Bartlett, 2008) as in Koo and Collins (2010). We train the full parser for 15 iterations of averaged

[7] `http://groups.csail.mit.edu/nlp/dpo3/`

perceptron training (Collins, 2002), choose the iteration with the best unlabeled attachment score (UAS) on the development set, and apply the model after that iteration to the test set.

We also ran MSTParser (McDonald and Pereira, 2006), the Berkeley constituency parser (Petrov and Klein, 2007), and the unmodified dpo3 Model 1 (Koo and Collins, 2010) using Conversion 2 (the current recommendations) for comparison. Since the converted Penn Treebank now contains a few non-projective sentences, we ran both the projective and non-projective versions of the second order (sibling) MSTParser. The Berkeley parser was trained on the constituency trees of the PTB patched with (Vadas and Curran, 2007), and then the predicted parses were converted using pennconverter.

Evaluation Metric The main evaluation metric used here is that of *unlabeled attachment score (UAS)*, defined as the percentage of words that have the correct parent. Each word has exactly one parent in both the gold tree and the predicted tree, so the unlabeled attachment score is the number of words for which the parent is correct divided by the total number of words.

2.6 Results and Discussion

Table 2.1 shows the unlabeled attachment scores, complete sentence exact match accuracies, and the accuracies of conjunctions and prepositions under Conversion 2.[8] The incorporation of the unlabeled data features (clusters and web counts) into the *dpo3* parser yields a significantly better parser than *dpo3* alone (93.54 UAS versus 93.21)[9], and is more than a 1.5% improvement over MSTParser.

[8] As is standard for English dependency parsing, five punctuation symbols :, ,, “, ”, and . are excluded from the results (Yamada and Matsumoto, 2003).

[9] If the (deprecated) Conversion 1 is used, the new features improve the UAS of *dpo3* from 93.04 to 93.51.

Model	UAS	Exact Match	Conjunctions	Prepositions
MSTParser (proj)	91.96	38.9	84.0	84.2
MSTParser (non-proj)	91.98	38.7	83.8	84.6
Berkeley (converted)	90.98	36.0	85.6	84.3
dpo3 (GrandSib)	93.21	**44.8**	**89.6**	**86.9**
dpo3+Unlabeled (Edge)	93.12	43.6	85.3	**87.0**
dpo3+Unlabeled (Sib)	93.15	43.7	85.5	86.8
dpo3+Unlabeled (Grand)	**93.55**	**46.1**	**90.6**	**87.5**
dpo3+Unlabeled (GrandSib)	**93.54**	**46.0**	**90.8**	**87.4**
- Clusters	93.10	**45.0**	**90.5**	**87.5**
- Prep,Conj Counts	**93.52**	**45.8**	**89.9**	**87.1**

Table 2.1: Test set accuracies under Conversion 2 of unlabeled attachment scores, complete sentence exact match accuracies, conjunction accuracy, and preposition accuracy. Bolded items are the best in each column, or not significantly different from the best in that column (sign test, $p < .05$).

2.6.1 Impact of Factorization

In all four metrics (attachment of all non-punctuation tokens, sentence accuracy, prepositions, and conjunctions), there is no significant difference between the version of the parser which uses the grandparent and sibling factorization (GrandSib) and the version which uses just the grandparent factorization (Grand). A parser which uses only grandparents (referred to as Model 0 in Koo and Collins (2010)) may therefore be preferable, as it contains far fewer parameters than a third-order parser.

While the grandparent factorization and the sibling factorization (Sib) are both "second-order" parsers, scoring up to two edges (involving three words) simultaneously, their results are quite different, with the sibling factorization scoring much worse. This is particularly notable in the conjunction case, where the sibling model is over 5% absolute worse in accuracy than the grandparent model. This relative difference holds regardless of whether one computes the attachment accuracy of conjunctions or whether one computes the accuracy of getting all edges involved with the conjunction correct (Table 2.4).

2.6.2 Impact of Unlabeled Data

The unlabeled data features improved the already state-of-the-art *dpo3* parser in UAS, complete sentence accuracy, conjunctions, and prepositions. However, because the sample sizes are much smaller for the latter three cases, only the UAS improvement is statistically significant.[10] Overall, the results in Table 2.1 show that while the inclusion of unlabeled data improves parser performance, increasing the size of factorization matters even more. Ablation experiments showed that cluster features have a larger impact on overall UAS, while count features have a larger impact on prepositions and conjunctions.

2.6.3 Comparison with Other Parsers

The *dpo3*+Unlabeled parser is significantly better than both versions of MSTParser and the Berkeley parser converted to dependencies across all four evaluations. *dpo3*+Unlabeled has an UAS 1.5% higher than MSTParser, which has an UAS 1.0% higher than the converted constituency parser. The MSTParser uses sibling scoring, so it is unsurprising that it performs less well on the new conversion.

While the converted constituency parser is not as good on dependencies as MSTParser overall, note that it is over a percent and a half better than MSTParser on attaching conjunctions (85.6% versus 84.0%). Conjunction scope may benefit from parallelism and higher-level structure, which is easily accessible when joining two matching non-terminals in a context-free grammar, but much harder to determine in the local views of graph-based dependency parsers. The dependencies arising from the Berkeley constituency trees have higher conjunction accuracies than either the edge-based or sibling-based dpo3+Unlabeled parser. However, once grandparents are included in the factorization, the dpo3+Unlabeled is significantly better at attaching conjunctions than the constituency parser, attaching conjunctions with an accuracy over 90%. Therefore, some of the disadvantages of dependency parsing compared with constituency parsing can be compensated for with larger factoriza-

[10]There are 49,892 non-punctuation tokens in the test set, compared with 2416 sentences, 1373 conjunctions, and 5854 prepositions.

tions.

Conjunctions		
Scoring	Conversion 1 (deprecated)	Conversion 2
Edge	86.3	85.3
Sib	**87.8**	85.5
Grand	**87.2**	**90.6**
GrandSib	**88.3**	**90.8**

Table 2.2: Unlabeled attachment accuracy for conjunctions. Bolded items are the best in each column, or not significantly different (sign test, $p < .05$).

Prepositions		
Scoring	Conversion 1 (deprecated)	Conversion 2
Edge	87.4	**87.0**
Sib	87.5	86.8
Grand	**87.9**	**87.5**
GrandSib	**88.4**	**87.4**

Table 2.3: Unlabeled attachment accuracy for prepositions. Bolded items are the best in each column, or not significantly different (sign test, $p < .05$).

2.6.4 Impact of Data Representation

Tables 2.2 and 2.3 show the results of the *dpo3*+Unlabeled parser for conjunctions and prepositions, respectively, under the two different conversions. The data representation has an impact on which factorizations perform best. Under Conversion 1, conjunctions are more accurate under a sibling parser than a grandparent parser, while the pattern is reversed for Conversion 2.

Scoring	Child=CC	Parent=CC	All Edges Incident to CC
Edge	85.3	90.7	78.9
Sib	85.5	91.1	80.3
Grand	90.6	92.6	85.9
GrandSib	90.8	91.6	85.6

Table 2.4: Different ways of measuring the accoracy of edges involving conjunctions: Child=CC is the accuracy of edges in which the child is the conjunction; Parent=CC is the accuracy of edges in which the parent is the conjunction in the gold-standard; the third column is the most strict, counting a conjunction as correct only if the set of edges incident to it exactly match between the gold-standard and the predicted tree (not counting excluded punctuation edges).

Conjunctions show a much stronger need for higher order factorizations than prepositions do. This is not too surprising, as prepositions have more of a selectional preference than conjunctions, and so the preposition itself is more informative about where it should attach. While prepositions do improve with larger factorizations, the improvement beyond edge-based is not significant for Conversion 2. One hypothesis for why Conversion 1 shows more of an improvement is that the wider scope leads to the semantic head being included; in Conversion 2, the semantic head is chosen as the parent of the preposition, so the wider scope is less necessary.

2.6.5 Preposition Error Analysis

Prepositions are *still* the largest source of errors in the dpo3+Unlabeled parser. We therefore analyze the errors made on the development set to determine whether the difficult remaining cases for parsers correspond to the (Hindle and Rooth, 1993) style PP-attachment classification task. In the PP-attachment classification task, the two choices for where the preposition attaches are the previous verb or the previous noun, and the preposition itself has a noun object. The ones that *do* attach to the preceeding noun or verb (not necessarily the preceeding word) and have a noun object (2323 prepositions) are attached by the

dpo3+Unlabeled grandparent-scoring parser with 92.4% accuracy, while those that do not fit that categorization (1703 prepositions) have the correct parent only 82.7% of the time.

Local attachments are more accurate — prepositions are attached with 94.8% accuracy if the correct parent is the immediately preceeding word (2364 cases) and only 79.1% accuracy if it is not (1662 cases). The preference is not necessarily for low attachments though: the prepositions whose parent is not the preceeding word are attached more accurately if the parent is the root word (usually corresponding to the main verb) of the sentence (90.8%, 587 cases) than if the parent is lower in the tree (72.7%, 1075 cases).

2.7 Conclusion

Features derived from unlabeled data (clusters and web counts) significantly improve a state-of-the-art dependency parser for English. We showed how well various factorizations are able to take advantage of these unlabeled data features, focusing our analysis on conjunctions and prepositions. Including grandparents in the factorization increases the accuracy of conjunctions over 5% absolute over edge-based or sibling-based scoring. The representation of the data is extremely important for how the problem should be factored– under the old Penn2Malt dependency representation, a sibling parser was more accurate than a grandparent parser. As some important relationships were represented as siblings and some as grandparents, there was a need to develop third-order parsers which could exploit both simultaneously (Koo and Collins, 2010). Under the new pennconverter standard, a grandparent parser is significantly better than a sibling parser, and there is no significant improvement when including both.

Chapter 3

Dynamic Programming for Higher Order Parsing of Gap-Minding Trees

Most of this chapter was originally published as Pitler et al. (2012). The extension to an arbitrary number of gaps is new and is presented in Section 3.8.

3.1 Introduction

We propose two new classes of trees between projective trees and the set of all spanning trees. These two classes provide a closer approximation to the set of plausible natural language dependency trees: unlike projective trees, a word can have descendants in more than one interval; unlike spanning trees, these intervals cannot be nested in arbitrary ways. We introduce *gap inheritance*, a new structural property on trees, which provides a way to quantify the degree to which these intervals can be nested. Different levels of gap inheritance define each of these two classes (Section 3.3).

The *1-Inherit* class of trees (Section 3.4) has *exactly* the same empirical coverage (Table 3.1) of natural language sentences as the class of *well-nested block degree 2 trees* (Bodirsky et al., 2005), yet the optimal scoring tree can be found in an order of magnitude less time (Section 3.4.1).

Gap-minding trees (the second class) have the property that all edges into an interval of descendants come from the same node. Non-contiguous intervals are therefore *decoupled* given this single node, and thus an algorithm which uses only single intervals (as in projective parsing) can produce trees in which a node has descendants in multiple intervals (as in parsing for well-nested block degree 2 trees (Gómez-Rodríguez et al., 2011)). A procedure for finding the optimal scoring tree in this space is given in Section 3.5, which can be searched in yet another order of magnitude faster than the *1-Inherit* class.

Unlike the class of spanning trees, it is still tractable to find the optimal tree in these new spaces when higher order factors are included. An extension which finds the optimal scoring gap-minding tree with scores over pairs of adjacent edges (grandparent scoring) is given in Section 3.6. These gap-minding algorithms have been implemented in practice and empirical results are presented in Section 3.7.

3.2 Preliminaries

In this section, we review some relevant definitions from previous work that characterize degrees of non-projectivity. We also review how well these definitions cover empirical data from six languages: Arabic, Czech, Danish, Dutch, Portuguese, and Swedish. These are the six languages whose CoNLL-X shared task data are either available open source[1] or from the LDC[2]. The CoNLL-X shared task converted parsed data into a standardized format. The standardized format contains one token per line and tab-separated fields including the word, part-of-speech tag, parent and other information (see `http://ilk.uvt.nl/conll/example.html` for examples). Arabic, Czech, Danish, and Swedish data came from existing dependency treebanks, while the Dutch and Portuguese data were converted from phrase structure trees.

Definition 1. *The* projection *of a node is the set of words in the subtree rooted at it (including itself).*

[1] `http://ilk.uvt.nl/conll/free_data.html`

[2] LDC catalogue numbers LDC2006E01 and LDC2006E02

A tree is *projective* if, for every node in the tree, that node's projection forms a contiguous interval in the input sentence order.

A tree is *non-projective* if the above does not hold, i.e., there exists at least one word whose descendants do not form a contiguous interval.

Definition 2. *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)*

Note that a projective tree will have block degree 1 (gap degree 0).

Two subtrees *interleave* if there are vertices l_1, r_1 from one subtree and l_2, r_2 from the other such that $l_1 < l_2 < r_1 < r_2$.

Definition 3. *A tree is* well-nested *if no two* disjoint *subtrees interleave (Bodirsky et al., 2005).*

	Arabic	Czech	Danish	Dutch	Portuguese	Swedish	Parsing
Well-nested +block degree 2	1458 (99.9)	72321 (99.5)	5175 (99.7)	12896 (96.6)	8650 (95.4)	10955 (99.2)	$O(n^7)$
+1-Inherit	1458 (99.9)	72321 (99.5)	5175 (99.7)	12896 (96.6)	8650 (95.4)	10955 (99.2)	$O(n^6)$
+0-Inherit	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 3.1: The number of sentences from the CoNLL-X training sets whose parse trees fall into each of the above classes. The two new classes of structures have more coverage of empirical data than projective structures, yet can be parsed faster than well-nested block degree 2 structures. Parsing times assume an edge-based factorization with no pruning of edges. The corresponding algorithms for the 1-Inherit and 0-Inherit classes are in Sections 3.4 and 3.5.

Well-nested trees that have block degree at most 2 (gap degree at most 1) are of both theoretical and practical interest, as they correspond to derivations in Lexicalized Tree Ad-

joining Grammar (Bodirsky et al., 2005) and cover the overwhelming majority of sentences found in treebanks for Czech and Danish (Kuhlmann and Nivre, 2006).

Table 3.1 shows the proportion of well-nested and block degree 2 sentences for Arabic, Czech, Danish, Dutch, Portuguese, and Swedish, ranging from 95.4% of Portuguese sentences to 99.9% of Arabic sentences.[3] This definition covers a substantially larger set of sentences than projectivity does — an assumption of projectivity covers only 63.6% (Dutch) to 90.2% (Swedish) of examples (Table 3.1).

3.3 Gap Inheritance

Empirically, natural language sentences seem to be mostly well-nested block degree 2 trees, but well-nested block degree 2 trees are quite expensive to parse ($O(n^7)$ (Gómez-Rodríguez et al., 2011)). The parsing complexity comes from the fact that the definition allows two non-contiguous intervals of a projection to be tightly coupled, with an unbounded number of edges passing back and forth between the two intervals; however, this type of structure seems unusual for natural language. We therefore investigate if we can define further structural properties that are both appropriate for describing natural language trees and which admit more efficient parsing algorithms.

Let us first consider an example of a tree which both has gap degree at most one and satisfies well-nestedness, yet appears to be an unrealistic structure for a natural language syntactic tree. Consider a tree which is rooted at node x_{n+2}, which has one child, node x_{n+1}, whose projection is $[x_1, x_{n+1}] \cup [x_{n+3}, x_{2n+2}]$, with n children (x_1, ..., x_n), and each child x_i has a child at x_{2n-i+3}. This tree is well-nested, has gap degree 1, but all n of x_{n+1}'s children have edges into the other projection interval.

We introduce a further structural restriction in this section, and show that trees satisfying our new property can be parsed more efficiently with no drop in empirical coverage.

[3]While some of the treebank structures are ill-nested or have a larger gap degree because of annotation decisions, some linguistic constructions in German and Czech are ill-nested or require at least two gaps under any reasonable representation (Chen-Main and Joshi, 2010, 2012).

Definition 4. *A child is* gap inheriting *if its parent has gap degree 1 and it has descendants on both sides of its parent's gap. The* inheritance degree of a node *is the number of its children which inherit its gap. The* inheritance degree of a tree *is the maximum inheritance degree over all its nodes.*

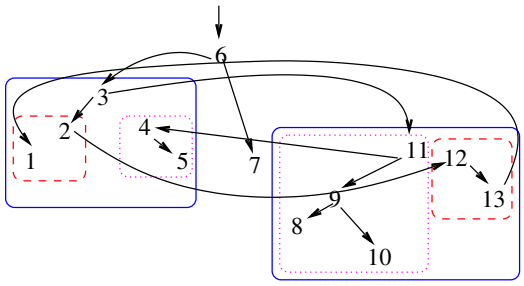
Figure 3.1 gives examples of trees with varying degrees of gap inheritance. Each projection of a node with a gap is shown with two matching rectangles. If a child has a projection rectangle nested inside each of the parent's projection rectangles, then that child inherits the parent's gap. Figure 3.1a shows a well-nested, block degree 2 tree (with inheritance degree 2), with both node 2 and node 11 inheriting their parent (node 3)'s gap (note that both the dashed and dotted rectangles each show up inside both of the solid rectangles). Figure 3.1b shows a tree with inheritance degree 1: there is now only one pair of rectangles (the dotted ones) which show up in both of the solid ones. Figure 3.1c shows a tree with inheritance degree 0: while there are gaps, each set of matching rectangles is contained within a single rectangle (projection interval) of its parent, i.e., the two dashed rectangles of node 2's projection are contained within the left interval of node 3; the two dotted rectangles of node 12's projection are contained within the right interval of node 3, etc.

We now ask:

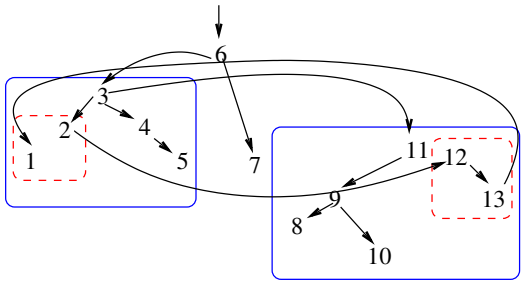
1. How often does gap inheritance occur in the parses of natural language sentences found in treebanks?

2. Furthermore, how often are there *multiple* gap inheriting children of the same node (inheritance degree at least two)?

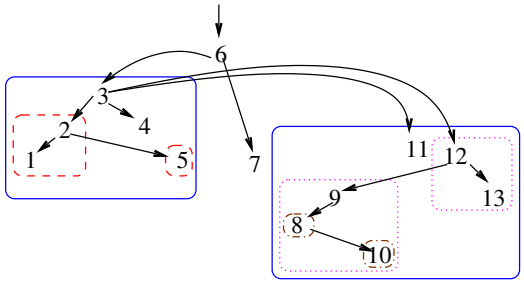
Table 3.1 shows what proportion of well-nested and block degree 2 trees have the added property of gap inheritance degree 0 or have gap inheritance degree 1. Over all six languages, there are *no* examples of multiple gap inheritance — trees with the 1 inheritance restriction have exactly the same empirical coverage as the unrestricted set of well-nested block degree 2 trees.



(a) Well-nested and Block Degree 2: The projections of both node 2 (the dashed red rectangles) and node 11 (dotted magenta) appear in both of node 3's intervals (the solid blue rectangles).



(b) +1-Inherit: Only node 2 inherits node 3's gap: the dashed red rectangles appear in each of the two solid blue rectangles.



(c) +0-Inherit: Even though node 3 has children with gaps (node 2 and node 12), neither of them inherit node 3's gap. There are several nodes with gaps, but every node with a gap is properly contained within just one of its parent's intervals.

Figure 3.1: Rectangles that match in color and style indicate the two projection intervals of a node, separated by a gap. In all three trees, node 3's two projection intervals are shown in the two solid blue rectangles. The number of children which inherit its gap vary, however; in 3.1a, two children have descendants within both sides; in 3.1b only one child has descendants on both sides; in 3.1c, none of its children do.

3.4 1-Inherit Trees

There are some reasons from syntactic theory why we might expect at most one child to inherit its parent's gap. Traditional Government and Binding theories of syntax (Chomsky, 1981) assume that there is an underlying projective (phrase structure) tree, and that gaps primarily arise through movement of subtrees (constituents). One of the fundamental assumptions of syntactic theory is that movement is upward in the phrase structure tree.[4]

Consider one movement operation and its effect on the gap degree of all other nodes in the tree: (a) it should have no effect on the gap degree of the nodes in the subtree itself, (b) it can create a gap for an ancestor node if it moves *out* of its projection interval, and (c) it can create a gap for a non-ancestor node if it moves *in* to its projection interval. Now consider which cases can lead to gap inheritance: in case (b), there is a single path from the ancestor to the root of the subtree, so the parent of the subtree will have no gap inheritance and any higher ancestors will have a single child inherit the gap created by this movement. In case (c), it is possible for there to be multiple children that inherit this newly created gap if multiple children had descendents on both sides. However, the assumption of upward movement in the phrase structure tree should rule out movement into the projection interval of a non-ancestor. Therefore, under these syntactic assumptions, we would expect at most one child to inherit a parent's gap.

3.4.1 Parsing Well-nested, Block degree 2, 1-Inherit Trees

Finding the optimal well-nested, block degree 2, 1-Inherit tree can be done by bottom-up constructing the tree for each node and its descendants. We can maintain subtrees with two intervals (two endpoints each) and one root ($O(n^5)$ space). Consider the most complicated possible case: a parent that has a gap, a (single) child which inherits the gap, and additional

[4]The Proper Binding Condition (Fiengo, 1977) asserts that a moved element leaves behind a *trace* (unpronounced element), which must be *c-commanded* (Reinhart, 1976) by the corresponding pronounced material in its final location. Informally, c-commanded means that the first node is descended from the lowest ancestor of the other that has more than one child.

children. An example of this is seen with the parent node 3 in Figure 3.1b.

This subtree can be constructed by first starting with the child spanning the gap, updating its root index to be the parent, and then expanding the interval indices to the left and right to include the other children. In each case, only one index needs to be updated at a time, so the optimal tree can be found in $O(n^6)$ time. In the Figure 3.1b example, the subtree rooted at 3 would be built by starting with the intervals $[1, 2] \cup [12, 13]$ rooted at 2, first adding the edge from 3 to 2 (so the root is updated to 3), then adding an edge from 3 to 4 to extend the left interval to $[1, 5]$, and then adding an edge from 3 to 11 to extend the right interval to $[8, 13]$. The subtree corresponds to the completed item $[1, 5] \cup [8, 13]$ rooted at 3.

This procedure corresponds to (Gómez-Rodríguez et al., 2011)'s $O(n^7)$ algorithm for parsing well-nested block degree 2 structures if the most expensive step (*Combine Shrinking Gap Centre*) is dropped; this step would only ever be needed if a parent node has more than one child inheriting its gap.

This is also similar in spirit to the algorithm described in (Satta and Schuler, 1998) for parsing a restricted version of TAG, in which there are some limitations on adjunction operations into the spines of trees.[5] That algorithm has similar steps and items, with the root portion of the item replaced with a node in a phrase structure tree (which may be a non-terminal).

3.5 Gap-minding Trees

The algorithm in the previous section used $O(n^5)$ space and $O(n^6)$ time. While more efficient than parsing in the space of well-nested and block degree 2 trees, this is still probably not practically implementable. Part of the difficulty lies in the fact that gap inheritance causes the two non-contiguous projection intervals to be coupled.

[5]That algorithm has a running time of $O(Gn^5)$, where as written G would likely add a factor of n^2 with bilexical selectional preferences; this can be lowered to n using the same technique as in (Eisner and Satta, 2000) for non-restricted TAG.

Definition 5. *A tree is called* gap-minding[6] *if it has gap degree at most one, is well-nested, and has gap inheritance degree 0.*

Gap-minding trees still have good empirical coverage (between 90.4% for Dutch and 97.7% for Swedish). We now turn to the parsing of gap-minding trees and show how a few consequences of its definition allow us to use items ranging over only *one* interval.

In Figure 3.1c, notice how each rectangle has edges incoming from *exactly one* node. This is not unique to this example; all projection intervals in a gap-minding tree have incoming edges from exactly one node outside the interval.

Claim 1. *Within a gap-minding tree, consider any node h with a gap (i.e., h's projection forms two non-contiguous intervals $[x_i, x_j] \cup [x_k, x_l]$). Let g be the parent of h.*

1. *For each of the intervals of h's projection:*
   (a) *If the interval contains h, the only edge incoming to that interval is from g to h.*
   (b) *If the interval does not contain h, all edges incoming to that interval come from h.*
2. *For the gap interval ($[x_{j+1}, x_{k-1}]$):*
   (a) *If the interval contains g, then the only edge incoming is from g's parent to g*
   (b) *If the interval does not contain g, then all edges incoming to that interval come from g.*

As a consequence of the above, $[x_i, x_j] \cup \{h\}$ forms a gap-minding tree rooted at h, $[x_k, x_l] \cup \{h\}$ also forms a gap-minding tree rooted at h, and $[x_{j+1}, x_{k-1}] \cup \{g\}$ forms a gap-minding tree rooted at g.

Proof. (Part 1): Assume there was a directed edge (x, y) such that y is inside a projection interval of h and x is not inside the same interval, and $x \neq y \neq h$. y is a descendant of h

[6]The terminology is a nod to the London Underground but imagines parents admonishing children to mind the gap.

since it is contained in h's projection. Since there is a directed edge from x to y, x is y's parent, and thus x must also be a descendant of h and therefore in another of h's projection intervals. Since x and y are in different intervals, then whichever child of h that x and y are descended from would have inherited h's gap, leading to a contradiction.

(Part 2): First, suppose there existed a set of nodes in h's gap which were not descended from g. Then g has a gap over these nodes. (g clearly has descendants on each side of the gap, because all descendants of h are also descendants of g). h, g's child, would then have descendants on both sides of g's gap, which would violate the property of no gap inheritance. It is also not possible for there to be edges incoming from other descendants of g outside the gap, as that would imply another child of g being ill-nested with respect to h. □

From the above, we can build gap-minding trees using only single intervals, potentially with a single node outside of the interval. Our objective is to find the maximum scoring gap-minding tree, in which the score of a tree is the sum of the scores of its edges. Let $\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m}))$ indicate the score of the directed edge from h to m.

Therefore, the main type of sub-problems we will use are:

1. $\mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{h}]$: The maximum score of any gap-minding tree, rooted at h, with vertices $[i, j] \cup \{h\}$ (h may or may not be within $[i, j]$).

This improves our space requirement, but not necessarily the time requirement. For example, if we built up the subtree in Figure 3.1c by concatenating the three intervals $[1, 5]$ rooted at 3, $[6, 7]$ rooted at 6, and $[8, 13]$ rooted at 3, and add the edge $6 \rightarrow 3$, we would still need 6 indices to describe this operation (the four interval endpoints and the two roots), and so we have not yet improved the running time over the *Inherit-1* case.

By part 2, we can concatenate one interval of a child with its gap, knowing that the gap is entirely descended from the child's parent, and forget the concatenation split point between the parent's other descendants and this side of the child. This allows us to substitute all operations involving 6 indices with two operations involving just 5 indices. For example,

in Figure 3.1c, we could first merge $[6, 7]$ rooted at 6 with $[8, 13]$ rooted at 3 to create an interval $[6, 13]$ and say that it is descended from 6, with the rightmost side descended from its child 3. That step required 5 indices. The following step would merge this concatenated interval ($[6, 13]$ rooted at 6 and 3) with $[1, 5]$ rooted at 3. This step also requires only 5 indices.

Our helper subtype we make use of is then:

2. $\mathbf{D}[\mathbf{i}, \mathbf{j}, \mathbf{h}, \mathbf{m}, \mathbf{b}]$: The maximum score of any set of two gap-minding trees, one rooted at h, one rooted at m, with vertices $[i, j] \cup \{h, m\}$ ($m \notin [i, j]$, h may or may not be in $[i, j]$), such that for some k, vertices $[i, k]$ are in the tree rooted at h if $b =$ *true* (and at m if $b =$ *false*), and vertices $[k + 1, j]$ are in the tree rooted at m (h).

Consider an optimum scoring gap-minding tree T rooted at p with vertices $V = [i, j] \cup \{h\}$ and edges E, where $E \neq \emptyset$. The form of the dynamic program may depend on whether:

- h is within (i, j) **(I)** or external to $[i, j]$ **(E)**[7]

We can exhaustively enumerate all possibilities for T by considering all valid combinations of the following binary cases:

- h has a single child **(S)** or multiple children **(M)**
- i and j are descended from the same child of h **(C)** or different children of h **(D)**

Note that case **(S/D)** is not possible: i and j cannot be descended from different children of h if h has only a single child. We therefore need to find the maximum scoring tree over the three cases of **S/C**, **M/C**, and **M/D**.

Claim 2. *Let T be the optimum scoring gap-minding tree rooted at h with vertices $V = [i, j] \cup \{h\}$. Then T and its score are derived from one of the following:*

[7] In the discussion we will assume that $h \neq i$ and $h \neq j$, since any optimum solution with $V = [i, j] \cup \{i\}$ and a root at i will be equivalent to $V = [i + 1, j] \cup \{i\}$ rooted at i (and similarly for $h = j$).

S/C *If h has a single child m in T, then if $h \in (i, j)$* **(I)***, T's score is* $\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m})) + \mathbf{C}[\mathbf{i}, \mathbf{h}-\mathbf{1}, \mathbf{m}] + \mathbf{C}[\mathbf{h}+\mathbf{1}, \mathbf{j}, \mathbf{m}]$*; if $h \notin [i, j]$* **(E)***, T's score is* $\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m})) + \mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{m}]$.

M/C *If h has multiple children in T and i and j are descended from the same child m in T, then there is a split point k such that T's score is:* $\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m})) + \mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{m}] + \mathbf{D}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{h}, \mathbf{m}, \mathbf{T}]$ *if x is on the left side of its own gap, and T's score is:* $\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m})) + \mathbf{C}[\mathbf{k}, \mathbf{j}, \mathbf{m}] + \mathbf{D}[\mathbf{i}, \mathbf{k}-\mathbf{1}, \mathbf{h}, \mathbf{m}, \mathbf{F}]$ *if m is on the right side.*

M/D *If h has multiple children in T and i and j are descended from different children in T, then there is a split point k such that T's score is* $\mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{h}] + \mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{h}]$.

T has the maximum score over each of the above cases, for all valid choices of m and k.

Proof. **Case S/C**: If h has exactly one child m, then the tree can be decomposed into the edge from h to m and the subtree rooted at m. If h is outside the interval, then the maximum scoring such tree is clearly $\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m})) + \mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{m}]$. If h is inside, then m has a gap across h, and so using Claim 1, the maximum scoring tree rooted at h with a single child m has score of $\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m})) + \mathbf{C}[\mathbf{i}, \mathbf{h}-\mathbf{1}, \mathbf{m}] + \mathbf{C}[\mathbf{h}+\mathbf{1}, \mathbf{j}, \mathbf{m}]$.

Case M/C: If there are multiple children and the endpoints are descended from the same child m, then the child m has to have gap degree 1. m itself is on either the left or right side of its gap. For the moment, assume m is in the left interval. By Claim 1, we can split up the score of the tree as the score of the edge from h to m ($\mathbf{Score}(\mathrm{Edge}(\mathbf{h}, \mathbf{m}))$), the score of the subtree corresponding to the projection of m to the left of its gap ($\mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{m}]$), and the score of the subtrees rooted at h with its remaining children and the subtree rooted at m corresponding to the right side of m's projection ($\mathbf{D}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{h}, \mathbf{m}, \mathbf{T}]$). The case in which m is on the right side of its gap is symmetric.

Case M/D: If there are multiple children and the endpoints are descended from different children of h, then there must exist a split point k that partitions the children of h into two non-empty sets, such that each child's projection is either entirely on the left or entirely on the right of the split point. We show one such split point to demonstrate that there always

exists at least one. Let m be the child of h that i is descended from, and let m_l and m_r be m's leftmost and right descendants, respectively.[8] Consider all the children of h (whose projections taken together partition $[i, j] - \{h\}$). No child can have descendants both to the left of m_r and to the right of m_r, because otherwise that child and m would be *ill-nested*. Therefore we can split up the interval at m_r to have two gap-minding trees, both rooted at h. The score of T is then the sum of the scores of the best subtree rooted at h over $[i, k]$ ($\mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{h}]$) and the score of the best subtree rooted at h over $[k + 1, j]$ ($\mathbf{C}[\mathbf{k} + \mathbf{1}, \mathbf{j}, \mathbf{h}]$).

The above cases cover all non-empty gap-minding trees, so the maximum will be found.

□

Using Claim 2 to Devise an Algorithm The above claim showed that any problem of type $\mathbf{C}$ can be decomposed into subproblems of types $\mathbf{C}$ and $\mathbf{D}$. From the definition of $\mathbf{D}$, any problem of type $\mathbf{D}$ can clearly be decomposed into two problems of type $\mathbf{C}$ — simply split the interval at the split point known to exist and assign h or m as the roots for each side of the interval, as prescribed by the boolean b:

$$\mathbf{D}(\mathbf{i}, \mathbf{j}, \mathbf{h}, \mathbf{m}, \mathbf{T}) = \mathbf{max_k C}[\mathbf{i}, \mathbf{k}, \mathbf{h}] + \mathbf{C}[\mathbf{k} + \mathbf{1}, \mathbf{j}, \mathbf{m}] \tag{3.1}$$

$$\mathbf{D}(\mathbf{i}, \mathbf{j}, \mathbf{h}, \mathbf{m}, \mathbf{F}) = \mathbf{max_k C}[\mathbf{i}, \mathbf{k}, \mathbf{m}] + \mathbf{C}[\mathbf{k} + \mathbf{1}, \mathbf{j}, \mathbf{h}]$$

Algorithm 1 makes direct use of the above claims. Algorithm 1 builds up trees in increasing sizes of $[i, j] \cup \{h\}$. The tree in $\mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{h}]$ corresponds to the maximum of four subroutines: SingleChild (**S/C**), EndpointsDiff (**M/D**), EndsFromLeftChild (**M/C**), and EndsFromRightChild (**M/C**). The $\mathbf{D}$ subproblems are filled in with the subroutine Max2Subtrees, which uses the above discussion. The maximum score of any gap-minding tree is then found in $\mathbf{C}[\mathbf{1}, \mathbf{n}, \mathbf{0}]$, and the tree itself can be found using backpointers.

[8] Note that $m_l = i$ by construction, and $m_r \neq j$ (because the endpoints are descended from different children).

Algorithm 1: MaxGapMindingTree

Init: $\forall_{i \in [1,n]} C[i,i,i] = 0$

for $size = 0$ *to* $n-1$ **do**

  for $i = 1$ *to* $n - size$ **do**

    $j = i + size$

    /* Endpoint parents */

    if $size > 0$ **then**

      $C[i,j,i] = C[i+1,j,i]$

      $C[i,j,j] = C[i,j-1,j]$

    /* Interior parents */

    for $h = i+1$ *to* $j-1$ **do**

      $C[i,j,h] = \max$ (SingleChild(i,j,h),

      EndpointsDiff(i,j,h),

      EndsFromLeftChild(i,j,h),

      EndsFromRightChild(i,j,h))

    /* Exterior parents */

    forall the $h \in [0, i-1] \cup [j+1, n]$ **do**

      $C[i,j,h] = \max$ (SingleChild(i,j,h),

      EndpointsDiff(i,j,h),

      EndsFromLeftChild(i,j,h),

      EndsFromRightChild(i,j,h))

    /* Helper subproblems */

    for $h \in [0, n]$ **do**

      forall the $m \in PosChild[h] \wedge m \notin [i,j]$ **do**

        if $h \neq j$ **then**

          $D[i,j,h,m,T] = Max2Subtrees(i,j,h,m,T)$

        if $h \neq i$ **then**

          $D[i,j,h,m,F] = Max2Subtrees(i,j,h,m,F)$

Final answer: $C[1,n,0]$

Function SingleChild(i,j,h)

$M = PosChild[h] \cap [i, j]$

`/* Interior h */`

if $h > i \wedge h < j$ **then**

return $\max_{m \in M} C[i, h-1, m] + C[h+1, j, m] + Score(\text{Edge}(h, m))$

`/* Exterior h */`

else

return $\max_{m \in M} C[i, j, m] + Score(\text{Edge}(h, m))$

Function EndpointsDiff(i,j,h)

return $\max_{k \in [i, j-1]} C[i, k, h] + C[k+1, j, h]$

Function EndsFromLeftChild(i,j,h)

`/* Interior h */`

if $h > i \wedge h < j$ **then**

$M = PosChild[h] \cap [i, h-1]$

forall the $m \in M \wedge m < h$ **do**

$K[m] = [m, h-1]$

`/* Exterior h */`

else

$M = PosChild[h] \cap [i, j]$

forall the $m \in M$ **do**

$K[m] = [m, j-2]$

return $\max_{m \in M, k \in K[m]} C[i, k, m] + Score(\text{Edge}(h, m)) + D[k+1, j, h, m, T]$

Function EndsFromRightChild(i,j,h)

```
/* Interior h                                                    */
```

if $h > i \wedge h < j$ **then**
  $M = PosChild[h] \cap [h+1, j]$

  forall the $m \in M \wedge m > h$ **do**
    $K[m] = [h+1, m]$

```
/* Exterior h                                                    */
```

else
  $M = PosChild[h] \cap [i, j]$

  forall the $m \in M$ **do**
    $K[m] = [i+2, m]$

return $\max_{m \in M, k \in K[m]} C[k, j, m] + Score(\text{Edge}(h, m)) + D[i, k-1, h, m, F]$

Function Max2Subtrees(i,j,h,m,hOnLeft)

```
/* Interior h                                                    */
```

if $h \geq i \wedge h \leq j$ **then**

  if *hOnLeft* **then**
    $K = [h, j-1]$

    return $\max_{k \in K} C[i, k, h] + C[k+1, j, m]$

  else
    $K = [i, h-1]$

    return $\max_{k \in K} C[i, k, m] + C[k+1, j, h]$

```
/* Exterior h                                                    */
```

else
  $K = [i, j-1]\}$

  if *hOnLeft* **then**
    return $\max_{k \in K} C[i, k, h] + C[k+1, j, m]$

  else
    return $\max_{k \in K} C[i, k, m] + C[k+1, j, h]$

3.5.1 Runtime analysis

If the input is assumed to be the complete graph (any word can have any other word as its parent), then the above algorithm takes $O(n^5)$ time. The most expensive steps are **M/C**, which take $O(n^2)$ time to fill in each of the $O(n^3)$ $\mathbf{C}$ cells. and solving a $\mathbf{D}$ subproblem, which takes $O(n)$ time on each of the $O(n^4)$ possible such problems.

Pruning: In practice, the set of edges considered (m) is not necessarily $O(n^2)$. Many edges can be ruled out beforehand, either based on the distance in the sentence between the two words (Eisner and Smith, 2010), the predictions of a local ranker (Martins et al., 2009), or the marginals computed from a simpler parsing model (Carreras, Collins, and Koo, 2008).

If we choose a pruning strategy such that each word has at most k potential parents (incoming edges), then the running time drops to $O(kn^4)$. The five indices in an **M/C** step were: i, j, k, h, and m. As there must be an edge from h to m, and m only has k possible parents, there are now only $O(kn^4)$ valid such combinations. Similarly, each $\mathbf{D}$ subproblem (which ranges over i, j, k, h, m) may only come into existence because of an edge from h to m, so again the runtime of these such steps drops to $O(kn^4)$.

3.6 Extension to Grandparent Factorizations

The ability to define slightly non-local features has been shown to improve parsing performance. In this section, we assume a *grandparent-factored* model, where the score of a tree is now the sum over scores of $\mathrm{Grand}(g, h, m)$ triples, where (g, h) and (h, m) are both directed edges in the tree. Let $\mathbf{Score}(\mathrm{Grand}(\mathbf{g}, \mathbf{h}, \mathbf{m}))$ indicate the score of this grandparent-parent-child triple. We now show how to extend the above algorithm to find the maximum scoring gap-minding tree with grandparent scoring.

Our two subproblems are now $\mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{h}, \mathbf{g}]$ and $\mathbf{D}[\mathbf{i}, \mathbf{j}, \mathbf{h}, \mathbf{m}, \mathbf{b}, \mathbf{g}]$; each subproblem has been augmented with an additional grandparent index g, which has the meaning that g is h's parent. Note that g must be outside of the interval $[i, j]$ (if it were not, a cycle would be introduced). Edge scores are now computed over (g, h, m) triples. In particular, claim 2 is

modified:

Claim 3. *Let T be the optimum scoring gap-minding tree rooted at h with vertices $V = [i,j] \cup \{h\}$, where $h \in (i,j)$* **(I)***, with a grandparent index g ($g \notin V$). Then T and its score are derived from one of the following:*

S/C *If h has a single child m in T, then if $h \in (i,j)$* **(I)***, T's score is* $\mathbf{Score}(\mathrm{Grand}(\mathbf{g},\mathbf{h},\mathbf{m})) + \mathbf{C}[\mathbf{i},\mathbf{h-1},\mathbf{m},\mathbf{h}] + \mathbf{C}[\mathbf{h+1},\mathbf{j},\mathbf{m},\mathbf{h}]$*; if $h \notin [i,j]$* **(E)***, T's score is* $\mathbf{Score}(\mathrm{Grand}(\mathbf{g},\mathbf{h},\mathbf{m})) + \mathbf{C}[\mathbf{i},\mathbf{j},\mathbf{m},\mathbf{h}]$.

M/C *If h has multiple children in T and i and j are descended from the same child m in T, then there is a split point k such that T's score is:* $\mathbf{Score}(\mathrm{Grand}(\mathbf{g},\mathbf{h},\mathbf{m})) + \mathbf{C}[\mathbf{i},\mathbf{k},\mathbf{m},\mathbf{h}] + \mathbf{D}[\mathbf{k+1},\mathbf{j},\mathbf{h},\mathbf{m},\mathbf{T},\mathbf{g}]$ *if m is on the left side of its own gap, and T's score is:* $\mathbf{Score}(\mathrm{Grand}(\mathbf{g},\mathbf{h},\mathbf{m})) + \mathbf{C}[\mathbf{k},\mathbf{j},\mathbf{m},\mathbf{h}] + \mathbf{D}[\mathbf{i},\mathbf{k-1},\mathbf{h},\mathbf{m},\mathbf{F},\mathbf{g}]$ *if m is on the right side.*

M/D *If h has multiple children in T and i and j are descended from different children in T, then there is a split point k such that T's score is* $\mathbf{C}[\mathbf{i},\mathbf{k},\mathbf{h},\mathbf{g}] + \mathbf{C}[\mathbf{k+1},\mathbf{j},\mathbf{h},\mathbf{g}]$.

T has the maximum score over each of the above cases, for all valid choices of m and k.

Note that for subproblems rooted at h, g is the grandparent index, while for subproblems rooted at m, g is the updated grandparent index. The $\mathbf{D}$ subproblems with the grandparent index are shown below:

$$\mathbf{D}(\mathbf{i},\mathbf{j},\mathbf{h},\mathbf{m},\mathbf{T},\mathbf{g}) = \mathbf{max_k}\mathbf{C}[\mathbf{i},\mathbf{k},\mathbf{h},\mathbf{g}] + \mathbf{C}[\mathbf{k+1},\mathbf{j},\mathbf{m},\mathbf{h}] \tag{3.2}$$
$$\mathbf{D}(\mathbf{i},\mathbf{j},\mathbf{h},\mathbf{m},\mathbf{F},\mathbf{g}) = \mathbf{max_k}\mathbf{C}[\mathbf{i},\mathbf{k},\mathbf{m},\mathbf{h}] + \mathbf{C}[\mathbf{k+1},\mathbf{j},\mathbf{h},\mathbf{g}]$$

We have added another index which ranges over n, so without pruning, we have now increased the running time to $O(n^6)$. However, every step now includes both a g and a h (and often an m), so there is at least one implied edge in every step. If pruning is done in such a way that each word has at most k parents, then each word's set of grandparent and

parent possibilities is at most k^2. To run all of the **S/C** steps, we therefore need $O(k^2n^3)$ time; for all of the **M/C** steps, $O(k^2n^4)$ time; for all of the **M/D** steps, $O(kn^4)$; for all of the **D** subproblems, $O(k^2n^4)$. The overall running time is therefore $O(k^2n^4)$, and we have shown that when edges are sufficiently pruned, grandparent factors add only an extra factor of *k*, and not a full extra factor of n.

3.7 Experiments

The space of projective trees is strictly contained within the space of gap-minding trees which is strictly contained within spanning trees. Which space is most appropriate for natural language parsing may depend on the particular language and the type and frequencies of non-projective structures found in it. In this section we compare the parsing accuracy across languages for a parser which uses either the Eisner algorithm (projective), MST (spanning trees), or MaxGapMindingTree (gap-minding trees) as its decoder for both training and inference.

We implemented both the basic gap-minding algorithm and the gap-minding algorithm with grandparent scoring as extensions to MSTParser[9]. MSTParser (McDonald et al., 2005b,a) uses the Margin Infused Relaxed Algorithm (Crammer and Singer, 2003) for discriminative training. Training requires a decoder which produces the highest scoring tree (in the space of valid trees) under the current model weights. This same decoder is then used to produce parses at test time. MSTParser comes packaged with the Eisner algorithm (for projective trees) and MST (for spanning trees). MSTParser also includes two second order models: one of which is a projective decoder that also scores siblings (Proj+Sib) and the other of which produces non-projective trees by rearranging edges after producing a projective tree (Proj+Sib+Rearr). We add a further decoder with the algorithm presented here for gap minding trees. The gap-minding decoder has both an edge-factored implementation and a version which scores grandparents as well.[10]

[9] http://sourceforge.net/projects/mstparser/

[10] The grandparent features used were identical to the features provided within MSTParser for the second-

The gap-minding algorithm is much more efficient when edges have been pruned so that each word has at most k potential parents. We use the weights from the trained MST models combined with the Matrix Tree Theorem (Smith and Smith, 2007; Koo, Globerson, Carreras, and Collins, 2007; McDonald and Satta, 2007) to produce marginal probabilities of each edge. We wanted to be able to both achieve the running time bound and yet take advantage of the fact that the size of the set of reasonable parent choices is variable. We therefore use a hybrid pruning strategy: each word's set of potential parents is the *smaller* of a) the top k parents (we chose $k = 10$) or b) the set of parents whose probabilities are above a threshold (we chose $th = .001$). The running time for the gap-minding algorithm is then $O(kn^4)$; with the grandparent features the gap-minding running time is $O(k^2n^4)$.

	Arabic	Czech	Danish	Dutch	Portuguese	Swedish
Training	1460	72703	5190	13349	9071	11042
Testing	146	365	322	386	288	389

Table 3.2: Number of sentences in the CoNLL-X shared task datasets.

The training and test sets for the six languages come from the CoNLL-X shared task.[11] We train the gap-minding algorithm on sentences of length at most 100[12] (the vast majority of sentences). The projective and MST models are trained on all sentences and are run without any pruning. The Czech training set is much larger than the others (Table 3.2) and so for Czech only the first 10,000 training sentences were used. Testing is on the full test set, with no length restrictions.

The results are shown in Table 3.3. The first three lines show the first order gap-minding decoder compared with the first order projective and MST decoders. The gap-minding

order sibling parsers, with one exception — many features are conjoined with a direction indicator, which in the projective case has only two possibilities. We replaced this two-way distinction with a six-way distinction of the six possible orders of the grandparent, parent, and child.

[11]MSTParser produces labeled dependencies on CoNLL formatted input. We replace all labels in the training set with a single dummy label to produce unlabeled dependency trees.

[12]Because of long training times, the gap-minding with grandparent models for Portuguese and Swedish were trained on only sentences up to 50 words.

	Arabic	Czech	Danish	Dutch	Portuguese	Swedish
Proj.	78.0	80.0	88.2	79.8	87.4	86.9
MST	78.0	80.4	88.1	84.6	86.7	86.2
Gap-Mind	77.6	80.8	88.6	83.9	86.8	86.0
Proj+Sib	78.2	80.0	88.9	81.1	87.5	88.1
+Rearr	78.5	81.3	89.3	85.4	88.2	87.7
GM+Grand	78.3	82.1	89.1	84.6	87.7	88.5

Table 3.3: Unlabeled Attachment Scores on the CoNLL-X shared task test set.

decoder does better than the projective decoder on Czech, Danish, and Dutch, the three languages with the most non-projectivity, even though it was at a competitive disadvantage in terms of both pruning and (on languages with very long sentences) training data. The gap-minding decoder with grandparent features is better than the projective decoder with sibling features on all six of the languages. On some languages, the local search decoder with siblings has the absolute highest accuracy in Table 3.3; on other languages (Czech and Swedish) the gap-minding+grandparents has the highest accuracy. While not directly comparable because of the difference in features, the promising performance of the gap-minding+grandparents decoder shows that the space of gap-minding trees is larger than the space of projective trees, yet unlike spanning trees, it is tractable to find the best tree with higher order features. It would be interesting to extend the gap-minding algorithm to include siblings as well.

3.8 Extension to Arbitrary Gap Degree

For well-nested structures generally, the running time increases exponentially with the gap degree: well-nested structures with a gap degree bounded by a constant k can be parsed in time $O(n^{5+2k})$ (Gómez-Rodríguez et al., 2011). When $k = 1$, this gives us the familiar $O(n^7)$ parsing time for well-nested structures with gap degree at most 1.

However, with the added restriction of no gap inheritance, the restriction to gap degree one is unnecessary. In this section, we show how to modify Algorithm 1 to find the maximum scoring tree that has no gap inheritance, is well-nested, and can have *arbitrary gap degree*. This change has *no* effect on the running time of the algorithm: the maximum scoring tree in this class can still be found in $O(n^5)$.

	Arabic	Czech	Danish	Dutch	Portuguese	Swedish	Parsing
Well-nested +block degree 2	1458 (99.9)	72321 (99.5)	5175 (99.7)	12896 (96.6)	8650 (95.4)	10955 (99.2)	$O(n^7)$
+0-Inherit	1394 (95.5)	70695 (97.2)	4985 (96.1)	12068 (90.4)	8481 (93.5)	10787 (97.7)	$O(n^5)$
Well-nested + 0-Inherit	1394 (95.5)	70883 (97.5)	4986 (96.1)	12116 (90.8)	8825 (97.3)	10792 (97.7)	$O(n^5)$

Table 3.4: Empirical coverage when the gap degree restriction is dropped.

The effect on empirical coverage of dropping the gap degree restriction is in Table 3.4. For Portuguese, the coverage is actually *higher* for no inheritance but *unbounded gap degree* than the case with gap degree 1 (block degree 2) but *unbounded inheritance degree*.

We can parse well-nested trees with no gap inheritance by modifying the definition of the **D** helper function:

$\mathbf{D}'[\mathbf{i}, \mathbf{j}, \mathbf{h}, \mathbf{m}, \mathbf{b}]$: The maximum score of the score of the edge from h to m plus the scores of any set of *two or more* gap-minding trees, *alternating between* trees rooted at h and rooted at m with vertices $[i, j] \cup \{h, m\}$ such that vertex i is in a tree rooted at h if $b =$ *true* (and at m if $b =$ *false*), and vertex j is always in a tree rooted at m.

The intuition is that now, rather than concatenating together just one pair of a node's interval and its gap, we can repeatedly alternate between concatenating on another interval or concatenating on another gap. No gap inheritance means that all the projection intervals of a node are independent given that node, and this holds equally well for an arbitrary number of intervals as it did when we had just two (gap degree one).

The two cases which need to be updated are below:

Claim 4. *Let T be the optimum scoring tree with no gap inheritance rooted at h with*

vertices $V = [i, j] \cup \{h\}$. Then T and its score are derived from one of the following:

M/C *If h has multiple children in T and i and j are descended from the same child m in T, then there is a split point k such that T's score is:* $\mathbf{C[i, k, m] + D'[k + 1, j, h, m, T]}$.

M/C: Let k be the rightmost vertex in m's leftmost projection interval. By no gap inheritance, we can split up the score of the tree as the the score of the subtree corresponding to m's leftmost interval ($\mathbf{C[i, k, m]}$), and the score of the edge from h to m, the score of the subtrees rooted at h with its remaining children and the subtrees rooted at m corresponding to all the other intervals of x's projection ($\mathbf{D'[k + 1, j, h, m, T, F]}$).

The other case which needs to be updated is the definition of $\mathbf{D'}$:

$$\mathbf{D'(i, j, h, m, T)} = \max_k \begin{cases} C[i, k, h] + D'[k + 1, j, h, m, F] \\ C[i, k, h] + C[k + 1, j, m] + Score(\text{Edge}(h, m)) \end{cases} \tag{3.3}$$

$$\mathbf{D'(i, j, h, m, F)} = \max_k C[i, k, m] + D'[k + 1, j, h, m, T]$$

When $b =$ *true*, $\mathbf{D'}$ is made up of two or more trees that alternate being rooted at h and m such that the leftmost subtree is rooted at h and the rightmost subtree is rooted at m. This could either have exactly two subtrees (base case), in which we concatenate two individual trees ($\mathbf{C[i, k, h]}$ and $\mathbf{C[k + 1, j, m]}$) and add the score of the edge from h to m. Otherwise, this interval has four or more subtrees and so is created by concatenating an interval rooted at h ($\mathbf{C[i, k, h]}$) to a $\mathbf{D'}$ alternating interval that begins with a tree rooted at m (and so $b =$ *false*).

When $b =$ *false*, then the number of subtrees is at least three and odd, and so this can only be built by concatenating a interval rooted at m to an existing alternating interval that begins with a tree rooted at h.

We do not pursue this modification experimentally at this time, as the next chapter will show a different type of non-projectivity (defined over edges, rather than subtrees) that has higher coverage in every language and lower asymptotic parsing time.

3.9 Conclusion

Gap inheritance, a structural property on trees, has implications both for natural language syntax and for natural language parsing. We have shown that the well-nested block degree 2 trees present in natural language treebanks *all* have zero or one children inherit each parent's gap. We also showed that the assumption of 1 gap inheritance removes a factor of n from parsing time, and the further assumption of 0 gap inheritance removes yet another factor of n. More recent work has shown that restricting the 1 gap inherit class to trees that are *head-split* (requiring a child that gaps over its parent to also inherit its parent's gap) can also be parsed in $O(n^5)$, with almost the same coverage as the 1 gap inherit class (Satta and Kuhlmann, 2013). The space of gap-minding trees provides a closer fit to naturally occurring linguistic structures than the space of projective trees, and unlike spanning trees, the inclusion of higher order factors does not substantially increase the difficulty of finding the maximum scoring tree in that space. Furthermore, we showed that unlike general well-nested trees, which have a parsing complexity that increases exponentially with the gap degree, arbitrarily large gap degrees pose no additional complexity for well-nested trees without gap inheritance.

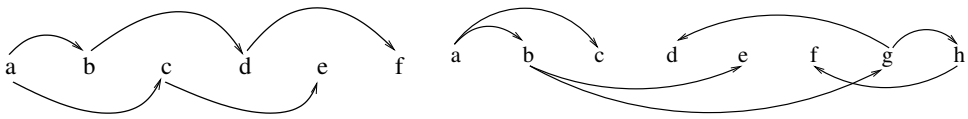
Chapter 4

Finding Optimal 1-Endpoint-Crossing Trees

Material in this chapter previously appeared in Pitler et al. (2013).

4.1 Introduction

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 4.1). 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 4.3). This is the key insight that allows efficient parsing; the $O(n^4)$ parsing algorithm is presented in Section 4.4. 1-Endpoint-Crossing trees are a subclass of 2-planar graphs (Section 4.6.1), 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 4.6.2; additional examples of other phenomena are discussed in Section 6.1.3).



(a) (b)

Figure 4.1: 4.1a is 1-Endpoint-Crossing, but is neither block degree 2 nor well-nested; 4.1b is gap-minding but not 2-planar.

4.2 Additional Definitions of Non-Projectivity

Definition 6. *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 7. *A dependency tree is **1-Endpoint-Crossing** if for any edge e, all edges that cross e share an endpoint p.*

Table 4.1 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]

Definition 8. *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* $\subset$ *gap-minding* $\subset$ *well-nested with block degree at most 2*. In Section 4.6.1, we prove the somewhat surprising fact that *1-Endpoint-Crossing* $\subset$ *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 4.1a), and *gap-minding* $\not\subseteq$ *2-planar* (Figure 4.1b).

[1]Conventional edges from the artificial root node to the root(s) of the sentence reduce the empirical coverage of 1-Endpoint-Crossing trees. When these artificial root edges are excluded, 97.0-99.8% of trees are 1-Endpoint-Crossing (Table 4.2). These edges have no effect on the coverage of well-nested trees with block degree at most 2, gap-minding trees, or projective trees.

	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 4.1: 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.

	Arabic	Czech	Danish	Dutch	Portuguese	Swedish
1-Endpoint-Crossing	1457 (99.8)	71810 (98.8)	5144 (99.1)	12785 (95.8)	9007 (99.3)	10902 (98.7)
Excluding artificial root edges	1457 (99.8)	72094 (99.2)	5156 (99.3)	12949 (97.0)	9007 (99.3)	10906 (98.8)

Table 4.2: Proportion of dependency trees that are 1-Endpoint-Crossing when edges from the artificial root are excluded.

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

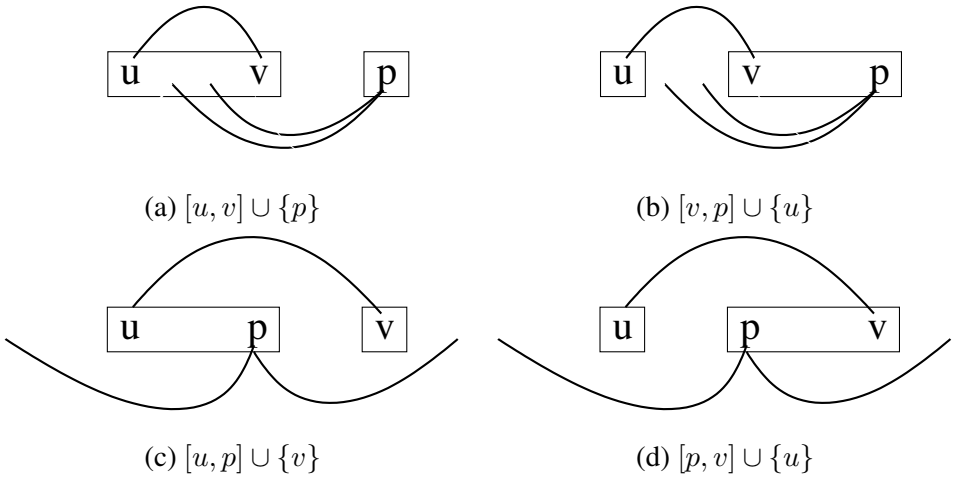
We introduce notation to facilitate the discussion:

Definition 9. *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

[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.



(a) $[u, v] \cup \{p\}$ (b) $[v, p] \cup \{u\}$

(c) $[u, p] \cup \{v\}$ (d) $[p, v] \cup \{u\}$

Figure 4.2: An edge e_{uv} and $\mathcal{P}t(e_{uv}) = p$ form two sets of isolated crossing regions (Lemma 1). 4.2a and 4.2b show $p \notin (u, v)$; 4.2c and 4.2d show $p \in (u, v)$.

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 10. *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* $\in (i, j)$

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 4.2a)**: 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 4.2b)**: 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 4.2c)**: 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 4.2d)**: Symmetric to the above.

□

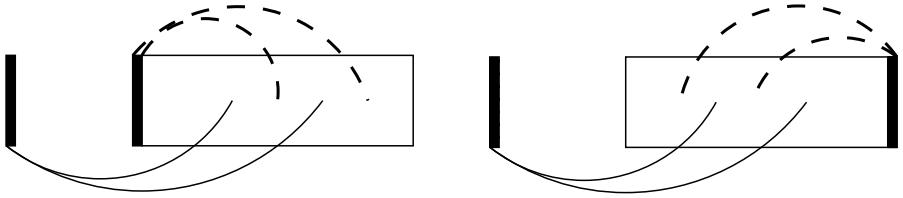
4.4 Parsing Algorithm

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.

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 4.3). $L[i, j, x]$, 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).

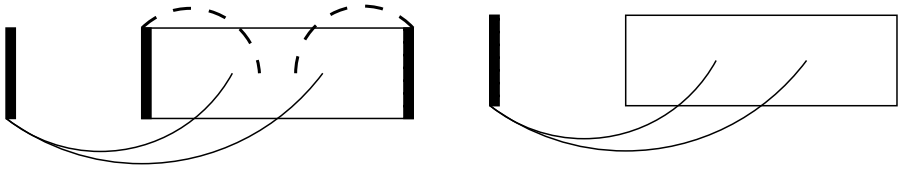
These distinctions allow the 1-Endpoint-Crossing property to be globally enforced; crossing edges in one region may constrain edges in another. For example, consider that Figure 4.2a allows edges with an endpoint at v to cross the edges from p, while Figure 4.2b allows edges from u into (v, p). Both *simultaneously* would cause a 1-Endpoint-Crossing violation for the edges in $\mathcal{P}(e_{uv})$. Figures 4.4 and 4.6 show valid combinations of the sub-problems in Figure 4.3.

The full dynamic program is shown in Section 4.5. The final answer must be a valid



(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**)

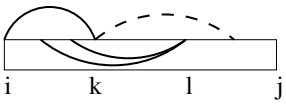
(d) **N**either

Figure 4.3: Isolated crossing region sub-problems.

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, 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.

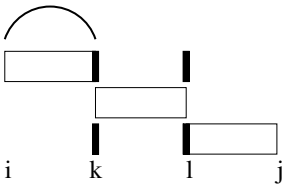
In the following subsections, we show that the optimal parse for each type of sub-problem can be decomposed into smaller valid sub-problems. If we take the maximum over all these possible combinations of smaller solutions, we can find the maximum scoring parse for that sub-problem. Note that the overall 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.

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



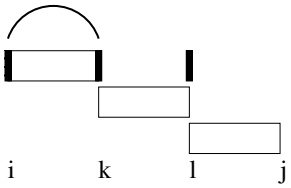
(i) **If the dashed edge exists**:

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

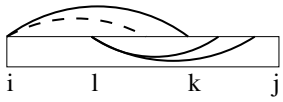


(ii) **If no edges like the dashed edge exist**:

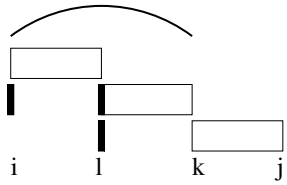
All edges from l into (i, k) may choose either i or k as their $\mathcal{P}t$. The interval decomposes into $S[e_{ik}] + LR[i, k, l] + Int[k, l] + Int[l, j]$:



(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 $S[e_{ik}] + R[i, l, k] + Int[l, k] + L[k, j, l]$:

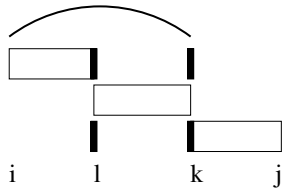


Figure 4.4: Decomposing an $Int[i, j]$ sub-problem, with $\mathcal{P}t(e_{ik}) = l$

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.

4.4.1 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:

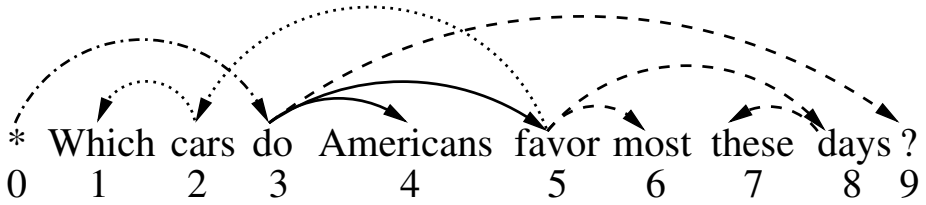
No edges between i and $[i+1, j]$: The same set of edges is also a valid $Int[i+1, j]$ sub-problem. b_i must be true for the $Int[i+1, j]$ sub-problem to ensure $i+1$ receives a parent.

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

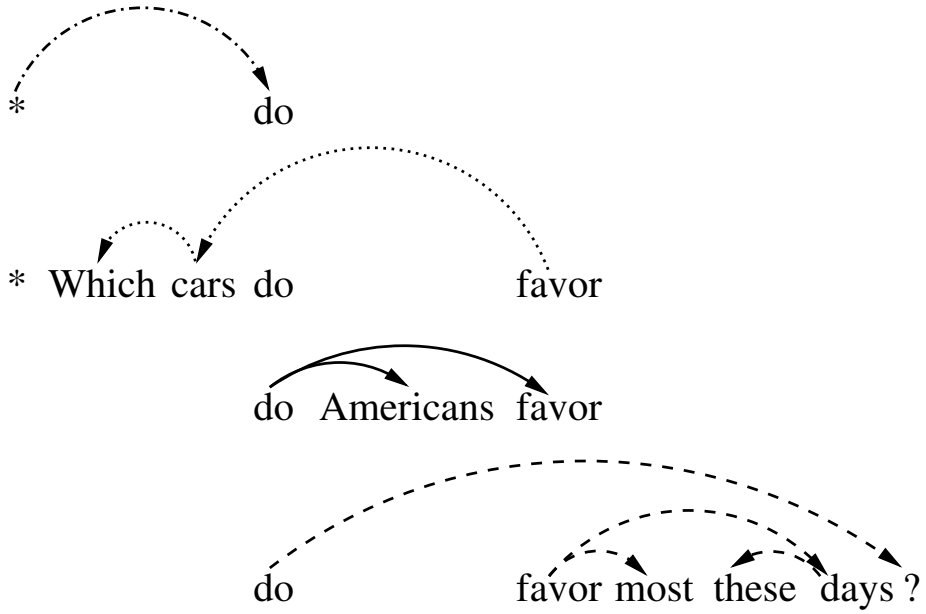
Furthest edge from i is crossed: This is the most interesting case, which uses two split points: the other endpoint of the edge (k), and $l = \mathcal{P}t(e_{ik})$. The dynamic program depends on the order of k and l.

l $\notin$ (i, k) (Figure 4.4a): 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 4.4a) exists, the decomposition is as in Figure 4.4a, Case i; otherwise, it is as in Figure 4.4a, Case ii. Figure 4.5 gives an example of Case i. The crossed edge e_{ik} is the edge from * to *do*, the crossing point l is *favor*, and there exists an edge from k into $(l, j]$ (from *do* into (*favor*, *?*]). 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 4.3b), 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 4.3a). In Case ii, each of the edges in $\mathcal{P}(e_{ik})$ may choose either i or k as



(a)



(b) The sentence in (a) is constructed according to Figure 4.4a, Case i.

Figure 4.5: Constructing a 1-Endpoint-Crossing non-projective English sentence from the WSJ Penn Treebank (Marcus et al., 1993), converted to dependencies with PennConverter (Johansson and Nugues, 2007).

their $\mathcal{P}t$, so the sub-problem $[i,k] \cup \{l\}$ is of type LR (Figure 4.3c). Note that $l = j$ is a special case of Case ii in which the rightmost interval $Int[l,j]$ is empty.

l $\in$ (i, k) (Figure 4.4b): $[i,l] \cup \{k\}$ 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 4.4b), 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 4.4b), 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).

4.4.2 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 4.4a, 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 4.3c 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_jj}. e_{ir_i} cannot cross e_{l_jj}, 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_jj}. Such a chain can be ruled out using a counting argument similar to that in the proof in Section 4.6.1. The decomposition is: $L[i, k, x] + R[k, j, x]$ for some $k \in (i, j)$.

4.4.3 Decomposing an N sub-problem

Consider the maximum scoring forest of type N over $[i, j] \cup \{x\}$ (Figure 4.3d; *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 book-keeping 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 $S[e_{ik}] + Int[k, j] + N[i, k, x]$.

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.

4.4.4 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

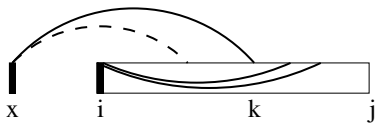
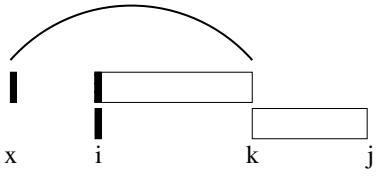
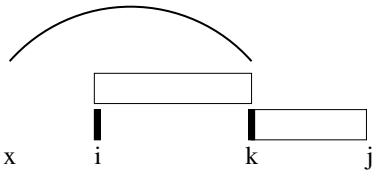


Figure 4.6: An L sub-problem over $[i, j] \cup \{x\}$, k is the neighbor of x furthest from i in the interval.

(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 $S[e_{xk}] + Int[i, k] + L[k, j, i]$:



plus the rest of the forest with that edge removed.

The interesting case is when there are edges between x and the interior (Figure 4.6). 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 $[k, j] \cup \{i\}$.

If there are additional edges between x and the interior (Case i in 4.6), 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 4.6), 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.

4.5 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: $\mathcal{T}F(x, S) :=$

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.

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

$$
Int[i,j,T,F] \leftarrow \text{symmetric to } Int[i,j,F,T]
$$

$$
Int[i,j,T,T] \leftarrow -\infty
$$

$$
\begin{aligned}
&LR[i,j,x,b_x] \leftarrow \max \\
&\left\{\begin{array}{l}
L[i,j,x,F,F,b_x] \\
R[i,j,x,F,F,b_x] \\
\max_{k\in(i,j)} \\
\quad \left\{\begin{array}{ll}
L[i,k,x,F,T,b_x] + R[k,j,x,F,F,F] & \\
L[i,k,x,F,F,F] + R[k,j,x,T,F,b_x] & \\
L\_JFromI[i,k,x] + R[k,j,x,F,F,T] & \text{if } b_x = T \\
L\_JFromX[i,k,x] + R\_XFromJ[k,j,x] & \text{if } b_x = T \\
L[i,k,x,F,F,T] + R\_IFromJ[k,j,x] & \text{if } b_x = T \\
L\_XFromI[i,k,x] + R\_JFromX[k,j,x] & \text{if } b_x = T
\end{array}\right.
\end{array}\right.
\end{aligned}
$$

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

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

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

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

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

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

$N\_XFromJ[i,j,x] \leftarrow$ symmetric to $N\_XFromI[i,j,x]$

$N\_JFromX[i,j,x] \leftarrow$ symmetric to $N\_IFromX[i,j,x]$

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

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

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

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

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

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

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

$R[i, j, x, b_i, b_j, F] \leftarrow$ 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 $L\_JFromI[i, j, x]$

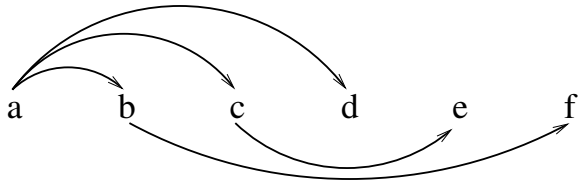


Figure 4.7: 2-planar but not 1-Endpoint-Crossing

4.6 Connections

4.6.1 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, Leighton, and Rosenberg, 1987).

In this section we prove *1-Endpoint-Crossing* $\subseteq$ *2-planar*. These classes are not equal (Figure 4.7). We first prove some properties about the *crossings graphs* (Gómez-Rodríguez and Nivre, 2010) of 1-Endpoint-Crossing trees. The crossings graph of a 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 4.1a and 4.1b are shown in Figures 4.8a and 4.8b, 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*

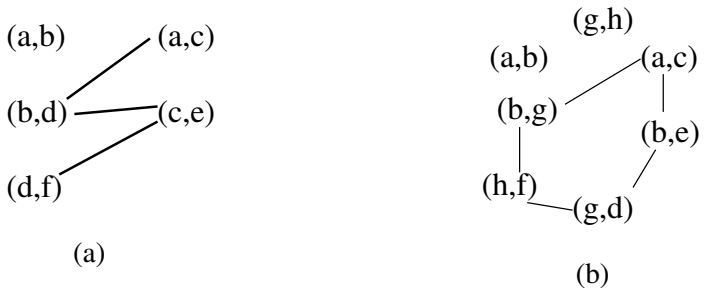


Figure 4.8: The crossing graphs for Figures 4.1a and 4.1b.

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 $4 + (n - 4) = n$. □

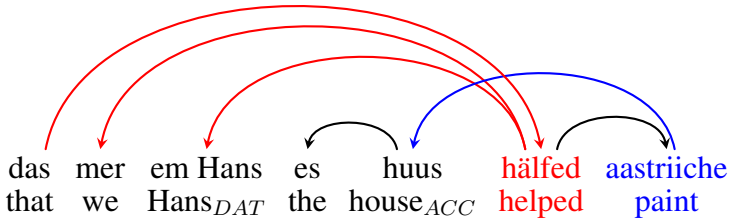
Theorem 4.1. *1-Endpoint-Crossing trees $\subseteq$ 2-planar.*

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. □

4.6.2 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 $(that, helped)$, $(helped, we)$, and $(helped, Hans)$ are each only crossed by an edge with an endpoint at *paint*; the edge $(paint, house)$ 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-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

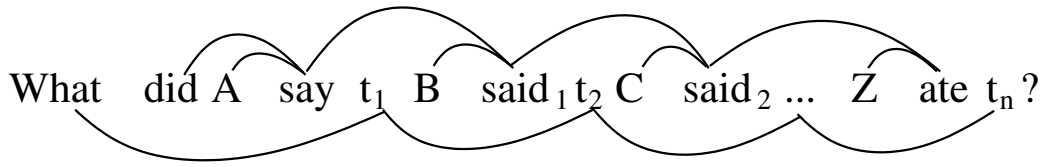


Figure 4.9: An example of *wh-movement* over a potentially unbounded number of clauses. The edges between the heads of each clause cross the edges from trace to trace, but all obey 1-Endpoint-Crossing.

on both grammatical judgments and scores on a comprehension task) (Bach, Brown, and Marslen-Wilson, 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, 1998). Figure 4.9 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, 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 4.9, 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 4.9 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.

4.7 A Simplified Form

The exact form of the dynamic program for projective dependency parsing can vary, with different forms using the same fundamental reasoning but leading to different constant

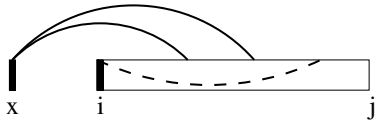
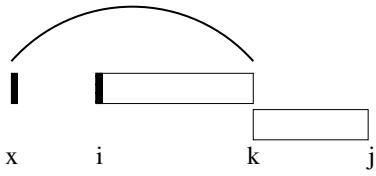
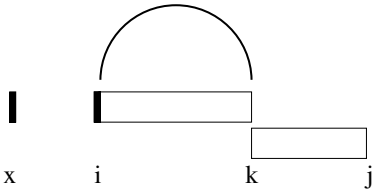


Figure 4.10: An L sub-problem over $[i, j] \cup \{x\}$ with at least two edges between x and (i, j). k is the vertex furthest from i connected to *either* x or i.

(i) **If no edges like the dashed edge exist**: e_{xk} had to choose i as its crossing point, so since no edges from i cross into $(k, j]$, $[k, j]$ is an isolated interval. The interval decomposes into $S[e_{xk}] + L[i, k, x] + Int[k, j]$:



(ii) **If the dashed edge exist**: e_{ik} had to choose x as its crossing point, so since no edges from x cross into $(k, j]$, $[k, j]$ is an isolated interval. The interval decomposes into $S[e_{ik}] + L[i, k, x] + Int[k, j]$:



factors (Eisner, 1996; Eisner and Satta, 1999). In a similar vein, we remark that the dynamic program in Section 4.5 can be simplified by eliminating all of the charts of types N_XFromI, N_IFromX, N_XFromJ, and N_JFromX. These charts are used in two places and are not essential in either: (i) constructing a L or R sub-problem and (ii) constructing an N sub-problem in which x receives its parent ($b_x = T$).

For L or R sub-problems, we remove the construction in Figure 4.6i (that uses variants of N sub-problems) and replace it with the constructions shown in Figures 4.10i and 4.10ii. Construction when there is only one edge incident to the exterior point remains the same (Figure 4.6ii).

For the N sub-problem in which exterior vertex x receives its parent, we note that *any* edge incident to the exterior point is a valid split point since none of these edges can be crossed inside the interval. Therefore, rather than choosing the vertex incident to x that is furthest to one side, we could go ahead and choose x's parent as our split point immediately.

These changes do not affect the asymptotic space or time, but reduce constant factors

somewhat, which may be useful for an implementation. The next chapter takes this modified form as its starting point.

4.8 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 for an edge-factored model. The next chapter examines parsing with 1-Endpoint-Crossing trees with more complex models.

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.

Chapter 5

A Crossing-Sensitive Third-Order Factorization for Dependency Parsing

Features over grandparents and siblings have greatly improved the accuracy of projective parsers, but have so far required large increases in complexity when applied to arborescences or mildly non-projective models. We introduce a "crossing-sensitive" generalization of the third-order factorization of Koo and Collins (2010) that trades off complexity in the *model* structure (i.e. scoring with features over pairs and triples of edges) with complexity in the *output* structure (i.e. producing crossing edges). When applied to a projective tree, the crossing-sensitive factorization *exactly* simplifies to Koo and Collins' Grand-Sibling model. Under this model, the optimal 1-Endpoint-Crossing tree (Chapter 4) can be found in $O(n^4)$ time, matching the asymptotic run-time of *both* the third-order *projective* parser and the *edge-factored* 1-Endpoint-Crossing parser. The crossing-sensitive third-order parser is significantly more accurate than the third-order projective parser under many experimental settings and significantly less accurate on none. Besides the asymptotic guarantees, the cost of running the crossing-sensitive third-order parser is low in practice, running at 0.37-0.47 times the speed of the third-order projective parser.

5.1 Introduction

Incorporating features with wider scope than single edges greatly improves the accuracy of projective parsers (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010). The third-order parser of Koo and Collins (2010) (named as such because it includes features over triples of edges connecting a grandparent, parent, and two siblings) has a runtime of $O(n^4)$, just one factor of n more expensive than the edge-factored model of Eisner (2000).

Incorporating these richer features *and* producing trees with crossing edges has been a challenge, however. For arborescences, including grandparent and/or sibling features is NP-hard (McDonald and Pereira, 2006; McDonald and Satta, 2007); for various definitions of mildly non-projective trees, even edge-factored versions are expensive and their extensions even more so. For example, even an edge-factored model takes $O(n^7)$ for parsing well-nested trees with block degree at most two (Gómez-Rodríguez et al., 2011); while an edge-factored model for 1-Endpoint-Crossing trees has an $O(n^4)$ parsing algorithm (Chapter 4), a straight-forward approach to including grandparent features would raise the runtime to $O(n^7)$ (Section 5.2.2).

The third-order projective parser of Koo and Collins (2010) and the edge-factored 1-Endpoint-Crossing parser described in Chapter 4 have some similarities: both use $O(n^4)$ time and $O(n^3)$ space, using sub-problems over intervals with one exterior vertex, which are constructed using one free split point.

The two parsers differ in *how the exterior vertex is used*: Koo and Collins (2010) use the exterior vertex to store a grandparent index, while Chapter 4 uses the exterior vertex to introduce crossed edges between the point and the interval.

Here we propose *merging* the two to achieve the best of both worlds – producing the best tree in the wider range of 1-Endpoint-Crossing trees while incorporating the identity of the grandparent and/or sibling of the child in the score of an edge whenever the local neighborhood of the edge is projective. The crossing-sensitive grandparent-sibling 1-Endpoint-Crossing parser proposed here takes $O(n^4)$ time, matching the runtime of both

	Projective	1-Endpoint-Crossing
Edge	$O(n^3)$	$O(n^4)$
	Eisner (2000)	Chapter 4
CS-GSib	$O(n^4)$	$\mathbf{O(n^4)}$
	Koo and Collins (2010)	This chapter

Table 5.1: Parsing time for various output spaces and model factorizations. CS-GSib refers to the (crossing-sensitive) grand-sibling factorization described in this chapter.

the third-order projective parser and of the edge-factored 1-Endpoint-Crossing parser (see Table 5.1).

Section 5.2 introduces some notation, reviews the grandparent-sibling factorization and parsing algorithm of Koo and Collins (2010) as well as the edge-factored 1-Endpoint-Crossing algorithm of Chapter 4, and discusses a naïve approach to applying the Grand-Sib factorization directly to 1-Endpoint-Crossing trees. The proposed crossing-sensitive factorization is defined in Section 5.3. The main technical difficulty for the parsing algorithm is showing that grandparent and/or sibling factors are used if and only if no crossings occur in the local neighborhood (Section 5.4). The implemented parser is significantly more accurate than the third-order projective parser in a variety of languages and treebank representations (Section 5.5).

5.2 Preliminaries

For any particular input sentence x, $x = w_1\ w_2\ ...\ w_n$, let $Y_{Proj}(x)$ be the set of projective trees over the set of vertices in the sequence $\{w_0, w_1, w_2, ..., w_n\}$ (where w_0 is the root node) and let Y_{1-EC} be the set of 1-Endpoint-Crossing trees over the same. Note that both $Y_{Proj}(x)$ and $Y_{1-EC}(x)$ are exponentially large in n, the length of the sentence, and that $Y_{Proj}(x) \subset Y_{1-EC}(x)$.

To avoid confusion between open intervals and edges, $\vec{e}_{ij}$ denotes the *directed edge*

from i to j (i.e. i is the parent of j) and (i, j) denotes the *set of vertices* in the open interval between i and j.

In all of the models described in this chapter, the score of a tree decomposes into the sum of scores of its edges. However, the various models differ in both the set of trees they search over ($Y_{Proj}(x)$ or $Y_{1-EC}(x)$) and in the amount of context they use to score an edge. The score for an edge $\vec{e}_{hm}$ *always* depends on at least its endpoints (i.e., $\mathrm{Score}(\mathrm{Edge}(\mathrm{h}, \mathrm{m}))$ over the parent h and the child m), and may *also* depend on whether $\vec{e}_{hm}$ is crossed ($\mathrm{Score}(\mathrm{CrossedEdge}(\mathrm{h}, \mathrm{m}))$) or uncrossed ($\mathrm{Score}(\mathrm{Edge}(\mathrm{h}, \mathrm{m}))$); on g: the parent of h ($\mathrm{Score}(\mathrm{Grand}(\mathrm{g}, \mathrm{h}, \mathrm{m}))$); s: the sibling of m inner to it ($\mathrm{Score}(\mathrm{Sib}(\mathrm{h}, \mathrm{m}, \mathrm{s}))$); or both g and s ($\mathrm{Score}(\mathrm{GrandSib}(\mathrm{g}, \mathrm{h}, \mathrm{m}, \mathrm{s}))$).

5.2.1 Grand-Sibling Projective Parsing

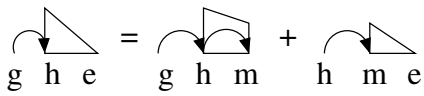
The grand-sibling projective parser of Koo and Collins (2010) produces the highest-scoring tree in $Y_{Proj}(x)$ with each edge $\vec{e}_{hm}$ in the tree scored under the grandparent-sibling model ($\mathrm{Score}(\mathrm{GrandSib}(\mathrm{g}, \mathrm{h}, \mathrm{m}, \mathrm{s}))$). The parser accomplishes this by adding an external grandparent index to each of the sub-problems used in the sibling factorization (McDonald and Pereira, 2006). Figure 6 in Koo and Collins (2010) provided a pictorial view of the algorithm; for convenience, we replicate it here in Figure 5.1.

The definitions of these sub-problems are as follows:

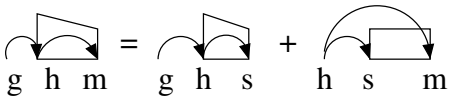
TriG[h,e,g] (Figure 5.1a): Maximum scoring projective tree over vertices $[h, e]$ rooted at h, with g the implied parent of h used to score edges from h to h's children; vertices in $(h, e]$ have no edges to vertices $\notin [h, e]$

TrapG[h,m,g] (Figure 5.1b): Maximum scoring projective tree over vertices $[h, m]$ rooted at h that includes the edge $\vec{e}_{hm}$, with g the implied parent of h used to score edges from h to h's children; vertices in (h, m) have no edges to vertices $\notin [h, m]$

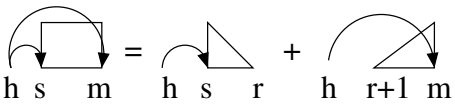
BoxG[s,m,h] (Figure 5.1c): Maximum scoring pair of projective trees over vertices $[s, m]$, with one tree rooted at s and the other at m, with h the implied parent of both m and s; vertices in (s, m) have no edges to vertices $\notin [s, m]$



(a) m is the child of h that e is descended from



(b) The edge $\vec{e}_{hm}$ is added to the tree; s is m's adjacent inner sibling



(c) r is s's outermost descendant; $r + 1$ is m's innermost descendant

Figure 5.1: Algorithm for grand-sibling projective parsing; figures are replications of those in Figure 6 in Koo and Collins (2010).

An edge $\vec{e}_{hm}$ is added to the tree in the "trapezoid" step (Figure 5.1b); this allows the edge to be scored conditioned on m's grandparent (g) and its adjacent inner sibling (s), as all four relevant indices are accessible. The algorithm uses the external grandparent index to score edges from the sub-problem root(s) to its children within the sub-problem. Even though there are two interval endpoints, only one grandparent index is necessary — for the "trapezoid" and "box" sub-problems, the parent of both interval endpoints is determined; for the "triangle", the non-root interval endpoint is forbidden to have any children outside of the interval, so its parent (within the interval) may be safely forgotten.

5.2.2 Edge-factored 1-Endpoint-Crossing Parsing

The edge-factored 1-Endpoint-Crossing parser of Chapter 4 produces the highest scoring tree in Y_{1-EC} with each edge $\vec{e}_{hm}$ scored according to $\mathrm{Score}(\mathrm{Edge}(\mathrm{h}, \mathrm{m}))$. The 1-Endpoint-Crossing property allows the tree to be built up in edge-disjoint pieces each consisting of intervals with one exterior point that has edges into the interval. For example, the tree in Figure 5.2 would be built up with the sub-problems shown in Figure 5.3.

To ensure that crossings *within* a sub-problem are consistent with the crossings that

happen as a result of combination steps, the algorithm uses four different "types" of sub-problems, indicating whether the edges incident to the exterior point may be internally crossed by edges incident to the left interval boundary point (L), right boundary point (R), either (LR), or neither (N). In Figure 5.3, the sub-problem over [*, *do*] $\cup$ {*favor*} would be of type R, and [*favor*, *?*] $\cup$ {*do*} of type L. In the figure, the same set of edges over [*favor*, *?*] $\cup$ {*do*} could also have been a valid type R problem, and indeed that version would be used if those edges were instead combining with a sub-problem that had *?* (the Right endpoint) as its external vertex.

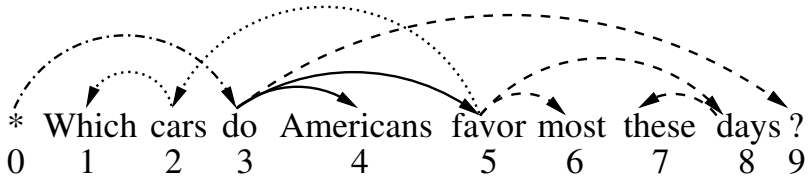


Figure 5.2: (Repeats Figure 4.5a) A 1-Endpoint-Crossing non-projective English sentence from the WSJ Penn Treebank (Marcus et al., 1993), converted to dependencies with PennConverter (Johansson and Nugues, 2007).

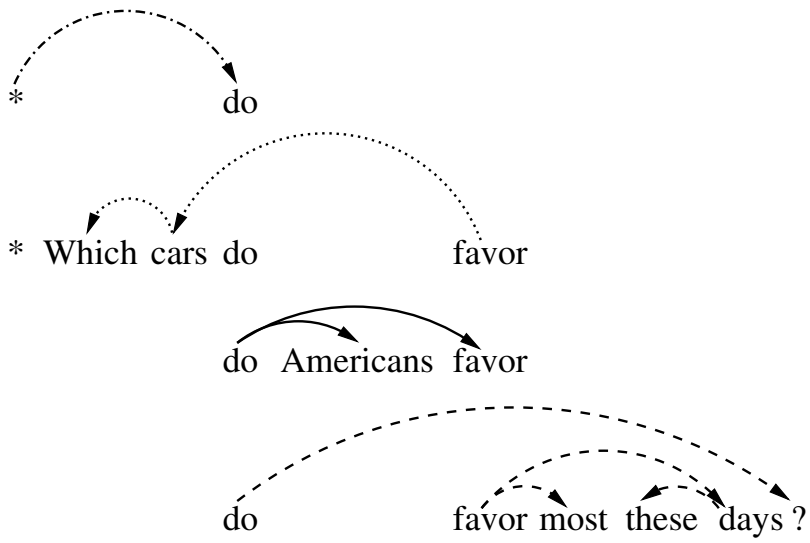


Figure 5.3: (Repeats Figure 4.5b) The sentence in Figure 5.2 is constructed using intervals with one exterior vertex to include the crossed edges (Chapter 4).

Naïve Approach to Including Grandparent Features

The example in Figure 5.2 illustrates the difficulty of incorporating grandparents into 1-Endpoint-Crossing parsing. The vertex *favor* has a parent or child in *all three* of the sub-problems (taking on the role of the exterior point, the right interval boundary point, and the left interval boundary point in turn). *favor* gets its parent (*do*) from *exactly one* of these sub-problems (in this case, the middle one). While in this example the parent happened to also be a boundary point, the parent could have been anywhere in the interval. In order to use grandparent scoring for the edges from *favor* to *favor*'s children in the other two sub-problems, we would need to augment the problems with the grandparent index *do*. We also must add the parent index *do* to the middle sub-problem to ensure consistency (i.e., that *do* is in fact the parent assigned). Thus, a first attempt to include grandparent features into 1-Endpoint-Crossing tree raises the runtime from $O(n^4)$ to $O(n^7)$ (all of the four indices need a "predicted parent" additional index; at least one edge is always implied so one of the "predicted parent" indices can be dropped).

5.3 Crossing-Sensitive Factorization

Factorizations for projective dependency parsing have often been designed to allow efficient parsing. For example, the algorithms in Eisner (2000) and McDonald and Pereira (2006) achieve their efficiency by assuming that children to the left of the parent and to the right of the parent are independent of each other. The algorithms of Carreras (2007) and Model 2 in Koo and Collins (2010) include grandparents for only the outermost grandchildren of each parent for efficiency reasons.

In a similar spirit, we avoid the blow-up in parent indices described in Section 5.2.2 by introducing a variant of the Grand-Sib factorization that scores crossed edges independently (as a $\mathrm{CrossedEdge}$ part) and uncrossed edges under either a grandparent-sibling, grandparent, sibling, or edge-factored model depending on whether relevant edges in its local neighborhood are crossed (see Table 5.2). Whether the part includes the sibling depends

Local Neighborhood Crossings	$Crossed(\vec{e}_{hs})$	$\neg Crossed(\vec{e}_{hs})$
$\neg GProj(\vec{e}_{hm})$	$\mathrm{Edge}(h, m)$	$\mathrm{Sib}(h, m, s)$
$GProj(\vec{e}_{hm})$	$\mathrm{Grand}(g, h, m)$	$\mathrm{GrandSib}(g, h, m, s)$

Table 5.2: Part type for an uncrossed edge $\vec{e}_{hm}$ for the crossing-sensitive third-order factorization (g is m's grandparent; s is m's inner sibling).

on whether the edge $\vec{e}_{hs}$ from the parent to the sibling is crossed. $GProj(\vec{e}_{hm})$ (Definition 13) determines whether $\vec{e}_{hm}$'s local neighborhood is sufficiently projective to include the grandparent in the part.

Our parser will find the optimal 1-Endpoint-Crossing tree under this new factorization, solving the optimization problem below:

$$\underset{y \in Y_{1-EC}}{\arg\max} \sum_{\substack{g,h,m,s \mid \\ \vec{e}_{gh} \in y, \\ \vec{e}_{hm} \in y, \\ Sib(m,s) \in y}} Score(Part(g, h, m, s)) \tag{5.1}$$

$$Part(g, h, m, s) =$$

$$\begin{cases} \mathrm{GrandSib}(g, h, m, s) & : \neg Crossed(\vec{e}_{hm}) \wedge GProj(\vec{e}_{hm}) \wedge \neg Crossed(\vec{e}_{hs}) \\ \mathrm{Grand}(g, h, m) & : \neg Crossed(\vec{e}_{hm}) \wedge GProj(\vec{e}_{hm}) \wedge Crossed(\vec{e}_{hs}) \\ \mathrm{Sib}(h, m, s) & : \neg Crossed(\vec{e}_{hm}) \wedge \neg GProj(\vec{e}_{hm}) \wedge \neg Crossed(\vec{e}_{hs}) \\ \mathrm{Edge}(h, m) & : \neg Crossed(\vec{e}_{hm}) \wedge \neg GProj(\vec{e}_{hm}) \wedge Crossed(\vec{e}_{hs}) \\ \mathrm{CrossedEdge}(h, m) & : Crossed(\vec{e}_{hm})) \end{cases}$$

A fully projective tree would decompose into *exclusively* $\mathrm{GrandSib}$ parts (as all edges would be uncrossed and $GProj$). As all projective trees are within the 1-Endpoint-Crossing search space, the optimization problem above includes all projective trees scored with grand-sibling features everywhere. Projective parsing with grand-sibling scores can be seen as a special case, as the crossing-sensitive 1-Endpoint-Crossing parser can simulate a grand-sibling projective parser by setting all $\mathrm{Crossed}(h, m)$ scores to $-\infty$.

This factorization allows the parser to learn different weight functions for crossed and uncrossed edges. Adding variables that indicate non-projectivity of edges into an integer

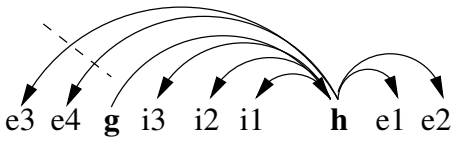


Figure 5.4: The exterior children are numbered first beginning on the side closest to the parent, then the side closest to the grandparent. There must be a path from the root to g, so the edges from h to its exterior children on the far side of g are guaranteed to be crossed.

linear programming formulation of dependency parsing has previously been found to improve a parser's accuracy (Martins et al., 2009).

To define $GProj$, we first need a few auxiliary definitions:

Definition 11. *Consider any edge $\vec{e}_{gh}$ in a given tree y. We can partition h's children into two disjoint sets:* $\mathbf{Interior_g(h)}$*, and* $\mathbf{Exterior_g(h)}$*. $Interior_g(h)$ consists of those children of h that lie between g and h (in the linear order of words in the sentence). $Exterior_g(h)$ consists of the complementary set of children.*

For each parent h, grandparent g, and subset $Interior_g(h)$ and $Exterior_g(h)$, we enumerate the children in each subset in the following order: for $Interior_g(h)$ the vertices are numbered from closest to h through furthest from h; for $Exterior_g(h)$, we first number the vertices on the side closest to h from closest to h through furthest, then wrap around to include the vertices on the side closest to g. Figure 5.4 shows a parent h, its grandparent g, and a possible sequence of three interior and four exterior children.

Note that for a projective tree, there would not be any children on the far side of g.

Definition 12. $\mathbf{Outer}(m)$ *is the set of siblings to m that are in the same subset of children and are later in the enumeration than m is.*

For example, in the tree in Figure 5.2, $Outer(\textit{most}) = \{\textit{days}, \textit{cars}\}$.

Definition 13. *An uncrossed edge $\vec{e}_{hm}$ is $GProj$ if both of the following hold:*

1. *The edge $\vec{e}_{gh}$ from the parent of h to h is not crossed*
2. *None of the edges from h to $Outer(m)$ (m's outer siblings) are crossed*

CrossedEdge(*,*do*)	Sib(*cars*, *Which*, -)
CrossedEdge(*favor*,*cars*)	Sib(*do*, *Americans*, -)
Sib(*do*, *favor*, *Americans*)	CrossedEdge(*do*,*?*)
Sib(*favor*, *most*, -)	Sib(*favor*, *days*, *most*)
GSib(*favor*, *days*, *these*, -)	

Table 5.3: Decomposing Figure 5.2 according to the crossing-sensitive third-order factorization described in Section 5.3. Null inner siblings are indicated with -.

In Figure 5.2, the edge from *do* to *Americans* is not $GProj$ because Condition (1) is violated, while the edge from *favor* to *most* is not $GProj$ because Condition (2) is violated. Table 5.3 lists the parts in the tree in Figure 5.2 according to this crossing-sensitive third-order factorization.

This definition eliminates the problematic grandparent cases discussed in Section 5.2.2, assuming that (1) for any sub-problem with an exterior point, all edges incident to the exterior point are crossed and (2) there exists at least one such edge (Section 5.4.1 will discuss how these assumptions are enforced).

Consider again the problematic case of the vertex *favor* in Figure 5.2, with children in both $[*, do] \cup \{favor\}$ and $[favor, ?] \cup \{do\}$. Under a standard grandparent-sibling factorization, all three sub-problems would have needed an additional index noting the parent of *favor*. Under the crossing-sensitive grandparent-sibling factorization, the sub-problem $[*, do] \cup \{favor\}$ now no longer needs this grandparent index, as all edges from *favor* to the interval are guaranteed to be crossed and thus scored independently.

Since the parent of *favor* is found in $[do, favor]$, all children of *favor* in $(favor, ?]$ are *Exterior* children. There must exist at least one child of *favor* in $[*, do] \cup \{favor\}$ (Assumption 2) and the edge to such a child must be crossed (Assumption 1). This child is on the opposite side of *favor*'s parent, and so would be an *Outer* sibling to *all* of the children of *favor* in $(favor, ?]$ (by "wrapping around"). Therefore all of the edges from *favor* to children in $(favor, ?]$ would violate Condition (2) and be $\neg GProj$, avoiding the need for a grandparent index.

Finally, since no children outside $[do, favor]$ will need to know *favor*'s parent, the middle sub-problem does not need to add a grandparent index for consistency reasons.

This section described the crossing-sensitive third-order factorization. The next section describes the parsing algorithm that finds the optimal 1-Endpoint-Crossing tree according to this factorization.

5.4 Parsing Overview

The GrandSib-Crossing parser finds the maximum scoring tree according to Expression 5.1 with a dynamic programming procedure reminiscent of Koo and Collins (2010) (for scoring projective portions of the tree with grandparent and/or sibling features) and of Chapter 4 (for including crossed edges). The parser also uses additional novel sub-problems for transitioning between the projective and non-projective portions of a tree.

Optimizing Expression 5.1 presents two technical difficulties:

1. The parser must know whether an edge is crossed when it is added
2. When adding an *uncrossed* edge, the parser must use the appropriate part for scoring according to whether *other* edges are crossed (Table 5.2).

Difficulty 1 is solved by adding crossed and uncrossed edges to the tree in distinct sub-problems (Section 5.4.1). Difficulty 2 is solved by producing different versions of subtrees over the same sets of vertices, both with and without a grandparent index, which differ in their assumptions about the tree *outside* of that set (Section 5.4.2). Section 5.4.3 contains the main statements of results. The list of all sub-problems and their corresponding invariants is given in Appendix B and the full dynamic program is in Appendix C.

5.4.1 Enforcing Crossing Edges

The parser adds crossed and uncrossed edges in distinct portions of the dynamic program. Uncrossed edges are added *only* through trapezoid sub-problems (that may or may not

have an additional grandparent index), while crossed edges are added in *non*-trapezoid sub-problems.

To add *all* uncrossed edges in trapezoid sub-problems, we (a) enforce that any edge added anywhere else must be crossed, and (b) introduce transitional sub-problems in order to build trapezoids when the edge $\vec{e}_{hm}$ is not crossed, but the edge to its inner sibling $\vec{e}_{hs}$ is (and so the construction step shown in Figure 5.1b cannot be used).

Crossing Conditions

The parsing algorithm in Chapter 4 included crossing edges by using "crossing region" sub-problems over intervals with an external vertex that *optionally* contained edges between the interval and the external vertex. An uncrossed edge could then be derived in multiple ways — either from a derivation that prohibited it from being crossed or from a derivation which allowed (but did not force) it to be crossed. The Crossing Conditions (Definition 15) remove this ambiguity and force the exterior point in a crossing region to be used *only* for crossed edges. Edges between the exterior point and the *interior* of the interval are now always crossed; edges to the interval boundary points are either guaranteed to be crossed or prohibited.

Each crossing region of Chapter 4 has a signature of $\mathrm{type}[i, j, x, b_i, b_j, b_x]$. i, j, and x are in $[0, n]$ and indicate that the sub-problem is over vertices $[i, j] \cup \{x\}$; $\mathrm{type} \in \{L, R, LR, N\}$ and indicates whether i and/or j can be the crossing point for edges between x and (i, j); b_i, b_j, and b_x are booleans indicating whether i, j, or x, respectively, receives its incoming edge from its parent within this sub-problem. This signature is used to divide the interval vertices $[i, j]$ into those that can be directly connected by an edge to x in this sub-problem (and such an edge is guaranteed to be crossed) and those that cannot be directly connected to x.

Definition 14. *$VCross(i, j, b_i, b_j, type)$ is the interval of vertices that are neither roots nor crossing points of the sub-problem. All internal vertices in (i, j) are never roots nor crossing points, yielding four possible cases that vary in whether the boundary vertices i*

and/or j are included. In particular:

$$VCross(i, j, b_i, b_j, type) =$$
$$\begin{cases} (i,j) & : type = LR \vee \left(\begin{array}{l} (type = L \vee b_i = F) \\ \wedge (type = R \vee b_j = F)) \end{array} \right) \\ [i,j) & : b_i = T \wedge (type = R \vee (type = N \wedge b_j = F)) \\ (i,j] & : b_j = T \wedge (type = L \vee (type = N \wedge b_i = F)) \\ [i,j] & : type = N \wedge b_i = T \wedge b_j = T \end{cases}$$

The Crossing Conditions (Definition 15) are enforced by construction, and taken together ensure that whenever a crossing region is used, edges between the exterior point and the interval are crossed. For example, by requiring at least one edge between *do* and (*favor*, *?*] and also between *favor* and $(*, do)$, the edges in the two sets are guaranteed to cross each other.

Definition 15. *The* Crossing Conditions *of a crossing region $type[i, j, x, b_i, b_j, b_x]$ are:*

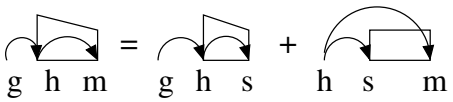
- *All edges in the sub-problem incident to x have the other endpoint of the edge in $VCross(i, j, b_i, b_j, type)$*
- *There exists at least one edge between x and $VCross(i, j, b_i, b_j, type)$*
- *All edges between x and vertices in $VCross(i, j, b_i, b_j, type)$ will be crossed by an edge outside of the sub-problem.*

Consider again the decomposition in Figure 5.3. Vertices *do* and *favor* co-occur in each of the three sub-problems, but now the *uncrossed* edge from *do* to *favor* is prohibited from being added in $[*, do] \cup \{favor\}$ or $[favor, ?] \cup \{do\}$. Consider the L sub-problem over $[favor, ?] \cup \{do\}$ (dashed edges). The Crossing Conditions for this sub-problem are satisfied: *do* only has edges to vertices in (*favor*, *?*] (the only edge incident to *do* is to *?* and *?* $\in$ (*favor*, *?*]), there exists at least one edge between *do* and (*favor*, *?*], and the edge from *do* to *?* is crossed by an edge outside the sub-problem (the edge from *favor* to *cars*). At

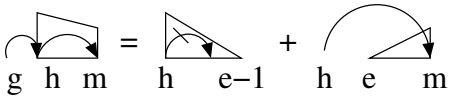
least one edge like the one from *favor* to *cars* that crosses the edge from *do* to *?* must exist, because the same set of Crossing Conditions also apply to the dotted edge sub-problem of type R over $[*, do] \cup \{favor\}$, requiring an edge between *favor* and $(*, do)$.

Trapezoids with Edge to Inner Sibling Crossed

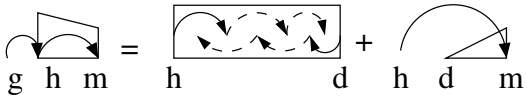
To add *all* uncrossed edges in trapezoid-style sub-problems, we must be able to construct a trapezoid over vertices $[h, m]$ whenever the edge $\vec{e}_{hm}$ is not crossed. The construction used in Koo and Collins (2010), repeated graphically in Figure 5.5a, *cannot* be used if the edge $\vec{e}_{hs}$ is crossed, as there would then exist edges between (h, s) and (s, m), making s an invalid split point. We therefore add some "transitional glue" to allow alternative ways to construct the trapezoid over $[h, m]$ when $\vec{e}_{hm}$ is not crossed but the edge $\vec{e}_{hs}$ to m's inner sibling is.



(a) Case 1: Edge from h to inner sibling s is *not* crossed (repeats Figure 5.1b)



(b) Case 2: $\vec{e}_{hs}$ is crossed, but the chain of crossing edges involving $\vec{e}_{hs}$ does not include any descendants of m. e is m's descendant furthest from m within (h, m). $s \in (h, e-1)$.



(c) Case 3: $\vec{e}_{hs}$ is crossed, and the chain of crossing edges involving $\vec{e}_{hs}$ includes descendants of m. Of m's descendants that are incident to edges in the chain, d is the one closest to m (d can be m itself). $s \in (h, d)$.

Figure 5.5: Ways to build a trapezoid when the edge $\vec{e}_{hs}$ to m's inner sibling may be crossed

The two additional ways of building trapezoids are shown graphically in Figures 5.5b and 5.5c. Consider the "chain of crossing edges" that includes the edge $\vec{e}_{hs}$. If none of

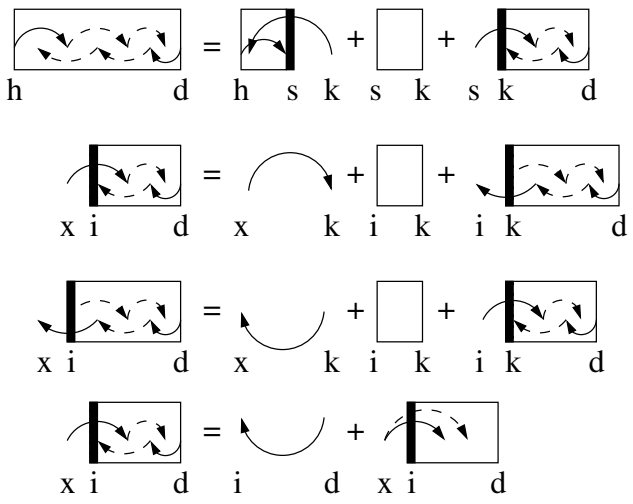


Figure 5.6: Constructing a chain of crossing edges

these edges are in the subtree rooted at m, then we can build the tree involving m and its inner descendants separately (Figure 5.5b) from the rest of the tree rooted at h. $[h, e-1]$ is now an interval with the furthest edge incident to h ($\vec{e}_{hs}$) crossed: these intervals are parsed choosing s and the crossing point of $\vec{e}_{hs}$ simultaneously (see Figure 4.4 in Chapter 4 for details).

Otherwise, the sub-tree rooted at m is involved in the chain of crossing edges (Figure 5.5c). The chain of crossing edges between h and d (m's descendant, which may be m itself) is built up first (Figure 5.6, discussed below), then concatenated with the triangle rooted at m containing m's inner descendants not involved in the chain (Figure 5.5c).

Chains of crossing edges are constructed using a combination of existing sub-problems from Chapter 4 and some additional specialized versions.

If an edge incident to d crosses $\vec{e}_{hs}$ directly, then the chain can be constructed with a few crossing region sub-problems, with the exact form varying depending on the choices of crossing points for the two edges in question.

Otherwise there are at least two internal edges connecting $\vec{e}_{hs}$ and an edge incident to d in this chain. The chain construction is shown pictorially in Figure 5.6. The chain is constructed from one side to the other, repeatedly applying two specialized types of L items that *require* an edge incident to the left endpoint to cross the edge from the exterior point,

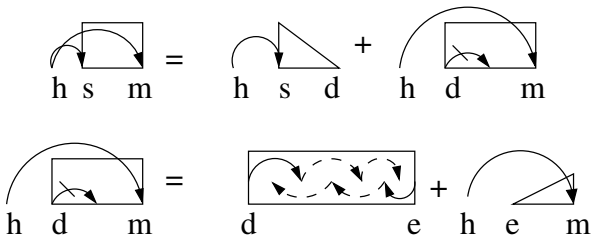


Figure 5.7: Constructing a box when edges in m and s's subtrees cross each other

and require this chain to extend all the way to the right endpoint. The chain alternates between adding an edge from the interval to the exterior point (right-to-left) or from the exterior point to the interval (left-to-right). The boundary edges of the chain can be crossed more times without violating the 1-Endpoint-Crossing property, and so the beginning and end of the chain can be unrestricted crossing regions.

These specialized chain sub-problems are also used to construct *boxes* (Figure 5.1c) over $[s, m]$ with shared parent h when neither edge $\vec{e}_{hs}$ nor $\vec{e}_{hm}$ is crossed, but the subtrees rooted at m and at s cross each other. The alternate way to construct a box is shown in Figure 5.7.

Lemma 4. *The GrandSib-Crossing parser adds all uncrossed edges and only uncrossed edges in a tree in a "trapezoid" sub-problem.*

The *only* part is easy: when a trapezoid is built over an interval $[h, m]$, all edges are internal to the interval, so no earlier edges could cross $\vec{e}_{hm}$. After the trapezoid is built, only the interval endpoints h and m are accessible for the rest of the dynamic program, and so an edge between a vertex in (h, m) and a vertex $\notin [h, m]$ can never be added. The Crossing Conditions ensure that every edge added in a non-trapezoid sub-problem is crossed.

Lemma 5. *The GrandSib-Crossing parser considers* all *1-Endpoint-Crossing trees and* only *1-Endpoint-Crossing trees.*

All trees that could have been built in Pitler et al. (2013) are still possible. It can be verified that the additional sub-problems added all obey the 1-Endpoint-Crossing property.

5.4.2 Reduced Context in Presence of Crossings

A crossed edge (added in a non-trapezoid sub-problem) is scored as a $\mathrm{CrossedEdge}$ part. An uncrossed edge added in a trapezoid sub-problem, however, may need to be scored according to a $\mathrm{GrandSib}$, Grand, Sib, or Edge part, depending on whether the relevant *other* edges are crossed. In this section we show that sibling and grandparent features are included in the GrandSib-Crossing parser as specified by Table 5.2.

Sibling Features

Lemma 6. *The GrandSib-Crossing parser scores an uncrossed edge $\vec{e}_{hm}$ with a* Sib *or* $\mathrm{GrandSib}$ *part if and only if $\vec{e}_{hs}$ is not crossed.*

Proof. Whether the edge to an uncrossed edge's inner sibling is crossed is known bottom-up through how the trapezoid is constructed, since the inner sibling is *internal* to the sub-problem. When $\vec{e}_{hs}$ is not crossed, the trapezoid is constructed as in Figure 5.1b, using the inner sibling as the free index split point. When the edge is not $\vec{e}_{hs}$ is crossed, the trapezoid is constructed as in Figure 5.5b or 5.5c; note that both ways force the edge to the inner sibling to be crossed. □

Grandparent Features for $GProj$ Edges

Koo and Collins (2010) include an external grandparent index for each of the sub-problems that the edges within use for scoring. We want to avoid adding such an external grandparent index to *any* of the crossing region sub-problems (to stay within the desired time and space constraints) or to interval sub-problems when the external context would make all internal edges $\neg GProj$. For each interval sub-problem, the parser constructs versions both with and without a grandparent index (Figure 5.8). Which version is used depends on the external context. In a *bad context*, all edges to children within an interval are guaranteed to be $\neg GProj$. We show that all boundary points in crossing regions are placed in bad contexts, and then that edges are scored with grandparent features if and only if they are $GProj$.

(a) For good contexts (b) For bad contexts

Figure 5.8: For each of the interval sub-problems in Koo and Collins (2010), the parser constructs versions with and without the additional grandparent index. Figure 5.8b is used if the edge from *do* to *favor* is crossed, or if there are any crossed edges from *favor* to children to the left of *do* or to the right of *days*. Otherwise, Figure 5.8a is used.

Bad Contexts for Interval Boundary Points For *exterior vertex* boundary points, all edges from it to its children will be crossed (Definition 15), so it does not need a grandparent index.

Lemma 7. *If a boundary point i's parent (call it g) is within a sub-problem over vertices $[i, j]$ or $[i, j] \cup \{x\}$, then for all uncrossed edges $\vec{e}_{im}$ with m in the sub-problem, the tree outside of the sub-problem is irrelevant to whether $\vec{e}_{im}$ is $GProj$.*

Proof. The sub-problem contains the edge $\vec{e}_{gi}$, so Condition (1) is checked internally. m cannot be x, since $\vec{e}_{im}$ is uncrossed. If g is x, then $\vec{e}_{im}$ is $\neg GProj$ regardless of the outer context. If both g and $m \in (i, j]$), then $Outer(m) \subseteq (i, j]$: If m is an interior child of i ($m \in (i, g)$) then $Outer(m) \subseteq (m, g) \subseteq (i, j]$. Otherwise, m is an exterior child ($m \in (g, j]$) and so by the "wrapping around" definition of $Outer$, $Outer(m) \subseteq (g, m) \subseteq (i, j]$. Thus Condition (2) is also checked exclusively within the sub-problem. □

We can therefore focus on interval boundary points that receive their parent outside of the sub-problem.

Definition 16. *The left boundary vertex of an interval $[i, j]$ is in a* bad context *($BadContext(i, j, L)$) if i receives its parent (call it g) from outside of the sub-problem and either of the following hold:*

1. **Grand-Edge Crossed***: $\vec{e}_{gi}$ is crossed*

2. **Outer-Child-Edge Crossed**: *An edge from i to a child of i outside of $[i, j]$ and Outer to j will be crossed (recall this includes children on the far side of g if g is to the left of i)*

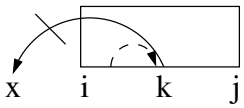
$BadContext(i, j, R)$ is defined symmetrically regarding j and j's parent and children.

Corollary 1. *If $BadContext(i, j, L)$, then for all $\vec{e}_{im}$ with $m \in (i, j]$, $\vec{e}_{im}$ is $\neg GProj$. Similarly, if $BadContext(i, j, R)$, for all $\vec{e}_{jm}$ with $m \in [i, j)$, $\vec{e}_{jm}$ is $\neg GProj$.*

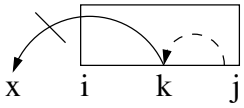
No Grandparent Indices for Crossing Regions We would exceed our desired $O(n^4)$ run-time if *any* crossing region sub-problems needed *any* grandparent indices. For all crossing region sub-problems over $[i, j] \cup \{x\}$, the dynamic program ensures $BadContext(i, j, L)$ if i gets its parent outside and $BadContext(i, j, R)$ if j gets its parent outside. Recall the example in Figure 5.3. For the crossing region over $[*, \textit{do}] \cup \{\textit{favor}\}$, the left boundary point is in a bad context because an outer child edge is crossed, while the right boundary point is in a bad context because the grand-edge is crossed. Similarly for the crossing region over $[\textit{favor}, ?] \cup \{do\}$, the left boundary point is in a bad context because an outer child edge is crossed, while the right boundary point gets its parent edge from within the sub-problem. Similar reasoning holds for all other ways crossing regions can be combined to construct an interval with its furthest edge crossed.

Crossing region sub-problems can also be combined to form larger crossing regions. Split points for the $L/R/N$ sub-problems by construction are incident to a crossed edge to a further vertex. If the crossed edge is the edge from the split point's parent to the split point, then the grand-edge is crossed and so both sides are in a bad context. If the crossed edge is from the split point to a child, then that child is Outer to all other children on the side in which it does not get its parent (see Figure 5.9 for examples).

In Chapter 4, if an LR sub-problem had edges from the exterior point crossed by both the left and the right boundary points, it was constructed by concatenating an L and an R sub-problem. Since the split point wasn't necessarily incident to a crossed edge, the split point might have $GProj$ edges to children on the side other than where it gets its



(a) Because k's parent is in $[i, k)$, x is Outer to all children of k in $(k, j]$



(b) Because k's parent is in $(k, j]$, x is Outer to all children of k in $[i, k)$

Figure 5.9: The edge $\vec{e}_{kx}$ is guaranteed to be crossed, so k is in a $\mathrm{BadContext}$ for whichever side it does not get its parent from.

parent; accommodating this would add another factor of n to the running time and space requirements to store the split point's parent. To avoid this increase in running time, they are instead built up as in Figure 5.10, which chooses the split point so that the edge from the parent of the split point to it is crossed.

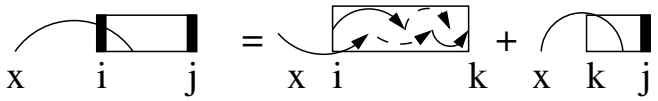


Figure 5.10: For all split points k, the edge from k's parent to k is crossed, so all edges from k to children on either side were $\neg GProj$. The case when the split point's parent is from the right is symmetric.

Lemma 8. *For all crossing region sub-problems $[i, j] \cup \{x\}$ with i's parent $\notin [i, j] \cup \{x\}$, $BadContext(i, j, L)$. Similarly, when j's parent $\notin [i, j] \cup \{x\}$, $BadContext(i, j, R)$.*

Corollary 2. *No grandparent indices are needed for any crossing region sub-problem.*

Triangles and Trapezoids with and without Grandparent Indices The presentation that follows assumes left-headed versions. Uncrossed edges are added in two distinct types of trapezoids: (1) $\mathrm{TrapG[h, m, g, L]}$ with an external grandparent index g, scores the edge $\vec{e}_{hm}$ with grandparent features, and (2) $\mathrm{Trap}[h, m, L]$ *without* a grandparent index, scores the edge $\vec{e}_{hm}$ without grandparent features. Triangles also have versions with

($\mathrm{TriG[h, e, g, L]}$ and without ($\mathrm{Tri[h, e, L]}$) a grandparent index. We show below that all $GProj$ edges are added in TrapG sub-problems, and all $\neg GProj$ uncrossed edges are added in Trap sub-problems.

Lemma 9. *For all $k \in (i, j)$, if $BadContext(i, j, L)$, then $BadContext(i, k, L)$. Similarly, if $BadContext(i, j, R)$, then $BadContext(k, j, R)$.*

Proof. $BadContext(i, j, L)$ implies either the edge from i's parent to i is crossed and/or an edge from i to a child of i outer to j is crossed. If the edge from i's parent to i is crossed, that also implies $BadContext(i, k, L)$. If a child of i is outer to j, then since $k \in (i, j)$, such a child is also outer to k. □

Lemma 10. *All left-rooted triangle sub-problems* $\mathrm{Tri[i, j, L]}$ *without a grandparent index are in a $BadContext(i, j, L)$. Similarly for all* $\mathrm{Tri[i, j, R]}$*, $BadContext(i, j, R)$.*

Proof. All triangles without grandparent indices are either placed immediately into a bad context (either by adding a crossed edge to the triangle's root from its parent, or by adding a crossed edge from the root to an outer child) or are combined with other sub-trees to form larger crossing regions (and therefore the triangle is in a bad context, using Lemmas 8 and 9). □

Lemma 11. *All triangle sub-problems with a grandparent index* $\mathrm{TriG[h, e, g, L]}$ *are placed in a $\neg BadContext(h, e, L)$. Similarly,* $\mathrm{TriG[e, h, g, R]}$ *are only placed in $\neg BadContext(h, e, R)$.*

Proof. We will consider where a triangle with a grandparent index $\mathrm{TriG[h, e, g, L]}$ can be placed in the full dynamic program and what each step would imply about the rest of the tree.

It can combine with a trapezoid to form another larger triangle (as in Figure 5.1a), which forces both $\vec{e}_{gh}$ to not be crossed and h is never accessible again and so can have no outer crossed children.

It can combine with another adjacent triangle to form a box with a grandparent index, which then combines with another trapezoid to form a larger trapezoid (Figure 5.1c followed by 5.1b). Again $\vec{e}_{gh}$ is not crossed and h is now internal to the larger trapezoid and so has no exterior children.

It can combine with a triangle (Figure 5.5b) or a chain of crossing edges (Figure 5.5c) to form a trapezoid from g to h, so $\vec{e}_{gh}$ is not crossed and the other sub-problem had no children of m.

Finally, it can be used to build a box with a grandparent index (Figure 5.7), combining with a chain of crossing edges which is then combined with another triangle with a grandparent index to form a box with a grandparent index, which can then only form a larger trapezoid. This last step enforces that $\vec{e}_{gh}$ cannot be crossed, and m is either an inner child of h (and h has no more children in g's direction by combining with the adjacent sub-problems), or m is an outer child of h (in which case h is no longer accessible after being absorbed into the interior of the trapezoid to its adjacent outer sibling). □

Lemma 12. *In a* $\mathrm{TriG[h, e, g, L]}$ *sub-problem, if an edge* $\vec{e}_{hm}$ *is not crossed and no edges from* i *to siblings of* m *in* $(m, e]$ *are crossed, then* $\vec{e}_{hm}$ *is* $GProj$.

Proof. This follows from (1) the edge $\vec{e}_{hm}$ is not crossed, (2) the edge $\vec{e}_{gh}$ is not crossed by Lemma 11, and (3) no outer siblings are crossed (outer siblings in $(m, e]$ are not crossed by assumption and siblings outer to e are not crossed by Lemma 11). □

Lemma 13. *An uncrossed edge* $\vec{e}_{hm}$ *scored with a* $\mathrm{GrandSib}$ *or* Grand *part (added through a* $\mathrm{TrapG}[h, m, g, L]$ *or* $\mathrm{TrapG}[m, h, g, R]$ *sub-problem) is* $GProj$.

Proof. A TrapG can be placed in only one of two places: (1) combining with descendants of m to form a triangle with a grandparent index rooted at h (indicating that m is the outermost inner child of h) or (2) combining with descendants of m and of m's adjacent outer sibling (call it o), forming a trapezoid from h to o (indicating that $\vec{e}_{ho}$ is not crossed). Such a trapezoid could again only be combined with further uncrossed outer siblings until the final triangle rooted at h with grandparent index g is built, indicating that h has collected

all its children to that side. As $\vec{e}_{hm}$ was not crossed, no edges from h to outer siblings within the triangle are crossed, and $\vec{e}_{hm}$ is within a TriG sub-problem, $\vec{e}_{hm}$ is $GProj$ by Lemma 12. □

Lemma 14. *An uncrossed edge $\vec{e}_{hm}$ scored with a* Sib *or* Edge *part (added through a* Trap[h, m, L] *or* Trap[m, h, L] *sub-problem) is $\neg GProj$.*

Proof. A Trap can only (1) form a triangle without a grandparent index, or (2) form a trapezoid to an outer sibling of m, until eventually a final triangle rooted at h without a grandparent index is built. This triangle without a grandparent index is then placed in a bad context (Lemma 10) and so $\vec{e}_{hm}$ is $\neg GProj$ (Corollary 1). □

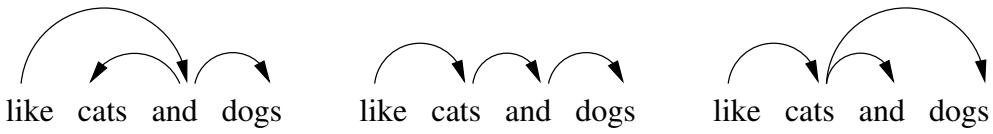
5.4.3 Summary

Lemma 15. *The algorithm for the GrandSib-Crossing parser runs in $O(n^4)$ time and $O(n^3)$ space.*

Proof. All sub-problems are either over intervals (two indices), intervals with a grandparent index (three indices), or crossing regions (three indices). No crossing regions require any grandparent indices (Corollary 2). The only sub-problems that require a maximization over two internal split points are over intervals and need no grandparent indices (as the furthest edges from each root are guaranteed to be crossed within the sub-problem). □

Theorem 4.2. *The GrandSib-Crossing parser correctly finds the maximum scoring 1-Endpoint-Crossing tree according to the crossing-sensitive third-order factorization (Expression 5.1) in $O(n^4)$ time and $O(n^3)$ space.*

Proof. The correctness of scoring follows from Lemmas 6, 13, and 14. The search space of 1-Endpoint-Crossing trees was in Lemma 5 and the time and space complexity described in Lemma 15. □



(a) Prague style (b) Mel'čukian style (c) Stanford style

Figure 5.11: Different representations for the conjunction in the phrase "like cats and dogs".

5.5 Experiments

The crossing-sensitive third-order parser was implemented as an alternative parsing algorithm within *dpo3*, the third-order parsing implementation of Koo and Collins (2010)[1].

In Chapter 2, we saw how the accuracy of parsers with various factorizations was impacted by how conjunctions were represented in treebanks. The treebanks used to prepare the CoNLL shared task data vary widely in their conventions for representing conjunctions, modal verbs, determiners, and other decisions (Zeman et al., 2012). Our experiments use the newly released HamleDT software (Zeman et al., 2012) that normalizes these treebanks into one standard format and also provides built-in transformations to other widely used conjunction styles. The relative proportions of 1-Endpoint-Crossing and projective trees are similar to the unnormalized case (see Table A.1 in Appendix A).

The three conjunction styles we experiment with are the default Prague style (Böhmová, Hajič, Hajičová, and Hladká, 2001, Figure 5.11a), Mel'čukian style (Mel'čuk, 1988, Figure 5.11b), and Stanford style (De Marneffe and Manning, 2008, Figure 5.11c). Examining Figure 5.11, we see that under our grandparent-sibling factorization, *cats* and *dogs* would never appear in the same scope for the Prague style (as they are siblings on different sides of the head *and*). As in Chapter 2, *cats*, *and*, and *dogs* do appear together in a grandparent relationship in the Mel'čukian style and in a sibling relationship in the Stanford style. We would therefore expect to see larger gains for including grandparents and siblings under the latter two representations.

The experiments also include a nearly projective dataset, the English Penn Treebank

[1] `http://groups.csail.mit.edu/nlp/dpo3/`

(Marcus et al., 1993), converted to dependencies with PennConverter (Johansson and Nugues, 2007). Table 5.4 gives the number of sentences in each of the training and test sets used.

We use marginal-based pruning based on an edge-factored arborescence model (McDonald et al., 2005b). For each word, we limit its set of potential parents to those with a marginal probability of at least .1 times the probability of the most probable parent, and cut off this list at a maximum of 20 potential parents per word. To ensure that there is always at least one projective and/or 1-Endpoint-Crossing tree achievable, we always include the artificial root as an option. Note that since we have limited the number of parents per word to a constant k (20), since each step of our parsing algorithm always includes at least one edge, the top-down implementation runs in $O(kn^3)$.

Following standard practice, English results use automatically produced part-of-speech tags and results exclude punctuation, while the results for all other languages use gold part-of-speech tags and include punctuation. Following Carreras (2007) and Koo and Collins (2010), before training we transform the training set trees to be the best achievable within the model class (i.e., the closest projective tree or 1-Endpoint-Crossing tree). We train all models for five iterations of averaged structured perceptron training. For English, we use the model after the iteration that performs best on the development set; for all other languages, we simply take the model produced after the fifth iteration.

	Dutch	Czech	Portuguese	Danish	Swedish	English
Training	13349	25364	9071	5190	11042	39832
Testing	386	286	288	322	389	2416

Table 5.4: Number of sentences in the datasets

5.5.1 Results

Results for edge-factored and (crossing-sensitive) grandparent-sibling factored models for both projective and 1-Endpoint-Crossing parsing are in Tables 5.5 and 5.6. In 14 out of the 16 experimental set-ups, the third-order 1-Endpoint-Crossing parser is more accurate than

the third-order projective parser. It is significantly better than the projective parser in 9 of the set-ups and significantly worse in none.

Impact of Data Representation

For the datasets using Mel'čukian and Stanford style conjunctions, the third-order 1-EC parser is always more accurate than the edge-factored 1-EC model. With Prague style conjunctions, the edge-factored model is more accurate for two of the five languages. Conjunctions are one of the main examples for which the independence assumptions implied by an edge-factored model are most strongly violated, and so the grandparent-sibling model's inability to include both conjuncts in the same scope for the Prague-style conjunctions negates much of the expected benefit of the grandparent-sibling factorization.

The most significant benefit from incorporating grandparent-sibling features into the 1-Endpoint-Crossing parser occurs with Stanford-style conjunctions. Unlike the *GSib* model, in which all edges receive grandparent and sibling contexts, in the *CS-GSib* an edge may lose access to its grandparent and/or sibling based on the pattern of crossing edges in its local neighborhood. With Mel'čukian style conjunctions, it might be more detrimental to lose access to the grandparent, while with Stanford style conjunctions, it might be worse to lose access to the adjacent sibling.

Table 5.7 shows how often each of the $\mathrm{GrandSib}$, Grand, Sib, Edge, and $\mathrm{CrossedEdge}$ parts would have been used when the 1-EC CS-GSib parser was producing the trees for the Mel'čukian and Stanford style test sets. In both representations, the parser is able to score with a sibling context more often than it is able to score with a grandparent, perhaps explaining why the datasets using the Stanford conjunction representation saw the largest gains from including the higher order factors into the 1-Endpoint-Crossing parser.

Speed

While the asymptotic times of the projective and 1-Endpoint-Crossing third order parsers match exactly ($O(n^4)$ for an unpruned model and $O(kn^3)$ with pruning), one may wonder

how large the hidden constants are in practice.

Table 5.8 shows the parsing speed of the trained parsers when parsing the Stanford-style data sets.[2] The third-order 1-Endpoint-Crossing parser runs only a small constant times slower than the third-order projective parser, running at a relative speed of .37-.47 times the speed of the third-order projective parser.

[2]Parsing speed is correlated with the amount of pruning. The level of pruning mentioned earlier is relatively permissive, retaining 39.0-60.7% of the edges in the complete graph; a higher level of pruning could likely achieve much faster parsing times with the same underlying parsing algorithms.

Model		Dutch	Czech	Portuguese	Danish	Swedish
		Prague				
Proj	GSib	80.45	85.12	88.85	**88.17**	85.50
Proj	Edge	80.38	84.04	88.14	**88.29**	**86.09**
1-EC	CS-GSib	**82.78**	**85.90**	**89.74**	**88.64**	85.70
1-EC	Edge	**83.33**	84.97	**89.21**	**88.19**	**86.46**
		Mel'čukian				
Proj	GSib	82.26	**87.96**	89.19	**90.23**	**89.59**
Proj	Edge	82.09	86.18	88.73	89.29	**89.00**
1-EC	CS-GSib	**86.03**	**87.89**	**90.34**	**90.50**	**89.34**
1-EC	Edge	85.28	**87.57**	**89.96**	**90.14**	88.97
		Stanford				
Proj	GSib	81.16	86.83	88.80	**88.84**	87.27
Proj	Edge	80.56	86.18	88.61	**88.69**	**87.92**
1-EC	CS-GSib	**84.67**	**88.34**	**90.20**	**89.22**	**88.15**
1-EC	Edge	83.62	87.13	89.43	**88.74**	87.36

Table 5.5: Overall Unlabeled Attachment Scores (UAS) for all words. CS-GSib refers to the crossing-sensitive grandparent-sibling factorization proposed in this chapter. Data sources: CoNLL-2006 shared task (Buchholz and Marsi, 2006) (Danish, Dutch, Portuguese, Swedish); CoNLL-2007 shared task (Nivre et al., 2007a) (Czech), normalized and then transformed to use one of three different conjunction representation styles using HamleDT (Zeman et al., 2012). For each data set, we bold the most accurate model and those not significantly different from the most accurate (sign test, $p < .05$). Languages are sorted in increasing order of projectivity (Table A.1).

Model	UAS
Proj GSib	**93.10**
Proj Edge	92.63
1-EC CS-GSib	**93.22**
1-EC Edge	92.80

Table 5.6: English results

Part Used	Dutch	Czech	Portuguese	Danish	Swedish
	Mel'čukian				
CrossedEdge	8.5	4.5	3.2	1.4	1.2
GrandSib	81.2	89.1	90.7	95.7	96.2
Grand	1.1	0.5	0.8	0.3	0.2
Sib	9.0	5.8	5.2	2.6	2.3
Edge	< 0.1	< 0.1	0	< 0.1	0
	Stanford				
CrossedEdge	8.4	5.1	3.3	2.0	1.8
GrandSib	81.4	87.8	90.5	94.2	95.2
Grand	1.1	0.5	0.7	0.3	0.3
Sib	8.9	6.5	5.2	3.5	2.6
Edge	< 0.1	0.1	0	< 0.1	0

Table 5.7: The proportion of edges in the output trees from the CS-GSib 1-Endpoint-Crossing parser that would have used each of the five part types for scoring to produce the predicted trees.

Model		Dutch	Czech	Portuguese	Danish	Swedish
Proj	GSib	222 w/s	189 w/s	231 w/s	268 w/s	183 w/s
1-EC	CS-GSib	100 w/s	89 w/s	86 w/s	104 w/s	71 w/s

Table 5.8: Parsing speed measured in words per second.

Chapter 6

Conclusions and Future Work

This thesis showed alternative factorizations and classes of output spaces for the dependency parsing problem, and provided efficient exact algorithms for solving these new formulations of the dependency parsing optimization problem.

In **Chapter 2**, we examined English dependency parsing using an existing projective parser capable of factorizations including grandparents and siblings. We showed that the accuracy gains from various features depended on how linguistic constructions (such as conjunctions) were represented in dependencies and whether the factorizations used included all relevant words within the same scope.

We then broadened our outlook to consider other languages besides English, which pose computational challenges due to crossing dependencies. Existing definitions of output spaces that allowed crossing dependencies become NP-hard for factorizations beyond single edges or require a prohibitive $O(n^7)$ parsing time for even an edge-factored model.

Chapter 3 introduced the characterization of *gap inheritance*, referring to whether a child node has descendants in more than one of its parent's descendant intervals. Our corpus analysis showed that across tens of thousands of trees for sentences in a variety of natural languages, *all* well-nested, block degree 2 dependency trees had gap inheritance degree at most 1. Further restricting the gap inheritance degree to be 0 still covered 90.4% of the structures in these corpora. We showed $O(n^6)$ and $O(n^5)$ edge-factored parsing

algorithms for well-nested, block degree 2 trees restricted to at most one gap inheriting child per parent and trees with no gap inheritance, respectively.

In **Chapter 4** we defined the *1-Endpoint-Crossing* property: whether all edges that cross a common edge are incident to a common vertex. This property holds for 95.8-99.8% of dependency parses across a variety of natural languages. We provided an $O(n^4)$ edge-factored parsing algorithm, continuing to narrow the gap between the time required for parsing projective and mildly non-projective languages.

Finally in **Chapter 5** we introduced a crossing-sensitive third-order factorization that simplifies to the standard grandparent-sibling factorization when applied to projective trees. We provided a parsing algorithm that finds the optimal 1-Endpoint-Crossing tree under this factorization in $O(n^4)$ time. The implemented parser is significantly more accurate than the third-order projective parser in nine out of sixteen experimental set-ups and is significantly less accurate on none. Moreover this benefit comes at no additional asymptotic cost, thereby providing us with a parser capable of producing a wider variety of structures that can be used in contexts where the third-order projective parser may have been used (see Table 6.1 for a summary).

6.1 Future Directions

We see several avenues for future work that builds on this thesis, which can be loosely divided into three categories: (i) faster and more accurate parsing, (ii) theoretical questions about these new tree classes, and (iii) applications to other natural language processing problems.

6.1.1 Developing Faster Variants of 1-Endpoint-Crossing Parsing

Most edges in dependency trees are short. When the length of edges is bounded by a constant, the maximum scoring *sequence* of projective trees can be found in $O(n)$ time (Eisner and Smith, 2010). This algorithm was used as a first-stage pruning step in a structured

Output Space	Coverage	Edge	CS-GrandSib
Projective	63.6-90.2%	$O(n^3)$	$O(n^4)$
Arborescences	100%	$O(n^2)$	?
Well-nested, block degree 2	95.4-99.9%	$O(n^7)$	?
1-Endpoint-Crossing	95.8-99.8%	$O(n^4)$	$O(n^4)$

Table 6.1: Coverage and (factorization-dependent) parsing time for classes of trees. Coverage is measured by proportion of training set sentences from CoNLL-X data (see Table 4.1 in Chapter 4 for details). CS-GrandSib is the crossing-sensitive grandparent-sibling factorization proposed in Chapter 5. When applied to projective trees, the CS-GrandSib factorization exactly simplifies to the GrandSib factorization of Koo and Collins (2010). For arborescences, incorporating grandparents and/or siblings *everywhere* has been shown to be NP-hard; the status of the problem of producing the maximum arborescence under the CS-GrandSib model is so far unknown. Similarly, it is so far not known what the parsing time would be for the well-nested block degree 2 trees under this model.

prediction cascade (Weiss and Taskar, 2010) that culminated in a full third-order projective parser which maintained comparable accuracy while parsing about 200 times faster (Rush and Petrov, 2012). It would be interesting to consider what could be done in a linear-time pass if the final output desired is a 1-Endpoint-Crossing tree.

While in the projective case it was sufficient to consider edge length, there is another distance to consider bounding in the 1-Endpoint-Crossing case: the distance between the endpoints of an edge and its crossing point. The linguistic examples of cross-serial dependencies and wh-movement presented in Chapter 4 tend to have *crossing points immediately adjacent to one of the endpoints of the edge*. Table 6.2 shows the coverage of various constant-sized maximum distances between one of the endpoints of an edge and its crossing point.

	Arabic		Czech		Danish		Dutch		Portuguese		Swedish	
1-Endpoint-Crossing	1457	(99.8)	71810	(98.8)	5144	(99.1)	12785	(95.8)	9007	(99.3)	10902	(98.7)
b=1	1418	(97.1)	69513	(95.6)	5023	(96.8)	12169	(91.2)	8532	(94.1)	10524	(95.3)
b=2	1440	(98.6)	71157	(97.9)	5099	(98.2)	12576	(94.2)	8872	(97.8)	10712	(97.0)
b=3	1449	(99.2)	71500	(98.3)	5125	(98.7)	12683	(95.0)	8953	(98.7)	10785	(97.7)
Projective	1297	(88.8)	55872	(76.8)	4379	(84.4)	8484	(63.6)	7353	(81.1)	9963	(90.2)
Sentences	1460		72703		5190		13349		9071		11042	

Table 6.2: Coverage of 1-Endpoint-Crossing trees, when for every edge e_{uv} the $\mathcal{P}t(e_{uv})$ is restricted to be within a distance b of u or v, on the CoNLL-X training sets (Buchholz and Marsi, 2006).

6.1.2 Alternative Descriptions of 1-Endpoint-Crossing Trees

Classes of trees can be defined in several ways: (i) by giving a generative procedure that gives rise to the class, (ii) by defining an automata that recognizes the class, or (iii) by giving a definition based on the tree's structure. This thesis defined classes of trees using the third approach.

For other classes of trees, the connection between these three ways of describing a class are better understood. Projective dependency grammars are weakly equivalent to context-free grammars (Gaifman, 1965). The context-free languages can be recognized by a pushdown automaton (Chomsky, 1962; Evey, 1963) (see Hopcroft, Motwani, and Ullman (2006, pp. 243-252)), leading to the popularity of transition-based parsing systems that use a stack for projective parsing (see Nivre (2008) for an overview).

Lexicalized Tree Adjoining Grammars (LTAG) (Joshi and Schabes, 1997) can be recognized by an embedded pushdown automata (Vijay-Shanker, 1987) that uses a sequence of stacks. Their derivation trees have also been structurally characterized by Bodirsky et al. (2005) as being well-nested and having gap degree at most one (see Chapter 3).

We pose the questions:

- Is there a natural generative procedure that gives rise to 1-Endpoint-Crossing trees?
- Is there a natural automata characterization that recognizes 1-Endpoint-Crossing trees?

Directional variants

Directional variants of 1-Endpoint-Crossing trees might be interesting to investigate. The definition of 1-Endpoint-Crossing trees is symmetric between a tree over a sentence and the reverse of the sentence. However, natural language is fundamentally asymmetric (Kayne, 1994). This is intuitive: human listeners hear sentences in the order in which the words were spoken. Explicitly defining directional versions of the 1-Endpoint-Crossing property may lead to a related class of trees more amenable to left-to-right parsing.

Consider a pair of crossing edges in a 1-Endpoint-Crossing tree. Each edge in the pair's crossing point is either the left or right endpoint of the other edge. The four cases of the choice of crossing points for legal 1-Endpoint-Crossing trees are shown in Figure 6.1. Figure 6.1d would appear to require two stacks for left-to-right recognition, while its symmetric variant (Figure 6.1a) appears easier for left-to-right parsing.

Chapter 4 showed that 1-Endpoint-Crossing trees are a sub-class of 2-planar trees (those that can be parsed using two stacks). We suspect that there exist directional variants related to the 1-Endpoint-Crossing class that may be recognizable using one stack and a small constant amount of storage (similar to the stack and constant-sized buffer used in the parser of Marcus (1980), but not necessarily used in the same way).

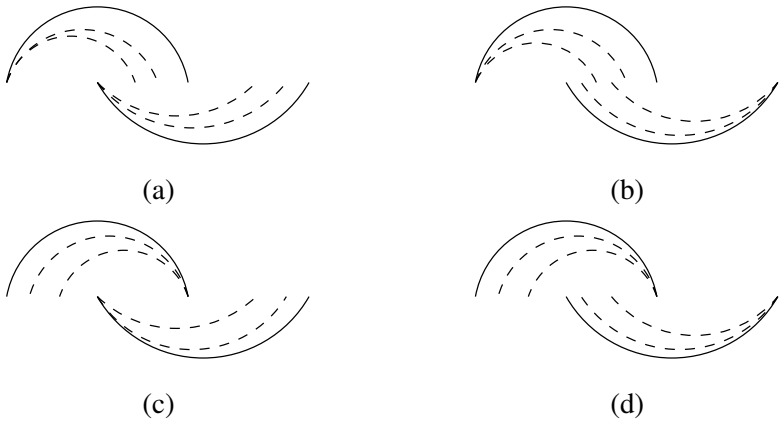
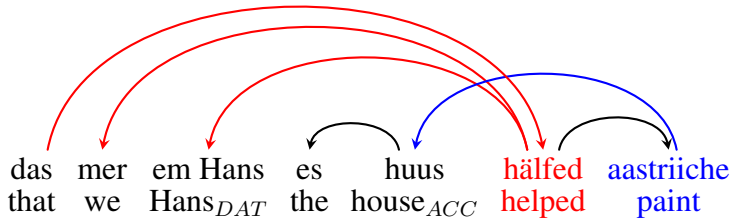


Figure 6.1: Crossing point possibilities for the solid edges

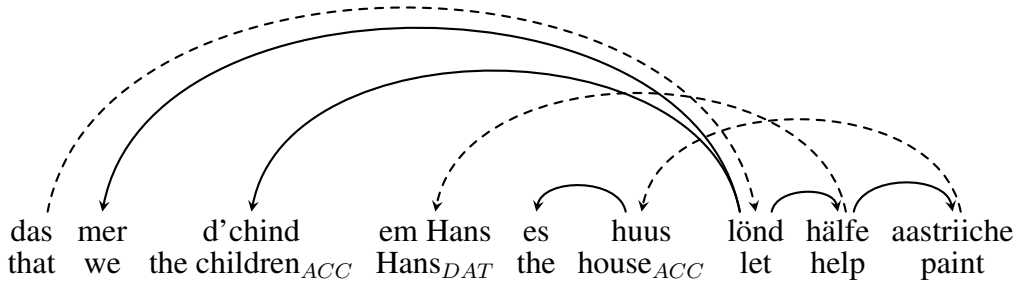
6.1.3 Linguistic Connections

Another area for future research is to investigate the linguistic relevance of these proposed tree classes: which phenomena lead to sentences that have 1-Endpoint-Crossing analyses, and which phenomena tend to introduce violations of the 1-Endpoint-Crossing property? Below is a (non-exhaustive) list of a few phenomena that give rise to crossed dependencies. For each phenomena we provide an example for illustrative purposes that has been discussed in the linguistics literature, a potential dependency tree analysis of the example, and whether these example analyses are 1-Endpoint-Crossing trees. Many of these phenomena yield 1-Endpoint-Crossing trees when the embedding is limited to two clauses but violate the 1-Endpoint-Crossing property when three or more clauses are involved.

Cross-serial dependencies The classic textbook example of a cross-serial sentence (e.g., Jurafsky and Martin (2008), p.538) is a Swiss-German sentence from (1) in Shieber (1985). While previously discussed in Section 4.6.2, for completeness it is reprised below:

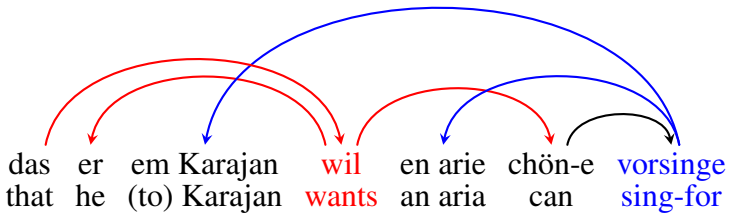


If a third level of embedding is added (paraphrased as "we let the children help Hans paint the house"), then the sentence (from (5) in Shieber (1985)) is *not* a 1-Endpoint-Crossing tree:



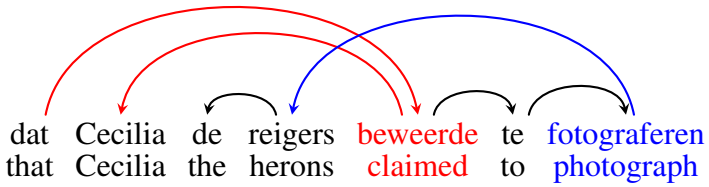
For example, the edge (*help,Hans*) is crossed by both (*that,let*) and by (*paint,house*), which do not share a vertex.

Verb projection raising Haegeman and Van Riemsdijk (1986) give the below example from Zurich German ((42b) in that paper) as difficult to account for under some syntactic analyses:



The sentence is glossed as "that he wants to be able to sing an aria for Karajan". The sentence presents difficulties for these analyses because *an aria* is between *wants* and *can sing-for*. This sentence is a 1-Endpoint-Crossing tree: even though the edge *(sing-for,(to) Karajan)* is crossed by multiple edges, they both share an endpoint at the verb *wants*.

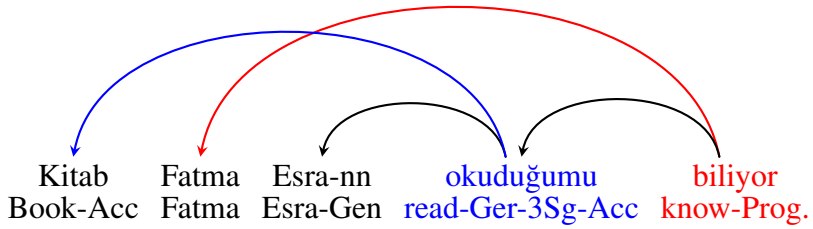
Remnant Extraposition Below is an example of remnant extraposition from Dutch, argued to be neither verb raising nor extraposition (example (8b) from Broekhuis, Den Besten, Hoekstra, and Rutten (1995)): The sentence is translated as "that Cecilia claimed to take a picture of the herons".



The edge (*photograph,herons*) is crossed by multiple edges, but all have a shared endpoint at the verb *claimed*. Note that a third level of embedding would cause a 1-Endpoint-Crossing violation, just as we saw for three clauses in the cross-serial case.[1]

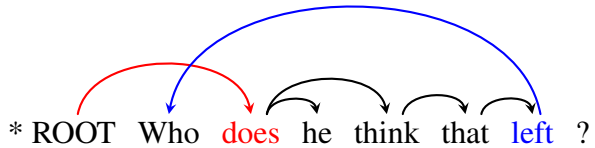
[1]We thank Aravind Joshi for pointing out this case.

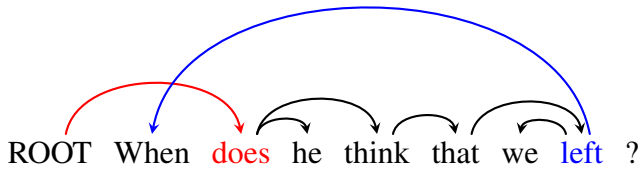
Scrambling Scrambling, when words occur in a non-canonical word order, also can give rise to discontinuous constituents (and thus crossing dependencies). Below is an example of scrambling from Turkish (example (6b) from Hoffman (1995)):



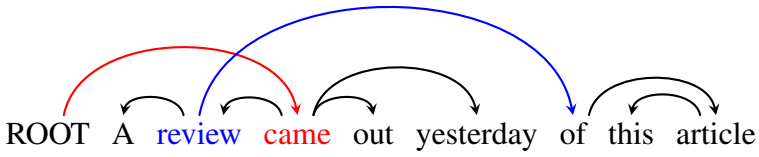
The sentence is translated as "As for the book, Fatma knows that Esra read it". Again the 1-Endpoint-Crossing property is satisfied.

Wh-movement Below are a few examples of wh-movement from (1) in Kroch (1989). The first is *ungrammatical* and the second is not. *Both* the ungrammatical and grammatical sentences are 1-Endpoint-Crossings. The sentences are drawn including an artificial root node to the left. Without including the artificial root edges, we would see no crossings at all in either sentence.

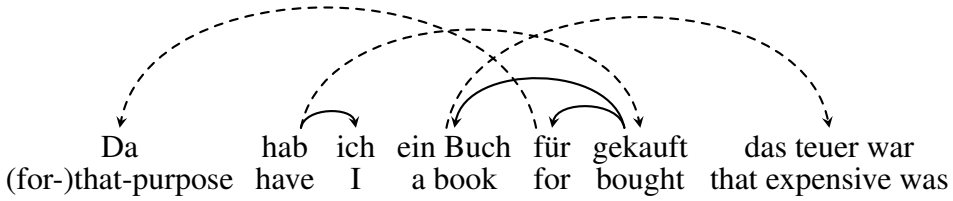




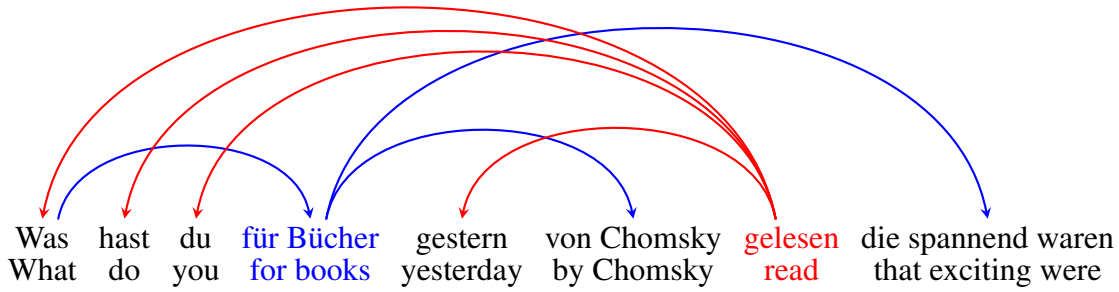
Extraposition Below is an English example of extraposition from Ross (1986) and a plausible dependency structure.



Dafür-split in German Chen-Main and Joshi (2012) motivate the necessity of structures that are not well-nested with two types of constructions, *dafür*-split and a combination of extraposition and a split quantifier in the same sentence. Even though 1-Endpoint-Crossing trees can produce ill-nested structure (e.g., Figure 4.1a in Chapter 4), none of these examples are 1-Endpoint-Crossing. Below is the first such sentence from Chen-Main and Joshi (2012) (example (1b)) and one plausible dependency analysis (from Figure 8 in that paper). The sentence is translated as "For that purpose, I bought a book that was expensive."



What-for split in German Chen-Main and Joshi (2012) use the *what-for* split in German to motivate the ability to produce trees with block degree greater than two (gap degree greater than one). All examples and analyses in that section are 1-Endpoint-Crossing trees. We show the most complex example below, that has gap degree 3/block degree 4 (example (5)/Figure 17 in that paper), translated as "What books by Chomsky that were exciting did you read yesterday?"



The edge *(for books, that exciting were)* is crossed by many edges, all with a shared vertex at *read*. Similarly the edge (*read*,*do*) is crossed by multiple edges, but they have a shared vertex at *for books*.

Treebank Examples Appendix A gives examples from the dependency treebanks we have used in this thesis that are not 1-Endpoint-Crossing trees. Many of these examples become 1-Endpoint-Crossing under other conventions and analyses. For example, the below English sentence:

(1) ROOT The units have worked on 37 investment banking deals this year , he says , though not all of them have panned out .

is not even 1-Endpoint-Crossing with PennConverter dependencies (which take *says* as the root of the sentence) but is *projective* with Stanford dependencies (which take *worked* to be the root of the sentence).

Discussion From the limited set of examples above, we see that a variety of types of discontinuities in language have at least some examples that are 1-Endpoint-Crossing trees. In many of these examples, *verbs* served as crossing points; a further exploration into this may be interesting.

Note that more complex sentences exhibiting the same phenomena may violate the 1-Endpoint-Crossing property. 1-Endpoint-Crossing trees are defined through a forbidden substructure characterization: a subgraph of three edges such that two vertex-disjoint edges both cross a third (Section 4.8). Any construction that involves only two crossing edges

will *always* be a 1-Endpoint-Crossing tree. Because edges incident to the same vertex cannot cross each other, any violation of the 1-Endpoint-Crossing property requires the involvement of at least six distinct words. Just as some difficult computational problems become tractable when some parameter of the problem is fixed to be a small constant (Downey and Fellows, 1999), it may be that some of the above types of discontinuities yield 1-Endpoint-Crossing trees when the number of clauses involved in these crossings is limited.

6.1.4 Applications beyond Parsing

Many other natural language applications build up output structures by concatenating intervals. For example, the semantic parser of Liang, Jordan, and Klein (2013) builds up structures through a procedure similar to projective parsing. Phrase-based machine translation systems (Koehn, Och, and Marcu, 2003) build up output translations by concatenating intervals of translated phrases. Perhaps the approach taken in this thesis of constructing output structures using intervals *with an exterior vertex* could also be useful in some of these applications.

While 1-Endpoint-Crossing trees were motivated by problems arising in natural language parsing, the problem statement and parsing algorithm are stated entirely in terms of vertices, edges, and their crossing pattern relative to the input sequence of vertices. There may be other structured prediction problems outside of natural language where these algorithms that can produce outputs with crossings might be useful, such as protein folding or other biological applications.

Appendix A

Additional Information About the Coverage of 1-Endpoint-Crossing Trees

To assist future research into 1-Endpoint-Crossing trees, we provide a script that verifies whether trees are 1-Endpoint-Crossing and if not, outputs the witnessing crossing edges. The script is available at: `http://www.cis.upenn.edu/~epitler/software/check1EC.pl`.

Table A.1 shows the coverage of projective trees and 1-Endpoint-Crossing trees on normalized treebanks with various conjunction representations.

Below are five randomly selected trees that are not 1-Endpoint-Crossing from English (Penn Treebank trees (Marcus et al., 1993) with PennConverter dependencies (Johansson and Nugues, 2007)), Danish (CoNLL-X, (Buchholz and Marsi, 2006)), and Dutch (CoNLL-X, (Buchholz and Marsi, 2006)) training sets. We provide the full sentence and the crossing edges that violate the 1-Endpoint-Crossing property.

For Danish and Dutch we also indicate whether the sentence still contains a 1-Endpoint-Crossing violation after normalizing the treebanks according to HamleDT (Zeman et al., 2012) (with the default Prague conjunction style).

Tree Class	Dutch	Czech	Portuguese	Danish	Swedish
			Prague		
Projective	64.1	75.5	78.6	84.1	88.0
1-Endpoint-Crossing	95.7	98.6	97.3	99.3	98.4
			Mel'čukian		
Projective	63.1	73.1	74.6	80.8	83.3
1-Endpoint-Crossing	95.4	97.5	96.5	98.9	97.6
			Stanford		
Projective	63.5	73.2	75.6	80.3	85.0
1-Endpoint-Crossing	95.0	97.5	96.4	98.4	97.8

Table A.1: Proportion of training set sentences covered in *normalized treebanks*, using one of three different styles of representation for conjunctions. Data sources: CoNLL-2006 shared task (Buchholz and Marsi, 2006) (Danish, Dutch, Portuguese, Swedish); CoNLL-2007 shared task (Nivre et al., 2007a) (Czech), normalized using HamleDT (Zeman et al., 2012).

A.1 English

(1) ROOT The units have worked on 37 investment banking deals this year , he says , though not all of them have panned out .

The edge $(ROOT, says)$ crosses $(have, .)$ and $(worked, though)$.

(2) ROOT Yet Israel will neither share power with all these Arabs nor , says its present prime minister , redraw its borders closer to its pre-1967 Jewish heartland .

The edge $(ROOT, says)$ is crossed by both $(will, .)$ and by $(nor, redraw)$.

(3) ROOT Some may have forgotten – and some younger ones may never have experienced – what it 's like to invest during a recession .

The edge $(forgotten, 's)$ is crossed by $(may, –)$, $(may, –)$, (may, and), and $(like, what)$.

(4) ROOT That 's leaving small investors with cold feet , they said , and prompting institutions to take a reserved stance on the sidelines as well , at least until the market in New York settles down somewhat .

The edge $(\mathit{ROOT}, \mathit{said})$ is crossed by $(\mathit{'s}, .)$ and $(\mathit{leaving}, \mathit{and})$.

(5) ROOT Chemical Waste Management Inc. , proposed global offering of 8,500,000 shares_1 of_1 common stock , of_2 which seven million of_3 the shares_2 will_1 be offered in the U.S. and 1,500,000 shares_3 will_2 be offered overseas , via Merrill Lynch Capital Markets -LRB- domestic -RRB- and Kidder , Peabody & Co . -LRB- international -RRB- .

The edge $(\mathit{shares}_3, \mathit{of}_2)$ crosses $(\mathit{shares}_1, \mathit{will}_1)$ and $(\mathit{and}, \mathit{will}_2)$.

A.2 Danish

(6) ROOT Som den drevne mindretalsregering , den er , har ministeriet Poul Schl uter IV netop lagt et finanslovforslag frem , som fortsætter den moderate økonomiske politik fra de senere år - men uden provokerende eller ideologiske spareforslag , som på forhånd kan støde de afgørende borgerlige midterpartier over i armene på Socialdemokratiet .

The edge $(\mathit{et}, \mathit{fortsætter})$ is crossed by $(\mathit{har}, ,)$ and $(\mathit{lagt}, \mathit{frem})$.

The tree is still not 1-Endpoint-Crossing after normalization, but the violation has shifted: in the normalized tree, the edge $(\mathit{finanslovforslag}, \mathit{men})$ is crossed by both $(\mathit{har}, ,)$ and $(\mathit{lagt}, \mathit{frem})$.

(7) ROOT Han omtaler Parzival som en af " brød-artiklerne " - og den går der 75 eksemplarer af om året stadig væk .

The edge $(\mathit{af}, \mathit{den})$ is crossed by $(\mathit{går}, \mathit{stadig})$, $(\mathit{75}, \mathit{om})$, and $(\mathit{og}, \mathit{går})$.

After normalization, the same edge is still a conflict, but it is instead crossed by $(\mathit{går}, \mathit{stadig})$, $(\mathit{eksemplarer}, \mathit{om})$, and $(\mathit{og}, \mathit{går})$.

(8) ROOT Fra Ungarn kommer de raske atleter og fra Afrika en palet med kolibrier - mærkerne fra Solens Rige får det til at svimle .

The edge $(ROOT, Afrika)$ is crossed by $(kommer, .)$, $(kommer, får)$, (og, en), and $(kommer, \text{-})$.

After normalization this is now a 1-Endpoint-Crossing tree.

(9) ROOT Men da skønnet over , i hvilken grad Folketinget er indblandet såvel er opfundet som afgøres af Ombudsmanden , er afvisningen blot at tage til efterretning .

The edge $(som, såvel)$ is crossed by (da, er), $(er, ,)$, $(opfundet, af)$, and $(er, skønnet)$.

After normalization this is now a 1-Endpoint-Crossing tree.

(10) ROOT Administrerende direktør Peter Christoffersen siger , at der hverken er forhandlinger eller sonderinger mellem Baltica og Skandia i øjeblikket .

The edge $(eller, hverken)$ is crossed by (er, i), (at, er), (er, der), and $(forhandlinger, mellem)$.

After normalization this is now a 1-Endpoint-Crossing tree.

A.3 Dutch

(11) ROOT Ook ” honger ” en het spitse ” Party-conversatie ” behoorden tot deze bepaalde categorie die een sociale inhoud van en aanmerkelijk beter uit_de_verf kwam dan de socialistische gedachten die vaak zeer banaal aandeden .

The edge $(die, kwam)$ is crossed by $(ROOT, sociale)$, $(ROOT, een)$, $(ROOT, van)$, $(ROOT, aanmerkelijk)$, $(ROOT, inhoud)$, and $(beter, dan)$.

The same violation remains after normalization.

(12) ROOT De lees- en weethonger in binnen- en buitenlandse gezinnen gaan we aan- en afvoeren

The edge (*aan-*, *en*) crosses (*weethonger*, *De*), (*ROOT*, *gaan*), and (*gaan*, *afvoeren*).

The same violation remains after normalization.

(13) ROOT Joe_Valachi zou er een kookboek over kunnen schrijven .

The edge (*schrijven*, *kookboek*) crosses both (*zou*, *kunnen*) and (*over*, *er*).

The same violation remains after normalization.

(14) ROOT " Ze hebben me geadviseerd , verder nauwkeurig onder behandeling te blijven bij een Amsterdamse nierspecialist en dokter Rolink , die mij door_en_door kent en die precies weet wat ik wel en niet kan .

This sentence has several violations: each of the edges (*blijven*, *verder*), (*blijven*, *nauwkeurig*), and (*blijven*, *onder*) crosses (*geadviseerd*, *te*) and both groups also cross (*behandeling*, *bij*).

The same violations remain after normalization.

(15) ROOT Wie is thans de VHO ? Postbus_2135 in Utrecht , waarheen 25 gulden moet worden gegireerd , blijkt op naam te staan van de VHO , Tafelbergdreef_20 in Utrecht .

The three edges (*blijkt*, *te*), (*staan*, *op*) and (*naam*, *van*) are in an $A\ B\ C\ A\ B\ C$ configuration and so all cross each other.

The same violations remain after normalization.

Appendix B

GrandSib-Crossing Parser Invariants

Each type of sub-problem is associated with both *interior* and *exterior* invariants (involving specified edges, bad (or not bad) contexts, or crossings). When two or more sub-problems are combined, their interior and exterior invariants are also combined to produce new interior and exterior invariants for the larger combined subforest. Table B.1 shows the full set of interior and exterior invariants for each type of sub-problem. For ease of exposition, left versions are given for asymmetric cases (i.e., $dir = L$); the right versions are symmetric.

	Interior	Exterior
TriG[i,j,g,L]	• Rooted at: i	• Includes edge $\vec{e}_{gi}$ • $\neg BadContext(i, j, L)$
Tri[i,j,L]	• Rooted at: i	• $BadContext(i, j, L)$
TrapG[i,j,g,L]	• Rooted at: i • Includes edge $\vec{e}_{ij}$	• Includes edge $\vec{e}_{gi}$ • $\neg BadContext(i, j, L)$

BoxG[i,j,g]	• Rooted at: i,j	• Includes edge $\vec{e}_{gi}$; $\vec{e}_{gi}$ not crossed • Includes edge $\vec{e}_{gj}$; $\vec{e}_{gj}$ not crossed • i and j are adjacent children on the same side of g
Trap[i,j,L]	• Rooted at: i • Includes edge $\vec{e}_{ij}$	• $BadContext(i, j, L)$
TwoRooted[i,j]	• Rooted at: i, j	• $BadContext(i, j, L)$ • $BadContext(i, j, R)$
OneFarCrossedG[i,j,g,L]	• Rooted at: i, j • Edge from j to leftmost child in (i, j) is crossed	• Includes edge $\vec{e}_{gi}$ • $\neg BadContext(i, j, L)$
LeftFarCrossed[i,j,L]	• Rooted at: i, j • Edge from i to rightmost child in (i, j) is crossed	• $BadContext(j, i, R)$
Chain[i,j]	• Rooted at: i, j • Edge from i to i's furthest child and edge from j to j's furthest child part of same chain of crossing edges	
TriFarCrossed[i,j,L]	• Rooted at: i • Edge from i to rightmost child in (i, j) crossed	

Page1[i,j,x]	• Rooted at: j, x • i, j, x involved in same chain of crossing edges • x has at least one child in (i, j) • i's parent is to the right of x's child or children	
Page2[i,j,x]	• Rooted at: i, j • i, j, x involved in same chain of crossing edges • x's parent in (i, j) • x has no children in $[i, j]$ • i has at least one child to the right of x's parent	
Chain_JFromI[i,j,x]	• Rooted at: i, x • i, j, x involved in same chain of crossing edges • j is descended from i • If i is not the parent of j, then x has exactly one child in (i, j)	
Chain_JFromX[i,j,x]	• Rooted at: i, x • i, j, x involved in same chain of crossing edges • j is descended from x • x has exactly one child in (i, j)	

LR[i,j,x,b_x]	• Rooted at: i, j, $\mathbb{1}_{[b_x=F]}x$ • Crossing Conditions • Exists some split point such that between x and left of the split point are only crossed by edges incident to i, and to the right only those incident to j	• Includes edge $\vec{e}_{ij}$ or edge $\vec{e}_{ji}$ • No other edge besides $\vec{e}_{ij}$/$\vec{e}_{ji}$ will cross the edges between x and $VCross(i, j, F, F, LR)$
L[i,j,x,b_i,b_j,b_x]	• Rooted at: $\mathbb{1}(b_i = F)i$, $\mathbb{1}(b_j = F)j$, $\mathbb{1}(b_x = F)x$ • Crossing Conditions • Edges between x and $VCross(i, j, b_i, b_j, L)$ have i as their crossing point	• Exists some external edge incident to i that will cross all edges between x and $VCross(i, j, b_i, b_j, L)$ • If $b_i = F$, $BadContext(i, j, L)$ • If $b_j = F$, $BadContext(i, j, R)$
N[i,j,x,b_i,b_j,b_x]	• Rooted at: $\mathbb{1}(b_i = F)i$, $\mathbb{1}(b_j = F)j$, $\mathbb{1}(b_x = F)x$ • Crossing Conditions • No edges between x and $VCross(i, j, b_i, b_j, N)$ are crossed	• Exists at least two external edges that force a point outside of $\{i, j\}$ to be the crossing point for the edges between x and $VCross(i, j, b_i, b_j, N)$ • If $b_i = F$, $BadContext(i, j, L)$ • If $b_j = F$, $BadContext(i, j, R)$

Table B.1: Internal and External Invariants

Appendix C

Full Dynamic Program for the GrandSib-Crossing Parser

The initial sub-problems are:

$\forall i Tri[i,i,L] = Tri[i,i,R] = TwoRooted[i,i] = TwoRooted[i,i+1] = 0$,

$\forall g, h > g \quad TriG[h,h,g,L] = GCO1(g,h,NIL) \quad TriG[h,h,g,R] = GCI1(g,h,NIL)$,

$\forall g, h < g \quad TriG[h,h,g,L] = GCI1(g,h,NIL) \quad TriG[h,h,g,R] = GCO(g,h,NIL)$.[1]

The final tree is in: $TriG[0,n,-1,L]$ (where 0 is the artificial root and -1 is an artificial parent of the root). All sub-problems left undefined (for example, $Int[i,j,T,T]$, in which all vertices must get their parent from the interval, or $L[i,i,x,b_i,b_j,b_x]$, which would have $VCross(i,i,b_i,b_j,L) = \emptyset$) have a score of $-\infty$.

$\mathrm{Score(GrandSib(g,h,m,s))}$ is abbreviated as $GS(g,h,m,s)$, $\mathrm{Score(Grand(g,h,m))}$ as $G(g,h,m)$, $\mathrm{Score(Sib(h,m,s))}$ as $S(h,m,s)$, $\mathrm{Score(Edge(h,m))}$ as $E(h,m)$, and

[1] These correspond to the null boundary cases described in Koo (2010, Appendix B.1.5).

These are used when $\vec{e}_{gh}$ is not crossed and there are no $GProj$ edges from h to any interior children (and so the score $GCI(g,h,NIL)$ is added); similarly when $\vec{e}_{gh}$ is not crossed and there are no $GProj$ edges from h to any exterior children ($GCO(g,h,NIL)$). In the projective case, since $\vec{e}_{gh}$ is never crossed and all edges to children are $GProj$, this simplifies to h having *no* inner (outer) children. In a non-projective tree, this covers both the case in which there are no children and the case in which the edge to the most outer interior/exterior child is crossed.

$\mathrm{Score}(\mathrm{CrossedEdge}(\mathrm{h}, \mathrm{m}))$ as $CE(h, m)$.

The implemented parser distinguishes betweeen inner and outer grandchildren; this can be determined from the indices of g, h, and m above.

Note that "triangles" have the same semantics as "intervals" where only one boundary point needs a parent. To emphasize the similarities to both the dynamic programs of Koo and Collins (2010) and Chapter 4, we use the most familiar vocabulary in each section and use the following syntactic sugar:
$TwoRooted[i, j] := Int[i, j, F, F]$, $Tri[i, j, L] := Int[i, j, F, T]$, and
$Tri[i, j, R] := Int[i, j, T, F]$.

$TriG[i,j,g,L] \leftarrow \max$
$$\begin{cases} \max_{k\in(i,j]} TrapG[i,k,g,L] + TriG[k,j,i,L] \\ TriFarCrossed[i,j,L] \end{cases}$$

$TriG[i,j,g,R] \leftarrow$ symmetric to above

$Tri[i,j,L] \leftarrow \max$
$$\begin{cases} \max_{k\in(i,j]} Trap[i,k,L] + TriG[k,j,i,L] \\ TriFarCrossed[i,j,L] \end{cases}$$

$Tri[i,j,R] \leftarrow$ symmetric to above

$TrapG[i,j,g,L] \leftarrow \max$
$$\begin{cases} GS(g,i,j,-) + TriG[i+1,j,i,R] \\ G(g,i,j) + Chain[i,j] \\ \max_{k\in(i,j)} \begin{cases} GS(g,i,j,k) + TrapG[i,k,g,L] + BoxG[k,j,i] \\ G(g,i,j) + TriFarCrossed[i,k,L] + TriG[k+1,j,i,R] \\ G(g,i,j) + Chain[i,k] + TriG[k,j,i,R] \end{cases} \end{cases}$$

$TrapG[i,j,g,R] \leftarrow$ symmetric to above

$Trap[i,j,L] \leftarrow \max$
$$\begin{cases} S(i,j,-) + TriG[i+1,j,i,R] \\ E(i,j) + Chain[i,j] \\ \max_{k\in(i,j)} \begin{cases} S(i,j,k) + Trap[i,k,L] + BoxG[k,j,i] \\ E(i,j) + TriFarCrossed[i,k,L] + TriG[k+1,j,i,R] \\ E(i,j) + Chain[i,k] + TriG[k,j,i,R] \end{cases} \end{cases}$$

$Trap[i,j,R] \leftarrow$ symmetric to above

$BoxG[i,j,g] \leftarrow \max_{k\in[i,j)}$
$$\begin{cases} TriG[i,k,g,L] + TriG[k+1,j,g,R] \\ TriG[i,k,g,L] + OneFarCrossedG[k,j,g,R] \end{cases}$$

$OneFarCrossedG[i,j,g,L] \leftarrow \max_{k\in[i,j)} TriG[i,k,g,L] + Chain[k,j]$

$OneFarCrossedG[i,j,g,R] \leftarrow$ symmetric to above

$LeftFarCrossed[i,j] \leftarrow \max_{k\in(i,j]} Chain[i,k] + Tri[k,j,R]$

$TwoRooted[i,j] \leftarrow \max$
$$\begin{cases} \max_{k\in[i,j)} Tri[i,k,L] + Tri[k+1,j,R] \\ \max_{k\in[i,j)} Tri[i,k,L] + LeftFarCrossed[k,j] \end{cases}$$

$Chain[i,j] \leftarrow \max_{k\in(i,j)} CE(i,k) + \max$
$$\begin{cases} LR[i,k,j,F] + TwoRooted[k,j] \\ \max_{l\in(i,k)} \\ \quad \begin{cases} CE(j,l) + TwoRooted[i,l] + L[l,k,i,F,F,F] + N[k,j,l,F,F,F] \\ CE(j,l) + TwoRooted[i,l] + TwoRooted[l,k] + L[k,j,l,F,F,F] \\ CE(j,l) + R[i,l,k,F,F,F] + TwoRooted[l,k] + L[k,j,l,F,F,F] \end{cases} \\ \max_{l\in(k,j)} R[i,k,l,F,F,F] + TwoRooted[k,l] + Page1[l,j,k] \end{cases}$$

$Page1[i,j,x] \leftarrow \max$
$$\begin{cases} CE(j,i) + L[i,j,x,F,F,F] \\ \max_{k\in(i,j)} CE(x,k) + TwoRooted[i,k] + Page2[k,j,i] \end{cases}$$

$Page2[i,j,x] \leftarrow \max_{k\in(i,j)} CE(k,x) + TwoRooted[i,k] + Page1[k,j,i]$

$TriFarCrossed[i,j,L] \leftarrow \max_{k \in (i,j)} CE(i,k) + \max$

$$\begin{cases} \max_{l \in (k,j], TF(T,\{b_l,b_m,b_r\})} \\ \quad \begin{cases} R[i,k,l,F,F,b_l] + Int[k,l,F,b_m] + L[l,j,k,b_r,T,F] \\ LR[i,k,l,b_l] + Int[k,l,F,b_m] + Int[l,j,b_r,T] \end{cases} \\ \max_{l \in (i,k), TF(T,\{b_l,b_m,b_r\})} \\ \quad \begin{cases} Int[i,l,F,b_l] + L[l,k,i,b_m,F,F] + N[k,j,l,F,T,b_r] \\ Int[i,l,F,b_l] + Int[l,k,b_m,F] + L[k,j,l,F,T,b_r] \\ R[i,l,k,F,b_l,F] + Int[l,k,b_m,F] + L[k,j,l,F,T,b_r] \end{cases} \end{cases}$$

$TriFarCrossed[i,j,R] \leftarrow$ symmetric to above

$LR[i,j,x,b_x] \leftarrow \max$

$$\begin{cases} L[i,j,x,F,F,b_x] \\ R[i,j,x,F,F,b_x] \\ \max_{k \in (i,j)} \\ \quad \begin{cases} Chain\_JFromI[i,k,x] + R[k,j,x,F,F,b_x] \\ Chain\_JFromX[i,k,x] + R\_XFromJ[k,j,x] \quad \text{if } b_x = T \\ Chain\_JFromX[i,k,x] + R[k,j,x,F,F,F] \quad \text{if } b_x = F \\ L[i,k,x,F,F,b_x] + Chain\_IFromJ[k,j,x] \\ L\_XFromI[i,k,x] + Chain\_IFromX[k,j,x] \quad \text{if } b_x = T \\ L[i,k,x,F,F,F] + Chain\_IFromX[k,j,x] \quad \text{if } b_x = F \end{cases} \end{cases}$$

$Chain\_JFromI[i,j,x] \leftarrow \max$

$$\begin{cases} CE(i,j) + L[i,j,x,F,F,F] \\ \max_{k \in (i,j)} CE(x,k) + Int[i,k,F,F] + Chain\_JFromX[k,j,i,L] \end{cases}$$

$Chain\_IFromJ[i,j,x] \leftarrow$ symmetric to above

$Chain\_JFromX[i,j,x] \leftarrow \max_{k \in (i,j)} CE(x,k) + Int[i,k,F,F] + Chain\_JFromI[k,j,i]$

$Chain\_IFromX[i,j,x] \leftarrow$ symmetric to above

$$N[i,j,x,b_i,b_j,F] \leftarrow \max_{k \in VCross(i,j,b_i,b_j,N)} CE(x,k) + \max \begin{cases} Int[i,k,b_i,F] + Int[k,j,F,b_j] \\ N[i,k,x,b_i,F,F] + Int[k,j,F,b_j] \end{cases}$$

$$N[i,j,x,b_i,b_j,T] \leftarrow \max_{k \in VCross(i,j,b_i,b_j,N)} CE(k,x) + \max \begin{cases} \max_{TF(T,\{b_l,b_r\})} Int[i,k,b_i,b_l] + Int[k,j,b_r,b_j] \\ N[i,k,x,b_i,F,F] + Int[k,j,T,b_j] \\ Int[i,k,b_i,T] + N[k,j,x,F,b_j,F] \end{cases}$$

$$L[i,j,x,b_i,b_j,F] \leftarrow \max_{k \in VCross(i,j,b_i,b_j,L)} \max \begin{cases} CE(x,k) + \max \begin{cases} Int[i,k,b_i,F] + Int[k,j,F,b_j] \\ \max_{TF(b_i,\{b_l,b_r\})} Int[i,k,b_l,F] + L[k,j,i,F,b_j,b_r] \\ L[i,k,x,b_i,F,F] + Int[k,j,F,b_j] \end{cases} \\ CE(i,k) + L[i,k,x,F,F,F] + Int[k,j,F,b_j] \quad \text{if } b_i = F \\ CE(i,k) + L\_IFromX[i,k,x] + Int[k,j,F,b_j] \quad \text{if } b_i = T \\ CE(k,i) + L\_JFromX[i,k,x] + Int[k,j,F,b_j] \quad \text{if } b_i = T \\ CE(k,i) + L[i,k,x,F,F,F] + Int[k,j,T,b_j] \quad \text{if } b_i = T \end{cases}$$

$$L[i,j,x,F,b_j,T] \leftarrow \max_{k \in VCross(i,j,F,b_j,L)} \max \begin{cases} CE(k,x) + \max \begin{cases} \max_{TF(T,\{b_l,b_r\})} \begin{cases} Int[i,k,F,b_l] + Int[k,j,b_r,b_j] \\ Int[i,k,F,b_l] + L[k,j,i,b_r,b_j,F] \end{cases} \\ L\_JFromI[i,k,x] + Int[k,j,F,b_j] \\ L[i,k,x,F,F,F] + Int[k,j,T,b_j] \end{cases} \\ CE(x,k) + L\_XFromI[i,k,x] + Int[k,j,F,b_j] \\ CE(i,k) + L[i,k,x,F,F,T] + Int[k,j,F,b_j] \end{cases}$$

$$L\_XFromI[i,j,x] \leftarrow \max \begin{cases} \max_{k\in(i,j)} CE(k,x)+ \\ \quad \begin{cases} Int[i,k,F,T] + Int[k,j,F,F] \\ Int[i,k,F,T] + L[k,j,i,F,F,F] \\ Int[i,k,F,F] + L\_IFromX[k,j,i] \\ L\_JFromI[i,k,x] + Int[k,j,F,F] \end{cases} \\ \max_{k\in(i,j)} CE(x,k) + L\_XFromI[i,k,x] + Int[k,j,F,F] \\ \max_{k\in(i,j)} CE(i,k) + L[i,k,x,F,F,T] + Int[k,j,F,F] \end{cases}$$

$$L\_IFromX[i,j,x] \leftarrow \max_{k\in(i,j)} \max \begin{cases} CE(x,k) + \max \\ \quad \begin{cases} Int[i,k,T,F] + Int[k,j,F,F] \\ Int[i,k,T,F] + L[k,j,i,F,F,F] \\ Int[i,k,F,F] + L\_XFromI[k,j,i] \\ L[i,k,x,T,F,F] + Int[k,j,F,F] \end{cases} \\ CE(k,i) + L\_JFromX[i,k,x] + Int[k,j,F,F] \\ CE(i,k) + L\_IFromX[i,k,x] + Int[k,j,F,F] \end{cases}$$

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

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

$$R[i,j,x,b_i,b_j,F] \leftarrow \text{symmetric to } L[i,j,x,b_i,b_j,F]$$

$$R[i,j,x,b_i,F,T] \leftarrow \text{symmetric to } L[i,j,x,F,b_j,T]$$

$R[i, j, x, b_i, T, T] \leftarrow$ 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 $L\_JFromI[i, j, x]$

Bibliography

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

M. Bansal and D. Klein. Web-scale features for full-scale parsing. In *Proceedings of ACL*, pages 693–702, 2011.

S. Bergsma, D. Yarowsky, and K. Church. Using large monolingual and bilingual corpora to improve coordination disambiguation. In *Proceedings of ACL*, pages 1346–1355, 2011.

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

M. Bodirsky, M. Kuhlmann, and M. Möhl. 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, 2005.

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. The Prague Dependency Treebank: Three-level annotation scenario. In Anne Abeillé, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers, 2001.

T. Brants and A. Franz. The Google web 1T 5-gram corpus version 1.1. *LDC2006T13*, 2006.

H. Broekhuis, H. Den Besten, K. Hoekstra, and J. Rutten. Infinitival complementation in Dutch: On remnant extraposition. *The Linguistic Review*, 12(2):93–122, 1995.

P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

S. Buchholz and E. Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 149–164, 2006.

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

X. Carreras, M. Collins, and T. Koo. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 9–16. Association for Computational Linguistics, 2008.

E. Charniak, D. Blaheta, N. Ge, K. Hall, J. Hale, and M. Johnson. BLLIP 1987-89 WSJ corpus release 1. *Linguistic Data Consortium, Philadelphia*, 2000.

J. Chen-Main and A. Joshi. Unavoidable ill-nestedness in natural language and the adequacy of Tree Local-MCTAG induced dependency structures. In *Proceedings of the Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 10)*, 2010.

J. Chen-Main and A. K. Joshi. A dependency perspective on the adequacy of Tree Local Multi-component Tree Adjoining Grammar. In *Journal of Logic and Computation Advance Access*, June 2012.

N. Chomsky. *Context-free grammars and pushdown storage*. 1962.

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

N. Chomsky. *Minimalist inquiries: the framework*. Number 15 in MIT occasional papers in linguistics. Distributed by MIT Working Papers in Linguistics, MIT, Dept. of Linguistics, 1998.

Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14(1396-1400):270, 1965.

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

K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

M. Collins. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, 2002.

M. Collins and J. Brooks. Prepositional phrase attachment through a backed-off model. In *Proceedings of the third workshop on very large corpora*, pages 27–38, 1995.

M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *The Journal of Machine Learning Research*, 9:1775–1822, 2008.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, March 2003. ISSN 1532-4435.

H. Cui, R. Sun, K. Li, M. Y. Kan, and T. S. Chua. 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, 2005.

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

M. De Marneffe and C. Manning. Stanford typed dependencies manual. 2008.

Y. Ding and M. Palmer. 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, 2005.

R. G. Downey and M. R. Fellows. Parameterized complexity, 1999.

J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71:233–240, 1967.

J. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, 1996.

J. Eisner. 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 2000. URL `http://cs.jhu.edu/~jason/papers/#iwptbook00`.

J. Eisner and G. Satta. 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, 1999.

J. Eisner and G. Satta. A faster parsing algorithm for Lexicalized Tree-Adjoining Grammars. In *Proceedings of the 5th Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+5)*, pages 14–19, 2000.

J. Eisner and N. A. Smith. Favor short dependencies: Parsing with soft and hard constraints on dependency length. In Harry Bunt, Paola Merlo, and Joakim Nivre, editors, *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, chapter 8, pages 121–150. Springer, 2010.

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

J. Evey. Application of pushdown-store machines. In *Proceedings of the November 12-14, 1963, fall joint computer conference*, AFIPS '63 (Fall), pages 215–227, 1963.

R. Fiengo. On trace theory. *Linguistic Inquiry*, 8(1):35–61, 1977.

T. Finley and T. Joachims. Training structural SVMs when exact inference is intractable. In *Proceedings of the 25th international conference on Machine learning*, pages 304–311. ACM, 2008.

H. Gaifman. Dependency systems and phrase-structure systems. *Information and control*, 8(3):304–337, 1965.

D. Gildea. Corpus variation and parser performance. In *Proceedings of EMNLP*, pages 167–202, 2001.

M. Goldberg. An unsupervised model for statistically determining coordinate phrase attachment. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 610–614. Association for Computational Linguistics, 1999.

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

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

L. Haegeman and H. Van Riemsdijk. Verb projection raising, scope, and the typology of rules affecting verbs. *Linguistic Inquiry*, 17(3):417–466, 1986.

D. Hindle and M. Rooth. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120, 1993.

B. Hoffman. *The Computational Analysis of the Syntax and Interpretation of "Free" Word Order in Turkish*. PhD thesis, University of Pennsylvania, 1995.

J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321455363.

L. Huang and K. Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*, pages 1077–1086, 2010.

X. Huang. Dealing with conjunctions in a machine translation environment. In *Proceedings of EACL*, pages 81–85, 1983.

R. Johansson and P. Nugues. Extended constituent-to-dependency conversion for English. In *Proc. of the 16th Nordic Conference on Computational Linguistics (NODALIDA)*, pages 105–112, 2007.

A. K. Joshi and Y. Schabes. Tree-adjoining grammars. *Handbook of formal languages*, 3: 69–124, 1997.

D. Jurafsky and J. H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics and speech recognition*. Prentice Hall, 2008.

R. S. Kayne. *The Antisymmetry of Syntax*. MIT Press, Cambridge, MA, 1994.

P. Koehn, F. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational*

Linguistics on Human Language Technology-Volume 1, pages 48–54. Association for Computational Linguistics, 2003.

T. Koo. *Advances in discriminative dependency parsing*. PhD thesis, Massachusetts Institute of Technology, 2010.

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

T. Koo, A. Globerson, X. Carreras, and M. Collins. Structured prediction models via the matrix-tree theorem. In *Proceedings of EMNLP-CoNLL*, 2007.

T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL*, pages 595–603, 2008.

T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP*, pages 1288–1298, 2010.

A. Kroch. Asymmetries in long distance extraction in a tree adjoining grammar. *Alternative conceptions of phrase structure*, pages 66–98, 1989.

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

M. Kuhlmann and J. Nivre. Mildly non-projective dependency structures. In *Proceedings of COLING/ACL*, pages 507–514, 2006.

A. Kulesza and F. Pereira. Structured learning with approximate inference. *Advances in neural information processing systems*, 20:785–792, 2007.

P. Liang, M. Jordan, and D. Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.

D. Lin, K. Church, H. Ji, S. Sekine, D. Yarowsky, S. Bergsma, K. Patil, E. Pitler, R. Lathbury, V. Rao, K. Dalwani, and S. Narsale. New tools for web-scale n-grams. In *Proceedings of LREC*, 2010.

M. P. Marcus. *Theory of Syntactic Recognition for Natural Languages*. MIT Press, 1980.

M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

A. F. T. Martins, N. A. Smith, and E. P. Xing. Concise integer linear programming formulations for dependency parsing. In *Proceedings of ACL*, pages 342–350, 2009.

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

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

R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98, 2005a.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 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, 2005b.

I. Mel'čuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, 1988.

J. Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, 2007a.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007b.

S. Petrov and D. Klein. Improved inference for unlexicalized parsing. In *Proceedings of NAACL*, pages 404–411, 2007.

E. Pitler. Attacking parsing bottlenecks with unlabeled data and relevant factorizations. In *Proceedings of ACL*, 2012.

E. Pitler, S. Bergsma, D. Lin, and K. Church. Using web-scale n-grams to improve base NP parsing performance. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 886–894, 2010.

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

E. Pitler, S. Kannan, and M. Marcus. Finding optimal 1-Endpoint-Crossing trees. *Transactions of the Association for Computational Linguistics*, 1:13–24, Mar 2013.

A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP*, pages 133–142, 1996.

A. Ratnaparkhi, J. Reynar, and S. Roukos. A maximum entropy model for prepositional phrase attachment. In *Proceedings of the workshop on Human Language Technology*, pages 250–255. Association for Computational Linguistics, 1994.

T. Reinhart. *The Syntactic Domain of Anaphora*. PhD thesis, Massachusetts Institute of Technology, 1976.

P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.

S. Riedel and J. Clarke. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 129–137. Association for Computational Linguistics, 2006.

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

J. Ross. *Infinite syntax*. Ablex Norwood, NJ:, 1986.

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

G. Satta and M. Kuhlmann. Efficient parsing for head-split dependency trees. *Transactions of the Association for Computational Linguistics*, 1:267–278, July 2013.

G. Satta and W. Schuler. Restrictions on tree adjoining languages. In *Proceedings of COLING-ACL*, pages 1176–1182, 1998.

L. Schwartz, T. Aikawa, and C. Quirk. Disambiguation of English PP attachment using multilingual aligned data. In *Proceedings of MT Summit IX*, 2003.

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

D. A. Smith and N. A. Smith. Probabilistic models of nonprojective dependency trees. In *Proceedings of EMNLP-CoNLL*, 2007.

R. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in neural information processing systems*, volume 16, 2003.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

D. Vadas and J. Curran. Adding noun phrase structure to the Penn Treebank. In *ACL*, pages 240–247, 2007.

K Vijay-Shanker. *A study of tree adjoining grammars*. PhD thesis, University of Pennsylvania, 1987.

R. Weischedel and A. Brunstein. BBN pronoun coreference and entity type corpus. *Linguistic Data Consortium, Philadelphia*, 2005.

R. Weischedel, M. Palmer, M. Marcus, E. Hovy, S. Pradhan, L. Ramshaw, N. Xue, A. Taylor, J. Kaufman, M. Franchini, et al. OntoNotes release 4.0. *LDC2011T03, Philadelphia, Penn.: Linguistic Data Consortium*, 2011.

D. Weiss and B. Taskar. Structured Prediction Cascades. In *AISTATS*, 2010.

H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of International Workshop of Parsing Technologies*, pages 195–206, 2003.

D. Zeman, D. Mareček, M. Popel, L. Ramasamy, J. Štěpánek, Z. Žabokrtský, and J. Hajič. HamleDT: To parse or not to parse? In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, 2012.

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

Y. Zhang and S. Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, pages 562–571, 2008.

G. Zhou, J. Zhao, K. Liu, and L. Cai. Exploiting web-derived selectional preference to improve statistical dependency parsing. In *Proceedings of ACL*, pages 1556–1565, 2011.