Advances in Discriminative Dependency Parsing

by

Terry Koo

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2010

Certified by .....
Michael Collins
Associate Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

Advances in Discriminative Dependency Parsing

by

Terry Koo

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Achieving a greater understanding of natural language syntax and parsing is a critical step in producing useful natural language processing systems. In this thesis, we focus on the formalism of dependency grammar as it allows one to model important head-modifier relationships with a minimum of extraneous structure. Recent research in dependency parsing has highlighted the discriminative structured prediction framework (McDonald et al., 2005a; Carreras, 2007; Suzuki et al., 2009), which is characterized by two advantages: first, the availability of powerful discriminative learning algorithms like log-linear and max-margin models (Lafferty et al., 2001; Taskar et al., 2003), and second, the ability to use arbitrarily-defined feature representations.

This thesis explores three advances in the field of discriminative dependency parsing. First, we show that the classic Matrix-Tree Theorem (Kirchhoff, 1847; Tutte, 1984) can be applied to the problem of non-projective dependency parsing, enabling both log-linear and max-margin parameter estimation in this setting. Second, we present novel third-order dependency parsing algorithms that extend the amount of context available to discriminative parsers while retaining computational complexity equivalent to existing second-order parsers. Finally, we describe a simple but effective method for augmenting the features of a dependency parser with information derived from standard clustering algorithms; our semi-supervised approach is able to deliver consistent benefits regardless of the amount of available training data.

Thesis Supervisor: Michael Collins
Title: Associate Professor

Acknowledgments

I thank my advisor Michael "Dr." Collins for his steadfast support and brilliant advice. He has been both mentor and friend for the past 7 years, and under his guidance I have developed from a clueless undergrad into a researcher who is ready to strike out on his own.

I also thank my committee, Regina Barzilay and Tommi Jaakkola, for their invaluable suggestions and comments during the process of formulating this thesis.

My gratitude goes out to my family as well for their love, patience, and encouragement during this lengthy process. They have been nothing short of phenomenal.

I offer a special thanks to my life partner Wen, who has provided me with unconditional love and support for several years and has thus far received nothing tangible in return, except this lousy dedication.

Finally, I would like to *sincerely* thank the Institvte for disbursing my fellowship funds and my advisor's grants to me in the form of Research Assistantships for the past few years, a service in exchange for which they have laid claim the copyright of this very thesis. Enjoy!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Achieving a greater understanding of natural language syntax and parsing is a critical step in producing useful natural language processing systems. High accuracy parsing is likely to be critical to many NLP applications, for example machine translation, question answering, information extraction, and text summarization. The recovery of syntactic structure can also be a useful preprocessing step for more complex forms of natural language analysis, such as semantic role labeling.

A useful abstraction for representing syntactic information is the *head-modifier dependency*, or dependency for short. A dependency is a directed relationship between two words: the *head*, which is the more important or essential word in the pair, and the *modifier*, which plays an auxiliary role and supplements the meaning of the head. Head-modifier dependencies provide a versatile, intuitive, and relatively uncontroversial—linguistically speaking—representation that is capable of capturing various important syntactic relationships. For example, in the sentence "John saw a great movie," we would place a dependency between the head "saw" and its modifier "movie" in order to indicate the verb-object relationship that exists between the two. As another example, we could also infer a dependency between "movie" and "great" to signify the fact that "great" specifies the quality of the "movie" in question; note that in this case "movie" would play the role of the head while "great" is its modifier.

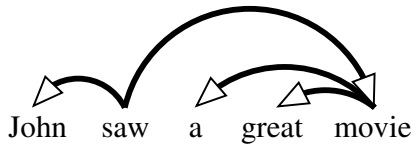


Figure 1-1: A dependency analysis for a simple sentence. Dependencies are depicted as directed arcs pointing from head to modifier, a convention that will be followed throughout this thesis. Note that the dependencies form a rooted, directed tree spanning the words of the sentence, with the root of the tree being the main verb "saw."

Besides being a convenient representation for syntactic relationships, head-modifier dependencies can also confer important practical benefits in parsing applications. For example, many of the most successful early approaches in statistical treebank parsing revolved around the modeling of head-modifier dependencies via the formalism of lexicalized context-free grammars (Collins, 1996, 1997, 1999; Charniak, 1997, 2000). However, these approaches required that a context-free parse tree be built in conjunction with the recovery of dependencies. In addition, these parsers must be trained on treebanks with rich phrase-structure annotations (Marcus et al., 1993) that may be unavailable or inappropriate for certain languages (Hajič et al., 2001; Hajič, 1998).

In this thesis, we focus instead on *dependency grammar*, which is a simple but powerful syntactic formalism that models head-modifier relationships with a minimum of extraneous structure. In dependency grammar, a complete syntactic analysis for a sentence is given by a dependency tree: a set of head-modifier dependencies that forms a rooted, directed tree spanning the words of the sentence. Figure 1-1 depicts the dependency tree associated with the simple example sentence used earlier. Note that dependency grammar does not attempt to model phrases or constituents, unlike the aforementioned parsing approaches built on lexicalized context-free grammars. As a result, dependency parsing algorithms are highly efficient (Eisner, 2000; Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005b) and moreover, the analyses produced by a dependency grammar are ideally suited for discriminative modeling with rich features due to their simple, surface-oriented nature (Yamada and Matsumoto, 2003;

McDonald et al., 2005a).

Recent research in discriminative dependency parsing has explored the framework of structured linear models with some encouraging success (McDonald et al., 2005a; Buchholz and Marsi, 2006; Nivre et al., 2007; Koo et al., 2008; Suzuki et al., 2009). We formalize the dependency parsing problem as a structured linear model as follows:

$$y^\star(\mathbf{x}) \;=\; \operatorname*{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}, y)$$

In brief, given a sentence $\mathbf{x}$, we compute its parse $y^\star(\mathbf{x})$ by searching for the highest-scored dependency tree in the set of compatible trees $\mathcal{Y}(\mathbf{x})$; scores are assigned using a linear model where $\mathbf{\Phi}(\mathbf{x}, y)$ is a feature-vector representation of the event that tree y is the analysis of sentence $\mathbf{x}$, and $\mathbf{w}$ is parameter vector containing associated weights. In general, performing a direct maximization over the set $\mathcal{Y}(\mathbf{x})$ is infeasible, and a common solution used in many parsing approaches is to introduce a part-wise factorization of the linear model:

$$\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}, y) \;=\; \sum_{p \in y} \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, p)$$

Above, we have assumed that the dependency tree y can be factored into a set of parts p, each of which represents a small substructure of y; for example, y might be factored into the set of its component dependencies. The parts are evaluated using a part-wise feature-vector mapping $\boldsymbol{\phi}(\mathbf{x}, p)$ that is only sensitive to one part at a time. The factorization thus establishes implicit independence restrictions between parts, which can be exploited to efficiently solve the combinatorial optimization problem involved in the search for the highest-scoring dependency tree.

In broad terms, a dependency parser in the factored structured linear modeling framework is defined by three main components:

1. A *feature mapping* $\boldsymbol{\phi}$ that provides a quantitative encoding of the sub-structures within a dependency tree. The features are the only source of information available to the parser so it is essential that $\boldsymbol{\phi}$ capture the types of information

   required for high-performance parsing.

2. A *parameter vector* $\mathbf{w}$ that quantifies the state of the parser by associating a real-valued weight with each available feature. Obtaining high-quality estimates of these parameters is clearly crucial, and there are many training algorithms that can be used to learn the parameters from an annotated dataset. Note, however, that some discriminative training methods may involve nontrivial computational problems that must be solved before training can take place.

3. A *factorization* that defines the method by which potential dependency analyses are decomposed within the parser. Note that the choice of factorization involves a tradeoff between complexity and expressiveness: factorizations that decompose each tree into small parts lead to fast parsing algorithms that impose strong independence restrictions, while factorizations with larger parts can capture greater context within the dependency tree at the cost of potentially greater computational complexity. The design of factorizations that are both expressive and efficiently parsable is thus of critical importance for practical parsing applications.

In this thesis, we treat each of these three components as an opportunity for increasing the performance of a discriminative dependency parser. The remainder of this document will present original work leading to three separate advances in the field of discriminative dependency parsing, each of which is aimed at improving one of the three components above.

1.1 Three Advances in Discriminative Dependency Parsing

Before we move on to the main body of the thesis, we first outline our three eponymous advances by describing the context in which each improvement occurs and summarizing the motivations and solutions involved in each situation.

1.1.1 Simple Semi-Supervised Feature Representations

Recent advances in natural language parsing have emphasized the use of statistical models. Often, the best-performing models have a heavy reliance on *lexicalized*, or word-based, features. For example, early successes in statistical treebank parsing employed lexicalized probabilistic context-free grammars (Collins, 1999; Charniak, 2000), which evaluate the probability of a parse tree using lexicalized distributions. More recently, the emphasis in statistical parsing research has fallen on applications of discriminative methods (McDonald, 2006; Finkel et al., 2008; Carreras et al., 2008). The discriminative nature of these parsing approaches allows them to define features that are arbitrarily predicated on the sentence being parsed, so that richly-lexicalized features can be defined with ease.

Although lexicalized features can be beneficial in parsing applications, they must be used with care due to statistical sparsity concerns. Thanks to the sheer breadth of the vocabulary and its long-tailed distribution, even a fairly simple set of lexicalized features is difficult to estimate reliably on any realistically-sized corpus. The issue of sparsity is further exacerbated by the fact that obtaining additional syntactically-annotated data is expensive, due to the highly-structured nature of syntactic analyses.

To provide a concrete illustration of the problem of sparsity, consider the case of a simple head-modifier bigram feature that tracks the identity of the head word and modifier word involved in some dependency. Figure 1-2 depicts the distribution of the frequencies, in English training data, of head-modifier bigrams that occur in the dependency annotations of the held-out English development set. It is easy to see that the majority of the head-modifier bigrams in the development set occur quite infrequently in training data. In fact, 50.44% of the head-modifier bigrams encountered in development data have appeared 5 times or less in the English training corpus, and 28.41% of the head-modifier bigrams in the development data have *never* been seen in training data. Features based on head-modifier bigrams would be useless or even misleading for low-frequency bigrams such as these. Note that the English training corpus contains nearly a million words of hand-annotated data, indicating

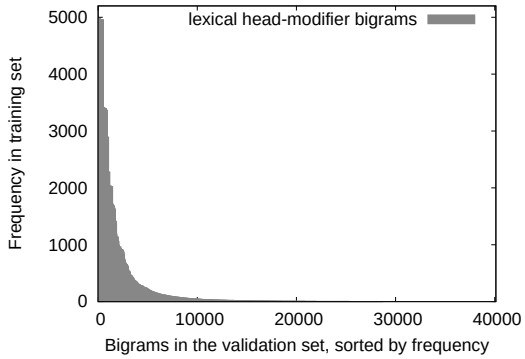


Figure 1-2: The frequency, in the English training corpus, of head-modifier bigrams encountered in the English held-out development corpus (Marcus et al., 1993). Specifically, for each head-modifier dependency that occurs in the annotated trees of the English development set, we examine the bigram of head and modifier words and count the number of times this bigram has occurred in the annotated dependencies of the English training data. The 40,117 development bigrams are sorted in order of decreasing frequency and the resulting frequencies are plotted above. For perspective, the training set contains 950,028 head-modifier dependencies.

the large scale of the dataset from which these statistics were drawn.

Generative parsers, like the lexicalized context-free grammars mentioned earlier (Collins, 1999; Charniak, 2000), combat statistical sparsity by employing various levels of backed-off distributions—e.g., conditioning on a part-of-speech tag or nonterminal label instead of a word—in order to obtain reliable probability estimates. In the discriminative setting, the analogous technique is to define multiple overlapping features involving various combinations of fine-grained information (e.g., words) and coarse-grained information (e.g., parts of speech). Note that both approaches are based on replacing words with coarse-grained *non*-lexical proxies; ideally, however, we would like to continue to leverage lexically-derived sources of information.

Considering the strengths and weaknesses of lexicalized features, it is attractive

to consider intermediate entities that are more coarse-grained than words, thereby alleviating the sparsity problems, while at the same time providing the lexical information that is essential for high performance parsing. Building on promising work by Miller et al. (2004), we consider the use of hierarchical word clusters derived from the Brown et al. (1992) clustering algorithm. Crucially, cluster-based information can easily be incorporated in the structured linear modeling framework by simply defining additional features predicated on cluster identities. We find that these cluster-based features provide a simple and reliable method for improving upon the performance of an otherwise standard dependency parsing approach.

1.1.2 Probabilistic Inference Algorithms for Non-Projective Dependency Parsing

One advantage of dependency grammar is that it provides a natural representation for *non-projective* structures, in which the nesting constraints inherent in ordinary syntactic structures are violated. An important recent breakthrough in non-projective parsing occurred when McDonald et al. (2005b) demonstrated that non-projective dependency trees are isomorphic to rooted, directed trees that span the words of the sentence being parsed. Thus, non-projective parsing can be accomplished by using the well-known Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) to find the maximum directed spanning tree of the input sentence. Note that previous research in non-projective parsing had generally focused on modified versions of projective parsing algorithms (Nivre and Nilsson, 2005, for example).

While the connection with directed spanning trees provided an elegant solution to the parsing problem, there were still some unresolved issues. In particular, there is a class of fundamental probabilistic inference algorithms that compute summations over sets of possible structures; these summations include quantities known as the *partition function* and *marginals*, which are useful for tasks like normalizing structured distributions or taking expectations with respect to distributions over structures. For example, in the case of hidden Markov models or conditional random fields (Laf-

ferty et al., 2001) for sequence labeling tasks, the well-known forward-backward algorithm (Baum et al., 1970) can be used to compute the relevant partition functions and marginals. As another common example, partition functions and marginals for context-free parse trees can be computed using the inside-outside algorithm (Baker, 1979).

Prior to our work, probabilistic inference algorithms of this kind had not been proposed for non-projective dependency parsing. Later in this thesis, we will show that efficient algorithms for computing partition functions and marginals can be obtained via applications of a classic result in combinatorics known as the Matrix-Tree Theorem (Kirchhoff, 1847). We demonstrate the empirical benefit of our new inference algorithms by using them to solve the computational problems involved in training log-linear non-projective parsers.

As an additional benefit, we show that the same inference algorithms can be used to train max-margin dependency parsers via dual exponentiated-gradient optimization (Taskar et al., 2003; Collins et al., 2008). Note that the dual exponentiated-gradient method has recently been proven to have a fast rate of convergence when training log-linear models; the proof of this fact is also included as part of this thesis.

1.1.3 Efficient Third-order Dependency Parsing Algorithms

In this thesis, we focus on parsing approaches which apply a *factorization* that breaks each dependency tree into a set of smaller *parts*. For many appropriate factorizations, it is possible to design efficient parsing algorithms that are capable of computing the highest-scoring dependency tree in polynomial time by exploiting the independence assumptions inherent in the part-wise factorization.

The most obvious type of factorization is a "first-order" factorization, in which each dependency tree is deconstructed into its individual dependencies; see Figure 1-3 for a depiction of this type of factorization. While this simple approach can obtain surprisingly high accuracy (see, e.g., McDonald et al., 2005a), it is often beneficial to employ "higher-order" factorizations, in which a dependency tree is decomposed into parts consisting of multiple dependencies. For example, recent work has demon-

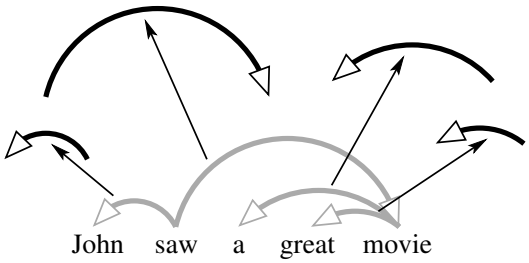


Figure 1-3: A simple dependency tree that has been decomposed into parts using a first-order factorization. A first-order parser would be able to score each of the dependencies independently of the others, and can then select the highest-scoring set of dependencies that results in a well-formed dependency tree. Efficient first-order parsers typically leverage either dynamic-programming techniques (Eisner, 2000) or spanning-tree-based algorithms (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005a).

strated that second-order parsers, which are able to evaluate pairs of dependencies, can consistently obtain substantial performance improvements over first-order parsers (McDonald and Pereira, 2006; Carreras, 2007; Koo et al., 2008; Suzuki et al., 2009).

In this thesis we will propose efficient new parsing algorithms for *third*-order factorizations, in which dependency trees are decomposed into parts consisting of three dependencies. Interestingly, by applying the proper dynamic-programming techniques our third-order parsing algorithms are able to achieve the same computational complexity as some commonly-used second-order parsers (Carreras, 2007). In an empirical evaluation, we show that our third-order parsers are able to provide further increases in performance over second-order parsers as well as results from previous work.

1.1.4 Contributions of This Thesis

Above, we have summarized the three advances in discriminative dependency parsing presented in this thesis. In the interests of providing a clear and concise delineation of the research encompassed by our three advances, we explicitly state the original

contributions of this thesis below.

1. A simple method for augmenting the features of any dependency parser by exploiting easily-obtained word clusters as coarse-grained lexical proxies.
2. New and efficient algorithms for computing partition functions and marginals for non-projective parsing, via applications of the Matrix-Tree Theorem (Kirchhoff, 1847).
3. Rate-of-convergence proofs for the dual exponentiated gradient algorithm that imply fast convergence when training log-linear models.
4. New and efficient dependency parsing algorithms that are able to utilize expressive third-order factorizations while retaining computational complexity equivalent to existing second-order parsers.

1.2 Outline of the Thesis

The remainder of this thesis is structured as follows.

Chapter 2 provides necessary background on dependency parsing, structured linear models, and discriminative parameter estimation methods.

Chapter 3 describes a semi-supervised method for improving the features of a standard dependency parser.

Chapter 4 describes probabilistic inference algorithms for distributions over non-projective dependency trees.

Chapter 5 presents our efficient third-order dependency parsing algorithms.

Chapter 6 concludes by summarizing the thesis and providing ideas for future work.

Chapter 2

Background

In this chapter, we introduce concepts and notation that will be used throughout this thesis.

2.1 Phrase-Structure Grammar

Although this thesis focuses on dependency parsing, a description of phrase-structure grammar offers a useful (and perhaps more familiar) point of comparison for dependency grammar. In addition, the importance of notions like head-modifier dependencies may be more clearly understood in the context of phrase-structure grammar.

Note that phrase-structure grammar is a well-studied formalism with a prolific literature, and a thorough presentation of the topic is well beyond the scope of this section (for a beginning see, e.g., Chomsky, 1956, 1969; Haegeman, 1991). Instead, we will present a highly simplified description, placing emphasis on details relevant to dependency grammar.

2.1.1 Phrase-Structure Trees

Phrase-structure grammar arranges sentences into a hierarchy of nested phrases. The question of what constitutes a phrase is a delicate matter and is a topic in itself, but for our purposes we define a phrase as a contiguous sequence of words that represents

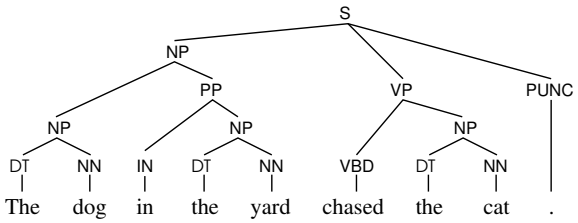


Figure 2-1: A phrase-structure tree for a simple sentence.

a cohesive unit of meaning. For example, "The dog" and "The dog in the yard" are both well-formed phrases.

At the lowest level of the tree, each word is treated as a one-word phrase that is labeled by its part-of-speech (POS) tag. At higher levels, successively larger phrases are created by concatenating smaller phrases, culminating in a phrase covering the entire sentence. For example, the phrase "The dog in the yard" is the composition of "The dog" and "in the yard". Figure 2-1 illustrates a phrase-structure parse of a simple sentence.

2.1.2 Head-Lexicalization and Head-Modifier Dependencies

Each phrase has a *head*, which can be loosely described as the core component of a phrase. For example, the head of the phrase "The dog in the yard" is the subphrase "The dog", and in turn the head of "The dog" is "dog". The siblings of the head-phrase are taken to be *modifiers* of the head. For example, "in the yard" is a modifier of "The dog", while at a lower level, "The" is a modifier of "dog". Note that modifiers in this sense are fairly intuitive: the phrase "in the yard" specifies the location of "The dog", just as "The" indicates that the dog in question is some contextually salient dog, in contrast to, e.g., "A dog" which would not refer to any particular dog.

Phrases are typically labeled for their heads: phrases like "The dog" and "the cat"

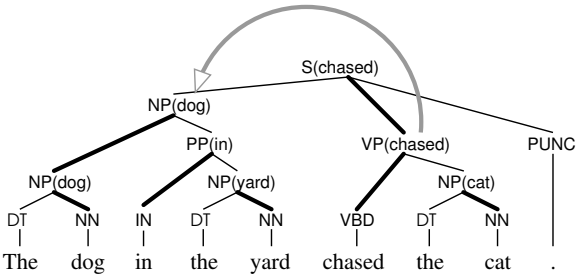


Figure 2-2: A lexicalized phrase-structure tree for a simple sentence. The head component of each phrase is indicated by a thick line. Headwords for the part-of-speech tags at the lowest level have been elided for clarity. The grey arc represents an example of a head-modifier dependency implied by the lexicalized phrase labels.

are noun phrases (NPs), phrases like "in the yard" are prepositional phrases (PPs), and phrases like "chased the cat" are verb phrases (VPs). This labeling practice is commonly extended by *lexicalizing* the grammar. Each phrase is assigned a *headword*, which is recursively defined as the headword of the phrase's head component, with the headword of each POS tag defined as the associated word. In a lexicalized grammar, the phrase "The dog in the yard" would then have the label NP(dog), while "chased the cat" would have the label VP(chased). Figure 2-2 illustrates a lexicalized version of the parse from Figure 2-1.

Head-modifier dependencies can be inferred between the headword of a head-phrase and the headwords of its modifiers. For example, the grey arc in Figure 2-2 illustrates a head-modifier dependency implied by the components of the sentential phrase S: the head of the dependency is "chased," which is the headword of the head sub-phrase VP, while the modifier is "dog," which is the headword of the modifier sub-phrase NP. These kinds of bilexical head-modifier dependencies, and headwords in general, have proven critical for the push toward high-performance phrase-structure parsing (see, e.g., Jelinek et al., 1994; Collins, 1996, 1999; Charniak, 2000).

Note that if all head-modifier dependencies are taken simultaneously, the result

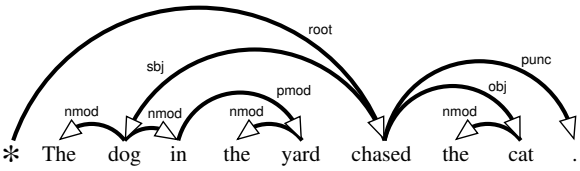


Figure 2-3: A dependency tree for a simple sentence. Dependencies are represented as arcs directed from head to modifier, with associated labels. The * symbol represents the abstract root of the sentence.

is a directed tree of dependencies that spans the words of the sentence. Dependency trees like these are the focus of dependency grammars, described next.

2.2 Dependency Grammar

Dependency grammar formalizes syntactic structure as a directed tree of head-modifier dependencies. Dependency grammar is less complex than lexicalized phrase-structure grammar, since head-modifier interactions are modeled directly without introducing the scaffold of phrase-structure grammar.

Figure 2-3 shows the dependency tree corresponding to the sentence in Figure 2-1. As is conventional (McDonald et al., 2005a), we augment each sentence with an *abstract root*, written as *. The abstract root is a token outside the sentence that serves as the predecessor of the sentence, similar to the null tokens used to model start and stop probabilities in HMMs.

Formally, a dependency analysis for some sentence is defined as a rooted, directed tree that spans the words of a sentence,[1] with the tree being rooted at the abstract root *. Any word that is a modifier of * is defined to be a *syntactic root*: a word that forms the syntactic core of the sentence. Under normal circumstances, the syntactic root would be the main verb of the sentence. The abstract root * is thus primarily a mathematical and notational convenience as it allows the selection of the syntactic root to be modeled as if it were a dependency like any other, while simultaneously providing a deterministic location for the root of the directed tree.

[1]Note that rooted, directed spanning trees are commonly referred to as *arborescences* in the combinatorics literature (see, e.g., Chu and Liu, 1965; Tutte, 1984).

In this thesis, we focus on a subtype of dependency grammar that is characterized by the additional constraint that modifiers on opposite sides of some head cannot interact with each other. This constraint is very similar to the independence properties intrinsic to split head-automaton dependency grammars (Eisner and Satta, 1999; Eisner, 2000), and is commonly assumed in a wide variety of dependency parsing approaches (McDonald et al., 2005a; McDonald and Pereira, 2006; Carreras, 2007). There are clear motivations for working with the constraint: left and right attachments can be performed separately, permitting the use of highly efficient parsing algorithms. As a concrete illustration, note that parsing algorithms for lexicalized context-free grammars can be used to parse dependency trees, but require $O(n^5)$ time in order to process an n-word sentence. In contrast, the efficient algorithm of Eisner (2000) can do so using only $O(n^3)$ time.

The remainder of this section first gives some further details about the relationship between phrase-structure grammar and dependency grammar, and then describes several important types of dependency trees.

2.2.1 Relationship to Phrase-Structure Grammar

Any phrase-structure tree can be converted to a dependency tree as long as the head component of each phrase is identified. Typically, the conversion is carried out semi-automatically, by applying a set of deterministic *head-rules* that provide a method for selecting the head component of each phrase (Collins, 1999; Yamada and Matsumoto, 2003).

This conversion process is not reversible since multiple phrase-structure trees can give rise to the same dependency tree; for example, Figure 2-4 depicts several phrase structure trees that correspond to the same dependency tree. Some of the reasons for this many-to-one relationship are that dependency-based representations conflate deep and flat phrase-structure trees, differences in the precedence of left and right modifiers, and so forth.

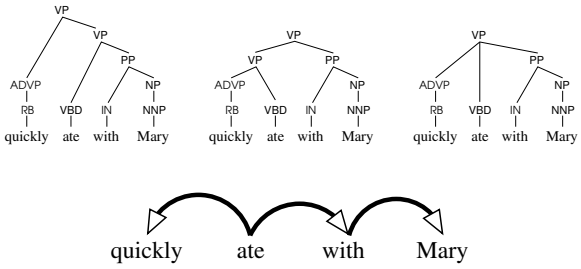


Figure 2-4: An example of several phrase-structure trees for the verb phrase "quickly ate with Mary." Three possible phrase-structure trees are shown above, all of which correspond to the single dependency tree below.

Interestingly, note that the one-way nature of the conversion provides a formal basis for the statement that dependency grammar is simpler than phrase-structure grammar. Although the loss of information might appear to be a disadvantage, in practice it is possible to define enriched dependency representations capable of reproducing almost all necessary details of phrase-structure grammars. As a concrete example, Carreras et al. (2008) describe a parser in which a modified form of dependency grammar is used to parse phrase structure trees with success. A key component of their approach is a heuristic method for converting their modified dependency parses into phrase-structure trees; while technically an imperfect conversion, the process is highly reliable in practice.

2.2.2 Labeled and Unlabeled Dependency Trees

Note that the dependency arcs in Figure 2-3 are annotated with labels representing different categories of head-modifier interactions, such as sbj and obj for verb-subject and verb-object interactions, respectively. Often, however, the dependency labels are omitted, in which case the dependency tree represents the bare head-modifier structure without specifying the types of interactions. We refer to dependency trees and dependency parsers as being *labeled* or *unlabeled* according to whether they use

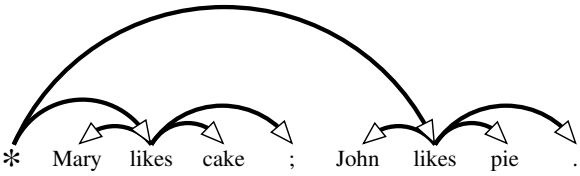


Figure 2-5: A multi-root dependency tree for a simple sentence.

or ignore dependency labels.

In the remainder of this document, we will generally place greater focus on unlabeled dependency parsing, for a number of reasons. First, unlabeled parsers are easier to describe and understand. In addition, an unlabeled parsing algorithm can usually be extended into a labeled algorithm through fairly obvious means.

There are also some important practical reasons for focusing on unlabeled parsing. First, the increased complexity of labeled parsing algorithms can make them more difficult to implement and verify. Second, labeled parsers generally have increased computational complexity as compared to their unlabeled counterparts; these computational penalties can be especially severe for higher-order parsers. Finally, while most dependency-based treebanks provide labeled annotations (Hajič et al., 2001, for example), there are situations where appropriate dependency labels can be difficult to obtain. For example, when converting a phrase-structure tree to a dependency tree, headword propagation provides a clearly-defined method for determining the bare structure of the dependency tree—i.e., determining the endpoints of each dependency. However, there is no acceptable method for producing the requisite dependency labels, and often, the labels for converted treebanks are assigned using ad hoc or task-specific methods: e.g., triples of nonterminals in phrase-structure-oriented contexts (Collins, 1999; Carreras et al., 2008), or hand-built categorizations in the context of plain dependency parsing (Koo et al., 2008, and also the hard-coded labelings of Joakim Nivre's Penn2MALT conversion tool).

As a final note, although the unlabeled parsing approach might seem to discard a

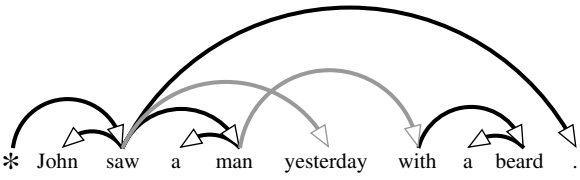


Figure 2-6: A dependency tree for a non-projective English sentence. The grey dependencies cross each other, indicating a violation of the nesting constraints assumed by a projective dependency tree.

good deal of information, in practice the dependency labels can be accurately recovered in a post-processing step (McDonald et al., 2006).

2.2.3 Single-root and Multi-root Dependency Trees

While in most cases, a proper dependency analysis should result in a tree with a single syntactic root, it is sometimes necessary for multiple roots to be proposed. Figure 2-5 shows two sentences joined by a semicolon; since there aren't any any head-modifier interactions between the two sentences, they are essentially independent and the dependency analysis therefore contains two separate trees. We refer to dependency trees as *single-root* when the abstract root * is restricted to have exactly one child, and *multi-root* when * may have one or more children; similar terminology is also used for parsers obeying these restrictions.

Independent, concatenated sentences like that shown in Figure 2-5 are uncommon in normal text, but multi-root parsing can also be useful for more pragmatic reasons. A multi-root parser is allowed to split up its analysis into multiple pieces, making it more robust to sentences that contain disconnected but coherent fragments. Such sentences can arise naturally—from disfluencies in the context of conversational speech—or artificially—from imperfect tokenization, text processing, or annotation.

2.2.4 Projective and Non-projective Dependency Trees

Up until this point, we have defined dependency structures as being equivalent to rooted, directed spanning trees. However, in many cases it is possible to impose an additional constraint by requiring that the trees be *projective*. Specifically, projectivity implies that for every head-modifier dependency, the words lying between the head and modifier—in the sequential order of the sentence—are descendents of the head. An equivalent and more convenient formulation of the projectivity constraint is in terms of *crossing dependencies*: if the dependencies are drawn as arcs above the words of the sentence (as they are universally depicted in the figures of this thesis), then a tree is projective if and only if none of its dependency arcs cross each other.

Figure 2-6 depicts an example of a non-projective tree; its crossing dependency is highlighted. We refer to dependency trees as being projective or non-projective according to whether they violate the projectivity constraint. Similarly, parsing algorithms and languages are referred to as projective or non-projective based on whether they enforce or relax the projectivity constraint, respectively.

Note that non-projectivity can be difficult to capture in a phrase-structure setting, due to the strictly nested nature of the structures. In particular, one interesting property of headword propagation in a phrase-structure tree is that the resulting dependency tree is always projective. Thus significant modifications are required in order to recover non-projective structures from the output of phrase-structure parsers. On the other hand, dependency grammar is able to capture both projective and non-projective structures in a single formalism. For languages like Czech and Dutch that contain a sizable degree of non-projectivity (Buchholz and Marsi, 2006), dependency grammar is therefore a very natural fit.

At the same time, many languages, such as Spanish and English, have a very low incidence of non-projective structures; in these situations, the projectivity constraint can be enforced with little cost in performance. Critically, by assuming projectivity it becomes possible to apply dynamic-programming techniques to the parsing problem (Eisner, 2000), including powerful higher-order parsers (McDonald and Pereira, 2006;

Carreras, 2007). Note that parsing with higher-order interactions is known to be NP-hard if non-projectivity is allowed (McDonald and Pereira, 2006; McDonald and Satta, 2007).

2.3 Discriminative Dependency Parsing

In this section, we formalize the dependency parsing task in the framework of structured linear models, following McDonald et al. (2005a).

2.3.1 Notational Conventions

Let $\mathbf{x}$ be a sentence drawn from $\mathcal{X}$, the set of all possible sentences. Assuming that $\mathbf{x}$ has $n = |\mathbf{x}|$ words, a dependency for that sentence is a tuple (h, m) where $h \in \{0, \ldots, n\}$ is the index of the head word in the sentence, 0 being the index of the abstract root *, and $m \in \{1, \ldots, n\}$ is the index of a modifier word. In the case of labeled dependencies, we augment the representation to (h, m, l), where $l \in \{1, \ldots, L\}$ is an index into a set of L possible labels. We define the notation $\mathcal{D}(\mathbf{x})$ to refer to all possible dependencies for a sentence $\mathbf{x}$:

$$\mathcal{D}(\mathbf{x}) \;=\; \Big\{(h, m) : h \in \{0, \ldots, n\}, m \in \{1, \ldots, n\}\Big\}.$$

Let $\mathcal{Y}$ be the set of all possible dependency structures. A dependency tree y corresponding to the sentence $\mathbf{x}$ is a set of n dependencies forming a directed tree rooted at index 0; we denote the subset of $\mathcal{Y}$ that spans $\mathbf{x}$ as $\mathcal{Y}(\mathbf{x})$. Note that the definition of $\mathcal{Y}(\mathbf{x})$ may change depending on whether or not we allow non-projective trees and multi-root trees. In the case that particular classes of dependency trees must be specified, we use the following notation: $\mathcal{Y}_p^s(\mathbf{x})$ denotes the set of all possible projective single-root trees spanning $\mathbf{x}$, and $\mathcal{Y}_{np}^s(\mathbf{x})$ denotes the set of all single-root non-projective trees for $\mathbf{x}$; the sets $\mathcal{Y}_p^m(\mathbf{x})$ and $\mathcal{Y}_{np}^m(\mathbf{x})$ are defined analogously for multi-root structures. For notational simplicity, we will use $\mathcal{Y}(\mathbf{x})$ when the category of tree is clear from the context, or in situations where any class of dependency trees

may be appropriate (e.g., in formulae that are equally applicable to all classes).

2.3.2 Structured Linear Models for Dependency Parsing

Formally, we treat dependency parsing as a structured linear model, as in McDonald et al. (2005a). In this framework, we represent parsing as a search for the highest-scoring structure where scores are assigned by a linear model:

$$y^\star(\mathbf{x};\mathbf{w}) \;=\; \operatorname*{argmax}_{y\in\mathcal{Y}(\mathbf{x})} \mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x},y) \tag{2.1}$$

In the above, $\mathbf{\Phi} : (\mathcal{X},\mathcal{Y}) \mapsto \mathbb{R}^d$ is defined so that $\mathbf{\Phi}(\mathbf{x},y)$ produces a d-dimensional vector representation of the event that dependency tree y is assigned to sentence $\mathbf{x}$. Each dimension in $\mathbf{\Phi}(\mathbf{x},y)$ is a *feature* that measures some quantifiable aspect of $\mathbf{x}$ and y; the vectors $\mathbf{\Phi}(\mathbf{x},y)$ are thus referred to as feature vectors. The following is an example of a 3-dimensional feature-vector representation:

$$\begin{aligned}
\Phi_1(\mathbf{x},y) &= \text{[Number of times the word ``dog'' modifies the word ``chased'']}\\
\Phi_2(\mathbf{x},y) &= \text{[Number of times the word ``cat'' modifies the word ``chased'']}\\
\Phi_3(\mathbf{x},y) &= \text{[Number of times the word ``likes'' modifies *]}
\end{aligned}$$

In this representation, the tree from Figure 2-3 would be mapped to the feature vector $\langle 1,1,0\rangle$, while the tree from Figure 2-5 would be represented by $\langle 0,0,2\rangle$. While the features above are quite simple, feature mappings can be defined arbitrarily and may include, e.g., interdependent or overlapping features, information derived from external sources like dictionaries, and so forth. The ability to exploit arbitrary features is a major advantage of structured linear models.

Returning to Eq. 2.1, $\mathbf{w}\in\mathbb{R}^d$ is referred to as a *parameter vector* containing d weights corresponding to the d separate features; these parameters are usually learned on a training corpus of examples $\{(\mathbf{x}_i,y_i)\}_{i=1}^n$, where each $(\mathbf{x}_i,y_i)$ pair provides a sentence and its gold-standard parse. The inner product $\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x},y)$ therefore evaluates to a cumulative score for the event that y is assigned to $\mathbf{x}$. Ideally, we would like $\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x},y)$ to be maximized for the correct dependency tree y_i, so that $y^\star(\mathbf{x};\mathbf{w})$

produces the proper output.

2.3.3 Factoring Structures for Efficient Parsing

A major weakness of the parser from Eq. 2.1 is that the maximization is performed over the set $\mathcal{Y}(\mathbf{x})$; as is often the case with structured models, the size of this set increases exponentially with the length of the sentence, making an explicit enumeration intractable. In order to perform the maximization efficiently, we assume that the dependency trees can be *factored* into smaller pieces.

Specifically, a factorization restricts the feature representation so that each feature is only sensitive to a limited region of y. Essentially, the factorization breaks each structure into sets of *parts*, which are local substructures of y with well-defined interactions. This restriction allows dynamic-programming methods or other algorithmic techniques to be applied in order to efficiently compute the maximization of Eq. 2.1. Formally, in the factored approach to dependency parsing, we restate the parsing problem as:

$$y^\star(\mathbf{x};\mathbf{w}) \;=\; \operatorname*{argmax}_{y\in\mathcal{Y}(\mathbf{x})} \sum_{p\in y} \mathbf{w}\cdot\boldsymbol{\phi}(\mathbf{x},p) \tag{2.2}$$

where y has been redefined as a set of factored parts that constitute a dependency tree, and $\boldsymbol{\phi}$ is defined as the part-wise local feature representation, which satisfies:

$$\mathbf{\Phi}(\mathbf{x},y) \;=\; \sum_{p\in y} \boldsymbol{\phi}(\mathbf{x},p)$$

Note that the arguments to $\boldsymbol{\phi}$ are $\mathbf{x}$ and p, indicating that $\boldsymbol{\phi}$ can only define features that are local to p; in contrast, the arguments to $\mathbf{\Phi}$ are $\mathbf{x}$ and y, so that $\mathbf{\Phi}$ has global scope over the entire dependency tree. Importantly, although $\boldsymbol{\phi}$ is local with respect to the dependency tree, $\boldsymbol{\phi}$ retains global scope over the input sentence $\mathbf{x}$ and can still make use of arbitrary external information sources.

The simplest type of factorization for dependency parsing is what we refer to as a first-order factorization, in which a given dependency tree y is broken into its n

component dependencies. In this case, the parsing problem would become:

$$y^\star(\mathbf{x};\mathbf{w}) \;=\; \operatorname*{argmax}_{y\in\mathcal{Y}(\mathbf{x})} \sum_{(h,m)\in y} \mathbf{w}\cdot\boldsymbol{\phi}(\mathbf{x},h,m) \tag{2.3}$$

where (h,m) represents the head and modifier indices of a dependency in y. In the case of a first-order factorization, the above parsing problem can be solved efficiently for projective parsing by applying dynamic-programming algorithms (Eisner, 2000). If non-projective trees are allowed, then dynamic-programming techniques cannot be applied but parsing can still be accomplished via efficient directed maximum spanning tree algorithms (McDonald et al., 2005b; Chu and Liu, 1965; Edmonds, 1967).

Dependency trees can also be factored into larger parts; we refer to such approaches as higher-order factorizations. In this case, instead of scoring each dependency independently as in a first-order parser, groups of two or more neighboring dependencies are scored as a whole. For example, one widely-used higher-order parser is that of McDonald and Pereira (2006), which includes "sibling" parts composed of a pair of dependencies with a shared head. Carreras (2007) defines a more complex factorization that includes both sibling parts and "grandparent" parts, which are composed of two dependencies arranged head-to-tail.[2] For both of these higher-order factorizations, efficient dynamic-programming algorithms exist for projective parsing. In the case of higher-order non-projective parsing, however, computational hardness results indicate that efficient parsing algorithms do not exist (McDonald and Pereira, 2006; McDonald and Satta, 2007).

2.4 Parameter Estimation Methods

In the previous section, we formalized dependency parsing as a factored structured linear model. Training a parser in this framework corresponds to estimating the parameter values associated with each feature—i.e., finding an estimate for $\mathbf{w}$. In this section, we describe some of the popular methods used for parameter estimation

[2] Naturally, there are additional subtleties involved in both factorizations that we will not explore here; we refer the reader to the relevant papers, as well as the material in Chapter 5.

Input: Training data $\{(\mathbf{x}_i,y_i)\}_{i=1}^n$
Number of iterations T

1. $\mathbf{w}=\mathbf{0}$ ◁ *initialize normal parameters*
2. $\mathbf{v}=\mathbf{0}$ ◁ *initialize summed parameters*
3. **for** $t=1\ldots T$ ◁ *for each iteration*
4. **for** $j=1\ldots n$ ◁ *repeat n times*
5. $i=\textsc{Random}[1,n]$ ◁ *choose a random example index*
6. $\hat{y}=y^\star(\mathbf{x}_i;\mathbf{w})$ ◁ *parse the sentence*
7. **if** $\hat{y}\neq y_i$ ◁ *was there a mistake?*
8. $\mathbf{w}=\mathbf{w}+\mathbf{\Phi}(\mathbf{x}_i,y_i)-\mathbf{\Phi}(\mathbf{x}_i,\hat{y})$ ◁ *update parameters*
9. **endif**
10. $\mathbf{v}=\mathbf{v}+\mathbf{w}$ ◁ *update summed parameters*
11. **endfor**
12. **endfor**
13. $\mathbf{v}=\frac{1}{Tn}\mathbf{v}$ ◁ *convert sum to average*

Output: Averaged parameters $\mathbf{v}$

Figure 2-7: Pseudocode for the structured perceptron with parameter averaging. Comments are indicated by ◁. Note that this version of the perceptron selects each example uniformly at random. However, a common alternative is to simply enumerate the dataset in sequential order—i.e., processing $(\mathbf{x}_1,y_1)$, then $(\mathbf{x}_2,y_2)$, and so on.

in structured linear models.

2.4.1 The Structured Perceptron

The first algorithm we consider is the structured perceptron, first introduced by Collins (2002). An extension of the classic perceptron (Rosenblatt, 1958) to structured data, the algorithm works by repeatedly parsing training examples and comparing the parsed output to the correct parses. The use of parameter averaging (Freund and Schapire, 1999) enables perceptron-trained parsers to achieve highly competitive levels of performance. Nevertheless, the perceptron is a simple and easy-to-understand algorithm.

The averaged perceptron begins with an initially zeroed parameter vector and proceeds in a series of T iterations, which are in turn divided into a series of trials.

Each trial involves selecting a random example from the training set, parsing that example, and checking the parser's prediction against the gold-standard structure. If the structures differ, then the parameters $\mathbf{w}$ are updated with the difference between the feature vectors of the gold standard and model prediction. The output of the algorithm is not the final parameter vector, but the average of all parameter vectors across every trial in the training run. Pseudocode for the algorithm is given in Figure 2-7.

First, note that the parameters are only updated in the case of a mistake. Furthermore, for factored feature representations, such as those which we use throughout this thesis, the difference computed in step 8 of the algorithm has an additional property: if the model prediction $\hat{y}$ is mostly correct, then only a few of its parts will differ from the parts in the gold standard y_i. Thus, the update performed on the parameter vector will only modify features pertaining to incorrect or missing parts.

The perceptron algorithm therefore exhibits a desirable quality: parameter sparsity. Consider that as the quality of the current parameter estimate $\mathbf{w}$ improves, the updates made to $\mathbf{w}$ become less frequent and less wide-ranging. In practice, this behavior results in a highly sparse parameter vector where only a fraction of the available parameters are non-zero. By using a sparse data structure to represent the parameter vector, it is possible to work with models that have massive numbers of features. For example, the parsers trained in Chapter 5 and Chapter 3 have feature dimensionalities ranging into the billions, but the characteristic sparsity of the structured perceptron allows these models to be trained efficiently.

The averaging of parameter vectors is crucial for obtaining best results with the perceptron algorithm. Often, the actual perceptron parameters $\mathbf{w}$ yield only mediocre parsing performance, while the averaged parameters $\mathbf{v}$ resulting from the same run are of much higher quality (Carreras, 2007, Table 2). We have observed throughout our experiments that averaged parameters obtain large performance improvements over the corresponding normal parameters; as a result, in all perceptron experiments we report results using the averaged parameters only.

Theoretical justifications for the perceptron algorithm generally come in two

forms. First, mistake bounds exist which demonstrate that the perceptron will converge on a linear separator—i.e., a set of parameters that obtains perfect accuracy on the training data—provided that the data is indeed separable. Collins (2002) proves a mistake bound for the structured perceptron; the proof is based on the classic mistake bound of Novikoff (1962). Second, the perceptron algorithm can be equated to performing stochastic sub-gradient descent on a convex objective function that is related to empirical error; stochastic gradient methods are widely used and there is a large body of work dedicated to their study (see, e.g., LeCun et al., 1998; Bottou, 2004; Zhang, 2004; Shalev-Shwartz et al., 2007). On the topic of parameter averaging, Freund and Schapire (1999) motivate averaging as a computationally-efficient approximation to a voting method, which they demonstrate to have desirable theoretical properties.

2.4.2 Log-Linear Models

Log-linear models are a popular parameter estimation method which define a conditional probability distribution over the possible parses for each sentence. The distribution is parameterized by the feature weights $\mathbf{w}$, which can then be estimated by fitting them to a dataset using standard optimization techniques.

In this thesis, we define a log-linear model as a structured, globally-normalized maximum-entropy model having the following form:

$$P(y \,|\, \mathbf{x}; \mathbf{w}) \;=\; \frac{1}{Z(\mathbf{x}; \mathbf{w})} \exp\big\{\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}, y)\big\}$$

where $Z(\mathbf{x}; \mathbf{w})$ is a normalization constant ensuring that the distribution sums to 1, sometimes referred to as the *partition function*:

$$Z(\mathbf{x}; \mathbf{w}) \;=\; \sum_{y \in \mathcal{Y}(\mathbf{x})} \exp\big\{\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}, y)\big\}$$

Conditional models of this form are also known as conditional random fields (CRFs), which were first introduced by Lafferty et al. (2001) in the context of sequence labeling. However, we prefer to use the term "log-linear model" as the parsing algorithms we work with are not naturally expressed as random fields (i.e., graphical models).

Parameter estimation for log-linear models generally revolve around optimization of a regularized conditional log-likelihood objective, though other alternatives exist (Smith et al., 2007, for example). The log-linear objective function is given below:

$$f_{\text{LL}}(\mathbf{w}) \;=\; \frac{C}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{n} \log P(y_i \,|\, \mathbf{x}_i; \mathbf{w}) \tag{2.4}$$

Note that this objective function has been phrased in terms of minimization. Thus, the negative conditional log-likelihood term encourages a close fit to the data, while the squared norm of the parameters discourages large parameter values. The tunable *regularization constant* C allows the balance between the two terms to be adjusted.

The f_{LL} objective function is smooth and convex, which is convenient for standard gradient-based optimization techniques. While earlier work with log-linear models focused on batch gradient descent methods like conjugate gradient descent and L-BFGS (Sha and Pereira, 2003), recent work has generally tended towards stochastic gradient descent (Smith and Smith, 2007; Finkel et al., 2008). New developments have also demonstrated that dual exponentiated gradient descent is a particularly effective optimization algorithm for log-linear models (see, e.g., Collins et al., 2008, or Appendix A). In general, these optimization techniques depend on the computation of difficult summations such as partition functions and part-wise marginal probabilities; these topics are discussed further in Chapter 4.

As probabilistic models, log-linear models have several advantages. First, they are capable of producing part-wise probability distributions, which can be exploited in highly effective coarse-to-fine pruning techniques like those developed by Carreras et al. (2008); similar pruning methods are also used in Chapter 5. Second, they facilitate the incorporation of additional probabilistic sources of information in a well-defined manner; for example, the combination of generative and discriminative probability models used in Suzuki et al. (2009).

2.4.3 Max-Margin Models

Another widely-used parameter estimation method for binary classification is the classic support vector machine (SVM) (Vapnik, 1995). A central concept in the binary SVM is the notion of *margin*, which measures the "width" of the separator defined by the parameter vector $\mathbf{w}$. There is a vast body of theoretical and empirical work that provides justification for the claim that parameter vectors obtaining large margins are desirable (see, e.g., Boser et al., 1992; Cortes and Vapnik, 1995; Vapnik, 1995, among many others).

Taskar et al. (2003) introduced an extension of the SVM to structured data; in this thesis, we refer to these extended SVMs as max-margin models. Note that Taskar-style max-margin models are by no means the only method for extending the SVM to structured data—see, e.g., Tsochantaridis et al. (2004) or Joachims et al. (2009) for alternatives. However, the Taskar et al. (2003) approach has the advantage of being particularly well-suited for the factored approaches to structured linear modeling that we explore in this thesis.

Central to the design of the max-margin model is an adaptation of the notion of margin to structured data. In the case of binary data, the margin is simple to measure since there are only two classes, but for structured data a large and in fact variable number of classes are possible for any given example. A simple approach might be to treat all incorrect parses equally, essentially requiring a constant margin between the correct parse y_i and all other parses $\hat{y}$; however, this approach ignores the internal structure of each incorrect tree. Taskar et al. (2003) instead distinguish between incorrect parses of varying severity, as quantified by an error function $\Delta(y_i, \hat{y})$ that computes, in a rough sense, the number of incorrect parts present in $\hat{y}$.[3] In essence, the max-margin model requires a variable margin between the correct parse y_i and all incorrect parses $\hat{y}$, where the desired amount of margin between y_i and $\hat{y}$ scales linearly with $\Delta(y_i, \hat{y})$.

Given some setting of the parameters $\mathbf{w}$, the degree to which the margin is violated

[3] A technical requirement is that $\Delta(y_i, y) = 0$ if $y = y_i$.

on some training example $(\mathbf{x}_i, y_i)$ can be calculated as

$$\xi(\mathbf{x}_i, y_i; \mathbf{w}) = \max_{y \in \mathcal{Y}(\mathbf{x}_i)} \left\{ \Delta(y_i, y) - \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}_i, y_i) + \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}_i, y) \right\}$$

This quantity is equivalent to the optimal amount of slack allocated to $(\mathbf{x}_i, y_i)$ in a soft-margin setting (Cortes and Vapnik, 1995), and is sometimes called the *hinge loss* in reference to the loss function characterizing the binary SVM. Note that ξ has the following properties: it is convex, being a maximization of linear terms, and satisfies $\xi(\mathbf{x}_i, y_i; \mathbf{w}) \geq 0$, with equality if the maximization is achieved by choosing $y = y_i$. In addition, ξ is non-differentiable, as the maximization of linear terms results in a polyhedral surface with sharp edges and corners.

The hinge loss can be computed efficiently as long as the error function $\Delta(y_i, \hat{y})$ can be decomposed into part-wise error components that follow the contours of the factorization used by the parser. We formalize this notion as follows:

$$\Delta(y_i, \hat{y}) = \sum_{\hat{p} \in \hat{y}} \delta(y_i, \hat{p})$$

Here, $\delta(y_i, \hat{p})$ is a part-wise function that computes the contribution to the error arising from the presence of part $\hat{p}$. By exploiting this decomposition, it is possible to compute $\xi(\mathbf{x}_i, y_i; \mathbf{w})$ by slightly modifying the scores provided to the parser; in brief, for every part p, the score of p is increased by $\delta(y_i, p)$.

Estimating the parameters in a max-margin model is accomplished by minimizing the following objective function:

$$f_{\text{MM}}(\mathbf{w}) = \frac{C}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^{n} \xi(\mathbf{x}_i, y_i; \mathbf{w}) \tag{2.5}$$

As with the log-linear objective function f_{LL}, this objective consists of a loss component that encourages a fit to the data—the summation of margin violations—and a regularizer that discourages large parameter values. The regularization constant C controls the balance between these two halves.

While the max-margin objective function f_{MM} is convex, the faceted nature of the hinge loss ξ renders it non-differentiable. Thus standard gradient-based optimization methods cannot be applied and most work in max-margin optimization focuses on the dual objective, which consists of a more conveniently-optimized quadratic form (see, e.g., Taskar et al., 2004; Bartlett et al., 2004; McDonald et al., 2005a; Koo et al., 2007; Collins et al., 2008). However, recent work in stochastic sub-gradient descent has suggested that primal optimization is a practical solution as well (Shalev-Shwartz et al., 2007).

2.5 Conclusion

In this chapter we have described dependency grammar, formalized the parsing problem as a factored structured linear model, and presented some common methods for parameter estimation. We conclude by re-framing the three eponymous advances in terms of the factored structured linear modeling framework. Below, we reproduce Eq. 2.2, deconstructing it into three main components:

$$y^\star(\mathbf{x}; \mathbf{w}) = \underbrace{\operatorname*{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \sum_{p \in y}}_{\text{factorization}} \quad \underbrace{\mathbf{w}}_{\text{parameters}} \cdot \underbrace{\boldsymbol{\phi}(\mathbf{x}, p)}_{\text{features}}$$

The leftmost component, which we refer to as the factorization, encompasses the method used to decompose each tree y into parts as well as the algorithms that enable efficient parsing and inference within this decomposition. The central component, the parameter vector $\mathbf{w}$, denotes the methods used estimate the parameters used in the parser. The final component, the features $\boldsymbol{\phi}$, indicates the the mapping used to represent each part within the parser. Each of these components represents a different opportunity for improvement, as we will see in the following chapters.

First, Chapter 3 presents a simple but effective method for improving the performance of a dependency parser by augmenting its feature mapping $\boldsymbol{\phi}$ with information derived from word clusters. As the word clusters are created from a large unlabeled text corpus, the augmented parsers thus constitute a semi-supervised learning approach. Empirical evaluations demonstrate that in a wide variety of situations and configurations, the parsers with augmented features can outperform their non-augmented counterparts.

Second, Chapter 4 presents new probabilistic inference algorithms that can efficiently compute certain important summations in the domain of non-projective parsing. These inference algorithms directly enable the use of log-linear models for the estimation of the parameter vector $\mathbf{w}$ and, in conjunction with exponentiated gradient optimization, also allow the use of max-margin models for parameter estimation. We demonstrate that the log-linear and max-margin parsers we are able to train can outperform a competitive baseline: the averaged perceptron.

Finally, Chapter 5 describes several novel dependency parsing algorithms, each of which is based on a new factorization of dependency trees y into higher-order parts p. In comparison to previous work on higher-order parsing, which has been largely based on second-order factorizations, the new parsing algorithms are able to use larger and more expressive third-order parts. We compare the new parsers against a series of highly competitive results from the literature and find that our enriched factorizations can improve upon previous work in supervised dependency parsing.

Chapter 3

Simple Semi-Supervised Dependency Parsing

Parts of this chapter are joint work with Xavier Carreras and Michael Collins, originally published in Koo et al. (2008).

We present a simple and effective semi-supervised method for training dependency parsers. We focus on the problem of lexical representation, introducing features that incorporate word clusters derived from a large unannotated corpus. We demonstrate the effectiveness of the approach in a series of dependency parsing experiments on the Penn Treebank and Prague Dependency Treebank, and we show that the cluster-based features yield substantial gains in performance across a wide range of conditions. For example, in the case of English unlabeled second-order parsing, we improve from a baseline accuracy of 92.02% to 93.16%, and in the case of Czech unlabeled second-order parsing, we improve from a baseline accuracy of 86.13% to 87.13%. In addition, we demonstrate that our method also improves performance when small amounts of training data are available, and can roughly halve the amount of supervised data required to reach a desired level of performance.

3.1 Introduction

In natural language parsing, lexical information is seen as crucial to resolving ambiguous relationships, yet lexicalized statistics are sparse and difficult to estimate directly. It is therefore attractive to consider intermediate entities which exist at a coarser level than the words themselves, yet capture the information necessary to resolve the relevant ambiguities.

In this chapter, we introduce lexical intermediaries via a simple two-stage semi-supervised approach. First, we use a large unannotated corpus to define word clusters, and then we use that clustering to construct a new cluster-based feature mapping for a discriminative learner. We are thus relying on the ability of discriminative learning methods to identify and exploit informative features while remaining agnostic as to the origin of such features. To demonstrate the effectiveness of our approach, we conduct experiments in dependency parsing, which has been the focus of much recent research—e.g., see work in the CoNLL shared tasks on dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007).

The idea of combining word clusters with discriminative learning has been previously explored by Miller et al. (2004), in the context of named-entity recognition, and their work directly inspired our research. However, our target task of dependency parsing involves more complex structured relationships than named-entity tagging; moreover, it is not at all clear that word clusters should have any relevance to syntactic structure. Nevertheless, our experiments demonstrate that word clusters can be quite effective in dependency parsing applications.

In general, semi-supervised learning can be motivated by two concerns: first, given a fixed amount of supervised data, we might wish to leverage additional unlabeled data to facilitate the utilization of the supervised corpus, increasing the performance of the model in absolute terms. Second, given a fixed target performance level, we might wish to use unlabeled data to reduce the amount of annotated data necessary to reach this target.

We show that our semi-supervised approach yields improvements for fixed datasets

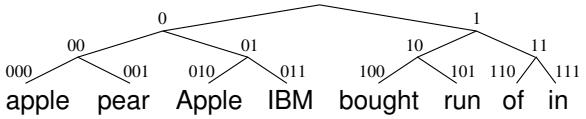


Figure 3-1: An example of a Brown word-cluster hierarchy. Each node in the tree is labeled with a bit-string indicating the path from the root node to that node, where 0 indicates a left branch and 1 indicates a right branch.

by performing parsing experiments on the Penn Treebank (Marcus et al., 1993) and Prague Dependency Treebank (Hajič et al., 2001; Hajič, 1998) (see Sections 3.4.1 and 3.4.3). By conducting experiments on datasets of varying sizes, we demonstrate that for fixed levels of performance, the cluster-based approach can reduce the need for supervised data by roughly half, which is a substantial savings in data-annotation costs (see Sections 3.4.2 and 3.4.4).

The remainder of this chapter is divided as follows: Section 3.2 gives background on word clustering, Section 3.3 describes the cluster-based features, Section 3.4 presents our experimental results, Section 3.6 discusses related work, and Section 3.7 concludes with ideas for future research.

3.2 Brown clustering algorithm

In order to provide word clusters for our experiments, we used the Brown clustering algorithm (Brown et al., 1992). We chose to work with the Brown algorithm due to its simplicity and prior success in other NLP applications (Miller et al., 2004; Liang, 2005). However, we expect that our approach can function with other clustering algorithms (as in, e.g., Li and McCallum, 2005). We briefly describe the Brown algorithm below.

The input to the algorithm is a vocabulary of words to be clustered and a corpus of text containing these words. Initially, each word in the vocabulary is considered to be in its own distinct cluster. The algorithm then repeatedly merges the pair of clusters which causes the smallest decrease in the likelihood of the text corpus, according to

a class-based bigram language model defined on the word clusters. By tracing the pairwise merge operations, one obtains a hierarchical clustering of the words, which can be represented as a binary tree as in Figure 3-1.

Within this tree, each word is uniquely identified by its path from the root, and this path can be compactly represented with a bit string, as in Figure 3-1. In order to obtain a clustering of the words, we select all nodes at a certain depth from the root of the hierarchy. For example, in Figure 3-1 we might select the four nodes at depth 2 from the root, yielding the clusters {apple,pear}, {Apple,IBM}, {bought,run}, and {of,in}. Note that the same clustering can be obtained by truncating each word's bit-string to a 2-bit prefix. By using prefixes of various lengths, we can produce clusterings of different granularities (Miller et al., 2004).

A straightforward implementation of the incremental pairwise mergings is impractical for realistic vocabulary sizes. Thus, the Brown et al. (1992) algorithm places a maximum on the number of possible clusters through the following heuristic procedure. First, the words of the vocabulary are arranged in order of decreasing frequency. Then, for each subsequent word, the algorithm creates a new cluster for that word; if the total number of clusters is greater than some user-defined constant C, then a pair of clusters is merged, reducing the total to C. After all words have been processed, the final stage of the algorithm merges the resulting C clusters into a binary hierarchy. The result is an algorithm that runs quickly in practice, while at the same time the early emphasis on high-frequency words results in a high-quality clustering. In our experiments, we set $C = 1000$ as the maximum number of clusters. We used the Liang (2005) implementation of the Brown algorithm to obtain the necessary word clusters.

3.3 Feature design

Key to the success of our approach is the use of features which allow word-cluster-based information to assist the parser. The feature sets we used are similar to other feature sets in the literature (McDonald et al., 2005a; Carreras, 2007), so we will

not attempt to give a exhaustive description of the features in this section. Rather, we describe our features at a high level and concentrate on our methodology and motivations. In our experiments, we employed two different feature sets: a baseline feature set which draws upon "normal" information sources such as word forms and parts of speech, and a cluster-based feature set that also uses information derived from the Brown cluster hierarchy.

3.3.1 Baseline features

Our first-order baseline feature set is similar to the feature set of McDonald et al. (2005a), and consists of indicator functions for combinations of words and parts of speech for the head and modifier of each dependency, as well as certain contextual tokens. We augment the McDonald et al. (2005a) feature set with backed-off versions of the "Surrounding Word POS Features" that include only one neighboring POS tag. We also add binned distance features which indicate whether the number of tokens between the head and modifier of a dependency is greater than 2, 5, 10, 20, 30, or 40 tokens.

Our second-order baseline features are the same as those of Carreras (2007) and include indicators for triples of part of speech tags for sibling interactions and grandparent interactions, as well as additional bigram features based on pairs of words involved these higher-order interactions. Examples of baseline features are provided in Table 3.1.

3.3.2 Cluster-based features

The first- and second-order cluster-based feature sets are supersets of the baseline feature sets: they include all of the baseline feature templates, and add an additional layer of features that incorporate word clusters. Following Miller et al. (2004), we use prefixes of the Brown cluster hierarchy to produce clusterings of varying granularity. We found that it was nontrivial to select the proper prefix lengths for the dependency parsing task; in particular, the prefix lengths used in the Miller et al. (2004) work

Baseline	Cluster-based
`ht,mt`	`hc4,mc4`
`hw,mw`	`hc6,mc6`
`hw,ht,mt`	`hc*,mc*`
`hw,ht,mw`	`hc4,mt`
`ht,mw,mt`	`ht,mc4`
`hw,mw,mt`	`hc6,mt`
`hw,ht,mw,mt`	`ht,mc6`
···	`hc4,mw`
	`hw,mc4`
	···
`ht,mt,st`	`hc4,mc4,sc4`
`ht,mt,gt`	`hc6,mc6,sc6`
···	`ht,mc4,sc4`
	`hc4,mc4,gc4`
	···

Table 3.1: Examples of baseline and cluster-based feature templates. Each entry represents a class of indicators for tuples of information. For example, "`ht,mt`" represents a class of indicator features with one feature for each possible combination of head POS-tag and modifier POS-tag. Abbreviations: `ht` = head POS, `hw` = head word, `hc4` = 4-bit prefix of head, `hc6` = 6-bit prefix of head, `hc*` = full bit string of head; `mt,mw,mc4,mc6,mc*` = likewise for modifier; `st,gt,sc4,gc4,...` = likewise for sibling and grandchild.

(between 12 and 20 bits) performed poorly in dependency parsing. One possible explanation is that the kinds of distinctions required in a named-entity recognition task (e.g., "Alice" versus "Intel") are much finer-grained than the kinds of distinctions relevant to syntax (e.g., "apple" versus "eat"). After experimenting with many different feature configurations, we eventually settled on a simple but effective methodology.

First, we found that it was helpful to employ two different types of word clusters:

1. Short bit-string prefixes (e.g., 4–6 bits), which we used as replacements for parts of speech.

2. Full bit strings, which we used as substitutes for word forms. Recall that the Brown et al. (1992) algorithm places a maximum, C, on the number of clusters present at any time during the procedure. In the Liang (2005) implementation, this results in a maximum of C clusters total, so that full bit strings are not equivalent to word forms.

Using these two types of clusters, we generated new features by mimicking the template structure of the original baseline features. For example, the baseline feature set includes indicators for word-to-word and tag-to-tag interactions between the head and modifier of a dependency. In the cluster-based feature set, we correspondingly introduce new indicators for interactions between pairs of short bit-string prefixes and pairs of full bit strings. Some examples of cluster-based features are given in Table 3.1.

Second, we found it useful to concentrate on "hybrid" features involving, e.g., one bit-string and one part of speech. In our initial attempts, we focused on features that used cluster information exclusively. While these cluster-only features provided some benefit, we found that adding hybrid features resulted in even greater improvements. One possible explanation is that the clusterings generated by the Brown algorithm can be noisy or only weakly relevant to syntax; thus, the clusters are best exploited when "anchored" to words or parts of speech.

Finally, we found it useful to impose a form of vocabulary restriction on the cluster-based features. Specifically, for any feature that is predicated on a word form, we eliminate this feature if the word in question is *not* one of the top-N most frequent words in the corpus. When N is between roughly 100 and 1,000, there is little effect on the performance of the cluster-based feature sets. For the experiments presented in this chapter, we used $N = 800$.

We hypothesize that, by eliminating the lower-frequency words, this restriction can encourage the discriminative learner to place more weight on the cluster-based features; the clusters thus serve as a kind of backed-off word form. More pragmatically, the vocabulary restriction reduces the size of the feature sets to manageable proportions. Interestingly, when the same vocabulary restriction is applied to the baseline features, which do not include cluster-based information, performance is substantially reduced, with increasing reductions as more words are discarded.

3.4 Experiments

In order to evaluate the effectiveness of the cluster-based feature sets, we conducted dependency parsing experiments in English and Czech. We test the features in a wide range of parsing configurations, including first-order and second-order parsers, and labeled and unlabeled parsers.

The English experiments were performed on the Penn Treebank (Marcus et al., 1993), using a standard set of head-selection rules (Yamada and Matsumoto, 2003) to convert the phrase structure syntax of the Treebank to a dependency tree representation. We used Joakim Nivre's Penn2Malt freely-available conversion tool[1]; dependency labels were obtained via the "Malt" hard-coded setting. We split the Treebank into a training set (Sections 2–21), a development set (Section 22), and several test sets (Sections 0,[2] 1, 23, and 24). The data partition and head rules were chosen to match previous work (Yamada and Matsumoto, 2003; McDonald et al., 2005a; McDonald and Pereira, 2006). The part of speech tags for the development and test data were automatically assigned by MXPOST (Ratnaparkhi, 1996), where the tagger was trained on the entire training corpus; to generate part of speech tags for the training data, we used 10-way cross-validation. English word clusters were derived from the BLLIP corpus (Charniak et al., 2000), which contains roughly 43 million words of Wall Street Journal text; we took measures to ensure that the sentences of the Penn Treebank were excluded from the text used for clustering.

The Czech experiments were performed on the Prague Dependency Treebank 1.0 (Hajič et al., 2001; Hajič, 1998), which is directly annotated with dependency structures. To facilitate comparisons with previous work (McDonald et al., 2005b; McDonald and Pereira, 2006), we used the training/development/test partition defined in the corpus and we also used the automatically-assigned part of speech tags provided in the corpus. Following Collins et al. (1999), we used a simplified version of the Czech part of speech tags, consisting of the first two characters of the morphological tag; this choice also matches the conditions of previous work (McDonald et al., 2005b;

[1] http://w3.msi.vxu.se/ nivre/research/Penn2Malt.html
[2] For computational reasons, we removed a single 249-word sentence from Section 0.

McDonald and Pereira, 2006). Czech word clusters were derived from the raw text section of the PDT 1.0, which contains about 39 million words of newswire text; as with the English datasets, this text was disjoint from the training and test corpora.

We trained the parsers using the averaged perceptron (Freund and Schapire, 1999; Collins, 2002), which represents a balance between strong performance and fast training times. To select the number of iterations of perceptron training, we performed up to 30 iterations and chose the iteration which optimized accuracy on the development set. Our feature mappings are quite high-dimensional, so we eliminated all features which occur only once in the training data. The resulting models still had very high dimensionality, ranging from tens of millions to as many as a billion features. Although these feature dimensionalities are quite high, the number of features which receive non-zero parameter values during perceptron training is only a small fraction of the feature dimensionality; we were thus able to train our parsers efficiently through the use of sparse data structures.

All results presented in this section are given in terms of parent-prediction accuracy, which measures the percentage of tokens that are attached to the correct head token. For labeled dependency structures, both the head token and dependency label must be correctly predicted. In addition, in English parsing we ignore the parent-predictions of punctuation tokens,[3] and in Czech parsing we retain the punctuation tokens; this matches previous work (Yamada and Matsumoto, 2003; McDonald et al., 2005a; McDonald and Pereira, 2006).

3.4.1 English main results

In our English experiments, we tested eight different parsing configurations, representing all possible choices between baseline or cluster-based feature sets, first-order (Eisner, 2000) or second-order (Carreras, 2007) factorizations, and labeled or unlabeled parsing.

Table 3.2 compiles our final test results and also includes two results from previous

[3] A punctuation token is any token whose gold-standard part of speech tag is one of {`` '' : , .}.

Sec	dep1	dep1c	MD1	dep2	dep2c	MD2
00	90.48	91.57 (+1.09)	—	91.76	92.77 (+1.01)	—
01	91.31	92.43 (+1.12)	—	92.46	93.34 (+0.88)	—
23	90.84	92.23 (+1.39)	90.9	92.02	93.16 (+1.14)	91.5
24	89.67	91.30 (+1.63)	—	90.92	91.85 (+0.93)	—

Sec	dep1-L	dep1c-L	—	dep2-L	dep2c-L	—
00	90.29	91.03 (+0.74)	—	91.33	92.09 (+0.76)	—
01	90.84	91.73 (+0.89)	—	91.94	92.65 (+0.71)	—
23	90.32	91.24 (+0.92)	—	91.38	92.14 (+0.76)	—
24	89.55	90.06 (+0.51)	—	90.42	91.18 (+0.76)	—

Table 3.2: Parent-prediction accuracies on Sections 0, 1, 23, and 24. Abbreviations: dep1/dep1c = first-order parser with baseline/cluster-based features; dep2/dep2c = second-order parser with baseline/cluster-based features; MD1 = McDonald et al. (2005a); MD2 = McDonald and Pereira (2006); suffix -L = labeled parser. Unlabeled parsers are scored using unlabeled parent predictions, and labeled parsers are scored using labeled parent predictions. Improvements of cluster-based features over baseline features are shown in parentheses.

work by McDonald et al. (2005a) and McDonald and Pereira (2006), for the purposes of comparison. We note a few small differences between our parsers and the parsers evaluated in this previous work. First, the MD1 and MD2 parsers were trained via the MIRA algorithm (Crammer and Singer, 2003; Crammer et al., 2004), while we use the averaged perceptron. In addition, the MD2 model uses only sibling interactions, whereas the dep2/dep2c parsers include both sibling and grandparent interactions.

There are some clear trends in the results of Table 3.2. First, performance increases with the order of the parser: edge-factored models (dep1 and MD1) have the lowest performance, adding sibling relationships (MD2) increases performance, and adding grandparent relationships (dep2) yields even better accuracies. Similar observations regarding the effect of model order have also been made by Carreras (2007).

Second, note that the parsers using cluster-based feature sets consistently outperform the models using the baseline features, regardless of model order or label usage. Some of these improvements can be quite large; for example, a first-order model using cluster-based features generally performs as well as a second-order model using base-

Tagger always trained on full Treebank

Size	dep1	dep1c	Δ	dep2	dep2c	Δ
1k	84.54	85.90	1.36	86.29	87.47	1.18
2k	86.20	87.65	1.45	87.67	88.88	1.21
4k	87.79	89.15	1.36	89.22	90.46	1.24
8k	88.92	90.22	1.30	90.62	91.55	0.93
16k	90.00	91.27	1.27	91.27	92.39	1.12
32k	90.74	92.18	1.44	92.05	93.36	1.31
All	90.89	92.33	1.44	92.42	93.30	0.88

Tagger trained on reduced dataset

Size	dep1	dep1c	Δ	dep2	dep2c	Δ
1k	80.49	84.06	3.57	81.95	85.33	3.38
2k	83.47	86.04	2.57	85.02	87.54	2.52
4k	86.53	88.39	1.86	87.88	89.67	1.79
8k	88.25	89.94	1.69	89.71	91.37	1.66
16k	89.66	91.03	1.37	91.14	92.22	1.08
32k	90.78	92.12	1.34	92.09	93.21	1.12
All	90.89	92.33	1.44	92.42	93.30	0.88

Table 3.3: Parent-prediction accuracies of unlabeled English parsers on Section 22. Abbreviations: Size = #sentences in training corpus; Δ = difference between cluster-based and baseline features; other abbreviations are as in Table 3.2.

line features. Moreover, the benefits of cluster-based feature sets combine additively with the gains of increasing model order. For example, consider the unlabeled parsers in Table 3.2: on Section 23, increasing the model order from dep1 to dep2 results in a relative reduction in error of roughly 13%, while introducing cluster-based features from dep2 to dep2c yields an additional relative error reduction of roughly 14%. As a final note, all 16 comparisons between cluster-based features and baseline features shown in Table 3.2 are statistically significant.[4]

3.4.2 English learning curves

We performed additional experiments to evaluate the effect of the cluster-based features as the amount of training data is varied. Note that the dependency parsers we use require the input to be tagged with parts of speech; thus the quality of the

[4] We used the sign test at the sentence level. The comparison between dep1-L and dep1c-L is significant at $p < 0.05$, and all other comparisons are significant at $p < 0.0005$.

part-of-speech tagger can have a strong effect on the performance of the parser. In these experiments, we consider two possible scenarios:

1. The tagger has a large training corpus, while the parser has a smaller training corpus. This scenario can arise when tagged data is cheaper to obtain than syntactically-annotated data.

2. The same amount of labeled data is available for training both tagger and parser.

Table 3.3 displays the accuracy of first- and second-order models when trained on smaller portions of the Treebank, in both scenarios described above. Note that the cluster-based features obtain consistent gains regardless of the size of the training set. When the tagger is trained on the reduced-size datasets, the gains of cluster-based features are more pronounced, but substantial improvements are obtained even when the tagger is accurate.

It is interesting to consider the amount by which cluster-based features reduce the need for supervised data, given a desired level of accuracy. Based on Table 3.3, we can extrapolate that cluster-based features reduce the need for supervised data by roughly a factor of 2. For example, the performance of the dep1c and dep2c models trained on 1k sentences is roughly the same as the performance of the dep1 and dep2 models, respectively, trained on 2k sentences. This approximate data-halving effect can be observed throughout the results in Table 3.3.

When combining the effects of model order and cluster-based features, the reductions in the amount of supervised data required are even larger. For example, in scenario 1 the dep2c model trained on 1k sentences is close in performance to the dep1 model trained on 4k sentences, and the dep2c model trained on 4k sentences is close to the dep1 model trained on the entire training set (roughly 40k sentences).

3.4.3 Czech main results

In our Czech experiments, we considered only unlabeled parsing,[5] leaving four different parsing configurations: baseline or cluster-based features and first-order or

[5] We leave labeled parsing experiments to future work.

dep1	dep1c	dep2	dep2c
84.49	86.07 (+1.58)	86.13	87.13 (+1.00)

Table 3.4: Parent-prediction accuracies of unlabeled Czech parsers on the PDT 1.0 test set, for baseline features and cluster-based features. Abbreviations are as in Table 3.2.

Parser	Accuracy
Nivre and Nilsson (2005)	80.1
McDonald et al. (2005b)	84.4
Hall and Novák (2005)	85.1
McDonald and Pereira (2006)	85.2
dep1c	86.07
dep2c	87.13

Table 3.5: Unlabeled parent-prediction accuracies of Czech parsers on the PDT 1.0 test set, for our models and for previous work.

second-order parsing. Note that our feature sets were originally tuned for English parsing, and except for the use of Czech clusters, we made no attempt to retune our features for Czech.

Czech dependency structures may contain non-projective edges, so we employ a maximum directed spanning tree algorithm (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005b) as our first-order parser for Czech. For the second-order parsing experiments, we used the Carreras (2007) parser. Since this parser only considers projective dependency structures, we "projectivized" the PDT 1.0 training set by finding, for each sentence, the projective tree which retains the most correct dependencies; our second-order parsers were then trained with respect to these projective trees. The development and test sets were *not* projectivized, so our second-order parser is guaranteed to make errors in test sentences containing non-projective dependencies. To overcome this, McDonald and Pereira (2006) use a two-stage approximate decoding process in which the output of their second-order parser is "deprojectivized" via greedy search. For simplicity, we did not implement a deprojectivization stage on top of our second-order parser, but we conjecture that such techniques may yield some additional performance gains; [6] we leave this to future work.

[6] See, e.g., McDonald and Pereira (2006, Table 2).

Size	dep1	dep1c	Δ	dep2	dep2c	Δ
1k	72.79	73.66	0.87	74.35	74.63	0.28
2k	74.92	76.23	1.31	76.63	77.60	0.97
4k	76.87	78.14	1.27	78.34	79.34	1.00
8k	78.17	79.83	1.66	79.82	80.98	1.16
16k	80.60	82.44	1.84	82.53	83.69	1.16
32k	82.85	84.65	1.80	84.66	85.81	1.15
64k	84.20	85.98	1.78	86.01	87.11	1.10
All	84.36	86.09	1.73	86.09	87.26	1.17

Table 3.6: Parent-prediction accuracies of unlabeled Czech parsers on the PDT 1.0 development set. Abbreviations are as in Table 3.3.

Table 3.4 gives accuracy results on the PDT 1.0 test set for our unlabeled parsers. As in the English experiments, there are clear trends in the results: parsers using cluster-based features outperform parsers using baseline features, and second-order parsers outperform first-order parsers. Both of the comparisons between cluster-based and baseline features in Table 3.4 are statistically significant.[7] Table 3.5 compares accuracy results on the PDT 1.0 test set for our parsers and several other recent papers.

3.4.4 Czech learning curves

As in our English experiments, we performed additional experiments on reduced sections of the PDT; the results are shown in Table 3.6. For simplicity, we did not retrain a tagger for each reduced dataset, so we always use the (automatically-assigned) part of speech tags provided in the corpus. Note that the cluster-based features obtain improvements at all training set sizes, with data-reduction factors similar to those observed in English. For example, the dep1c model trained on 4k sentences is roughly as good as the dep1 model trained on 8k sentences.

3.4.5 Additional results

Here, we present three additional results that are aimed at characterizing the behavior of the cluster-based feature sets. In Table 3.7, we show the development-set

[7] We used the sign test at the sentence level; both comparisons are significant at $p < 0.0005$.

N	dep1	dep1c	dep2	dep2c
100	89.19	92.25	90.61	93.14
200	90.03	92.26	91.35	93.18
400	90.31	92.32	91.72	93.20
800	90.62	92.33	91.89	93.30
1600	90.87	—	92.20	—
All	90.89	—	92.42	—

Table 3.7: Parent-prediction accuracies of unlabeled English parsers on Section 22. Abbreviations: N = threshold value; other abbreviations are as in Table 3.2. We did not train cluster-based parsers using threshold values larger than 800 due to computational limitations.

dep1-P	dep1c-P	dep1	dep2-P	dep2c-P	dep2
77.19	90.69	90.89	86.73	91.84	92.42

Table 3.8: Parent-prediction accuracies of unlabeled English parsers on Section 22. Abbreviations: suffix -P = model without POS; other abbreviations are as in Table 3.2.

performance of second-order parsers as the threshold for lexical feature elimination (see Section 3.3.2) is varied. Note that the performance of cluster-based features is fairly insensitive to the threshold value, whereas the performance of baseline features clearly degrades as the vocabulary size is reduced.

In Table 3.8, we show the development-set performance of the first- and second-order parsers when features containing part-of-speech-based information are eliminated. Note that the performance obtained by using clusters *without* parts of speech is close to the performance of the baseline features.

In Figure 3-2, we plot the frequency, in the English training set, of head-modifier bigrams occurring in the English development set. Note that the frequency distribution of lexical bigrams (darkest grey) has previously been illustrated in Figure 1-2. However, Figure 3-2 also depicts analogous frequency curves for bigrams of 6-bit cluster prefixes (lightest grey) and 8-bit cluster prefixes (medium grey). Note that the cluster bigrams, in addition to being more frequent overall, exhibit a much more robust tail that decays toward zero at a far slower rate. While 28.41% of the lexical bigrams occurring in development data have never been seen in training data, only 0.91% of the 6-bit bigrams and 1.19% of the 8-bit bigrams are similarly unseen. Based

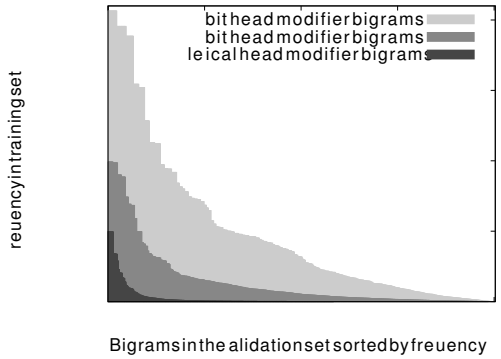


Figure 3-2: The frequency, in the English training corpus, of head-modifier word bigrams and cluster prefix bigrams encountered in the English held-out development corpus (Marcus et al., 1993). Specifically, for each head-modifier dependency that occurs in the annotated trees of the English development set, we examine the bigram of head and modifier words and count the number of times that this bigram has occurred in the annotated dependencies of the English training data. In addition, we also map the word bigrams to bigrams of 6-bit and 8-bit cluster prefixes and report the number of times each cluster-based bigram occurring in development data occurs in the training set. The 40,117 development bigrams are sorted in order of decreasing frequency—with each type of bigram being sorted independently—and the resulting frequencies are plotted above. For perspective, the training set contains 950,028 head-modifier dependencies.

on Figure 3-2 it should be clear that the cluster-based feature sets are far easier to estimate than their baseline counterparts.

3.5 Alternate Clusterings

In our experiments, we have focused on the clusters of the Brown et al. (1992) algorithm due to their successful application in previous work (Miller et al., 2004). However, there are a wide variety of alternative methods for producing word clusters, some of which we discuss in this section.

3.5.1 Split-Merge Hidden Markov Model Clustering

One alternative clustering method that we explored was the use of hidden Markov models. There are standard and well-known methods for learning these clusterings on unlabeled data using the expectation-maximization (EM) algorithm (Baum et al., 1970). Unfortunately, EM training is only guaranteed to find a local optimum of the likelihood objective and can easily produce models of widely-varying quality based on the initialization of the EM iterations. In addition, we would like to retain the beneficial hierarchical nature of the Brown clustering, which allows word clusters at multiple granularities to be simultaneously employed. In an effort to avoid low-quality local optima while producing a hierarchical clustering, we adapted the heuristic split-merge EM training method proposed by Petrov et al. (2006), which involves two alternating steps:

Split: Each state of the hidden Markov model is divided into two states, whose parameters are initialized with the parameters of the parent state plus some small amount of random noise to break symmetry. Standard EM training is then used to re-estimate the parameters of the split model. Note that the question of initialization is conveniently addressed in this step by utilizing the parameters of the unsplit model.

Merge: Each of the pairs of states that were split in the previous step is evaluated using a heuristic that approximates the loss in likelihood that would occur if the pair were re-merged into a single state. After considering all pairs of recently-split states, the least useful half of the splits are then reversed; specifically, the splits are sorted according to the approximate loss in likelihood—as determined by the heuristic—and the half of the pairs with the smallest predicted losses are merged together again. These merging steps are crucial as they allow the training algorithm to prune away uninformative splits that might otherwise clutter the hierarchy. Note that the original split-merge method of Petrov et al. (2006) was used to learn latent annotations for a context-free grammar and the merging heuristic they proposed was based on the inside and outside summa-

tions computed by the inside-outside algorithm (Baker, 1979). As this heuristic was not directly applicable in our setting, we substituted an analogous heuristic based on forward and backward summations taken from the forward-backward algorithm (Baum et al., 1970).

We initialized the split-merge process by estimating a 2-state hidden Markov model via EM training on the unlabeled data, and subsequent split-merge iterations were then applied as described above. Thus, a series of increasingly fine-grained hidden Markov models is learned from the unlabeled dataset. Note that the split-merge process produces a binary hierarchy that can be immediately applied in the setting of our cluster-based feature sets.

Unfortunately, our preliminary experiments along this line did not result in useful or reasonable clusterings. One possible explanation for the poor quality of the clusterings is that the initial 2-state model was incapable of producing a pair of states from which meaningful splits could be derived. Note that the original Petrov et al. (2006) split-merge approach was applied to the task of refining an existing treebank grammar by adding latent annotations; thus, their base model already contains significant structure which helps to guide the subsequent EM training phases. Consequently, one method that we may explore in future work might be to apply the split-merge iterations to a hidden Markov model whose initial structure is defined by a part-of-speech tagger estimated from labeled data. Another possibility might be to create an initial model based on the first few levels of the Brown cluster hierarchy; for example, we could begin by using 4-bit prefixes of the Brown clusters to estimate the parameters of a 16-state hidden Markov model.

As a side note, an interesting advantage of these hidden Markov model clustering approaches is that the word clusters are "soft," or context-dependent—i.e., the same word can receive different clusters in different situations. Soft clusterings have also been explored in previous work by Pereira et al. (1993). In contrast, the Brown algorithm produces a "hard" clustering in which every word is assigned to exactly one cluster, regardless of context. On the other hand, context-sensitivity may not be as importance as one might expect, due to the property that within any given domain

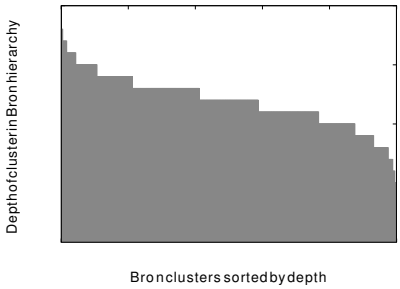


Figure 3-3: The depths of each of the 1,000 Brown clusters in the hierarchy, for clusters derived from English unlabeled data (Charniak et al., 2000). Specifically, for each cluster we compute its depth in the Brown hierarchy—i.e., the length of its bit-string—and plot the depth values in sorted order above. While most clusters are between 11 and 13 levels deep (55.5% of the clusters lie within this range of depths, to be exact), the depths range from 18 at the deepest to 4 at the shallowest.

a word will usually have only one usage (Gale et al., 1992). For example, the word "rose" could refer to a noun (the flower) or a past tense verb (moved upward), but in the context of Wall Street Journal text (Marcus et al., 1993) the word is almost exclusively used as a verb (e.g., "markets rose yesterday").

3.5.2 Methods for Truncating the Cluster Hierarchy

In the experiments described in this chapter, we followed the practice of Miller et al. (2004) and used simple fixed-length prefixes of the cluster bit-strings to define word clusterings of various granularities. In terms of the cluster hierarchy, fixed-length prefixes correspond to truncating the hierarchy at a fixed distance from the root—e.g., taking 4-bit prefixes corresponds to using only the first 4 levels of the hierarchy.

The use of prefixes in this manner is thus rather *ad hoc* and a valid question is whether there are more principled methods for determining the points at which to truncate the hierarchy. Note that the Brown et al. (1992) algorithm does not make any special effort to produce a "balanced" hierarchy—i.e., a hierarchy in which the

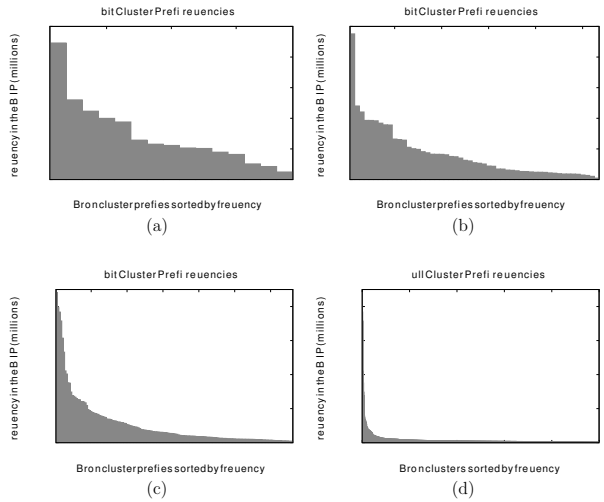


Figure 3-4: The frequency of each of the Brown clusters in the unlabeled corpus, for clusters derived from English unlabeled data (Charniak et al., 2000). Specifically, for each cluster we measure the number of times it occurs in the unlabeled corpus from which it was derived, sort the frequencies in descending order, and plot them above. We performed this for 4-bit cluster prefixes (a), 6-bit prefixes (b), 8-bit prefixes (c), and whole clusters (d). Note that in all cases, the distribution of frequencies is highly uneven. In addition, while there are a total of 16 4-bit prefixes, there are only 54 6-bit prefixes and 166 8-bit prefixes, due to the fact that some parts of the hierarchy do not extend as deeply as others (see Figure 3-3).

leaves are situated at roughly even depths and where every branch has approximately the same population. Thus, there is no reason to expect that cutting the hierarchy at a fixed depth should produce a reasonable clustering; in contrast, consider the superficially similar technique of Huffman (1952) coding, which defines a binary hierarchy of clusters that exhibits provably optimal balance (note, however, that these codes are assigned with the specific goal of optimizing the balance of the hierarchical clustering).

In fact, it is easy to demonstrate that the Brown clusterings we used in our parsing experiments are highly imbalanced: Figure 3-3 shows that the depths of the leaves in our English clustering were quite uneven, while Figure 3-4 clearly shows that the frequencies of the clusters vary widely at all granularities. Although we have already demonstrated that simple fixed-length prefixes can achieve large increases in parsing performance, it may still be profitable to implement techniques that can adaptively choose to use shorter or longer prefixes at various points in the hierarchy, in an effort to delineate a more balanced contour within the hierarchy. Such methods could easily be implemented by, e.g., exploring successively lower levels of the cluster hierarchy until some population threshold is reached. Moving beyond these simple heuristics, recall that the Brown et al. (1992) algorithm is based on an approximate maximization of the average mutual information between clusters that are sequentially adjacent in the text corpus. Therefore, an attractive prospect for future work might be the development of principled information-theoretic methods for selecting truncation points within the hierarchy.

3.5.3 Syntax-Based Clusterings

Along a different vein, it would be interesting to investigate clusterings derived from syntactic context rather than sequential context. While this idea has been explored in previous work, such approaches have generally been based on splitting existing coarse-grained entities like part-of-speech tags or nonterminal labels (Matsuzaki et al., 2005; Koo and Collins, 2005; Petrov et al., 2006; Finkel et al., 2007). In contrast to this previous work, we propose syntax-based clustering methods which are entirely lexical. For example, it may be possible to modify the Brown et al. (1992) algorithm so that the agglomerative merging process is predicated on mutual information with respect to head-modifier bigram context as opposed to sequential bigram context. Such an approach would have similarities to previous work in word clustering based on distributional similarities in syntactic contexts (Pereira et al., 1993).

The advantage of syntax-based clustering approaches is that the resulting word clusters would reflect syntactic relationships, presumably increasing their utility within

parsing applications. On the other hand, part-of-speech categories already capture a great deal of information about the syntactic role of each word, which may overlap with the informational content of syntax-based clusters. In addition, any syntax-based clustering approach would require data with syntactic annotations, which would either need to be created by human annotators or automatically assigned by a parser. In the case of the former, the expensive nature of human annotation would lead to a scarcity of training data and difficulties when applying syntax-based clustering in resource-poor settings; in the case of the latter, errors in the automatically-assigned parses might negatively impact the quality of the resultant clustering.

3.6 Related Work

As mentioned earlier, our approach was inspired by the success of Miller et al. (2004), who demonstrated the effectiveness of using word clusters as features in a discriminative learning approach. Our research, however, applies this technique to dependency parsing rather than named-entity recognition.

In this chapter, we have focused on developing new representations for lexical information. Previous research in this area includes several models which incorporate hidden variables (Matsuzaki et al., 2005; Koo and Collins, 2005; Petrov et al., 2006; Titov and Henderson, 2007). These approaches have the advantage that the model is able to learn different usages for the hidden variables, depending on the target problem at hand. Crucially, however, these methods do not exploit unlabeled data when learning their representations. In addition, performing inference in the presence of hidden variables can impose additional computational burdens.

Finkel et al. (2007) describe a method for automatically inducing a set of unsupervised POS tags (i.e., word clusters) based on their context within the structure of a dependency tree. Their approach is based on an application of hierarchical Dirichlet processes (Teh et al., 2006), which allows the number of word clusters to be defined by the data; note, however, that the method is not entirely without hyper-parameters (e.g., the tuning parameters γ, α_0 of the Dirichlet processes). Syntax-derived clusters

such as these could also be exploited in our method, and might provide a more relevant source of lexically-related information. Unfortunately, syntax-based clustering methods are reliant on the availability of high-quality syntactic annotations, which are expensive to obtain. In contrast, our method requires only abundantly-available raw text.

Finkel et al. (2007) also describe experiments where hierarchical Dirichlet processes are used to learn a splitting of POS tags based on the structural context of a dependency tree, similar to work by Koo and Collins (2005) and Petrov et al. (2006). The split POS tags are then used to train an improved dependency parser on the Penn Treebank, using the standard data split. These experiments are carried out in a generative parsing framework as opposed to the discriminative methods used here. The overall performance levels attained—86.18% UAS for first-order generative dependency parsers and 87.35% UAS for second-order parsing with a sibling factorization similar to McDonald and Pereira (2006)—are similar to the performance obtained by our first- and second-order parsers when using 1k syntactically annotated sentences and a fully-trained POS tagger, or 2k syntactically annotated sentences and a reduced POS tagger (see Table 3.3[8]).

Semi-supervised phrase structure parsing has been previously explored by McClosky et al. (2006), who applied a reranked parser to a large unsupervised corpus in order to obtain additional training data for the parser; this self-training approach was shown to be quite effective in practice. However, their approach depends on the usage of a high-quality parse reranker, whereas the method described here simply augments the features of an existing parser. Note that our two approaches are compatible in that we could also design a reranker and apply self-training techniques on top of the cluster-based features.

Wang et al. (2005) used distributional similarity scores to smooth a generative probability model for dependency parsing and obtained improvements in a Chinese parsing task. Our approach is similar to theirs in that the Brown algorithm can

[8] Bear in mind that Table 3.3 contains results from the development set (Section 22) rather than the test set (Section 23), so the performance levels are not directly comparable. However, in practice we observe only small differences in performance between the two.

be viewed as producing clusters based on similarity within the context of sequential bigrams, and the cluster-based features can be viewed as being a kind of "backed-off" version of the baseline features. However, our work is focused on discriminative learning as opposed to generative models. Note that distributional clusters and self-training methods were both used by Charniak (1997), who applied them to the task of treebank phrase-structure parsing.

Appearing simultaneously with Koo et al. (2008), Finkel et al. (2008) present a discriminative, weighted, lexicalized CFG parser formulated as a feature-rich log-linear model. The parser includes, as one of its features, a cluster label derived from a distributional clustering algorithm (Clark, 2000); similar to the incorporation of Brown et al. (1992) clusters in our approach. However, there is no characterization as to whether the cluster-based features were found to be particularly useful in their parser. Empirically, the performance of the Finkel et al. (2008) parser is comparable to earlier efforts in CFG parsing (e.g., Model 2 of Collins, 1999).

More recently, Suzuki et al. (2009) have built on the work presented in this chapter, combining cluster-based features with the generative-discriminative semi-supervised learning method of Suzuki and Isozaki (2008). The method was evaluated in experiments with varying amounts of unlabeled data, scaling in some cases to over 3 billion words of raw text.

3.7 Conclusions

In this chapter, we have presented a simple but effective semi-supervised learning approach and demonstrated that it achieves substantial improvement over a competitive baseline in two broad-coverage dependency parsing tasks. Despite this success, there are several ways in which our approach might be improved.

To begin, recall that the Brown clustering algorithm is based on a bigram language model. Intuitively, there is a "mismatch" between the kind of lexical information that is captured by the Brown clusters and the kind of lexical information that is modeled in dependency parsing. A natural avenue for further research would be the

development of clustering algorithms that reflect the syntactic behavior of words; e.g., an algorithm that attempts to maximize the likelihood of a treebank, according to a probabilistic dependency model. Alternately, one could design clustering algorithms that cluster entire head-modifier arcs rather than individual words.

Another idea would be to integrate the clustering algorithm into the training algorithm in a limited fashion. For example, after training an initial parser, one could parse a large amount of unlabeled text and use those parses to improve the quality of the clusters. These improved clusters can then be used to retrain an improved parser, resulting in an overall algorithm similar to that of McClosky et al. (2006).

Setting aside the development of new clustering algorithms, a final area for future work is the extension of our method to new domains, such as conversational text or other languages, and new NLP problems, such as machine translation.

Chapter 4

Structured Prediction Models via the Matrix-Tree Theorem

Parts of this chapter are joint work with Amir Globerson, Xavier Carreras, and Michael Collins, originally published in Koo et al. (2007).

This chapter provides an algorithmic framework for learning statistical models involving directed spanning trees, or equivalently non-projective dependency structures. We show how partition functions and marginals for directed spanning trees can be computed by an adaptation of Kirchhoff's Matrix-Tree Theorem. To demonstrate an application of the method, we perform experiments which use the algorithm in training both log-linear and max-margin dependency parsers. The new training methods give improvements in accuracy over perceptron-trained models.

4.1 Introduction

Learning with structured data typically involves searching or summing over a set with an exponential number of structured elements, for example the set of all parse trees for a given sentence. Methods for summing over such structures include the inside-outside algorithm for probabilistic context-free grammars (Baker, 1979), the forward-backward algorithm for hidden Markov models (Baum et al., 1970), and the belief-propagation algorithm for graphical models (Pearl, 1988). These algorithms

compute marginal probabilities and partition functions, quantities which are central to many methods for the statistical modeling of complex structures (e.g., the EM algorithm (Baker, 1979; Baum et al., 1970), contrastive estimation (Smith and Eisner, 2005), training algorithms for CRFs (Lafferty et al., 2001), and training algorithms for max-margin models (Bartlett et al., 2004; Taskar et al., 2003)).

This chapter describes inside-outside-style algorithms for the case of directed spanning trees. These structures are equivalent to non-projective dependency parses (McDonald et al., 2005b), and more generally could be relevant to any task that involves learning a mapping from a graph to an underlying spanning tree. Unlike the case for projective dependency structures, partition functions and marginals for non-projective trees cannot be computed using dynamic-programming methods such as the inside-outside algorithm. In this chapter we describe how these quantities can be computed by adapting a well-known result in graph theory: Kirchhoff's Matrix-Tree Theorem (Tutte, 1984; Kirchhoff, 1847). A naïve application of the theorem yields $O(n^4)$ and $O(n^6)$ algorithms for computation of the partition function and marginals, respectively. However, our adaptation finds the partition function and marginals in $O(n^3)$ time using simple matrix determinant and inversion operations.

We demonstrate an application of the new inference algorithm to non-projective dependency parsing. Specifically, we show how to implement two popular supervised learning approaches for this task: globally-normalized log-linear models and max-margin models. Log-linear estimation critically depends on the calculation of partition functions and marginals, which can be computed by our algorithms. For max-margin models, Bartlett et al. (2004) have provided a simple training algorithm, based on exponentiated gradient (EG) updates, that requires computation of marginals and can thus be implemented within our framework. Both of these methods explicitly minimize the loss incurred when parsing the entire training set. This contrasts with the online learning algorithms used in previous work with spanning-tree models (McDonald et al., 2005b).

We applied the above two *marginal-based* training algorithms to six languages with varying degrees of non-projectivity, using datasets obtained from the CoNLL-

X shared task (Buchholz and Marsi, 2006). Our experimental framework compared three training approaches: log-linear models, max-margin models, and the averaged perceptron. Each of these was applied to both projective and non-projective parsing. Our results demonstrate that marginal-based training yields models which outperform those trained using the averaged perceptron.

In summary, the contributions of this chapter are:

1. We introduce algorithms for *inside-outside-style* calculations for directed spanning trees, or equivalently non-projective dependency structures. These algorithms should have wide applicability in learning problems involving spanning-tree structures.

2. We illustrate the utility of these algorithms in log-linear training of dependency parsing models, and show improvements in accuracy when compared to averaged-perceptron training.

3. We also train max-margin models for dependency parsing via an EG algorithm (Bartlett et al., 2004). The experiments presented here constitute the first application of this algorithm to a large-scale problem. We again show improved performance over the perceptron.

The goal of our experiments is to give a rigorous comparative study of the marginal-based training algorithms and a highly-competitive baseline, the averaged perceptron, using the same feature sets for all approaches. We stress, however, that the purpose of this work is not to give competitive performance on the CoNLL data sets; this would require further engineering of the approach.

Similar adaptations of the Matrix-Tree Theorem have been developed independently and simultaneously by Smith and Smith (2007) and McDonald and Satta (2007); see Section 4.5 for more discussion.

4.2 Three Inference Problems

As mentioned in Section 2.3.2, the dependency parsers we consider consist of a feature mapping $\mathbf{\Phi}$ and a parameter vector $\mathbf{w}$. In a typical supervised learning setting, the parameters $\mathbf{w}$ are learned from a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where each $\mathbf{x}_i$ is a sentence and each y_i is the proper dependency structure for $\mathbf{x}_i$.

In this chapter, we will address three inference problems that arise in training and decoding for discriminative dependency parsers. Much of the previous work on learning $\mathbf{w}$ has focused on learning local models (see Section 4.5). McDonald et al. (2005a,b) trained global models using online algorithms such as the perceptron algorithm or MIRA. Here, we consider training algorithms based on work in conditional random fields (CRFs) (Lafferty et al., 2001) and max-margin methods (Taskar et al., 2003). The inference problems we describe below will prove to be critical in the implementation of these training methods.

To begin, we introduce a slightly different parameterization of the parsing problem. Assume that we have a vector $\boldsymbol{\theta}$ with values $\theta_{h,m} \in \mathbb{R}$ for every dependency $(h, m) \in \mathcal{D}(\mathbf{x})$; i.e., $\boldsymbol{\theta}$ is a vector assigning a weight to every possible dependency in the sentence $\mathbf{x}$. Within this parameterization, we define a conditional distribution over all dependency structures $y \in \mathcal{Y}(\mathbf{x})$ as follows:

$$P(y \mid \mathbf{x}; \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}; \boldsymbol{\theta})} \exp\left\{ \sum_{(h,m) \in y} \theta_{h,m} \right\} \quad (4.1)$$

$$Z(\mathbf{x}; \boldsymbol{\theta}) = \sum_{y \in \mathcal{Y}(\mathbf{x})} \exp\left\{ \sum_{(h,m) \in y} \theta_{h,m} \right\} \quad (4.2)$$

The function $Z(\mathbf{x}; \boldsymbol{\theta})$ is commonly referred to as the *partition function*. As before, the set $\mathcal{Y}(\mathbf{x})$ can be defined as $\mathcal{Y}_p^s(\mathbf{x})$, $\mathcal{Y}_{np}^s(\mathbf{x})$, $\mathcal{Y}_p^m(\mathbf{x})$ or $\mathcal{Y}_{np}^m(\mathbf{x})$, depending on the goals of the parsing task in question; see Figure 4-1 for examples of these four classes of dependency structures.

Given the distribution $P(y \mid \mathbf{x}; \boldsymbol{\theta})$, we can define the *marginal probability* of a

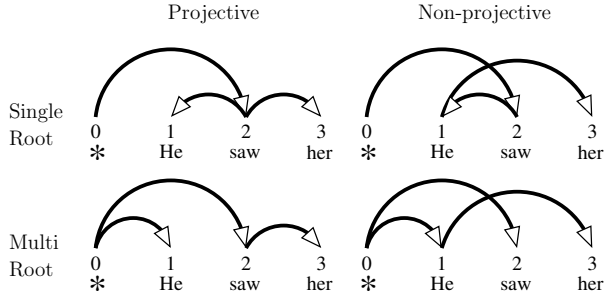


Figure 4-1: Examples of the four types of dependency structures.

dependency (h, m) as

$$\mu_{h,m}(\mathbf{x}; \boldsymbol{\theta}) = \sum_{y \in \mathcal{Y}(\mathbf{x}) : (h,m) \in y} P(y \mid \mathbf{x}; \boldsymbol{\theta})$$

The inference problems are then as follows:

Problem 1: **Decoding:**
Find $\text{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \sum_{(h,m) \in y} \theta_{h,m}$

Problem 2: **Computation of the Partition Function:** Calculate $Z(\mathbf{x}; \boldsymbol{\theta})$.

Problem 3: **Computation of the Marginals:**
For all $(h, m) \in \mathcal{D}(\mathbf{x})$, calculate $\mu_{h,m}(\mathbf{x}; \boldsymbol{\theta})$.

Note that all three problems require a maximization or summation over the set $\mathcal{Y}(\mathbf{x})$, which is exponential in size. There is a clear motivation for being able to solve Problem 1: by setting $\theta_{h,m} = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, h, m)$, the optimal dependency structure $y^*(\mathbf{x}; \mathbf{w})$ (see Eq. 2.1) can be computed. In this chapter the motivation for solving Problems 2 and 3 arises from training algorithms for discriminative models. As we will describe in Section 4.4, both log-linear and max-margin models can be trained via methods that make direct use of algorithms for Problems 2 and 3.

In the case of projective dependency structures (i.e., $\mathcal{Y}(\mathbf{x})$ defined as $\mathcal{Y}_p^s(\mathbf{x})$ or $\mathcal{Y}_p^m(\mathbf{x})$), there are well-known algorithms for all three inference problems. Decoding can be carried out using Viterbi-style dynamic-programming algorithms, for example the $O(n^3)$ algorithm of Eisner (1996). Computation of the marginals and partition function can also be achieved in $O(n^3)$ time, using a variant of the inside-outside algorithm (Baker, 1979) applied to the Eisner (1996) data structures (Paskin, 2001).

In the non-projective case (i.e., $\mathcal{Y}(\mathbf{x})$ defined as $\mathcal{Y}_{np}^s(\mathbf{x})$ or $\mathcal{Y}_{np}^m(\mathbf{x})$), McDonald et al. (2005b) describe how the CLE algorithm (Chu and Liu, 1965; Edmonds, 1967) can be used for decoding. However, it is not possible to compute the marginals and partition function using the inside-outside algorithm. We next describe a method for computing these quantities in $O(n^3)$ time using matrix inverse and determinant operations.

4.3 Spanning-Tree Inference via the Matrix-Tree Theorem

In this section we present algorithms for computing the partition function and marginals, as defined in Section 4.2, for non-projective parsing. We first reiterate the observation of McDonald et al. (2005a) that non-projective parses correspond to directed spanning trees on a complete directed graph of n nodes, where n is the length of the sentence. The above inference problems thus involve summation over the set of all directed spanning trees. Note that this set is exponentially large, and there is no obvious method for decomposing the sum into dynamic-programming-like subproblems. This section describes how a variant of Kirchhoff's Matrix-Tree Theorem (Tutte, 1984) can be used to evaluate the partition function and marginals efficiently.

In what follows, we consider the single-root setting (i.e., $\mathcal{Y}(\mathbf{x}) = \mathcal{Y}_{np}^s(\mathbf{x})$), leaving the multi-root case (i.e., $\mathcal{Y}(\mathbf{x}) = \mathcal{Y}_{np}^m(\mathbf{x})$) to Section 4.3.3. For a sentence $\mathbf{x}$ with n words, define a complete directed graph G on n nodes, where each node corresponds to a word in $\mathbf{x}$, and each edge corresponds to a dependency between two words

in $\mathbf{x}$. Note that G does *not* include the root-symbol $h = 0$, nor does it account for any dependencies $(0, m)$ headed by the root-symbol. We assign non-negative weights to the edges of this graph, yielding the following weighted adjacency matrix $A(\boldsymbol{\theta}) \in \mathbb{R}^{n \times n}$, for $h, m = 1 \ldots n$:

$$A_{h,m}(\boldsymbol{\theta}) \;=\; \begin{cases} 0, & \text{if } h = m \\ \exp\{\theta_{h,m}\}, & \text{otherwise} \end{cases}$$

To account for the dependencies $(0, m)$ headed by the root-symbol, we define a vector of root-selection scores $\mathbf{r}(\boldsymbol{\theta}) \in \mathbb{R}^n$, for $m = 1 \ldots n$:

$$r_m(\boldsymbol{\theta}) \;=\; \exp\{\theta_{0,m}\}$$

Let the weight of a dependency structure $y \in \mathcal{Y}^s_{np}(\mathbf{x})$ be defined as:

$$\psi(y; \boldsymbol{\theta}) \;=\; r_{\text{root}(y)}(\boldsymbol{\theta}) \prod_{(h,m) \in y \colon h \neq 0} A_{h,m}(\boldsymbol{\theta})$$

Here, $\text{root}(y) = m : (0, m) \in y$ is the child of the root-symbol; there is exactly one such child, since $y \in \mathcal{Y}^s_{np}(\mathbf{x})$. Eq. 4.1 and Eq. 4.2 can be rephrased as:

$$P(y \,|\, \mathbf{x}; \boldsymbol{\theta}) \;=\; \frac{\psi(y; \boldsymbol{\theta})}{Z(\mathbf{x}; \boldsymbol{\theta})} \tag{4.3}$$

$$Z(\mathbf{x}; \boldsymbol{\theta}) \;=\; \sum_{y \in \mathcal{Y}^s_{np}(\mathbf{x})} \psi(y; \boldsymbol{\theta}) \tag{4.4}$$

In the remainder of this section, we drop the notational dependence on $\mathbf{x}$ for brevity.

The original Matrix-Tree Theorem addressed the problem of counting the number of undirected spanning trees in an undirected graph. For the models we study here, we require a sum of *weighted* and *directed* spanning trees. Tutte (1984) extended the Matrix-Tree Theorem to this case. We briefly summarize his method below.

First, define the Laplacian matrix $L(\boldsymbol{\theta}) \in \mathbb{R}^{n \times n}$ of G, for $h, m = 1 \ldots n$:

$$L_{h,m}(\boldsymbol{\theta}) \;=\; \begin{cases} \sum_{h'=1}^{n} A_{h',m}(\boldsymbol{\theta}) & \text{if } h = m \\ -A_{h,m}(\boldsymbol{\theta}) & \text{otherwise} \end{cases}$$

Second, for a matrix X, let $X^{(h,m)}$ be the *minor* of X with respect to row h and column m; i.e., the determinant of the matrix formed by deleting row h and column m from X. Finally, define the weight of any directed spanning tree of G to be the product of the weights $A_{h,m}(\boldsymbol{\theta})$ for the edges in that tree.

Theorem 1 *(Tutte, 1984, p. 140). Let $L(\boldsymbol{\theta})$ be the Laplacian matrix of G. Then $L^{(m,m)}(\boldsymbol{\theta})$ is equal to the sum of the weights of all directed spanning trees of G which are rooted at m. Furthermore, the minors vary only in sign when traversing the columns of the Laplacian (Tutte, 1984, p. 150):*

$$\forall h, m\colon\; (-1)^{h+m} L^{(h,m)}(\boldsymbol{\theta}) \;=\; L^{(m,m)}(\boldsymbol{\theta}) \tag{4.5}$$

4.3.1 Partition Functions via Matrix Determinants

From Theorem 1, it directly follows that

$$L^{(m,m)}(\boldsymbol{\theta}) \;=\; \sum_{y \in \mathcal{U}(m)} \prod_{(h,m) \in y \colon h \neq 0} A_{h,m}(\boldsymbol{\theta})$$

where $\mathcal{U}(m) = \{y \in \mathcal{Y}^s_{np} : \text{root}(y) = m\}$. A naïve method for computing the partition function is therefore to evaluate

$$Z(\boldsymbol{\theta}) \;=\; \sum_{m=1}^{n} r_m(\boldsymbol{\theta}) L^{(m,m)}(\boldsymbol{\theta})$$

The above would require calculating n determinants, resulting in $O(n^4)$ complexity. However, as we show below $Z(\boldsymbol{\theta})$ may be obtained in $O(n^3)$ time using a single determinant evaluation.

Define a new matrix $\hat{L}(\boldsymbol{\theta})$ to be $L(\boldsymbol{\theta})$ with the first row replaced by the root-selection scores:

$$\hat{L}_{h,m}(\boldsymbol{\theta}) \;=\; \begin{cases} r_m(\boldsymbol{\theta}) & h = 1 \\ L_{h,m}(\boldsymbol{\theta}) & h > 1 \end{cases}$$

This matrix allows direct computation of the partition function, as the following proposition shows.

Proposition 1 *The partition function in Eq. 4.4 is given by $Z(\boldsymbol{\theta}) = \left|\hat{L}(\boldsymbol{\theta})\right|$.*

Proof: *Consider the row expansion of $\left|\hat{L}(\boldsymbol{\theta})\right|$ with respect to row 1:*

$$\begin{aligned} \left|\hat{L}(\boldsymbol{\theta})\right| &= \sum_{m=1}^{n} (-1)^{1+m} \hat{L}_{1,m}(\boldsymbol{\theta}) \hat{L}^{(1,m)}(\boldsymbol{\theta}) \\ &= \sum_{m=1}^{n} (-1)^{1+m} r_m(\boldsymbol{\theta}) L^{(1,m)}(\boldsymbol{\theta}) \\ &= \sum_{m=1}^{n} r_m(\boldsymbol{\theta}) L^{(m,m)}(\boldsymbol{\theta}) \;=\; Z(\boldsymbol{\theta}) \end{aligned}$$

The second line follows from the construction of $\hat{L}(\boldsymbol{\theta})$, and the third line follows from Eq. 4.5. ■

4.3.2 Marginals via Matrix Inversion

The marginals we require are given by

$$\mu_{h,m}(\boldsymbol{\theta}) \;=\; \frac{1}{Z(\boldsymbol{\theta})} \sum_{y \in \mathcal{Y}^s_{np} \,:\, (h,m) \in y} \psi(y; \boldsymbol{\theta})$$

To calculate these marginals efficiently for all values of (h, m) we use a well-known identity relating the log partition-function to marginals

$$\mu_{h,m}(\boldsymbol{\theta}) \;=\; \frac{\partial \log Z(\boldsymbol{\theta})}{\partial \theta_{h,m}}$$

Since the partition function in this case has a closed-form expression (i.e., the determinant of a matrix constructed from $\boldsymbol{\theta}$), the marginals can also obtained in closed form.

Using the chain rule, the derivative of the log partition-function in Proposition 1 is

$$\begin{aligned} \mu_{h,m}(\boldsymbol{\theta}) &= \frac{\partial \log \left|\hat{L}(\boldsymbol{\theta})\right|}{\partial \theta_{h,m}} \\ &= \sum_{h'=1}^{n} \sum_{m'=1}^{n} \frac{\partial \log \left|\hat{L}(\boldsymbol{\theta})\right|}{\partial \hat{L}_{h',m'}(\boldsymbol{\theta})} \frac{\partial \hat{L}_{h',m'}(\boldsymbol{\theta})}{\partial \theta_{h,m}} \end{aligned}$$

To perform the derivative, we use the identity

$$\frac{\partial \log |X|}{\partial X} \;=\; \left(X^{-1}\right)^T$$

and the fact that $\frac{\partial \hat{L}_{h',m'}(\boldsymbol{\theta})}{\partial \theta_{h,m}}$ is nonzero for only a few h', m'. Specifically, when $h = 0$, the marginals are given by

$$\mu_{0,m}(\boldsymbol{\theta}) \;=\; r_m(\boldsymbol{\theta}) \left[\hat{L}^{-1}(\boldsymbol{\theta})\right]_{m,1}$$

and for $h > 0$, the marginals are given by

$$\begin{aligned} \mu_{h,m}(\boldsymbol{\theta}) \;=\; & (1 - \delta_{1,m}) A_{h,m}(\boldsymbol{\theta}) \left[\hat{L}^{-1}(\boldsymbol{\theta})\right]_{m,m} - \\ & (1 - \delta_{h,1}) A_{h,m}(\boldsymbol{\theta}) \left[\hat{L}^{-1}(\boldsymbol{\theta})\right]_{m,h} \end{aligned}$$

where $\delta_{h,m}$ is the Kronecker delta. Thus, the complexity of evaluating *all* the relevant marginals is dominated by the matrix inversion, and the total complexity is therefore $O(n^3)$.

4.3.3 Multiple Roots

In the case of multiple roots, we can still compute the partition function and marginals efficiently. In fact, the derivation of this case is simpler than for single-root structures. Create an extended graph G' which augments G with a dummy root node that has edges pointing to all of the existing nodes, weighted by the appropriate root-selection scores. Note that there is a bijection between directed spanning trees of G' rooted

at the dummy root and multi-root structures $y \in \mathcal{Y}^m_{np}(\mathbf{x})$. Thus, Theorem 1 can be used to compute the partition function directly: construct a Laplacian matrix $L(\boldsymbol{\theta})$ for G' and compute the minor $L^{(0,0)}(\boldsymbol{\theta})$. Since this minor is also a determinant, the marginals can be obtained analogously to the single-root case. More concretely, this technique corresponds to defining the matrix $\hat{L}(\boldsymbol{\theta})$ as

$$\hat{L}(\boldsymbol{\theta}) \;=\; L(\boldsymbol{\theta}) + \text{diag}(\mathbf{r}(\boldsymbol{\theta}))$$

where $\text{diag}(\mathbf{v})$ is the diagonal matrix with the vector $\mathbf{v}$ on its diagonal.

4.3.4 Labeled Trees

The techniques above extend easily to the case where dependencies are labeled. For a model with L different labels, it suffices to define the edge and root scores as $A_{h,m}(\boldsymbol{\theta}) = \sum_{\ell=1}^{L} \exp\{\theta_{h,m,\ell}\}$ and $r_m(\boldsymbol{\theta}) = \sum_{\ell=1}^{L} \exp\{\theta_{0,m,\ell}\}$. The partition function over labeled trees is obtained by operating on these values as described previously, and the marginals are given by an application of the chain rule. Both inference problems are solvable in $O(n^3 + Ln^2)$ time.

4.4 Training Algorithms

This section describes two methods for parameter estimation that rely explicitly on the computation of the partition function and marginals.

4.4.1 Log-Linear Estimation

In conditional log-linear models (Johnson et al., 1999; Lafferty et al., 2001), a distribution over parse trees for a sentence $\mathbf{x}$ is defined as follows:

$$P(y \,|\, \mathbf{x}; \mathbf{w}) \;=\; \frac{\exp\left\{\sum_{(h,m)\in y} \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, h, m)\right\}}{Z(\mathbf{x}; \mathbf{w})} \tag{4.6}$$

where $Z(\mathbf{x}; \mathbf{w})$ is the partition function, a sum over $\mathcal{Y}^s_p(\mathbf{x})$, $\mathcal{Y}^s_{np}(\mathbf{x})$, $\mathcal{Y}^m_p(\mathbf{x})$ or $\mathcal{Y}^m_{np}(\mathbf{x})$.

We train the model using the approach described by Sha and Pereira (2003). Assume that we have a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. The optimal parameters are taken to be $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\; L(\mathbf{w})$ where

$$L(\mathbf{w}) \;=\; -C \sum_{i=1}^{N} \log P(y_i \,|\, \mathbf{x}_i; \mathbf{w}) + \frac{1}{2} \|\mathbf{w}\|^2$$

The parameter $C > 0$ is a constant dictating the level of regularization in the model.

Since $L(\mathbf{w})$ is a convex function, gradient descent methods can be used to search for the global minimum. Such methods typically involve repeated computation of the loss $L(\mathbf{w})$ and gradient $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$, requiring efficient implementations of both functions. Note that the log-probability of a parse is

$$\log P(y \,|\, \mathbf{x}; \mathbf{w}) \;=\; \sum_{(h,m)\in y} \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, h, m) - \log Z(\mathbf{x}; \mathbf{w})$$

so that the main issue in calculating the loss function $L(\mathbf{w})$ is the evaluation of the partition functions $Z(\mathbf{x}_i; \mathbf{w})$. The gradient of the loss is given by

$$\begin{aligned}\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \;&=\; \mathbf{w} - C \sum_{i=1}^{N} \sum_{(h,m)\in y_i} \boldsymbol{\phi}(\mathbf{x}_i, h, m) \\ &\quad + C \sum_{i=1}^{N} \sum_{(h,m)\in \mathcal{D}(\mathbf{x}_i)} \mu_{h,m}(\mathbf{x}_i; \mathbf{w}) \boldsymbol{\phi}(\mathbf{x}_i, h, m)\end{aligned}$$

where

$$\mu_{h,m}(\mathbf{x}; \mathbf{w}) \;=\; \sum_{y \in \mathcal{Y}(\mathbf{x}) :\, (h,m)\in y} P(y \,|\, \mathbf{x}; \mathbf{w})$$

is the marginal probability of a dependency (h, m). Thus, the main issue in the evaluation of the gradient is the computation of the marginals $\mu_{h,m}(\mathbf{x}_i; \mathbf{w})$.

Note that Eq. 4.6 forms a special case of the log-linear distribution defined in Eq. 4.1 in Section 4.2. If we set $\theta_{h,m} = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, h, m)$ then we have $P(y \,|\, \mathbf{x}; \mathbf{w}) = P(y \,|\, \mathbf{x}; \boldsymbol{\theta})$, $Z(\mathbf{x}; \mathbf{w}) = Z(\mathbf{x}; \boldsymbol{\theta})$, and $\mu_{h,m}(\mathbf{x}; \mathbf{w}) = \mu_{h,m}(\mathbf{x}; \boldsymbol{\theta})$. Thus in the projective case the inside-outside algorithm can be used to calculate the partition function and marginals, thereby enabling training of a log-linear model; in the non-projective case the algorithms in Section 4.3 can be used for this purpose.

4.4.2 Max-Margin Estimation

The second learning algorithm we consider is the large-margin approach for structured prediction (Taskar et al., 2003, 2004). Learning in this framework again involves minimization of a convex function $L(\mathbf{w})$. Let the *margin* for parse tree y on the i'th training example be defined as

$$m_{i,y}(\mathbf{w}) \;=\; \sum_{(h,m)\in y_i} \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i, h, m) - \sum_{(h,m)\in y} \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i, h, m)$$

The loss function is then defined as

$$L(\mathbf{w}) = C \sum_{i=1}^{N} \max_{y \in \mathcal{Y}(\mathbf{x}_i)} \left(E_{i,y} - m_{i,y}(\mathbf{w})\right) + \frac{1}{2} \|\mathbf{w}\|^2$$

where $E_{i,y}$ is a measure of the part-wise error for parse y on the i'th training sentence. Taskar et al. (2003) suggest error measures based on the part-wise Hamming distance between the true structure y_i and the model's hypothesis. In this chapter we take $E_{i,y}$ to be the number of incorrect dependencies in the parse tree y when compared to the gold-standard parse tree y_i.

The definition of $L(\mathbf{w})$ makes use of the expression $\max_{y\in\mathcal{Y}(\mathbf{x}_i)} \left(E_{i,y} - m_{i,y}(\mathbf{w})\right)$ for the i'th training example, which is commonly referred to as the *hinge loss*. Note that $E_{i,y_i} = 0$, and also that $m_{i,y_i}(\mathbf{w}) = 0$, so that the hinge loss is always non-negative. In addition, the hinge loss is 0 if and only if $m_{i,y}(\mathbf{w}) \geq E_{i,y}$ for all $y \in \mathcal{Y}(\mathbf{x}_i)$. Thus the hinge loss directly penalizes margins $m_{i,y}(\mathbf{w})$ which are less than their corresponding losses $E_{i,y}$.

Figure 4-2 shows an algorithm for minimizing $L(\mathbf{w})$ that is based on the exponentiated gradient algorithm for large-margin optimization described by Bartlett et al. (2004). The algorithm maintains a set of weights $\theta_{i,h,m}$ for $i = 1 \ldots N, (h, m) \in \mathcal{D}(\mathbf{x}_i)$, which are updated example-by-example. The algorithm relies on the repeated com-

Inputs: Training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$.

Parameters: Regularization constant C, starting point β, number of passes over training set T.

Data Structures: Real values $\theta_{i,h,m}$ and $l_{i,h,m}$ for $i = 1 \ldots N, (h, m) \in \mathcal{D}(\mathbf{x}_i)$. Learning rate η.

Initialization: Set learning rate $\eta = \frac{1}{C}$. Set $\theta_{i,h,m} = \beta$ for $(h, m) \in y_i$, and $\theta_{i,h,m} = 0$ for $(h, m) \notin y_i$. Set $l_{i,h,m} = 0$ for $(h, m) \in y_i$, and $l_{i,h,m} = 1$ for $(h, m) \notin y_i$. Calculate initial parameters as

$$\mathbf{w} = C \sum_i \sum_{(h,m)\in\mathcal{D}(\mathbf{x}_i)} \delta_{i,h,m} \boldsymbol{\phi}(\mathbf{x}_i, h, m)$$

where $\delta_{i,h,m} = (1 - l_{i,h,m} - \mu_{i,h,m})$ and the $\mu_{i,h,m}$ values are calculated from the $\theta_{i,h,m}$ values as described in Eq. 4.7.

Algorithm: Repeat T passes over the training set, where each pass is as follows:

Set $obj = 0$

For $i = 1 \ldots N$

- For all $(h, m) \in \mathcal{D}(\mathbf{x}_i)$, set $\theta'_{i,h,m} = \theta_{i,h,m} + \eta C \left(l_{i,h,m} + \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i, h, m)\right)$
- For example i, calculate marginals $\mu_{i,h,m}$ from $\theta_{i,h,m}$ values, and marginals $\mu'_{i,h,m}$ from $\theta'_{i,h,m}$ values (see Eq. 4.7).
- Update the parameters: $\mathbf{w} = \mathbf{w} + C \sum_{(h,m)\in\mathcal{D}(\mathbf{x}_i)} \left(\mu_{i,h,m} - \mu'_{i,h,m}\right) \boldsymbol{\phi}(\mathbf{x}_i, h, m)$
- For all $(h, m) \in \mathcal{D}(\mathbf{x}_i)$, set $\theta_{i,h,m} = \theta'_{i,h,m}$
- Set $obj = obj + C \sum_{(h,m)\in\mathcal{D}(\mathbf{x}_i)} l_{i,h,m} \mu'_{i,h,m}$

Set $obj = obj - \frac{1}{2} \|\mathbf{w}\|^2$. If obj has decreased compared to last iteration, set $\eta = \frac{\eta}{2}$.

Output: Parameter values $\mathbf{w}$.

Figure 4-2: The EG Algorithm for max-margin estimation. The learning rate η is halved each time the dual objective function (see (Bartlett et al., 2004)) fails to increase. In our experiments we chose $\beta = 9$, which was found to work well during development of the algorithm.

putation of marginal values $\mu_{i,h,m}$, which are defined as follows:[1]

$$\mu_{i,h,m} = \sum_{y \in \mathcal{Y}(\mathbf{x}_i) : (h,m) \in y} P(y \mid \mathbf{x}_i) \quad (4.7)$$

$$P(y \mid \mathbf{x}_i) = \frac{\exp\left\{\sum_{(h,m) \in y} \theta_{i,h,m}\right\}}{\sum_{y' \in \mathcal{Y}(\mathbf{x}_i)} \exp\left\{\sum_{(h,m) \in y'} \theta_{i,h,m}\right\}}$$

A similar definition is used to derive marginal values $\mu'_{i,h,m}$ from the values $\theta'_{i,h,m}$. Computation of the $\boldsymbol{\mu}$ and $\boldsymbol{\mu}'$ values is again inference of the form described in Problem 3 in Section 4.2, and can be achieved using the inside-outside algorithm for projective structures, and the algorithms described in Section 4.3 for non-projective structures.

Note that in recent work, the EG algorithm has been expanded from max-margin estimation to both log-linear and max-margin estimation (Collins et al., 2008). In addition, improved proofs of convergence and proofs for rates of convergence have also been given, with applicability to learning objectives satisfying various conditions. As it turns out, EG is able to obtain a fast $O(log(\frac{1}{\epsilon}))$ rate of convergence when optimizing a log-linear objective function, as described by Collins et al. (2008). Since the proof of this fast rate of convergence was developed by the current author, it is included in Appendix A as an ancillary contribution.

4.5 Related Work

Global log-linear training has been used in the context of PCFG parsing (Johnson, 2001). Kaplan et al. (2004) explore a similar application of log-linear models to LFG parsing. Max-margin learning has been applied to PCFG parsing by Taskar et al. (2004). They show that this problem has a QP dual of polynomial size, where the dual variables correspond to marginal probabilities of CFG rules. A similar QP dual

[1] Bartlett et al. (2004) write $P(y \mid \mathbf{x}_i)$ as $\alpha_{i,y}$. The $\alpha_{i,y}$ variables are dual variables that appear in the dual objective function, i.e., the convex dual of $L(\mathbf{w})$. Analysis of the algorithm shows that as the $\theta_{i,h,m}$ variables are updated, the dual variables converge to the optimal point of the dual objective, and the parameters $\mathbf{w}$ converge to the minimum of $L(\mathbf{w})$.

may be obtained for max-margin projective dependency parsing. However, for non-projective parsing, the dual QP may be difficult to formulate compactly (Chopra, 1989). Nevertheless, alternative optimization methods like that of Tsochantaridis et al. (2004), or the EG method presented here, can still be applied.

The majority of previous work on dependency parsing has focused on local (i.e., classification of individual edges) discriminative training methods (Yamada and Matsumoto, 2003; Nivre et al., 2004; Cheng et al., 2005). Non-local (i.e., classification of entire trees) training methods were used by McDonald et al. (2005a), who employed online learning.

Dependency parsing accuracy can be improved by allowing second-order features, which consider more than one dependency simultaneously. McDonald and Pereira (2006) define a second-order dependency parsing model in which interactions between adjacent siblings are allowed, and Carreras (2007) defines a second-order model that allows grandparent and sibling interactions. Both authors give polytime algorithms for exact projective parsing. By adapting the inside-outside algorithm to these models, partition functions and marginals can be computed for second-order projective structures, allowing log-linear and max-margin training to be applied via the framework developed in this chapter. For higher-order non-projective parsing, however, computational complexity results (McDonald and Pereira, 2006; McDonald and Satta, 2007) indicate that exact solutions to the three inference problems of Section 4.2 will be intractable. Exploration of approximate second-order non-projective inference is a natural avenue for future research.

Two other groups of authors have independently and simultaneously proposed adaptations of the Matrix-Tree Theorem for structured inference on directed spanning trees (McDonald and Satta, 2007; Smith and Smith, 2007). There are some algorithmic differences between these papers and the current work. First, we define both multi-root and single-root algorithms, whereas the other papers only consider multi-root parsing. This distinction can be important as one often expects a dependency structure to have exactly one child attached to the root-symbol, as is the case in a single-root structure. Second, McDonald and Satta (2007) propose an $O(n^5)$ algorithm for computing the marginals, as opposed to the $O(n^3)$ matrix-inversion approach used by Smith and Smith (2007) and ourselves.

In addition to the algorithmic differences, both groups of authors consider applications of the Matrix-Tree Theorem which we have not discussed. For example, both papers propose minimum-risk decoding, and McDonald and Satta (2007) discuss unsupervised learning and language modeling, while Smith and Smith (2007) define hidden-variable models based on spanning trees.

In this chapter we used EG training methods only for max-margin models (Bartlett et al., 2004). However, Globerson et al. (2007) have recently shown how EG updates can be applied to efficient training of log-linear models.

4.6 Experiments on Dependency Parsing

In this section, we present experimental results applying our inference algorithms for dependency parsing models. Our primary purpose is to establish comparisons along two relevant dimensions: projective training vs. non-projective training, and marginal-based training algorithms vs. the averaged perceptron. The feature representation and other relevant dimensions are kept fixed in the experiments.

4.6.1 Data Sets and Features

We used data from the CoNLL-X shared task on multilingual dependency parsing (Buchholz and Marsi, 2006). In our experiments, we used a subset consisting of six languages; Table 4.1 gives details of the data sets used. Our subset includes the two languages with the lowest accuracy in the CoNLL-X evaluations (Turkish and Arabic), the language with the highest accuracy (Japanese), the most non-projective language (Dutch), a moderately non-projective language (Slovene), and a highly projective language (Spanish). All languages but Spanish have multi-root parses in their data. We are grateful to the providers of the treebanks that constituted the data of our experiments (Hajič et al., 2004; van der Beek et al., 2002; Kawata and Bartels, 2000; Džeroski et al., 2006; Toruella and Antonín, 2002; Oflazer et al., 2003).

For each language we created a validation set that was a subset of the CoNLL-X training set for that language. The remainder of each training set was used to train the models for the different languages. The validation sets were used to tune the meta-parameters (e.g., the value of the regularization constant C) of the different training algorithms. We used the official test sets and evaluation script from the CoNLL-X task. All of the results that we report are for unlabeled dependency parsing. Note that our algorithms also support labeled parsing (see Section 4.3.4). A major implication of introducing labels is that the feature space greatly increases with the number of labels, regardless of the training method used. Initial experiments with labeled models showed the same trend that we report here for unlabeled parsing, so for simplicity we conducted extensive experiments only for unlabeled parsing.

The non-projective models were trained on the CoNLL-X data in its original form. Since the projective models assume that the dependencies in the data are non-crossing, we created a second training set for each language where non-projective dependency structures were automatically transformed into projective structures. The transformations were performed by running the projective parser with score +1 on correct dependencies and -1 otherwise: the resulting trees are guaranteed to be projective and to have a minimum loss with respect to the correct tree. Note that only the training sets were transformed; all projective models were trained on these new training sets. Our feature space is based on that of McDonald et al. (2005a); it should be noted that McDonald et al. (2006) use a richer feature set based on second-order interactions that is incomparable to our features. The features are binary indicator functions, each evaluating the presence of a certain pattern in a dependency. The most basic feature patterns consider the form, part-of-speech, lemma and other morpho-syntactic attributes of the head and the modifier of a dependency. At a higher level, the representation exploits a variety of conjunctions of the forms and part-of-speech tags of the following items: the head and modifier; the head, modifier, and any token in between them; the head, modifier, and the two tokens following or preceding them.

language	%cd	train	val.	test
Arabic	0.34	49,064	5,315	5,373
Dutch	4.93	178,861	16,208	5,585
Japanese	0.70	141,966	9,495	5,711
Slovene	1.59	22,949	5,801	6,390
Spanish	0.06	78,310	11,024	5,694
Turkish	1.26	51,827	5,683	7,547

Table 4.1: Characterization of the multilingual datasets. The 2nd column (%cd) is the percentage of crossing dependencies in the training and validation sets. The last three columns report the size in tokens of the training, validation and test sets.

4.6.2 Results

We performed experiments using three training algorithms: the averaged perceptron (Collins, 2002), log-linear training (via conjugate gradient descent), and max-margin training (via the EG algorithm). Each of these algorithms was trained using projective and non-projective methods, yielding six training settings per language. The different training algorithms have various meta-parameters, which we optimized on the validation set for each language/training-setting combination. The averaged perceptron has a single meta-parameter, namely the number of iterations over the training set. The log-linear models have two meta-parameters: the regularization constant C and the number of gradient steps T taken by the conjugate-gradient optimizer. The EG approach also has two meta-parameters: the regularization constant C and the number of iterations, T.

We trained the perceptron for 100 iterations, and chose the iteration which led to the best score on the validation set. Note that in all of our experiments, the best perceptron results were actually obtained with 30 or fewer iterations. For the log-linear and EG algorithms we tested a number of values for C, and for each value of C ran 100 gradient steps or EG iterations, finally choosing the best combination of C and T found in validation.

For models trained using non-projective algorithms, both projective and non-projective parsing was tested on the validation set, and the highest scoring of these two approaches was then used to decode test data sentences.

Table 4.2 reports test results for the six training scenarios. These results show that

	Perceptron		Max-Margin		Log-Linear	
	p	np	p	np	p	np
Ara	71.74	71.84	71.74	72.99	73.11	**73.67**
Dut	77.17	78.83	76.53	**79.69**	76.23	79.55
Jap	91.90	91.78	92.10	**92.18**	91.68	91.49
Slo	78.02	78.66	79.78	**80.10**	78.24	79.66
Spa	81.19	80.02	81.71	**81.93**	81.75	81.57
Tur	71.22	71.70	**72.83**	72.02	72.26	72.62

Table 4.2: Test results for multilingual parsing. The p and np columns show results with projective and non-projective training respectively.

	Ara	Dut	Jap	Slo	Spa	Tur	AV
P	71.74	78.83	91.78	78.66	81.19	71.70	79.05
E	72.99	**79.69**	**92.18**	**80.10**	**81.93**	72.02	**79.82**
L	**73.67**	79.55	91.49	79.66	81.57	**72.26**	79.71

Table 4.3: Comparison between the three training algorithms for multilingual parsing. P = perceptron, E = EG, L = log-linear models, and AV is an average across the results for the different languages.

for Dutch, which is the language in our data that has the highest number of crossing dependencies, non-projective training gives significant gains over projective training for all three training methods. For the other languages, non-projective training gives similar or even improved performance over projective training.

Table 4.3 gives an additional set of results, which were calculated as follows. For each of the three training methods, we used the validation set results to choose between projective and non-projective training. This allows us to make a direct comparison of the three training algorithms. Table 4.3 shows the results of this comparison.[2] The results show that log-linear and max-margin models both give a higher average accuracy than the perceptron. For some languages (e.g., Japanese), the differences from the perceptron are small; however for other languages (e.g., Arabic, Dutch or Slovene) the improvements seen are quite substantial.

[2] We ran the sign test at the sentence level to measure the statistical significance of the results aggregated across the six languages. Out of 2,472 sentences total, log-linear models gave improved parses over the perceptron on 448 sentences, and worse parses on 343 sentences. The max-margin method gave improved/worse parses for 500/383 sentences. Both results are significant with $p \leq 0.001$.

4.7 Conclusions

This chapter describes inference algorithms for spanning-tree distributions, focusing on the fundamental problems of computing partition functions and marginals. Although we concentrate on log-linear and max-margin estimation, the inference algorithms we present can serve as black-boxes in many other statistical modeling techniques.

Our experiments suggest that marginal-based training produces more accurate models than perceptron learning. Notably, this is the first large-scale application of the EG algorithm, and shows that it is a promising approach for structured learning.

In line with McDonald et al. (2005b), we confirm that spanning-tree models are well-suited to dependency parsing, especially for highly non-projective languages such as Dutch. Moreover, spanning-tree models should be useful for a variety of other problems involving structured data.

Chapter 5

Efficient Third-order Dependency Parsers

Parts of this chapter are joint work with Michael Collins, originally published in Koo and Collins (2010). Appendix B provides additional details regarding the parsing algorithms presented in this chapter.

We present algorithms for higher-order dependency parsing that are "third-order" in the sense that they can evaluate sub-structures containing three dependencies, and "efficient" in the sense that they require only $O(n^4)$ time. Importantly, our new parsers can utilize both sibling-style and grandchild-style interactions. We evaluate our parsers on the Penn Treebank and Prague Dependency Treebank, achieving unlabeled attachment scores of 93.04% and 87.38%, respectively.

5.1 Introduction

Dependency grammar has proven to be a very useful syntactic formalism, due in no small part to the development of efficient parsing algorithms (Eisner, 2000; McDonald et al., 2005b; McDonald and Pereira, 2006; Carreras, 2007), which can be leveraged for a wide variety of learning methods, such as feature-rich discriminative models (Lafferty et al., 2001; Collins, 2002; Taskar et al., 2003). These parsing algorithms share an important characteristic: they *factor* dependency trees into sets of *parts*

that have limited interactions. By exploiting the additional constraints arising from the factorization, maximizations or summations over the set of possible dependency trees can be performed efficiently and exactly.

A crucial limitation of factored parsing algorithms is that the associated parts are typically quite small, losing much of the contextual information within the dependency tree. For the purposes of improving parsing performance, it is desirable to increase the size and variety of the parts used by the factorization.[1] At the same time, the need for more expressive factorizations must be balanced against any resulting increase in the computational cost of the parsing algorithm. Consequently, recent work in dependency parsing has been restricted to applications of second-order parsers, the most powerful of which (Carreras, 2007) requires $O(n^4)$ time and $O(n^3)$ space, while being limited to second-order parts.

In this chapter, we present new third-order parsing algorithms that increase both the size and variety of the parts participating in the factorization, while simultaneously maintaining computational requirements of $O(n^4)$ time and $O(n^3)$ space. We evaluate our parsers on the Penn WSJ Treebank (Marcus et al., 1993) and Prague Dependency Treebank (Hajič et al., 2001), achieving unlabeled attachment scores of 93.04% and 87.38%.

The remainder of this chapter is divided as follows: Sections 5.2 and 5.3 give background, Sections 5.4 and 5.5 describe our new parsing algorithms, Section 5.6 discusses related work, Section 5.7 presents our experimental results, and Section 5.8 concludes.

5.2 Dependency Parsing

In this chapter, we will attempt to describe our parsing algorithms in the most widely-applicable terms possible. Thus, in this section we briefly reformulate the parsing problem in terms of generalized part-scoring functions, so that our parsers can remain agnostic as to the source of the scores that form the input of the parser.

For a sentence $\mathbf{x}$, we reframe dependency parsing as a search for the highest-scoring dependency tree:

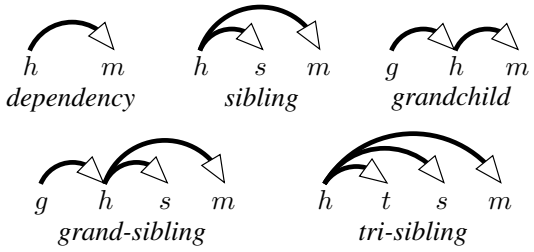
$$y^\star(\mathbf{x}) \;=\; \operatorname*{argmax}_{y \in \mathcal{Y}(\mathbf{x})} \sum_{p \in y} \textsc{Score}(\mathbf{x}, p) \tag{5.1}$$

where $\textsc{Score}(\mathbf{x}, p)$ is a general function that assigns a real-valued score to the event that part p is appears in the syntactic analysis of $\mathbf{x}$. As usual, $\mathcal{Y}(\mathbf{x})$ is the set of all trees compatible with $\mathbf{x}$, and we have assumed that each tree y can be *factored* into a set of small *parts*. Clearly, defining

$$\textsc{Score}(\mathbf{x}, p) \;=\; \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, p)$$

would return us to the framework of structured linear models. Alternative definitions may be useful for different learning methods; for example, if $\textsc{Score}(\mathbf{x}, p)$ produces log-probability values derived from a generative model, then our parsing algorithms would recover the maximum-probability dependency trees under this model.

We define the *order* of a part according to the number of dependencies it contains, with analogous terminology for factorizations and parsing algorithms. In the remainder of this chapter, we will focus on factorizations utilizing the following parts:



Specifically, Sections 5.4.1, 5.4.2, and 5.4.3 describe parsers that, respectively, factor trees into grandchild parts, grand-sibling parts, and a mixture of grand-sibling and tri-sibling parts.

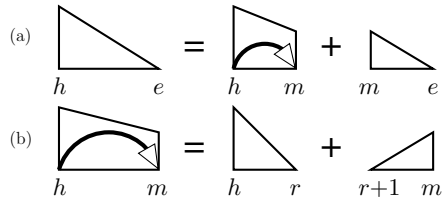


Figure 5-1: The dynamic-programming structures and derivations of the Eisner (2000) algorithm. Complete spans are depicted as triangles and incomplete spans as trapezoids. For brevity, we elide the symmetric right-headed versions.

5.3 Existing parsing algorithms

Our new third-order dependency parsers build on ideas from existing parsing algorithms. In this section, we provide background on two relevant parsers from previous work.

5.3.1 First-order factorization

The first type of parser we describe uses a "first-order" factorization, which decomposes a dependency tree into its individual dependencies. Eisner (2000) introduced a widely-used dynamic-programming algorithm for first-order parsing; as it is the basis for many parsers, including our new algorithms, we summarize its design here.

The Eisner (2000) algorithm is based on two interrelated types of dynamic-programming structures: *complete* spans, which consist of a headword and its descendents on one side, and *incomplete* spans, which consist of a dependency and the region between the head and modifier.

Formally, we denote a complete span as $C_{h,e}$ where h and e are the indices of the span's headword and endpoint. An incomplete span is denoted as $I_{h,m}$ where h and m are the index of the head and modifier of a dependency. Intuitively, a complete span represents a "half-constituent" headed by h, whereas an incomplete span is only a partial half-constituent, since the constituent can be extended by adding more

[1] For examples of how performance varies with the degree of the parser's factorization see, e.g., McDonald and Pereira (2006, Tables 1 and 2), Carreras (2007, Table 2), Koo et al. (2008, Tables 2 and 4), or Suzuki et al. (2009, Tables 3–6).

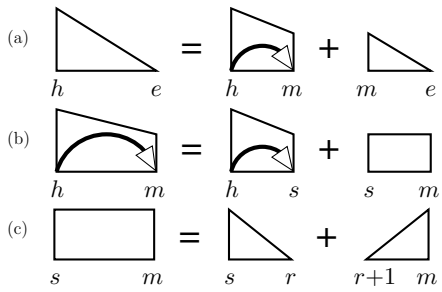


Figure 5-2: The dynamic-programming structures and derivations of the second-order sibling parser; sibling spans are depicted as boxes. For brevity, we elide the right-headed versions.

modifiers to m.

Each type of span is created by recursively combining two smaller, adjacent spans; the constructions are specified graphically in Figure 5-1. An incomplete span is constructed from a pair of complete spans, indicating the division of the range $[h, m]$ into constituents headed by h and m. A complete span is created by "completing" an incomplete span with the other half of m's constituent. The point of concatenation in each construction—m in Figure 5-1(a) or r in Figure 5-1(b)—is the *split point*, a free index that must be enumerated to find the optimal construction.

In order to parse a sentence $\mathbf{x}$, it suffices to find optimal constructions for all complete and incomplete spans defined on $\mathbf{x}$. This can be accomplished by adapting standard chart-parsing techniques (Cocke and Schwartz, 1970; Younger, 1967; Kasami, 1965) to the recursive derivations defined in Figure 5-1. Since each derivation is defined by two fixed indices (the boundaries of the span) and a third free index (the split point), the parsing algorithm requires $O(n^3)$ time and $O(n^2)$ space (Eisner, 1996; McAllester, 1999).

5.3.2 Second-order sibling factorization

As remarked by Eisner (1996) and McDonald and Pereira (2006), it is possible to rearrange the dynamic-programming structures to conform to an improved factorization that decomposes each tree into *sibling* parts—pairs of dependencies with a shared head. Specifically, a sibling part consists of a triple of indices (h, m, s) where (h, m) and (h, s) are dependencies, and where s and m are successive modifiers to the same side of h.

In order to parse this factorization, the second-order parser introduces a third type of dynamic-programming structure: *sibling* spans, which represent the region between successive modifiers of some head. Formally, we denote a sibling span as $S_{s,m}$ where s and m are a pair of modifiers involved in a sibling relationship. Modified versions of sibling spans will play an important role in the new parsing algorithms described in Section 5.4.

Figure 5-2 provides a graphical specification of the second-order parsing algorithm. Note that incomplete spans are constructed in a new way: the second-order parser combines a smaller incomplete span, representing the next-innermost dependency, with a sibling span that covers the region between the two modifiers. Sibling parts (h, m, s) can thus be obtained from Figure 5-2(b). Despite the use of second-order parts, each derivation is still defined by a span and split point, so the parser requires $O(n^3)$ time and $O(n^2)$ space.

5.3.3 Carreras factorization

In this section, we briefly review the Carreras (2007) second-order factorization, which improves upon the sibling factorization by introducing *grandchild* parts—pairs of dependencies connected head-to-tail. Specifically, a grandchild part is defined as a triple of indices (h, m, c) where (h, m) and (m, c) are dependencies. An important limitation of the Carreras (2007) factorization is that it only defines grandchild parts for the outermost dependencies; i.e., (h, m, c) is examined only if c is the outermost modifier of m in some direction. In Section 5.4, we will describe parsing algorithms

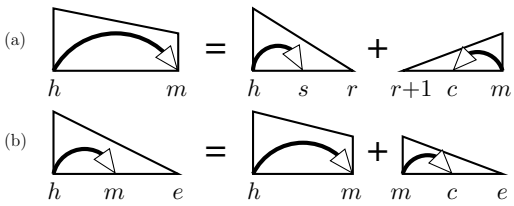


Figure 5-3: The dynamic-programming structures and derivations of the second-order Carreras (2007) algorithm.

in which this restriction is lifted.

Like the first-order factorization, the parsing algorithm for this factorization is based on complete and incomplete spans. However, the complete spans are augmented with a third index giving the identity of the outermost modifier of the head of the complete span. These extra indices define grandchild parts (h, m, c) in Figure 5-3(a,b), as well as sibling parts (h, m, s) in Figure 5-3(a). The modeling of grandchild parts comes with a cost: the parsing algorithm requires $O(n^4)$ time and $O(n^3)$ space, a factor of $O(n)$ increase in both quantities. Note that even though there are five indices in Figure 5-3(a), seemingly implying an $O(n^5)$ runtime, the indices s and c are independent of each other and the algorithm can be optimized to deal with each index separately; thus they effectively act as one index. The increase in space complexity arises from the fact that each complete structure has three indices. Despite the higher computational complexity of the Carreras (2007) parser, as compared to the previous two parsing algorithms, this parser has been applied in several demanding situations (see, e.g., Koo et al., 2008; Carreras et al., 2008; Suzuki et al., 2009).

5.4 New third-order parsing algorithms

In this section we describe our new third-order dependency parsing algorithms. Our overall method is characterized by the augmentation of each span with a "grandparent" index: an index external to the span whose role will be made clear below. This

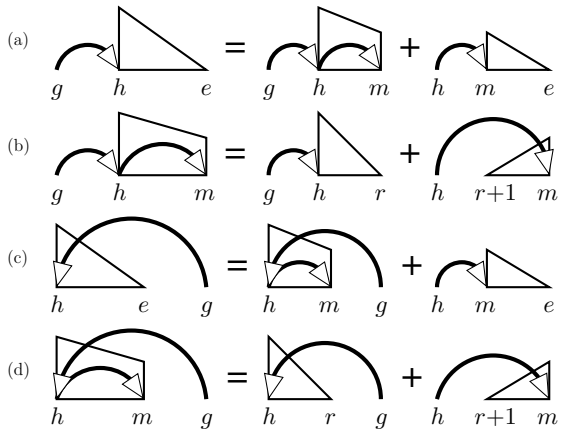


Figure 5-4: The dynamic-programming structures and derivations of Model 0. For brevity, we elide the right-headed versions. Note that (c) and (d) differ from (a) and (b) only in the position of g.

section presents three parsing algorithms based on this idea: Model 0, a second-order parser, and Models 1 and 2, which are third-order parsers.

5.4.1 Model 0: all grandchildren

The first parser, Model 0, factors each dependency tree into a set of *grandchild* parts—pairs of dependencies connected head-to-tail. Specifically, a grandchild part is a triple of indices (g, h, m) where (g, h) and (h, m) are dependencies.[2]

In order to parse this factorization, we augment both complete and incomplete spans with grandparent indices; for brevity, we refer to these augmented structures as *g-spans*. Formally, we denote a complete g-span as $C^g_{h,e}$, where $C_{h,e}$ is a normal complete span and g is an index lying outside the range $[h, e]$, with the implication

[2]The Carreras (2007) parser also uses grandchild parts but only in restricted cases; see Section 5.6 for details.

```
OptimizeAllSpans(x)
1.  ∀ g, i   C^g_{i,i} = 0                                   ◁ base case
2.  for w = 1 ... (n − 1)                                    ◁ span width
3.      for i = 1 ... (n − w)                                ◁ span start index
4.          j = i + w                                        ◁ span end index
5.          for g < i or g > j                               ◁ grandparent index
6.              I^g_{i,j} = max_{i≤r<j} {C^g_{i,r} + C^i_{j,r+1}} +
                            ScoreG(x, g, i, j)
7.              I^g_{j,i} = max_{i≤r<j} {C^g_{j,r+1} + C^j_{i,r}} +
                            ScoreG(x, g, j, i)
8.              C^g_{i,j} = max_{i<m≤j} {I^g_{i,m} + C^i_{m,j}}
9.              C^g_{j,i} = max_{i≤m<j} {I^g_{j,m} + C^j_{m,i}}
10.         endfor
11.     endfor
12. endfor
```

Figure 5-5: A pseudocode sketch for a bottom-up chart parser for Model 0. ScoreG is the scoring function for grandchild parts. We use the g-span identities as shorthand for their chart entries (e.g., $I^g_{i,j}$ refers to the entry containing the maximum score of that g-span).

that (g, h) is a dependency. Incomplete g-spans are defined analogously and are denoted as $I^g_{h,m}$.

Figure 5-4 depicts complete and incomplete g-spans and provides a graphical specification of the Model 0 dynamic-programming algorithm. The algorithm resembles the first-order parser, except that every recursive construction must also set the grandparent indices of the smaller g-spans; fortunately, this can be done deterministically in all cases. For example, Figure 5-4(a) depicts the decomposition of $C^g_{h,e}$ into an incomplete half and a complete half. The grandparent of the incomplete half is copied from $C^g_{h,e}$ while the grandparent of the complete half is set to h, the head of m as defined by the construction. Clearly, grandchild parts (g, h, m) can be read off of the incomplete g-spans in Figure 5-4(b,d). Moreover, since each derivation copies the grandparent index g into successively smaller g-spans, grandchild parts will be produced for *all* grandchildren of g.

Model 0 can be parsed by adapting standard top-down or bottom-up chart parsing

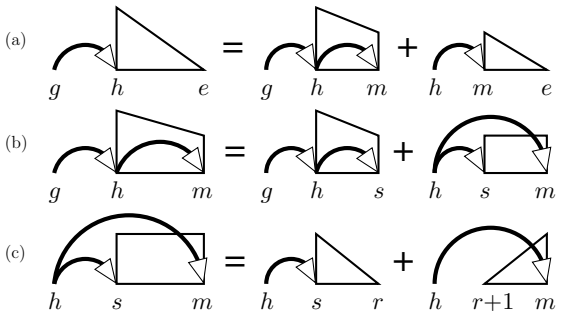


Figure 5-6: The dynamic-programming structures and derivations of Model 1. Right-headed and right-grandparented versions are omitted.

techniques. For concreteness, Figure 5-5 provides a pseudocode sketch of a bottom-up chart parser for Model 0; although the sketch omits many details, it suffices for the purposes of illustration. The algorithm progresses from small widths to large in the usual manner, but after defining the endpoints (i, j) there is an additional loop that enumerates all possible grandparents. Since each derivation is defined by three fixed indices (the g-span) and one free index (the split point), the complexity of the algorithm is $O(n^4)$ time and $O(n^3)$ space.

Note that the grandparent indices cause each g-span to have non-contiguous structure. For example, in Figure 5-4(a) the words between g and h will be controlled by some other g-span. Due to these discontinuities, the correctness of the Model 0 dynamic-programming algorithm may not be immediately obvious. While we do not provide a full proof of correctness here, we note that each structure on the right-hand side of Figure 5-4 lies completely within the structure on the left-hand side. This nesting of structures implies, in turn, that the usual properties required to ensure the correctness of dynamic programming hold.

5.4.2 Model 1: all grand-siblings

We now describe our first third-order parsing algorithm. Model 1 decomposes each tree into a set of *grand-sibling* parts—combinations of sibling parts and grandchild parts. Specifically, a grand-sibling is a 4-tuple of indices (g, h, m, s) where (h, m, s) is a sibling part and (g, h, m) and (g, h, s) are grandchild parts.

In order to parse this factorization, we introduce sibling g-spans $S^h_{m,s}$, which are composed of a normal sibling span $S_{m,s}$ and an external index h, with the implication that (h, m, s) forms a valid sibling part. Figure 5-6 provides a graphical specification of the dynamic-programming algorithm for Model 1. The overall structure of the algorithm resembles the second-order sibling parser, with the addition of grandparent indices; as in Model 0, the grandparent indices can be set deterministically in all cases. Note that the sibling g-spans are crucial: they allow grand-sibling parts (g, h, m, s) to be read off of Figure 5-6(b), while simultaneously propagating grandparent indices to smaller g-spans.

Like Model 0, Model 1 can be parsed via adaptations of standard chart-parsing techniques; we omit the details for brevity. Despite the move to third-order parts, each derivation is still defined by a g-span and a split point, so that parsing requires only $O(n^4)$ time and $O(n^3)$ space.

5.4.3 Model 2: grand-siblings and tri-siblings

Higher-order parsing algorithms have been proposed which extend the second-order sibling factorization to parts containing multiple siblings (McDonald and Pereira, 2006, also see Section 5.6 for discussion). In this section, we show how our g-span-based techniques can be combined with a third-order sibling parser, resulting in a parser that captures both grand-sibling parts and *tri-sibling* parts—4-tuples of indices (h, m, s, t) such that both (h, m, s) and (h, s, t) are sibling parts.

In order to parse this factorization, we introduce a new type of dynamic-programming structure: sibling-augmented spans, or *s-spans*. Formally, we denote an incomplete s-span as $I_{h,m,s}$ where $I_{h,m}$ is a normal incomplete span and s is an index lying in the

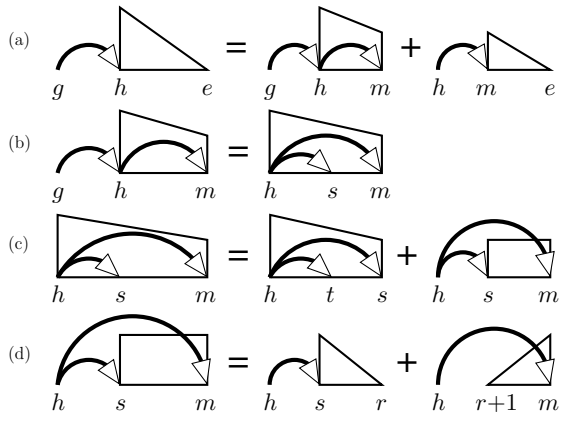


Figure 5-7: The dynamic-programming structures and derivations of Model 2. Right-headed and right-grandparented versions are omitted.

strict interior of the range $[h, m]$, such that (h, m, s) forms a valid sibling part.

Figure 5-7 provides a graphical specification of the Model 2 parsing algorithm. An incomplete s-span is constructed by combining a smaller incomplete s-span, representing the next-innermost pair of modifiers, with a sibling g-span, covering the region between the outer two modifiers. As in Model 1, sibling g-spans are crucial for propagating grandparent indices, while allowing the recovery of tri-sibling parts (h, m, s, t). Figure 5-7(b) shows how an incomplete s-span can be converted into an incomplete g-span by exchanging the internal sibling index for an external grandparent index; in the process, grand-sibling parts (g, h, m, s) are enumerated. Since every derivation is defined by an augmented span and a split point, Model 2 can be parsed in $O(n^4)$ time and $O(n^3)$ space.

It should be noted that unlike Model 1, Model 2 produces grand-sibling parts only for the outermost pair of grandchildren,[3] similar to the behavior of the Carreras (2007)

[3]The reason for the restriction is that in Model 2, grand-siblings can only be derived via Figure 5-7(b), which does not recursively copy the grandparent index for reuse in smaller g-spans

parser. In fact, the resemblance is more than passing, as Model 2 can *emulate* the Carreras (2007) algorithm by "demoting" each third-order part into a second-order part:

$$\textsc{ScoreGS}(\mathbf{x}, g, h, m, s) = \textsc{ScoreG}(\mathbf{x}, g, h, m)$$
$$\textsc{ScoreS3}(\mathbf{x}, h, m, s, t) = \textsc{ScoreS}(\mathbf{x}, h, m, s)$$

where ScoreG, ScoreS, ScoreGS and ScoreS3 are the scoring functions for grandchildren, siblings, grand-siblings and tri-siblings, respectively. The emulated version has the same computational complexity as the original, so there is no practical reason to prefer it over the original. Nevertheless, the relationship illustrated above highlights the efficiency of our approach: we are able to recover third-order parts in place of second-order parts, at no additional cost.

5.4.4 Discussion

The technique of grandparent-index augmentation has proven fruitful, as it allows us to parse expressive third-order factorizations while retaining an efficient $O(n^4)$ runtime. In fact, our third-order parsing algorithms are "optimally" efficient in an asymptotic sense. Since each third-order part is composed of four separate indices, there are $\Theta(n^4)$ distinct parts. Any third-order parsing algorithm must at least consider the score of each part, hence third-order parsing is $\Omega(n^4)$ and it follows that the asymptotic complexity of Models 1 and 2 cannot be improved.

The key to the efficiency of our approach is a fundamental asymmetry in the structure of a directed tree: a head can have any number of modifiers, while a modifier always has exactly one head. Factorizations like that of Carreras (2007) obtain grandchild parts by augmenting spans with the indices of modifiers, leading to limitations on the grandchildren that can participate in the factorization. Our method, by "inverting" the modifier indices into grandparent indices, exploits the structural asymmetry.

As a final note, the parsing algorithms described in this section fall into the category of *projective* dependency parsers, which forbid crossing dependencies. If crossing

as Model 1 does in Figure 5-6(b).

dependencies are allowed, it is possible to parse a first-order factorization by finding the maximum directed spanning tree (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005b). Unfortunately, designing efficient higher-order non-projective parsers is likely to be challenging, based on recent hardness results (McDonald and Pereira, 2006; McDonald and Satta, 2007).

5.5 Extensions

We briefly outline a few extensions to our algorithms; we hope to explore these in future work.

5.5.1 Probabilistic inference

Many statistical modeling techniques are based on partition functions and marginals—summations over the set of possible trees $\mathcal{Y}(\mathbf{x})$. Straightforward adaptations of the inside-outside algorithm (Baker, 1979) to our dynamic-programming structures would suffice to compute these quantities.

5.5.2 Labeled parsing

Our parsers are easily extended to labeled dependencies. Direct integration of labels into Models 1 and 2 would result in third-order parts composed of three labeled dependencies, at the cost of increasing the time and space complexities by factors of $O(L^3)$ and $O(L^2)$, respectively, where L bounds the number of labels per dependency.

5.5.3 Word senses

If each word in $\mathbf{x}$ has a set of possible "senses," our parsers can be modified to recover the best joint assignment of syntax and senses for $\mathbf{x}$, by adapting methods in Eisner (2000). Complexity would increase by factors of $O(S^4)$ time and $O(S^3)$ space, where S bounds the number of senses per word.

5.5.4 Increased context

If more vertical context is desired, the dynamic-programming structures can be extended with additional ancestor indices, resulting in a "spine" of ancestors above each span. Each additional ancestor lengthens the vertical scope of the factorization (e.g., from grand-siblings to "great-grand-siblings"), while increasing complexity by a factor of $O(n)$. Horizontal context can also be increased by adding internal sibling indices; each additional sibling widens the scope of the factorization (e.g., from grand-siblings to "grand-tri-siblings"), while increasing complexity by a factor of $O(n)$.

5.6 Related work

Our method augments each span with the index of the head that governs that span, in a manner superficially similar to parent annotation in CFGs (Johnson, 1998). However, parent annotation is a *grammar* transformation that is independent of any particular sentence, whereas our method annotates spans with *indices* into the current sentence. These indices allow the use of arbitrary features predicated on the position of the grandparent (e.g., word identity, POS tag, contextual POS tags) without affecting the asymptotic complexity of the parsing algorithm. Efficiently encoding this kind of information into a sentence-independent grammar transformation would be challenging at best.

Eisner (2000) defines dependency parsing models where each word has a set of possible "senses" and the parser recovers the best joint assignment of syntax and senses. Our new parsing algorithms could be implemented by defining the "sense" of each word as the index of its head. However, when parsing with senses, the complexity of the Eisner (2000) parser increases by factors of $O(S^3)$ time and $O(S^2)$ space (ibid., Section 4.2). Since each word has n potential heads, a direct application of the word-sense parser leads to time and space complexities of $O(n^6)$ and $O(n^4)$, respectively, in contrast to our $O(n^4)$ and $O(n^3)$.[4]

[4] In brief, the reason for the inefficiency is that the word-sense parser is unable to exploit certain constraints, such as the fact that the endpoints of a sibling g-span must have the same head. The

Eisner (2000) also uses head automata to score or recognize the dependents of each head. An interesting question is whether these automata could be coerced into modeling the grandparent indices used in our parsing algorithms. However, note that the head automata are defined in a sentence-independent manner, with two automata per word in the vocabulary (ibid., Section 2). The automata are thus analogous to the rules of a CFG and attempts to use them to model grandparent indices would face difficulties similar to those already described for grammar transformations in CFGs.

It should be noted that third-order parsers have previously been proposed by McDonald and Pereira (2006), who remarked that their second-order sibling parser (see Figure 5-2) could easily be extended to capture $m > 1$ successive modifiers in $O(n^{m+1})$ time (ibid., Section 2.2). To our knowledge, however, Models 1 and 2 are the first third-order parsing algorithms capable of modeling grandchild parts. In our experiments, we find that grandchild interactions make important contributions to parsing performance (see Table 5.3).

Carreras (2007) presents a second-order parser that can score both sibling and grandchild parts, with complexities of $O(n^4)$ time and $O(n^3)$ space. An important limitation of the parser's factorization is that it only defines grandchild parts for outermost grandchildren: (g, h, m) is scored only when m is the outermost modifier of h in some direction. Note that Models 1 and 2 have the same complexity as Carreras (2007), but strictly greater expressiveness: for each sibling or grandchild part used in the Carreras (2007) factorization, Model 1 defines an enclosing grand-sibling, while Model 2 defines an enclosing tri-sibling or grand-sibling.

The factored parsing approach we focus on is sometimes referred to as "graph-based" parsing; a popular alternative is "transition-based" parsing, in which trees are constructed by making a series of incremental decisions (Yamada and Matsumoto, 2003; Attardi, 2006; Nivre et al., 2006; McDonald and Nivre, 2007). Transition-based parsers do not impose factorizations, so they can define arbitrary features on the tree as it is being built. As a result, however, they rely on greedy or approximate algorithms to search for the highest-scoring tree.

word-sense parser would needlessly enumerate all possible pairs of heads in this case.

5.7 Parsing experiments

In order to evaluate the effectiveness of our parsers in practice, we apply them to the Penn WSJ Treebank (Marcus et al., 1993) and the Prague Dependency Treebank (Hajič et al., 2001; Hajič, 1998).[5] We use standard training, validation, and test splits[6] to facilitate comparisons. Accuracy is measured with unlabeled attachment score (UAS): the percentage of words with the correct head.[7]

5.7.1 Features for third-order parsing

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

$$\textsc{ScorePart}(\mathbf{x}, p) \;=\; \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}, p)$$

Here, $\boldsymbol{\phi}$ is a feature-vector mapping and $\mathbf{w}$ is a vector of associated parameters. Following standard practice for higher-order dependency parsing (McDonald and Pereira, 2006; Carreras, 2007), Models 1 and 2 evaluate not only the relevant third-order parts, but also the lower-order parts that are implicit in their third-order factorizations. For example, Model 1 defines feature mappings for dependencies, siblings, grandchildren,

[5]For English, we extracted dependencies using Joakim Nivre's Penn2Malt tool with standard head rules (Yamada and Matsumoto, 2003); for Czech, we "projectivized" the training data by finding best-match projective trees.

[6]For Czech, the PDT has a predefined split; for English, we split the Sections as: 2–21 training, 22 validation, 23 test.

[7]As in previous work, English evaluation ignores any token whose gold-standard POS tag is one of {`` '' : , .}.

and grand-siblings, so that the score of a dependency parse is given by:

$$\begin{aligned}\textsc{Model1Score}(\mathbf{x}, y) \;=\; & \\ & \sum_{(h,m)\in y} \mathbf{w}_{\text{dep}} \cdot \boldsymbol{\phi}_{\text{dep}}(\mathbf{x}, h, m) \\ & \sum_{(h,m,s)\in y} \mathbf{w}_{\text{sib}} \cdot \boldsymbol{\phi}_{\text{sib}}(\mathbf{x}, h, m, s) \\ & \sum_{(g,h,m)\in y} \mathbf{w}_{\text{gch}} \cdot \boldsymbol{\phi}_{\text{gch}}(\mathbf{x}, g, h, m) \\ & \sum_{(g,h,m,s)\in y} \mathbf{w}_{\text{gsib}} \cdot \boldsymbol{\phi}_{\text{gsib}}(\mathbf{x}, g, h, m, s)\end{aligned}$$

Above, y is simultaneously decomposed into several different types of parts; trivial modifications to the Model 1 parser allow it to evaluate all of the necessary parts in an interleaved fashion. A similar treatment of Model 2 yields five feature mappings: the four above plus $\boldsymbol{\phi}_{\text{tsib}}(\mathbf{x}, h, m, s, t)$, which represents tri-sibling parts.

The lower-order feature mappings $\boldsymbol{\phi}_{\text{dep}}$, $\boldsymbol{\phi}_{\text{sib}}$, and $\boldsymbol{\phi}_{\text{gch}}$ are based on feature sets from previous work (McDonald et al., 2005a; McDonald and Pereira, 2006; Carreras, 2007), to which we added lexicalized versions of several features. For example, $\boldsymbol{\phi}_{\text{dep}}$ contains lexicalized "in-between" features that depend on the head and modifier words as well as a word lying in between the two; in contrast, previous work has generally defined in-between features for POS tags only. As another example, our second-order mappings $\boldsymbol{\phi}_{\text{sib}}$ and $\boldsymbol{\phi}_{\text{gch}}$ define lexical trigram features, while previous work has generally used POS trigrams only.

Our third-order feature mappings $\boldsymbol{\phi}_{\text{gsib}}$ and $\boldsymbol{\phi}_{\text{tsib}}$ consist of four types of features. First, we define *4-gram features* that characterize the four relevant indices using words and POS tags; examples include POS 4-grams and mixed 4-grams with one word and three POS tags. Second, we define *4-gram context features* consisting of POS 4-grams augmented with adjacent POS tags: for example, $\boldsymbol{\phi}_{\text{gsib}}(\mathbf{x}, g, h, m, s)$ includes POS 7-grams for the tags at positions $(g, h, m, s, g{+}1, h{+}1, m{+}1)$. Third, we define *backed-off features* that track bigram and trigram interactions which are absent in the lower-order feature mappings: for example, $\boldsymbol{\phi}_{\text{tsib}}(\mathbf{x}, h, m, s, t)$ contains features

predicated on the trigram (m, s, t) and the bigram (m, t), neither of which exist in any lower-order part. Fourth, noting that coordinations are typically annotated as grand-siblings, we define *coordination features* for certain grand-sibling parts. For example, $\boldsymbol{\phi}_{\text{gsib}}(\mathbf{x}, g, h, m, s)$ contains features examining the implicit head-modifier relationship (g, m) that are only activated when the POS tag of s is a coordinating conjunction.

Finally, we make two brief remarks regarding the use of POS tags. First, we assume that input sentences have been automatically tagged in a pre-processing step.[8] Second, for any feature that depends on POS tags, we include *two* copies of the feature: one using normal POS tags and another using coarsened versions[9] of the POS tags.

5.7.2 Averaged perceptron training

There are a wide variety of parameter estimation methods for structured linear models, such as log-linear models (Lafferty et al., 2001) and max-margin models (Taskar et al., 2003). We chose the averaged structured perceptron (Freund and Schapire, 1999; Collins, 2002) as it combines highly competitive performance with fast training times, typically converging in 5–10 iterations. We train each parser for 10 iterations and select parameters from the iteration that achieves the best score on the validation set.

5.7.3 Coarse-to-fine pruning

In order to decrease training times, we follow Carreras et al. (2008) and eliminate unlikely dependencies using a form of coarse-to-fine pruning (Charniak and Johnson,

[8]For Czech, the PDT provides automatic tags; for English, we used MXPOST (Ratnaparkhi, 1996) to tag validation and test data, with 10-fold cross-validation on the training set. Note that the reliance on POS-tagged input can be relaxed slightly by treating POS tags as word senses; see Section 5.5.3 and McDonald (2006, Table 6.1).

[9]For Czech, we used the first character of the tag; for English, we used the first two characters, except `PRP` and `PRP$`.

Beam	Pass	Orac	Acc1	Acc2	Time1	Time2
0.0001	26.5	99.92	93.49	93.49	49.6m	73.5m
0.001	16.7	99.72	93.37	93.29	25.9m	24.2m
0.01	9.1	99.19	93.26	93.16	6.7m	7.9m

Table 5.1: Effect of the marginal-probability beam on English parsing. For each beam value, parsers were trained on the English training set and evaluated on the English validation set; the same beam value was applied to both training and validation data. **Pass** = %dependencies surviving the beam in training data, **Orac** = maximum achievable UAS on validation data, **Acc1**/**Acc2** = UAS of Models 1/2 on validation data, and **Time1**/**Time2** = minutes per perceptron training iteration for Models 1/2, averaged over all 10 iterations. For perspective, the English training set has a total of 39,832 sentences and 950,028 words. A beam of 0.0001 was used in all experiments outside this table.

2005; Petrov and Klein, 2007). In brief, we train a log-linear first-order parser[10] and for every sentence $\mathbf{x}$ in training, validation, and test data we compute the marginal probability $P(h, m \,|\, \mathbf{x})$ of each dependency. Our parsers are then modified to ignore any dependency (h, m) whose marginal probability is below $0.0001 \times \max_{h'} P(h', m \,|\, \mathbf{x})$. Table 5.1 provides information on the behavior of the pruning method.

5.7.4 Main results

Table 5.2 lists the accuracy of Models 1 and 2 on the English and Czech test sets, together with some relevant results from related work.[11] The models marked "†" are *not* directly comparable to our work as they depend on additional sources of information that our models are trained without—unlabeled data in the case of Koo et al. (2008) and Suzuki et al. (2009) and phrase-structure annotations in the case of Carreras et al. (2008). All three of the "†" models are based on versions of the Carreras (2007) parser, so modifying these methods to work with our new third-order parsing algorithms would be an interesting topic for future research. For example,

[10]For English, we generate marginals using a projective parser (Baker, 1979; Eisner, 2000); for Czech, we generate marginals using a non-projective parser (Smith and Smith, 2007; McDonald and Satta, 2007; Koo et al., 2007). Parameters for these models are obtained by running exponentiated gradient training for 10 iterations (Collins et al., 2008).

[11]Model 0 was not tested as its factorization is a strict subset of the factorization of Model 1.

Parser	Eng	Cze
McDonald et al. (2005a,2005b)	90.9	84.4
McDonald and Pereira (2006)	91.5	85.2
Koo et al. (2008), standard	92.02	86.13
Model 1	93.04	87.38
Model 2	92.93	87.37
Koo et al. (2008), semi-sup†	93.16	87.13
Suzuki et al. (2009)†	93.79	88.05
Carreras et al. (2008)†	93.5	—

Table 5.2: UAS of Models 1 and 2 on test data, with relevant results from related work. Note that Koo et al. (2008) is listed with standard features and semi-supervised features. †: see main text.

Models 1 and 2 obtain results comparable to the semi-supervised parsers of Koo et al. (2008), and additive gains might be realized by applying their cluster-based feature sets to our enriched factorizations.

5.7.5 Ablation studies

In order to better understand the contributions of the various feature types, we ran additional ablation experiments; the results are listed in Table 5.3, in addition to the scores of Model 0 and the emulated Carreras (2007) parser (see Section 5.4.3). Interestingly, grandchild interactions appear to provide important information: for example, when Model 2 is used without grandchild-based features ("Model 2, no-G" in Table 5.3), its accuracy suffers noticeably. In addition, it seems that grandchild interactions are particularly useful in Czech, while sibling interactions are less important: consider that Model 0, a second-order grandchild parser with no sibling-based features, can easily outperform "Model 2, no-G," a *third*-order sibling parser with no grandchild-based features.

5.8 Conclusion

We have presented new parsing algorithms that are capable of efficiently parsing third-order factorizations, including both grandchild and sibling interactions. There are

Parser	Eng	Cze
Model 0	93.07	87.39
Carreras (2007) emulation	93.14	87.25
Model 1	93.49	87.64
Model 1, no-3$^{\text{rd}}$	93.17	87.57
Model 2	93.49	87.46
Model 2, no-3$^{\text{rd}}$	93.20	87.43
Model 2, no-G	92.92	86.76

Table 5.3: UAS for modified versions of our parsers on validation data. The term **no-3$^{\text{rd}}$** indicates a parser that was trained and tested with the third-order feature mappings $\boldsymbol{\phi}_{\text{gsib}}$ and $\boldsymbol{\phi}_{\text{tsib}}$ deactivated, though lower-order features were retained; note that "Model 2, no-3$^{\text{rd}}$" is not identical to the Carreras (2007) parser as it defines grandchild parts for the pair of grandchildren. The term **no-G** indicates a parser that was trained and tested with the grandchild-based feature mappings $\boldsymbol{\phi}_{\text{gch}}$ and $\boldsymbol{\phi}_{\text{gsib}}$ deactivated; note that "Model 2, no-G" emulates the third-order sibling parser proposed by McDonald and Pereira (2006).

several possibilities for further research involving our third-order parsing algorithms. One idea would be to consider extensions and modifications of our parsers, some of which have been suggested in Sections 5.5 and 5.7.4. A second area for future work lies in applications of dependency parsing. While we have evaluated our new algorithms on standard parsing benchmarks, there are a wide variety of tasks that may benefit from the extended context offered by our third-order factorizations; for example, the 4-gram sub-structures enabled by our approach may be useful for dependency-based language modeling in machine translation (Shen et al., 2008).

Chapter 6

Conclusion

This chapter provides some brief concluding remarks and discusses topics for future research.

6.1 Summary of the Thesis

This thesis has explored three advances in the field of discriminative dependency parsing. First, we described a simple but effective method for augmenting the features of an existing parser with information derived from standard clustering algorithms (Brown et al., 1992). Second, we showed that the classic Matrix-Tree Theorem (Kirchhoff, 1847; Tutte, 1984) can be applied to the problem of non-projective dependency parsing, enabling both log-linear and max-margin parameter estimation. Finally, we presented novel third-order dependency parsing algorithms that are capable of evaluating expressive third-order parts while retaining computational complexity equivalent to existing second-order parsers.

6.2 Ideas for Future Work

One natural idea for future work would be to evaluate the effects of combining our various advances: e.g., training a third-order parser with cluster-based features and fast log-linear training via dual exponentiated gradient optimization. In order to real-

ize such a model, it would be necessary to overcome several engineering hurdles, such as the increased complexity of the feature set and implementation of the dynamic-programming algorithms required to compute the necessary partition functions and marginals. In addition, the potential for non-sparsity in the parameter vectors may pose a significant challenge due to the extremely high dimensionality of the feature space.

There are also a number of ideas that we have not been able to explore in this thesis; it could be fruitful to revisit some of these ideas in future work. To begin, the Brown et al. (1992) cluster hierarchies used in our semi-supervised parsing experiments, while demonstrably effective, exhibit some unusual and possibly undesirable properties—namely, the depth and frequency distribution of the clusters is highly variable. A more principled approach to working with the Brown clusters that attempts to address these issues might result in even greater increases in parsing performance.

We have also performed some initial studies on the application of split-merge training techniques (Petrov et al., 2006) to a hidden Markov model clustering approach (Baum et al., 1970). While these preliminary efforts have failed, the overall approach nevertheless remains attractive, as a successful implementation of the method would combine the hierarchical nature of the Brown clustering with the context-sensitive nature of clusterings based on hidden Markov models.

Based on interesting recent work in randomized ensembles for generative phrase-structure parsing (Petrov, 2010), a natural topic for future work would be the exploration of similar random ensembles for discriminative dependency parsing. One possible approach would be to obtain multiple clusterings extracted via randomized versions of the Brown algorithm; an ensemble of cluster-based parsers could then be trained, each of the parsers utilizing a cluster-based feature set predicated on a different random clustering.

Conversely, it would be interesting to develop cluster-based parsers that simultaneously depend on several independent clusterings. In addition to the randomized methods mentioned above, sets of clusterings could be extracted from text corpora drawn from several different domains, or by employing several different clustering al-

gorithms. However, defining cluster-based feature mappings that draw upon multiple clusterings would be a laborious task. Furthermore, if interactions between the different types of clusters are to be considered then the number of features that need to be processed would dramatically increase, rendering direct approaches computationally infeasible.

A potential resolution to the issues above may be found by shifting to kernelized dual representations: for instance, feature mappings based on arbitrary combinations of several available sources of information can easily be defined by applying a polynomial kernel. Dependency parsers utilizing dual representations would face their own challenges, however, as the large datasets encountered in high-performance parsing applications produce prohibitively large numbers of potential support vectors. For example, the Penn Treebank (Marcus et al., 1993) training set contains almost 28 million potential dependencies; a kernel defined on pairs of dependencies would thus be associated with a kernel matrix of 28 million rows and columns, which would be impractical to calculate or even instantiate using current hardware. Viable methods for training dependency parsers with dual representations must therefore focus on online or incremental approaches (Collins, 2002; Shalev-Shwartz et al., 2007), especially those which explicitly attempt to reduce the number of support vectors participating in the classifier (Crammer et al., 2003; Dekel et al., 2008).

In a different vein, the proofs found in Appendix A demonstrate that the dual exponentiated gradient optimization algorithm has a fast rate of convergence for objective functions that satisfy certain conditions. As new computing architectures based on multi-core processors and distributed computing continue to rise in importance, interest is being shifted towards learning algorithms that can be parallelized easily. Considering recent results reported by McDonald et al. (2010), another possible avenue for future work would be to extend the convergence analysis of the dual exponentiated gradient algorithm to the case of parallelized training.

Finally, explorations into applications of dependency parsing algorithms provide another rich source of future work. Recent work in machine translation (Shen et al., 2008; Carreras and Collins, 2009) and semantic role labeling (Surdeanu et al., 2008;

Hajič et al., 2009) has made direct use of dependency parsing, and our improvements in parsing performance may prove useful in these domains. Note that while the performance of dependency parsers continues to increase, the best parsing techniques still fall short of the near-perfect accuracies obtained by, e.g., part-of-speech tagging. Thus, techniques which retain sets of alternative parses—such as reranking (Collins, 2000; Collins and Koo, 2005) or forest-based approaches (Huang, 2008; Mi et al., 2008)—may be important in these applications.

Appendix A

Rates of Convergence for Exponentiated Gradient Algorithms

This appendix provides proofs of fast rates of convergence for the dual exponentiated gradient optimization method of Collins et al. (2008) under certain conditions.

A.1 Preliminaries

In this section, we briefly summarize some results, primarily drawn from Collins et al. (2008), which are necessary in order to understand the proofs in this appendix.

A.1.1 Dual Optimization Problems

It is sometimes difficult or inconvenient to solve an optimization in its originally-posed, or *primal*, form. Provided that certain conditions are met, methods exist for defining equivalent *dual* optimization problems, which may exhibit more desirable properties. Before we move further, it should be noted that dual optimization is a rich and well-studied topic, and a full description is beyond the scope of this appendix. What follows is instead a greatly simplified presentation that focuses on the concrete

goal of defining the dual optimization problems for log-linear and max-margin models (Lebanon and Lafferty, 2002; Taskar et al., 2003; Collins et al., 2008).

We begin by restating, for convenience, the primal objective functions for log-linear and max-margin models:

$$\begin{aligned} f_{\text{LL}}(\mathbf{w}) &= \frac{C}{2}\left\|\mathbf{w}\right\|^2 - \sum_{i=1}^{n} \log P(y_i \,|\, \mathbf{x}_i; \mathbf{w}) \\ f_{\text{MM}}(\mathbf{w}) &= \frac{C}{2}\left\|\mathbf{w}\right\|^2 + \sum_{i=1}^{n} \xi(\mathbf{x}_i, y_i; \mathbf{w}) \end{aligned}$$

Estimating the primal parameters, $\mathbf{w}$, for a log-linear or max-margin model corresponds to minimizing the appropriate primal objective.

The dual objective functions for log-linear and max-margin models are defined in terms of dual parameters $\boldsymbol{\alpha}$, defined as follows:

$$\begin{aligned} \boldsymbol{\alpha} &= (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n) \\ \boldsymbol{\alpha}_i &= (\alpha_{i,y})_{y \in \mathcal{Y}(\mathbf{x}_i)} \end{aligned}$$

That is, the duals $\boldsymbol{\alpha}$ consist of a vector of n sub-vectors $\boldsymbol{\alpha}_i$—one $\boldsymbol{\alpha}_i$ for each training example $(\mathbf{x}_i, y_i)$—where each sub-vector in turn contains $|\mathcal{Y}(\mathbf{x}_i)|$ values $\alpha_{i,y}$—one $\alpha_{i,y}$ for each possible tree y in the set $\mathcal{Y}(\mathbf{x}_i)$. In both the log-linear and max-margin dual optimization problems, the dual parameters $\boldsymbol{\alpha}$ are subject to the following constraints:

$$\begin{aligned} \alpha_{i,y} &\geq 0 \qquad \forall i, y \\ \sum_{y \in \mathcal{Y}(\mathbf{x}_i)} \alpha_{i,y} &= 1 \qquad \forall i \end{aligned}$$

In other words, each sub-vector $\boldsymbol{\alpha}_i$ must form a distribution over the possible parses of the i^{th} example. For convenience, we define the set Δ_i as the set of distributions over trees for $\mathbf{x}_i$, so that $\boldsymbol{\alpha}_i \in \Delta_i$ for all i. We denote the set of vectors of distributions for all n training examples as

$$\Delta^n = \Delta_1 \times \cdots \times \Delta_n$$

Note that Δ^n describes the domain of the dual parameters—i.e., $\boldsymbol{\alpha} \in \Delta^n$.

The dual of the log-linear optimization problem can be derived via application of the convex conjugate (Jaakkola and Haussler, 1999; Lebanon and Lafferty, 2002), and is stated below:

$$Q_{\text{LL}}(\boldsymbol{\alpha}) \;=\; \frac{1}{2C}\left\|\mathbf{w}(\boldsymbol{\alpha})\right\|^2 + \sum_{i,y} \alpha_{i,y} \log \alpha_{i,y} \tag{A.1}$$

Here, $\mathbf{w}(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}^d$ is a function that converts a dual vector to a vector in the primal space, and is defined as follows:

$$\begin{aligned}\mathbf{w}(\boldsymbol{\alpha}) \;&=\; \sum_{i,y} \alpha_{i,y}\Big(\boldsymbol{\Phi}(\mathbf{x}_i, y_i) - \boldsymbol{\Phi}(\mathbf{x}_i, y)\Big) \\ &=\; \sum_{i}\left(\boldsymbol{\Phi}(\mathbf{x}_i, y_i) - \sum_{y} \alpha_{i,y}\boldsymbol{\Phi}(\mathbf{x}_i, y)\right)\end{aligned}$$

which is roughly analogous to the "empirical minus expected" computation that arises when taking the gradient of f_{LL}.

As mentioned in Section 2.4.3, many techniques for estimating the parameters of SVMs and max-margin models concentrate on dual optimization. The dual of the max-margin optimization problem can be constructed by applying Lagrange multipliers (Taskar et al., 2003, 2004), and is stated below:

$$Q_{\text{MM}}(\boldsymbol{\alpha}) \;=\; \frac{1}{2C}\left\|\mathbf{w}(\boldsymbol{\alpha})\right\|^2 - \sum_{i,y} \alpha_{i,y}\Delta(y_i, y) \tag{A.2}$$

where $\mathbf{w}(\boldsymbol{\alpha})$ is as defined above for Q_{LL}, and $\Delta(y_i, y)$ is the error function of the max-margin model (see Section 2.4.3).

Note that both dual objective functions are smooth and convex. In addition, the conversion function $\mathbf{w}(\boldsymbol{\alpha})$ has a special significance: if $\boldsymbol{\alpha}^\star$ is an optimal set of parameters for Q_{LL} or Q_{MM}, then $\mathbf{w}^\star = \frac{1}{C}\mathbf{w}(\boldsymbol{\alpha}^\star)$ is optimal for f_{LL} or f_{MM}, respectively.

A.1.2 Exponentiated Gradient Updates

Estimating the dual parameters for a log-linear or max-margin model corresponds to minimizing the appropriate dual objective function, with one important complication: the dual parameters must at all times satisfy the distributional constraints implied by $\boldsymbol{\alpha} \in \Delta^n$. In order to cope with these constraints we employ exponentiated gradient (EG) updates, introduced by Kivinen and Warmuth (1997) as an alternative to standard gradient-based methods. Given a learning rate $\eta > 0$ and a function $Q(\boldsymbol{\alpha})$, we formalize the EG update rule as a vector-valued function $\boldsymbol{\sigma}(\boldsymbol{\alpha}; \eta, Q) : \Delta^n \mapsto \Delta^n$.

$$\sigma_{i,y}(\boldsymbol{\alpha}; \eta, Q) \;=\; \frac{1}{Z(\boldsymbol{\alpha}_i; \eta, Q)}\alpha_{i,y} \exp\left\{-\eta \frac{\partial Q}{\partial \alpha_{i,y}}\right\}$$

Note that the gradient is negated as we are interested in minimization. In the above, $Z(\boldsymbol{\alpha}_i; \eta, Q)$ is a normalization constant defined as:

$$Z(\boldsymbol{\alpha}_i; \eta, Q) \;=\; \sum_{i,y} \alpha_{i,y} \exp\left\{-\eta \frac{\partial Q}{\partial \alpha_{i,y}}\right\}$$

Thanks to the explicit normalization, the EG update $\boldsymbol{\sigma}$ always produces a new set of duals that satisfies the distributional constraints.

Note that the update function depends on the learning rate η and optimization objective Q; however, in situations where the identities of η and Q are obvious from the context, we will simply write $\boldsymbol{\sigma}(\boldsymbol{\alpha})$. In addition, as the output of $\boldsymbol{\sigma}$ is an element of Δ^n we allow the use of subscripts to refer to the example-wise sub-vectors of the EG update; for example, $\boldsymbol{\sigma}_i(\boldsymbol{\alpha})$ denotes the result of performing an EG update on the i^{th} sub-vector $\boldsymbol{\alpha}_i$ only. We define two possible methods for applying EG updates:

Batch Updating In this situation, we update all of the dual sub-vectors $\boldsymbol{\alpha}_i$ simultaneously. Specifically, given that the current setting of the dual parameters is $\boldsymbol{\alpha}^t$, the updated duals $\boldsymbol{\alpha}^{t+1}$ are computed as:

$$\boldsymbol{\alpha}^{t+1} \;=\; \boldsymbol{\sigma}(\boldsymbol{\alpha}^t; \eta, Q)$$

(Randomized) Online Updating In this scheme, we update one randomly-chosen sub-vector at a time. Specifically, given that the current setting of the dual parameters is $\boldsymbol{\alpha}^t$, we first choose an index k uniformly at random from $\{1, \ldots, n\}$, and the updated duals $\boldsymbol{\alpha}^{t+1}$ are then computed as:

$$\boldsymbol{\alpha}^{t+1} \;=\; (\boldsymbol{\alpha}^t_1, \ldots, \boldsymbol{\alpha}^t_{k-1}, \boldsymbol{\sigma}_k(\boldsymbol{\alpha}^t; \eta, Q), \boldsymbol{\alpha}^t_{k+1}, \ldots, \boldsymbol{\alpha}^t_n)$$

For convenience, we define another update function $\boldsymbol{\sigma}(\boldsymbol{\alpha}, k; \eta, Q)$ which represents the result of updating only the k^{th} sub-vector of $\boldsymbol{\alpha}$:

$$\boldsymbol{\sigma}_i(\boldsymbol{\alpha}, k; \eta, Q) \;=\; \begin{cases} \boldsymbol{\sigma}_k(\boldsymbol{\alpha}; \eta, Q) & \text{if } i = k \\ \boldsymbol{\alpha}_i & \text{otherwise} \end{cases}$$

Note that the online EG update can now be rewritten as $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\sigma}(\boldsymbol{\alpha}^t, k; \eta, Q)$.

A.1.3 Relevant Definitions

Here, we briefly define some concepts that are required in the ensuing analysis. First, for any function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$, any index $i \in \{1, \ldots, n\}$, and any vector $\boldsymbol{\alpha} \in \Delta^n$ we define the *i^{th} sub-function* of Q as:

$$Q_{\boldsymbol{\alpha},i}(\boldsymbol{\rho}) \;=\; Q(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{i-1}, \boldsymbol{\rho}, \boldsymbol{\alpha}_{i+1}, \ldots, \boldsymbol{\alpha}_n)$$

where $\boldsymbol{\rho} \in \Delta_i$ is a distribution over the trees of the i^{th} example. In essence, $Q_{\boldsymbol{\alpha},i}(\boldsymbol{\rho})$ fixes all arguments of Q to values taken from $\boldsymbol{\alpha}$, except the i^{th} sub-vector. These sub-functions are necessary for the analysis of online-style EG optimization.

Next, for any convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$, we define the *Bregman divergence* with respect to Q (Bregman, 1967):

$$B_Q[\mathbf{u} \,\|\, \mathbf{v}] \;=\; Q(\mathbf{u}) - Q(\mathbf{v}) - \nabla Q(\mathbf{v}) \cdot (\mathbf{u} - \mathbf{v})$$

where $\mathbf{u}, \mathbf{v} \in \Delta^n$. Intuitively, the Bregman divergence measures the error between

Q and a linear approximation to Q—i.e., a tangent hyperplane. Provided that Q is convex, the Bregman divergence with respect to Q is always non-negative. Although the Bregman divergence is similar to a distance measure, in general it does not satisfy the technical requirements (it can be asymmetric and does not necessarily satisfy the triangle inequality).

A well-known special case of the Bregman divergence is the *KL-divergence*, which can be derived as the Bregman divergence with respect to the negative entropy function. We define two forms of the KL-divergence:

$$\begin{aligned} D_i[\mathbf{u}_i \,\|\, \mathbf{v}_i] \;&=\; \sum_y u_{i,y} \log \frac{u_{i,y}}{v_{i,y}} \\ D[\mathbf{u} \,\|\, \mathbf{v}] \;&=\; \sum_i D_i[\mathbf{u}_i \,\|\, \mathbf{v}_i] \end{aligned}$$

where $\mathbf{u}, \mathbf{v} \in \Delta^n$. The first definition evaluates a single pair of distributions, while the second definition evaluates two vectors of distributions.

Bregman divergences provide a useful method for establishing classes of convex functions Q; in our analysis, we require the following classifications:

- For any τ such that $\tau > 0$, we say that a convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$ is *τ-upper-bounded* if, for all $\mathbf{u}, \mathbf{v} \in \Delta^n$,

$$B_Q[\mathbf{u} \,\|\, \mathbf{v}] \;\leq\; \tau D[\mathbf{u} \,\|\, \mathbf{v}]$$

- For any μ, τ such that $\tau > \mu > 0$, we say that a convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$ is *(μ, τ)-bounded* if, for all $\mathbf{u}, \mathbf{v} \in \Delta^n$,

$$\mu D[\mathbf{u} \,\|\, \mathbf{v}] \;\leq\; B_Q[\mathbf{u} \,\|\, \mathbf{v}] \;\leq\; \tau D[\mathbf{u} \,\|\, \mathbf{v}]$$

- For any τ such that $\tau > 0$, we say that a convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$ is *τ-online-upper-bounded* if, for all $i \in \{1, \ldots, n\}$ and all $\mathbf{p}, \mathbf{q} \in \Delta_i$,

$$B_{Q_{\boldsymbol{\alpha},i}}[\mathbf{p} \,\|\, \mathbf{q}] \;\leq\; \tau D_i[\mathbf{p} \,\|\, \mathbf{q}]$$

- For any μ, τ such that $\tau > \mu > 0$, we say that a convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$ is (μ, τ)-*online-bounded* if it is τ-online-upper-bounded and, for all $\mathbf{u}, \mathbf{v} \in \Delta^n$,

$$\mu D[\mathbf{u} \,\|\, \mathbf{v}] \;\; \leq \;\; B_Q[\mathbf{u} \,\|\, \mathbf{v}]$$

  Note that for (μ, τ)-online-bounded functions, the lower bound pertains to B_Q while the upper bound is placed on $B_{Q_{\boldsymbol{\alpha},i}}$.

The definition of τ-online-upper-boundedness is predicated on the Bregman divergence with respect to the sub-functions $Q_{\boldsymbol{\alpha},i}$ rather than the entire Q. Each $Q_{\boldsymbol{\alpha},i}$ has greatly reduced variability as compared to Q, because it only depends on a single sub-vector as opposed to all sub-vectors simultaneously. In turn, this property allows tighter bounds τ to be placed on online variants of the EG algorithm, leading to an improved theoretical analysis. In addition, we have confirmed that online EG achieves much faster convergence than batch EG in practice (Collins et al., 2008).

A.1.4 Relevant Lemmata

In the interests of a self-contained description, this section restates several results from Collins et al. (2008). We begin with the following lemma, which concerns the Bregman divergence between a dual vector $\boldsymbol{\alpha}$ and an arbitrary reference vector $\mathbf{u}$.

Lemma A.1.1: *[From Collins et al. (2008), see also Kivinen and Warmuth (2001).] For any convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$, any comparison vector $\mathbf{u} \in \Delta^n$, and any initial vector $\boldsymbol{\alpha}^t$ in the interior of Δ^n, if $\boldsymbol{\alpha}' = \boldsymbol{\sigma}(\boldsymbol{\alpha})$ is derived from $\boldsymbol{\alpha}$ using the EG update with learning rate η then*

$$B_Q[\mathbf{u} \,\|\, \boldsymbol{\alpha}] + Q(\boldsymbol{\alpha}) - Q(\mathbf{u}) \;\; = \;\; \frac{1}{\eta}\Big(D[\mathbf{u} \,\|\, \boldsymbol{\alpha}] - D[\mathbf{u} \,\|\, \boldsymbol{\alpha}'] + D[\boldsymbol{\alpha} \,\|\, \boldsymbol{\alpha}']\Big) \qquad \text{(A.3)}$$

Proof: See Collins et al. (2008, Lemma 10 in Appendix A). ■

The following lemma shows that batch EG updates, for an appropriately-chosen learning rate, result in consistent improvement of the dual objective.

Lemma A.1.2: *[From Collins et al. (2008).] For any convex τ-upper-bounded function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$, any learning rate η satisfying $0 < \eta \leq \frac{1}{\tau}$, and any initial vector $\boldsymbol{\alpha}$, if $\boldsymbol{\alpha}' = \boldsymbol{\sigma}(\boldsymbol{\alpha})$ is derived from the EG update rule then*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}') \;\; \geq \;\; \frac{1}{\eta} D[\boldsymbol{\alpha} \,\|\, \boldsymbol{\alpha}'] \qquad \text{(A.4)}$$

Proof: See Collins et al. (2008, Lemma 2 in Section 5.3). ■

As an aside, the KL-divergence satisfies $D[\boldsymbol{\alpha} \,\|\, \boldsymbol{\alpha}'] \geq 0$ with $D[\boldsymbol{\alpha} \,\|\, \boldsymbol{\alpha}'] = 0$ only when $\boldsymbol{\alpha} = \boldsymbol{\alpha}'$; therefore, the dual objective will continue to improve (i.e., decrease) until the updates reach a fixed point. The following corollary extends the above lemma to the case of online EG updates.

Corollary A.1.3: *[From Collins et al. (2008).] For any convex τ-online-upper-bounded function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$, any learning rate η satisfying $0 < \eta \leq \frac{1}{\tau}$, and any initial vector $\boldsymbol{\alpha}$, if $\boldsymbol{\alpha}' = \boldsymbol{\sigma}(\boldsymbol{\alpha}, i)$ is derived by applying the online EG update rule to the i^{th} sub-vector of $\boldsymbol{\alpha}$ then*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}') \;\; \geq \;\; \frac{1}{\eta} D_i[\boldsymbol{\alpha}_i \,\|\, \boldsymbol{\alpha}'_i] \qquad \text{(A.5)}$$

In addition, if the online EG update is applied to a randomly-chosen index, we have

$$Q(\boldsymbol{\alpha}) - \mathbf{E}_k\big[Q(\boldsymbol{\sigma}(\boldsymbol{\alpha}, k))\big] \;\; \geq \;\; \frac{1}{\eta}\mathbf{E}_k\big[D_k[\boldsymbol{\alpha}_k \,\|\, \boldsymbol{\sigma}_k(\boldsymbol{\alpha}, k)]\big] \qquad \text{(A.6)}$$

where $\mathbf{E}_k[\cdot]$ indicates the expected value with respect to the (uniformly random) choice of the update index k.

Proof: For Eq. A.5, see Collins et al. (2008, Equation 19 in Appendix D); for Eq. A.6, the result immediately follows from invoking Eq. A.5 for all k. ■

Finally, we present the following lemma, which establishes bounds on the log-linear objective function.

Lemma A.1.4: *[From Collins et al. (2008).] The log-linear dual objective $Q_{\text{LL}}(\boldsymbol{\alpha})$ is (μ, τ)-bounded and (μ, τ)-online-bounded.*

Proof: See Section 5.4 and Lemma 7 of Collins et al. (2008). ■

A.2 $O(\log(\frac{1}{\epsilon}))$ Rate of Convergence for Batch EG

This section proves a fast rate of convergence for batch EG updating on (μ, τ)-bounded functions. The proof is a slightly modified version of the Proof of Lemma 6 in Collins et al. (2008), which was originally written by the current author.

Theorem A.2.1: *Suppose that the convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$ is (μ, τ)-bounded and the learning rate η satisfies $0 < \eta \leq \frac{1}{\tau}$. Let $\boldsymbol{\alpha}^1 \in \Delta^n$ be the initial setting of the dual parameters, with subsequent settings obtained via the batch EG update rule: $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\sigma}(\boldsymbol{\alpha}^t)$. Then after T rounds of batch EG updating, for any comparison vector $\mathbf{u} \in \Delta^n$*

$$Q(\boldsymbol{\alpha}^{T+1}) - Q(\mathbf{u}) \;\; \leq \;\; \frac{\exp\{-\eta\mu T\}}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^1]$$

Proof: From Lemma A.1.2 we have for all t

$$Q(\boldsymbol{\alpha}^t) - Q(\boldsymbol{\alpha}^{t+1}) \;\; \geq \;\; \frac{1}{\eta} D[\boldsymbol{\alpha}^t \,\|\, \boldsymbol{\alpha}^{t+1}]$$

Combining this result with Lemma A.1.1 gives

$$Q(\boldsymbol{\alpha}^{t+1}) - Q(\mathbf{u}) \;\; \leq \;\; \frac{1}{\eta}\Big(D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] - D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^{t+1}]\Big) - B_Q[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t]$$

Since $Q(\boldsymbol{\alpha})$ is (μ, τ)-bounded, we know that $B_Q[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] \geq \mu D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t]$ and we can apply $-B_Q[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] \leq -\mu D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t]$, resulting in

$$\begin{aligned} Q(\boldsymbol{\alpha}^{t+1}) - Q(\mathbf{u}) \;\; &\leq \;\; \frac{1}{\eta}\Big(D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] - D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^{t+1}]\Big) - \mu D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] \\ Q(\boldsymbol{\alpha}^{t+1}) - Q(\mathbf{u}) \;\; &\leq \;\; \frac{1 - \eta\mu}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] - \frac{1}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^{t+1}] \qquad \text{(A.7)} \end{aligned}$$

From Lemma A.1.2, we know that $Q(\boldsymbol{\alpha}^t)$ decreases monotonically with t, so that $Q(\boldsymbol{\alpha}^{T+1}) \leq Q(\boldsymbol{\alpha}^{t+1})$ for all $t \leq T$. Thus, if for some $t \leq T$ it occurs that $Q(\boldsymbol{\alpha}^{t+1}) - Q(\mathbf{u}) \leq 0$, then it follows that $Q(\boldsymbol{\alpha}^{T+1}) \leq Q(\boldsymbol{\alpha}^{t+1}) \leq Q(\mathbf{u})$ and Theorem A.2.1 trivially holds. Otherwise, it must be the case that $Q(\boldsymbol{\alpha}^{t+1}) - Q(\mathbf{u}) \geq 0$ for all $t \leq T$; substituting this into Eq. A.7, we get that for all $t \leq T$,

$$\begin{aligned} 0 \;\; &\leq \;\; \frac{1 - \eta\mu}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] - \frac{1}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^{t+1}] \\ D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^{t+1}] \;\; &\leq \;\; (1 - \eta\mu) D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^t] \end{aligned}$$

By repeatedly applying this inequality for $t = 1, \ldots, (T-1)$ we get

$$D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^T] \;\; \leq \;\; (1 - \eta\mu)^{T-1} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^1] \qquad \text{(A.8)}$$

Finally, we rewrite Eq. A.7 with the setting $t = T$:

$$\begin{aligned} Q(\boldsymbol{\alpha}^{T+1}) - Q(\mathbf{u}) \;\; &\leq \;\; \frac{1 - \eta\mu}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^T] - \frac{1}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^{T+1}] \\ Q(\boldsymbol{\alpha}^{T+1}) - Q(\mathbf{u}) \;\; &\leq \;\; \frac{(1 - \eta\mu)^T}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^1] \qquad \text{(A.9)} \\ Q(\boldsymbol{\alpha}^{T+1}) - Q(\mathbf{u}) \;\; &\leq \;\; \frac{\exp\{-\eta\mu T\}}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^1] \qquad \text{(A.10)} \end{aligned}$$

where Eq. A.9 eliminates the negative contribution of $-\frac{1}{\eta} D[\mathbf{u} \,\|\, \boldsymbol{\alpha}^{T+1}]$ and substitutes Eq. A.8, and Eq. A.10 applies the common inequality $\log(1 - x) \leq -x$. ■

Note that choosing the optimal comparison vector $\mathbf{u} = \boldsymbol{\alpha}^\star$ for Theorem A.2.1 directly proves a $O(\log(\frac{1}{\epsilon}))$ rate of convergence. From Lemma A.1.4, we know that the log-linear dual Q_{LL} is (μ, τ)-bounded, so we can conclude that batch EG updating results in a $O(\log(\frac{1}{\epsilon}))$ rate of convergence for log-linear optimization.

A.3 $O(\log(\frac{1}{\epsilon}))$ Rate of Convergence for Online EG

This section proves a fast rate of convergence for randomized online EG updating on (μ,τ)-online-bounded functions. The proof is a modified version of the Proof of Lemma 9 in Collins et al. (2008), which was originally written by the current author.

Theorem A.3.1: *Suppose that the convex function $Q(\boldsymbol{\alpha}) : \Delta^n \mapsto \mathbb{R}$ is (μ,τ)-online-bounded and the learning rate η satisfies $0 < \eta \leq \frac{1}{\tau}$. Let $\boldsymbol{\alpha}^1 \in \Delta^n$ be the initial setting of the dual parameters, with subsequent settings obtained via the randomized online EG update rule: $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\sigma}(\boldsymbol{\alpha}^t, k_t)$ where k_t is chosen uniformly at random. Then after T rounds of randomized online EG updating, for any comparison vector $\mathbf{u} \in \Delta^n$,*

$$\mathbf{E}_{\mathbf{k}}\Big[Q(\boldsymbol{\alpha}^{T+1})\Big] - Q(\mathbf{u}) \;\leq\; \exp\Big\{-T\frac{\eta\mu}{n}\Big\}\left(\frac{1}{\eta}D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^1] + Q(\boldsymbol{\alpha}^1) - Q(\boldsymbol{\alpha}^\star)\right)$$

where $\mathbf{k} = (k_1,\ldots,k_T)$ is the vector of random indices chosen by the algorithm, and $Q(\boldsymbol{\alpha}^\star)$ is the value of the objective at the optimum.

Proof: We begin with the following rearrangement of Eq. A.3, which holds for any $\mathbf{u},\boldsymbol{\alpha} \in \Delta^n$,

$$\begin{aligned} Q(\boldsymbol{\alpha}) - Q(\mathbf{u}) &= \frac{1}{\eta}\Big(D[\mathbf{u}\,\|\,\boldsymbol{\alpha}] - D[\mathbf{u}\,\|\,\boldsymbol{\sigma}(\boldsymbol{\alpha})] + D[\boldsymbol{\alpha}\,\|\,\boldsymbol{\sigma}(\boldsymbol{\alpha})]\Big) - B_Q[\mathbf{u}\,\|\,\boldsymbol{\alpha}] \\ &= \frac{1}{\eta}\sum_{i=1}^n\Big(D_i[\mathbf{u}_i\,\|\,\boldsymbol{\alpha}_i] - D_i[\mathbf{u}_i\,\|\,\boldsymbol{\sigma}_i(\boldsymbol{\alpha})] + D_i[\boldsymbol{\alpha}_i\,\|\,\boldsymbol{\sigma}_i(\boldsymbol{\alpha})]\Big) - B_Q[\mathbf{u}\,\|\,\boldsymbol{\alpha}] \end{aligned} \tag{A.11}$$

From the definition of the online EG update function $\boldsymbol{\sigma}(\boldsymbol{\alpha}, i)$, we see that for all i,

$$D_i[\mathbf{u}_i\,\|\,\boldsymbol{\alpha}_i] - D_i[\mathbf{u}_i\,\|\,\boldsymbol{\sigma}_i(\boldsymbol{\alpha})] \;=\; D[\mathbf{u}\,\|\,\boldsymbol{\alpha}] - D[\mathbf{u}\,\|\,\boldsymbol{\sigma}(\boldsymbol{\alpha}, i)]$$

Applying this identity and Corollary A.1.3 to Eq. A.11, we get

$$\begin{aligned} Q(\boldsymbol{\alpha}) - Q(\mathbf{u}) &\leq \frac{1}{\eta}\sum_{i=1}^n\Big(D[\mathbf{u}\,\|\,\boldsymbol{\alpha}] - D[\mathbf{u}\,\|\,\boldsymbol{\sigma}(\boldsymbol{\alpha}, i)]\Big) + \\ &\qquad \sum_{i=1}^n\Big(Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\sigma}(\boldsymbol{\alpha}, i))\Big) - B_Q[\mathbf{u}\,\|\,\boldsymbol{\alpha}] \\ Q(\boldsymbol{\alpha}) - Q(\mathbf{u}) &\leq \frac{n}{\eta}\left(D[\mathbf{u}\,\|\,\boldsymbol{\alpha}] - \sum_{i=1}^n\frac{1}{n}D[\mathbf{u}\,\|\,\boldsymbol{\sigma}(\boldsymbol{\alpha}, i)]\right) + \\ &\qquad n\left(Q(\boldsymbol{\alpha}) - \sum_{i=1}^n\frac{1}{n}Q(\boldsymbol{\sigma}(\boldsymbol{\alpha}, i))\right) - B_Q[\mathbf{u}\,\|\,\boldsymbol{\alpha}] \end{aligned}$$

Applying the definition of (μ,τ)-online-boundedness and recognizing the formula for expectation with respect to a uniform distribution, we obtain

$$\begin{aligned} Q(\boldsymbol{\alpha}) - Q(\mathbf{u}) \;\leq\; &\frac{n-\eta\mu}{\eta}D[\mathbf{u}\,\|\,\boldsymbol{\alpha}] - \frac{n}{\eta}\mathbf{E}_i\big[D[\mathbf{u}\,\|\,\boldsymbol{\sigma}(\boldsymbol{\alpha}, i)]\big] + \\ &n\Big(Q(\boldsymbol{\alpha}) - \mathbf{E}_i\big[Q(\boldsymbol{\sigma}(\boldsymbol{\alpha}, i))\big]\Big) \end{aligned} \tag{A.12}$$

Note that the above applies to any vector $\boldsymbol{\alpha} \in \Delta^n$. In particular, for any sequence of random indices $\mathbf{k} = (k_1,\ldots,k_T)$, and for any time-step $t \leq T$, Eq. A.12 is true for the dual parameters $\boldsymbol{\alpha} = \boldsymbol{\alpha}^t$. Averaging across all possible sequences of random index choices $\mathbf{k}$, we get that for any $t \leq T$,

$$\begin{aligned} \mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^t)\big] - Q(\mathbf{u}) \;\leq\; &\frac{n-\eta\mu}{\eta}\mathbf{E}_{\mathbf{k}}\big[D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^t]\big] - \frac{n}{\eta}\mathbf{E}_{\mathbf{k}}\big[D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^{t+1}]\big] + \\ &n\Big(\mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^t)\big] - \mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^{t+1})\big]\Big) \end{aligned}$$

In the above, we have transformed certain expectations of the form $\mathbf{E}_i[\ldots\boldsymbol{\sigma}(\boldsymbol{\alpha}, i)\ldots]$ into expectations $\mathbf{E}_{\mathbf{k}}[\ldots\boldsymbol{\alpha}^{t+1}\ldots]$. This transformation, while correct, does involve a subtle and lengthy argument; for the details see Collins et al. (2008, Proof of Lemma 8 in Appendix D). We now apply a slight rearrangement, obtaining

$$\begin{aligned} \mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^t)\big] - Q(\mathbf{u}) \;\leq\; &\frac{n-\eta\mu}{\eta}\mathbf{E}_{\mathbf{k}}\big[D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^t]\big] - \frac{n}{\eta}\mathbf{E}_{\mathbf{k}}\big[D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^{t+1}]\big] + \\ &n\Big(\mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^t)\big] - Q(\mathbf{u})\Big) - n\Big(\mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^{t+1})\big] - Q(\mathbf{u})\Big) \end{aligned} \tag{A.13}$$

where we have simply added and subtracted $nQ(\mathbf{u})$. For convenience, define

$$\mathcal{Q}^t \;\equiv\; \mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^t)\big] - Q(\mathbf{u}) \qquad \text{and} \qquad \mathcal{D}^t \;\equiv\; \frac{1}{\eta}\mathbf{E}_{\mathbf{k}}\big[D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^t]\big]$$

If for any $t \leq T$ it occurs that $\mathcal{Q}^t \leq 0$, then from Corollary A.1.3 we know that $\mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^t)\big]$ decreases monotonically and therefore $\mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^{T+1})\big] \leq \mathbf{E}_{\mathbf{k}}\big[Q(\boldsymbol{\alpha}^t)\big] \leq 0$, so that Theorem A.3.1 holds trivially. Otherwise, we assume that $\mathcal{Q}^t \geq 0$ for all $t \leq T$. Rewriting Eq. A.13 using the new notation, we get

$$\begin{aligned} \mathcal{Q}^t &\leq (n-\eta\mu)\mathcal{D}^t - n\mathcal{D}^{t+1} + n\mathcal{Q}^t - n\mathcal{Q}^{t+1} \\ n\mathcal{Q}^{t+1} + n\mathcal{D}^{t+1} &\leq (n-1)\mathcal{Q}^t + (n-\eta\mu)\mathcal{D}^t \end{aligned}$$

Given that $\eta \leq \frac{1}{\tau}$ and $\mu < \tau$, it follows that $\eta\mu \leq \frac{\mu}{\tau} < 1$. Consequently, we see that $n - 1 < n - \eta\mu$, leading to the following simplification:

$$\begin{aligned} n\Big(\mathcal{Q}^{t+1} + \mathcal{D}^{t+1}\Big) &\leq (n-\eta\mu)\Big(\mathcal{Q}^t + \mathcal{D}^t\Big) \\ \mathcal{Q}^{t+1} + \mathcal{D}^{t+1} &\leq \Big(1 - \frac{\eta\mu}{n}\Big)\Big(\mathcal{Q}^t + \mathcal{D}^t\Big) \end{aligned}$$

As in the proof of Theorem A.2.1, we can repeatedly apply this inequality, obtaining

$$\mathcal{Q}^{T+1} + \mathcal{D}^{T+1} \;\leq\; \Big(1 - \frac{\eta\mu}{n}\Big)^T\Big(\mathcal{Q}^1 + \mathcal{D}^1\Big)$$

Note that $\mathcal{Q}^{T+1} \leq \mathcal{Q}^{T+1} + \mathcal{D}^{T+1}$, as $\mathcal{D}^t \geq 0$ by the definition of the KL-divergence. Applying this and expanding the definitions of $\mathcal{Q}^t$ and $\mathcal{D}^t$, we obtain

$$\begin{aligned} \mathbf{E}_{\mathbf{k}}\Big[Q(\boldsymbol{\alpha}^{T+1})\Big] - Q(\mathbf{u}) &\leq \Big(1 - \frac{\eta\mu}{n}\Big)^T\Big(Q(\boldsymbol{\alpha}^1) - Q(\mathbf{u}) + \frac{1}{\eta}D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^1]\Big) \\ \mathbf{E}_{\mathbf{k}}\Big[Q(\boldsymbol{\alpha}^{T+1})\Big] - Q(\mathbf{u}) &\leq \exp\Big\{-T\frac{\eta\mu}{n}\Big\}\left(Q(\boldsymbol{\alpha}^1) - Q(\boldsymbol{\alpha}^\star) + \frac{1}{\eta}D[\mathbf{u}\,\|\,\boldsymbol{\alpha}^1]\right) \end{aligned}$$

In the first line above, note that the expectations on the right-hand side can be safely removed as neither $\boldsymbol{\alpha}^1$ nor $\mathbf{u}$ depend on the random indices $\mathbf{k}$. The second line above follows from the inequality $\log(1-x) \leq -x$, as well as the assumption that $\boldsymbol{\alpha}^\star$ is optimal, implying that $Q(\boldsymbol{\alpha}^\star) \leq Q(\mathbf{u})$ for all $\mathbf{u} \in \Delta^n$. ■

As we stated in the case of batch EG updating, choosing the optimal comparison vector $\mathbf{u} = \boldsymbol{\alpha}^\star$ for Theorem A.3.1 directly shows an $O(\log(\frac{1}{\epsilon}))$ rate of convergence for this algorithm. From Lemma A.1.4, we know that the log-linear dual Q_{LL} is (μ,τ)-online-bounded; hence, randomized online EG updating results in a $O(\log(\frac{1}{\epsilon}))$ rate of convergence for log-linear optimization.

Finally, note that the online rate of convergence is slower than the batch rate by a factor of n, but this is exactly compensated for by the fact that each batch update requires a pass over all n training examples, whereas each online update processes only one example. Moreover, as mentioned in Section A.1.3, additional speedups result from the fact that the value of the upper bound τ required by online EG can be much smaller than the corresponding upper bound required by batch EG (Collins et al., 2008).

Appendix B

Third-order Dependency Parsing Algorithms

This appendix provides additional descriptions of the parsing algorithms presented in Chapter 5.

B.1 Implementation Details

In this section, we focus on aspects of the third-order parsing algorithms that have been elided in the main text. While these details are unnecessary for a basic understanding of the parsers, they are critical for a correct and high-performance implementation of our algorithms. Topics discussed in this section include the instantiation of implicit lower-order parts, ordering and positionality of parts, scoring parts with null elements, and the modifications required for single-rooted and multi-rooted parsing.

B.1.1 Use of Implicit Lower-Order Parts

As mentioned in Section 5.2, higher-order parts contain lower-order parts by definition. For example, a grand-sibling part encompasses three dependency parts, a sibling part, and two grandchild parts. In addition to third-order interactions, our new parsing algorithms also evaluate second- and first-order interactions. Use of lower-order

parts in this manner is common practice; for example, McDonald and Pereira (2006) and Carreras (2007) both define second-order parsers in which first-order scores continue to be utilized.

Scoring lower-order interactions can be motivated as a form of backoff. Third-order interactions are powerful but may be difficult to estimate in all configurations; the use of second-order interactions and plain dependencies thus provides a solid basis.

In our approach, we explicitly define separate lower-order scoring functions. For example, the Model 0 parser evaluates two types of interactions: grandchild parts and plain dependencies. Within this parser the score of each tree decomposes as:

$$\textsc{Score}(\mathbf{x}, y) = \sum_{(h,m)\in y} \textsc{ScoreD}(\mathbf{x}, h, m) + \sum_{(g,h,m)\in y} \textsc{ScoreG}(\mathbf{x}, g, h, m)$$

where ScoreD and ScoreG are the scoring functions for dependencies and grandchild parts, respectively. Note that the above equation separates the two types of scores into different summations over the target tree y; however, within the parsing algorithm, the different scoring functions are of course interleaved.

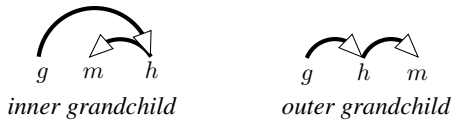
Technically, the use of separate lower-order scoring functions is redundant, since the evaluation of lower-order interactions could simply be rolled into the relevant higher-order scoring function. However, this would place a complex burden on the designer of the scoring function (i.e., the user): each lower-order part can generally appear in multiple higher-order parts, requiring careful consideration to avoid double-counting lower-order scores. In addition, drastic inefficiencies may also arise even if the double-counting problem is avoided: for example, a third-order parser following this approach may end up evaluating first-order scores $O(n^4)$ times, even though there are only n^2 total dependencies.

Finally, if one prefers, the lower-order scores can be easily disabled by defining them as identically 0.

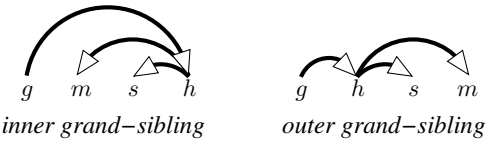
B.1.2 Ordering of Indices within Parts

In the main text, we have more or less treated each type of part as a homogeneous pool—e.g., all dependencies are scored alike—but in practice, we find it helpful to make a few distinctions based on the ordering of the indices within a part.

First, we distinguish between two types of grandchild interactions: *inner* and *outer* grandchild parts. In the former, the grandchild is in between the grandparent and parent in a zig-zag fashion, while in the latter, the grandchild is on the side facing away from the grandparent. The diagram below depicts these different configurations.



Our parsers define separate scoring functions for these two types of grandchild interactions. We also distinguish between inner and outer versions of grand-sibling interactions, which are shown below.



The use of the inner/outer distinction for grandchild-based interactions follows the practice set out in the Carreras (2007) parser. Note that the establishment of separate scoring functions is technically unnecessary, as the inner/outer configuration of the indices could be determined by simple inspection of the indices. However, given that the two types of grandchild parts appear in very different contexts and have overtly different shapes, we felt that the distinction was valid. In addition, without separate

scoring functions for inner and outer grandchildren, the evaluation of null grandchild parts (described in Section B.1.5 below) would be more complicated.

In addition to these distinctions for grandparent parts, we also distinguish between left-headed and right-headed versions of every part we score. However, we do not explicitly define separate left- and right-headed scoring functions, mostly in an effort to reduce the number of different scoring functions to manageable levels. Instead, this distinction is made by inspecting the relevant indices within each part and is thus not explicitly enforced by the parsing algorithms. Nevertheless, the left/right distinction is critical for high-performance parsing.

As a final note, for grandchild-based parts we determine left- or right-headedness by inspecting the ordering of the head—index h, the word in the middle of the grandchild relationship—and its modifier—index m, the word at the tail of the grandchild relationship.

B.1.3 Scoring Parts with Positionality

In addition to distinctions between higher- and lower-order parts and the ordering of indices within parts, we also distinguish parts based on their *positionality*. Positionality can be roughly described as a measurement of how close the modifiers in the part are to their relevant head. For example, the Model 2 parser defines three positionalities for first-order parts (h, m): position 1, meaning that m is the innermost modifier of h in some direction; position 2, meaning that m is the second-innermost modifier of h in some direction; and position 3+, meaning that m is the third-innermost or further modifier of h. Similar positionalities can be assigned to second-order sibling and grandchild parts. Third-order parts, on the other hand, do not have positionalities as our parsing algorithms do not distinguish between nearer and further versions of third-order parts.[1]

The rationale for introducing positionality is that the closeness of a modifier can change the role of that modifier. For example, in most languages we might expect to

[1] However, it is possible to modify them in order to detect this without increasing their asymptotic complexity. This is a minor improvement that may be explored in future work.

find complements in closer positions than adjuncts; note that the significance of the complement-adjunct distinction has been demonstrated by Collins (1999).

Some parsers (Koo et al., 2007; Carreras, 2007) have implemented a notion of *adjacency* in their first-order scores that is superficially similar to the notion of positionality defined here (p.c.). Adjacency, in the context of this previous work, indicates a dependency where $|h - m| = 1$, so that the head and modifier are adjacent in the sequential order of the sentence. While adjacent dependencies would necessarily occupy position 1 in our terminology, the converse is not true: the first modifier of some head need not be sequentially adjacent to that head.

B.1.4 A Listing of Part-Scoring Functions

Below, we enumerate the different part-scoring functions used by our third-order parsing algorithms:

- DEP1(h, m) : Evaluates an innermost dependency.
- DEP2+(h, m) : Evaluates a dependency that is second-innermost or further (only used by Models 0 and 1).
- DEP2(h, m) : Evaluates a second-innermost dependency (only used by Model 2).
- DEP3+(h, m) : Evaluates a dependency that is third-innermost or further (only used by Model 2).
- SIB1+(h, m, s) : Evaluates a sibling part at any position (only used by Model 1).
- SIB1(h, m, s) : Evaluates a sibling part where m and s are the two innermost modifiers of h (only used by Model 2).
- SIB2+(h, m, s) : Evaluates a sibling part where m and s are not the two innermost modifiers of h (only used by Model 2).

- GCI1(g, h, m) : Evaluates an inner grandchild part. For Models 0 and 1, m is the innermost modifier of h, and for Model 2, m is the outermost modifier of h.
- GCI2+(g, h, m) : Evaluates an inner grandchild part where m is the second-innermost or further modifier of h (only used by Models 0 and 1).
- GCI2(g, h, m) : Evaluates an inner grandchild part where m is the second-outermost modifier of h (only used by Model 2).
- GCO1(g, h, m) : Evaluates an outer grandchild part. For Models 0 and 1, m is the innermost modifier of h, and for Model 2, m is the outermost modifier of h.
- GCO2+(g, h, m) : Evaluates an outer grandchild part where m is the second-innermost or further modifier of h (only used by Models 0 and 1).
- GCO2(g, h, m) : Evaluates an outer grandchild part where m is the second-outermost modifier of h (only used by Model 2).
- TSIB(h, m, s, t) : Evaluates a tri-sibling part at any position (only used by Model 2).
- GCIS(g, h, m, s) : Evaluates an inner grand-sibling part. For Model 1, the part is at any position, and for Model 2, m and s are the two outermost modifiers of h.
- GCOS(g, h, m, s) : Evaluates an outer grand-sibling part. For Model 1, the part is at any position, and for Model 2, m and s are the two outermost modifiers of h.

Note that the naming convention used in the above is different than what appears in the main text and elsewhere. Here, and in the remainder of this appendix, we have opted to eliminate the redundant SCORE prefix in the interests of brevity.

The various distinctions that we have introduced—lower-order scoring functions, orderings of indices, and positionalities—are not necessary features of our parsers.

The distinctions can easily be ignored by simply eliminating or re-mapping the relevant scoring functions; for example, positionality can be ignored by mapping all scoring functions for a given type of part to a single, non-positional function. As these distinctions can be more conveniently established within the parser than outside of it, however, we felt that the best approach was to maintain them and allow the user to decide which to retain.

B.1.5 Scoring Parts with Null Elements

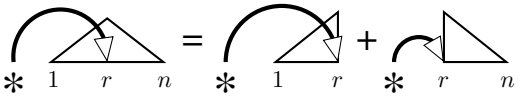
Often, important information can be recovered from parts that exist at the boundaries of a dependency tree. The clearest example of this is the dependencies of the abstract root *: the modifiers of * are the syntactic roots of the sentence, and thus play a critical role in the analysis of the sentence. However, the * token itself is essentially an null element used to define the boundary of the sentence, similar to the common practice of adding null tokens in order to allow an HMM to detect the beginning and end of a sentence. In addition to the obvious case of *, we also use null tokens in the following scoring functions to indicate that grandchildren are missing from one or the other side of some modifier:

- GCI1(g, h, NIL) : Evaluates a dependency (g, h) where h has no inner modifiers—i.e., no modifiers on the side of h closer to g.
- GCO1(g, h, NIL) : Evaluates a dependency (g, h) where h has no outer modifiers—i.e., no modifiers on the side of h further from g.
- GCI2(g, h, NIL) : Evaluates a dependency (g, h) where h has exactly one inner modifier (only used by Model 2).
- GCO2(g, h, NIL) : Evaluates a dependency (g, h) where h has exactly one outer modifier (only used by Model 2).

These scoring configurations, which we refer to as "null parts," arise through the degenerate cases of the parsing algorithms. In the following sections, we will point out situations where the null parts can be evaluated.

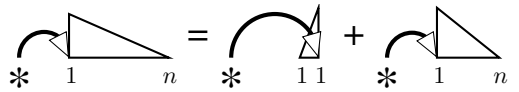
B.1.6 Single-Root and Multi-Root Variants

Both single-root and multi-root variants of our dynamic-programming parsers can be implemented using standard methods. In order to obtain a single-root parse of a sentence, it suffices to first compute the scores of all spans covering the range $[1, n]$ and then explicitly consider all possible single-root analyses. For Models 0, 1, and 2, a single-root analysis of a sentence can be created by combining two complete g-spans that, taken together, form a sentence-spanning constituent:



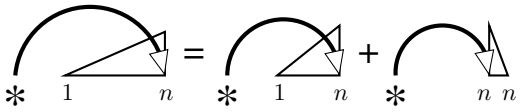
In this case, n is the length of the sentence and r corresponds to the index of the single syntactic root. Clearly, this operation is efficient, costing a total of $O(n)$ time per sentence. Note that the single-root construction also gives us an opportunity to add in the score DEP1$(0, r)$,[2] which evaluates the selection of r as a syntactic root.

There are two degenerate cases associated with the single-root construction: $r = 1$ and $r = n$. In these situations, r will have no modifiers to one side; thus these degenerate cases give us an opportunity to score certain null parts. For example, in the case that $r = 1$, we have the construction



Note that the left-headed complete g-span is an empty span containing only the root index 1, implying that the root has no left modifiers. Thus, the null part score GCI1$(0, 1, \text{NIL})$ can be included, in addition to the root dependency score DEP1$(0, 1)$. At the other extreme, if $r = n$, we have

[2] Recall that the abstract root is located at index 0.



In this case, the appropriate scores would be GCO1$(0, n, \text{NIL})$ and DEP1$(0, 1)$.

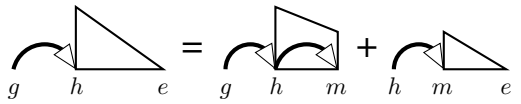
Multi-root parsing is a simpler matter. Without the need to enforce a single-root constraint, it suffices to simply find all spans covering the range $[0, n]$—i.e., treat the abstract root as a real token prepended on the sentence. The best multi-root parse of the sentence can then be extracted by examining the complete g-span headed at 0 and spanning the entire sentence. This complete g-span will by definition contain the score of the best tree headed at 0, for any number of syntactic roots.

B.2 Model 0

In this section, we will examine each of the possible recursive subproblems of Model 0. For each subproblem, we will list the parts that can be scored during the processing of this subproblem, as well as any degenerate cases and null parts that can be scored. In this section and the remainder of this appendix, in addition to distinguishing between left-headed or right-headed spans, we will also refer to spans as being left-grandparented and right-grandparented when appropriate.

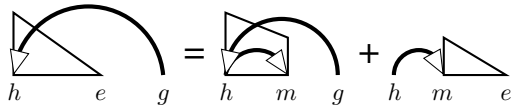
B.2.1 Complete G-Spans

The recursive construction of a left-headed, left-grandparented complete g-span is given below.



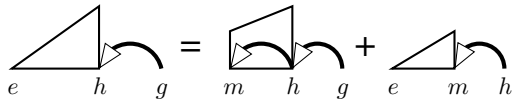
In this situation we do not score any parts, although we could conceivably score both first-order and grandchild interactions by using the split point m. However, scoring these parts at this point in the parser would be inefficient as each grandchild part (g, h, m) can be evaluated multiple times: once for each left-headed complete g-span with $e \geq m$. In fact, the same inefficiency applies to all forms of complete g-spans, so we never evaluate any part scores when creating these structures. Note that this construction has a single degenerate case, $m = e$, which allows the null grandchild part GCO1(h, m, NIL) to be scored.

We now turn to the case of a left-headed, right-grandparented complete g-span.



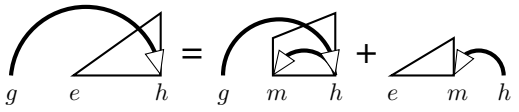
As explained above, we do not score any parts here for reasons of efficiency. The degenerate case is again $m = e$, leading to a null outer grandchild part GCO1(h, m, NIL). Note that despite the reversal of the grandparent index g, the null part is still an outer grandchild part as it arises from the dependency (h, m), not the grandparent dependency (g, h).

The right-headed cases are simply mirror images of the left-headed versions, but we include them for completeness. First, we display the recursive subproblem for right-headed, right-grandparented complete g-spans.



As usual, there are no part scores, and the degenerate case $m = e$ triggers evaluation of the null outer grandchild part GCO1(h, m, NIL).

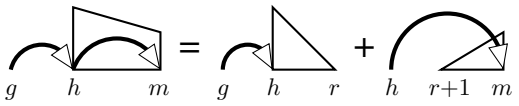
The final case is shown below: right-headed, left-grandparented complete g-spans.



Once again, the degenerate case $m = e$ indicates a null outer grandchild part scored by GCO1(h, m, NIL).

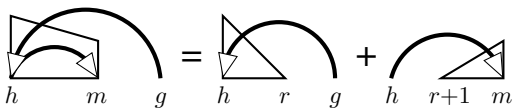
B.2.2 Incomplete G-Spans

As mentioned above, we compute part-scores only when constructing incomplete parts, ensuring that every grandchild part is evaluated exactly once. We begin by examining left-headed, left-grandparented incomplete g-spans.



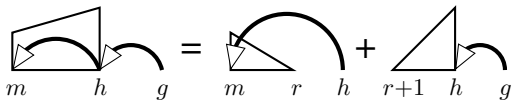
In the above, we score both grandchild parts (g, h, m) and dependency parts (h, m). The exact scoring functions called depend on the split point r. In the degenerate case $r = h$, it should be clear that m must be the innermost modifier of h, so the appropriate scores are DEP1(h, m) and GCO1(g, h, m). Otherwise, if $r \neq h$, then the appropriate scoring functions are DEP2+(h, m) and GCO2+(g, h, m). There is a second degenerate case when $r + 1 = m$, which indicates a null inner grandchild part GCI1(h, m, NIL). Note that the two degenerate cases may coincide when $m = h + 1$.

Left-headed, right-grandparented incomplete g-spans are shown below.



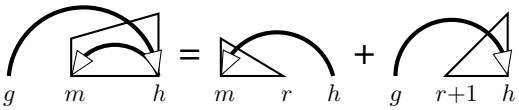
Here, we score inner grandchild parts, due to the reversed position of the grandparent index. Specifically, if $r = h$, we evaluate DEP1(h, m) and GCI1(g, h, m); otherwise, we use DEP2+(h, m) and GCI2+(g, h, m). In the case that $r + 1 = m$, we also evaluate GCI1(h, m, NIL)—as we saw in Section B.2.1 above, the position of g does not affect the type of null grandchild part involved.

We now move on to right-headed, right-grandparented incomplete g-spans.



In this construction, if $r + 1 = h$ then we score DEP1(h, m) and GCO1(g, h, m); otherwise we score DEP2+(h, m) and GCO2+(g, h, m). The second degenerate case is indicated by $r = m$ and implies the null part GCI1(h, m, NIL).

Finally, right-headed, left-grandparented incomplete g-spans are depicted below.



Here, the grandparent position is again reversed, so we would score DEP1(h, m) and GCI1(g, h, m) if $r + 1 = h$, and DEP2+(h, m) and GCI2+(g, h, m) otherwise. As in the above, if $r = m$ then we score the null part GCI1(h, m, NIL).

In all cases, although the split point does affect the positionality of the scoring function, none of the parts involved directly depend on the value of r. Therefore, it is possible to push the part-scoring outside of the loop enumerating values of r, similar to the pseudocode sketch shown in Figure 5-5. For example, one could simply evaluate both GCO1(g, h, m) and GCO2+(g, h, m) before entering the loop over r and select whichever is appropriate within the loop. Since dependency parts also do not depend on the grandparent index g, their evaluation can be pushed outside of

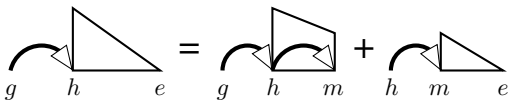
the loop over g as well. For our experiments, however, we never created a stand-alone implementation of the Model 0 parser. Instead, we simply emulated Model 0 by running Model 1 while deactivating all scores except dependencies and grandchild parts. Since both parsers have the same asymptotic complexity, there is little reason to implement a separate Model 0.

B.3 Model 1

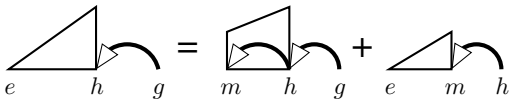
In this section, we will examine the Model 1 parsing algorithm. Although the complete g-spans of Model 1 are constructed in an identical fashion to those of Model 0, we will not elide them in the interests of completeness. However, our discussion of these aspects of the Model 1 algorithm will be brief.

B.3.1 Complete G-Spans

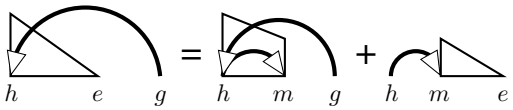
As in Model 0, none of the complete g-spans are associated with any part-scores, except null parts that arise in degenerate cases. The recursive derivations of the four types of complete g-spans are depicted below. First, we have the left-headed and left-grandparented version,



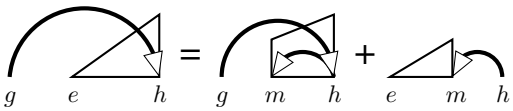
as well as its mirror image.



The grandparents can be reversed, leading to the pair depicted below: left-headed and right-grandparented incomplete spans,



and their mirror image.



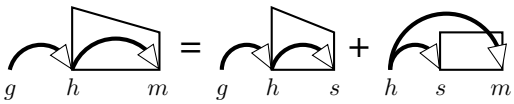
All four constructions share a single degenerate case, $m = e$, which allows the null grandchild part $\textsc{Gco1}(h, m, \textsc{nil})$ to be scored.

Before we move on, we note that complete g-spans play a slightly different role in Model 1 than they do in Model 0. Specifically, in Model 1 the structure $C^g_{h,e}$ only appears when the head h has accepted all of its modifiers in the direction of e, whereas in Model 0, the head of $C^g_{h,e}$ can still accept additional modifiers. The reason for this subtle change is the introduction of sibling g-spans, which alter the derivation of incomplete g-spans so that they are self-recursive rather than recursing on complete g-spans (see Section B.3.2). Consequently, in Model 1 a complete g-span $C^g_{h,e}$ never contains within it a smaller complete g-span $C^g_{h,e'}$, and it follows that complete g-spans are non-recursive, only appearing at their maximum width. The non-recursivity of complete spans is also a property of the second-order McDonald and Pereira (2006) parser as well as the Model 2 third-order parser.

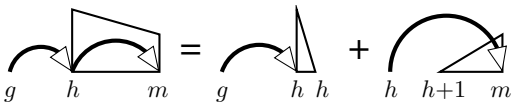
B.3.2 Incomplete G-Spans

The incomplete g-spans of Model 1 are constructed in a fashion similar to the incomplete spans of the second-order McDonald and Pereira (2006) algorithm, with the

exception that grandparent indices are retained throughout. We begin by examining left-headed, left-grandparented incomplete g-spans.

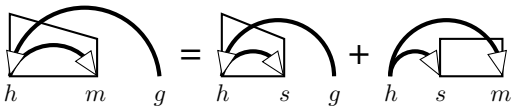


In the above, we score dependency parts (h, m), grandchild parts (g, h, m), and grand-sibling parts (g, h, m, s). Note that the presence of s automatically implies that m is not the innermost modifier of h, so we use the scoring functions $\textsc{Dep2+}(h, m)$, $\textsc{Gco2+}(g, h, m)$, and $\textsc{Gcos}(g, h, m, s)$. In addition to the above, which represents the "normal" construction of an incomplete g-span, there is an alternate derivation that represents the situation where m is the innermost modifier of h. In this case we use a significantly different construction, as shown below.

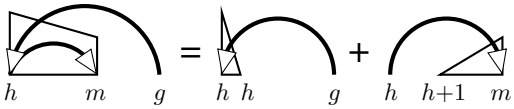


Here, instead of recursing on a smaller incomplete g-span and sibling g-span, we instead recurse on a single complete g-span (the empty complete g-span being non-recursive). This construction is accompanied with the part-scores $\textsc{Dep1}(h, m)$ and $\textsc{Gco1}(g, h, m)$ as m is demonstrably the innermost modifier of h. Note that this alternate derivation has a single degenerate case, when $h + 1 = m$; in this situation the null part score $\textsc{Gci1}(h, m, \textsc{nil})$ must also be added.

Left-headed, right-grandparented incomplete g-spans are shown below.

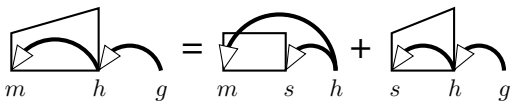


Here, due to the reversed position of the grandparent index we evaluate $\textsc{Dep2+}(h, m)$, $\textsc{Gci2+}(g, h, m)$, and $\textsc{Gcis}(g, h, m, s)$. The alternate construction for the situation where m is the first modifier is given below.

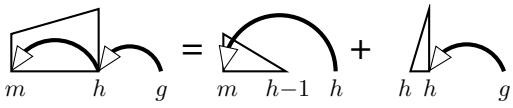


The relevant first-modifier scores are $\textsc{Dep1}(h, m)$ and $\textsc{Gci1}(g, h, m)$. Again, there is a single degenerate case that occurs if $h + 1 = m$, which triggers the null part $\textsc{Gci1}(h, m, \textsc{nil})$.

We now move on to right-headed, right-grandparented incomplete g-spans.

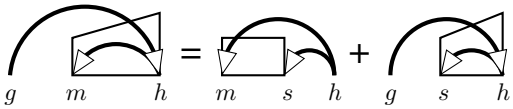


In this construction we would evaluate the scores $\textsc{Dep2+}(h, m)$, $\textsc{Gco2+}(g, h, m)$, and $\textsc{Gcos}(g, h, m, s)$. The alternate first-modifier construction is shown below.

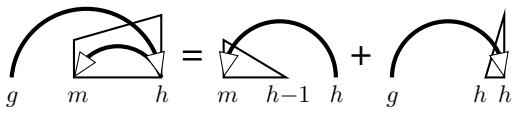


This derivation is scored with $\textsc{Dep1}(h, m)$ and $\textsc{Gco1}(g, h, m)$. The degenerate case is defined by $h - 1 = m$, which implies the null part $\textsc{Gci1}(h, m, \textsc{nil})$.

Finally, right-headed, left-grandparented incomplete g-spans are depicted below.



This first diagram illustrates the normal case, which involves the scores $\textsc{Dep2+}(h, m)$, $\textsc{Gci2+}(g, h, m)$, and $\textsc{Gcis}(g, h, m, s)$. The alternate first-modifier construction is shown below.

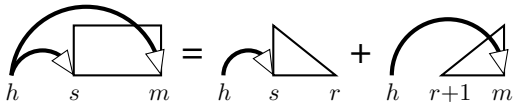


In this situation we would use the scores $\textsc{Dep1}(h, m)$ and $\textsc{Gci1}(g, h, m)$; in the degenerate case $h - 1 = m$, we evaluate $\textsc{Gci1}(h, m, \textsc{nil})$.

As in Model 0, it is possible to avoid repeatedly re-evaluating the lower-order scoring functions by pushing them outside of the loop over the split point s (and, in the case of first-order parts, outside the loop over g as well). Note that we could have evaluated sibling parts (h, m, s) while constructing incomplete g-spans, but we do not do so for reasons of efficiency. Specifically, each sibling part (h, m, s) would need to be re-evaluated for every grandparent index, and it would not be possible to push the evaluation of the sibling scores outside the loop over g due to their dependence on the split point s. Instead, we take the more efficient and more natural approach of evaluating the sibling scores in the sibling g-spans.

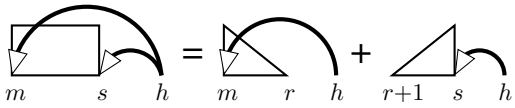
B.3.3 Sibling G-Spans

Here, we describe the methods by which the Model 1 parser recursively constructs sibling g-spans. As we mentioned above, sibling part scores are evaluated during the derivation of these spans. We must also be careful to examine the degenerate cases of the sibling g-spans must in order to capture all possible null parts. Unlike the other types of g-spans, there are only two possible configurations for sibling g-spans: left-headed and right-headed. The left-headed version is shown below.



Since Model 1 is incapable of distinguishing between the first pair of modifiers and subsequent pairs, this construction is scored with $\textsc{Sib1+}(h, m, s)$. There are two possible degenerate cases to consider: first, if $r = s$, then we add the null part score $\textsc{Gco1}(h, s, \textsc{nil})$; second, if $r+1 = m$, then we add the null part score $\textsc{Gci1}(h, m, \textsc{nil})$. Note that these two degenerate cases can co-occur if $s + 1 = m$.

Right-headed sibling g-spans are depicted below.



As in the left-headed case, we add the score $\textsc{Sib1+}(h, m, s)$ in this derivation. The degenerate cases in this situation are: first, if $r+1 = s$ then we add the null part score $\textsc{Gco1}(h, s, \textsc{nil})$; second, if $r = m$ then we add the null part score $\textsc{Gci1}(h, m, \textsc{nil})$.

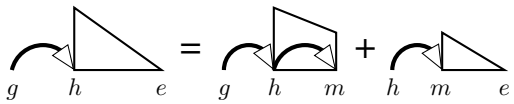
B.4 Model 2

In this section, we will examine the Model 2 parsing algorithm. The complete g-spans of Model 2 have derivations identical to those of Models 0 and 1, and in addition, the sibling g-spans of Model 2 have constructions identical to those of Model 1. In order to provide a self-contained description, however, we will include a shortened specification of these constructions in this section.

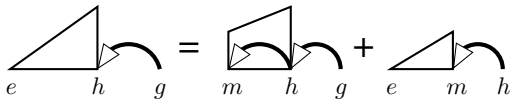
B.4.1 Complete G-Spans

As in Models 0 and 1, none of the complete g-spans are associated with any part-scores, except null parts that arise in degenerate cases. The recursive derivations of

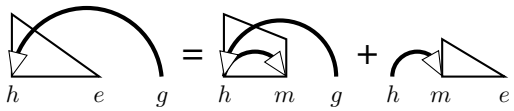
the four types of complete g-spans are depicted below. First, we have the left-headed and left-grandparented version,



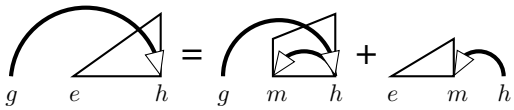
as well as its mirror image.



The grandparents can be reversed, leading to the pair depicted below: left-headed and right-grandparented incomplete spans,



and their mirror image.

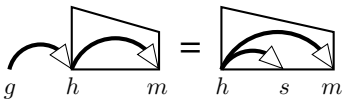


All four constructions share a single degenerate case, $m = e$, which allows the null grandchild part $\textsc{Gco1}(h, m, \textsc{nil})$ to be scored.

B.4.2 Incomplete G-Spans

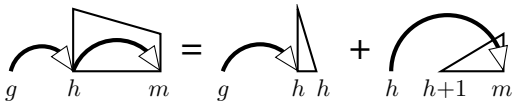
Incomplete g-spans in Model 2 are constructed in a very different way from those of Model 1. Whereas Model 1's version is self-recursive, being constructed from a smaller incomplete g-span, Model 2's version is not self-recursive. Instead, in Model 2 an incomplete g-span is converted into an incomplete s-span, so that incomplete g-spans are only instantiated for the outermost layer of grandchildren; in addition, the conversion also causes the parser to "forget" the grandparent index. The non-recursivity and elimination of the grandparent are exactly the reasons why Model 2 can only define grandchild-based parts for the outermost pair of grandchildren.

We begin by examining left-headed, left-grandparented incomplete g-spans.



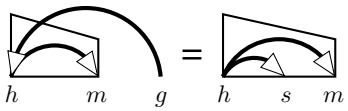
In the above, we score grandchild parts (g, h, m) and (g, h, s) as well as grand-sibling parts (g, h, m, s). Specifically, we invoke the scoring functions $\textsc{Gco1}(g, h, m)$ and $\textsc{Gco2}(g, h, s)$ which, as described in Section B.1.4, refer to the outermost and second-outermost grandchild interaction; the grand-sibling interaction is scored via $\textsc{Gcos}(g, h, m, s)$. Note that we do not score the dependency (h, m), leaving this instead for the incomplete s-span to take care of.

As in Model 1, there is an alternative construction for incomplete g-spans that represents the situation where m is the innermost modifier of h—i.e., where s does not exist. In this case we use the same recursive derivation as Model 1, which is reproduced below.

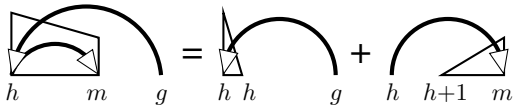


Here, instead of converting the g-span into an s-span, we recurse on a single complete g-span. This construction is accompanied with the scores $\textsc{Gco1}(g, h, m)$ and $\textsc{Gco2}(g, h, \textsc{nil})$. In addition, since there is no incomplete s-span involved, the dependency score must also be discharged here, so we add the score $\textsc{Dep1}(h, m)$. As in Model 1, this alternate derivation has a degenerate case when $h + 1 = m$; in this situation the null part score $\textsc{Gci1}(h, m, \textsc{nil})$ must be added.

Left-headed, right-grandparented incomplete g-spans are shown below.

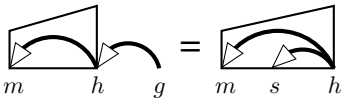


Here, due to the reversed position of the grandparent index we evaluate $\textsc{Gci1}(g, h, m)$, $\textsc{Gci2}(g, h, s)$, and $\textsc{Gcis}(g, h, m, s)$. Note that the incomplete s-span used here remains the same as for the left-grandparented version above. The alternate first-modifier construction is shown below.

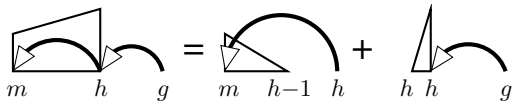


The relevant scores are $\textsc{Dep1}(h, m)$, $\textsc{Gci1}(g, h, m)$, and $\textsc{Gci2}(g, h, \textsc{nil})$, and in the degenerate case $h + 1 = m$, we add the null part score $\textsc{Gci1}(h, m, \textsc{nil})$.

We now move on to right-headed, right-grandparented incomplete g-spans.

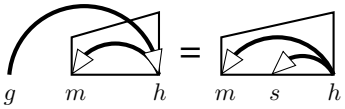


In this situation we evaluate $\textsc{Gco1}(g, h, m)$, $\textsc{Gco2}(g, h, s)$, and $\textsc{Gcos}(g, h, m, s)$. The alternate first-modifier construction is shown below.

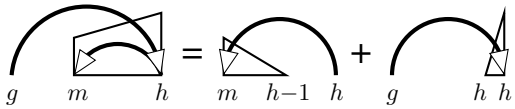


This derivation is scored with $\textsc{Dep1}(h, m)$, $\textsc{Gco1}(g, h, m)$, and $\textsc{Gco2}(g, h, \textsc{nil})$. The degenerate case is defined by $h - 1 = m$, which implies the null part $\textsc{Gci1}(h, m, \textsc{nil})$.

Finally, right-headed, left-grandparented incomplete g-spans are depicted below.



This first diagram illustrates the normal case, which involves the scores $\textsc{Gci1}(g, h, m)$, $\textsc{Gci2}(g, h, s)$, and $\textsc{Gcis}(g, h, m, s)$. The alternate first-modifier construction is shown below.

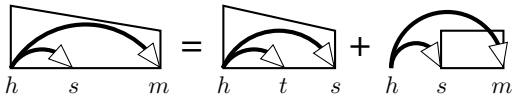


In this situation we would apply the part-scores $\textsc{Dep1}(h, m)$, $\textsc{Gci1}(g, h, m)$, and $\textsc{Gci2}(g, h, \textsc{nil})$; in the degenerate case $h - 1 = m$, we evaluate $\textsc{Gci1}(h, m, \textsc{nil})$.

As in Models 0 and 1, it is possible to avoid repeatedly re-evaluating some of the lower-order scoring functions by pushing them outside of the loop over the split point s. Note that in general, we have not evaluated either dependency parts (h, m) or sibling parts (h, m, s) in these constructions, the reason being that their positionality can only be determined within the context of the incomplete s-spans.

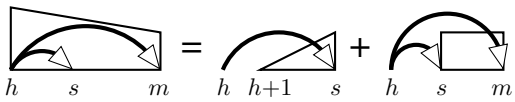
B.4.3 Incomplete S-Spans

Incomplete s-spans are a new type of dynamic-programming structure used by Model 2, which consist of standard Eisner-style incomplete spans augmented with internal sibling indices, similar to the complete spans of Carreras (2007). Due to this extra index, these structures increase the amount of horizontal context available to the parser, allowing the evaluation of tri-sibling parts as well as extended positionality information for dependencies and siblings. Like sibling g-spans, there are only two configurations of incomplete s-spans: left-headed and right-headed. The left-headed versions are depicted below.



In this derivation, we score dependency parts (h, m), sibling parts (h, m, s), and tri-sibling parts (h, m, s, t). Note that we cannot score sibling parts in the sibling g-spans—as was done in Model 1—because the positionality of a sibling part can only be determined in this construction. Specifically, the presence of the split point t indicates that m and s are not the innermost pair of modifiers of h, so we score the sibling interaction using the function $\textsc{Sib2+}(h, m, s)$, while the dependency is scored with the function $\textsc{Dep3+}(h, m)$. The tri-sibling part is scored as $\textsc{Tsib}(h, m, s, t)$.

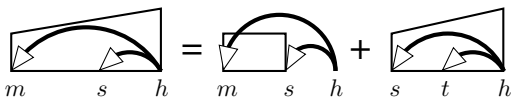
Similar to the incomplete g-spans described above, there is an alternate construction for incomplete s-spans that covers the situation where s is the innermost modifier of h—i.e., t does not exist. The derivation in this case is shown below.



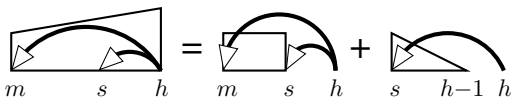
Here, s and m are clearly the first pair of modifiers of h, so we score this construction with $\textsc{Dep1}(h, s)$, $\textsc{Dep2}(h, m)$, and $\textsc{Sib1}(h, m, s)$. There is a degenerate case that

occurs when $h + 1 = s$; in this situation we add the null part score $\textsc{Gci1}(h, s, \textsc{nil})$.

The recursive derivation of the right-headed incomplete s-spans is depicted below.



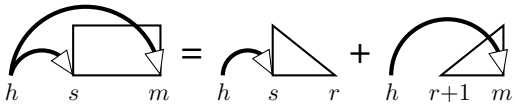
As in the left-headed version, we add the scores $\textsc{Dep3+}(h, m)$, $\textsc{Sib2+}(h, m, s)$, and $\textsc{Tsib}(h, m, s, t)$. In the case that s is the first modifier of h, we use the following alternate derivation.



Here, we use the scores $\textsc{Dep1}(h, s)$, $\textsc{Dep2}(h, m)$, and $\textsc{Sib1}(h, m, s)$. In the degenerate case $h - 1 = s$, we also add the null part score $\textsc{Gci1}(h, s, \textsc{nil})$.

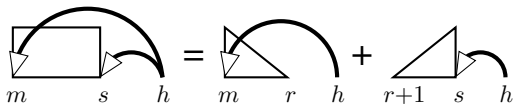
B.4.4 Sibling G-Spans

Here, we describe the recursive construction of sibling g-spans. As mentioned earlier, sibling parts have two different positionalities in Model 2, and the positionality can only be determined within an incomplete s-span. Consequently, we do not evaluate sibling scores at this point in the parser, which is different than the behavior described in Model 1. In other respects, however, these recursive constructions are identical to those seen in Model 1. The left-headed version is shown below.



As in Model 1, there are two degenerate cases: first, if $r = s$ then we add the null part score $\textsc{Gco1}(h, s, \textsc{nil})$; second, if $r + 1 = m$ then we add the null part score $\textsc{Gci1}(h, m, \textsc{nil})$. The two degenerate cases can co-occur when $s + 1 = m$.

The derivation of right-headed sibling g-spans is depicted below.



Here, the degenerate cases are: first, if $r + 1 = s$ then we add the null part score $\textsc{Gco1}(h, s, \textsc{nil})$; second, if $r = m$ then we add the null part score $\textsc{Gci1}(h, m, \textsc{nil})$.

B.5 Extensions

This section provides additional descriptions and discussion of the extensions to the third-order parser mentioned in Section 5.5.

B.5.1 Parsing with Word Senses and Dependency Labels

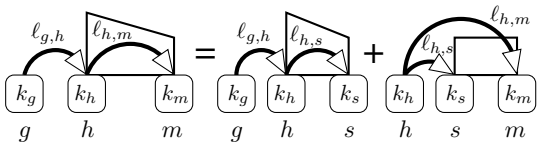
Here, we briefly describe the modifications required in order to parse while simultaneously recovering both word senses and dependency labels, following the general techniques of Eisner (2000). In the interests of brevity, we will not provide a full description of the algorithm as was done in previous sections. As the modifications are quite mechanical, however, the sketch provided in this section should suffice for the purpose of illustration.

The overall approach is to augment each part and each dynamic-programming structure with senses and labels. Formally, we assume that word senses can be represented as indices in the set $\{1, \ldots, K\}$ while dependency labels can be represented as indices in $\{1, \ldots, L\}$; here, K and L denote the total number of senses and labels. For any position i within the sentence, we denote the sense of the word at that position as k_i, and for any dependency (h, m), we denote the label of that dependency as $\ell_{h,m}$.

Each part p is associated with extra indices that completely define all of the word senses and dependency labels contained in p. For example, a first-order part is no longer represented as a pair (h, m), but a 5-tuple $(h, m, k_h, k_m, \ell_{h,m})$. A grandchild part would be redefined as a 8-tuple $(g, h, m, k_g, k_h, k_m, \ell_{g,h}, \ell_{h,m})$. Similarly, each g-span and s-span is associated with extra indices defining the word senses and dependency labels intrinsic to that span. For example, an incomplete g-span might be defined by the notation $I^{g,k_g,\ell_{g,h}}_{h,m,k_h,k_m,\ell_{h,m}}$, capturing three additional word-sense indices k_g, k_h, and k_m, as well as two additional dependency-label indices $\ell_{g,h}$ and $\ell_{h,m}$.

Despite the significant increase in notational complexity, parsing algorithms that recover senses and labels are fairly simple to specify, being essentially identical in form to their unlabeled counterparts. The main modification is to modify each recursive derivation so that the parts and sub-spans involved agree with each other on sense on label indices at the points where they overlap. A more subtle issue is that in the modified parsers a split point is generally augmented with a sense index and label index, with the exception of the split points of incomplete g-spans in Model 0 and sibling g-spans in Models 1 and 2.

To illustrate the method, we will consider an example of a recursive construction that would appear in a modified version of the Model 1 parser: the derivation of left-headed, left-grandparented incomplete g-spans.



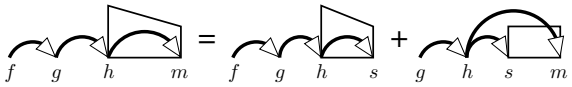
In this case, instead of a single split point s, we now have a triple $(s, k_s, \ell_{h,s})$. Note that word sense indices and dependency label indices are propagated into the smaller components in the obvious manner. The scores associated this construction would be $\textsc{Dep2+}(h, m, k_h, k_m, \ell_{h,m})$, $\textsc{Gco2+}(g, h, m, k_g, k_h, k_m, \ell_{g,h}, \ell_{h,m})$, and $\textsc{Gcos}(g, h, m, s, k_g, k_h, k_m, k_s, \ell_{g,h}, \ell_{h,m}, \ell_{h,s})$. Based on this example, the modifications required for the other recursive derivations and other parsing algorithms should hopefully be clear.

As a result of the additional indices, parsing algorithms that recover senses and labels entail a much higher degree of complexity than unlabeled parsers—e.g., time and space requirements of $O(n^4K^4L^3)$ and $O(n^3K^3L^2)$, respectively, for the augmented version of Model 1 sketched above. It is therefore essential to carefully prune the search space of these modified parsers. However, the possible benefits of employing parsers capable of simultaneously considering 4-tuples of word senses and triplets of dependency labels are intriguing. For example, such parsers could be used to recover labeled dependency trees while simultaneously using the word senses to POS-tag the sentence while it is being parsed.

B.5.2 Extended Vertical Markovization

In this section, we briefly describe the modifications necessary in order to extend the vertical context available to our third-order parsing algorithms. The overall method is quite simple: instead of a single grandparent index, two or more ancestor indices would be used. For example, in order to increase the vertical context of Model 1 by one step, we add a great-grandparent index to the chain of ancestors, converting g-spans into "gg-spans."

We make this idea concrete by exploring one of the recursive derivations involved in a version of Model 1 that is extended with great-grandparent indices. Specifically, we depict a left-headed, left-grandparented, left-great-grandparented incomplete gg-span below:



Here, we would evaluate the parts (h, m), (g, h, m), and (g, h, m, s),[3] as in the original

[3]The grand-sibling parts might also be deferred to the sibling gg-spans in this extended Model 1 parser, for reasons of efficiency.

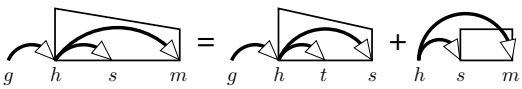
version of Model 1, as well as new great-grandchild parts (f, g, h, m) and great-grand-sibling parts (f, g, h, m, s). Note that the number of possible orderings of grandparent indices would greatly increase as compared to the standard Model 1.

In a practical implementation of this extended Model 1, additional modifications may be required in order to account for dependencies involving the abstract root *. In order to avoid tiresome special cases, one convenient approach might be to define multiple abstract root tokens; for example, in the case of a parser with grandparents and great-grandparents, $*$ and $**$. The meaning of $*$ would remain largely the same—words that modify $*$ are considered to be the syntactic roots of the sentence—except that a new dependency $(**, *)$ would be added, so that the root of the tree structure is $**$ instead of $*$. The use of multiple abstract roots is reminiscent of the use of multiple null tokens in higher-order sequence labeling algorithms, such as trigram HMMs.

B.5.3 Extended Horizontal Markovization

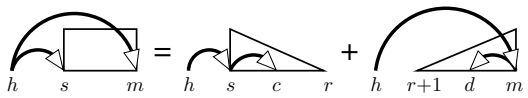
In this section, we briefly describe the modifications necessary in order to extend the horizontal context available to our third-order parsing algorithms. At the core of the approach is the addition of internal "sibling indices" to each dynamic-programming structure, similar to the incomplete s-spans of Model 2. For example, in order to increase the horizontal context of Model 1 by one step, we would augment the complete and incomplete g-spans with sibling indices, creating complete and incomplete "gs-spans." Interestingly, note that it is unnecessary to add sibling indices to the sibling g-spans, which remain largely unmodified in structure.

Below, we provide a concrete example by depicting the derivation of incomplete gs-spans in a modified version of Model 1 that uses one additional sibling index.



Here, we would evaluate the parts (h, m), (g, h, m), and (g, h, m, s), as in the original version of Model 1. In addition, we would score tri-sibling parts (h, m, s, t), which are normally only seen in Model 2, and new grand-tri-sibling parts (g, h, m, s, t).

The construction of the sibling g-spans in this parser is an issue that deserves some special attention. While sibling g-spans in the modified Model 1 parser would retain the same structure as their counterparts in the original Model 1, their derivation would be altered, as shown below.



Note that there are 6 distinct indices (h, m, s, r, c, d), which might lead one to believe that parsing this factorization requires $O(n^6)$ time. Critically, however, the indices c and d in the above are independent—i.e., there is no part that directly depends on both indices simultaneously. Therefore, once the values of h, m, s, and r are set, the values of c and d can be enumerated in two separate loops, as opposed to two nested loops. The overall runtime thus remains $O(n^5)$, as stated in the main text. It should be clear that even when the amount of horizontal context is increased beyond a single extra sibling index, the sibling indices on either side of r in this derivation will remain independent. Thus, similar optimizations will allow the extended factorization to be parsed in the desired runtime. As a final remark, recall that this independent-index optimization is identical to the technique used in Carreras (2007), where an apparently $O(n^5)$ parsing algorithm is shown to be $O(n^4)$.

Bibliography

Giuseppe Attardi. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of the 10th CoNLL*, pages 166–170. Association for Computational Linguistics, 2006.

James Baker. Trainable Grammars for Speech Recognition. In *Proceedings of the 97th meeting of the Acoustical Society of America*, 1979.

Peter L. Bartlett, Michael Collins, Ben Taskar, and David McAllester. Exponentiated Gradient Algorithms for Large-Margin Structured Classification. In *NIPS*, 2004.

Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41:164–171, 1970.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the 5th Workshop on Computational Learning Theory*, pages 144–152. Association for Computing Machinery, 1992.

Léon Bottou. Stochastic Learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, 2004.

L.M. Bregman. The Relaxation Method of Finding the Common Point of Convex Sets and its Application to the Solution of Problems in Convex Programming. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479, 1992.

Sabine Buchholz and Erwin Marsi. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the 10th CoNLL*, pages 149–164. Association for Computational Linguistics, 2006.

Xavier Carreras. Experiments with a Higher-Order Projective Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 957–961. Association for Computational Linguistics, 2007.

Xavier Carreras and Michael Collins. Non-Projective Parsing for Statistical Machine Translation. In *Proceedings of EMNLP*, pages 200–209. Association for Computational Linguistics, 2009.

Xavier Carreras, Michael Collins, and Terry Koo. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *Proceedings of the 12th CoNLL*, pages 9–16. Association for Computational Linguistics, 2008.

Eugene Charniak. A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL*, 2000.

Eugene Charniak. Statistical Parsing with a Context-Free Grammar and Word Statistics. In *Proceedings of the 14th AAAI and 9th IAAI*, pages 598–603, 1997.

Eugene Charniak and Mark Johnson. Coarse-to-fine N-best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd ACL*, 2005.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, and Mark Johnson. *BLLIP 1987–89 WSJ Corpus Release 1, LDC No. LDC2000T43.* Linguistic Data Consortium, 2000.

Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. Machine Learning-Based Dependency Analyzer for Chinese. In *Proceedings of the International Conference on Chinese Computing*, 2005.

Noam Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(3):113–124, September 1956.

Noam Chomsky. *Aspects of the Theory of Syntax.* The MIT Press, 1969.

Sunil Chopra. On the Spanning Tree Polyhedron. *Operations Research Letters*, 8: 25–29, 1989.

Y.J. Chu and T.H. Liu. On the Shortest Arborescence of a Directed Graph. *Science Sinica*, 14:1396–1400, 1965.

Alexander Clark. Inducing Syntactic Categories by Context Distribution Clustering. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning*, pages 91–94. Association for Computational Linguistics, 2000.

John Cocke and Jacob T. Schwartz. Programming Languages and Their Compilers: Preliminary Notes. Technical report, New York University, 1970.

Michael Collins. Discriminative Reranking for Natural Language Parsing. In *Proceedings of ICML*, 2000.

Michael Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 7th EMNLP*, pages 1–8. Association for Computational Linguistics, 2002.

Michael Collins. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th ACL*, pages 184–191, 1996.

Michael Collins. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th ACL*, pages 16–23, 1997.

Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 1999.

Michael Collins and Terry Koo. Discriminative Reranking for Natural Language Parsing. *Computational Linguistics*, 31(1):25–69, 2005.

Michael Collins, Jan Hajič, Lance Ramshaw, and Christoph Tillmann. A Statistical Parser for Czech. In *Proceedings of the 37th ACL*, pages 505–512. Association for Computational Linguistics, 1999.

Michael Collins, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L. Bartlett. Exponentiated Gradient Algorithms for Conditional Random Fields and Max-Margin Markov Networks. *Journal of Machine Learning Research*, 9:1775–1822, Aug 2008.

Corinna Cortes and Vladimir N. Vapnik. Support-vector Networks. *Machine Learning*, 20(3):273–297, 1995.

Koby Crammer and Yoram Singer. Ultraconservative Online Algorithms for Multiclass Problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

Koby Crammer, Jaz Kandola, and Yoram Singer. Online Classification on a Budget. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2003.

Koby Crammer, Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. Online Passive-Aggressive Algorithms. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *NIPS 16*, pages 1229–1236. MIT Press, 2004.

Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. *SIAM Journal on Computing*, 37(5), January 2008.

Sašo Džeroski, Tomaž Erjavec, Nina Ledinek, Petr Pajas, Zdenek Žabokrtský, and Andreja Žele. Towards a Slovene dependency treebank. In *Proceedings of the Fifth Intern. Conf. on Language Resources and Evaluation (LREC)*, 2006.

Jack R. Edmonds. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

Jason 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, 2000.

Jason Eisner. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th COLING*, pages 340–345. Association for Computational Linguistics, 1996.

Jason Eisner and Giorgio Satta. Efficient Parsing for Bilexical Context-Free Grammars and Head-Automaton Grammars. In *Proceedings of the 37th ACL*, pages 457–464, 1999.

Jenny R. Finkel, Trond Grenager, and Christopher D. Manning. The Infinite Tree. In *Proceedings of the 45th ACL*, pages 272–279. Association for Computational Linguistics, 2007.

Jenny R. Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, Feature-based, Conditional Random Field Parsing. In *Proceedings of ACL/HLT-2008*, pages 959–967, 2008.

Yoav Freund and Robert E. Schapire. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296, 1999.

William A. Gale, Kenneth W. Church, and David Yarowsky. One Sense Per Discourse. In *Proceedings of the Workshop on Speech and Natural Language in the Human Language Technology Conference*, pages 233–237, 1992.

Amir Globerson, Terry Koo, Xavier Carreras, and Michael Collins. Exponentiated gradient algorithms for log-linear structured prediction. In Zoubin Ghahramani, editor, *Proceedings of the 24th International Conference on Machine Learning*, pages 305–312, New York, NY, 2007. ACM Press.

Liliane Haegeman. *Introduction to Government and Binding Theory*. Wiley-Blackwell, 1991.

J. Hajič, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. Prague Arabic dependency treebank: Development in data and tools. In *Proceedings of the NEMLAR Intern. Conf. on Arabic Language Resources and Tools*, pages 110–117, 2004.

J. Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria A. Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL): Shared Task*, pages 1–18, 2009.

Jan Hajič. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, pages 12–19. 1998.

Jan Hajič, Eva Hajičová, Petr Pajas, Jarmila Panevova, and Petr Sgall. *The Prague Dependency Treebank 1.0, LDC No. LDC2001T10*. Linguistics Data Consortium, 2001.

Keith Hall and Václav Novák. Corrective Modeling for Non-Projective Dependency Parsing. In *Proceedings of the 9th IWPT*, pages 42–52. Association for Computational Linguistics, 2005.

Liang Huang. Forest Reranking: Discriminative Parsing with Non-Local Features. In *Proceedings of ACL-08: HLT*, pages 586–594. Association for Computational Linguistics, 2008.

David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. In *Proceedings of the IRE*, pages 1098–1102, 1952.

Tommi S. Jaakkola and David Haussler. Probabilistic Kernel Regression Models. In *Proceedings of the 7th Conference on AI and Statistics*. Morgan Kaufmann, 1999.

Fred Jelinek, John Lafferty, David M. Magerman, Robert L. Mercer, Adwait Ratnaparkhi, and Salim Roukos. Decision tree parsing using a hidden derivation model. In *Proceedings of DARPA Speech and Natural Language Workshop*, 1994.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane Training of Structural SVMs. *Machine Learning*, 77(1), 2009.

Mark Johnson. Joint and Conditional Estimation of Tagging and Parsing Models. In *Proceedings of ACL*, pages 322–329, 2001.

Mark Johnson. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24(4):613–632, 1998.

Mark Johnson, Suart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. Estimators for Stochastic "Unification-Based" Grammars. In *Proceedings of the 37th ACL*, 1999.

Ronald M. Kaplan, Stefan Riezler, Tracy H. King, John T. Maxwell, Alexander Vasserman, and Richard Crouch. Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of HLT-NAACL*, 2004.

Tadao Kasami. An Efficient Recognition and Syntax-analysis Algorithm for Context-free Languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab, 1965.

Yasuhiro Kawata and Julia Bartels. Stylebook for the Japanese treebank in VERBMOBIL. Verbmobil-Report 240, Seminar für Sprachwissenschaft, Universität Tübingen, 2000.

Gustav Kirchhoff. Über die Auflösung der Gleichungen, auf welche man bei der untersuchung der linearen verteilung galvanischer Ströme geführt wird. *The Annals of Physical Chemistry*, 72:497–508, 1847.

Jyrki Kivinen and Manfred K. Warmuth. Relative Loss Bounds for Multidimensional Regression Problems. *Machine Learning*, 45(3):301–329, 2001.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated Gradient versus Gradient Descent for Linear Predictors. *Information and Computation*, 132(1):1–63, 1997.

Terry Koo and Michael Collins. Hidden-Variable Models for Discriminative Reranking. In *Proceedings of HLT-EMNLP*, pages 507–514. Association for Computational Linguistics, 2005.

Terry Koo and Michael Collins. Efficient Third-order Dependency Parsers. In *Proceedings of the 48th ACL*, page (to appear). Association for Computational Linguistics, 2010.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. Structured Prediction Models via the Matrix-Tree Theorem. In *Proceedings of EMNLP-CoNLL*, pages 141–150. Association for Computational Linguistics, 2007.

Terry Koo, Xavier Carreras, and Michael Collins. Simple Semi-supervised Dependency Parsing. In *Proceedings of the 46th ACL*, pages 595–603. Association for Computational Linguistics, 2008.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th ICML*, pages 282–289. Morgan Kaufmann, 2001.

G. Lebanon and J. Lafferty. Boosting and Maximum Likelihood for Exponential Models. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 447–454. MIT Press, 2002.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of IEEE*, 86(11):2278–2324, November 1998.

Wei Li and Andrew McCallum. Semi-Supervised Sequence Modeling with Syntactic Topic Models. In *Proceedings of the 20th AAAI*, pages 813–818. AAAI Press, 2005.

Percy Liang. Semi-Supervised Learning for Natural Language. Master's thesis, Massachusetts Institute of Technology, 2005.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with Latent Annotations. In *Proceedings of the 43rd ACL*, pages 75–82. Association for Computational Linguistics, 2005.

David A. McAllester. On the Complexity Analysis of Static Analyses. In *Proceedings of the 6th Static Analysis Symposium*, pages 312–329. Springer-Verlag, 1999.

David McClosky, Eugene Charniak, and Mark Johnson. Effective Self-Training for Parsing. In *Proceedings of HLT-NAACL*, pages 152–159. Association for Computational Linguistics, 2006.

Ryan McDonald. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, July 2006.

Ryan McDonald and Joakim Nivre. Characterizing the Errors of Data-Driven Dependency Parsers. In *Proceedings of EMNLP-CoNLL*, pages 122–131. Association for Computational Linguistics, 2007.

Ryan McDonald and Fernando Pereira. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the 11th EACL*, pages 81–88. Association for Computational Linguistics, 2006.

Ryan McDonald and Giorgio Satta. On the Complexity of Non-Projective Data-Driven Dependency Parsing. In *Proceedings of IWPT*, 2007.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd ACL*, pages 91–98. Association for Computational Linguistics, 2005a.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530. Association for Computational Linguistics, 2005b.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual Dependency Parsing with a Two-Stage Discriminative Parser. In *Proceedings of the 10th CoNLL*, pages 216–220, 2006.

Ryan McDonald, Keith Hall, and Gideon Mann. Distributed Training Strategies for the Structured Perceptron. In *Proceedings of NAACL*, 2010.

Haitao Mi, Liang Huang, and Qun Liu. Forest-Based Translation. In *Proceedings of ACL-08: HLT*, pages 192–199. Association for Computational Linguistics, 2008.

Scott Miller, Jethran Guinness, and Alex Zamanian. Name Tagging with Word Clusters and Discriminative Training. In *Proceedings of HLT-NAACL*, pages 337–342. Association for Computational Linguistics, 2004.

Joakim Nivre and Jens Nilsson. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd ACL*, pages 99–106. Association for Computational Linguistics, 2005.

Joakim Nivre, Johan Hall, and Jens Nilsson. Memory-Based Dependency Parsing. In *Proceedings of the 8th CoNLL*, pages 49–56, 2004.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of the 10th CoNLL*, pages 221–225. Association for Computational Linguistics, 2006.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 915–932. Association for Computational Linguistics, 2007.

A.B. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. Building a Turkish treebank. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, chapter 15. Kluwer Academic Publishers, 2003.

Mark A. Paskin. Cubic-time Parsing and Learning Algorithms for Grammatical Bigram Models. Technical Report UCB/CSD-01-1148, University of California, Berkeley, 2001.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference (2nd edition)*. Morgan Kaufmann Publishers, 1988.

Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional Clustering of English Words. In *Proceedings of the 31st ACL*, pages 183–190, 1993.

Slav Petrov. Products of random latent variable grammars. In *Proceedings of the North American ACL*. 2010.

Slav Petrov and Dan Klein. Improved Inference for Unlexicalized Parsing. In *Proceedings of HLT-NAACL*, pages 404–411. Association for Computational Linguistics, 2007.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 21st COLING and 44th ACL*, pages 433–440. Association for Computational Linguistics, 2006.

Adwait Ratnaparkhi. A Maximum Entropy Model for Part-Of-Speech Tagging. In *Proceedings of the 1st EMNLP*, pages 133–142. Association for Computational Linguistics, 1996.

Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408, 1958.

Fei Sha and Fernando Pereira. Shallow Parsing with Conditional Random Fields. In *Proceedings of HLT-NAACL*, 2003.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of the 24th ICML*, pages 807–814, 2007.

Libin Shen, Jinxi Xu, and Ralph Weischedel. A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model. In *Proceedings of the 46th ACL*, pages 577–585. Association for Computational Linguistics, 2008.

David A. Smith and Noah A. Smith. Probabilistic Models of Nonprojective Dependency Trees. In *Proceedings of EMNLP-CoNLL*, pages 132–140. Association for Computational Linguistics, 2007.

Noah A. Smith and Jason Eisner. Contrastive Estimation: Training Log-Linear Models on Unlabeled Data. In *Proceedings of the 43rd ACL*, pages 354–362, 2005.

Noah A. Smith, Douglas L. Vail, and John D. Lafferty. Computationally Efficient M-Estimation of Log-Linear Structure Models. In *Proceedings of the 45th ACL*, pages 752–759. Association for Computational Linguistics, 2007.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In *Proceedings of the 12th CoNLL*, 2008.

Jun Suzuki and Hideki Isozaki. Semi-Supervised Sequential Labeling and Segmentation Using Giga-Word Scale Unlabeled Data. In *Proceedings of ACL-08: HLT*, pages 665–673. Association for Computational Linguistics, 2008.

Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. An Empirical Study of Semi-supervised Structured Conditional Models for Dependency Parsing. In *Proceedings of EMNLP*, pages 551–560. Association for Computational Linguistics, 2009.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max margin markov networks. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *NIPS*. MIT Press, 2003.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher D. Manning. Max-Margin Parsing. In *Proceedings of EMNLP*, 2004.

Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476): 1566–1581, 2006.

Ivan Titov and James Henderson. Constituent Parsing with Incremental Sigmoid Belief Networks. In *Proceedings of the 45th ACL*, pages 632–639. Association for Computational Linguistics, 2007.

Montserrat Civit Toruella and M^{a} Antònia Martí Antonín. Design principles for a Spanish treebank. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT)*, 2002.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *Proceedings of ICML*, 2004.

William T. Tutte. *Graph Theory*. Addison-Wesley, 1984.

Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*, 2002.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

Qin Iris Wang, Dale Schuurmans, and Dekang Lin. Strictly Lexical Dependency Parsing. In *Proceedings of the 9th IWPT*, pages 152–159. Association for Computational Linguistics, 2005.

Hiroyasu Yamada and Yuji Matsumoto. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the 8th IWPT*, pages 195–206. Association for Computational Linguistics, 2003.

David H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

Tong Zhang. Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms. In *Proceedings of ICML*, 2004.