EFFICIENT INFERENCE FOR TREES AND ALIGNMENTS: MODELING MONOLINGUAL AND BILINGUAL SYNTAX WITH HARD AND SOFT CONSTRAINTS AND LATENT VARIABLES

by

David Arthur Smith

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

October, 2010

Abstract

Much recent work in natural language processing treats linguistic analysis as an inference problem over graphs. This development opens up useful connections between machine learning, graph theory, and linguistics.

The first part of this dissertation formulates syntactic dependency parsing as a dynamic Markov random field with the novel ingredient of global constraints. Global constraints are enforced by calling combinatorial optimization algorithms as subroutines during message-passing inference in the graphical model, and these global constraints greatly improve on the accuracy of collections of local constraints. In particular, combinatorial subroutines enforce the constraint that the parser's output must form a tree. This is the first application that uses efficient computation of marginals for combinatorial problems to improve the speed and accuracy of belief propagation. If the dependency tree is projective, the tree constraint exploits the inside-outside algorithm; if non-projective, with discontiguous constituents, it exploits the directed matrix-tree theorem, here newly applied to NLP problems. Even with second-order features or latent variables, which would make exact parsing asymptotically slower or NP-hard, approximate inference with belief propagation is as efficient as a simple edge-factored parser times a constant factor. Furthermore, such features significantly improve parse accuracy over exact first-order methods. Incorporating additional features increases the runtime additively rather than multiplicatively.

The second part extends these models to capture correspondences among non-isomorphic structures. When bootstrapping a parser in a low-resource target language by exploiting a parser in a high-resource source language, models that score the alignment and the correspondence of divergent syntactic configurations in translational sentence pairs achieve higher accuracy in parsing the target language. These noisy (quasi-synchronous) mappings have further applications in adapting parsers across domains, in learning features of the syntax-semantics interface, and in question answering, paraphrasing, and information retrieval.

Primary Reader: Jason Eisner

Secondary Reader: David Yarowsky

Tertiary Reader: Sanjeev Khudanpur

Acknowledgements

Education incurs unpayable debts. It is a curious mixture of unknowably delayed and surprisingly instant gratification; many people have transferred momentum to get me through the former and shared my joy in the latter.

I want to thank Jason Eisner for the energy and technical insight he provided throughout our collaboration. Jason is relentless in peeling away the layers of cruft to find the core research problems. But I am most impressed with his aptitude and enthusiasm for teaching, in lectures, seminars, or one-on-one; he is constantly looking for several different ways to explain an idea. I look forward one day, if digital preservation specialists do their jobs, to reading the collected technical emails of Jason Eisner—the most characteristic expressions of his teaching style.

I want to thank Jason Eisner and David Yarowsky for making the Natural Language Processing Lab a congenial and exciting place to do research. Sanjeev Khudanpur, Keith Hall, Damianos Karakos, and Chris Callison-Burch, Bill Byrne, and the late, inimitable Frederik Jelinek made the Center for Language and Speech Processing—especially its seminars and student seminars—an intellectual home.

ACKNOWLEDGEMENTS

The best educators at most universities are your fellow students. With Noah Smith, I have enjoyed a long and productive collaboration, mostly so we would have an excuse to go to lunch and visit Pittsburgh in February. Roy Tromble put up with my ranting and critiqued my ideas, and he and Nicole helped make Baltimore feel a bit more like home. Markus Dreyer helped me keep a sense of perspective; Gideon Mann and Charles Schafer provided technical insight and snark. John Blatz du Rivage, Silviu Cucerzan, Elliott Drábek, Erin Fitzgerald, Eric Goldlust, Radu "Hans" Florian, Nikesh Garera, Arnab Ghoshal, Shankar Kumar, Zhifei Li, Lambert Mathias, Delip Rao, Yi Su, Paola Virga, and Peng Xu made CLSP safe for research and weekly doggerel.

Alex Fraser, Dan Gildea, Franz Och, Anoop Sarkar, Libin Shen, and Kenji Yamada made the 2003 CLSP workshop more enjoyable, despite having our disks wiped. Franz was also kind enough to let me spend the summer of 2005 with Google's crack MT team.

Greg Crane and the Perseus Project, first at Harvard and then at Tufts University, provided invaluable guidance and support in turning my interest in the classics into a career in computational linguistics. In the process, working at Perseus gave me the practical experience to work in an empirical discipline. I owe my thanks to many Perseids: Elli Mylonas, Jacob Sisk, Maria Daniels, Lisa Cerrato, Jeff Rydberg-Cox, Anne Mahoney, and Robert Chavez.

On the nearer shore of my graduate career, I want to thank Bruce Croft, James Allan, and Manmatha of the Center for Intelligent Information Retrieval at the University of Massachusetts, Amherst, for the chance to start new collaborations and Andy Barto and the

Computer Science Department for their hospitality.

My parents, Susan Crouse Smith and Arthur Larry Smith, kindled my interest in language and my appreciation for mathematics. To them, to my brothers Peter and James, and to Ralph and Judy Mentzer and Samantha and Rolf Loken, I owe my gratitude for their never-failing help and encouragement.

More than anyone else, however, I want to thank my wife Cynthia Mentzer. She has been my sounding board and first line of mental defense against all the frustrations that threaten to derail a long academic journey. For that and for all that we do together that brings us joy, I dedicate this dissertation to her.

To Cynthia

The winter is past. The rain is over and gone.
The flowers appear upon the earth.
The time of the singing of birds is come.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Computer science and related disciplines represent many of their core problems with graphs. Graph algorithms form an important component in introductory CS courses and in the development and teaching of complexity theory. Much recent work in natural language processing has taken advantage of this legacy to formulate linguistic analysis as a graph inference problem: dependency trees, word alignments, and coreference chains may all be represented as graph structures, and, given an appropriate problem formulation, efficient algorithms exist to search for optimal graphs.

In this introductory chapter, we work out some of the implications of this simple formulation. In particular, we will see how a **declarative specification** of linguistic inference problems, in terms of hard and soft constraints on solutions, unifies recent work in natural language processing (§1.1) and linguistics (§1.2). Moreover, we will see how algorithmic tools from graph theory and machine learning (§1.4) can make linguistic inference more efficient and how approximation algorithms can find good solutions to intractable problems.

1.1 Linguistic Inference as Graph Inference

If, as in figure 1.1, we specify the words of a sentence as nodes, the task of finding a dependency tree for those words can be formulated as a maximum directed spanning tree problem (McDonald et al., 2005b). Figure 1.2 illustrates finding a word alignment between German and English sentences by bipartite matching. If we want to allow a richer set of alignments, e.g. with 2-to-1 correspondences among words, we can use the more general technique of network flow (Taskar et al., 2005; Lacoste-Julien et al., 2006). Perhaps most familiar in NLP, the Viterbi algorithm (Viterbi, 1967) for finding the best state sequence in a hidden Markov model is equivalent to finding the shortest path in a directed graph known as the trellis.

Casting NLP tasks as graph problems allows us to exploit many classical results in graph theory and algorithms. Dependency parsers can use the cubic-time Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) or its trickier quadratic-time variant (Tarjan, 1977) to find the best combination of edges that form a spanning tree. Word alignment systems can use linear programming or specialized bipartite matching algorithms such as Ford-Fulkerson or Edmonds-Karp. NLP researchers can draw on years of experience in implementing these and similar algorithms to find efficient solutions to many problems.

These standard algorithms, however, only apply to graph inference problems under certain conditions. To apply MST algorithms to parsing, the score of a tree must be *edge factored*—the product of the scores of individual dependency edges in a tree. Likewise, bipartite matching and network flow algorithms work with cost functions that factor into scores on individual alignment links. These restrictions preclude natural expressions of many linguistic constraints. In the sentence in figure 1.1, we would like to capture the correlation between "answer" and "question" via the preposition "to". That correlation, however, is mediated by two separate dependency links, and an edge-factored model cannot express our preference to have both links present in the final tree (McDonald and Satta, 2007).

Similarly, in the aligned sentences in figure 1.2, observe that the three-word phrase "Auf diese Frage" at the beginning of the German sentence is translated, as a unit, by "to this question" at the end of the English sentence. A model that scores the alignment of isolated German–English word pairs cannot explicitly capture this useful notion of "monotonicity", i.e. the alignment of source word s_i with target word t_j makes the alignment of s_{i+1} with t_{j+1} more likely.[1]

Dijkstra's dynamic programming algorithm to find the shortest (directed) path through a graph has a long history of applications in computer science (Dijkstra, 1959). Viterbi (1967) described a specialized version where the nodes are topologically sorted (as they would be when corresponding to time steps in a hidden Markov model) and the graph is

[1]In conditional or discriminative edge-factored models of both dependencies and alignments, feature functions may use arbitrary amounts of the observed input, such as relative word positions. Although less naturally expressed, such features can recover some of the benefits of scoring large pieces of structure.

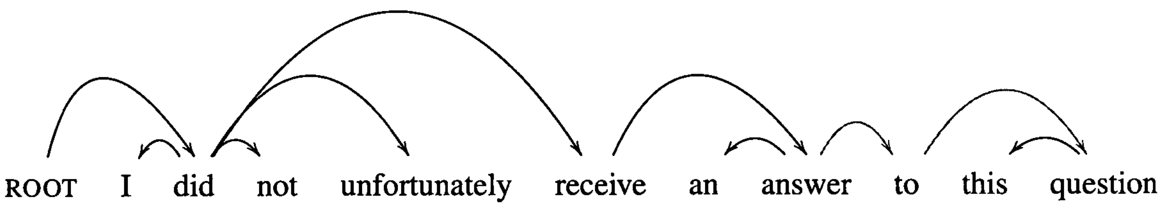

ROOT I did not unfortunately receive an answer to this question

Figure 1.1: A dependency graph: the words are nodes connected by a direct spanning tree, or a spanning arborescence. A model that only considered the acceptability of individual edges is unable to express generalizations about pairs of edges, such as the dependency trigram *answer* $\rightarrow$ *to* $\rightarrow$ *question*.

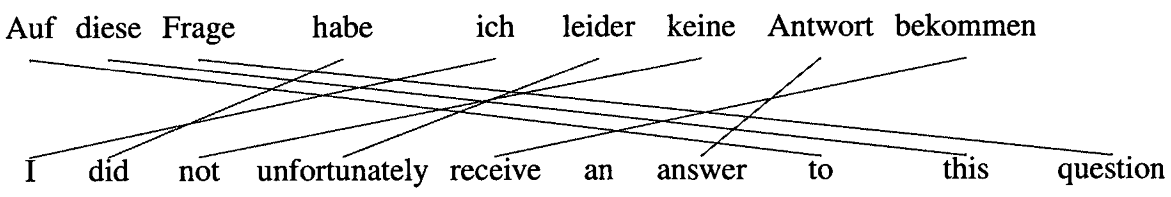

Auf diese Frage habe ich leider keine Antwort bekommen
I did not unfortunately receive an answer to this question

Figure 1.2: A word alignment as a bipartite graph, whose edges connect German and English words. A model that only scored individual word-to-word alignments would be unable to ensure that the contiguous German phrase "Auf diese Frage" ought to correspond to the contiguous English phrase "to this question".

acyclic. This Viterbi algorithm can then infer the most likely sequence of hidden states. Consider a hidden Markov model of part-of-speech tags where successive tags have a first-order Markov dependency—i.e., a bigram tagging model. With T tags, the Viterbi algorithm decodes a sequence of length n in $O(T^2n)$ time. A trigram model's state space has cardinality $O(T^2)$, so with appropriate restrictions on legal transitions, the Viterbi algorithm takes $O(T^3n)$ time. It is easy to show that additional dependencies among labels an unbounded distance apart would require the state space of the Viterbi algorithm to grow exponentially in n. Despite their intractability, long-distance constraints on labels are useful in, for example, expressing our intuitions about word-sense variation. A "one sense per discourse" constraint (Gale et al., 1992) has led to significant improvements in named-entity classification systems since in the vast majority of documents, all instances of "Lincoln" will be to the sixteenth U.S. president, or the senator from Arkansas, or the capital of Nebraska, or the car model, but not some mixture of them (Sutton and McCallum, 2004; Finkel et al., 2005).

Other theories of language formulated as graphs do not even admit polynomial-time inference from input utterances to output structures in their simplest incarnations. Instead of trees, for instance, some researchers have developed theories of syntax as directed acyclic graphs (DAGs), a strict superset of directed trees (Chen-Main, 2006; Buch-Kromann, 2006; Hudson, 2007). Even with edge-factored costs, a parser cannot exactly search the space of DAGs in polynomial time (Karp, 1972).

Computational linguists working to express linguistic analysis as a graph inference task have increasingly, therefore, turned to approximation algorithms. In the past, NLP researchers have commonly used heavy pruning, reranking n-best lists, or rescoring packed representations, such as lattices and parse forests. Many are now turning to formulating their inference problems in general-purpose frameworks such as integer linear programs (ILP) or graphical models. Exact inference algorithms exist for ILP and graphical models, but they are in general intractable. Some constraints may have to be relaxed in order to find a solution quickly. NLP has thus benefited from machine learning methods such as Gibbs sampling, rejection sampling; form certain variational approximations such as mean field; and from work in linear programming relaxations. We can view many of these techniques as performing constrained optimization: find a graph with an optimal score such that it satisfies certain hard constraints such as being a tree, or DAG, or a matching.

In this thesis, we propose borrowing another variational approximation from machine learning, namely, **loopy belief propagation** (BP). One advantage of this formulation is that, unlike some other general-purpose approaches such as ILP, we can exploit the well-studied combinatorial structure of the graph problems discussed above: our BP implementation can call simple parsers, or aligners, or linear sequence labelers, as subroutines. We sketch the BP algorithm in §1.4 below. But there are other reasons to prefer this setup of declarative constraints plus general-purpose inference. We will now briefly step back from NLP and trace some strands within linguistics that also lead to formulating language problems as systems of constraints.

1.2 Grammars and Constraints

Computer science inherited many of the graph algorithms described above from its antecedents in applied mathematics and operations research. With computational linguistics, computer science also shares the heritage of formal languages and complexity—in particular, the Chomsky hierarchy of languages and their attendant recognition algorithms (Hopcroft and Ullman, 1979). Formal grammars are perhaps most familiarly presented as rewriting systems, with a generative story from the root symbol to terminal leaves. Work on probabilistic context-free grammars in NLP takes the now-natural step of assigning probabilities to each rewrite rule, conditioned on its left-hand-side nonterminal.

The Chomsky hierarchy also influenced a strain of work in theoretical linguistics that aimed to put an upper bound on the complexity of natural language syntax. In the late 1970s and early 1980s, many researchers argued over whether English was context-free (Pullum, 1984); others exhibited constructions in Dutch and Swiss German with productive non-context-freeness (Shieber, 1985). Several other formalisms, such as tree adjoining grammar and combinatory categorial grammar, were shown to have equivalent mildly context-sensitive power (Joshi et al., 1991) and to facilitate accounts of various linguistic cruces (Steedman, 2000; Frank, 2002).[2]

Many traditional and generative linguists, however, have highlighted declarative constraints on grammaticality, rather than a derivational approach. Head-Driven Phrase Struc-

[2]Investigations of the "context-freeness of natural language" focus on the complexity of syntax, but this does not preclude proposals for tighter, finite-state bounds on subsystems such as phonology and morphology.

ture Grammar, Lexical Functional Grammar, and Optimality Theory in particular are presented as systems of constraints.[3] The possibility of more precise formal specification and computational implementation motivated many non-generative linguistic theories. Indeed, several broad-coverage syntactic parsers have been built according to constraint-based theories (e.g. Johnson et al., 1999; Riezler et al., 2000; Sagae et al., 2007).

Rather than focusing directly on the description of processes to generate syntactic structures, or to transform them, constraint-based formalisms aim at declarative descriptions of grammatical structures. Since a formal grammar is not directly specified, these approaches have been called "grammarless" (Rogers, 1997). The use of declarative representations, however, does not imply an inability to reason about the complexity of language recognition or parsing problems. The field of descriptive complexity concerns the mapping of problem descriptions in logic onto complexity classes (Immerman, 1999). This description of computational objects, specifically languages, in logic has given rise to a characterization of much of the constraint-based research program as "model-theoretic" (Rogers, 1996; Potts and Pullum, 2002; Pullum, 2007).

The example of Optimality Theory (OT) is particularly instructive in light of the methods we will develop in this thesis. Some grammar formalisms only employ **hard constraints**: no structure that violates a hard constraint is grammatical. In OT, an underlying input form is associated with a candidate set of possible output structures aligned to

[3]Sag et al. (2004) and Falk (2001) provides good introductions to HPSG and LFG, respectively. Optimality Theory (Prince and Smolensky, 2004) has exercised the greatest influence in phonology; for optimality-theoretic syntax, see the papers in Legendre et al. (2001) and an OT encoding of LFG constraints in Kuhn (2003).

the input. OT posits that grammatical forms are selected from among the candidates by systems of **violable** constraints in **strict dominance**. The grammatical, observed form's highest-ranked violated constraint will be the lowest ranked constraint violated by any of the forms.

In an interesting further development, some recent work in OT has turned towards so-called "log-linear grammar" or "maximum entropy grammar": instead of being ranked, OT's violable constraints are assigned real-valued weights, and the candidate form with the lowest sum of weighted violations is selected (Goldwater and Johnson, 2003; Hayes and Wilson, 2008). With a change of sign, this is equivalent to the **linear models**, which are now widely used in NLP and machine learning generally. In other words, for a given input $\mathbf{x}$ the linear model selects from among the candidates $\mathcal{Y}_{\mathbf{x}}$ the output $\mathbf{y}^*$ with the *highest* sum of real-valued *features* $\mathbf{f}$ weighted by real-valued *weights* $\mathbf{w}$:

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \tag{1.1}$$

In OT, and in linear models generally, the features $\mathbf{f}$ are functions of the input and the candidate output $\mathbf{y}$. (In this thesis, we will also refer to features with finite values as **soft constraints**.)

We can also define a probability distribution over outputs given inputs by exponentiating and renormalizing this sum (hence the name log-linear).

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} e^{\mathbf{w} \cdot \mathbf{f}(\mathbf{x},\mathbf{y})} \tag{1.2}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}'} e^{\mathbf{w} \cdot \mathbf{f}(\mathbf{x},\mathbf{y}')} \tag{1.3}$$

An important advantage of linear and log-linear models with real-valued weights is the ability to use standard algorithms from machine learning. In addition, the probabilistic interpretation allows log-linear models to sum over hidden representations (e.g., in phonology, Pater et al., 2010). For further background on log-linear models and their properties, see appendix A.

In this spirit, we investigate log-linear models of syntactic structure that are parametrized by constraints on substructures of linguistic analysis. The feature functions $\mathbf{f}$, in other words, will usually depend only on finite subsets of the input and output and ask such questions as: how good is this dependency link? how compatible are these two dependency links? how many children should this verb take? should this German compound word align with this three-word English phrase? Constraints on substructures are necessary to capture the basic insight of generative linguistics: grammars describe infinite sets of strings with finite resources.

1.3 Probabilistic Structured Models

So far, we have mentioned one inference problem we wish to solve with linguistic models: given an input, output space, feature functions, and weights, find the output with the highest score (equation 1.1). But how, for instance, should we set the weight vector $\mathbf{w}$ if we have labeled examples in the form of sentences and their true dependency trees? And what other inference problems might be useful? Finally, should we make use of hidden (or latent) variables, i.e. random variables that are not in the input, and are not specified or desired in the output, but that hopefully help us to relate the two?

A wide variety of approaches have been developed for training the weights of the linear models described so far. Among the most popular are the perceptron and support vector machine. Given certain conditions, these methods provide a training procedure guaranteed to converge. Both are designed to minimize the number of errors made by the model on training data—subject to constraints, in the case of the SVM, on the L_2 norm of the weight vector that improve generalization performance.

Log-linear models, also known as maximum entropy or (multiclass) logistic regression models, are a natural extension of linear models that turn output scores into probabilities. There are several reasons to adopt this probabilistic framework:

- Maximum likelihood estimation provides a well-motivated training procedure for $\mathbf{w}$ with or without hidden variables. When there are no hidden variables, the maximum likelihood objective is convex.

- In addition to identifying the best tree, the model can also compute posterior probabilities of output trees or subtrees. For example, we can determine the probability that a certain noun is the subject of a certain verb, considering the model's uncertainty about other parsing decisions.

- The posterior distribution over outputs allows us to calculate the expected cost of making decisions. These expectations enable minimum risk training and minimum

Bayes risk decoding , which take a loss function into account and often lead to better results. We can also calculate measures of uncertainty such as entropy.

- We can naturally model hidden variables. Most algorithms for error minimization, however, assume that *all structures are fully observed* at training time. When modeling syntax, for example, which depends on certain morphological properties such as words, it is often helpful to take our uncertainty about the morphological analysis into account. When evaluating whether one sentence is a good translation of another, we would like to model our uncertainty about how individual words or phrases are aligned.

- Probabilistic models facilitate well-founded system combination, e.g. noisy channel models, mixtures, and products of experts. In addition to factored training of individual systems, we can train the pipeline jointly while integrating out the intermediate results.

The advantage is not all on one side, of course. Probabilistic models compare unfavorably to error-driven training in other ways:

- Unlike some models such as SVMs, probabilistic models lack provable, distribution-free bounds on generalization error.

- Learning is often more expensive for globally normalized probabilistic models—indeed, asymptotically more expensive. For instance, we can solve maximum weighted bipartite matching for an edge-factored word alignment model in cubic time, but the summation problem necessary for maximum likelihood training is equivalent to computing the permanent of a matrix, a #P-complete problem (Taskar et al., 2005).

- Training procedures that only consider the best output of the current model need to examine the features only of the true output and the one-best output, rather than features in all valid outputs (Roark et al., 2004).

We have hinted at a final consideration that is in practice somewhat correlated with the probabilistic divide. The convexity of inference problems—and thus our ability to find the globally optimal solution without brute force methods—weights heavily in many decisions about machine learning methods. As noted above, when there are no hidden variables, maximum likelihood training, shares with SVM and perceptron learning a guarantee of convexity. The likelihood function of log-linear models with hidden variables is no longer convex in its parameters, so gradient-based training is no longer guaranteed to find the global optimum. The problem of local optima also affects error-driven training of neural networks with hidden layers. Even without hidden variables, a loss-aware objective function such as risk has local optima, though in practice minimum risk training can be more effective than maximum likelihood. Techniques such as deterministic annealing can help us find better optima, though without formal guarantees (Rao and Rose, 2001; Smith and Eisner, 2006a).

1.4 Approximate Inference with Belief Propagation

To solve the inference problems required by structured models, we propose the approximation technique of loopy belief propagation (BP). In this thesis, we show that BP can be used to *train* and *decode* complex models for monolingual and bilingual parsing.

Our BP approach introduces a new ingredient: the parser calls simpler algorithms as subroutines, so it still exploits the useful, well-studied combinatorial structure of the graph problems we reviewed in §1.1. Recently, decoders employing belief propagation (Duchi et al., 2006), constraint relaxation (Tromble and Eisner, 2006), and forest reranking (Huang, 2008) have called simpler combinatorial solvers. In contrast, generic NP-hard solution techniques like Integer Linear Programming (Riedel and Clarke, 2006) do not exploit combinatorial substructure. Ours is the first application, however, that uses efficient computations of marginals for these subproblems to speed up belief propagation.

When applying our method to dependency parsing, we wish to make a tree's score depend on *higher-order* features, which consider arbitrary interactions among two or more edges in the parse (and perhaps also other, latent variables such as part-of-speech tags or edge labels). Such features can help accuracy—as we show. Alas, they raise the polynomial runtime of projective parsing, and render non-projective parsing NP-hard (McDonald and Satta, 2007). Hence we seek approximations.

We will show how BP's "message-passing" discipline offers a principled way for higher-order features to incrementally adjust the numerical edge weights that are fed to a fast *first-order* parser. Thus the first-order parser is influenced by higher-order interactions among edges—but not asymptotically slowed down by considering the interactions itself.

BP's behavior in our setup can be understood intuitively as follows. Inasmuch as the first-order parser finds that edge e is probable, the higher-order features will kick in and discourage other edges e' to the extent that they prefer not to coexist with e.[4] Thus, the *next* call to the first-order parser assigns lower probabilities to parses that contain these e'. (The method is approximate because a first-order parser must equally penalize *all* parses containing e', even those that do not in fact contain e.)

This behavior is somewhat similar to parser stacking (Nivre and McDonald, 2008b; Martins et al., 2008), in which a first-order parser derives some of its input features from the full 1-best output of another parser.[5] In our method, a first-order parser derives such input features from its *own* previous full output (but probabilistic output rather than just 1-best). This circular process is *iterated* to convergence. Our method also permits the parse to interact cheaply with other variables. Thus first-order parsing, part-of-speech tagging, and other tasks on a common input could mutually influence one another.

Our approach to efficient inference requires an innovation to BP—the use of combinatorial algorithms to enforce *global* constraints, e.g. that the parse is a tree. The tractability of some such global constraints points the way toward applying BP to other computation-

[4]This may be reminiscent of adjusting a Lagrange multiplier on e' until some (hard) constraint is satisfied.
[5]§2.3.4 elaborates the connections among BP and iterated methods such as stacking and reranking.

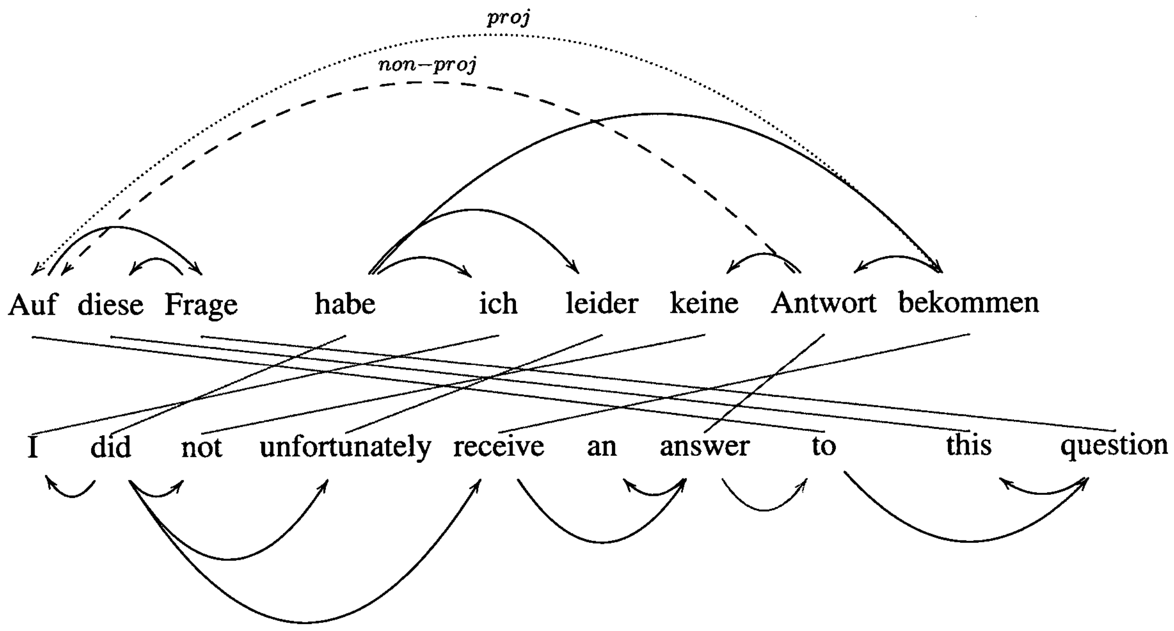

proj
non–proj
Auf diese Frage habe ich leider keine Antwort bekommen
I did not unfortunately receive an answer to this question

Figure 1.3: Multi-structure inference: Bilingual parsing. The extraposed German phrase *Auf diese Frage* leads to a non-projective dependency link, labeled *non-proj*, that crosses the link from *habe* to *bekommen*. Some treebanks and parsers prefer completely projective trees and would reattach this link higher up—here, the link labeled *proj* whose parent is *bekommen*. In this example, the nonprojective $Auf \leftarrow Antwort$ link reflects the $answer \rightarrow$ *to* relationship from the English.

ally intensive NLP problems: in chapter 3, we extend our BP parser to handle nonprojective trees, with crossing dependency links (figure 1.3) and discontinuous constituents; in chapter 4, we present efficient inference for aligned trees; and in chapter 5, we sketch possible future applications from morphology to semantics, and from machine translation to information retrieval.

With the development of the BP machinery to coordinate a first-order parser with other constraints, it is easy to construct procedures for **multi-structure inference.**[6] Figure 1.3, for example, depicts the output of a bilingual parser as a combination of two dependency trees and an alignment. Figure 1.4 shows a version of the semantic role labeling task for assigning argument structures, with a dependency tree above and a non-tree semantic graph below (Hacioglu, 2004). Inference on some of these subgraphs can take advantage of the combinatorial algorithms mentioned in §1.1 above, and BP negotiates among these subgraphs to find approximately optimal solutions.

Loopy BP has occasionally been used before in NLP, with good results, to handle non-local features (Sutton and McCallum, 2004) or joint decoding (Sutton et al., 2004). There is also a line of work in syntactic parsing that relies on unweighted hard constraints (Maruyama, 1990; Harper et al., 1995; Duchier and Debusmann, 2001; Duchier, 2003; Wang, 2003; McCrae et al., 2008; Foth et al., 2006; Debusmann, 2006). For inference, these systems use algorithms, related to belief propagation, for enforcing **arc consistency** (Mackworth, 1977).[7] Some of these constraint-based parsers have broad coverage; others describe a fragment of a language in order to explore theories of human sentence processing. The problem of learning a good model is, however, less elegantly handled when we are restricted to hard constraints. Standard error-driven and probabilistic training methods, such as the perceptron and maximum likelihood, cannot be applied when feature weights

[6]The term is due to Noah Smith, who helped clarify our thinking on this topic. In addition to providing a framework for many joint inference problems in NLP, multi-structure inference could be useful for implementing theories of syntax that are explicitly presented as aligned hierarchical structures, e.g. LFG, with c-structure and f-structure; Autolexical Grammar (Sadock, 1991); and, less formally, the multistratal approach of Jackendoff (2002).

[7]In a parallel to our work, Régin (1994) employed a bipartite matching algorithm within unweighted constraint propagation; see §2.4.2.

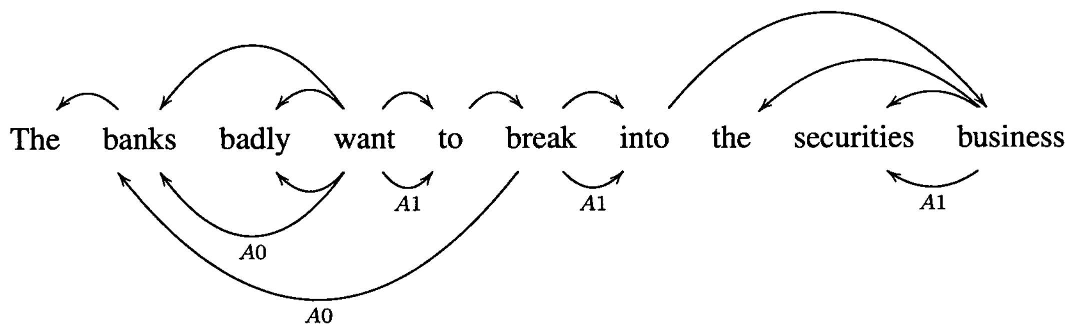


Figure 1.4: Multi-structure inference: Semantic role labeling. The dependency tree is drawn above the sentence; the (non-tree) semantic graph is below. The *banks* fills the $A0$ ("agent") argument of both the *want* and *break* predicates.

are constrained to be $\log 1 = 0$ or $\log 0 = -\infty$. Instead, researchers either design a set of hard constraints by hand or relax a set of possible constraints until the training data can be parsed.

1.5 A Map of the Thesis

In chapter 2, we elaborate on the graph-based approach to dependency parsing and frame it as variable assignment in a graphical model. We introduce a method to embed simple parsers within a larger graphical model.

In chapter 3, we show how to extend these results for parsing non-projective languages, where dependency edges may cross.

Chapter 4 applies the forgoing results on efficient inference to adapting syntactic models among different domains, within and between languages.

Chapter 5 sketches applications of these techniques to problems in syntax, adaptation, and other noisy mappings among combinatorial structures.

Chapter 6 summarizes the modeling and algorithmic contributions of this thesis and its implications for further work in NLP.

Chapter 2

Dependency Parsing by Belief Propagation

In this chapter, we describe the application of loopy belief propagation to the structured prediction problem of dependency parsing.[1]

2.1 Dependency Syntax

Dependency parsers are modern representatives of an old idea. Before the American structuralists pioneered constituency analysis, grammatical treatments of syntax often focused on the **relations** among words (as well as their linear order) (Tesnière, 1969; Mel'čuk, 1988; Sleator and Temperley, 1993; Hudson, 2007). In figure 1.1, the noun "ques-

[1]A condensed version of this material appeared in (Smith and Eisner, 2008, §§1–8.4). This chapter provides a more thorough exposition of the belief propagation algorithm and constraints and features for dependency parsing. Our paper also discussed nonprojective parsing, which we take up in chapter 3.

tion" (the head, or parent) is said to govern, or be modified by, the determiner "this" (the child). Many theories and models in constituency syntax also take these relations between heads and modifiers into account.

Recently, dependency parsing has received renewed interest, both in the parsing literature (Buchholz and Marsi, 2006; Nivre and McDonald, 2008a) and in applications like translation (Alshawi et al., 2000; Quirk et al., 2005; Galley and Manning, 2009) and information extraction (Culotta and Sorensen, 2004; Bunescu and Mooney, 2005). Dependency parsing can be used to provide a "bare bones" syntactic structure that approximates semantic argument structure, and it has the additional advantage of admitting fast parsing algorithms while maintaining high accuracy (Eisner, 1996; McDonald et al., 2005a).

Many linguistic annotation projects are now creating dependency treebanks—e.g., the long-running Prague Dependency Treebank project (Böhmová et al., 2003). Standard evaluation benchmarks such as CoNLL Shared Tasks have encouraged progress in automatic dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007).

2.1.1 Models of dependency trees

The latest state-of-the-art statistical dependency parsers are **discriminative**, meaning that they are based on classifiers trained to score output trees, given an input sentence.

In one widely-used class of models, a tree is constructed by the output of local decisions made in sequence, hence the names *transition-* or *action-based* parsers (Nivre, 2003; Hall et al., 2006). These parsers are similar to the shift-reduce parsers for context-free grammars, which have been used in NLP for some time (e.g., Ratnaparkhi et al., 1994). Transition-based models for sequence labeling are often called maximum-entropy, or conditional, Markov models McCallum et al. (2000).

In this thesis, we are concerned with another major class of discriminative dependency parsers, which treat syntactic analysis as graph optimization, in the manner we surveyed in §1.1. Conceptually, these *graph-based* parsers assign a score to each possible output tree and search for the output with the highest score. More specifically, we are interested in the conditional, probabilistic versions of graph-based models, for reasons sketched in §1.3 above: maximum likelihood training and the natural handling of hidden variables.[2]

In addition to transition-based and graph-based models, generative models of dependency trees (Eisner, 1996), the analogues of HMMs and PCFGs, have also been widely used. While such models do not currently achieve state-of-the-art performance on dependency parsing tasks, they are useful for language modeling in, e.g., speech recognition (Wang and Harper, 2002) and machine translation (Shen et al., 2008). An interesting twist on such models combines a simple generative component with multiple, overlapping *hard* constraints (Wang, 2003).

2.1.2 Constraints on dependency trees

As noted above, global models of graph structure have produced state-of-the-art results on many dependency parsing tasks. For a given input sentence $\mathbf{x}$, these models define a set

[2]The analogous models for sequence labeling are known as linear-chain conditional random fields (CRFs) (Lafferty et al., 2001).

of possible output dependency trees and assign a **score** to each tree $\mathbf{y}$ with the real-valued function $s(\mathbf{x}, \mathbf{y})$. Parsing is the task of finding the tree with the highest score.

More formally, let $\mathbf{x} = \langle x_1, ..., x_n \rangle$ be a sequence of words (possibly with POS tags, lemmata, and morphological information) that are the input to a parser. The output $\mathbf{y}$ will refer to a directed, unlabeled dependency tree, which is a map $\mathbf{y} : \{1, ..., n\} \rightarrow \{0, ..., n\}$ from child indices to parent indices; x_0 is the invisible "root" or "wall" symbol.

Let $\mathcal{Y}_\mathbf{x}$ be the set of valid dependency trees for $\mathbf{x}$. We define a valid tree as a directed graph over nodes $\mathbf{x}$ such that every word has **in-degree 1**, with the sole edge pointing from the word's parent, $x_{\mathbf{y}(i)} \rightarrow x_i$, except for the root x_0, which has in-degree 0. In addition, trees are **acyclic**. In other words, the graph is a **directed spanning tree**, or a **spanning arborescence**, rooted at x_0.

Much work on dependency parsing imposes a third condition on membership in $\mathcal{Y}$: the tree must be **projective**. Projectivity is usually stated as a prohibition on descendants of one node "intruding" on the span of the descendants of another node not its ancestor—in short, as a prohibition on *discontinuous constituents* (Nivre, 2003). To be precise,

$$(\forall i, j, k)\ ((x_i \rightarrow x_j \vee x_j \rightarrow x_i) \wedge i < k < j) \Rightarrow (x_i \rightarrow^* x_k \vee x_j \rightarrow^* x_k) \qquad (2.1)$$

where $\rightarrow^*$ is the transitive closure of the child, i.e. the descendant, relation.

It is sometimes more convenient, as we shall see below (§3.4.2), to express projectivity as a constraint on *all pairs of edges* rather than on arbitrarily long descendant chains. For each of two dependency links $x_p \rightarrow x_c$ and $x_{p'} \rightarrow x_{c'}$, consider their linear order in the sentence, according to the indices of the nodes in $\mathbf{x}$. Let $\ell = \min(p, c)$ and $r = \max(p, c)$;

similarly, let $\ell' = \min(p', c')$ and $r' = \max(p', c')$. A projective tree is a tree such that:

$$\neg[(\ell < \ell' \wedge \ell' < r \wedge r < r') \vee (\ell' < \ell \wedge \ell < r' \wedge r' < r)] \tag{2.2}$$

Both formulations of projectivity also apply to dependency links from the root node x_0.

Many graph-based and transition-based dependency models use the projectivity constraint, both because it is often observed in natural languages and the treebanks that try to reflect them and because efficient and effective greedy (Nivre, 2003) and dynamic programming algorithms exist for many inference problems for projective trees (Eisner, 1996, and appendix B below). But the projectivity constraint is not always observed—indeed, it is violated quite often in treebanks for Czech, Dutch, and other languages with the loosely-defined quality of “free word order”. In chapter 3, we will turn to algorithms for nonprojective trees.

Many treebanks observe a final constraint: exactly one node in $\{x_1, \ldots, x_n\}$ should have an in-edge from x_0. Trees or treebanks that obey this constraint are *single-rooted*; otherwise, they are *multirooted*.

Dependency structures extracted from constituency treebanks are usually projective and single-rooted. Dependency treebanks often use multirooted structures to attach final punctuation and clauses in parataxis (e.g., Arabic *wa*-consecutive) directly to the root node. Some conversions from constituency to dependency structure, however, do introduce nonprojectivity—for example, by reattaching an extraposed element to the parent of its trace.

2.1.3 Edge-factored models

Edge-factored models are a useful special case of the graph-based approach. Recall that in a log-linear parsing model, the score of a tree (1.1) is the log of its unnormalized probability (1.2). In an edge-factored model, the score of a tree is the sum of the scores of its component dependency links. Let the real-valued score of a dependency link from x_i to x_j be a real-valued function $s(i, j)$ than depends (implicitly) on the input x and that single edge. The score of an input-output pair is then

$$s(\mathbf{x}, \mathbf{y}) = \sum_{x_i \to x_j \in \mathbf{y}} s(i, j) \tag{2.3}$$

and the probability of output given input is

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z} \prod_{x_i \to x_j \in \mathbf{y}} e^{s(i,j)} \tag{2.4}$$

We define $u(i, j) = e^{s(i,j)}$ for convenience.

Edge-factored models admit exact, efficient algorithms for finding the best tree, for finding the total weight of all trees (the normalizer Z), and for finding the marginal probabilities of individual edges are features. These inference problems can be solved by cubic-time dynamic programming for projective parsing (see Eisner, 1996, and the algorithms in appendix B). In addition, dependency parsers, which do not infer hidden structure beyond a linear number of edges, can avoid the grammar constants that burden treebank-based constituency parsers with a large number of rules. These advantages have led to widespread use of edge-factored parsers (McDonald et al., 2005b).[3] While we will take up nonpro-

[3]For projective models, some scoring functions that consider pairs of edges also admit efficient algorithms (Eisner, 2000; McDonald and Pereira, 2006).

jective parsing in chapter 3, we confine ourselves to projective models for the rest of this chapter.

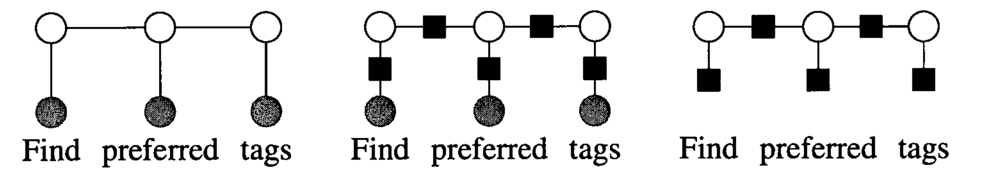
2.2 Graphical Models of Dependency Trees

Having defined the constraints on the output graph of our parser (projective directed trees), we now turn to the modeling architecture we will use to evaluate candidate outputs. Our model is expressed as a factor graph, a particular class of graphical models that uses a bipartite undirected graph to express the independence assumptions and computational relations of different parts of the model. Since we are using graphs to model graphs, we will, to avoid terminological confusion, use "links" to refer to edges in the output dependency tree and reserve the term "edge" for describing the graphical model's factor graph.

2.2.1 Factor graphs

We will model distributions over structures, including dependency graphs, in terms of graphical models known as **factor graphs** (Kschischang et al., 2001). We first introduce these objects informally by appealing to a familiar special case: linear-chain conditional random fields. Later, in §2.2.3, we will provide more rigorous definitions.

Figure 2.1a depicts a linear-chain conditional random field (CRF) for tagging a simple three-word sentence. The open nodes represent the hidden variables, which take on values from the inventory of tags. For simplicity, let the available tags be a, n, and v for adjective,



(a) Conditional random field (b) Factor graph (c) Observations absorbed by factors

Figure 2.1: Graphical models for sequence labeling

noun, and verb. The closed nodes represent the observed words of the sentence. Edges between an observed and a hidden variable represent the compatibility of a word with a tag; edges between neighboring hidden variables represent the first-order Markov dependence among tags (Lafferty et al., 2001).

A factor graph makes variable dependencies more explicit. Graphically, we express dependencies among variables as square black nodes connected to variables (figure 2.1b). Since variables depend on each other only through factors, the graph is now bipartite.

We have simplified the figures by linking each hidden tag variable to exactly one observed word variable. This is by analogy to hidden Markov models with their tag-to-word emission probabilities. Conditional random fields achieve much of their advantage over HMMs in practice by allowing the model designer to condition each hidden variable on *all* of the observed values. We implicitly indicate this relationship by suppressing observed variables and absorbing their effects into the remaining factors in the factor graph (figure 2.1c).

The semantics of the factor nodes are easily understood when we use the graphical model to score an assignment to the tag variables. Each factor node will contribute a multiplicative factor (its potential) to the score of the overall assignment. The potential contributed by each factor depends only on the values of its neighboring variables, and this functional dependence may be represented by a table whose dimensionality is the number of those neighbors. If the three tags in our running example are v a n, the contribution of the binary factor between the first two tags is the entry in row v and column a (figure 2.2). If we multiply the highlighted potentials, we see that the unnormalized probability assigned to the sequence v a n is $1 \cdot 3 \cdot 0.3 \cdot 0.1 \cdot 0.2 = 0.018$.

The factor graph representation provides a useful framework for what follows, by separating the objects we wish to evaluate (variable assignments, §2.2.2) from how we will evaluate them (factors, §§2.2.4–2.2.5). Factor graphs will also clarify the presentation of our inference procedures with loopy belief propagation (§2.3).

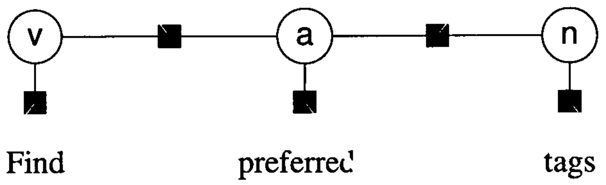
2.2.2 Observed and hidden variables for parsing

To apply this machinery, we must formulate dependency parsing as a search for an optimal **assignment** to the variables of a factor graph. We encode a dependency tree using the following variables:

Sentence. For an n-word input sentence, let $W = W_0 W_1 \cdots W_n$, where W_0 is always the special symbol ROOT. This representation allows confusion networks, where we know the number of words but not their identity, but not general lattices. In all

	a	n	v
a	1	3	0
n	0	1	2
v	**1**	2	0

	a	**n**	v
a	1	**3**	0
n	0	1	2
v	1	2	0



a	0
n	0.02
v	**0.3**

a	**0.1**
n	0
v	0.3

a	0
n	**0.2**
v	0.2

Figure 2.2: Potentials score variable assignments. Multiplying the highlighted entries in the potential tables, which correspond to these particular variable values, yields 0.018 as the unnormalized probability of the complete assignment.

the experiments in this thesis, the words are observed, and these variables will be absorbed into factors.

Tags. If desired, the variables $T = T_1 T_2 \cdots T_n$ may specify tags on the n words, drawn from some tagset $\mathcal{T}$ (e.g., parts of speech). These variables are needed iff the tags are to be inferred jointly with the parse—a case we will analyze, since BP provides considerable asymptotic speedups. In our experiments here, however, as in most current work on data-driven dependency parsing, the input already has fixed parts of speech; in other words, these variables are observed and will be absorbed into factors.

Links. The $O(n^2)$ boolean variables $\{L_{ij} : 0 \leq i \leq n, 1 \leq j \leq n, i \neq j\}$ correspond to the possible links in the dependency parse (figure 2.3). $L_{ij} = \mathsf{true}$ is interpreted as meaning that there exists a dependency link from parent $i \rightarrow$ child j. We can omit variables L_{ii} corresponding to self loops.[4]

Link roles, etc. It would be straightforward to add other variables, such as a binary variable L_{irj} that is true iff there is a link $i \xrightarrow{r} j$ labeled with role r (e.g., AGENT, PATIENT, TEMPORAL ADJUNCT).

[4]We could have chosen a different representation with $O(n)$ integer variables $\{P_j : 1 \leq j \leq n\}$, writing $P_j = i$ for $O(n)$ values instead of $L_{ij} = \mathsf{true}$. This representation can achieve the same asymptotic runtime for BP by using sparse messages, but some constraints and algorithms would be harder to explain. Briefly, a naive implementation of constraints on pairs of connected dependency links, such as the GRAND factors in §2.2.5 below, would require $O(n^4)$ potential table entries, rather than the $O(n^3)$ potentials we need for Boolean variables. The extra potentials are expended on pairs of dependency links that cannot be in a grandparent relationship: $1 \rightarrow 3, 3 \rightarrow 6$ is a legal grandparent–parent–child chain, but $1 \rightarrow 3, 4 \rightarrow 5$ is not because their middle terms differ. The integer-valued representation can still achieve $O(n^3)$ runtime if we only consider potentials $\neq 1$ (cf. familiar sparse matrix operations that avoid storing zero-valued entries).

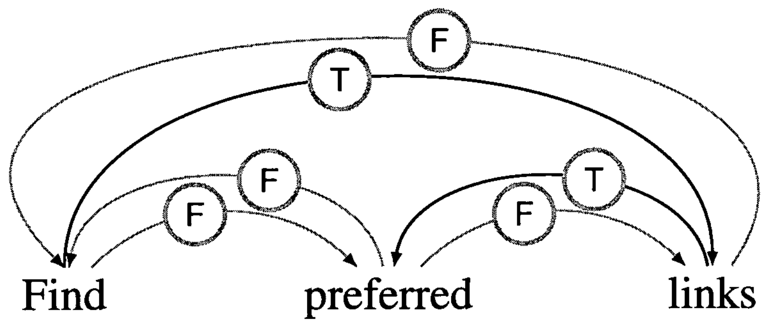


Figure 2.3: $O(n^2)$ boolean variables indicate which dependency edges from the complete graph (excluding self loops) are present. For clarity, we omit dependency links from the ROOT node (and their variables) and variables for the observed words.

2.2.3 Markov random fields

We wish to define a probability distribution over all configurations, i.e., all joint assignments $\mathcal{A}$ to these variables. Each assignment $\mathcal{A}$ represents a **parse**—though often an illegal parse in which the links with $L_{ij} = \textsf{true}$ do not form a valid tree. Without any constraints to prohibit cycles, for instance, $L_{2,5}$ and $L_{5,2}$ might both be $\textsf{true}$.

Our distribution is simply an undirected graphical model, or Markov random field (MRF):

$$p(\mathcal{A}) \stackrel{\text{def}}{=} \frac{1}{Z}\prod_m F_m(\mathcal{A}) \tag{2.5}$$

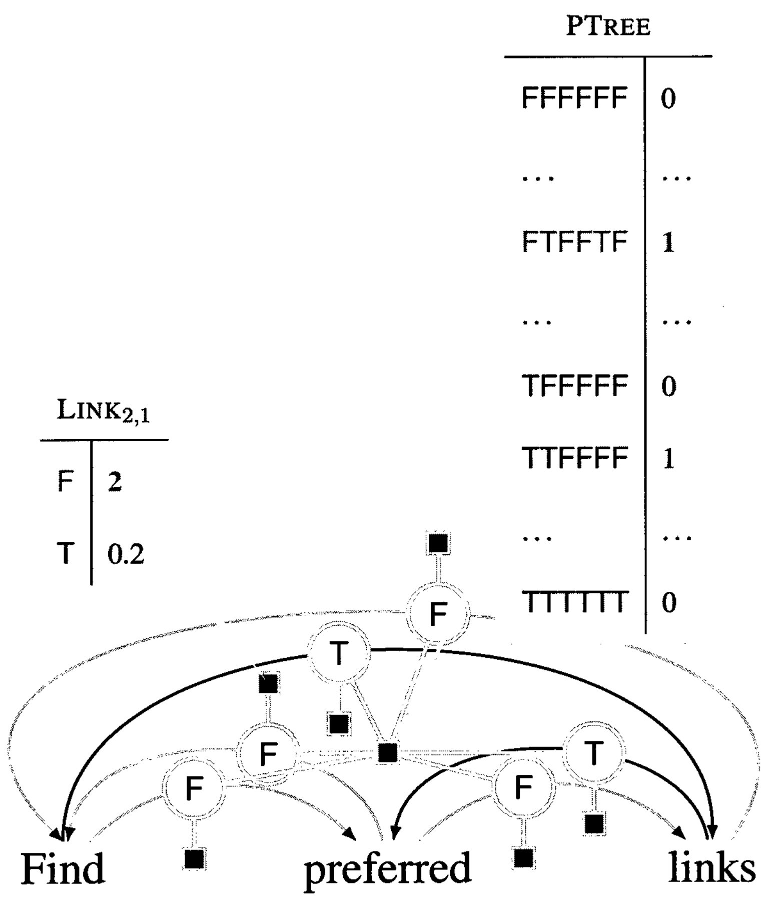
specified by the collection of **factors** $F_m : \mathcal{A} \mapsto \mathbb{R}^{\geq 0}$. A **hard** factor returns values in $\{0, 1\}$, thus serving as a hard constraint that rules out some parses by forcing their probability to 0. Other, **soft** factors merely raise or lower the score of some parses.

Each factor is a function that consults only a *subset* of $\mathcal{A}$. We say that the factor has **degree** d if it depends on the values of d variables in $\mathcal{A}$, and that it is **unary**, **binary**, **ternary**, or **global** if d is respectively 1, 2, 3, or unbounded (grows with n).[5]

Our overall model is properly called a *dynamic* MRF, since we must construct different-size MRFs for input sentences of different lengths (Sutton et al., 2004). Parameters are shared both across and within these MRFs, so that only finitely many parameters are needed.

Our model is a special case in another respect: it is a *conditional* random field, whose factor functions $F_m(\mathcal{A})$ may depend freely on any of the observed variables—the input sentence W. It also depends on a known (learned) parameter vector $\mathbf{w}$. For notational simplicity, we suppress these extra arguments when writing and drawing factor functions, and when computing their degree. In this treatment, these observed variables are not specified by $\mathcal{A}$, but instead are absorbed into the very definition of F_m (represented graphically in figure 2.1c).

§§2.2.4–2.2.5 lay out all of our parsing-specific factors; *their details are not too important*. In defining a factor F_m, we often state the circumstances under which it **fires**. These are the only circumstances that allow $F_m(\mathcal{A}) \neq 1$. When F_m does not fire, $F_m(\mathcal{A}) = 1$ and does not affect the product in (2.5).



LINK2,1	
F	2
T	0.2

PTREE	
FFFFFF	0
...	...
FTFFTF	1
...	...
TFFFFF	0
TTFFFF	1
...	...
TTTTTT	0

Figure 2.4: The PTREE constraint connects to all Link variables. It contributes a factor of if the Link variables represent a valid, projective tree, and a factor of 0 if they do not, thus ruling out that assignment. A potential table that explicitly, as here, represents all assignments to all Link variables would have 2^{n^2} entries. In §2.4 below, we will see how to apply this constraint in cubic-time.

2.2.4 List of hard constraints

A **hard** factor F_m fires only on parses $\mathcal{A}$ that *violate* some specified condition. It has value 0 on those parses, acting as a hard constraint to rule them out.

For parsing, we will consider these:

PTREE. A hard global constraint on all the L_{ij} variables at once. It requires that exactly n of these variables be true, and that the corresponding links form a projective, directed tree that is rooted at position 0 (§2.1.2). PTREE($\mathcal{A}$) has value 1 or 0 according to whether $\mathcal{A}$ satisfies these conditions (figure 2.4).

EXACTLY1. A *family* of $O(n)$ hard global constraints, indexed by $1 \leq j \leq n$. EXACTLY1$_j$ requires that j have exactly one parent, i.e., exactly one of the L_{ij} variables must be true. Note that EXACTLY1 is implied by PTREE.

ATMOST1. A weaker version. ATMOST1$_j$ requires j to have one or zero parents.

NAND. A *family* of hard binary constraints. NAND$(L_{ij}, L_{k\ell})$ requires that L_{ij} and $L_{k\ell}$ may not both be true. We will be interested in certain subfamilies.

NOT2. Shorthand for the family of $O(n^3)$ binary constraints $\{\text{NAND}(L_{ij}, L_{kj})\}$. These are collectively equivalent to ATMOST1, but expressed via a larger number of simpler (lower-degree) constraints, which can make the BP approximation less effective (footnote 12).

[5] In practice, we have used bounded factors up to degree 4, e.g. for the bilingual quasi-synchronous grammar constraints in chapter 4.

NO2CYCLE. Shorthand for the family of $O(n^2)$ binary constraints $\{\text{NAND}(L_{ij}, L_{ji})\}$.

As noted above, not all languages and linguistic formalisms use the projectivity constraint. In chapter 3, we will introduce techniques for nonprojective trees, including a TREE factor, which link PTREE is a global factor that ensures that the link variables form a directed tree, but not necessarily a projective one.

2.2.5 List of soft constraints

A **soft** factor F_m acts as a soft constraint that prefers some parses to others. In our experiments, it is always a log-linear function returning positive values:

$$F_m(\mathcal{A}) \stackrel{\text{def}}{=} \exp \sum_{h \in \text{features}(F_m)} \mathrm{w}_h f_h(\mathcal{A}, W, m) \tag{2.6}$$

where w is a learned, finite collection of weights and f is a corresponding collection of feature functions, some of which are used by F_m. (Note that f_h is permitted to consult the observed input W. It also sees which factor F_m it is scoring, to support reuse of a single feature function f_h and its weight w_h by unboundedly many factors in a model.)

The soft factors start with the simplest scores on individual dependency links:

LINK. A family of unary soft factors that judge the links in a parse $\mathcal{A}$ individually. LINK$_{ij}$ fires iff $L_{ij} = \textsf{true}$, and then its value depends on (i, j), W, and w. The features comprised by LINK are described in §2.8.1.

A first-order (or "edge-factored") parsing model (McDonald et al., 2005a) contains *only* LINK factors, along with a global TREE or PTREE factor. Though there are $O(n^2)$ link factors (one per L_{ij}), only n of them fire on any particular parse, since the global factor ensures that exactly n are true (cf. McAllester et al., 2004).

We now add our new ingredient that makes this a "second-order" model: soft binary constraints on the links, most importantly PAIR constraints that penalize or reward two links being simultaneously present.

These constraints have weight one unless the variables they constrain are both true; otherwise, they have weight zero (for a "hard" NAND) or some other nonnegative value (for a "soft" constraint). If we attach a weight greater than one to a candidate pair of siblings, say $\text{PAIR}(L_{4,6}, L_{6,8}) > 1$, the model will prefer that both of these links be present at the same time, for instance if the features of this binary factor indicate that word 4 is a ditransitive verb and words 6 and 8 would make good objects for it. On the other hand, it might be that $\text{PAIR}(L_{6,2}, L_{6,4}) < 1$ if words 2 and 4 are nouns and the local context indicates that word 6 is a verb that can only take a single noun argument on its left. We can similarly constrain grandparent–parent–child triples, e.g., $\text{PAIR}(L_{2,5}, L_{5,6})$ (figure 2.5).

PAIR. A binary factor $\text{PAIR}(L_{ij}, L_{k\ell})$ fires with some value iff L_{ij} and $L_{k\ell}$ are both true (and both exist). Thus, it penalizes or rewards a pair of links for being simultaneously present. This is a soft version of NAND from §2.2.4.

GRAND. Shorthand for the family of $O(n^3)$ binary factors $\{\text{PAIR}(L_{ij}, L_{jk})\}$, which evaluate grandparent–parent–child configurations, $i \rightarrow j \rightarrow k$. For example, whether preposition j attaches to noun i might depend to some extent on its object k: in figure 1.1 the prepositional phrase headed by *to*, with the object *question*, should be a good child for the noun *answer*.

SIB. Shorthand for the family of $O(n^3)$ binary factors $\{\text{PAIR}(L_{ij}, L_{ik})\}$, which judge whether two children of the same parent are compatible. For example, a given verb may not like to have two noun children both to its left.[6] The children do not need to be adjacent.

CHILDSEQ. A family of $O(n)$ global factors. CHILDSEQ$_i$ scores i's *sequence* of children; hence it consults all variables of the form L_{ij}. If word 5 has children $2, 7, 9$ under $\mathcal{A}$, then CHILDSEQ$_i$ uses bigram models to score the left sequence #2# and right sequence #79# (where # is a boundary symbol).[7]

VALENCE. A family of $O(n)$ global factors. VALENCE$_{di}$ scores the number of i's children to its left (if $d = L$) or right ($d = R$). Counts could increase up to n, but for better smoothing, counts over a certain amount will usually be binned. We will then, for instance, score configurations where a head has zero, one, or two or more right children.

§2.8.2 describes the features used to compute the values of these factors. Additional factors might consider c-command, subjacency violations, etc.

The following are useful if tags are hidden:

[6] A similar binary factor could directly discourage giving the verb two SUBJECTs, if the model has variables for link roles.

[7] This follows the parametrization of a weighted split head-automaton grammar (Eisner and Satta, 1999). The score in this case is a product of subfactors of that score bigrams in the child sequence, including the boundary symbols.

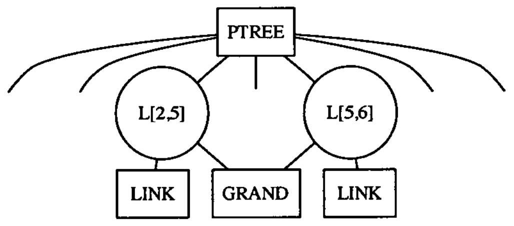


Figure 2.5: A fragment of a factor graph, illustrating a few of the unary, binary, and global factors that affect variables L_{25} and L_{56}. The GRAND factor induces a loop.

TAG$_i$ is a unary factor that evaluates whether T_i's value is consistent with W (especially W_i).

TAGLINK$_{ij}$ is a ternary version of the LINK$_{ij}$ factor whose value depends on L_{ij}, T_i and T_j (i.e., its feature functions consult the tag variables to decide whether a link is likely). One could similarly enrich the other features above to depend on tags and/or link roles; TAGLINK is just an illustrative example.

TRIGRAM is a global factor that evaluates the tag sequence T according to a trigram model. It is a product of subfactors, each of which scores a trigram of adjacent tags T_{i-2}, T_{i-1}, T_i, possibly also considering the word sequence W (as in CRFs).[8]

[8]For efficiency, this constraint cannot simply comprise a family of $O(n)$ ternary factors, each scoring three adjacent tags. That setup would induce a loopy graph. Recall that to construct a trigram POS HMM model, for instance, one needs to model transitions between tag *pairs*. In other words, we need a family of $O(n)$ binary constraints among $O(n)$ variables whose values $\in T \times T$, i.e. the "junction tree" of the naive collection of trigram factors.

2.3 A Sketch of Belief Propagation

MacKay (2003, chapters 16 and 26) provides an excellent introduction to belief propagation, a generalization of the forward-backward algorithm that is deeply studied in the graphical models literature (Yedidia et al., 2004, for example). We sketch the method in terms of our parsing task.

2.3.1 Where BP comes from

The basic BP idea is simple. The boolean variable like L_{34} maintains a *distribution* over values true and false—a "belief"—that is periodically recalculated based on the current *distributions* at other variables.[9] Multinomial variables maintain distributions over their domains.

Readers familiar with Gibbs sampling can regard this as a kind of deterministic approximation. In Gibbs sampling, L_{34}'s *value* is periodically resampled based on the current *values* of other variables. Loopy BP works not with random samples but their expectations. Hence it is approximate but tends to converge much faster than Gibbs sampling will mix.

[9]Or, more precisely—this is the tricky part—based on versions of those other distributions that do not factor in L_{34}'s reciprocal influence on *them*. This prevents (e.g.) L_{34} and T_3 from mutually reinforcing each other's existing beliefs.

2.3.2 What BP accomplishes

Given an input sentence W and a parameter vector $\mathbf{w}$, the collection of factors F_m defines a probability distribution (2.5). The parser should determine the values of the individual variables. In other words, we would like to marginalize equation (2.5) to obtain the distribution $p(L_{34})$ over $L_{34} = \textsf{true}$ vs. $\textsf{false}$, the distribution $p(T_4)$ over tags, etc.

If the factor graph is *acyclic*, then BP computes these marginal distributions exactly. Given an HMM or a linear-chain CRF, for example, BP reduces to the forward-backward algorithm.

BP's estimates of these distributions are called **beliefs about the variables**. BP also computes **beliefs about the factors**, which are useful in learning $\mathbf{w}$ (see §2.7). For example, if the model includes the factor TAGLINK$_{ij}$, which is connected to variables L_{ij}, T_i, T_j, then BP will estimate the marginal joint distribution $p(L_{ij}, T_i, T_j)$ over (boolean, tag, tag) triples.

When the factor graph has loops, BP's beliefs are usually not the true marginals of equation (2.5) (which are in general intractable to compute). Indeed, BP's beliefs may not be the true marginals of *any* distribution $p(\mathcal{A})$ over assignments, i.e., they may be globally inconsistent. All BP does is to incrementally adjust the beliefs till they are at least *locally* consistent: e.g., the beliefs at factors TAGLINK$_{ij}$ and TAGLINK$_{ik}$ must both imply[10] the same belief about variable T_i, their common neighbor.

[10]In the sense that marginalizing the belief $p(L_{ij}, T_i, T_j)$ at the factor yields the belief $p(T_i)$ at the variable.

2.3.3 The BP algorithm

BP treats inference as an iterated negotiation among many constraints, each represented by factors and each much simpler than the full model. This iterated negotiation among the factors is handled by **message passing** along the edges of the factor graph. A **message** to or from a variable is a (possibly unnormalized) probability distribution over the values of that variable.

In the factor graph (e.g., the fragment in figure 2.5), many factors may connect to—and hence influence—a given variable such as L_{34}. If X is a variable *or* a factor, $\mathcal{N}(X)$ denotes its set of neighbors.

The variable V sends a message to factor F, saying "My *other* neighboring factors G jointly suggest that I have posterior distribution $q_{V \to F}$ (assuming that they are sending me independent evidence)." Meanwhile, factor F sends messages to V, saying, "Based on my factor function and the messages received from my *other* neighboring variables U about *their* values (and assuming that those messages are independent), I suggest you have posterior distribution $r_{F \to V}$ over *your* values."

To be more precise, BP at each iteration k (until convergence) updates two kinds of messages:[11]

$$q^{(k+1)}_{V \to F}(v) = \kappa \prod_{G \in \mathcal{N}(V), G \neq F} r^{(k)}_{G \to V}(v) \tag{2.7}$$

[11]This is literally true. We are not updating all messages in a particular order, but in parallel: all v-to-f, then all f-to-v. This requires a bit of care for sequence constraints such as CHILDSEQ or TRIGRAM (§2.5.3), where the propagator runs forward and then backward along the sequence in a single update.

from variables to factors, and

$$r^{(k+1)}_{F\to V}(v) = \kappa \sum_{\mathcal{A} \text{ s.t. } \mathcal{A}[V]=v} F(\mathcal{A}) \prod_{U\in\mathcal{N}(F), U\neq V} q^{(k)}_{U\to F}(\mathcal{A}[U]) \tag{2.8}$$

from factors to variables. Each message is a probability distribution over values v of V, normalized by a scaling constant κ. Alternatively, messages may be left as unnormalized distributions, choosing $\kappa \neq 1$ only as needed to keep beliefs from over- or underflow. Messages are initialized to uniform distributions. Algorithm 2.1 presents the "synchronous" variant of BP, which we use here, where messages are sent from all the factors to all the variables and then vice versa.

Whenever we wish, we may compute the beliefs at V and F:

$$b^{(k+1)}_{V\to}(v) \overset{\text{def}}{=} \kappa \prod_{G\in\mathcal{N}(V)} r^{(k)}_{G\to V}(v) \tag{2.9}$$

$$b^{(k+1)}_{F\to}(\mathcal{A}) \overset{\text{def}}{=} \kappa\, F(\mathcal{A}) \prod_{U\in\mathcal{N}(F)} q^{(k)}_{U\to F}(\mathcal{A}[U]) \tag{2.10}$$

These beliefs do not truly characterize the expected behavior of Gibbs sampling (§2.3.1), since the products in (2.9)–(2.10) make conditional independence assumptions that are valid only if the factor graph is acyclic. Furthermore, on cyclic ("loopy") graphs, BP might only converge to a local optimum (Weiss and Freedman, 2001), or it might not converge at all. Still, as we see in our experiments, BP often leads to good, fast approximations.

When the factor graph is a tree, BP correctly computes the marginal probability distributions of all variables, and the expected values of all potential functions, with respect to

Algorithm 2.1 Belief propagation

```
 1: procedure BP($\mathcal{G}$)                                        ▷ Factor graph $\mathcal{G}$
 2:     $r, q \leftarrow$ uniform distribution
 3:     repeat
 4:         for $F \in \mathcal{G}$ s.t. $F$ is a factor do
 5:             if $F$ has a specialized propagator then
 6:                 PROPAGATE($F$)                                    ▷ Algorithm 2.2
 7:             else
 8:                 for $V \in \mathcal{N}(F)$ do
 9:                     $r_{F \to V}(v) \leftarrow \sum_{\mathcal{A} \text{ s.t. } \mathcal{A}[V]=v} F(\mathcal{A}) \prod_{U \in \mathcal{N}(F), U \neq V} q_{U \to F}(\mathcal{A}[U])$
10:                     Renormalize messages by $\kappa = \sum_v r_{F \to V}(v)$
11:                 end for
12:             end if
13:         end for
14:         for $V \in \mathcal{G}$ s.t. $V$ is a variable do
15:             for $F \in \mathcal{N}(V)$ do
16:                 $q_{V \to F}(v) \leftarrow \prod_{G \in \mathcal{N}(V), G \neq F} r_{G \to V}(v)$
17:                 Renormalize messages by $\kappa = \sum_v q_{V \to F}(v)$
18:             end for
19:         end for
20:     until messages converge or max. iterations
21: end procedure
```

the joint distribution $p(\mathbf{x})$ (MacKay, 2003). The forward-backward algorithm for HMMs is such an exact BP procedure.[12] Many useful features, however, may induce loops in the factor graph (figure 2.5).

Figure 2.6 presents the computation of factor-to-variable messages in the sum-product algorithm as matrix multiplication. (For consistency, we transpose the outgoing message vector.) The contributions of various assignments to the source variable (on the left) are multiplied by the appropriate potentials and summed up in the message keyed to assignments to the destination variable (on the right). In contrast, variable-to-factor messages are computed by component-wise vector multiplication. When a factor has more than two neighbors, its potential table will have correspondingly more dimensions and use higher-dimensional analogues to matrix multiplication.[13] Our implementation of message passing uses this matrix formulation to make the code more compact. In future, moreover, we should be able to exploit sparse matrix implementations to work with sparse messages, where only a few values (and potentials) are non-zero.

2.3.4 Related iterative methods in structured prediction

Most implementations of belief propagation conserve space by replacing messages or beliefs at time k if they change at time $k+1$. It is often helpful to analyze the system by

[12]The (weighted) CKY and inside-outside algorithms for PCFGs perform inference not on simple graphs but on hypergraphs. McAllester et al. (2004) show how the constraints in a context-free grammar may be represented by a case-factor diagram.

[13]If the arity of all of the neighboring variables is the same, we could use standard tensor notation, but that condition is not always satisfied.

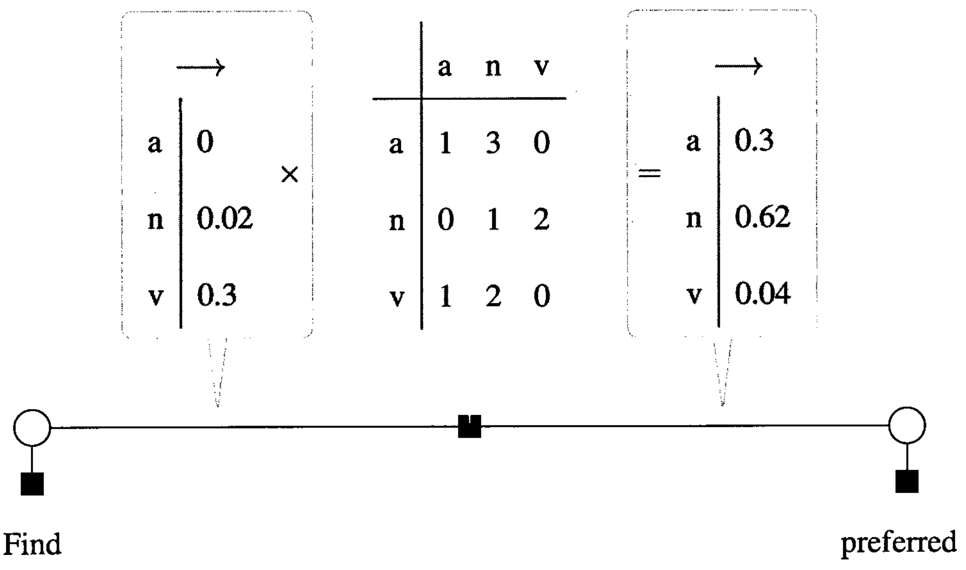


Figure 2.6: Computing BP messages in the example factor graph from figure 2.2. In the sum-product algorithm, factor-to-variable messages are computed by multiplying the message vector incoming from the variable on the left by the matrix of potentials to produce an outgoing message vector on the right. Variable-to-factor messages are computed by component-wise vector multiplication.

unrolling the computation. For instance, if we run BP for K steps, we can work with a directed acyclic computation graph with K echelons.

This unrolled view clarifies some connections to related methods in structured prediction and suggests some ways in which we might refine BP. When classifiers are **stacked**, for example, the output of one level of classification is input to the next level. Some stacked classification schemes use the marginal distributions of the previous level as input, but more often it is the one-best output that is passed on. These two alternatives are analogous to sum-product and max-product belief propagation. Two-level one-best stacking schemes have recently proved effective in dependency parsing (Nivre and McDonald, 2008b; Martins et al., 2008). The second-level parser can achieve improved performance by treating as observed larger substructures in the first-level output—in effect, by taking advantage of second-order features without the need for expensive second-order inference. In a similar way, binary factors like GRAND compute their messages based on the beliefs of two variables at the previous round of BP. In BP, however, a factor that propagations messages multiple times during several iterations of BP will use the same potentials each time; in stacking, the parameters of the first- and second-level models are not shared, which makes it possible to learn the first-level parameters independently. We can thus view stacked classification as belief propagation on an unrolled computation graph with a fixed depth where each factor appears exactly once.[14]

[14]Transformation based learning (Brill, 1995), which has been popular for some sequence labeling tasks, is a special case of stacking: the method learns a sequence of transformations that match a pattern in the output of previous transformations and replaces a label at matching locations with another. While stacking in general can use any method for each classifier, TBL specifically chooses the transformation that most reduces the error in the output on training data.

Reranking is another widely used method in structured prediction. Like stacking, it is generally two-tiered, where the output of one level is input to a second. In this case, however, the first level outputs a set of candidates (an n-best list) and their scores under the first-level model; the second-level classifier then chooses its output from among those candidates. (In contrast, stacking generally assumes the same hypothesis space for the two levels of classifiers.) Reranking allows the use of features that can be easily evaluated for a single hypothesis but not for the full hypothesis space (Collins and Duffy, 2002; Charniak and Johnson, 2005). When the hypotheses generated by the first level are encoded in a packed representation—such as a lattice for finite-state methods, or a packed forest or hypergraph for context-free grammars—this process is called **rescoring** (Roark, 2002). As with stacking, inference in the first-level model proceeds without knowledge of (e.g., messages influenced by) later second-level inference. The method of pipeline iteration is an extension of reranking that does allow later stages to exert such influence (Hollingshead and Roark, 2007).[15]

2.4 Achieving Low Asymptotic Runtime

One iteration of standard BP simply updates all the messages as in equations (2.7)–(2.8): one message per edge of the factor graph.

[15]"Coarse-to-fine" describes a broad class of methods that expands the representation of the hypothesis space in areas of high probability (Charniak et al., 2006). Unlabeled dependencies, for instance, could be a coarse representation of which labeled dependencies are the refinement. This approach, therefore, compresses the hypothesis space by changing representation rather than by selecting an n-best list of hypotheses.

Therefore, adding new factors to the model increases the runtime per iteration *additively*, by increasing the number of messages to update. We believe this is a compelling advantage over dynamic programming—in which new factors usually increase the runtime and space *multiplicatively* by exploding the number of distinct items in the chart.

For example, with unknown tags T, a model with PTREE+TAGLINK will take only $O(n^3+n^2g^2)$ time for BP, compared to $O(n^3g^2)$ time for dynamic programming (Eisner and Satta, 1999). Adding TRIGRAM, which is string-local rather than tree-local, will increase this only to $O(n^3 + n^2g^2 + ng^3)$, compared to $O(n^3g^6)$ for dynamic programming.

If there are g tags, then propagating through these $O(n^2)$ ternary soft constraints takes time $O(n^2g^2)$ per pass. Meanwhile, propagating through the global PTREE constraint takes an additional $O(n^3)$ time per pass. It is therefore tolerable to have a moderately large g (say, $g = O(\sqrt{n})$) before the former term begins to dominate.

By contrast, an exact method for this model, in the projective case, would take $O(n^3g^2)$ time (Eisner and Satta, 1999), where one pays a serious penalty for large g. The runtime advantage to using the BP approximation is that it splits the second-order parsing problem into a message-passing algorithm that alternately tries to find (1) links that are compatible with one another and with the word tags, and (2) word tags that are compatible with the links.

Even more dramatically, adding the SIB family of $O(n^3)$ PAIR$_{ij,ik}$ factors will add only $O(n^3)$ to the runtime of BP (Table 2.1). By contrast, the runtime of dynamic programming becomes exponential, because each item must record its headword's full *set* of current children.

As the analogy to forward-backward, or inside-outside, suggests, there are optimal orders (possibly not unique) for updating messages in tree-structured graphs. In loopy graphs, we might still alternate between updating all variable-factor q messages and all factor-variable r messages; however, an alternative algorithm, known as asynchronous belief propagation, prioritizes which messages get updated based on their weights, and can lead to faster convergence (Elidan et al., 2006; Sutton and McCallum, 2007). These strategies are similar to best-first parsing algorithms (Caraballo and Charniak, 1998).

Table 2.1 shows our asymptotic runtimes for all factors in §§2.2.4–2.2.5. Remember that if several of these factors are included, the total runtime is *additive*. We may ignore the cost of propagators at the variables. Each outgoing message from a variable can be computed in time proportional to its cardinality, which may be amortized against the cost of generating the corresponding incoming message. The message is at most equal in size to the cardinality of the variable's value set. If the incoming message is sparse, however, we do not need to compute outgoing components that correspond to the zeroes in the incoming vector. We now analyze the cost of propagating messages from particular factors.

2.4.1 Propagators for local constraints

How long does updating each factor message take? The runtime of summing over all assignments $\sum_{\mathcal{A}}$ in equation (2.8) may appear prohibitive. Crucially, however, $F(\mathcal{A})$ only depends on the values in $\mathcal{A}$ of F's neighboring variables $\mathcal{N}(F)$. So this sum is proportional

factor family	degree (each)	runtime (each)	count	runtime (total)
PTREE	$O(n^2)$	$O(n^3)$	1	$O(n^3)$
EXACTLY1	$O(n)$	$O(n)$	n	$O(n^2)$
ATMOST1	$O(n)$	$O(n)$	n	$O(n^2)$
NOT2	2	$O(1)$	$O(n^3)$	$O(n^3)$
NO2CYCLE	2	$O(1)$	$O(n^2)$	$O(n^2)$
LINK	1	$O(1)$	$O(n^2)$	$O(n^2)$
GRAND	2	$O(1)$	$O(n^3)$	$O(n^3)$
SIB	2	$O(1)$	$O(n^3)$	$O(n^3)$
CHILDSEQ	$O(n)$	$O(n^2)$	$O(n)$	$O(n^3)$
VALENCE	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
TAG	1	$O(g)$	$O(n)$	$O(ng)$
TAGLINK	3	$O(g^2)$	$O(n^2)$	$O(n^2g^2)$
TRIGRAM	$O(n)$	$O(ng^3)$	1	$O(ng^3)$

Table 2.1: Asymptotic runtimes of the propagators for various factors (where n is the sentence length and g is the size of the tag set $\mathcal{T}$). Global factors are those with non-constant degree. An iteration of standard BP propagates through each factor once. Running a factor's propagator will update all of its outgoing messages, based on its current incoming messages.

to a sum over *restricted* assignments to just those variables.[16]

For example, computing a message from TAGLINK$_{ij} \to T_i$ only requires iterating over all (boolean, tag, tag) triples.[17] The runtime to update that message is therefore $O(2 \cdot |\mathcal{T}| \cdot |\mathcal{T}|)$.

2.4.2 Propagators for global constraints

The above may be tolerable for a ternary factor. But how about global factors? EXACTLY1$_j$ has n neighboring boolean variables: surely we cannot iterate over all 2^n assignments to these! PTREE is even worse, with 2^{n^2} assignments to consider. In §2.5, we will present a new technique for deriving specialized algorithms to handle these summations more efficiently (algorithm 2.2).

A historical note is in order. Traditional constraint satisfaction corresponds to the special case of (2.5) where *all* factors F_m are hard constraints (with values in $\{0,1\}$). In that case, loopy BP reduces to an algorithm for generalized arc consistency (Mackworth, 1977; Bessière and Régin, 1997; Dechter, 2003), and updating a factor's outgoing messages is known as **constraint propagation**. Régin (1994) introduced an efficient propagator for a global constraint, ALLDIFFERENT, by adapting combinatorial bipartite matching algorithms.

Maruyama (1990) proposed using hard constraints to rule out certain parses and prun-

[16]The constant of proportionality may be folded into κ; it is the number of assignments to the *other* variables.

[17]Separately for *each* value v of T_i, get v's probability by summing over assignments to (L_{ij}, T_i, T_j) s.t. $T_i = v$.

ing the set of valid trees with constraint propagation. As later developed (Harper et al., 1995; Wang, 2003; McCrae et al., 2008; Foth et al., 2006), these hard constraints could be combined with generative models, for language modeling in speech recognition. Standard methods for learning linear or log-linear models do not work for hard constraints; instead, hard constraint sets are hand-designed, or greedy relaxation procedures drop constraints until the training set can be parsed. In this work, we avoid the problem of learning hard constraints by using them only for structural properties of the output, such as trees.

In spirit of global constraints for arc consistency, we will demonstrate efficient propagators for our global constraints, e.g. by adapting combinatorial algorithms for *weighted parsing*. We are unaware of any previous work on global factors in sum-product BP, although for *max-product* BP,[18] Duchi et al. (2006) independently showed that a global 1-to-1 alignment constraint—a kind of weighted ALLDIFFERENT—permits an efficient propagator based on weighted bipartite matching.

2.5 Combinatorial Constraint Propagators

Propagating the local factors is straightforward (§2.4.1): we enumerate all of the assignments of values to variables participating in a message. We now explain how to handle the global factors: EXACTLY1, PTREE, TRIGRAM, CHILDSEQ, and VALENCE.

When factors connect to more than $O(1)$ variables, enumerating variable assignments

[18]Max-product replaces the sums in equations (2.7)–(2.8) with maximizations. For the special case of linear chains, this replaces the forward-backward algorithm with its Viterbi approximation.

becomes exponential. The EXACTLY1 factor in §2.2 above could be used to ensure that a word had only one parent in the dependency tree. Since this factor is connected to n variables, the naive r calculation would sum over 2^n possible assignments. Although we will use the stronger PTREE factor, which also forbids cycles in the graph and crossing edges, it is useful to see how the messages work for this simpler combinatorial constraint.

Say that we have three variables A, B, and C connected to an EXACTLY1 factor. Let $(q_A(1) = a, q_A(0) = 1 - a)$ be the message sent by A to the EXACTLY1 factor; this is a distribution that indicates how much the *other* factors would like A to be 1 vs. 0. We can represent this distribution with the single probability a; similarly, B and C send q messages b and c to the EXACTLY1 factor.

The EXACTLY1 factor must then send a message back to A, indicating how much the EXACTLY1 factor—together with the incoming messages from B and C—would like A to be 1 or 0. The total probability of all configurations that satisfy the EXACTLY1 constraint is

$$Z = a(1-b)(1-c) + b(1-a)(1-c) + c(1-a)(1-b)$$

We may write this as

$$\begin{aligned} Z &= Z_A(1) + Z_A(0) \\ &= a \cdot r_A(1) + (1-a) \cdot r_A(0) \end{aligned}$$

where $Z_A(i)$ is the total probability of configurations in which $A = i$. We can simplify these computations by normalizing all of the incoming qs by their zero-belief, i.e. making the messages an odds ratio. We then have a new normalizing constant

$$Z = \frac{a}{1-a} + \frac{b}{1-b} + \frac{c}{1-c},$$

and the message from EXACTLY1 to the variable A is simply $(r_A(1) = 1, r_A(0) = Z - \frac{a}{1-a})$. For n variables, we now no longer need to iterate through 2^n assignments; instead, we compute Z in $O(n)$ steps and then subtract the odds ratio.

2.5.1 Exploiting marginal beliefs

This example motivates a general procedure that *works backwards from the marginal beliefs.* Let F be a factor and V be one of its neighboring variables. At any time, F has a marginal belief about V (see footnote 10),

$$b_{F\rightarrow}^{(k+1)}(V = v) = \sum_{\mathcal{A} \text{ s.t. } \mathcal{A}[V]=v} b_{F\rightarrow}^{(k+1)}(\mathcal{A}) \qquad (2.11)$$

a sum over (2.10)'s products of incoming messages. By the definition of $r_{F\rightarrow V}$ in (2.8), and distributivity, we can also express the marginal belief (2.11) as a pointwise product of outgoing and incoming messages, each of which is a (possibly unnormalized) distribution over values v of V:[19]

$$b_{F\rightarrow}^{(k+1)}(V = v) = r_{F\rightarrow V}^{(k+1)}(v) \cdot q_{V\rightarrow F}^{(k)}(v) \qquad (2.12)$$

If we can quickly sum up the marginal belief (2.11), then (2.12) says we can divide out *each* particular incoming message $q_{V\rightarrow F}^{(k)}$ in turn to obtain its corresponding outgoing message $r_{F\rightarrow V}^{(k+1)}$.

[19]This is the analogue of the familiar product of forward and backward messages that is used to extract posterior marginals from an HMM.

Algorithm 2.2 Calculate outgoing messages from factor F

```
1: procedure PROPAGATE($F$)
2:     Normalize incoming messages $q_V(v)$
3:     $b() \leftarrow \sum_{\mathcal{A}} b(\mathcal{A})$                          ▷ Partition function
4:     for $V \leftarrow \mathcal{N}(F)$ do
5:         $r_V(v) \leftarrow b(V = v) \;/\; q_V(v)$                  ▷ Marginal beliefs
6:     end for
7: end procedure
```

Note that the marginal belief and both messages are unnormalized distributions over values v of V. F and k are clear from context below, so we simplify the notation so that (2.11)–(2.12) become

$$b(V = v) \;=\; \sum_{\mathcal{A} \text{ s.t. } \mathcal{A}[V]=v} b(\mathcal{A}) \;=\; r_V(v) \cdot q_V(v)$$

This equality leads to the schematic algorithm 2.2 for calculating a combinatorial factor's outgoing messages. Clearly, a factor's messages take at least $O(|\mathcal{N}(F)|)$ time to compute. But how are the partition function $b()$ and the marginals $b(V = v)$ computed? Different factors will employ different combinatorial algorithms to obtain these quantities.

In deriving algorithms for the marginals, it is helpful to observe that factor beliefs are (unnormalized) joint distributions over the values of their neighboring variables. (If the distribution is normalized, the partition function $b() = 1$.) The belief $b(V = v)$ is therefore also an expectation $\mathbf{E}_{b(\mathcal{A})}[V = v]$.[20]

[20]If the algorithm for computing $b()$ uses the appropriate operations, we can use an expectation semiring (Eisner, 2002) to get $b(V = v)$. Alternately, if we can derive the gradient of $b()$ with respect to the incoming messages $q_V(v)$, we can exploit the equivalence of feature expectations and gradients in log-linear models. We use this approach when we derive the message computations for PTREE in §2.5.4.

2.5.2 Propagating EXACTLY1 and ATMOST1

We can now return to **EXACTLY1**$_j$. Observe that it is a *sparse* hard constraint: even though there are 2^n assignments to its n neighboring variables $\{L_{ij}\}$, the factor function returns 1 on only n *satisfying* assignments and 0 on the rest. In fact, for a given i, $b(L_{ij} = \text{true})$ in (2.11) is defined by (2.10) to have exactly one non-zero summand, in which $\mathcal{A}$ puts $L_{ij} = \text{true}$ and all other $L_{i'j} = \text{false}$. Fleshing out algorithm 2.2, we compute the outgoing messages (2.11) to all parent variables i in $O(n)$ total time (algorithm 2.3).[21]

The **ATMOST1** constraint is very similar: the partition function $b()$ needs an additional term π to account for the event that all of the variables are false.

2.5.3 Propagating sequence constraints

TRIGRAM must sum over assignments to the tag sequence T. The belief (2.10) in a given assignment is a product of trigram scores (which play the role of transition weights) and incoming messages q_{T_j} (playing the role of emission weights). The marginal belief (2.11) needed above, $b(T_i = t)$, is found by summing over assignments where $T_i = t$. All marginal beliefs are computed together in $O(ng^3)$ total time by the forward-backward algorithm.[22]

[21]Taking $\pi = 1$, as in our implementation, gives the same results, up to a constant. We present the normalized version for clarity. When modeling Boolean variables, the odds ratio $\bar{q}_{L_{ij}} \in \mathbb{R}$ can be used to represent the incoming message $q_{L_{ij}}$ everywhere, although we do not use this optimization for general two-value messages.

[22]Which is itself an exact BP algorithm, but on a different graph—a junction tree formed from the graph of TRIGRAM subfactors. Each variable in the junction tree is a *bigram*. If we had simply replaced the global TRIGRAM factor with its subfactors in the full factor graph, we would have had to resort to Generalized BP Yedidia et al. (2004) to obtain the same exact results.

Algorithm 2.3 Calculate EXACTLY1's outgoing messages

```
1: procedure PropagateExactly1($j, n$)          ▷ Child index $j$, sentence length $n$
2:     $\pi \leftarrow 1$                                   ▷ Normalizing constant
3:     for $i \leftarrow 0..n, i \neq j$ do
4:         $\bar{q}_{L_{ij}} \leftarrow q_{L_{ij}}(\text{true}) \,/\, q_{L_{ij}}(\text{false})$      ▷ Cf. algorithm 2.2, line 2
5:         $\pi \leftarrow \pi \cdot q_{L_{ij}}(\text{false})$
6:     end for
7:     $b() \leftarrow 0$
8:     for $i \leftarrow 0..n, i \neq j$ do
9:         $b(L_{ij} = \text{true}) \leftarrow \pi \cdot \bar{q}_{L_{ij}}$
10:        $b() \leftarrow b() + b(L_{ij} = \text{true})$      ▷ Cf. algorithm 2.2, line 3
11:    end for
12:    for $i \leftarrow 0..n, i \neq j$ do
13:        $b(L_{ij} = \text{false}) \leftarrow b() - b(L_{ij} = \text{true})$
14:        $r_{L_{ij}}(\text{true}) \leftarrow b(L_{ij} = \text{true}) \,/\, q_{L_{ij}}(\text{true})$      ▷ Cf. algorithm 2.2, line 5
15:        $r_{L_{ij}}(\text{false}) \leftarrow b(L_{ij} = \text{false}) \,/\, q_{L_{ij}}(\text{false})$
16:    end for
17: end procedure
```

CHILDSEQ$_i$, like TRIGRAM, is propagated by a forward-backward algorithm. In this case, the algorithm is easiest to describe by replacing CHILDSEQ$_i$ in the factor graph by a collection of local subfactors, which pass messages in the ordinary way. Roughly speaking,[23] at each $j \in [1, n]$, we introduce a new variable C_{ij}—a hidden state whose value is the position of i's previous child, if any (so $0 \leq C_{ij} < j$). So the ternary subfactor on $(C_{ij}, L_{ij}, C_{i,j+1})$ has value 1 if $L_{ij} = \textsf{false}$ and $C_{i,j+1} = C_{i,j}$; a sibling-bigram score (PAIR$_{iC_{ij},iC_{i,j+1}}$) if $L_{ij} = \textsf{true}$ and $C_{i,j+1} = j$; and 0 otherwise. The sparsity of this factor, which is 0 almost everywhere, is what gives CHILDSEQ$_i$ a total runtime of $O(n^2)$ rather than $O(n^3)$. It is equivalent to forward-backward on an HMM with n observations (the L_{ij}) and n states per observation (the C_j), with a *deterministic* (thus sparse) transition function.

We treat **VALENCE**$_{di}$ similarly as a collection of local factors in a Markov chain. The values of intermediate variables B_{dij} range over the count of the number of children of i on side d that we have observed between i and j.

How do these sequence factors differ from collections of local factors? As noted above, the forward-backward algorithm is a special case of belief propagation *but with a particular ordering of messages.* To get the optimal linear runtime, we need to interleave factor and variables messages forward along the chain and then back. We can thus treat CHILDSEQ$_i$, e.g., as a global factor and compute all its correct outgoing messages on a *single* BP iteration. Handling the subfactors in parallel, (2.7)–(2.8), would need $O(n)$ iterations.

[23]Ignoring the treatment of boundary symbols "#" (see §2.2.5).

2.5.4 Constraint propagation by parsing

PTREE must sum over assignments to the $O(n^2)$ neighboring variables $\{L_{ij}\}$. Although we avoided summing over *all* assignments to variables adjacent to the EXACTLY1 factor, we still simply listed all n *satisfying* assignments. This approach is intractable for PTREE because there are now exponentially many non-zero summands in (2.11), viz. the set **P** of assignments in which the values of $\{L_{ij}\}$ correspond to a valid projective tree.[24] As with the sequence constraints above, we will use dynamic programming, but instead of the forward-backward algorithm, which is a special case of BP, we will use the inside-outside algorithm on hypergraphs.

The inside-outside algorithm was developed for probabilistic context-free grammars (Baker, 1979) but can be extended to grammars with positive, real-valued weights (Smith and Johnson, 2007). The quantities computed by the inside-outside algorithm are: the **inside weight** $\beta_A(i, j)$, which is the sum of the weight of all trees with root non-terminal A that span words i to j; and the **outside weight** $\alpha_A(i, j)$, which is the sum of the weights of all structures that complete a tree rooted at A and spanning i to j. For start symbol S, the special inside weight $\beta_S(0, n)$ is the weight of all trees rooted at S that span the entire input. (This is the partition function Z for globally normalized models of trees.) The posterior probability that constituent A spans i to j is thus $\beta_A(i, j) \cdot \alpha_A(i, j)/\beta_S(0, n)$.[25]

[24]The number of projective trees is related to the Catalan number, q.v. §3.1 below.

[25]The normalizing constant may be 0 if no valid trees can be constructed. We may see this condition if two incompatible constituents each have probability numerically equal to one. We have not observed this sort of numerical problem with projective dynamic programming, though it does occur sometimes with nonprojective parsing.

This posterior may also be written in terms of the gradient of the sum over all trees: thus, $\beta_A(i,j) \cdot \partial\beta_S(0,n)/\partial\beta_A(i,j)$ (Li and Eisner, 2009).

Inference for projective edge-factored dependency trees is performed by specialized versions of the dynamic programs for general CFGs (Eisner, 1996). In appendix B, we give pseudocode for, e.g., finding the best tree (algorithm B.1), computing the inside weights (algorithm B.2), and computing the gradient (algorithm B.3) for edge-factored models. As in §2.1.3 above, $u(i,j)$ is the exponentiated weight of a dependency link from x_i to x_j; we define $g(i,j) = \partial \log Z/\partial u(i,j)$. The posterior probability of $x_i \rightarrow x_j$ is therefore $u(i,j) \cdot g(i,j)$. Importantly, this is precisely the marginal belief that the link is true $b(L_{ij} = \textsf{true})$. The dynamic programming algorithms for computing the $u(i,j)$ and $g(i,j)$ run in cubic time.

What values serve as the $u(i,j)$ input rule probabilities in inside and outside passes? As in §2.5.1, we use the odds ratio of the incoming true-over-false message values for that edge:

$$u(i,j) = \bar{q}_{L_{ij}} = q_{L_{ij}}(\textsf{true})/q_{L_{ij}}(\textsf{false}) \tag{2.13}$$

We can thus divide the marginal belief about an edge by the corresponding $u(i,j)$ to get the true component of the outgoing message. Since the posterior probability is already normalized, the normalizing constant is 1. The messages are thus:

$$r_{L_{ij}}(\textsf{true}) = g(i,j) \tag{2.14}$$

$$r_{L_{ij}}(\textsf{false}) = 1 - u(i,j) \cdot g(i,j) \tag{2.15}$$

Algorithm 2.4 Calculate PTREE's outgoing messages

```
1: procedure PROPAGATEPTREE($n$)                          ▷ Sentence length $n$
2:     for $i \leftarrow 0..n, j \leftarrow 1..n, i \neq j$ do
3:         $u(i,j) \leftarrow q_{L_{ij}}(\text{true}) \,/\, q_{L_{ij}}(\text{false})$
4:     end for
5:     $C \leftarrow$ EFINSIDE($u, n$)                     ▷ Algorithm B.2, $O(n^3)$
6:     $g \leftarrow$ EFGRADIENT($u, C, n$)                ▷ Algorithm B.3, $O(n^3)$
7:     for $i \leftarrow 0..n, j \leftarrow 1..n, i \neq j$ do
8:         $r_{L_{ij}}(\text{true}) \leftarrow g(i,j)$
9:         $r_{L_{ij}}(\text{false}) \leftarrow 1 - u(i,j) \cdot g(i,j)$
10:    end for
11: end procedure
```

up to the constant $\pi \overset{\text{def}}{=} \prod_{ij} q_{L_{ij}}(\text{false})$, as above.

Algorithm 2.4 shows the full procedure, which calls two $O(n^3)$ subroutines to compute the inside weights and the gradients (outside weights divided by Z), with a total $O(n^3)$ runtime. Thus, as outlined in the introduction (§1.4), a first-order parser is called each time we propagate through the global PTREE constraint, using edge weights that include the first-order LINK factors but also multiply in any current messages from higher-order factors. The parsing algorithm simultaneously computes the partition function $b()$ and all $O(n^2)$ marginal beliefs $b(L_{ij} = \text{true})$. In both cases, we thereby compute all $O(n^2)$ outgoing messages in $O(n^3)$ total time.

This propagation procedure could be sped up further by using a faster first-order parser with pruning or by prohibiting non-root attachments beyond a given length. Eisner and Smith (2005) showed that such a length restriction yields linear-time parsing with little loss in accuracy. The $O(n^3)$ PTREE propagator, however, does not asymptotically dominate propagation of the cubic number of second-order soft constraints, and we find that PTREE does not dominate runtime in practice.

The hard tree constraint weights all valid trees equally. Could we use a *weighted* parser in lieu of the PTREE factor? Clearly, there is no advantage in a weighted edge-factored parser: we could simply multiply any link weights into the LINK factors (or add extra unary factors). A dynamic-programming parser with weights for, e.g., child sequences, head valence, and grandparent relationships could use outside weights to calculate exact marginals for dependency links as with the edge-factored parser above.

If we had a parser already trained, a weighted tree factor could be used as a prior over trees. We could also incorporate some of the second-order constraints into the dynamic programming. For example, Eisner (2000) and McDonald and Pereira (2006) show how to incorporate CHILDSEQ features into a dependency parser while maintaining $O(n^3)$ runtime. As in generalized belief propagation (Yedidia et al., 2004), this technique could reduce search error, perhaps at the cost of runtime (depending on which extra features were included).

2.6 Decoding Trees

BP computes local beliefs, e.g. the conditional probability that a link L_{ij} is present. But if we wish to output a single well-formed dependency tree, we need to find a single assignment to all the $\{L_{ij}\}$ that satisfies the PTREE constraint.

Our final belief about the PTREE *factor* is a distribution over such assignments, in which a tree's probability is proportional to the probability of its edge weights $u(i,j)$ (incoming messages). We could simply return the mode of this distribution (found by using a 1-best first-order parser) or the k-best trees, or take samples.

In our experiments, we actually take the edge weights to be not the messages $u(i,j)$ from the links, but the full beliefs $\bar{b}_{L_{ij}}$ at the links (where $\bar{b}_{L_{ij}} \overset{\text{def}}{=} \log b_{L_{ij}}(\mathsf{true})/b_{L_{ij}}(\mathsf{false})$). These are passed into a fast algorithm for maximum projective spanning tree (Eisner, 1996) (or, in the next chapter, for maximum spanning tree). This procedure is equivalent to minimum Bayes risk (MBR) parsing (Goodman, 1996; Smith and Smith, 2007) with a dependency accuracy loss function (see §A.9).

Notice that the foregoing decoding approaches do not enforce any hard constraints other than PTREE in the final output. In addition, they only recover values of the L_{ij} variables. They marginalize over other variables such as tags and link roles. This solves the problem of "nuisance" variables (which merely fragment probability mass among refinements of a parse). On the other hand, it may be undesirable for variables whose values we desire to recover.

An alternative is to attempt to find the most probable assignment to all variables—using the max-product algorithm (footnote 18) or one of its recent variants. The estimated marginal beliefs become "max marginals," which assess the 1-best assignment consistent with each value of the variable.

We can indeed build max-product propagators for our global constraints. PTREE still propagates in $O(n^3)$ time: simply change the first-order parser's semiring (Goodman, 1999) to use max instead of sum. The 1-best parsing edge-factored algorithm (appendix B, algorithm B.1) finds the "Viterbi" inside weights; the maximizing version of the gradient computation finds the outside weights (appendix B, algorithm B.4).

If max-product BP converges, we may simply output each variable's favorite value (according to its belief), if unique. However, max-product BP tends to be unstable on loopy graphs, and we may not wish to wait for full convergence in any case. A more robust technique for extracting an assignment is to mimic Viterbi decoding, and "follow backpointers" of the max-product computation along some spanning subtree of the factor graph. A subtler approach is a greedy or branch-and-bound procedure where we fix the values of some variables and run inference recursively in hopes that this will help BP converge on other variables—much as propagation and backtracking search are interleaved in constraint satisfaction (Dechter, 2003).

A slower but potentially more stable alternative is **deterministic annealing**. Replace each factor $F_m(\mathcal{A})$ with $F_m(\mathcal{A})^{1/T}$, where $T > 0$ is a temperature. As $T \to 0$ ("cooling"), the distribution (2.5) retains the same mode (the MAP assignment), but becomes more sharply peaked at the mode, and sum-product BP approaches max-product BP. Deterministic annealing runs sum-product BP while gradually reducing T toward 0 as it iterates. By starting at a high T and reducing T slowly, it often manages in practice to find a good local optimum. We may then extract an assignment just as we do for max-product.

Finally, if we step outside the semantics of the factor graph, we can use general-purpose techniques such as **integer linear programming** (ILP) to search for an optimal assignment to all the variables. The score of a parse is a linear function since it is simply the inner product of features and weights. The values of the link variables, for instance, are constrained to be integers—a fortiori, since as booleans they are in $\{0, 1\}$. The condition that the link variables form a tree can also be enforced by linear constraints. But how many constraints are needed? Riedel and Clarke (2006) performed 1-best dependency parsing using ILP with cutting-plane relaxation so that they only need to consider a few of the exponentially many constraints prohibiting cycles. Martins et al. (2009a) expressed the tree constraint with only a quadratic number of constraints in the ILP: instead of directly prohibiting cycles, they used the encoding of single-commodity flow as an ILP to ensure that each word had a single parent and that the graph was connected, which implies that there are no cycles. In practice, Martins et al. find that solving the LP relaxation of the ILP is, unsurprisingly, much faster. When the LP returns fractional solutions, instead of rounding each variable independently, they pass the values to a first-order parser, just as we do with the variable beliefs. This gives them an end-to-end polynomial runtime.

2.7 Training

With access to a procedure for finding the best assignment to all variables under the model, we can train the model parameters using non-probabilistic methods such as the perceptron (Collins, 2002) or MIRA (Crammer et al., 2006). Linear programming relaxation methods would allow us to employ other large-margin training procedures (Taskar, 2004; Martins et al., 2009b). Instead, we will model a probability distribution over structured outputs—trees, in the case of a dependency parser. Our training method uses beliefs computed by BP. Instead of the beliefs at the variables we used for decoding the previous section, we will require the beliefs at the *factors*.

We choose the weight vector $\mathbf{w}$ to maximize the log-probability under (2.5) of training data, regularized by early stopping or by a quadratic penalty (see §§A.3–A.6). If all variables are observed in training, this objective function is *convex* (as for any log-linear model). Let a fully observed training tree $\mathbf{y}^*$ correspond to the variable assignment $\mathcal{A}^*$. The objective to be maximized is then the following log-likelihood derived from (2.5):

$$L \stackrel{\text{def}}{=} \log p(\mathcal{A}^*) = \sum_m F_m(\mathcal{A}^*) - \log Z \tag{2.16}$$

where the partition function $Z = \sum_{\mathcal{A}} p(\mathcal{A})$. See §A.3 and following for more on maximum conditional likelihood and the derivation of the gradient.

We employ stochastic gradient descent (§A.4 and Bottou, 2003), for its efficient convergence to good models and ease of implementation (algorithm 2.5). In addition, SGD does not require us to compute the objective function itself but only to (approximately) estimate its gradient. (But see §2.7.2 for approximating the objective function, as well.)

2.7.1 Approximating the gradient

The gradient of our objective function is:

$$\nabla_{\mathbf{w}} L = \sum_m \log F_m(\mathcal{A}^*) - \nabla_{\mathbf{w}} \log Z \tag{2.17}$$

The first term is just the score of a particular variable assignment $\mathcal{A}^*$, which is a constant with respect to w; the difficult step in computing this gradient is finding $\nabla_{\mathbf{w}} \log Z$. (Recall that Z, like each F_m, depends implicitly on W and w.) As usual for log-linear models,

$$\nabla_{\mathbf{w}} \log Z = \sum_m \mathbf{E}_{p(\mathcal{A})}[\nabla_{\mathbf{w}} F_m(\mathcal{A})] \tag{2.18}$$

Since $\nabla_{\mathbf{w}} F_m(\mathcal{A})$ only depends on the assignment $\mathcal{A}$'s values for variables that are connected to F_m in the factor graph, its expectation under $p(\mathcal{A})$ depends only on the marginalization of $p(\mathcal{A})$ to those variables jointly. Fortunately, BP provides an estimate of that marginal distribution, namely, its belief (2.10) about the factor F_m, given W and w (§2.3.2).[26]

The BP framework makes it tempting to extend an MRF model with various sorts of latent variables, whose values are not specified in training data. It is straightforward to train under these conditions. When counting which features fire on a training parse or (for error-driven training) on an current erroneous parse, we can find *expected* counts if these parses are not fully observed, by using BP to sum over latent variables. In either case, we adjust the parameters to increase $\sum_m \mathbf{E}_p[\nabla_{\mathbf{w}} F_m(\mathcal{A})]$, where p is the distribution conditioned on a

[26]This approximates the estimate that would be built up more slowly by Gibbs sampling. One could use coarser estimates at earlier stages of training, by running fewer iterations of BP. Note that the hard constraints like the tree factor do not depend on w at all; so their summands in equation (2.18) will be 0.

Algorithm 2.5 Stochastic gradient descent learning with BP

```
1: function SGD($\vec{x}, \vec{y}$)                        ▷ Vector of sentences x and true labels y
2:     $\mathbf{w} \leftarrow 0$
3:     $t \leftarrow 0$                                   ▷ Timesteps used to tune learning rate
4:     repeat
5:         for ($\mathbf{x} \in \vec{x}, \mathbf{y} \in \vec{y}$) do
6:             $\mathcal{G} \leftarrow$ factor graph constructed from w, x, and y
7:             $\mathcal{A}^* \leftarrow$ variable assignment induced by y
8:             BP($\mathcal{G}$)                                   ▷ Algorithm 2.1
9:             $\nabla_{\mathbf{w}} L \leftarrow \sum_m \log F_m(\mathcal{A}^*) - \sum_{m'} \sum_{\mathcal{A}} b_{F_{m'} \to}(\mathcal{A}) \nabla_{\mathbf{w}} F_{m'}(\mathcal{A})$
10:            $\mathbf{w} \leftarrow \mathbf{w} - \eta(t) \cdot \nabla_{\mathbf{w}} L$           ▷ Learning rate $\eta$ may decay with $t$
11:            $t \leftarrow t + 1$
12:        end for
13:    until w converges or max. iterations
14:    return w
15: end function
```

training parse, and to decrease it where p is the current distribution conditioned on nothing or on an erroneous parse. (See appendix A for more on training log-linear models with latent variables.)

2.7.2 Approximating the partition function

For other training procedures, and to evaluate the likelihood of the training data (e.g., on a held-out set), we need to approximate the partition function itself. Quasi-Newton methods such as L-BFGS (Liu and Nocedal, 1989), which are popular for batch-training conditional random fields, also require the value of the objective function to scale the update. When sum-product belief propagation converges, it does so at a fixed point of the Bethe free energy approximation to the true likelihood (Yedidia et al., 2000, 2004):

$$F_\beta = \sum_m \sum_{\mathcal{A}} b_m(\mathcal{A}) \log \frac{b_m(\mathcal{A})}{F_m(\mathcal{A})} - \sum_i (q_i - 1) \sum_{x_i} b_i(v) \log b_i(v) \tag{2.19}$$

where q_i is the degree of variable node i. It is of course intractable to compute the first, divergence term for combinatorial factors by brute-force summing over all assignments $\mathcal{A}$. Instead, for PTREE, we can compute the edge-factored tree entropy in cubic time using the formulation from §A.10. We use these factors as hard constraints, so the potentials F_m simply multiply in 1 for all valid trees. As we noted above, however, there is no bar to using a weighted parser as a factor.

2.8 Features and Feature Selection

Although BP makes it cheap to incorporate many non-local features and latent variables at once, we kept our models relatively simple for the monolingual parsers in this thesis.

2.8.1 Edge-factored features

Our first-order LINK$_{ij}$ factors replicate McDonald et al. (2005a). Following equation (2.6), they are defined using binary features that look at words i and j, the distance $j - i$, and the tags (provided in W) of words at, around, and between i and j.

At first, it may seem surprising that a parser that only predicts local parent-child attachments with a linear model, and relies on fairly weak global graph constraints such as arborescence and projectivity, performs as well as it does. Clearly, an intelligent choice of features is necessary to make these models work.

The LINK factors adopt the three categories of features from the edge-factored model of McDonald et al. (2005a):

1. **Edge features** are functions of the candidate parent and child words, their absolute sentence positions, and any of their attributes from earlier stages of processing. These features consider, for instance, whether the candidate child is to the left or right or the head and whether it is one word or twenty words away from the head. (In McDonald's model, distance is binned into 1, 2, 3, 4, and 5+.) The literal and case-folded form of the child and/or head are considered, as well as their first five characters, to approximate lemmatization. If, as here, a part-of-speech tagger is run before parsing, the tags of parent and child can be treated as observed and used by feature functions.

   Many edge features are of course highly informative: determiners are very likely to be left children of nouns, and nouns are very unlikely to be children of determiners; prepositions prefer to take nouns as right children. The edge-factored assumption, however, keeps us from stipulating that a preposition govern *only one* noun on its right or that a noun take *only one* determiner. For these and similar reasons, edge-factored parsers cannot achieve state-of-the-art performance with edge-factored features alone.

2. **Adjacency features** are functions of the sequence of POS tags on either side of the parent and child, as well as the tags of the parent and child themselves. From this space of $O(T^6)$ features, McDonald's model only uses features that consider at most four tags at a time.

   Adjacency features can approximate some of the properties of models that can score substructures with more than one dependency link at a time, while looking only at a single link and the input observations in x. A feature could fire, for instance, when a determiner attaches to a noun that occurs right after a comma, which would indicate that the noun is in a different phrase from the determiner. One would hope that this feature would have a negative weight. A verb that occurs right after a modal is likely to be a main verb and an attractive landing spot for sentence adverbials.

3. **Between features** are functions of the POS tags of the parent, child, and of some word between the parent and child positions. These features thus consider if a candidate link from a determiner to a noun, say, had to jump over a verb and thus become less likely.

When using this style of edge-factored model, it is important to note that it is less conveniently extended to parsing lattices than some other models whose inference uses dynamic programming. The adjacency features would require keeping $O(T^2)$ states for each word when making attachment decisions. The between features would require a dynamic programming computation between pairs of nodes in the lattice to collect statistics on which tags might occur between them. Similar computations would also be necessary if we wished to forgo running an independent POS tagging step in favor of joint morphological and syntactic inference.

2.8.2 Higher-order features

Our second-order features consider similar properties of the input and context. The feature templates for GRAND look at triples of tags and of coarse tags (as defined in the CoNLL data). A feature also fires if the grandparent falls between the child and parent, inducing crossing dependency links. The CHILDSEQ factors included features for tags, and likewise coarse tags, on adjacent sibling pairs and parent-sibling-sibling triples. Each of these features also have versions that were conjoined with link direction—pairs of directions in the grandparent case—or with signed link length of the child or farther sibling. Lengths were binned per McDonald et al. (2005a).

2.8.3 Feature selection

The feature functions described above can lead to a large set of features, particularly when we include lexicalization, and the feature space grows with the training data. As with most cases of structured prediction, we need to prune this feature growth to achieve acceptable runtime and memory usage without unduly hurting accuracy. While the problem of feature selection is not, of course, unique to dependency models, a judicious design of the feature space is very important for implementing a practical parser.

While a sentence of length n may be scored with $O(n)$ features, there are $O(n^2)$ candidate dependency links and thus a quadratic number of possible features. This observation leads to a common feature selection technique for supervised learning: use only the **supported features**, i.e. those features that fire on the links that form the correct tree in each training sentence. This criterion does not imply that no features fire on the incorrect links; rather, a link that is incorrect in one context may share features with a correct link from elsewhere in the training set. When a large number of the candidates are sparse features such as words and pairs of words, using only supported features often greatly reduces the size of the model; also, the supported-feature criterion often leads to parsers with better generalization, due to a lack of many nearly irrelevant features.

Unfortunately, when training on unlabeled or very small labeled sets (as in chapter 4), the set of supported features is unknown or highly variable. Several alternate feature-selection criteria are available, though none of them balance accuracy and model size as attractively as supported features.

- Select features that fire in at least some number of candidate dependency links, or some number of sentences. The latter criterion, with a threshold of 10 sentences, was used by Smith and Eisner (2007) and Druck et al. (2009).

- Since most dependencies are short (Eisner and Smith, 2005), select features that fire on short candidate links. Since some features in the original McDonald model bin all dependency lengths greater than five together, we can examine candidate links of length up to five without cutting of vast swaths of features.

- When we seek to improve a base parser by exploiting unlabeled data, via standard semi-supervised learning or other techniques (e.g., cross-language projection in chapter 4), we can use links "supported" by the predictions of the base parser to augment our feature set. A related technique, sometimes used for supervised learning, involves training the base model on-line with sparse, e.g. perceptron, updates (Roark et al., 2004).

- When the set of POS tags is small enough, we may extract unlexicalized features from all candidate links (as in the experiments in §4.7.3). In order to get some of the benefits of lexicalization, we can select features that reference common lexical items. An alternative is to use random or learned projections to reduce the dimensionality of lexicalized features (Ganchev and Dredze, 2008; Weinberger et al., 2009).

The feature selection techniques described above are generally applicable to various learning methods—batch and on-line, supervised and unsupervised. It usually takes one or two passes through the data to produce a new, sparser featurization, which the training code can save for more efficient future passes. An alternative approach is to use an on-line learning method and consider all candidate features for each example, but perform sparse updates via methods such as perceptron (Roark et al., 2004) or MIRA (Crammer and Singer, 2003; Crammer et al., 2006) or stochastic gradient descent with L1 regularization (Tsuruoka et al., 2009).

2.9 Experiments

How good is the approximation that BP produces by local message passing? For certain factors, we can use slow but exact dynamic programming (DP) to search for MAP trees and compute expectations in polynomial time. How does BP compare in speed and accuracy?

2.9.1 Experimental procedures

We trained and tested on the English data from the 2007 CoNLL Dependency Parsing Shared Task (Nivre et al., 2007). The CoNLL organizers converted the English data from the Penn Treebank to dependencies using a trace-recovery algorithm that induced some very slight non-projectivity—about 1% of links crossed other links. We removed sentences with nonprojective links from both training and test data. The input to training and test procedures W consisted of part-of-speech-tagged words (bearing both "coarse" and "fine" tags), so T variables were not used.

We trained all models using stochastic gradient descent (§2.7). SGD initialized $\mathbf{w} = 0$ and ran for 10 consecutive passes over the data; we picked the stopping point that performed best on held-out data.

When comparing runtimes for parsers, we took care to produce comparable implementations. All beliefs and dynamic programming items were stored and indexed using the high-level Dyna language,[27] while all inference and propagation was written in C++. The

[27]This dominates runtime, and probably slows down all our parsers by a factor of 4–11 owing to known inefficiencies in the Dyna prototype we used Eisner et al. (2005).

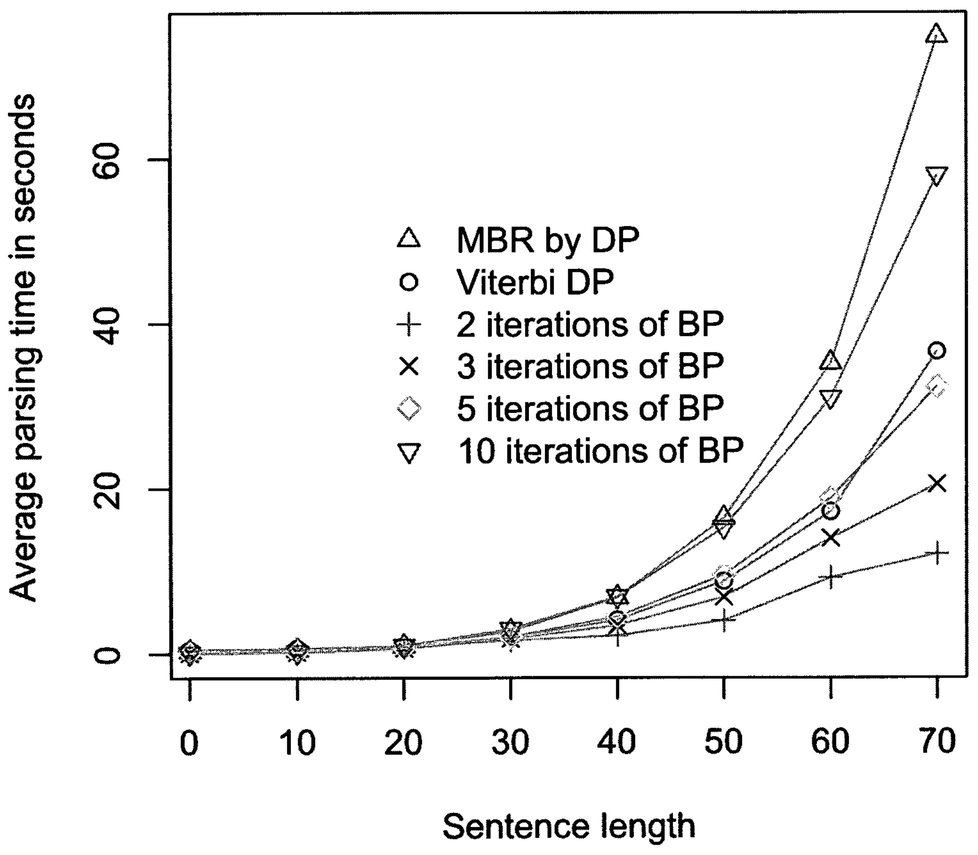


Figure 2.7: Runtime of BP parser on various sentence lengths compared to $O(n^4)$ dynamic programming

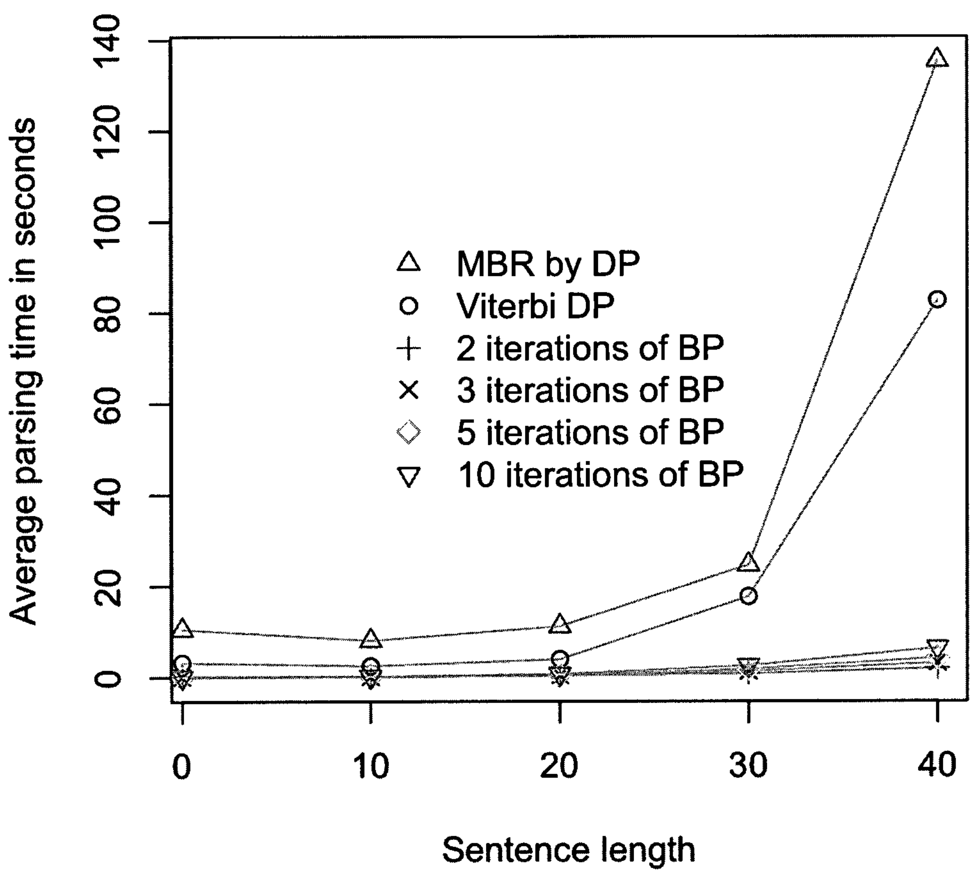


Figure 2.8: Runtime of BP parser on various sentence lengths compared to $O(n^5)$ dynamic programming. DP is so slow for length > 45 that we do not even show it.

BP parser averaged 1.8 seconds per sentence for non-projective parsing and 1.5 seconds per sentence for projective parsing (1.2 and 0.9 seconds/sentence for $\leq$ 40 words), using our standard setup, which included five iterations of BP and the final MBR tree decoding pass. For experiments with BP in later chapters, we stored beliefs and messages in native C++ data structures and achieved higher absolute speeds.

In our tables, we boldface the best result in each column along with any results that are not significantly worse (paired permutation test, $p < .05$).

2.9.2 Faster higher-order projective parsing

We built a first-order projective parser—one that uses only factors PTREE and LINK—and then compared the cost of incorporating second-order factors, GRAND and CHILDSEQ, by BP versus DP. We trained these parsers using exact DP, using the inside-outside algorithm to compute equation (2.18).

Under DP, the first-order runtime of $O(n^3)$ is increased to $O(n^4)$ with GRAND, and to $O(n^5)$ when we add CHILDSEQ as well. BP keeps runtime down to $O(n^3)$—although with a higher constant factor, since it takes several rounds to converge, and since it computes more than just the best parse.[28]

Figures 2.7–2.8 compare the empirical runtimes for various input sentence lengths. With only the GRAND factor, exact DP can still find the Viterbi parse (though not the MBR parse[28]) faster than ten iterations of the asymptotically better BP (Fig. 2.7), at least

[28]Viterbi parsing in the log domain only needs the $(\max, +)$ semiring, whereas both BP and any MBR

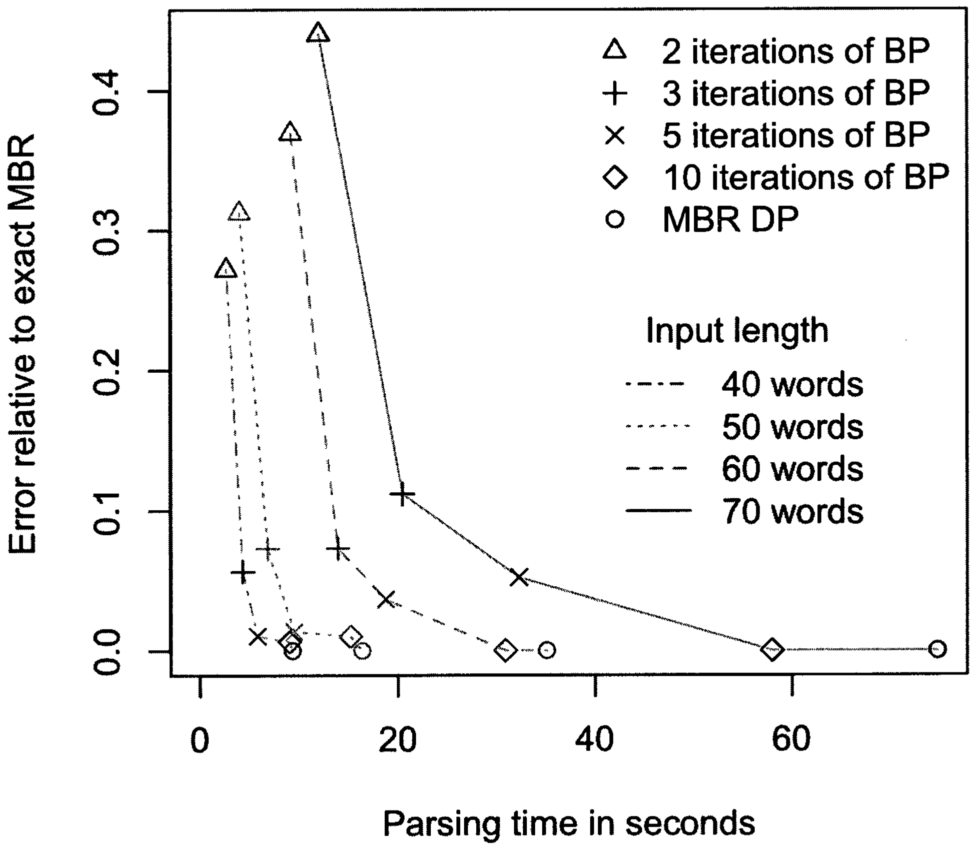


Figure 2.9: Runtime vs. search error after different numbers of BP iterations. This shows the simpler model of Fig. 2.7, where DP is still relatively fast.

for sentences with $n \leq 75$. However, once we add the CHILDSEQ factor, BP is always faster—dramatically so for longer sentences (Fig. 2.8). More complex models would widen BP's advantage.

Fig. 2.9 shows the tradeoff between runtime and search error of BP in the former case (GRAND only). To determine BP's search error at finding the MBR parse, we measured its dependency accuracy not against the gold standard, but against the optimal MBR parse under the model, which DP is able to find. We measure the proportion of dependency links in the true MAP parse that BP is able to recover. After 10 iterations, the overall macro-averaged search error compared to $O(n^4)$ DP MBR is 0.4%; compared to $O(n^5)$ (not shown), 2.4%. More BP iterations may help accuracy. In future work, we plan to compare BP's speed-accuracy curve on more complex projective models with the speed-accuracy curve of *pruned* or *reranked* DP.

2.10 Conclusions

In this chapter, we showed how to coordinate a fast first-order parser with higher-order features, by "hiding" it within one step of a general approximate inference algorithm for graphical models—loopy belief propagation. For projective parsing, it greatly speeds up dynamic programming—in theory and in practice—at the cost of small amounts of search error.

parsing must use the slower $(+, \log+)$ so that they can compute marginals (Goodman, 1999). DP does become empirically slower than BP, even in Fig. 2.7, if we ask it to find the MBR parse as BP does.

In the next chapter, we present another application of combinatorial optimization to propagating BP messages. The projectivity constraint on dependency trees is violated by many constructions in the world's languages (§3.1). Fortunately, we can replace the dynamic programs used in this chapter, such as the inside-outside algorithm, with other algorithms for directed spanning trees.

We are interested in extending the ideas in this chapter to phrase-structure and lattice parsing, and in trying other higher-order features, such as those used in parse reranking (Charniak and Johnson, 2005; Huang, 2008) and history-based parsing (Nivre and McDonald, 2008b). We could also introduce new variables, e.g., edge labels, nonterminal refinements (Matsuzaki et al., 2005), or secondary links M_{ij} (not constrained by the tree factor) that augment the parse with representations of control, binding, etc. (Sleator and Temperley, 1993; Buch-Kromann, 2006). For further discussion of extensions to our parsing models, see §5.1. Finally, we are interested in the further application of combinatorial algorithms within more generalized and efficient graphical model inference (§5.6).

Chapter 3

Algorithms for Nonprojective Dependency Parsing

The PTREE constraint for projective dependency trees presented in chapter 2 (and elaborated in appendix B) employs dynamic programming. DP methods are familiar to NLP researchers in applications of Dijkstra's shortest-path algorithm, Viterbi decoding for HMMs, and CKY parsing. In order to build up subtrees of increasing sizes, we exploited the projectivity constraint—the span of words descended from a particular head could not be interrupted by a word not so descended.

In this chapter, we will see that syntactic phenomena in several languages are more naturally handled if we relax this projectivity constraint (§3.1). We show how classical results from graph theory, which have only recently been applied in NLP, lead to $O(n^3)$ inference algorithms for edge-factored, nonprojective models—in particular, the Chu-Liu-Edmonds algorithm for the maximum weighted directed spanning tree and Tutte's directed matrix-tree theorem for the weighted sum. The matrix-tree theorem also enables efficient algorithms for computing the first- and second-order gradients of likelihood (§§3.2.3–3.2.4). We present experiments with maximum-likelihood learning of edge-factored nonprojective models and with minimum Bayes-risk decoding (§3.3.1).

Results from graph theory also show, however, the intractability of exact inference for models of nonprojective trees that score combinations of two or more dependency links. As in the previous chapter, we adapt exact inference algorithms for edge-factored parsing into $O(n^3)$ time propagators for a nonprojective TREE constraint in a factor graph. In §3.4, we present experiments on nonprojective parsing with loopy BP that outperform previous approximate inference procedures in accuracy and in asymptotic and empirical runtime.

3.1 Nonprojective Syntax

In §2.1.2, we enumerated three constraints on directed graphs that dependency parsers are designed to enforce: every node in the graph must have **in-degree 1** except for an in-degree 0 root node, the graph must be **acyclic**, and the graph must be **projective** with respect to the linear order of the words. (An optional constraint that the root have exactly one child might also be enforced.) This chapter explores models and methods for **nonprojective** trees, which relax the projectivity constraint.

Once the projectivity constraint is removed, dependency trees over n words become

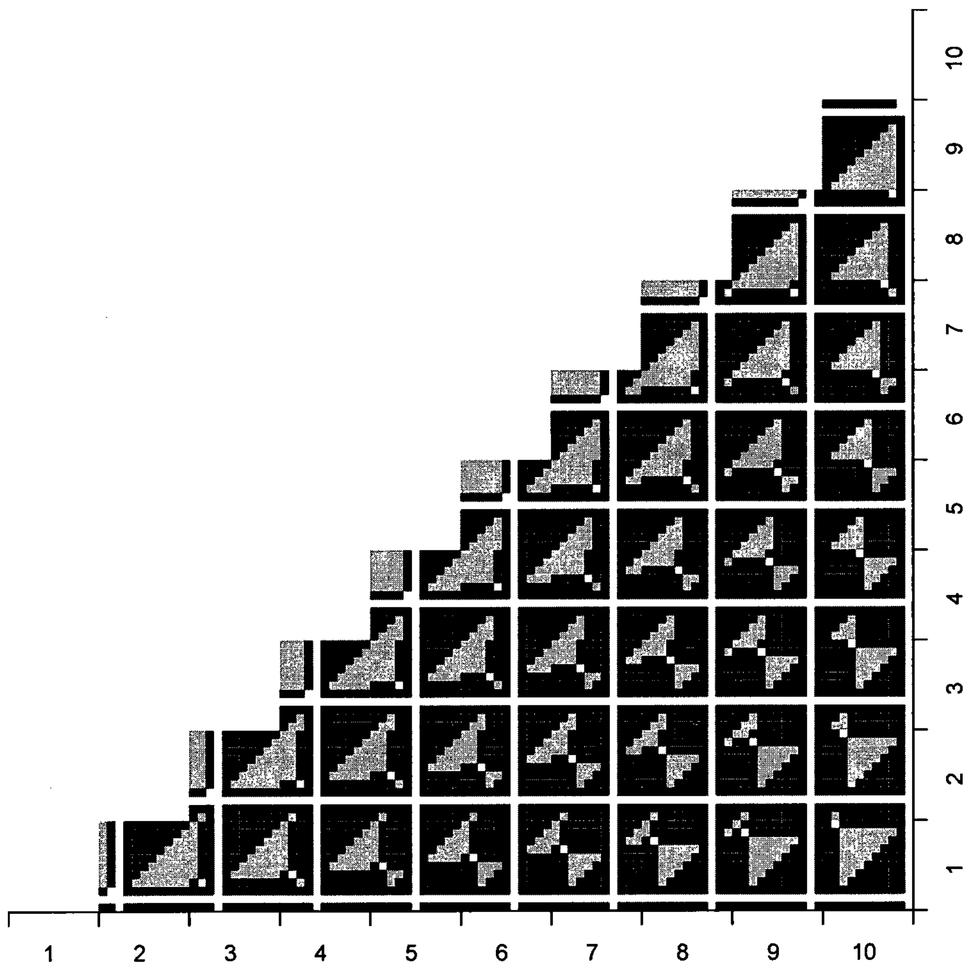


Figure 3.1: Valid edge pairs in projective and nonprojective trees for a ten-word sentence. Both the horizontal and vertical axes enumerate all possible edges, with the numbered blocks grouping children and their potential parents; a point in the graph is filled in if those edges can coexist in a tree. Wide white horizontal and vertical lines correspond to forbidden self-loops. Isolated white squares correspond to forbidden 2-cycles. Black cells (about 75% of the total) are allowed in both projective and nonprojective trees; grey cells indicate crossing edge pairs that are only allowed in nonprojective trees.

isomorphic to directed spanning trees over a graph of $n + 1$ nodes (including the root). (In contrast to trees with undirected edges, directed spanning trees are often called spanning arborescences.) When we look at the space of possible trees, the projectivity constraint has quite a large effect. Consider the sequence of Catalan numbers, which count the number of balanced binary bracketings of a sequence and thus related by a constant factor to the number of projective dependency trees. The nth Catalan number is $C_n = \frac{1}{n+1} \sum_{i=0}^{n} \binom{n}{i}^2$, which by Stirling's approximation is $C_n \approx \frac{4^n}{n^{3/2}\sqrt{\pi}}$. How many nonprojective trees are there? By Cayley's formula, there are n^{n-2} *undirected* spanning trees over n nodes; iterating over choice of root gives n^{n-1} directed spanning trees. Thus for an n-word sentence, there are $O(2\sqrt{\pi}(\frac{n}{4})^{n+1/2})$ times as many nonprojective as projective trees. Figure 3.1 shows the contrast in an alternate way: the crossing edge pairs allowed in nonprojective trees are shown in gray, against non-crossing edge pairs in black.

Such a strong constraint is thus not often given up lightly. In figure 3.2, we see an English sentence where the verb's adverbial child intrudes inside the object noun phrase. Since such nonprojective phenomena are quite rare in English, parsers can often achieve higher accuracy by being willing to get such attachments wrong than by allowing complete nonprojective freedom. The empirical tradeoff is different for languages with extensive scrambling phenomena (figures 3.3–3.4), such as Czech, Dutch, and Latin (Kuhlmann and Möhl, 2007; Gildea, 2010).

Among the languages included in the 2007 CoNLL Dependency Parsing Shared Task (Nivre et al., 2007), which we used for our experiments, there is a wide range of occur-

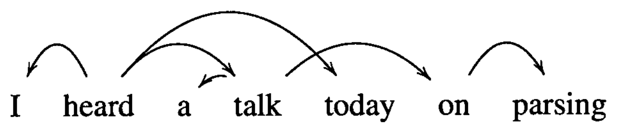


Figure 3.2: Mild nonprojectivity in English

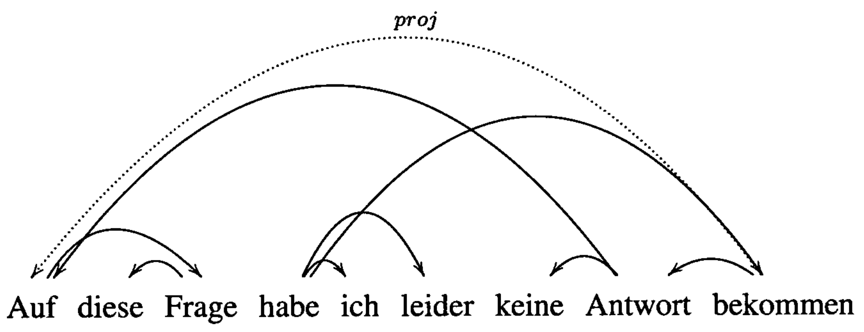
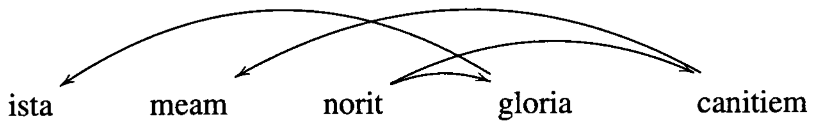


Figure 3.3: Extraposed prepositional phrase inducing nonprojectivity in German. The alternate attachment of *Auf* via the *proj* is used by some annotators.



that-NOM	my-ACC	may-know	glory-NOM	going-gray-ACC

That glory may last until my going gray.

Figure 3.4: Wild nonprojectivity in Latin

rence of nonprojectivity. The English data for that task were converted from the Penn Treebank to dependencies using a trace-recovery algorithm that induced some very slight nonprojectivity—the approximately 1% of links that cross other links are the lowest among these languages. Danish represents a slightly more nonprojective language (3% crossing links). Dutch is the most nonprojective language in the corpus (11%).

3.2 Edge-Factored Models of Nonprojective Trees

Recall from §1.2 that we can take the score for an input-output pair $s(\mathbf{x}, \mathbf{y})$, exponentiate it to get an unnormalized probability $u(\mathbf{x}, \mathbf{y})$, and then renormalize it by $Z_{\mathbf{x}} = \sum_{\mathbf{y}'} u(\mathbf{x}, \mathbf{y}')$ to get $p(\mathbf{y} \mid \mathbf{x})$ (1.2). In §2.1.3, furthermore, we considered edge-factored tree models where a tree's score is measured by the sum of edge scores or, equivalently, the product of exponentiated edges scores. For a particular input sentence $\mathbf{x}$, we will write for a dependency link $x_i \rightarrow x_j$ that $u(i, j) = e^{s(i,j)}$.

As in §2.1.2, we define a weighted directed graph G over the vertices $V = \{x_0, x_1, \ldots, x_n\}$, corresponding to the ROOT symbol and the n words in the sentence. The edges of G are defined as $E = \{x_i \rightarrow x_j : i \neq j, j \neq 0\}$. With the weight of $x_i \rightarrow x_j$ defined, as above, as $s(i, j)$, we define the **maximum (directed) spanning tree** (MST) as the directed tree $\mathbf{y}$ that maximizes $\sum_{x_i \rightarrow x_j \in \mathbf{y}} s(i, j)$ and that contains all G's vertices. A spanning tree, whether it is the maximum or not, is a subgraph of G that must satisfy all of the constraints on directed trees enumerated in §2.1.2, except for projectivity, to wit: ROOT must have no in-arcs, every other node must have in-degree 1, and the subgraph must be acyclic. To distinguish the MST problem for directed graphs from the more familiar one for undirected graphs, the terms maximum (spanning) arborescence are sometimes used.

Without the projectivity restriction, we cannot use the dynamic programs from chapter 2 to find the tree with the highest score or the sum of all trees. These algorithms (e.g., algorithm B.1) build spans of the left and right descendants of a headword. It can be shown by induction that these spans will be contiguous since the dynamic program starts by building dependencies where the parent and child are adjacent. But the projectivity constraint can be stated as a prohibition on discontinuous constituents (2.1), and so these dynamic programs cannot recover nonprojective trees.[1]

In this section, we discuss other algorithms from the graph-theory literature for efficient edge-factored inference on nonprojective trees. Solving the maximization problem allows us, among other things, to find the maximum a posteriori parse tree under an edge-factored model (§3.2.1); solving the summation problem allows us to compute the normalizing constant $Z_{\mathbf{x}}$ and its gradient (§3.2.2). The algorithms we will discuss run in $O(n^3)$ time and are thus efficient nonprojective analogues to the Eisner algorithm for projective edge-factored parsing (algorithm B.1) and its inside-outside variant for summing and marginalization

[1]Although the cubic-time parsing algorithm of Eisner (1996) and its extensions for computing inside and outside weights no longer applies, that does not mean that polynomial-time dynamic programming for non-projective parsing is impossible in all cases. Gildea (2010) shows that nonprojective trees can be recovered by a linear context-free rewriting system parser that runs in time exponential in the number of "gaps" in a constituent. The number of such gaps necessary for parsing several existing treebanks is much smaller than the average sentence length. In this chapter, however, we demonstrate fully cubic-time parsing for nonprojective trees, which is thus strictly faster than $O(n^6)$ LCFRS parsing with only one gap allowed.

(algorithms B.2 and B.3).

3.2.1 Maximum spanning tree parsing

As we saw above, edge-factored nonprojective parsing searches the space of spanning trees of a directed graph over the words of a sentence. Prim's, Kruskal's, and Borůvka's algorithms for finding the highest-weighted *undirected* spanning tree are widespread in introductory courses and textbooks (e.g., Cormen et al., 1990). Less widely known are algorithms for finding directed spanning trees: we will follow McDonald et al. (2005b), who first pointed out the connection of the MST problem to nonprojective parsing, and use the Chu-Liu-Edmonds (CLE) algorithm (Chu and Liu, 1965; Edmonds, 1967).

Like Borůvka's algorithm, CLE starts by attaching each node to its parent via the in-edge with the highest weight. This is a spanning subgraph, since it covers all nodes, and if it is acyclic, it is the MST. If there is a cycle, it is "contracted" into a single vertex: the new vertex has the union of the sets of in- and out-edges of members of the cycle. It can be shown that the MST of the contracted graph has edges equivalent to the MST on the full graph (Georgiadis, 2003). CLE can thus recursively call itself on the contracted graph and then break the cycle in the full graph: the node in the full graph that is the destination for the edge into the contracted node must no longer have an in-edge from another member of the cycle. We can see that attaching each node to its best parent and checking for cycles takes $O(n^2)$ time; furthermore, it can be shown that no more than $O(n)$ cycles need to be contracted. The Chu-Liu-Edmonds algorithm therefore runs in $O(n^3)$ time.

Tarjan (1977) achieved an even better $O(n^2)$ runtime for dense graphs (as we have here) by efficient data structures for disjoint union. We are not aware of MST implementations in NLP that use this more involved algorithm. At any rate, the relatively small graphs used in parsing, with node counts in the low hundreds, may not repay the necessary constant overhead of Tarjan's algorithm.

Algorithms also exist for finding the k-best spanning trees in time proportional to k, e.g. in $O(kn^2)$ time on a dense graph using Tarjan's speedups (Camerini et al., 1980). Hall (2007) used this approach to find the k-best trees from an edge-factored model and then applied full-tree reranking.

3.2.2 The matrix-tree theorem and the partition function

The Chu-Liu-Edmonds algorithm enables maximum a posteriori decoding for edge-factored nonprojective models; likewise, error-driven training with the perceptron (Koo et al., 2007) and large-margin methods such as MIRA (McDonald et al., 2005b) require only the maximum over spanning trees. (MIRA can also take advantage of k-best lists.)

Training to maximize conditional likelihood—as well as computing likelihood, entropy, and other quantities—requires the partition function Z_{x} (the sum of the scores of all trees licensed by x) and/or its gradient.[2] For a sufficiently skewed distribution, where a large amount of the probability mass is concentrated in a small number of trees, we might be able to approximate the sum of scores of all trees adequately by enumerating the top k

[2]See §A.3 in the appendices for background on conditional likelihood training.

trees. In this subsection, however, we will discuss a more direct and efficient solution to summing over exponentially many spanning trees in cubic time.[3]

Define the **Kirchhoff matrix** $\mathbf{K} \in \mathbb{R}^{(n+1)\times(n+1)}$, also known as the graph Laplacian, by

$$[\mathbf{K}]_{mom,kid} = \begin{cases} -u(mom, kid) & \text{if } mom \neq kid \\ \sum_{i=0}^{n} u(i, kid) & \text{if } mom = kid. \end{cases}$$

where mom indexes a parent node and kid a child node. For simplicity in notation, we make the dependence of the Kirchhoff matrix on the input $\mathbf{x}$ and parameters $\mathbf{w}$ implicit. $\mathbf{K}$ (Figure 3.5) can be regarded as a special weighted adjacency matrix in which the jth diagonal entry is the sum of edge-scores directed into vertex j (i.e., x_j is the child). Note that the sum includes the score of attaching x_j to the root x_0.

In our notation and in one specific form, the Directed Matrix Tree Theorem (Tutte, 1948, 1984) states:[4]

Theorem 1 *The determinant of the submatrix* $\mathbf{K}(r;r)$ *obtained by removing row and column* r *from the Kirchhoff matrix* $\mathbf{K}$ *is equal to the sum of scores of all directed spanning*

[3]We first published these results in Smith and Smith (2007). Two independent papers, published concurrently with that one, report closely related results to ours. Koo et al. (2007) and McDonald and Satta (2007) both describe how the Matrix Tree Theorem can be applied to computing the sum of scores of edge-factored dependency trees and the edge marginals. Koo et al. compare conditional likelihood training (as here) to the averaged perceptron and a maximum margin model trained using exponentiated-gradient (Bartlett et al., 2004); the latter requires the same marginalization calculations as conditional log-linear estimation. McDonald and Satta discuss a variety of applications (including minimum Bayes-risk decoding) and give complexity results for non-edge-factored models. Interested readers are referred to those papers for further discussion.

[4]There are proven generalizations of this theorem (Chen, 1965; Chaiken, 1982; Minoux, 1999); we give the most specific form that applies to our case, originally proved by Tutte in 1948. The undirected matrix-tree theorem was exploited by Jaakkola et al. (1999) for inference in graphical models. As usual, the reader should take care not to confuse the tree structure represented by the graphical model with the tree structure (or lack thereof due to loops) of the factor graph representation.

$$\mathbf{K} = \begin{bmatrix} 0 & -u(0,1) & -u(0,2) & \cdots & -u(0,n) \\ 0 & \sum_{i=0}^{n} u(i,1) & -u(1,2) & \cdots & -u(1,n) \\ 0 & -u(2,1) & \sum_{i=0}^{n} u(i,2) & \cdots & -u(2,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -u(n,1) & -u(n,2) & \cdots & \sum_{i=0}^{n} u(i,n) \end{bmatrix}$$

Figure 3.5: The Kirchhoff matrix for an n-word sentence. Children run across the columns and parents run down the rows. Since the root note does not have a parent, the first column is zero. Since we define $u(i,i) = 0$, there are only n terms in the sums in the diagonal cells.

trees in $\mathcal{Y}_{\mathbf{x}}$ rooted at x_r. Formally, $|\mathbf{K}(0;0)| = Z_{\mathbf{x}}$.

See Tutte (1984) for a proof, which involves decomposing the Kirchhoff matrix by minors. In §3.2.3 below, we exploit this decomposition to derive its gradient.

For brevity, let $\mathbf{K}_0 = \mathbf{K}(0;0)$. To compute $Z_{\mathbf{x}}$, we need only take the determinant of $\mathbf{K}_0$, which can be done in $O(n^3)$ time using the standard LU factorization used to compute the matrix inverse. Since all of the edge weights used to construct the Kirchhoff matrix are positive, it is diagonally dominant and therefore non-singular (i.e., invertible).

3.2.3 The gradient of the partition function

The first derivative of $\log Z$ with respect to a single weight w_ℓ is equal to the ℓth feature's expected value (see §A.5):

$$\frac{\partial \log Z}{\partial w_\ell} = \mathbf{E}[f_\ell(\mathbf{x}, \cdot)] \tag{3.1}$$

This is one reason that marginals are so important in maximum likelihood estimation (§A.4): they provide a gradient (vector of first derivatives) to tell an optimization algorithm which direction to move the current estimate of $\mathbf{w}$ at each round. Many likelihood-based training algorithms exploit this identity by computing expectations (e.g., by sampling); others use symbolic differentiation of the partition function computation (e.g., algorithm B.3 for projective parsing). We will instead use some facts from linear algebra and the chain rule to derive the gradient/expectation values.

First, for any weight w_ℓ, and by Theorem 1 and the edge-factoring assumption (2.3):

$$\begin{aligned}
\frac{\partial \log Z}{\partial w_\ell} &= \frac{\partial \log |\mathbf{K}_0|}{\partial w_\ell} \\
&= \frac{1}{|\mathbf{K}_0|}\frac{\partial |\mathbf{K}_0|}{\partial w_\ell} \\
&= \frac{1}{|\mathbf{K}_0|}\sum_{j=1}^{n}\sum_{i=0}^{n}\frac{\partial |\mathbf{K}_0|}{\partial u(i,j)}\frac{\partial u(i,j)}{\partial w_\ell} \\
&= \frac{1}{|\mathbf{K}_0|}\sum_{j=1}^{n}\sum_{i=0}^{n} u(i,j) f_\ell(\mathbf{x},i,j)\frac{\partial |\mathbf{K}_0|}{\partial u(i,j)} \qquad (3.2)
\end{aligned}$$

We let $u(i,i) = 0$ for simplicity of notation. The last line follows from the definition of $u(i,j)$ as $\exp[\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, i, j)]$. Now, since $u(i,j)$ affects the Kirchhoff matrix in *at most* two cells—(i, i) and (i, j), the latter only when $i > 0$—we know that

$$\begin{aligned}\frac{\partial|\mathbf{K}_0|}{\partial u(i,j)} &= \frac{\partial|\mathbf{K}_0|}{\partial[\mathbf{K}_0]_{j,j}}\frac{\partial[\mathbf{K}_0]_{j,j}}{\partial u(i,j)} + \frac{\partial|\mathbf{K}_0|}{\partial[\mathbf{K}_0]_{i,j}}\frac{\partial[\mathbf{K}_0]_{i,j}}{\partial u(i,j)} \\ &= \frac{\partial|\mathbf{K}_0|}{\partial[\mathbf{K}_0]_{j,j}} - \frac{\partial|\mathbf{K}_0|}{\partial[\mathbf{K}_0]_{i,j}}\end{aligned} \tag{3.3}$$

We have now reduced the problem of the gradient to a linear function of $\nabla|\mathbf{K}_0|$ with respect to the cells of the matrix itself.

At this point, we simplify notation and consider an arbitrary matrix $\mathbf{A}$. The **minor** $m_{i,j}$ of a matrix $\mathbf{A}$ is the determinant of the submatrix obtained by striking out row i and column j of $\mathbf{A}$. The **cofactor** $c_{i,j}$ of $\mathbf{A}$ is given by $(-1)^{i+j}m_{i,j}$. Laplace's formula defines the determinant as a linear combination of matrix cofactors of an arbitrary row i:

$$|\mathbf{A}| = \sum_{j=1}^{n}[\mathbf{A}]_{i,j}c_{i,j} \tag{3.4}$$

It should be clear that any $c_{i,k}$ is constant with respect to the cell $[\mathbf{A}]_{i,j}$, since it is formed by removing row i of $\mathbf{A}$. It should also be clear that other entries of $\mathbf{A}$ are constant with respect to the cell $[\mathbf{A}]_{i,j}$. Therefore:

$$\frac{\partial|\mathbf{A}|}{\partial[\mathbf{A}]_{i,j}} = c_{i,j} \tag{3.5}$$

The inverse matrix $\mathbf{A}^{-1}$ can also be defined in terms of cofactors:

$$[\mathbf{A}^{-1}]_{j,i} = \frac{c_{i,j}}{|\mathbf{A}|} \tag{3.6}$$

Combining (3.5) and (3.6), we have:

$$\frac{\partial|\mathbf{A}|}{\partial[\mathbf{A}]_{i,j}} = |\mathbf{A}|[\mathbf{A}^{-1}]_{j,i} \tag{3.7}$$

Plugging back in through (3.3) to (3.2), we have:

$$\begin{aligned}
\mathbf{E}[f_\ell; \mathbf{w}] &= \frac{\partial \log Z}{\partial w_\ell} \\
&= \sum_{j=1}^{n} \sum_{i=0}^{n} u(i,j) f_\ell(x,i,j) \left(\left[\mathbf{K}_0^{-1}\right]_{j,j} - \left[\mathbf{K}_0^{-1}\right]_{j,i} \right) \qquad (3.8)
\end{aligned}$$

where $\left[\mathbf{K}_0^{-1}\right]_{j,0}$ is taken to be 0. Note that the cofactors do not need to be computed directly. We proposed in §3.2.2 to get $Z_{\mathbf{x}}$ by computing the LU factorization of the Kirchhoff matrix. We can then start from this factorization and backsolve to get the inverse matrix, which as we pointed out is known to exist since $\mathbf{K}_0$ is diagonally dominant. This process gives us all $O(n^2)$ entries of $\mathbf{K}_0^{-1}$ in $O(n^3)$ time.

3.2.4 The second derivative and minimizing risk

In addition to providing the gradient for the partition function, and thus the likelihood objective function, feature expectations form the basis for other objective functions. In particular, **minimum risk** training for dependency parsers seeks to minimize the expected number of erroneous dependency links according to the model (Smith and Eisner, 2006a). Information-theoretic measures such as entropy and Kullback-Leibler divergence, which are optimized in some semi-supervised learning methods, require similar computations (see Smith and Eisner, 2007; Druck et al., 2009, and §§A.8, A.10 in the appendix). To perform gradient-based optimization on these quantities, we need to differentiate again: in the most general terms, we need the matrix of second partial derivatives of all pairs of parameters (the Hessian). For many problems, however, we simply need the gradient of the appropriate objective function with respect to the parameters $\mathbf{w}$.

In this subsection, we derive the gradient of the inverse Kirchhoff matrix and, from it, the gradient of the risk function for dependency trees.[5] Second-derivative computations are also necessary for backpropagation through an unrolled BP computation containing a combinatorial tree factor (§5.3.2).

We wish to calculate the expected number of correct (or incorrect) dependency attachments made by a model. This is very simple given the ability to marginalize over edge-factored features. Suppose for a given example $\mathbf{x}$ we know its true parse $\mathbf{y}^*$. For a given candidate tree $\mathbf{y}$, the attachment loss is

$$L_{\mathbf{y}^*}(\mathbf{y}) \sum_{i=1}^{|x|} \begin{cases} 0 & \text{if } y_i = y_i^* \\ 1 & \text{otherwise} \end{cases} \tag{3.9}$$

It should be clear that minimizing this loss is equivalent to *maximizing* the number of correct attachments, which will provide a more convenient form for computational ends:

$$D_{\mathbf{y}^*}(\mathbf{y}) \stackrel{\text{def}}{=} \sum_{i=1}^{|x|} \begin{cases} 1 & \text{if } y_i = y_i^* \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Crucially, D can be treated as an edge-factored feature function. It happens to depend on the true tree as well as $\mathbf{x}$ and $\mathbf{y}$, so it is not a feature function we could use within the model, but during training, when each y_i^* is available, this is a perfectly reasonable feature whose

[5]We have applied some of the material in this section in a technical report on efficient generalized expectation training (Druck and Smith, 2009). Previous work on generalized expectation training for dependency parsing had relied on a brute-force $O(n^5)$ gradient computation (Druck et al., 2009).

expectation can be calculated using the results of §3.2.3. Using (3.8), we have

$$\mathbf{E}[D_{\mathbf{y}^*};\mathbf{w}] = \sum_{i=1}^{|x|} u(y_j^*, j)\left(\left[\mathbf{K}_0^{-1}\right]_{j,j} - \left[\mathbf{K}_0^{-1}\right]_{j,y_j^*}\right) \tag{3.11}$$

where y_i^* is the index of the correct parent of the ith word, x_i. To perform efficient numerical optimization of this function, we require its gradient with respect to the weights $\mathbf{w}$. The partial derivative of an inverse matrix $\mathbf{A}^{-1}$ cell (k,ℓ) with respect to cell (i,j) of $\mathbf{A}$ is:

$$\frac{\partial\left[\mathbf{A}^{-1}\right]_{k,\ell}}{\partial[\mathbf{A}]_{i,j}} = -\left[\mathbf{A}^{-1}\right]_{k,i}\left[\mathbf{A}^{-1}\right]_{j,\ell} \tag{3.12}$$

Exploiting the same properties of the Kirchhoff matrix used in (3.3), we have

$$\frac{\partial\left[\mathbf{K}_0^{-1}\right]_{k,\ell}}{\partial u(i,j)} = -\left[\mathbf{K}_0^{-1}\right]_{k,i}\left[\mathbf{K}_0^{-1}\right]_{i,\ell} + \left[\mathbf{K}_0^{-1}\right]_{k,j}\left[\mathbf{K}_0^{-1}\right]_{i,\ell} \tag{3.13}$$

Using the chain rule, this gives:

$$\begin{aligned}\frac{\partial\mathbf{E}[D;\mathbf{w}]}{\partial u(i,j)} &= \delta(y_i^*, j)\left(\left[\mathbf{K}_0^{-1}\right]_{i,i} - \left[\mathbf{K}_0^{-1}\right]_{i,j}\right)\\ &\quad + \sum_{c=1}^{|x|} u(y_c^*, c)\left[\left(\left[\mathbf{K}_0^{-1}\right]_{i,c} - \left[\mathbf{K}_0^{-1}\right]_{i,y_c^*}\right)\right.\\ &\quad \left.\cdot\left(\left[\mathbf{K}_0^{-1}\right]_{c,j} - \left[\mathbf{K}_0^{-1}\right]_{c,i}\right)\right]\end{aligned} \tag{3.14}$$

Another application of the chain rule yields $\partial\mathbf{E}[D;\mathrm{w}]/\partial\mathrm{w}_m$ for the mth weight. The total gradient would sum over such terms across the entire corpus, of course.

3.2.5 Hidden variables

Another application of probability distributions over trees is hidden-variable learning. Recently, EM has been used to learn hidden variables in parse trees; these can be head-child annotations (Chiang and Bikel, 2002), latent head features (Matsuzaki et al., 2005;

Prescher, 2005; Dreyer and Eisner, 2006), or hierarchically-split nonterminal states (Petrov et al., 2006).

To date, we know of no successful attempts to apply hidden variables to supervised dependency tree models. If the trees are constrained to be projective, EM is easily applied using the inside-outside variant of the parsing algorithm described by Eisner (1996) to compute the marginal probability. Moving to the nonprojective case, there are two difficulties: (a) we must marginalize over nonprojective trees and (b) we must define a generative model over $(\mathbf{x}, \mathbf{Y})$.

We have already shown in §3.2.2 how to solve (a); here we could avoid (b) by maximizing *conditional* likelihood, marginalizing out the hidden variable, denoted $\mathbf{z}$:

$$\max_{\mathbf{w}} \sum_{\mathbf{x},\mathbf{y}} \tilde{p}(\mathbf{x}, \mathbf{y}) \log \sum_{\mathbf{z}} p_{\mathbf{w}}(\mathbf{y}, \mathbf{z} \mid \mathbf{x}) \tag{3.15}$$

This sort of conditional training with hidden variables was carried out by Koo and Collins (2005), for example, in reranking; it is related to the information bottleneck method (Tishby et al., 1999) and contrastive estimation (Smith and Eisner, 2005a).

In Smith and Smith (2007), we performed experiments learning hidden labels for an edge-factored parser, but performance was no better than a parser without hidden labels. While relatively simple clustering methods have been shown to help CFG parsing, the latent features there are on nonterminal states, which are the structural "bridge" between context-free rules. Adding features to those states is a way of pushing information—encoded indirectly, perhaps—farther around the tree, and therefore circumventing the strict independence assumptions of probabilistic CFGs.

decode	train	Arabic	Czech	Danish	Dutch	
MAP	MIRA	79.9	**81.4**	86.6	**90.0**	
	CL	80.4	80.2	**87.5**	**90.0**	(§3.2.2)
MBR	MIRA	79.4	80.3	85.0	87.2	(§3.3.1)
	CL	**80.5**	80.4	**87.5**	**90.0**	(§§3.2.2 & 3.3.1)

Table 3.1: Unlabeled dependency parsing accuracy (on test data) for two training methods (MIRA, as in McDonald et al. (2005b), and conditional likelihood estimation) and with maximum *a posteriori* (MAP) and minimum Bayes-risk (MBR) decoding. **Boldface** scores are best in their column on a permutation test at the .05 level.

In an edge-factored dependency model, on the other hand, latent features on the edges are locally "summed out" when we compute the scores of possible attachments; it should be clear that the edge clusters do not circumvent any independence assumptions. We might attempt to learn clusters in tandem with estimating a richer, non-edge-factored model (§3.4).[6]

3.3 Experiments with Edge-Factored Training

Now that we have developed the machinery for probabilistic edge-factored models of nonprojective trees, we compare conditional-likelihood training to the online MIRA training used by McDonald et al. (2005b). Four languages with relatively common nonprojec-

[6]See §5.1.1 for further discussion of hidden variables for monolingual parsing. In chapter 4, we discuss bilingual parsing models where we sum out source-language and alignment variables when inferring target-language trees.

tive phenomena were tested: Arabic (Hajič et al., 2004), Czech (Böhmová et al., 2003), Danish (Kromann, 2003), and Dutch (van der Beek et al., 2002). The Danish and Dutch datasets were prepared for the CoNLL 2006 shared task (Buchholz and Marsi, 2006); Arabic and Czech are from the 2007 shared task. We used the same edge-factored features as in the projective parsing experiments in the last chapter (§2.8.1).

Also as in the projective experiments, our conditional training used the online gradient-based method of stochastic gradient descent. Training with MIRA and conditional estimation take about the same amount of time: approximately 50 sentences per second. Training proceeded as long as an improvement on held-out data was evident. The accuracy of the hypothesized parses for the two models, on each language, are shown in the top two rows of Table 3.1 (labeled "MAP" for maximum *a posteriori*, meaning that the highest-weighted tree is hypothesized).

The two methods are, not surprisingly, close in performance; conditional likelihood outperformed MIRA on Arabic and Danish, underperformed MIRA on Czech, and the two tied on Dutch. Results are significant at the .05 level on a permutation test. Conditional estimation is in practice more prone to over-fitting than maximum margin methods, though we did not see any improvement using zero-mean Gaussian priors (variance 1 or 10).

These experiments serve to validate conditional estimation as a competitive learning algorithm for nonprojective parsing models, and the key contribution of the summing algorithm from the last section that permits conditional estimation.

3.3.1 Minimum Bayes-risk decoding

A second application of probability distributions over trees is the alternative decoding algorithm known as **minimum Bayes-risk** (MBR) decoding. The more commonly used maximum *a posteriori* decoding (also known as "Viterbi" decoding) that we applied in §3.3 sought to minimize the expected whole-tree loss:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}}\, p_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}}\, \mathbf{E}_{p_{\mathbf{w}}(\mathbf{Y}\mid\mathbf{x})}\left[-\delta(\mathbf{y},\mathbf{Y})\right] \tag{3.16}$$

Minimum Bayes-risk decoding generalizes this idea to an arbitrary loss function ℓ on the proposed tree:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmin}}\, \mathbf{E}_{p_{\mathbf{w}}(\mathbf{Y}\mid\mathbf{x})}\left[\ell(\mathbf{y},\mathbf{Y})\right] \tag{3.17}$$

This technique was originally applied in speech recognition (Goel and Byrne, 2000) and translation (Kumar and Byrne, 2004). Goodman (1996) proposed a similar idea in probabilistic context-free parsing, seeking to maximize expected recall, and later applied it to finding the best derived tree in "data-oriented" tree-substitution grammar parsing (Goodman, 2003). In the recent parsing literature, MBR has been widely used for decoding with hidden variables (Matsuzaki et al., 2005; Dreyer and Eisner, 2006; Petrov and Klein, 2007; Cohen and Smith, 2007).

The most common loss function used to evaluate dependency parsers is the number of attachment errors, so we seek to decode using:

$$\begin{aligned}\hat{\mathbf{y}} &= \operatorname*{argmin}_{\mathbf{y}} \mathbf{E}_{p_{\mathbf{w}}(\mathbf{Y}|\mathbf{x})}\left[\sum_{i=1}^{n} -\delta(\mathbf{y}(i), \mathbf{Y}(i))\right] \\ &= \operatorname*{argmax}_{\mathbf{y}} \sum_{i=1}^{n} p_{\mathbf{w}}(\mathbf{Y}(i) = \mathbf{y}(i) \mid \mathbf{x}) \qquad (3.18)\end{aligned}$$

To apply this decoding method, we make use of (3.8), which gives us the posterior probabilities of edges under the model, and the same Chu-Liu-Edmonds maximum directed spanning tree algorithm used for maximum *a posteriori* decoding (§3.2.1). Note that this decoding method can be applied regardless of how the model is trained. It merely requires assuming that the tree scores under the trained model (probabilistic or not) can be treated as unnormalized log-probabilities over trees given the sentence $\mathbf{x}$.

We applied minimum Bayes-risk decoding to the models trained using MIRA and using conditional estimation (see §3.3). Table 3.1 shows that, across languages, minimum Bayes-risk decoding *hurts* slightly the performance of a MIRA-trained model, but *helps* slightly or does not affect the performance of a conditionally-trained model. Since MIRA does not attempt to model the distribution over trees, this result is not surprising; interpreting weights as defining a conditional log-linear distribution is questionable under MIRA's training criterion.

One option, which we do not test here, is to use minimum Bayes-risk decoding *inside* of MIRA training, to propose a hypothesis tree (or k-best trees) at each *training* step. Doing this would more closely match the training conditions with the testing conditions; however, it is unclear whether there is a formal interpretation of such a combination, for example its relationship to the "factored MIRA" in McDonald et al. (2005a).

In other dependency parsing scenarios (Smith and Eisner, 2007), minimum Bayes-risk decoding has been found to offer significant advantages—why not here? Minimum Bayes-risk makes use of global statistical dependencies in the posterior when making local decisions. But in an edge-factored model, the edges are all conditionally independent, given that $\mathbf{y}$ is a spanning tree.

As a *post hoc* experiment, we compared purely greedy attachment (attach each word to its maximum-weighted parent, without any tree constraints). Edge scores as defined in the model were compared to minimum Bayes-risk posterior scores, and the latter were consistently better (though this always under-performed optimal spanning-tree decoding, unsurprisingly). This comparison serves only to confirm that minimum Bayes-risk decoding is a way to circumvent independence assumptions (here made by a decoder), but only when the trained model *does not* make those particular assumptions. As we showed in §2.6 above, we can also apply minimum Bayes-risk decoding to the approximate marginals computed by loopy belief propagation. When we adapt loopy BP to approximate higher-order nonprojective parsing (§3.4), we will thus use MBR to produce final output trees.

3.4 Higher-Order Nonprojective Parsing

In the foregoing sections, we developed methods for nonprojective inference in edge-factored models. In particular, applying the matrix-tree theorem allowed us to experiment with probabilistic models of nonprojective trees. But what about higher-order constraints, such as the features from chapter 2 that considered grandparents and siblings? Those features increased the degree of the polynomial runtime for projective parsers, though we managed to speed up inference with BP at a small cost in accuracy; unfortunately, exact nonprojective inference with grandparent, sibling, valence, and similar features has been shown to be NP-hard (McDonald and Pereira, 2006; McDonald and Satta, 2007).[7] We can still, however, apply the BP formulation—with one small change: we can no longer use the projective PTREE factor. Once we handle that requirement, we will be able to perform higher-order nonprojective parsing in cubic time.

3.4.1 A nonprojective tree constraint

To use loopy BP for nonprojective parsing, we need to modify the hard PTREE constraint from chapter 2 to allow nonprojective trees as well. As in the projective case, this new TREE factor will run an edge-factored, cubic-time parser to calculate the messages to send back to the link variables L_{ij}. In this case, we will use the nonprojective parsers developed in the first part of this chapter.

In §2.5.4, we derived the messages from the parsing factor to the link variables as:

$$r_{L_{ij}}(\text{true}) = g(i,j) \tag{3.19}$$

$$r_{L_{ij}}(\text{false}) = 1 - u(i,j) \cdot g(i,j) \tag{3.20}$$

[7]For example, if the valence constraint forced each parent to have a single child and TREE, as usual, forced each child to have a single parent, we could solve the Hamilton path problem.

where we defined:

$$g(i,j) = \frac{\partial \log Z}{\partial u(i,j)} = \frac{1}{Z}\frac{\partial Z}{\partial u(i,j)} \tag{3.21}$$

Importantly, the message derivation depended only on the existence of the gradient and not on any property of the parser such as projectivity (or edge-factored-ness). We can derive this gradient for the nonprojective case by combining (3.3) with (3.7):

$$g(i,j) = \left[\mathbf{K}_0^{-1}\right]_{j,j} - \left[\mathbf{K}_0^{-1}\right]_{j,i} \tag{3.22}$$

Using the matrix-tree machinery, the TREE factor can enforce a nonprojective tree constraint in $O(n^3)$ time per iteration during sum-product belief propagation.[8] As we pointed out in chapter 2, this runtime is *added* to that of the other factors in the factor graph. Since, as before, the runtime of the total non-tree factors is cubic, the overall runtime is still $O(n^3)$ per iteration.

What about max-product BP? With PTREE, this is simply a matter of changing semirings in the dynamic program: compare the outside weights from algorithm B.3 to the Viterbi outside weights from algorithm B.4 in appendix B. While it is convenient to compute the messages for TREE in closed form with the inverse Kirchhoff matrix, this approach does not allow such a simple change for max-product. Note also that the greedy Chu-Liu-Edmonds algorithm and its k-best extensions do not store the intermediate quantities of a dynamic programming approach. It would thus appear that computing the max-marginals

[8]A dynamic algorithm could incrementally update the outgoing messages if only a few incoming messages have changed (as in asynchronous BP). In the case of TREE, dynamic matrix inverse allows us to update any row or column (i.e., messages from all parents or children of a given word) and find the new inverse in $O(n^2)$ time (Sherman and Morrison, 1950).

for TREE must take at least $O(n^4)$ time—in other words, we would be forced to compute $O(n^2)$ max-marginals each with a $O(n^2)$ run of Tarjan's MST algorithm.[9]

3.4.2 Penalizing crossing links

Even in languages that all nonprojective links, most dependencies are projective (Kuhlmann and Nivre, 2006; Kuhlmann and Möhl, 2007; Gildea, 2010). We would thus like to enforce a *soft* constraint on crossing links in conjunction with the *hard* TREE constraint. NOCROSS is a family of $O(n^2)$ global constraints. If the parent-to-j link crosses the parent-to-ℓ link, then NOCROSS$_{j\ell}$ fires with a value that depends only on j and ℓ. (If j and ℓ do not each have exactly one parent, NOCROSS$_{j\ell}$ fires with value 0; i.e., it incorporates EXACTLY1$_j$ and EXACTLY1$_\ell$.)[10]

NOCROSS$_{j\ell}$ must sum over assignments to $O(n)$ neighboring variables $\{L_{ij}\}$ and $\{L_{k\ell}\}$. The non-zero summands are assignments where j and ℓ each have exactly one parent. At step 1, $\pi \overset{\text{def}}{=} \prod_i q_{L_{ij}}(\text{false}) \cdot \prod_k q_{L_{k\ell}}(\text{false})$. At step 2, the marginal belief $b(L_{ij} = \text{true})$ sums over the n non-zero assignments containing $i \rightarrow j$. It is $\pi \cdot \bar{q}_{L_{ij}} \cdot \sum_k \bar{q}_{L_{k\ell}} \cdot \text{PAIR}(L_{ij}, L_{k\ell})$, where $\text{PAIR}(L_{ij}, L_{k\ell})$ is $x_{j\ell}$ if $i \rightarrow j$ crosses $k \rightarrow \ell$ and is 1 otherwise. $x_{j\ell}$ is some factor value defined by equation (2.6) to penalize or reward

[9]The procedure we use to compute the LU factorization is not amenable to a change to the $(\max, \times)$ semiring since it uses division. In both sum-product and max-product BP, outgoing messages do not need to be computed for values that the incoming message assigns zero probability. We can thus get an asymptotic speedup if we restrict the number of possible parents for a node by, e.g., a length restriction (as in the Vine Grammar of Eisner and Smith, 2005), or by using a simple model's posteriors to prune.

[10]In effect, we have combined the $O(n^4)$ binary factors $\text{PAIR}(L_{ij}, L_{k\ell})$ into $O(n^2)$ groups, and made them more precise by multiplying in EXACTLYONE constraints (see footnote 12). This will permit $O(n^3)$ total computation if we are willing to sacrifice the ability of the PAIR weights to depend on i and k.

the crossing. Steps 3–4 are just as in $\textsc{Exactly1}_j$.

The question is how to compute $b(L_{ij} = \textsf{true})$ for each i in only $O(1)$ time,[11] so that we can propagate each of the $O(n^2)$ $\textsc{NoCross}_{j\ell}$ in $O(n)$ time. This is why we allowed $x_{j\ell}$ to depend only on j, ℓ. We can rewrite the sum $b(L_{ij} = \textsf{true})$ as

$$\pi \cdot \bar{q}_{L_{ij}} \cdot (x_{j\ell} \cdot \sum_{\text{crossing } k} \bar{q}_{L_{k\ell}} + 1 \cdot \sum_{\text{noncrossing } k} \bar{q}_{L_{k\ell}}) \tag{3.23}$$

To find this in $O(1)$ time, we precompute for each ℓ an array of partial sums $Q_\ell[s,t] \overset{\text{def}}{=} \sum_{s \le k \le t} \bar{q}_{L_{k\ell}}$. Since $Q_\ell[s,t] = Q_\ell[s,t-1] + \bar{q}_{L_{t\ell}}$, we can compute each entry in $O(1)$ time. The total precomputation time over all ℓ, s, t is then $O(n^3)$, with the array Q_ℓ shared across all factors $\textsc{NoCross}_{j'\ell}$. The crossing sum is respectively $Q_\ell[0, i-1] + Q_\ell[j+1, n]$, $Q_\ell[i+1, j-1]$, or 0 according to whether $\ell \in (i,j)$, $\ell \notin [i,j]$, or $\ell = i$. (There are no $\textsc{NoCross}_{j\ell}$ factors with $\ell = j$.) The non-crossing sum is $Q_\ell[0,n]$ minus the crossing sum.

For their features, the $\textsc{NoCross}_{j\ell}$ factors consider the tag and coarse tag attributes of the two child words j and ℓ, separately or jointly.

3.4.3 Experiments

In addition to speeding up projective parsing (§2.9.2), the BP approximation can be used to improve the accuracy of *non*-projective parsing by adding higher-order features. These would be NP-hard to incorporate exactly, and DP cannot be used.

We used sum-product BP with a nonprojective TREE factor to train conditional log-linear parsing models of two highly nonprojective languages, Danish and Dutch, as well as

[11]Symmetrically, we compute $b(L_{k\ell} = \textsf{true})$ for each k.

		Danish	Dutch	English
(a)	TREE+LINK	85.5	87.3	88.6
	+NOCROSS	86.1	88.3	89.1
	+GRAND	86.1	**88.6**	89.4
	+CHILDSEQ	**86.5**	**88.5**	90.1
(b)	Proj. DP	86.0	84.5	**90.2**
	+hill-climbing	86.1	87.6	**90.2**

Table 3.2: (a) Percent unlabeled dependency accuracy for various nonprojective BP parsers (5 iterations only), showing the cumulative contribution of different features. (b) Accuracy for an projective DP parser with all features. For relatively nonprojective languages (Danish and especially Dutch), the exact projective parses can be improved by nonprojective hill-climbing—but in those cases, just running our nonprojective BP is better and faster (asymptotically and empirically). Dutch, the most nonprojective language, shows the most gain from nonprojective parsing. English, even though slightly nonprojective here, fares better with projective parsing and is not improved by hill-climbing.

slightly nonprojective English (§3.1). In all three languages, the first-order nonprojective parser greatly overpredicts the number of crossing links. We thus added NoCross factors, as well as Grand and ChildSeq as before. All of these significantly improve the first-order baseline, though not necessarily cumulatively (Table 3.2).

Finally, Table 3.2 compares loopy BP to a previously proposed "hill-climbing" method for approximate inference in nonprojective parsing (McDonald and Pereira, 2006). Hill-climbing decodes our richest nonprojective model by finding the best *projective* parse under that model—using slow, *higher-order* DP—and then greedily modifies words' parents until the parse score (2.5) stops improving.

BP for nonprojective languages is much faster and more accurate than the hill-climbing method. Also, hill-climbing only produces an (approximate) 1-best parse, but BP also obtains (approximate) marginals of the distribution over *all* parses.

3.4.4 Importance of global hard constraints

Given the BP architecture, do we even need the hard Tree constraint? Or would it suffice for more local hard constraints to negotiate locally via BP?

We investigated this for nonprojective first-order parsing. Table 3.3 shows that global constraints are indeed important, and that it is essential to use Tree during training. At test time, the weaker but still global Exactly1 may suffice (followed by MBR decoding to eliminate cycles), for total time $O(n^2)$. Using only Exactly1 in edge-factored parsing is tantamount to predicting each word's parent independently (Hall, 2007). Note also that

Decoding	Danish	Dutch	English
NOT2	81.8 (76.7)	83.3 (75.0)	87.5 (66.4)
ATMOST1	85.4 (82.2)	**87.3** (86.3)	88.5 (84.6)
EXACTLY1	**85.7** (85.0)	87.0 (86.7)	88.6 (86.0)
+ NO2CYCLE	85.0 **(85.2)**	86.2 (86.7)	88.5 (86.2)
TREE	**85.5 (85.5)**	**87.3 (87.3)**	88.6 **(88.6)**
PTREE	**85.8**	83.9	**88.8**

Table 3.3: After training a nonprojective first-order model with TREE, decoding it with weaker constraints is asymptotically faster (except for NOT2) but usually harmful. (Parenthetical numbers show that the harm is compounded if the weaker constraints are used in training as well; even though this matches training to test conditions, it may suffer more from BP's approximate gradients.) Decoding the TREE model with the even stronger PTREE constraint can actually be helpful for a more projective language. All results use 5 iterations of BP.

for first-order parsing, none of these constraint families except for NOT2 and NO2CYCLE induce any loops in the graph, so approximation error does not explain their performance differences.

Table 3.3 includes NOT2, which takes $O(n^3)$ time, merely to demonstrate how the BP approximation becomes more accurate for training and decoding when we join the simple NOT2 constraints into more global ATMOST1 constraints. This does not change the distribution (2.5), but makes BP enforce stronger local consistency requirements at the factors, relying less on independence assumptions. In general, one can get better BP approximations by replacing a group of factors $F_m(\mathcal{A})$ with their product.[12]

The above experiments concern *gold-standard* accuracy under a given *first-order*, *nonprojective* model. Flipping all three of these parameters for Danish, we confirmed the pattern by instead measuring *search error* under a *higher-order*, *projective* model (PTREE+LINK+GRAND), when PTREE was weakened during decoding. Compared to the MBR parse under that model, the search errors from decoding with weaker hard constraints were 2.2% for NOT2, 2.1% for EXACTLY1, 1.7% for EXACTLY1 + NO2CYCLE, and 0.0% for PTREE.

[12]In the limit, one could replace the product (2.5) with a single all-purpose factor; then BP would be exact—but slow. (In constraint satisfaction, joining constraints similarly makes arc consistency slower but better at eliminating impossible values.)

3.5 Conclusion

We have shown how to carry out exact marginalization under an edge-factored, conditional log-linear model over nonprojective dependency trees. The method has cubic runtime in the length of the sequence and is very fast in practice. It can be used in conditional training of such a model, in minimum Bayes-risk decoding (regardless of how the model is trained), and in training with hidden variables. By incorporating this edge-factored, nonprojective parser into a TREE constraint within BP, we demonstrated results competitive with state-of-the-art dependency parsers for several languages.

Chapter 4

Quasi-Synchronous Models for Adaptation

Many problems in natural language processing operate on several structures simultaneously. These correlated structures may represent different linguistic layers: to infer syntactic structure, we need to know the morphology; to infer semantic roles, we need the syntax. We may also be interested in the correlations between parallel structures from different sources: texts and translations, questions and answers, documents and queries, or articles and abstracts. We may want to align textual narratives with database records or with visual scenes.

In this chapter, we exploit the generality of factor-graph modeling and BP inference to model aligned linguistic structures. In particular, we explore the problem of **adaptation**, where we have an accurate classifier for one kind of structure and want to make predictions about another, related kind of structure. We start simply, by discussing mapping between annotation schemes in the same language (§4.1) but then generalize the model to account for noisy correspondences between dependency trees in different languages (§4.2).

The models in this chapter are based on **quasi-synchronous grammar** (Smith and Eisner, 2006b). Unlike synchronous grammars commonly used in machine translation (Shieber and Schabes, 1990), quasi-synchronous grammars do not require source and target trees to be isomorphic at either the level of derived trees or some underlying derivation tree level. This flexibility supports convenient modeling of a wide range of syntactic **divergences** between languages (§4.5).

The features of the models in this chapter are broadly divided into "monolingual" features, which consider only target-language phenomena, and "bilingual" features, which consider the syntax of source and target and the alignment between them. Given this general framework (§4.3), we present models for adaptation between different dependency styles (§4.4), for unsupervised parser projection between languages (§4.6), and for efficient bilingual parsing to improve dependency accuracy in the target language (§4.7).

The additive runtime effect of adding new factors to BP inference allows significant efficiency gains: the bilingual parsing models do joint inference over two trees and their alignment in $O(m^2n^2)$ time (for sentences of lengths m and n in two languages) instead of at least $O(m^3n^3)$ time for synchronous parsing.

4.1 Parser Adaptation

Consider the problem of learning a dependency parser, which must produce a directed tree whose vertices are the words of a given sentence. There are many differing conventions for representing syntactic relations in dependency trees. Say that we wish to output parses in the Prague style (see figure 4.1) and so have annotated a small **target corpus**—e.g., 100 sentences—with those conventions. A parser trained on those hundred sentences will achieve mediocre dependency accuracy (the proportion of words that attach to their correct parent).

But what if we also had a large number of trees in the CoNLL style (the **source corpus**)? Ideally they should help train our parser. But unfortunately, a parser that learned to produce perfect CoNLL-style trees would, for example, get both links "wrong" when its coordination constructions were evaluated against a Prague-style gold standard.

If it were just a matter of this one construction, the obvious solution would be to write a few rules by hand to transform the large source training corpus into the target style. Suppose, however, that there were many more ways that our corpora differed. Then we would like to *learn a statistical model to transform one style of tree into another.*

We may not possess hand-annotated training data for this tree-to-tree transformation task. That would require the two corpora to annotate some of the *same* sentences in different styles.

But fortunately, we can automatically obtain a noisy form of the necessary paired-tree training data. A parser trained on the source corpus can parse the sentences in our target

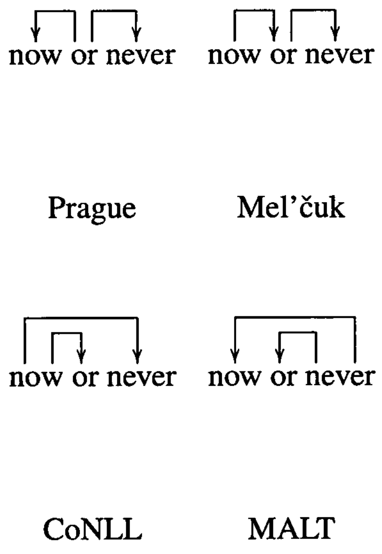


Figure 4.1: Four of the five logically possible schemes for annotating coordination show up in human-produced dependency treebanks. (The other possibility is a reverse Mel'čuk scheme.) These treebanks also differ on other conventions.

corpus, yielding trees (or more generally, probability distributions over trees) in the source style. We will then learn a tree transformation model relating these noisy source trees to our known trees in the target style. This model should enable us to convert the original large source corpus to target style, giving us additional training data in the target style.

4.2 Parser Projection

For many target languages, however, we do not have the luxury of a large parsed "source corpus" in the language, even one in a different style or domain as above. Thus, we may seek other forms of data to augment our small target corpus. One option would be to leverage unannotated text (McClosky et al., 2006; Smith and Eisner, 2007). But we can also try to transfer syntactic information from a parsed source corpus *in another language.* This is an extreme case of out-of-domain data. This leads to the second task of this chapter: *learning a statistical model to transform a syntactic analysis of a sentence in one language into an analysis of its translation.*

Tree transformations are often modeled with *synchronous* grammars. Suppose we are given a sentence W^S in the "source" language and its translation W^T into the "target" language. Their syntactic parses T^S and T^T are presumably not independent, but will tend to have some parallel or at least correlated structure. So we could *jointly* model the parses T^S, T^T and the alignment A between them, with a model of the form $p(T^T, A, T^S \mid W^T, W^S)$.

Such a joint model captures how T^T, A, T^S mutually constrain each other, so that even partial knowledge of some of these three variables can help us to recover the others when training or decoding on bilingual text. This idea underlies a number of recent papers on syntax-based alignment (using T^T and T^S to better recover A), grammar induction from bitext (using A to better recover T^T and T^S), parser projection (using T^S and A to better recover T^T), as well as full joint parsing (Smith and Smith, 2004; Burkett and Klein, 2008).

So far, this is very similar to the monolingual parser adaptation scenario, but there are a few key differences. Since the source and target sentences in the bitext are in different languages, there is no longer a trivial alignment between the words of the source and target trees. Given word alignments, we could simply try to project dependency links in the

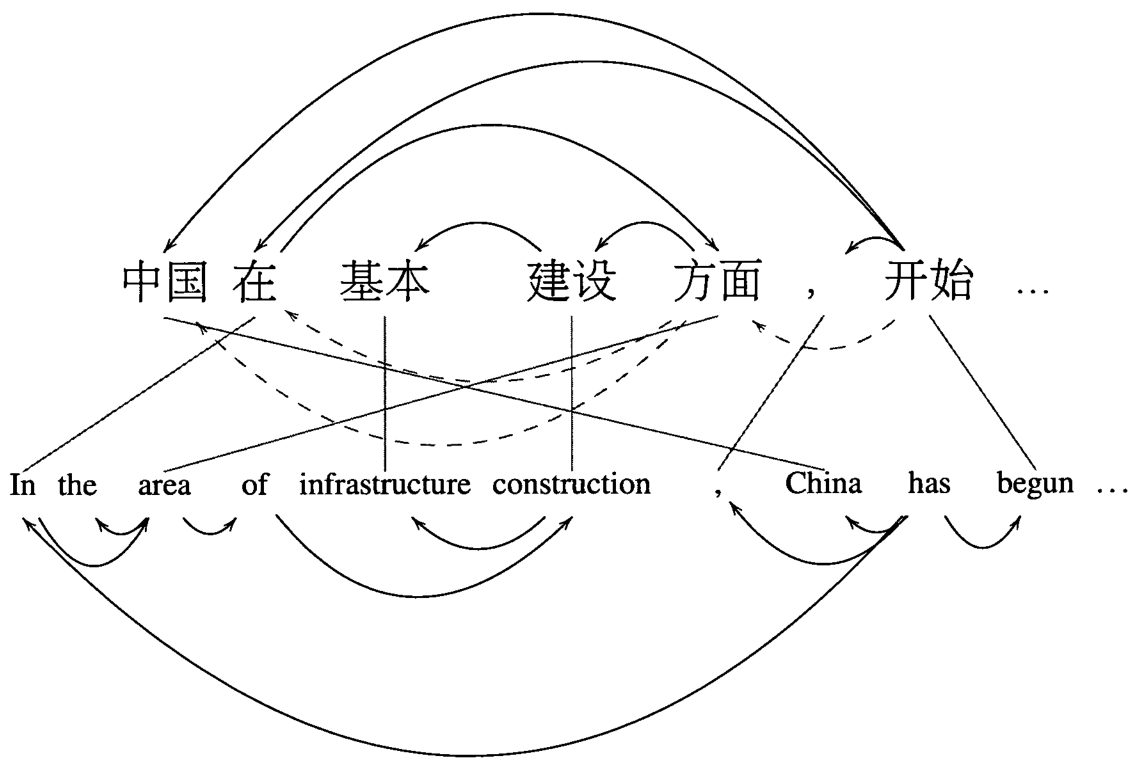

中国 在 基本 建设 方面 , 开始 …
In the area of infrastructure construction , China has begun …

Figure 4.2: With the English tree and alignment provided by a parser and aligner at test time, the Chinese parser finds the correct dependencies (see §4.7). A monolingual Chinese parser's incorrect links are shown with dashed lines.

source tree onto the target text. A link-by-link projection, however, could result in invalid trees on the target side, with cycles or disconnected words. Instead, our models learn the necessary transformations that align and transform a source tree into a target tree by means of quasi-synchronous grammar (QG) features.

Figure 4.2 shows an example of bitext helping disambiguation: in this case, the parser is trained with a large number of English trees but only a small number of Chinese trees and some unannotated Chinese-English sentence pairs. With the help of the English tree and alignment, the parser is able to recover the correct Chinese dependencies using QG features. Incorrect edges from the monolingual parser are shown with dashed lines. (The bilingual parser corrects additional errors in the second half of this sentence, which has been removed to improve legibility.) The parser is able to recover the long-distance dependency from the first Chinese word (*China*) to the last (*begun*), while skipping over the intervening noun phrase that confused the undertrained monolingual parser. Although, due to the auxiliary verb, "China" and "begun" are *siblings* in English and not in direct dependency, the QG features still leverage this indirect projection.

4.3 Quasi-Synchronous Models

Such examples inspire the key novel features of the "quasi-synchronous" models (Smith and Eisner, 2006b) that we will now present: A does not have to be a "well-behaved" syntactic alignment. Any portion of T^T can align to any portion of T^S, or to NULL. Nodes that are syntactically related in T^S do *not* have to translate into nodes that are syntactically related in T^T—although the model score is usually higher if they do.

This property makes our approach especially promising for aligning freely (figure 4.3), or erroneously, translated sentences, and for coping with syntactic **divergences** observed between even closely related languages (Dorr, 1994; Fox, 2002). We can patch together an alignment without accounting for all the details of the translation process.

What should our model of source and target trees look like? In our view, traditional

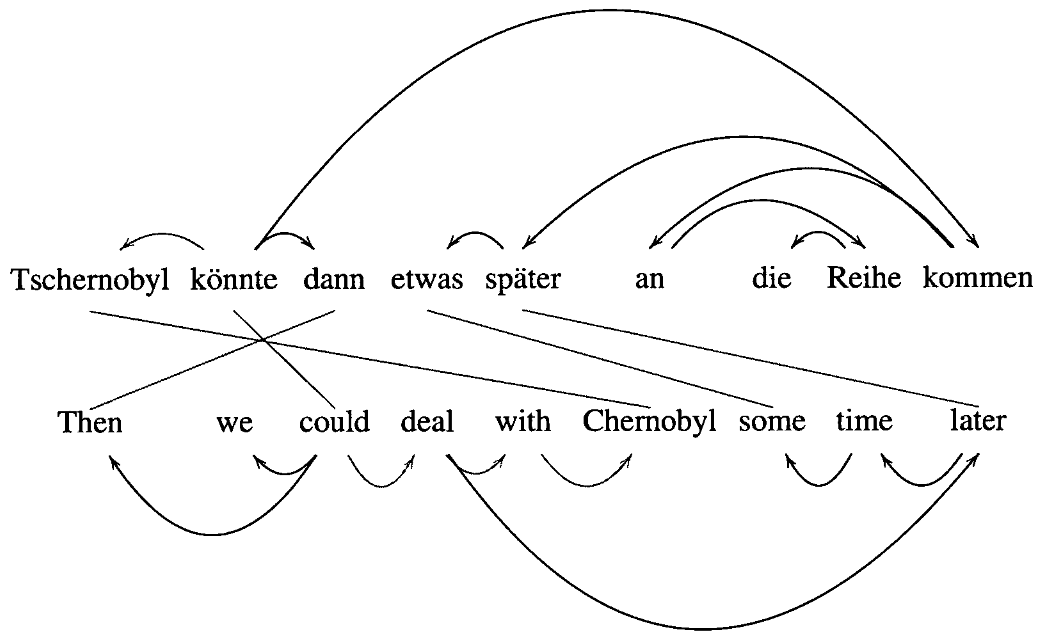

Tschernobyl könnte dann etwas später an die Reihe kommen
Then we could deal with Chernobyl some time later

Figure 4.3: German and English dependency parses and their alignments from our system where German is the target language. *Tschernobyl* depends on *könnte* even though their English analogues are not in a dependency relationship.

German: *Tschernobyl könnte dann etwas später an die Reihe kommen* .

Literally: Chernobyl could then somewhat later on the queue come.

English: *Then we could deal with Chernobyl some time later* .

approaches based on synchronous grammar are problematic both computationally and linguistically. Full inference takes $O(n^6)$ time or worse (depending on the grammar formalism). Yet synchronous models only consider a limited hypothesis space: e.g., parses must be projective, and alignments must decompose according to the recursive parse structure. (For example, two nodes can be aligned only if their respective parents are also aligned.) The synchronous model's probability mass function is also restricted to decompose in this way, so it makes certain conditional independence assumptions; put another way, it can evaluate only certain properties of the triple (T^T, A, T^S).

We instead model (T^T, A, T^S) as an *arbitrary graph* that includes dependency links among the words of each sentence as well as arbitrary alignment links between the words of the two sentences. This permits non-synchronous and many-to-many alignments. The only hard constraint we impose is that the dependency links within each sentence must constitute a valid monolingual parse—a directed (projective or nonprojective) spanning tree.

Given the two sentences W^S, W^T, our probability distribution over possible graphs considers local features of the parses, the alignment, and both jointly. Thus, we learn what local syntactic configurations tend to occur in each language and how they correspond across languages. As a result, we might learn that parses are "mostly synchronous," but that there are some systematic cross-linguistic divergences and some instances of sloppy (non-parallel or inexact) translation. Our model is thus a form of quasi-synchronous grammar (QG) (Smith and Eisner, 2006b). In that paper, QG was applied to word alignment and has since found applications in question answering (Wang et al., 2007), paraphrase detection (Das and Smith, 2009), machine translation (Gimpel and Smith, 2009), and summarization (Woodsend et al., 2010).

The score s of a given tuple of trees, words, and alignment can thus be written as a dot product of weights $\mathbf{w}$ with features $\mathbf{f}$ and $\mathbf{g}$:

$$s(T^T, T^S, A, W^T, W^S) = \sum_i w_i f_i(T^T, W^T) \tag{4.1}$$

$$+ \sum_j w_j g_j(T^T, T^S, A, W^T, W^S) \tag{4.2}$$

$$+ \sum_k w_k h_k(T^S, W^S) \tag{4.3}$$

The features $\mathbf{f}$ look only at target words and dependencies. In the conditional models of §4.4 and §4.7, these features are those of an edge-factored dependency parser (McDonald et al., 2005a). In the generative models of §4.6, $\mathbf{f}$ has the form of a dependency model with valence (Klein and Manning, 2004). All models, for instance, have a feature template that considers the parts of speech of a potential parent-child relation.

In order to benefit from the source language, we also need to include bilingual features $\mathbf{g}$. When scoring a candidate target dependency link from word $x \to y$, these features consider the relationship of their corresponding source words x' and y'. (The correspondences are determined by the alignment A.) For instance, the source tree T^S may contain the link $x' \to y'$, which would cause a feature for *monotonic* projection to fire for the $x \to y$ edge. If, on the other hand, $y' \to x' \in T^S$, a *head-swapping* feature fires. If $x' = y'$, i.e. x and y align to the same word, the *same-word* feature fires. Similar features fire when x' and y'

are in grandparent-grandchild, sibling, c-command, or none-of-the above relationships, or when y aligns to NULL. These alignment classes are called *configurations* (figure 4.4 and Smith and Eisner, 2006b). When training is conditioned on the target words (see §4.4 and §4.7 below), we conjoin these configuration features with the part of speech and coarse part of speech of one or both of the source and target words, i.e. the feature template has from one to four tags.

The source features $\mathbf{h}$ look only at the source words and dependencies. In the sections on monolingual adaptation (§4.4) and unsupervised projection (§4.6), we only use the one-best source tree for each sentence, so these features are ignored. In the experiments with learning bilingual parsers (§4.7), these features are used to score different source trees in a joint model. Even there, however, their weights w_k are *learned* on monolingual source data and held fixed while training and testing on bitext.

In conditional training, the exponentiated scores s are normalized by a constant: $Z = \sum_t \exp[s(T^T, T^S, A, W^T, W^S)]$. For the generative model, the locally normalized generative process is explained in §4.6.3.4.

Previous researchers have written fix-up rules to massage the projected links after the fact and learned a parser from the resulting trees (Hwa et al., 2005). Instead, our models learn the necessary transformations that align and transform a source tree into a target tree. Other researchers have tackled the interesting task of learning parsers from unparsed bitext alone (Kuhn, 2004; Snyder et al., 2009); our methods take advantage of investments in high-resource languages such as English. In work most closely related to this, Ganchev et al. (2009) constrain the posterior distribution over target-language dependencies to align to source dependencies some "reasonable" proportion of the time ($\approx$ 70%, cf. table 4.2 below). This approach performs well but cannot directly learn regular cross-language non-isomorphisms; for instance, some fixup rules for auxiliary verbs need to be introduced. Finally, Huang et al. (2009) use features, somewhat like QG configurations, on the shift-reduce actions in a monolingual, target-language parser.

4.4 Experiments with Monolingual Adaptation

As discussed in §4.2, the adaptation scenario is a special case of parser projection where the word alignments are one-to-one and observed. To test our handling of QG features, we performed experiments in which training saw the correct parse trees in both source and target domains, and the mapping between them was simple and regular. We also performed experiments where the source trees were replaced by the noisy output of a trained parser, making the mapping more complex and harder to learn.

We used the subset of the Penn Treebank from the CoNLL 2007 shared task and converted it to dependency representation while varying two parameters: (1) CoNLL vs. Prague coordination style (figure 4.1), and (2) preposition the head vs. the child of its nominal object.

We trained an edge-factored dependency parser (McDonald et al., 2005a) on "source" domain data that followed one set of dependency conventions. We then trained an edge-

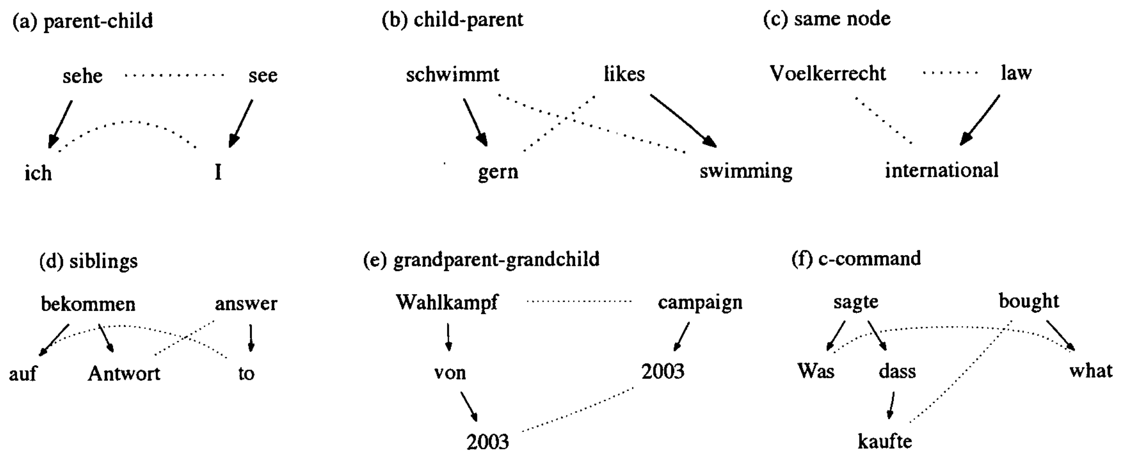


Figure 4.4: When a head h aligned to h' generates a new child a aligned to a' under the QCFG, h' and a' may be related in the source tree as, among other things, (a) parent–child, (b) child–parent, (c) identical nodes, (d) siblings, (e) grandparent–grandchild, (f) c-commander–c-commandee, (g) none of the above. Here German is the source and English is the target. Case (g), not pictured above, can be seen in figure 4.3, in English-German order, where the child-parent pair *Tschernobyl könnte* correspond to the words *Chernobyl* and *could*, respectively. Since *could* dominates *Chernobyl*, they are not in a c-command relationship.

	% Dependency Accuracy on Target											
	CoNLL-PrepHead			CoNLL-PrepChild			Prague-PrepHead			Prague-PrepChild		
Source	0	10	100	0	10	100	0	10	100	0	10	100
Gold CoNLL-PrepHead	100	99.6	99.6	79.5	96.9	97.8	90.5	95.0	98.1	71.0	92.7	95.4
Parse CoNLL-PrepHead	89.5	88.9	89.0	71.4	85.9	87.9	82.5	84.3	87.8	65.2	82.2	86.1
Gold CoNLL-PrepChild	79.5	96.6	97.3	100	99.6	99.6	71.0	91.3	95.5	89.9	94.5	97.9
Parse CoNLL-PrepChild	71.0	84.2	86.8	88.1	87.5	88.0	64.9	80.7	84.9	80.9	83.5	86.1
Gold Prague-PrepHead	90.5	95.5	96.7	71.0	92.0	94.2	100	99.6	99.6	79.6	97.4	98.1
Parse Prague-PrepHead	83.0	87.1	87.4	65.6	84.2	85.9	88.5	88.3	88.0	70.7	86.4	86.8
Gold Prague-PrepChild	71.0	91.6	93.8	89.9	95.6	96.4	79.6	96.0	97.1	100	99.6	99.6
Parse Prague-PrepChild	65.3	81.7	84.6	81.2	84.5	86.1	70.4	83.2	85.3	86.9	86.1	86.8

Table 4.1: Adapting a parser to a new annotation style. We learn to parse in a "target" style (wide column label) given some number (narrow column label) of supervised target-style training sentences. As a font of additional features, *all training and test sentences* have already been augmented with parses in some "source" style (row label): either gold-standard parses (an oracle experiment) or else the output of a parser trained on 18k source trees (more realistic). If we have 0 training sentences, we simply output the source-style parse. But with 10 or 100 target-style training sentences, each off-diagonal block learns to adapt, mostly closing the gap with the diagonal block in the same column. In the diagonal blocks, source and target styles match, and the QG parser degrades performance when acting as a "stacked" parser.

factored parser with QG features on a small amount of "target" domain data. The source parser outputs were produced for all target data, both training and test, so that features for the target parser could refer to them.

In this task, we know what the gold-standard source language parses are for any given text, since we can produce them from the original Penn Treebank. We can thus measure the contribution of adaptation loss alone, and the combined loss of imperfect source-domain parsing with adaptation (table 4.1).

When no target domain trees are available, we simply have the performance of the source domain parser on this out-of-domain data. Training a target-domain parser on as few as 10 sentences shows substantial improvements in accuracy. In the "gold" conditions, where the target parser starts with perfect source trees, accuracy approaches 100%; in the realistic "parse" conditions, where the target-domain parser gets noisy source-domain parses, the improvements are quite significant but approach a lower ceiling imposed by the performance of the source parser. In the diagonal cells, source and target styles match, so training the QG parser amounts to a "stacking" technique (Nivre and McDonald, 2008a; Martins et al., 2008). The small training size and over-regularization of the QG parser mildly hurts in-domain parsing performance.

The adaptation problem in this section is a simple proof of concept of the QG approach; however, more complex and realistic adaptation problems exist. Monolingual adaptation is perhaps most obviously useful when the source parser is a black-box or rule-based system or is trained on unavailable data. One might still want to use such a parser in some new context, which might require new data or a new annotation standard.

We are also interested in scenarios where we want to avoid expensive retraining on large reannotated treebanks. We would like a linguist to be able to annotate a few trees according to a hypothesized theory and then quickly use QG adaptation to get a parser for that theory. One example would be adapting a dependency parser to produce constituency parses. The Hindi-Urdu Treebank Project, for example, has developed an automated procedure to convert the dependency annotations of linguists in India into a constituency style more convenient for many analyses Xia et al. (2009). We have concentrated here on adapting between two dependency parse styles, in order to line up with the cross-lingual tasks to which we now turn.

4.5 Cross-Lingual Projection: Background

As in the adaptation scenario above, many syntactic structures can be transferred from one language to another. In this section, we evaluate the extent of this direct projection on a small hand-annotated corpus. In §4.6, we will use a QG generative model to learn dependency parsers from bitext when there are no annotations in the target language. Finally, in §4.7, we show how QG features can augment a target-language parser trained on a small set of labeled trees.

For syntactic annotation projection to work at all, we must hypothesize, or observe, that at least some syntactic structures are preserved in translation. Hwa et al. (2005) have called this intuition the **Direct Correspondence Assumption** (DCA). With the notation changed to match ours, they formulate the DCA as follows:

> Given a pair of sentences W^T and W^S that are translations of each other with syntactic structure T^T and T^S, if nodes x' and y' of T^S are aligned with nodes x and y of T^T, respectively, and if syntactic relationship $R(x', y')$ holds in T^S, then $R(x, y)$ holds in T^T.

The validity of this assumption clearly depends on the node-to-node alignment of the two trees. We again work in a dependency framework, where syntactic nodes are simply lexical items. This allows us to use existing work on word alignment.

Hwa et al. (2005) tested the DCA under idealized conditions by obtaining hand-corrected dependency parse trees of a few hundred sentences of Spanish-English and Chinese-English bitext. They also used human-produced word alignments. Since their word alignments could be many-to-many, they introduced a heuristic Direct Projection Algorithm (DPA) for resolving them into component dependency relations. It should be noted that this process introduced empty words into the projected target language tree and left words that are unaligned to English detached from the tree; as a result, they measured performance in dependency F-score rather than accuracy. With manual English parses and word alignments, this DPA achieved 36.8% F-score in Spanish and 38.1% in Chinese. With Collins-model English parses and GIZA++ word alignments, F-score was 33.9% for Spanish and 26.3% for Chinese. Compare this to the Spanish attach-left baseline of 31.0% and the Chinese attach-right baselines of 35.9%. These discouragingly low numbers led them to write language-specific transformation rules to fix up the projected trees. After these rules were applied to the projections of automatic English parses, F-score was 65.7% for

Corpus	Prec.[%]	Rec.[%]
Spanish	64.3	28.4
(no punc.)	72.0	30.8
Chinese	65.1	11.1
(no punc.)	68.2	11.5

Table 4.2: Precision and recall of direct dependency projection via one-to-one links alone.

English and 52.4% for Chinese.

While these F-scores were low, it is useful to look at a subset of the alignment: dependencies projected across one-to-one alignments before the heuristic fix-ups had a much higher precision, if lower recall, than Hwa et al.'s final results. Using Hwa et al.'s data, we calculated that the precision of projection to Spanish and Chinese via these one-to-one links was $\approx$ 65% (table 4.2). There is clearly more information in these direct links than one would think from the F-scores. To exploit this information, however, we need to overcome the problems of (1) learning from partial trees, when not all target words are attached, and (2) learning in the presence of the still considerable noise in the projected one-to-one dependencies—e.g., at least 28% error for Spanish non-punctuation dependencies.

What does this noise consist of? Some errors reflect fairly arbitrary annotation conventions in treebanks, e.g. should the auxiliary verb govern the main verb or vice versa. (Examples like this remind us that the projection problem contains the adaptation problem above.) Other errors arise from divergences in the complements required of certain head words. In the German-English translation pair, with co-indexed words aligned,

$$\text{[an [den Libanon}_1\text{]] denken}_2 \leftrightarrow \text{remember}_2\ \text{Lebanon}_1$$

we would prefer that the preposition *an* attach to *denken*, even though the preposition's object *Libanon* aligns to a direct child of *remember*. In other words, we would like the grandparent-parent-child chain of *denken* $\rightarrow$ *an* $\rightarrow$ *Libanon* to align to the parent-child pair of *remember* $\rightarrow$ *Lebanon*. Finally, naturally occurring bitexts contain some number of free or erroneous translations. Machine translation researchers often seek to strike these examples from their training corpora; "free" translations are not usually welcome from an MT system.

4.6 Unsupervised Cross-Lingual Projection

First, we consider the problem of parser projection when there are zero target-language trees available. As in much other work on unsupervised parsing, we try to learn a generative model that can predict target-language sentences. Our novel contribution is to *condition the probabilities of the generative actions* on the dependency parse of a source-language translation. Thus, our generative model is a quasi-synchronous grammar, exactly as in (Smith and Eisner, 2006b).[1]

When training on target sentences W^T, therefore, we tune the model parameters to maximize not $\sum_t p(T^T, W^T)$ as in ordinary EM for monolingual CFGs, but rather

[1]Our task here is new; they used it for alignment.

$\sum_t p(T^T, W^T, A \mid T^S, W^S)$. We hope that this *conditional EM* training will drive the model to posit appropriate syntactic relationships in the latent variable T^T, because—thanks to the structure of the QG model—that is the easiest way for it to exploit the extra information in T^S, W^S to help predict W^T.

The contrastive estimation of Smith and Eisner (2005b) also used a form of conditional EM, with similar motivation. They suggested that EM grammar induction, which learns to predict W^T, unfortunately learns mostly to predict lexical topic or other properties of the training sentences that do not strongly require syntactic latent variables. To focus EM on modeling the *syntactic* relationships, they conditioned the prediction of W^T on almost complete knowledge of the lexical items. Similarly, we condition on W^S, a translation of W^T. Furthermore, our QG model structure makes it easy for EM to learn to exploit the (explicitly represented) syntactic properties of that translation when predicting W^T.

At test time, T^S, W^S are not made available, so we just use the trained model to find $\text{argmax}_t\, p(T^T \mid W^T)$, backing off from the conditioning on T^S, W^S and summing over A.

Below, we present the specific generative model (§4.6.1) and some details of training (§4.6.2). We will then compare three approaches (§4.6.3):

§4.6.3.2 a straight EM baseline (which does not condition on T^S, W^S at all)

§4.6.3.3 a "hard" projection baseline (which naively projects T^S, W^S to derive direct supervision in the target language)

§4.6.3.4 our conditional EM approach above (which makes T^S, W^S available to the learner for "soft" indirect supervision via QG)

4.6.1 Generative Models

Our base models of target-language syntax are generative dependency models that have achieved state-of-the art results in unsupervised dependency structure induction. The simplest version, called Dependency Model with Valence (DMV), has been used in isolation and in combination with other models (Klein and Manning, 2004; Smith and Eisner, 2006c). The DMV generates the right children, and then independently the left children, for each node in the dependency tree. Nodes correspond to words, which are represented by their part-of-speech tags. At each step of generation, the DMV stochastically chooses whether to stop generating, conditioned on the currently generating head; whether it is generating to the right or left; and whether it has yet generated any children on that side. If it chooses to continue, it then stochastically generates the tag of a new child, conditioned on the head. The parameters of the model are thus of the form

$$p(\mathit{stop} \mid \mathit{head}, \mathit{dir}, \mathit{adj}) \tag{4.4}$$

$$p(\mathit{child} \mid \mathit{head}, \mathit{dir}) \tag{4.5}$$

where head and child are part-of-speech tags, $\mathit{dir} \in \{\text{left}, \text{right}\}$, and $\mathit{adj}, \mathit{stop} \in \{\text{true}, \text{false}\}$. ROOT is stipulated to generate a single right child.

Bilingual configurations that condition on T^S, W^S (§4.3) are incorporated into the generative process as in Smith and Eisner (2006b). When the model is generating a new child for word x, aligned to x', it first chooses a configuration and then chooses a source word y' in that configuration. The child y is then generated, conditioned on its parent x, most recent sibling a, and its source analogue y'.

4.6.2 Details of EM Training

As in previous work on grammar induction, we learn the DMV from part-of-speech-tagged target-language text. We use expectation maximization (EM) to maximize the likelihood of the data. Since the likelihood function is non-convex in the unsupervised case, our choice of initial parameters can have a significant effect on the outcome. Although we could also try many random starting points, the initializer in Klein and Manning (2004) performs quite well: it sets the probability of two tags' being in a dependency relationship to their co-occurrence count in the training corpus, weighted by the distance between them at each co-occurrence.

The base dependency parser generates the right dependents of a head separately from the left dependents, which allows $O(n^3)$ dynamic programming for an n-word target sentence. Since the QG annotates nonterminals of the grammar with single nodes of T^S, and we consider two nodes of T^S when evaluating the above dependency configurations, QG parsing runs in $O(n^3m^2)$ for an m-word source sentence. If, however, we restrict candidate senses for a target child c to come from links in an IBM Model 4 Viterbi alignment, we achieve $O(n^3k^2)$, where k is the maximum number of possible words aligned to a given target language word. In practice, $k \ll m$, and parsing is not appreciably slower than in the monolingual setting.

If all configurations were equiprobable, the source sentence would provide no information to the target. In our QG experiments, therefore, we started with a bias towards direct parent–child links and a very small probability for breakages of locality. The values of other configuration parameters seem, experimentally, less important for insuring accurate learning.

4.6.3 Experiments

Our experiments compare learning on target language text to learning on parallel text. In the latter case, we compare learning from high-precision one-to-one alignments alone, to learning from all alignments using a QG.

4.6.3.1 Corpora

Our development and test data were drawn from the German TIGER and Spanish Cast3LB treebanks as converted to projective dependencies for the CoNLL 2007 Shared Task (Brants et al., 2002; Civit Torruella and Martí Antonín, 2002). We made one change to the annotation conventions in German: in the dependencies provided, words in a noun phrase governed by a preposition were all attached to that preposition. This meant that in the phrase *das Kind* ("the child") in, say, subject position, *das* was the child of *Kind*; but, in *für das Kind* ("for the child"), *das* was the child of *für*. This seems to be a strange choice in converting from the TIGER constituency format, which does in fact annotate NPs inside

PPs; we have standardized prepositions to govern only the head of the noun phrase. We did *not* change any other annotation conventions to make them more like English. In the Spanish treebank, for instance, control verbs are the children of their verbal complements: in *quiero decir* ("I want to say"="I mean"), *quiero* is the child of *decir*. In German coordinations, the coordinands all attach to the first, but in English, they all attach to the last. These particular divergences in annotation style hurt all of our models equally (since none of them have access to labeled trees). These annotation divergences are one motivation for experiments below that include some target trees.

Our training data were subsets of the 2006 Statistical Machine Translation Workshop Shared Task, in particular from the German-English and Spanish-English Europarl parallel corpora (Koehn, 2002). The Shared Task provided pre-built automatic GIZA++ word alignments, which we used to facilitate replicability. Since these word alignments do not contain posterior probabilities or null links, nor do they distinguish which links are in the IBM Model intersection, we treated all links as equally likely when learning the QG. Target language words unaligned to any source language words were the only nodes allowed to align to NULL in QG derivations.

We parsed the English side of the bitext with the projective dependency parser described by McDonald et al. (2005a) trained on the Penn Treebank §§2–20. Much previous work on unsupervised grammar induction has used gold-standard part-of-speech tags (Smith and Eisner, 2006c; Klein and Manning, 2004, 2002). While there are no gold-standard tags for the Europarl bitext, we did train a conditional Markov model tagger on a few thousand

	Dependency accuracy [%]	
Baselines	German	Spanish
Modify prev.	18.2	28.5
Modify next	27.5	21.4
EM	30.2	25.6
Hard proj.	66.2	59.1
Hard proj. w/EM	58.6	53.0
QG w/EM	**68.5**	**64.8**

Table 4.3: Test accuracy with unsupervised training methods

tagged sentences. This is the only supervised data we used in the target. We created versions of each training corpus with the first thousand, ten thousand, and hundred thousand sentence pairs, each a prefix of the next. Since the target-language-only baseline converged much more slowly, we used a version of the corpora with sentences 15 target words or fewer.

4.6.3.2 Fully Unsupervised EM

Using the target side of the bitext as training data, we initialized our model parameters as described in §4.6.2 and ran monolingual EM. We checked convergence on a development set and measured unlabeled dependency accuracy on held-out test data. We compare performance to simple attach-right and attach left baselines (Table 4.3). For mostly head-final German, the "modify next" baseline is better; for mostly head-initial Spanish, "modify previous" wins. Even after several hundred iterations, performance was slightly, but not significantly better than the baseline for German. EM training did not beat the baseline for Spanish.[2]

4.6.3.3 Hard Projection, Semi-Supervised EM

The simplest approach to using the high-precision one-to-one word alignments is labeled "hard projection" in the table. We filtered the training corpus to find sentences where enough links were projected to completely determine a target language tree. Of course, we needed to filter more than 1000 sentences of bitext to output 1000 training sentences in this way. We simply perform supervised training with this subset, which is still quite noisy (§4.5), and performance quickly plateaus. Still, this method substantially improves over the baselines and unsupervised EM.

Restricting ourselves to fully projected trees seems a waste of information. We can also simply take all one-to-one projected links, impute expected counts for the remaining dependencies with EM, and update our models iteratively. This approach ("hard projection with EM"), however, performed worse than using only the fully projected trees. In fact, only the first iteration of EM with this method made any improvement; afterwards, EM

[2]While these results are worse than those obtained previously for this model, the experiments in Klein and Manning (2004) only used sentences of 10 words or fewer, without punctuation, and with gold-standard tags. Punctuation in particular seems to trip up the initializer: since a sentence-final period appears in most sentences, EM often decides to make it the head.

degraded accuracy further from the numbers in Table 4.3.

4.6.3.4 Soft Projection: QG & Conditional EM

The quasi-synchronous model used all of the alignments in re-estimating its parameters and performed significantly better than hard projection. Unlike EM on the target language alone, the QG's performance does not depend on a clever initializer for initial model weights—all parameters of the generative model except for the QG configuration features were initialized to zero. Setting the prior to prefer direct correspondence provides the necessary bias to initialize learning.[3]

Error analysis showed that certain types of dependencies eluded the QG's ability to learn from bitext. The Spanish treebank treats some verbal complements as the heads of main verbs and auxiliary verbs as the children of participles; the QG, following the English, learned the opposite dependency direction. Spanish treebank conventions for punctuation were also a common source of errors. In both German and Spanish, coordinations (a common bugbear for dependency grammars) were often mishandled: both treebanks attach the later coordinands and any conjunctions to the first coordinand; the reverse is true in English. Finally, in both German and Spanish, preposition attachments often led to errors, which is not surprising given the unlexicalized target-language grammars. Rather than trying to adjudicate which dependencies are "mere" annotation conventions, it would be useful to test learned dependency models on some extrinsic task such as relation extraction or machine

[3]The learner could also gradually dampen the bilingual features $\mathbf{g}$ as it became more confident in monolingual $\mathbf{f}$. Annealing has been successful in other grammar induction settings (Smith and Eisner, 2006c).

translation.

4.7 Learning Bilingual Parsers

We now consider the problem of parser projection when some target language trees are available. As in the monolingual adaptation case (§4.4), we train a conditional model (*not* a generative DMV plus QG alignments as in the previous section) of the alignment and source and target trees given the source and target sentences, using monolingual and bilingual features. Unlike the setup there, however, we now represent the source as well as the target tree with random variables, so that we can perform joint inference of both of them. There is a final important difference: in the monolingual case, the alignment between identical source and target word strings was trivial; here, we need to model alignments as well. Just as the factor-graph model of syntax presented in §2.2.2 mapped dependency trees to variable assignments, we model a traditional word alignment with variables in a factor graph.

4.7.1 Graphical models of word alignment

In many generative alignment models, the source words are observed and generate the target words. The alignments are inferred from the identity of the generated words. Supervised training of alignment models, in contrast, usually condition on the source and target words and infer the alignments alone. In this section, we formulate this latter class of models as a factor graph.

4.7.1.1 Variables

Of the several ways we might represent an alignment of two sentences of lengths m (source) and n (target), we choose the most straightforward: a set of Boolean variables that record which words correspond. To avoid confusion, note that there are no variables here for the event of "aligning to null", which plays a large part in, for instance, the IBM translation models. To test for a null alignment, one can query whether the fertility variable (see below) for a word has the value 0.

The variables in our alignment model are thus:

Word alignments The $O(mn)$ Boolean variables $\{A_{ij} : 1 \leq i \leq m, 1 \leq j \leq n\}$ correspond to possible alignments between source and target words. Note that for aligned sentence pairs, the ROOT nodes of their parse trees are assumed to align, and we may add an always-true variable A_{00} as necessary.

Fertility The $O(m)$ variables $\{F_i^S : 1 \leq i \leq m\}$ and $O(n)$ variables $\{F_j^T : 1 \leq j \leq n\}$ correspond to the number of target words aligned to each source word and vice versa. In other words, they indicate the "fertility" of source and target words. The arity of these variables depends on the number of bins into which we want to divide the fertility. In experiments below, $F_i \in \{0, 1, 2, 3+\}$.

We could also introduce variable for alternative representations that have appeared in the literature, including aligning contiguous spans—a natural approach when aligning constituency trees—and the generalized hidden "cepts" proposed in the original IBM translation papers (Brown et al., 1990). If an alignment is constrained to have a small arity in one or both directions, one could efficiently replace the Boolean variables with multinomials. In that case, the fertility constraint in one direction could be simplified to a unary factor attached to multinomial alignment variables.

4.7.1.2 Alignment constraints

ALIGN A family of $O(mn)$ unary soft constraints. ALIGN$_{ij}$ fires if source word i and target word j are aligned.

SOURCEFERT A family of $O(m)$ soft constraints, each of which is a linear chain of length $O(n)$.

TARGETFERT A family of $O(n)$ soft constraints, each of which is a linear chain of length $O(m)$.

Further constraints which might be included in an alignment model are:

ALIGNHMM Either of two global constraints that enforce an HMM alignment from source to target or vice versa (Vogel et al., 1996).

ALIGNITG A global hard constraint that with degree $O(m^2n^2)$ and runtime $O(m^3n^3)$ that enforces an ITG alignment constraint (Burkett et al., 2010).

Matching methods are also popular for discriminative alignment models. While enforcing the constraint during marginal inference that an alignment be a matching is a $\#P$-complete problem, if the bipartite graph of possible alignments is Pfaffian (a superset of planar), we can find the weighted sum of all matchings in $O(n^3)$ time by the determinant of the Pfaffian matrix of the graph. We could ensure that the alignment graph is planar if and only if it had no minors $K_{3,3}$. Restricting the number of possible alignment candidates to two per word (on one side), would be overly restrictive in most cases. We will not, therefore, employ such global matching constraints in this chapter.

4.7.2 Graphical models of parallel trees

To this alignment model, we now add variables and factors to represent source and target trees.

4.7.2.1 Variables

In section 2.2, we defined dependency parsing as a search for optimal assignment on link variables L_{ij}. In a similar vein, we define bilingual dependency parsing of a sentence pair with m source words and n target words as an assignment to the following variables:

Source links The $O(m^2)$ Boolean variables $\{L^S_{ij} : 0 \leq i \leq m, 1 \leq j \leq m, i \neq j\}$ correspond to possible links in the dependency parse of the source sentence.

Target links The $O(n^2)$ Boolean variables $\{L^T_{ij} : 0 \leq i \leq n, 1 \leq j \leq n, i \neq j\}$ correspond to possible links in the dependency parse of the target sentence.

For various training and testing scenarios, some of the above variables may be hidden or observed.

An assignment to the above variables allows us to read off dependency trees for both sides of a bitext and an alignment between them. In order to support other constraints in the model, we introduce auxiliary variables:

Grandparents The $O(n^2)$ Boolean variables $\{G^T_{ij} : 0 \leq i \leq n, 1 \leq j \leq n, i \neq j\}$ correspond to possible grandparent relations between pairs of words in the target sentence; $O(m^2)$ Boolean variables may be introduced for the source sentence.

Siblings The $O(n^2)$ Boolean variables $\{S^T_{ij} : 1 \leq i \leq n, 1 \leq j \leq n, i \neq j\}$ correspond to possible sibling relations between pairs of words in the target sentence; $O(m^2)$ Boolean variables may be introduced for the source sentence.

Our graphical modeling framework is flexible enough to accommodate other choices in linguistic representation. Trees on one or both sides of the bitext could be represented by $O(n^2)$ constituency variables; to align two constituency trees, we would, before pruning, need $O(m^2n^2)$ constituent alignment variables.

4.7.2.2 Monolingual constraints

The model regulates language-internal behavior using the hard (§2.2.4) and soft (§2.2.5) constraints described previously for monolingual parsing. These constraints may differ for each side of the bitext: the source language might be projective, and use a PTREE hard constraint to prohibit crossing edges, while the target language might be nonprojective and use TREE. Features on the target and source trees correspond to the first and third terms, respectively, in the joint scoring function (4.1).

The source or target models might also include higher-order constraints outlined in §2.2.5, such as GRAND or SIB. But even without those constraints, we might still want factors that simply *indicate* whether three source words, for instance, are in a grandparent-parent-child relationship. As we saw in the last chapter, divergent constructions might lead to siblings in one language aligning to parent and child in the other. These indicator variables, such as G_{ij} and S_{ij} must be backed by machinery that sums over all the possible intermediate parent nodes.

4.7.2.3 Quasi-synchronous constraints

Quasi-synchronous factors express soft constraints on the relationship of source and target trees—i.e., the second term in the joint scoring function (4.1). These constraints consider pairs of alignment variables and at most two other variables that characterize the syntactic relationships of the aligned words.

MONO fires if the parent and child in a target dependency align to the parent and child in a source dependency, respectively.

SWAP fires if the parent and child in a target dependency align to the child and parent in a source dependency, respectively.

SDOUBLE fires if the parent and child word in a source dependency align to the same target word.

TDOUBLE fires if the parent and child word in a target dependency align to the same source word.

SGRAND fires if the parent and child in a target dependency align to the grandparent and grandchild in the source tree.

SSIB fires if the parent and child in a target dependency align to two siblings in the source tree.

TGRAND fires if the parent and child in a source dependency align to the grandparent and grandchild in the target tree.

TSIB fires if the parent and child in a source dependency align to two siblings in the target tree.

4.7.2.4 A synchronous hard constraint

Although, as we have discussed above, a constraint on synchronous trees seems to restrictive for many translation alignment problems, it is interesting to note that we can easily implement a hard, synchronous grammar constraint on the alignment and dependency variables. In synchronous derivations, we must ensure that if the children in a source and a target dependency link are aligned, their parents are aligned also. The converse, however, is not required: the child of an aligned target word need not itself be aligned.

Let m and k indicate target words and m' and k' indicate source words. We attach the factor $\textsc{Synch}_{m',k',m,k}$ to the variables $A_{m'm}$, $A_{k'k}$, $L_{m'k'}$, and L_{mk}. We must ensure that

$$L_{m'k'} \wedge A_{k'k} \wedge L_{mk} \Rightarrow A_{m'm}$$

$$L_{m'k'} \wedge A_{k'k} \wedge A_{m'm} \Rightarrow L_{mk}$$

$$L_{mk} \wedge A_{k'k} \wedge A_{m'm} \Rightarrow L_{m'k'}$$

Each of the SYNCH constraints is thus a quaternary factor whose potentials are 0 only when one of following is true

$$L_{m'k'} \wedge A_{k'k} \wedge L_{mk} \wedge \neg A_{m'm}$$

$$L_{m'k'} \wedge A_{k'k} \wedge A_{m'm} \wedge \neg L_{mk}$$

$$L_{mk} \wedge A_{k'k} \wedge A_{m'm} \wedge \neg L_{m'k'}$$

and 1 otherwise.

Note that this family of constraints is more relaxed than parsing both sentences with a commonly-used synchronous grammar such as an ITG. For instance, SYNCH does not require that either the source or target tree be projective, and it allows arbitrary scrambling of the children of each head.

4.7.2.5 Pruning the search space

In previous sections (§§4.4, 4.6), we considered models of target syntax conditioned on a single source tree. Models of aligned trees that follow the recipe in this section can be more general, at the cost of runtimes that, while polynomial, may be impractical. If we are interested in inference over target trees, one option is to fall back to a single source tree at both training and test time. In addition to reducing the quadratic number of L^S variables to a linear number of dependency links, we see even bigger savings with higher-order structures such as grandparents and siblings—a reduction from $O(n^3)$ to $O(n)$ factors necessary to infer these relationships. Relations such as c-command are also much more tractable to compute, on the source side only, of course.

Pegging a single source tree, however, leaves us open to errors in a monolingual source parser, which may be exacerbated if the bitext we wish to parse is outside of the source-parser's training distribution. Cues from the target structure ought to be able to disambiguate some structures on the source side. To strike a happy medium between the full source model and a single parse, we first run an edge-factored source parser by itself to calculate the posterior probability of the L^S variables. We retain only the variables for each child with the top K posteriors, pegging the rest to false. On monolingual source development data, setting $K = 20$ achieves 99% recall of correct edges, and $K = 10$ gets 98% of the correct links.[4]

[4]Alternate pruning schemes could select links whose posteriors were within a multiplicative or additive factor of the link with the highest posteriors. These schemes did not appear to have significantly different tradeoffs on the WSJ English data. Pruning dependencies before full inference using a simpler model appears in, e.g., Martins et al. (2009a). Their simpler model was a parent-prediction model; our edge-factored parser

On both the source and target sides, there are a cubic number of factors required to determine whether a pair of words are in a grandparent or sibling relationship and a quadratic number of variables to indicate those relationships. In other work, we have investigated methods to reduce by orders of magnitude the number of second-order factors required to improve a first-order parser (Riedel and Smith, 2010; Riedel et al., 2010). The goal here is not to add factors solely to pick the best monolingual parse or to minimize the divergence between the pruned model are the full model. Rather, our goal is to add grandparent and sibling variables that are likely to be true, and avoid adding variables that are likely to be false. When constructing the graphical model for the source or target side, therefore, we only add GRAND or SIB factors between to L variables when the sum of those variables' log odds is greater than zero. Note that this pruning decision should be made once for both the denominator and the numerator of our objective function. Otherwise, the grandparent relations among links in the true target tree will be weighted too highly.

Finally, we restrict the set of alignment variables A that can be true to those that appear in the union alignment of a base word-alignment model (Liang et al., 2006). We also add the intersection alignments of the base model as additional features for the unary ALIGN factors.

was efficient enough to avoid training a separate model just for pruning.

4.7.3 Bilingual parsing setup

For these experiments, we used the LDC's English-Chinese Parallel Treebank (ECTB). Since manual word alignments also exist for a part of this corpus, we were able to measure the loss in accuracy (if any) from the use of our English parser and word aligner. The source-language English dependency parser was trained on the Wall Street Journal, where it achieved 91% dependency accuracy on development data; however, it was only 80.3% accurate when applied to our task, the English side of the ECTB.[5]

Both the monolingual target-language parser and the projected parsers are trained to optimize conditional likelihood of the target trees T^T with twenty iterations of stochastic gradient ascent. At both training and test time, BP ran for 40 iterations or until convergence on each example. We trained all models on differing amounts of bitext from the ECTB. The target-language baselines only looked at the target side of the bitext, but the bilingual-parsing models saw observed target trees T^T at training time. Except in the experiments in §4.7.6 to tease out the effect of gold-standard alignments source trees, neither alignments nor source trees were observed at training (or test) time. The parameters of the source parser were never changed during bitext training; only the alignment, bilingual, and target-language parameters were adjusted.

[5]It would be useful to explore whether the techniques of §4.4 above could be used to improve English accuracy by domain adaptation. Furthermore, a model with QG features trained to perform well on Chinese should not suffer in principle from an inaccurate, but consistent, English parser, but the results in figure 4.7 indicate a significant benefit can be had from better English parsing or from joint Chinese-English inference.

4.7.4 Importance of learning non-isomorphic mappings

Figure 4.5 plots the target-language unlabeled dependency accuracy on held-out bitext. The different training and testing conditions are:

Target only training merely trains a monolingual parser on the target side of the bitext. Due to the small size of the training data, we use LINK and VALENCE factors.

Synchronous parsing takes the monolingually trained source and target models and applies the synchronous hard constraint SYNCH (§4.7.2.4). All alignment links permitted by the baseline alignment model receive equal log-potential values of 0. This approach is analogous to Smith and Smith (2004) and Burkett and Klein (2008), who used slightly different synchronous grammars.

QG-mono parsing also combines monolingually trained source and target models, but with a *soft* MONO constraint with log-potential 1. As in the synchronous parsing case, all alignment links are equally likely. The setup expresses a soft preference for monotonic dependency alignments.

QG-mono training is the first of the bilingually trained models. In addition to training the alignment and target features on bitext, this setup includes only the MONO QG factor. This approach is comparable to Burkett et al. (2010), who include a trainable factor that rewards monotonic syntactic alignments.

QG training trains a model on bitext with all QG factors from §4.7.2.3.

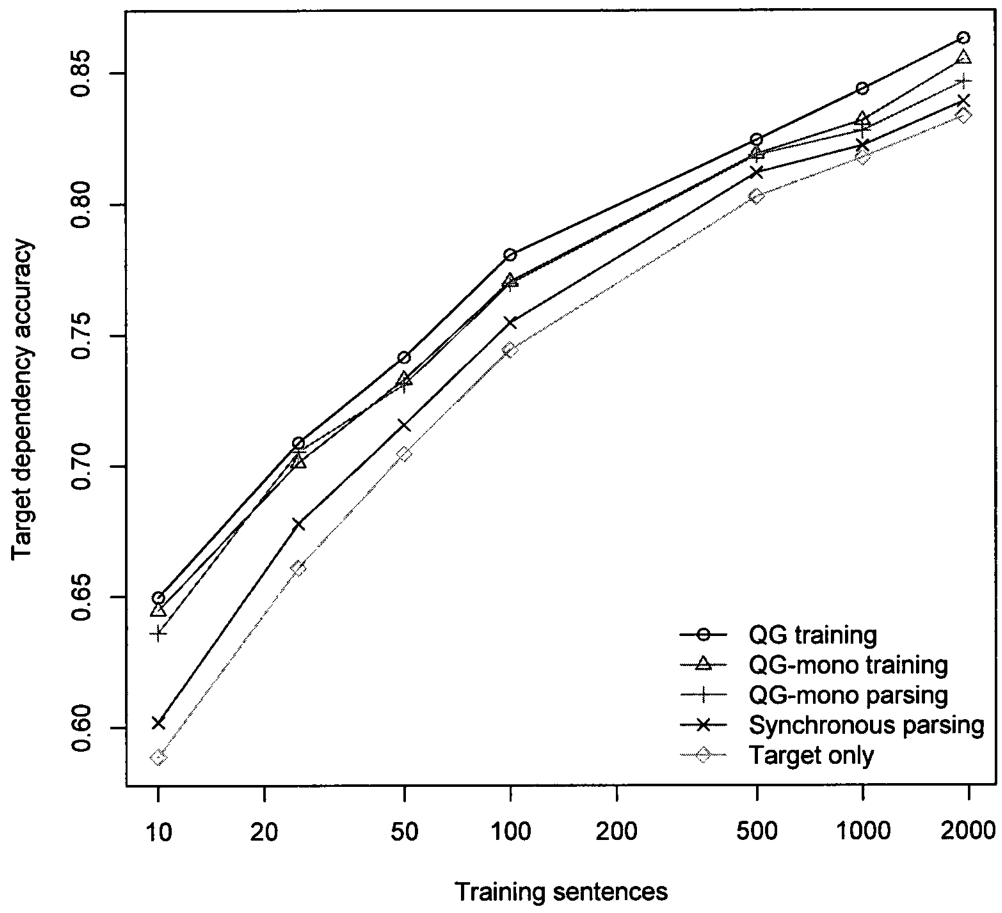


Figure 4.5: Bitext parsing with quasi-synchronous features. Bilingual parsing after QG training is equivalent to having twice as much data in the target language.

The QG-based methods consistently outperform the baseline target-only and synchronous methods. Full QG training outperforms the QG-mono conditions, with increasing margins (around 1% absolute) as the amount of training data with which to tune its parameters increases. Across different training sizes, QG training produces a parser as accurate as one trained on twice the amount of monolingual data.

4.7.5 Summing out source trees

The number of QG factors scales with the number of variables needed to represent a distribution over source trees T^S. If we have a pipelined system, where we use a monolingual source-language parser to get a single T^S, then bitext parsing speeds up asymptotically and empirically. How useful is it to do joint inference over T^T, A, and T^S simultaneously? Figure 4.6 shows that for QG models trained on small amounts of bitext, joint inference get only an insignificant gain (around 0.2% absolute) but when the QG model is trained on 1000 sentences or more, the gains are more substantial (around 1%). For comparison, we show the effect of using the true source trees. In all cases, training conditions matched test—e.g., we used the 1-best source parses in training the model that decoded with 1-best source parses.

The empirical differences in inference time are clear from table 4.4, which compares average number of seconds per sentence on development data. Target-only inference with the higher-order VALENCE factors is already an order of magnitude slower than edge-factored parsing. Joint inference over source and target trees is twice as slow as a pipelined approach that first finds the best source tree. More surprisingly, parsing with only the monotonic QG factors is much faster, whether we pipeline inference over T^S or not. The large number of loops induced in the factor graph by the full set of QG factors seems to be having a substantial effect on the convergence of BP. Note also that enforcing the synchronous hard constraint takes even more time: the SYNCH family needs to enforce at least as many hard constraints as there are soft QG constraints.

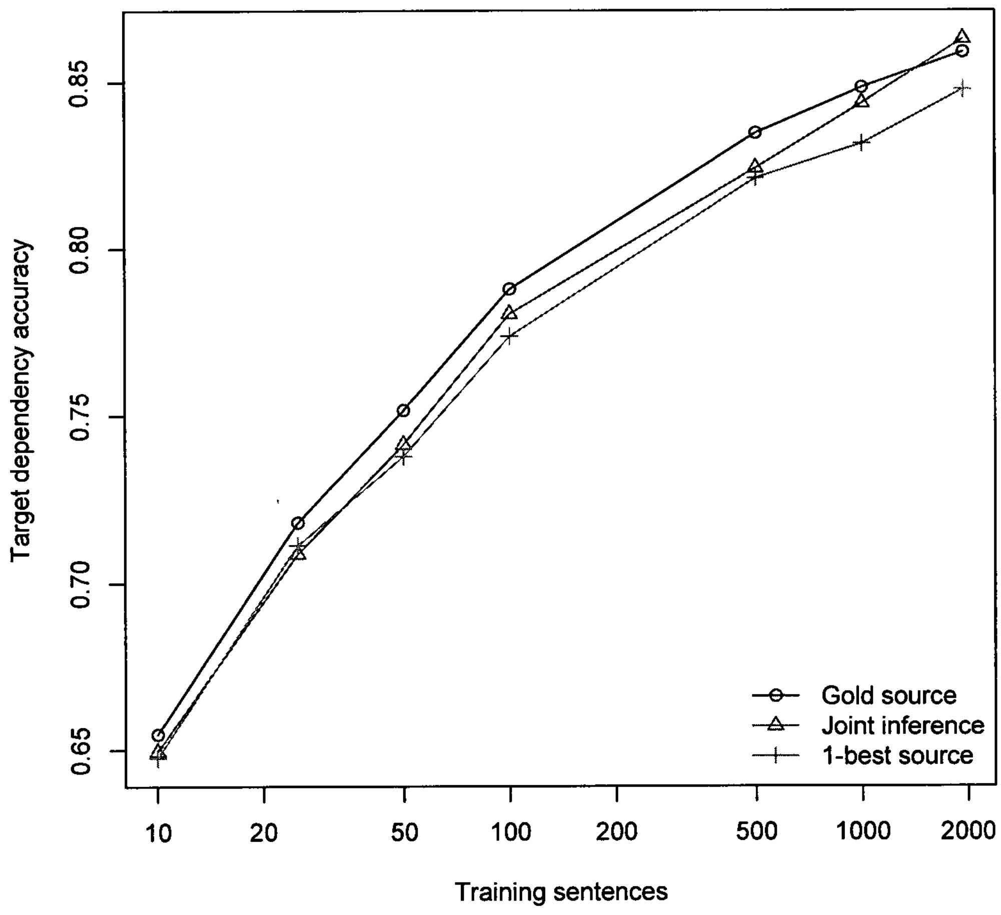


Figure 4.6: Pipelined and joint decoding on bitext with QG training

Inference procedure	**s/sentence**
Edge-factored target	0.06
Target only	0.54
1-best source	1.43
Joint source-target	2.56
1-best source, monotonic	1.15
Monotonic	1.52
Synchronous	2.72

Table 4.4: Bitext inference runtimes

4.7.6 Effect of true alignments and source trees

Projection performance is, not surprisingly, better if we know the true source trees at training and test time (figure 4.7). Even with the 1-best output of the source parser, QG features help produce a parser as accurate as one trained on twice the amount of monolingual data. In ablation experiments, we included bilingual factors only for MONO configurations, with no features for head-swapping, grandparents, etc. When using 1-best English parses, parsers trained only with direct-projection and monolingual features performed worse; when using gold English parses, parsers with direct-projection-only features performed better when trained with more Chinese trees. The penalty for using automatic alignments instead of gold alignments is negligible; in fact, using *Source text* alone is often higher than +*Gold alignments*. Using gold source trees, however, significantly outperforms using 1-best source trees; at small training sizes, adding in gold alignments has an additional effect.

4.8 Discussion

The two related problems of parser adaptation and projection are often approached in different ways. Many adaptation methods operate by simple augmentations of the target feature space, as we did in our monolingual adaptation setup (Daumé, 2007). Parser projection, on the other hand, often uses a multi-stage pipeline (Hwa et al., 2005). The methods presented here move parser projection much closer in efficiency and simplicity to monolingual parsing.

Building on the belief propagation work in chapter 2, we can jointly infer two dependency trees and their alignment, under a joint distribution $p(T^T, A, T^S \mid W^T, W^S)$ that evaluates the full graph of dependency and alignment edges. We have shown how to parametrize this joint model in terms of source- and target-language parser features, alignment features, and features inspired by quasi-synchronous grammar.

We showed that augmenting a parser with quasi-synchronous features can lead to significant improvements—first in experiments with adapting to different dependency representations in English, and then in cross-language parser projection and bilingual parsing. As with many domain adaptation problems, it is quite helpful to have some annotated target data, especially when annotation styles vary (Dredze et al., 2007). Our experiments show

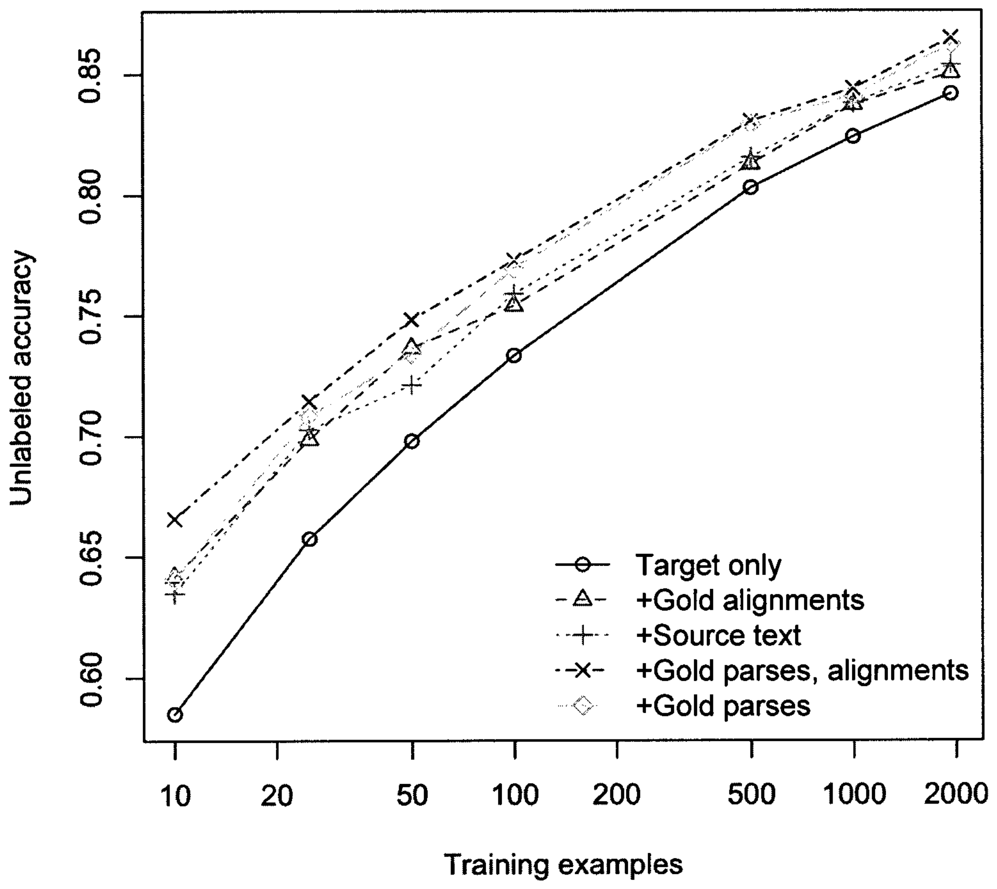


Figure 4.7: Parser projection with target trees. The penalty for using automatic alignments instead of gold alignments is negligible; in fact, using *Source text* alone is often higher than +*Gold alignments*. Using gold source trees, however, significantly outperforms using 1-best source trees; at small training sizes, adding in gold alignments has an additional effect.

that unsupervised QG projection improves on parsers trained using only high-precision projected annotations and far outperforms, by more than 35% absolute dependency accuracy, unsupervised EM. When a small number of target-language parse trees is available, a bilingually trained parser gives a boost equivalent to doubling the number of target trees.

Chapter 5

Extensions and Future Work

In previous chapters, we have focused on models of syntax, specifically dependency syntax, using the combination of graphical models, approximate inference, and graph optimization. We now explore some of the loose ends remaining in our treatment of monolingual and bilingual syntax (§5.1), as well as a broad array of important new application areas (§§5.2–5.5) and methods for improving the efficiency and accuracy of inference in factor graphs (§5.6).

5.1 Modeling Syntax

Before we turn to applications further afield, we sketch some of the areas of monolingual syntactic modeling opened up by the techniques under discussion. After discussing some natural extensions to our models of monolingual dependency parsing (§5.1.1), we outline an approach to modeling constituency structures, the dominant approach to syntactic representation in linguistics (§5.1.2). In any case, most linguists, computational and otherwise, represent syntactic structures with trees, and so our presentation of "graph inference" for syntax has been confined to the special case of trees as output structures. In §5.1.3, we discuss some alternatives for modeling syntax with non-tree graphs, and in §5.1.4, we discuss the application to multi-tree syntactic representations. These multi-structure views of syntax lead naturally to models for coordinating several linguistic levels in the next section (§5.2).

5.1.1 Further experiments in dependency parsing

While chapters 2 and 3 explored approaches to dependency syntax, they were by no means exhaustive. In this section, we outline three natural extensions to the models presented earlier: labeled parsing, nonprojective parsing with a hidden projective structure, and parsing with hidden subcategorization variables.

We focused almost exclusively on unlabeled parsing in the experiments. Preliminary experiments with *hidden* labels in §3.2.5 did not, for reasons we discussed there, have any effect on unlabeled accuracy. On the other hand, that does not imply that *supervised* training of edge-factored (non-loopy) labeled models would be ineffective or that it would simply be equivalent to assigning labelings to the one-best tree. As in the hidden label experiments, we would need $O(n^2)$ multinomial variables taking values from $\epsilon \cup \mathcal{L}$, the union of the empty string with the set of labels. One of the efficiency drawbacks of the hidden label experiments was that any dependency could take any label. For supervised training, we should be able to restrict the set of candidate labels using, e.g., POS of candidate parent and child with little loss in recall. Such sparsification would be even more useful once binary interactions among the labels is added, which would require in the worst case $O(|\mathcal{L}|^2 n^2)$ potential values.

The experiments within higher-order nonprojective parsing in §3.4 show a significant increase in accuracy over projective models when trained and tested on treebanks with high nonprojectivity. In the English trees, the conversion to dependency trees left a small amount (1% of edges) of nonprojectivity due to reattaching extraposed nodes to traces. But even with NoCross factors to discourage overzealous edge crossing, the projective parser outperformed the nonprojective one. Somewhat in the spirit of multi-tree models of syntax (§5.1.4 below), it could be useful to train a model with a set of hidden link variables connected to a PTree constraint and another set of observable link variables connected to a Tree constraint. The corresponding projective and nonprojective variables would be connected by binary soft constraints parametrized to learn which attributes of dependency links were likely to lead to crossing. In other words, since even in Czech and Latin, most links are projective, we would do better to model the exceptional crossing, rather than the common non-crossing, cases.

Finally, while individual hidden role labels were shown to be ineffective, previous research has shown than modeling such hidden phenomena as the argument/adjunct distinction and their roles in subcategorization frames can improve parser accuracy (Collins, 1997). The CHILDSEQ and VALENCE constraints (§2.2.5), for instance, are implemented as finite-state machines that make deterministic transitions given the link variables. In other words, if we know the dependency tree, we know the child sequence of each head. If, in addition to a boolean link variable, each dependency link were also modeled by a ternary argument/adjunct variable, then the VALENCE constraint, for instance, could count only arguments. Further elaborations on this idea, however, are now often covered by low-level semantic representations under the rubric of **semantic role labeling**, which we take up in §5.2.2 below.

5.1.2 Constituency parsing

This thesis has concentrated on models of dependency trees. Models of constituency structure, most commonly context-free grammars, are far more widespread in NLP. Such models have not, however, always taken the form $p(T|W)$.

While speech recognition researchers used probabilistic CFGs in the 1970s (Baker, 1979), their use in broad-coverage syntactic parsing expanded significantly with the advent of the Penn Treebank and similar datasets (Marcus et al., 1993). As with sequence labeling tasks, generative (Carroll and Charniak, 1992) and action-based models (Magerman, 1995) achieved earlier adaption than more computationally globally normalized models. Some of the earliest globally normalized parsing models were for the richer formalism of LFG (Johnson et al., 1999), but in recent years, the computational cost of conditionally trained log-linear CFG parsers has become more acceptable (Finkel et al., 2008). Log-linear models of constituents have also been attractive in information extraction tasks, where the grammar constant is nowhere near as large as a treebank grammar. Constituency parsing methods can thus be used to segment and label information fields and named entities with nested structure more naturally than linear-chain CRFs (Viola and Narasimhan, 2005; Finkel and Manning, 2009); CFG parsing models have also been used to apply semantic labels to the nodes of trees such as table cells (Jousse et al., 2006). If nested structures are not required, semi-CRFs provide a finite-state alternative for detecting spans with an upper bound on their length (Sarawagi and Cohen, 2005).

We can formulate constituency parsing as a graphical model, as we did for dependency parsing, first by defining variables to encode trees.

- Let U be the set of nonterminals allowed in our trees—we do not say, in the grammar, because the grammar may not be explicitly represented.
- Let $\{B_{ij} : 0 \leq i, j \leq n, i < j\}$ be $O(n^2)$ boolean variables such that $B_{ij} = \mathsf{true}$ iff there is a *bracket* spanning i to j.
- Similarly, let C_{ij} be $O(n^2)$ *multinomial* variables taking values $\epsilon \cup U$. If there is a constituent of type u from i to j, then $C_{ij} = u$, and ϵ otherwise.

This representation cannot capture unrestricted constituency trees over non-empty yields: since there is only one bracket variable per span, we cannot represent unary productions above the leaf level.[1] Any tree, however, may be represented by a context-free grammar,

[1]A kludge would be to add some bounded number of unary productions with extra variables, but this would make enforcing a tree constraint more complicated.

and any CFG may be converted to Chomsky normal form, which eliminates non-leaf unary productions (Hopcroft and Ullman, 1979). In practice, treebanks usually have trace nodes removed and are converted to a Markovized binary-branching format to reduce sparsity in the extracted rules (Johnson, 1998). An alternative representation that avoids some of these drawbacks is a case-factor diagram (McAllester et al., 2004).

To coordinate bracket and constituent variables, we need a family of $O(n^2)$ hard "e pluribus enum" (EPU) constraints to map boolean to multinomial variables. The potential $\text{EPU}_{ij} = 1$ if $B_{ij} = \mathsf{false} \wedge C_{ij} = \epsilon$ or if $B_{ij} = \mathsf{true} \wedge C_{ij} \neq \epsilon$; the potential is otherwise 0. When a bracketing has a very low probability, it may not be worth while to extract features for or propagate messages from the corresponding constituent variable. This suggests that a coarse-to-fine strategy could avoid instantiating many constituent variables or could clamp their values to ϵ (Charniak et al., 2006). One could also introduce constituent variables at several levels of refinement. Hard constraints, similar to EPU, would enforce deterministic refinements: e.g., $\mathsf{N} \sim \mathsf{NNS}$ gets potential 1, but $\mathsf{N} \sim \mathsf{VBZ}$ gets potential 0. Non-deterministic refinements would require soft constraints. Different nonterminal refinements could also represent different Markovizations of an underlying treebank. During training, we could exploit the sum-product algorithm to sum over different Markovizations.

As in the dependency parsing case, we also introduce families of $O(n^2)$ unary soft constraints on the basic variables: BRACKET_{ij} factors and CONSTIT_{ij} factors. The former requires a single potential that fires if B_{ij} is present; the latter requires $|U| + 1$ potentials that fire if a constituent of the appropriate type (or no constituent) is present. The features used to calculate these potentials could examine the words or punctuation at the edges of, or just outside of, the given constituent; the length of certain constituents; their proximity to the beginning or end; and POS tag sequences (if available). In the bracketing case, many of these features are similar to the Constituent-Context Model of Klein and Manning (2002)—but in a globally normalized, and not a deficient generative, model.

Hard constraints enforce that the variables represent a tree that spans the whole sentence with no overlapping brackets. Bounded factors could represent certain of these constraints—e.g., we could introduce $O(n^4)$ binary factors to prohibit crossing brackets—but global constraints over all variables, as with PTREE and TREE, are asymptotically faster. We sketch the properties of the hard constraints on the B_{ij} variables CKYBRACKET, which ensures that they form a binary-branching tree, and EARLEYBRACKET, which allows for a greater fanout.

In both cases, the odds ratios of the bracket values function as "grammar" weights on the productions. Call these quantities O_{ij}. Consider the case of the CKY constraint. A bracketing parser uses the grammar with a single nonterminal X

$$X \rightarrow X\ X$$

$$X \rightarrow a$$

where a is a dummy terminal replicated n times. When running the inside algorithm to calculate the weight of all trees, we accumulate inside quantities $\beta_X(i,j)$ (in the notation of Manning and Schütze, 1999). Each rule that adds an increment to $\beta_X(i,j)$ is be multiplied by O_{ij}. [2] Having calculated $\beta_X(0, n)$, we can compute the outside probabilities and the posterior probabilities of brackets. Propagation for CKYBRACKET thus takes $O(n^3)$ time with a grammar constant of 1.

The version of inside-outside for EARLEYBRACKET might naively seem to take $O(n^4)$ time: unless we put an arbitrary limit on constituent fanout, we need rules with from 1 to n nonterminals on the right-hand side. Observe, however, that the dotted rules in the Earley chart can "forget" the number of X nonterminals consumed so far. Equivalently, we can represent the right-hand side of the bracketing grammar rule by a finite-state machine, so that $X \rightarrow XX+$. This allows us to reuse slots in the Earley stack decoder and achieve cubic runtime. An upper bound on fanout might, nevertheless, be empirically useful.

The combination of soft unary and tree constraints formulates a "bracket-factored" or "constituent-factored" model. Such models will not, in general, be very effective with tree grammars with a high degree of deterministic dependencies among constituents, e.g. from conversion to Chomsky normal form. Like edge-factored dependency models, however, their low grammar constant enables very efficient, though still cubic-time, inference.

The natural next step for a constituency model is to add $O(n^3)$ rewrite rule factors. For binarized trees, we connect a ternary REWRITES$_{ijk}$ to C_{ij}, C_{jk}, and C_{ik} for $i < j < k$. In a naive implementation, each REWRITE factor has a potential table with $(|U|+1)^3$ entries (including structural zeros to prohibit combining ϵ constituents). Most grammars will benefit

[2]With a "felicitous" order of updates, as in traditional CKY, to $\beta_X(i, j)$, we can factor out O_{ij} and wait until the inside value is fixed before multiplying it in. This entire procedure is similar to the procedure for maximum constituent recall decoding (Goodman, 1996), except that all quantities in the inside algorithm are summed instead of maximized.

from sparse implementations. In the dependency parsing case (§2.4), the addition of, e.g., grandparent constraints induces a loopy factor graph but maintains $O(n^3)$ asymptotic runtime compared to $O(n^4)$ for dynamic programming. Here, we pay the cost of approximate inference with no runtime speedup. If we want to enforce all of the constraints encoded in a CFG, we should simply use a *weighted* constituency tree factor, i.e. a parser that can compute the posterior probabilities of constituents for our C_{ij} variables.

What, then, are the advantages of a factor graph representation? As shown in Riedel et al. (2010), we can achieve significant speedups if we have a reasonably accurate local (here, constituent-factored) model that we augment with a few "important" higher-order factors.[3] In other words, instead of encoding the whole grammar, we could add REWRITE factors as necessary to achieve good performance. In MAP decoding, such **relaxation** methods have been shown to be quite effective (Tromble and Eisner, 2006; Riedel and Clarke, 2006). More recent work has applied an analogous technique of minimizing the divergence between full and sparse models to marginal inference (Riedel and Smith, 2010; Riedel et al., 2010). Extending this argument, we can implement a simple CFG as weighted tree factor and apply the selected higher-order constraints—such as grandparent or sibling annotation—using binary factors rather than much larger grammars. Finally, this representation can serve as a subgraph of larger models, for aligning constituency trees in different languages (cf. Smith and Smith, 2004; Burkett et al., 2010) or for aligning constituency trees to dependency trees, LFG f-structures, or logical forms.

[3]A special case might involve factoring REWRITE constraints from ternary to binary factors that use $O(3|U|^2n^3)$ instead of $O(|U|^3n^3)$ potential values.

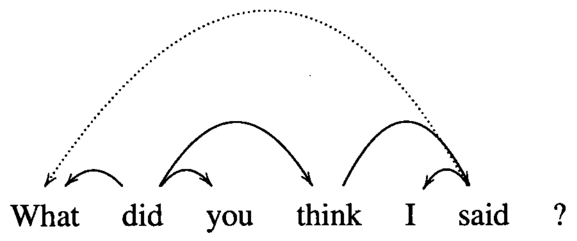


Figure 5.1: A syntactic DAG: the dotted edge connects the extraposed *What* to its "original" head.

5.1.3 Non-tree syntax

At the beginning of this thesis (§1.1), we presented graph inference as a paradigm for linguistic inference. In the majority of the document, however, the outputs of our models have been constrained to special cases such as projective (chapter 2) and nonprojective (chapter 3) spanning trees for dependency structures, and alignments with constrained degree (chapter 4). When costs are edge-factored, these are all problems where efficient algorithms exist (if only for MAP inference, in the case of alignments).

Some syntactic theories, however, have proposed more general representations, most commonly (connected) directed acyclic graphs (DAGs) (Chen-Main, 2006; Buch-Kromann, 2006; Hudson, 2007). In the sentence, "What did you think I said?" (figure 5.1), the extraposed interrogative *what* could be said to have a grammatical relation to the matrix verb *did* and its "original" governor *said.* In cases of ellipsis, such as "Sam cooked and ate the beans," these "syntactic graph" theories connect the shared subject *Sam* to both verbs.

Finding the highest-weighted acyclic subgraph of a directed graph has long been known to be NP-hard (Karp, 1972).[4] While there exist approximation algorithms for the general max DAG problem (Hassin and Rubenstein, 1994), we observe that many theories of syntactic graphs distinguish between "primary" edges, which form a tree, and optional "secondary" edges, which induce undirected cycles.[5] We could thus model the primary edges with link variables subject to a tree constraint, as usual, and introduce a matching set of secondary link variables, which would be constrained to be false if its corresponding primary link variable were true.

But how can we prevent the secondary links from inducing cycles in the combined graph? Assume that we know the full primary tree. A secondary link will only induce a directed cycle if it runs from a word to one of its ancestors. For MAP inference, we could continue this procedure by greedily adding secondary edges and maintaining sets of ancestors. For marginal inference, we can efficiently compute beliefs about ancestor relations only for projective trees, and without a procedure for updating these beliefs given secondary edges, we cannot encode all of the necessary constraints. Despite the lack of exact inference, one property may make DAG inference effective in practice: we will usually want secondary links to be much sparser that primary links and can enforce this constraint with factors that score the number of secondary edges. We can also adopt relaxation techniques to add members of the exponentially sized family of cycle-prohibiting constraints as necessary (Riedel and Clarke, 2006) or possibly multi-commodity flows for MAP inference (cf. single commodity flows in Martins et al., 2009a). Another approach is outlined

[4]Since the weights are positive, a maximum subgraph is guaranteed to be connected.
[5]This "coloring" of the links is similar to the multiplanar parsing of Yli-Jyrä (2003).

in our discussion of semantic role labeling, in §5.2.2 below: we model a syntactic tree and extra semantic links, although the semantic structures can double the syntactic ones.

5.1.4 Multi-structure syntax

Formalisms such as Lexical Functional Grammar (LFG) and Autolexical Syntax express grammaticality constraints by reference to multiple structural representations (Falk, 2001; Sadock, 1991). In LFG, for instance, an utterance must satisfy both the constituency c-structure and the functional f-structure. The c-structure is expressed as a context-free grammar and would thus, on its own, admit efficient inference algorithms; however, intersection with functional constraints, which can refer to arbitrary features of the utterance and its parse, makes inference intractable. One solution to LFG parsing is simply to parse with a (sparse) CFG alone and then exhaustively enumerate c-structures while knocking out those that violate f-structure constraints (Johnson et al., 1999). Other work on LFG has exploited dynamic programming with a packed representation of trees to achieve more efficient, though still exponential runtime (Geman and Johnson, 2002). Researchers, at least as early as Maxwell and Kaplan (1993), have explored the advantages of polynomially solvable subproblems—in their case, they combined cubic-time (Boolean) chart parsing for the c-structure and exponential-time Boolean satisfiability solving for the f-structure in LFG.[6] These strategies are applied to an unweighted grammar, but it would be interest-

[6]Maxwell and Kaplan (1993) describe interleaved pruning strategies reminiscent of message passing between the PTREE factor and other constraints and non-interleaved strategies—both top-down, as in coarse-to-fine parsing, and bottom-up, as in forest reranking (Huang, 2008).

ing to combine a factor for weighted constituency parsing as in §5.1.2 (with or without a grammar) with factors for functional constraints. Similarly, Autolexical Syntax's parallel constituency-tree representations could communicate with each other via message passing.

5.2 Multiple Linguistic Levels

The boundary between syntax—the core module in generative linguistics—and other processes such as morphology or compositional semantics varies across different theories. In NLP applications, however, morphology and semantics are usually handled separately. Much of the f-structure syntax in LFG, for instance, is similar to "semantic roles" when handled by NLP researchers. In this section, we briefly sketch models that combine our factor-graph representations of syntax with extensions for morphological and semantic disambiguation.

5.2.1 Syntax and Morphology

We have assumed, in the parsing models discussed for far, that part-of-speech tags have already been assigned to the words. Finite-state sequence taggers—HMMs, conditional Markov models, conditional random fields, etc.—can achieve quite high accuracies for English (e.g., Toutanova et al., 2003); in any case, if we are focused solely on dependency accuracy, we may not care if the tags we use for our features are *correct* so long as they are *consistent*. For other languages, morphological disambiguation is not as accurate

(Smith et al., 2005).[7] Finite-state models perform particularly poorly on case, in languages that mark for it. While differences in inflectional syncretism may provide accidental cues for disambiguating case locally, many case decisions would be nearly deterministic if a dependency tree were known. In contrast, an ambiguity in verb tense will often be left underspecified by the syntax of a single sentence, except when sequence of tenses is followed in subordinate clauses: compare "I like plums" to "You knew I liked plums," but then substitute an ambiguous verb such as *hit*.

The most recent work on joint morphology and syntax for data-driven parsing has been for modern Hebrew (Cohen and Smith, 2007; Goldberg and Tsarfaty, 2008). As with most recent work on modern standard Arabic, morphological analysis for modern Hebrew usually concentrates on segmenting an orthographic word into a stem, proclitics and enclitics, and possibly some morphological suffixes. We anticipate that joint morphological and syntactic inference will be even more helpful for languages such as Czech or Latin, where multi-attribute fusional morphology combines with free word order to foil local sequence methods.

When discussing the runtime advantages of the BP approach (§2.4), we noted that joint edge-factored inference over tags and dependencies would take $O(n^3 + g^2n^2)$ with BP for a vocabulary of g tags, as opposed to $O(g^2n^3)$ with dynamic programming. The Penn Treebank approach of fusing word classes and morphological attributes leads to an uncomfortably large g for a language such as Latin or ancient Greek, where a verbal lemma can

[7]The "parts of speech" in the Penn Treebank and some other corpora fuse traditional word-class information, such as nouns, verbs, etc., with morphological information, such as VBZ for "3rd person singular present tense verb".

be expressed in thousands of forms (Crane, 1991). The approach of Smith et al. (2005), where morphological attributes such as gender, number, case, tense, etc., were represented by separate variables, should be more appropriate. We can further reduce the search space if the values of thpse variables can be reduced, for common words at any rate, by morphological dictionaries, which linguists have constructed for many languages (Beesley and Karttunen, 2003).

5.2.2 Semantic Role Labeling

Another layer of annotation closely tied to syntax are certain forms of shallow semantics. In particular, a substantial amount of effort in the NLP community has been devoted to semantic role labeling (SRL), which, while not a complete account of compositional semantics, manages to abstract away some phenomena of surface syntax (as, famously, active vs. passive) in order to describe underlying argument structures of nouns, verbs, and adjectives. SRL has been performed consequent to both constituency (Gildea and Jurafsky, 2000) and dependency parsing (Hacioglu, 2004). While most of the best systems are pipelined and used the one-best syntactic tree, some models have shown improvements with joint inference (Toutanova et al., 2005).

The dependency version of SRL produces a labeled, directed graph over the words of a sentence, some of which are **predicates** and some (including both predicates and non-predicates) are **arguments** with edges from those predicates (figure 5.2). The *semantic roles* filled by the arguments label the edges. Unlike a syntactic dependency structure, the

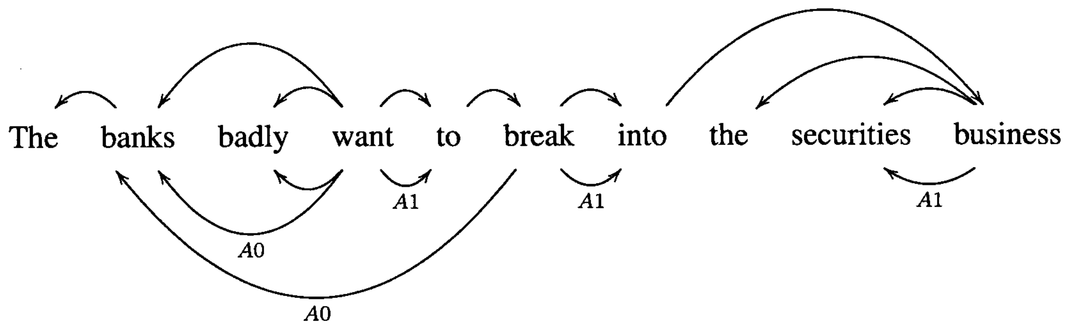
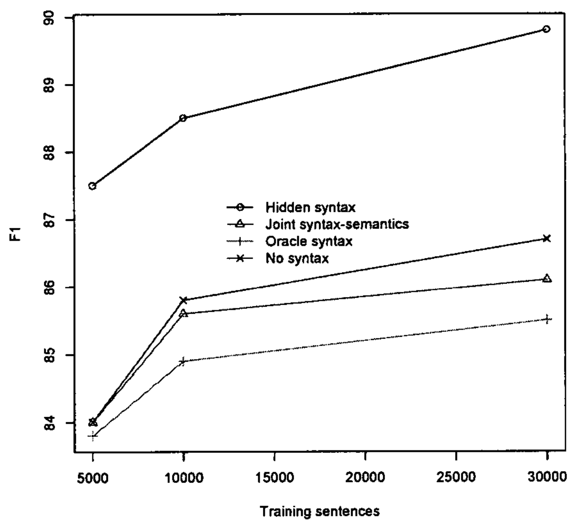


Figure 5.2: Semantic role labeling with dependency trees: The dependency tree is drawn above, and the semantic graph is shown below the sentence. Predicates correspond to lexical items, and the semantic graph need not cover all words. Some words, such as *banks* may be arguments of more than one predicate.

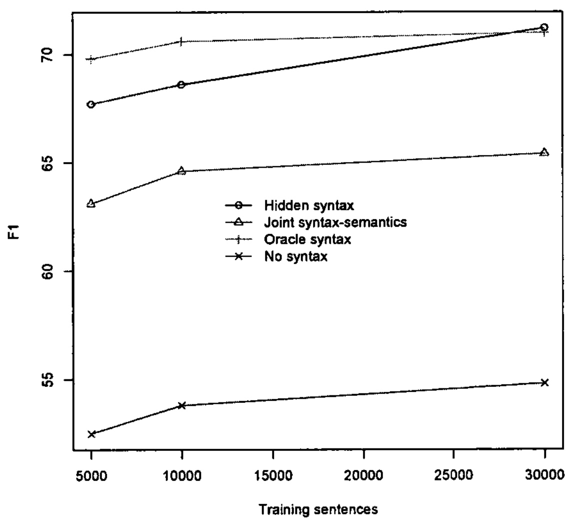
SRL graph is by no means a tree: the graph is *not connected*, since some words are neither predicates nor arguments; a word may serve as an argument for multiple predicates, as in the figure, and thus have *multiple parents*; and the graph is even sometimes *cyclic*, with nouns and verbs mutually taking each other as arguments. Even the subgraphs comprising the arguments of a single predicate are not entirely well-behaved. For example, while most predicates have a single $A0$ argument (usually the agent) and $A1$ (theme), about 5% of predicates have multiple arguments in the same role.

There is, nevertheless, a strong correlation between syntactic dependencies and SRL links, and previous work has naturally depended on features of the syntactic tree to train SRL models. It is easy to augment our factor-graph dependency model with extra variables to represent whether a word is a predicate and variables for whether two words are in a predicate-argument relation. The semantic link variables can then be connected to each other with factors that score the arity of predicates—how many arguments, and how many $A0$ arguments, should a predicate have? Finally, the semantic link variables can be connected to dependency variables, as in the quasi-synchronous bilingual factors in §4.7.2.3. Some predicate-argument links may correspond exactly to dependency links, as in the *banks* $\xleftarrow{A0}$ *want* SRL link in figure 5.2. In other cases, a direct semantic link may arise from a more distant syntactic connection—for example, the control construction in the example means that the *banks* $\xleftarrow{A0}$ *break* semantic relation corresponds to a c-commanding syntactic relation.

This close relation between syntax and semantics makes it unremarkable that an SRL model without access to dependency structures performs much worse than one that uses the true dependency tree—viz. "No syntax" vs. "Oracle syntax" in figure 5.3. Also, "Joint syntax-semantics" inference incurs some loss compared to the oracle syntax. With a joint factor-graph model of dependency syntax and semantic roles, we can both train and test with the dependency tree as a hidden variable. What is surprising is that this setup ("Hidden syntax") outperforms the joint condition, which has access to the true dependency trees at training time, and even, at high levels of SRL training data, the setup where we know the true dependency tree at test time. The syntactic trees produced by this hidden-variable setup only get about 50% dependency accuracy compared to treebank annotations, but that is of less importance if the purpose of the model is solely semantic role labeling.



(a) Predicates



(b) Arguments

Figure 5.3: Performance of semantic role labeling. Treating syntax as a hidden variable performs better, on F1 measure for predicate and argument prediction, than joint training and decoding for syntax and semantics.

5.3 Bilingual Training for Monolingual Parsers

In chapter 4, we discussed three monolingual and bilingual parser adaptation setups, but there is much more work to be done in this area. Consider, in particular, the case of unsupervised parser projection: we wish to train on bitext and test on monolingual target-language sentences. The monolingual target-language features that are most important at test time will have been trained in the presence of source-language data. Since we don't observe these source sentences in monolingual target-language texts, they need to be inferred. In other words, we would need a target-to-source translation system in order to use our target-language parser!

For the unsupervised parser projection experiments in §4.6, we parsed target-language text with target features alone and ignored the small number of quasi-synchronous and alignment features. While this strategy was enough to beat unsupervised baselines, it fails to improve over parsers trained on even a small number of unlabeled trees. Moreover, the generative model was much simpler than the joint model from §4.7. When we try to play the same trick with the joint model—simply ignore all non-target features—monolingual target parsing accuracy is even worse. In this section, therefore, we discuss some approaches to learning monolingual parsers using the richer joint model.

5.3.1 Self-training and co-training

The simplest approach to using bilingual data is a form of self-training. We acquire trees on the target side of the bitext by training a bilingual parser and decoding or using simple projection (§4.6.3.3). Then, we train a monolingual parser on the target side of the bitext, with its mixture of hand- and machine-labeled trees.[8]

Another training workaround would be to stipulate that monolingual target text simply connects to non-existent source text with NULL alignments. In other words, we would have some bitext training data and some monolingual target-language training data whose source side was null. This retrenchment, however, means that the bilingual factors, which on bilingual training data helped the target-language factors to find the right T^T, may be undertrained for their task. The situation appears even worse if we have some features that strongly prefer some NULL-aligned dependencies over others. In addition, we may over-regularize some bilingual features if we train on a mix of monolingual target and bilingual text.

We can acknowledge that monolingual and bilingual parsing require separate parameterizations and perform multitask learning with some of the monolingual features tied, or regularized to be close to each other. In preliminary experiments, this approach was not successful—there was no reason to expect that the *values* of the target-language's parameters should be close in monolingual and bilingual training.

Finally, we can adopt a **co-training** approach where the monolingual and bilingual

[8]Smith and Smith (2004) presented an even simpler version of this training setup: the target language parser was trained only on monolingual trees and only the *inference* was bilingual.

models train each other using bitext. Each model is able to infer T^T from W^T (and also the source side of the bitext, in the bilingual model's case), and so the output of each can be supervision for the other. The process may be started by training either a monolingual or a bilingual model.

Figure 5.4 shows the monolingual accuracy of two co-trained parsers. In one case, co-training was initialized by training a monolingual parser on target trees alone; in the other, co-training started by training the bilingual parser on bitext with annotated target trees (just as in training the bitext parsers in §4.7). The plot also shows the learning curves from earlier experiments on target-language training alone and, as an upper bound, the performance of the bilingual parser, which has access to the source sentence at test time. The co-trained parsers consistently outperform the target-only baseline, but while for seed sizes of 10 and 100 trees there are 4.5% and 1.5% absolute improvements respectively, the improvement is 0.4–0.6% with 1000 target trees. To achieve more robust results, we may need to address the sample selection problem: which trees produced by one parser should be used to train the other? At present, we find that using output on the whole corpus performs better than a confidence-based threshold, but performance of co-training still declines after the first iteration.

5.3.2 Noisy channel models

An alternative approach is to cast parser projection as a noisy channel model. To achieve this, we factor the joint distribution $p(T^S, T^T, A \mid W^S, W^T)$ into $p(T^S, A \mid$

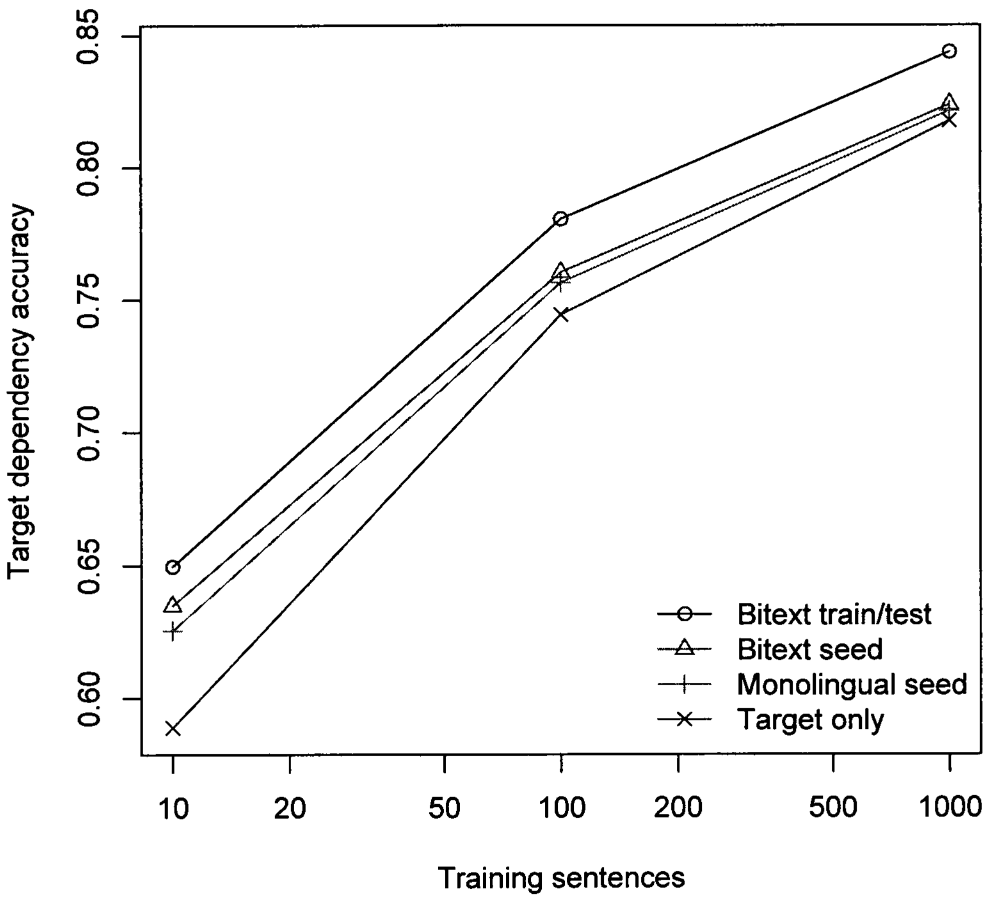


Figure 5.4: Monolingual accuracy of a parser co-trained on bitext

$$\begin{tikzcd}
 & W^T \arrow[dl] & \\
T^T \arrow[dr] & & W^S \arrow[dl] \\
 & T^S, A &
\end{tikzcd}$$

Figure 5.5: A noisy-channel model of parser projection

$T^T, W^S)p(T^T \mid W^T)$ (figure 5.5). If T^S is observed at training time, or if we trust a $p(T^S \mid W^S)$ parser enough to treat its one-best output as observed, we have the following optimization:

$$\max \sum_{A,T^T} p(T^{*S}, A \mid T^T, W^S)p(T^T \mid W^T) \tag{5.1}$$

What does this model do when parsing monolingual W^T? As a child of T^T in the directed graphical model, the unobserved T^S, A falls away without affecting the distribution of T^T, and we can simply use the monolingual parameters of $p(T^T \mid W^T)$ without change.

If we don't trust the one-best output of $p(T^S \mid W^S)$, we can still encourage the trees T^S produced by quasi-synchronous projection to agree with the output forest F^S of a monolingual S-parser. We introduce a Boolean "agreement" variable M that is true if its parents T^s and F^S match. During training, we stipulate that M is observed as true (figure 5.6). This exact matching criterion could, with some more involved machinery, be softened to a minimum risk objective such as maximizing the number of matching dependency links.

This "agreement" model presents a slightly trickier scenario for monolingual parsing of W^T. If the agreement variable M is still observed as true, we are thrown back on the uncomfortable situation we started with: W^S needs to be imputed via machine translation.

$$\begin{tikzcd}
 & W^T \arrow[ld] \arrow[rd, "MTsystem"] & \\
T^T \arrow[d] & & W^S \arrow[d] \arrow[lld] \\
T^S, A \arrow[rd] & & F^S \arrow[ld] \\
 & M &
\end{tikzcd}$$

Figure 5.6: Modeling agreement between a source parser and projected source trees

Instead, we should simply treat M as unobserved and let the descendants of T^T fall away.

While introducing directed edges between supernodes in the model has made inference simpler at a high level, we can no longer get BP to give us gradients directly to optimize (5.1). Procedurally, we first perform inference on T^T and get marginals over link variables and other variables that connect to A and T^S. These marginals become the fixed messages used in BP inference on A and T^S. The gradient of these T^T marginals with respect to the log-likelihood of T^{*S} is the belief vector as usual. We then need to calculate the gradient of the marginals of T^T with respect to the parameters of that model. For edge-factored models of T^T, this involves a second derivative computation, as described for projective parsers in §B.5 and nonprojective parsers in §3.2.4.

In preliminary experiments with the simple noisy channel model trained to predict a single source tree, we find that the target-language parser is no more accurate (and often worse) than the naive approach of training the joint model and then setting the bilingual feature weights to zero. If the target language parser is trained partly from target trees and partly as a component of the noisy channel, we have not seen any improvement over training on target trees alone.

5.4 Applications of Noisy Alignment

We first presented quasi-synchronous grammar as a generative model of target trees and words (Smith and Eisner, 2006b). Although originally proposed for word alignment and machine translation, QG models were applied in chapter 4 to monolingual and bilingual parser projection. We also discussed the use of QG bilingual configurations—aligned source and target subtrees—as features in a joint model of aligned source and target trees.

The generative form of QG has found useful applications in tasks that score the probability of one document being generated from another, as in language-model-based information retrieval. Wang et al. (2007), for instance, modeled the probability that a question could be generated by a noisy transformation from a candidate answer. Das and Smith (2009) evaluated paraphrase correspondence by a QG in both directions. Woodsend et al. (2010) used a QG to model the generation of headlines and captions from documents.

In many information retrieval applications, however, we do not have large numbers of document-query pairs to train on. Although "learning-to-rank" approaches have become more popular, the number of free parameters in IR models remains in the tens, as opposed to the millions used to train monolingual and bilingual parsers in this thesis. One promising setup for learning alignments between queries and documents is a bootstrapping approach based on pseudo-relevance feedback (Bendersky et al., 2010; Xue et al., 2010). The top results from an query can be used as examples of aligned documents to learn the parameters of the QG model.

Another type of noisy alignment is that between text documents and databases of facts possibly expressed in those documents (Bellare and McCallum, 2009). These databases can serve as "distant supervision" for entity and relation extraction and classification in text in the absence of annotated examples (Yao et al., 2010). While the "template" string between two entity mentions is usually taken as a feature, or even a deterministic indicator, of the relation between those entities (Ravichandran and Hovy, 2002), it might be useful to learn models with hidden syntactic variables to map mentions to entities and relations.

Finally, machine translation could benefit both from the flexibility of modeling with quasi-synchronous features but also from runtime speedups from BP. Decoding is very expensive with a synchronous grammar composed with an n-gram language model (Chiang, 2007)—but the analysis of tagging above (§2.4) suggests that BP might incorporate a language model rapidly. Gimpel and Smith (2009) applied a QG-based decoder to the task of rescoring translation lattices, and it might be useful to incorporate some of their hard constraints on coverage of the source string into a joint model of source and target trees.

5.5 Inferring Conversational Graphs

It is interesting to note here another application of graph inference outside of dependency parsing. Conversations are the basis for social interaction in most online environments: users agree or disagree with, comment on or quote, the posts and links submitted by other users. One important inference task is to reconstruct the relationships between these conversational acts. Some forums and social networking sites encode the reply structure, e.g., by indentation, but other forums, comment threads, and chatrooms simply stream posts as they are received. If each post (barring the post that starts the thread) responds to exactly one previous post, then a thread is a directed tree; forum threads, blogs, email, and other media where participants can respond to multiple posts form a directed acyclic graph (DAG).

Given models that score whether a pair of posts in a thread form a parent-child pair, we can use edge-factored inference to reconstruct a conversational tree (Seo et al., 2009). In fact, when timestamps are provided, a spanning tree can be constructed in one greedy pass: simply attach each post to the previous post with the best parent score. Many posts are quite impoverished, and discourse features over an entire conversation will be necessary for more accurate reconstruction. It will therefore be interesting to investigate higher-order models and BP inference for conversational trees and DAGs.

5.6 More Efficient Inference

We saw in §2.9.2 that BP parsing with combinatorial factors could be faster than dynamic programming with higher-order models, and in §4.7 we demonstrated bilingual parsing with better runtime than synchronous parsing. Our models are still often too slow, due to the constant overheads from large numbers of factors and their features. In Riedel and Smith (2010) and Riedel et al. (2010), we present methods for incrementally adding factors to a factor graph given a procedure for computing marginals under a base model (e.g., a fast edge-factored parser). A greedy approach to adding higher-order factors is especially useful when local models are a decent approximation to the full model distribution. We found that by adding less than 1% of the grandparent and sibling factors, we could achieve runtimes that empirically scaled linearly with sentence length without any loss in dependency accuracy.

Finally, we can take advantage of general improvements to BP proposed by the machine learning community. For example, instead of updating all messages in parallel at every iteration, it is empirically faster to serialize updates using a priority queue (Elidan et al., 2006; Sutton and McCallum, 2007). These methods need alteration to handle our global propagators, which update all their outgoing messages at once. Rather than prioritizing messages, BP should prioritize factors.

Chapter 6

Conclusions

The diligent reader deserves some resolution after attending throughout the preceding chapters and their expository involutions; the savvy reader merits a reward for turning first to the concluding arguments. In this brief chapter, we discharge our obligations to both by a summary of the novel contributions of this thesis and a sketch of their broader implications.

6.1 Algorithmic Contributions

The new algorithmic work in this thesis is of direct relevance to natural language processing and related fields, by contributing:

- the derivation and application of combinatorial optimization algorithms as subroutines in sum-product belief propagation (§2.5);
- $O(n^3)$ runtime guarantees for approximate but accurate projective and nonprojective

parsing with higher-order features for siblings, grandparents, and other linguistically motivated constraints (§§2.4, 3.4); and

- the application of the directed matrix-tree theorem to nonprojective dependency parsing, enabling
    - $O(n^3)$ computation of the partition function (§3.2.2);
    - $O(n^3)$ computation of first-order gradient and entropy (§3.2.3); and
    - further efficient algorithms for second-order gradients, minimum risk training, and minimum Bayes risk decoding (§§3.2.4, 3.3.1).

6.2 Modeling Contributions

This thesis also contributes significant empirical results in learning monolingual and bilingual models of syntax:

- Matrix-tree computations inside BP allow efficient, effective learning and inference for higher-order nonprojective parsers. These parsers significantly outperform probabilistic edge-factored models (also presented in this thesis) and previously-proposed greedy hill-climbing inference methods, especially for highly nonprojective languages such as Dutch, where 11% of dependency links cross other links (§3.4.3).
- Quasi-synchronous models can learn to adapt between different dependency representations in English with high accuracy (§4.4).

- Unsupervised quasi-synchronous projection improves on parsers trained using only high-precision projected annotations and far outperforms, by more than 35% absolute dependency accuracy, unsupervised EM (§4.6).
- When a small number of target-language parse trees is available, a bilingually trained parser gives a boost equivalent to doubling the number of target trees (§4.7).

6.3 Broader Implications

The introductory chapter described a research program for modeling linguistic analysis with systems of hard and soft constraints. In the following chapters, we developed probabilistic inference algorithms for these systems of constraints and learned empirical models from linguistic data. The penultimate chapter described some of our ideas and preliminary investigations for extending this research program in computational linguistics.

The algorithms and models presented in this dissertation promise to enable a wealth of new applications in language technologies such as information retrieval, question answering, machine translation, paraphrase, and summarization, in social network analysis, and in more distant fields—in particular, to applications that need to learn noisy correspondences among multiple structures. More broadly, our generalization of belief propagation to incorporate marginal computations from black-box combinatorial algorithms should enable more efficient and modular inference in the many domains that use graphical models. The work in this thesis has, for example, already inspired applications to multiple sequence alignment of proteins (Bouchard-Côté and Jordan, 2010). Our view of message passing among combinatorial subroutines has been extended in work on Lagrangian relaxation for NLP problems (Rush et al., 2010; Koo et al., 2010). Such modular design principles for graphical models will become increasingly important as we seek to model expanding collections of data with ever more intricate statistical dependencies.

Appendix A

Log-Linear Models

In this Appendix, we review probabilistic structured prediction with **log-linear models**, with and without latent variables, and the training of such models by maximizing conditional likelihood. We review linear models of dependency trees, which dispense with traditional grammars in favor of **constraints** on individual dependency edges and collections of edges. These constraints are parametrized by a linear combination of **features** and their weights. We close with an algorithmic review of combinatorial inference problems for log-linear models: maximum *a posteriori* (MAP) inference to find the most probable output tree; marginal inference to find the posterior probability of particular trees or subtrees; the entropy of the distribution over trees; and other related quantities such as the outside Viterbi score, risk, and its gradient.

A.1 Linear Classification

In order to decide on the correct output, we assign a score s, which is a function of an input-output pair:

$$s(\mathbf{x},\mathbf{y}) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x},\mathbf{y}) \tag{A.1}$$

We thus have a **linear model**, where the score is a weighted sum, or dot product of **feature functions** f with parameters w. Given an input, a set of possible outputs, a weight vector, and feature functions, a classifier must find the output with the highest score. More formally, solve:

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}_\mathbf{x}} \mathbf{w} \cdot \mathbf{f}(\mathbf{x},\mathbf{y}) \tag{A.2}$$

A.2 Log-Linear Models

First, some notation: The training examples of input strings x are indexed by i; possible outputs y for a given x_i are indexed by j; possible hidden structures z for a given x_i and y_{ij} are indexed by k. Individual feature functions and weights in the vectors $\mathbf{f}$ and $\mathbf{w}$ are indexed by ℓ.

In order to turn a linear combination of features and weights into a probability model, we exponentiate the s_{ijk} scores and then normalize them so that the probabilities sum to 1. We may also abbreviate the exponentiated, unnormalized scores with u, i.e. $u_{ijk} \equiv e^{s_{ijk}}$.

Without hidden structure the model is thus:

$$p(y_{ij} \mid x_i) = \frac{1}{Z_i} e^{s_{ij}} \tag{A.3}$$

$$Z_i = \sum_{j'} e^{s_{ij'}} \tag{A.4}$$

If, for instance, we want to parse with such a model, one straightforward approach is to find the output with the larger posterior probability of y—hence maximum a posteriori (MAP) decoding. Since the normalizing constant Z_i is, in fact, constant with respect to a given input, we can, in the absence of hidden structure, find the most probable output without exponentiating or normalizing the scores. In other words, the decoding problem is exactly the same as it was for simple linear models.

With hidden structure, the probability of the output becomes:

$$p(y_{ij} \mid x_i) = \sum_k p(y_{ij}, z_{ijk} \mid x_i) \tag{A.5}$$

$$= \frac{1}{Z_i} \sum_k e^{s_{ijk}} \tag{A.6}$$

$$Z_i = \sum_{j'k'} e^{s_{ij'k'}} \tag{A.7}$$

Here we sum out the hidden structures $z_{ijk}q$, which are often termed **nuisance variables** with respect to the output, and can then search for the best y. Unfortunately, for many problems where the space of possible outputs and hidden structures is combinatorial, this MAP problem is asymptotically harder than the corresponding problem without hidden structure—indeed, in many cases it is NP-hard. It is therefore common to see the following approximation:

$$\max_j \sum_k p(y_{ij}, z_{ijk} \mid x_i) \approx \max_j \max_k p(y_{ij}, z_{ijk} \mid x_i) \tag{A.8}$$

which treats z as just another kind of output.[1]

A.3 Maximum Likelihood Training

The standard approach to training log-linear models is to maximize the **likelihood** of the observed training data. We assume that training examples are independent and identically distributed (iid). The maximization problem to be solved in training is thus

$$w^* = \operatorname*{argmax}_w \prod_i \sum_k p(y_i^*, z_{ik}^* \mid x_i) \tag{A.9}$$

where y_i^* is the correct output structure and z_{ik}^* are the hidden structures compatible with the correct output.

The product form is somewhat inconvenient to optimize. The logarithm function is monotonic, and can thus results in the same maximizer. This **log likelihood** objective is:

$$L \equiv \sum_i \log \frac{1}{Z_i} \sum_k e^{s_{ik}^*} \tag{A.10}$$

$$= \sum_i \left[\log \sum_k e^{s_{ik}^*} - \log Z_i \right] \tag{A.11}$$

Note that without hidden structure, the objective function simplifies to:

$$L \equiv \sum_i s_i^* - \log Z_i \tag{A.12}$$

[1]The approximation is often, confusingly, also termed MAP decoding/inference. Some communities employ **Most Probable Explanation** (MPE) for the approximate version and reserve MAP for (A.8), but it is probably too late to insist on too much consistency on this point.

A.4 Gradient-Based Training Methods

Now that we have defined an objective function, we need an optimization procedure. All of the commonly used optimizers for log-linear models are gradient methods: they use first derivative and sometimes (approximate) second derivative computations to move the objective function towards the optimum. We must beware of one distinction, however: the likelihood of a log-linear model without hidden structure is concave, and thus gradient methods are guaranteed to reach the a unique, globally optimal solution. By introducing hidden structure, we also introduce non-convexity into the objective function. Gradient methods may only reach a local optimum.

Learning methods may be broadly divided into **online** and **batch** procedures, which update parameters after seeing one or a few training examples, or all of them.

Stochastic gradient descent is the simplest online optimizer (Bottou, 2003). The parameter update rule is:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta^{(t)} \nabla L^{(t)} \tag{A.13}$$

At each time-step t, we approximate the true gradient of our objective function by the gradient on one, or a few, examples. The quantity $\eta^{(t)}$ is the **learning rate** for the tth update. The learning rate is often made smaller with successive updates to speed convergence. This update rule is quite similar to the perceptron, which uses the features in the highest-scoring structure if it fails to match the training output and zero otherwise. (The gradient is zero at a local optimum.)

Quasi-Newton methods used gradient information and estimated second-derivative information to update parameters after a complete sweep through the training data. Among the most popular such methods is L-BFGS (Limited-Memory Broyden, Fletcher, Goldfarb, and Shanno, a.k.a LMVM) (Liu and Nocedal, 1989).

A.5 The Gradient of Log Likelihood

For both online and batch methods, we need to compute the gradient of L (A.10). With respect to a given parameter w_ℓ, we have

$$\frac{\partial L}{\partial w_\ell} = \sum_i \frac{\partial}{\partial w_\ell} \log \sum_k e^{s^*_{ik}} - \frac{\partial}{\partial w_\ell} \log Z_i \tag{A.14}$$

We differentiate the numerator as:

$$\frac{\partial}{\partial w_\ell} \log \sum_k e^{s^*_{ik}} = \frac{1}{\sum_{k'} e^{s^*_{ik'}}} \sum_k e^{s^*_{ik}} \frac{\partial s^*_{ik}}{\partial w_\ell} \tag{A.15}$$

$$= \sum_k p(z^*_{ik} \mid y^*_i, x_i) f_\ell(x_i, y^*_i, z^*_{ik}) \tag{A.16}$$

and the denominator as:

$$\frac{\log Z_i}{\partial w_\ell} = \frac{\partial}{\partial w_\ell} \log \sum_{jk} e^{s_{ijk}} \tag{A.17}$$

$$= \frac{1}{Z_i} \sum_{ij} e^{s_{ijk}} \frac{\partial s_{ijk}}{\partial w_\ell} \tag{A.18}$$

$$= \sum_{jk} p(y_{ij}, z_{ijk} \mid x_i) f_\ell(x_i, y_{ij}, z_{ijk}) \tag{A.19}$$

$$= \mathbf{E}_p[f_\ell(x_i, \cdot, \cdot)] \tag{A.20}$$

Note that the partial derivative of the score s_{ijk} with regard to a given parameter is just the value of the corresponding feature function. Note also the simplification with expectations.

The gradient may be concisely written thus:

$$\frac{\partial L}{\partial w_\ell} = \mathbf{E}_p[f_\ell(x_i, y_i^*, \cdot)] - \mathbf{E}_p[f_\ell(x_i, \cdot, \cdot)] \tag{A.21}$$

If there is no hidden structure, the first expectation is deterministic and this is even simpler:

$$\frac{\partial L}{\partial w_\ell} = f_\ell(x_i, y_i^*) - \mathbf{E}_p[f_\ell(x_i, \cdot)] \tag{A.22}$$

A.6 Regularization

Directly optimizing (A.10) will, with some methods, cause some of the parameters w_ℓ to go to infinity since we can always make the log-likelihood better by assigning higher weight to the correct y^*. We therefore introduce a regularization term.

A common method of regularization adds a penalty term to the log likelihood with the sum of squares of the weights. This quadratic or L_2 regularization can also be viewed as a Gaussian prior on the parameters w: the weights are assumed a prior to be distributed according to a multivariate normal with zero mean and a covariance matrix with diagonal all σ^2.

The L_2-regularized objective function is:

$$F \equiv \sum_i \log \sum_k p(y_i^*, z_{ik}^* \mid x_i) - \frac{1}{2\sigma^2} \sum_\ell w_\ell^2 \tag{A.23}$$

In other words, if a parameter w_ℓ becomes too large in the positive or negative direction, the regularization term will penalize the objective function unless that parameter is able to increase likelihood by a greater amount.

The amount of regularization is controlled by a variance parameter σ^2. When variance is low, regularization is strong, and parameters stay close to zero. When variance is high, parameters are able to range farther from zero in either direction.[2]

The gradient of the regularized objective function is:

$$\frac{\partial F}{\partial w_\ell} = \mathbf{E}_p[f_\ell(x_i, y_i^*, \cdot)] - \mathbf{E}_p[f_\ell(x_i, \cdot, \cdot)] - \frac{1}{\sigma^2} w_\ell \tag{A.24}$$

A.7 Motivation for Maximum Likelihood

The exponential form of log-linear models is often motivated by the related maximum entropy problem. This derivation strictly only applies to the likelihood objective function we introduced above. More generally, we can see the likelihood as one of a class of distance-minimizing objective functions.

In particular, consider the point distribution p^*, which assigns probability 1 to the correct output structure y^* and probability zero to other events. An information-theoretic expression of the difference between two distributions is the Kullback-Leibler divergence:

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

[2]Most models use this setup with a single variance parameter. If we have prior knowledge that certain features are more helpful, we could increase their individual variances. If we have prior knowledge that if one feature w_a is high then another w_b is likely to be high, we might introduce covariance terms. These changes would require replacing the inverse variance $\frac{1}{\sigma^2}$ with an inverse covariance matrix Σ^{-1}.

If we minimize the divergence between the ideal p^* and our model p_w, we have (omitting hidden zs for clarity)

$$\min_w D(p^* \parallel p) = \min_w \sum_j p^*(y_{ij} \mid x_i) \log \frac{p^*(y_{ij} \mid x_i)}{p_w(y_{ij} \mid x_i)} \tag{A.25}$$

$$= \min_w - \log p_w(y_i^* \mid x_i) \tag{A.26}$$

since the probability under p^* for all y_{ij} other than y_i^* is zero and with the convention that $0 \log 0 = 0$.

The divergence-minimization approach is also useful in the analysis of bootstrapping and self-training algorithms for semi-supervised learning (Abney, 2004; Smith and Eisner, 2007).

A.8 Minimum Risk

This derivation makes it clear that the likelihood function is very strict: all incorrect structures—whether they contain one mistaken dependency link or twenty—are assigned zero weight by p^* during training. We might prefer to penalize some ungrammatical structures more heavily than others, which insight leads to other training and decoding strategies for probabilistic models.

Researchers in many areas of natural language processing have long observed that high likelihood, or even low perplexity, is not always well correlated with low error rate. In speech recognition, an early bastion of empirically driven methods, researchers have for a while applied methods to directly minimize errors during training (Bahl et al., 1988). More recently, almost every statistical machine translation system has adopted minimum error training to directly minimize a translation evaluation metric such as BLEU on held-out data (Och, 2003). In contrast to the likelihood surface, however, the error surface for discrete structured prediction is not only riddled with local minima, but piecewise constant and not everywhere differentiable with respect to the model parameters. A small change in the parameters w of a dependency model will probably not result in any change in the dependency links predicted by the parser. For this reason, most practical applications of direct error minimization adjust only a small number of hyperparameters (about 20 in many MT systems).

If we have a probabilistic model, we can avoid some of the drawbacks of the error surface by instead minimizing the expected error, or **risk**. If we have a loss function L that assesses a penalty for having the model predict y when the true structure is y^*, we have the following objective:

$$\min_w \sum_i \mathbf{E}_{p_w} L(y_{i\cdot}, y_i^*) = \min_w \sum_i \sum_j p_w(y_{ij} \mid x_i) L(y_{ij}, y_i^*) \tag{A.27}$$

We can thus see (A.25) above as minimizing the risk under an "all or nothing", or 0-1, loss function.

While risk, unlike likelihood, is not necessarily convex, it is continuous and differentiable. Minimum risk training has been shown to improve training in MT (Smith and Eisner, 2006a); in particular, the now differentiable objective can be optimized by gradient-based methods, which makes training large numbers of parameters more practical (Li and Eisner, 2009).

A.9 Minimum Bayes Risk

Probabilistic models also enable a class of loss-aware decoding procedures, called **minimum Bayes risk** (MBR), which were adopted in speech recognition (Goel and Byrne, 2000) and thence became widespread in statistical MT (Kumar and Byrne, 2002, 2004; Tromble et al., 2008; Kumar et al., 2009).

As opposed to minimum risk *training*, a parser doesn't have access to the correct analyses at *test* time. Instead, we can use the probability model p to compute the risk of each possible output. If we weight those outputs by a prior distribution $\tilde{p}$, we get the MBR objective:

$$\mathbf{y}^* = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}_\mathbf{x}} \tilde{p}(y \mid x) \sum_{y' \in \mathcal{Y}_\mathbf{x}} p_w(y' \mid x) L(y, y') \tag{A.28}$$

Many applications employ a uniform prior distribution. While we might estimate $\tilde{p}$ in various ways, a useful class of priors impose structural constraints. Goodman (1996), for instance, described an MBR objective for constituency parsing that he called "maximum constituent recall". He calculated the posterior probability of constituents of the base model p and then fed those probabilities as weights to a CKY parser to find the best valid tree.

A further application of MBR decoding is in approximating the consensus problem when trying to sum out hidden variables. Goodman (2003), again, showed how to sum over derivation trees in a tree substitution grammar (trained, in that instance, according to the Data-Oriented Parsing framework) to find the best derived tree according to an MBR criterion. Matsuzaki et al. (2005) and Dreyer and Eisner (2006) used a similar method to parse with PCFGs with latent annotations, and Cohen and Smith (2007) parsed Hebrew while summing over morphological segmentations.

A.10 Entropy

Finally, due to the results of information theory, probabilistic models possess natural measures of the uncertainty of their predictions, known as entropy.

The most widely used such measure is **Shannon entropy** (entropy *tout court*), which is the expectation of the negative log probability:

$$\begin{aligned} H(p) &= -\sum_j p(y_j \mid x) \log p(y_j \mid x) && \text{(A.29)} \\ &= -\sum_j [p(y_j \mid x) \log e^{s_j} - p(y_j \mid x) \log Z] && \text{(A.30)} \\ &= -\sum_j p(y_j \mid x) \cdot s_j + \log Z \sum_j p(y_j \mid x) && \text{(A.31)} \\ &= -\mathbf{E}_p s + \log Z && \text{(A.32)} \end{aligned}$$

The **Rényi entropy** is a theoretically and computationally attractive generalization:

$$R_\alpha = \frac{1}{1-\alpha} \log \left(\sum_j p(y_j \mid x)^\alpha \right) \tag{A.33}$$

$$= \frac{1}{1-\alpha} \log \left(Z_i^{-\alpha} \sum_j e^{\alpha s_{ij}} \right) \tag{A.34}$$

$$= \frac{1}{1-\alpha} \left(\log \sum_j e^{\alpha s_{ij}} - \alpha \log Z_i \right) \tag{A.35}$$

It can be shown that $\lim_{\alpha \to 1} R_\alpha(p)$ is in fact the Shannon entropy $H(p)$ and that $\lim_{\alpha \to \infty} R_\alpha(p) = -\log \max_y p(y)$, i.e. the negative log probability of the modal or "Viterbi" label (Arndt, 2001; Karakos et al., 2007). The $\alpha = 2$ case, widely used as a measure of purity in decision tree learning, is often called the "Gini index." Finally, when $\alpha = 0$, we get the log of the number of possible outcomes, $\log |\mathcal{Y}|$ in our notation.

Appendix B

Algorithms for Dependency Parsing

In our discussions of dependency parsing in the body of chapter 2, and in the foregoing appendix on log-linear models, we have presented certain structured modeling problems with a blithe declarative innocence. Simply take the argmax, or the sum or expectation, and ignore the combinatorial spaces to be searched.

As noted above, the paradigm of edge-factored parsing is attractive because there exist efficient algorithms for solving these problems. As with other algorithms commonly used in natural language processing, such as the Viterbi and forward-backward algorithms for hidden Markov models or the CKY algorithm for constituency parsing, we can use dynamic programming to search the space of projective dependency trees with edge-factored weights.

In this Appendix, we review the algorithms for solving several inference problems for edge-factored dependency parsing.

B.1 Finding the Best Parse

When speaking of "parsing" proper, we usually mean solving a maximization problem to find the highest-weighted tree according to some model. With dynamic programming, we can incrementally infer the best way of building subtrees of progressively larger sizes until we find the best subtree that spans the entire input sentence.

We present the algorithm here (algorithm B.1) using a notation similar to McDonald et al. (2005a). The exponentiated weight for the dependency edge from a parent word indexed i to child j is denoted $u(i, j) = e^{s(i,j)}$. The dynamic program works by filling in cells in a chart C with subtrees of increasing width; C is indexed by a quadruple (i, j, d, c), consisting of the left and right edges of a head's span, a direction (L or R) indicating whether the head is gathering left or right children, and a Boolean for whether the subtree is *closed*, i.e., whether the head can stop taking any more children in that direction.
It is easy to see from the form of the algorithm that it takes $O(n^3)$ time: the two nested loops range over the sentence length n, as do the inner maximizations over r. In the terminology of Eisner and Satta (1999), this is a "split-head" parser, since each child is attached without considering any of its siblings, let alone siblings on the opposite side of the head. An important practical advantage of edge-factored parsing is the lack of a grammar constant, which often dominates CKY parsing (Klein and Manning, 2001).

For clarity in comparing this algorithm to related ones below, we show only the operations for computing the *weight* of the best parse tree. To extract the best tree itself, we need to record *which* split point maximized the combined weight of each pair of subtrees, i.e. an

Algorithm B.1 Find the score of the best projective tree with edge-factored weights

```
1: function EFMAX($u, n$)                          ▷ Edge weight array $u$ and sentence length $n$
2:     $C[i, i, d, c] \leftarrow 0, \forall\, i \in \{0..n\}, d \in \{L, R\}, c \in \{0, 1\}$
3:     for $width \leftarrow 1..n$ do
4:         for $i \leftarrow 0..n$ do
5:             $k \leftarrow i + width$
6:             if $k > n$ then break
7:             $C[i, k, L, 0] \leftarrow \max_{i \leq j < k}(C[i, j, R, 1] \cdot C[j + 1, k, L, 1] \cdot u(k, i))$
8:             $C[i, k, R, 0] \leftarrow \max_{i \leq j < k}(C[i, j, R, 1] \cdot C[j + 1, k, L, 1] \cdot u(i, k))$
9:             $C[i, k, L, 1] \leftarrow \max_{i \leq j < k}(C[i, j, L, 1] \cdot C[j, k, L, 0])$
10:            $C[i, k, R, 1] \leftarrow \max_{i < j \leq k}(C[i, j, R, 0] \cdot C[j, k, R, 1])$
11:        end for
12:    end for
13:    return $C$                                  ▷ Best score is $C[0, n, R, 1]$
14: end function
```

argmax_j in addition to the $\max_j$ in lines 7–10. When the algorithm terminates, we can then retrace the sequence of decisions that formed the best tree, as one follows backpointers to extract the best sequence of HMM states in the Viterbi algorithm.

Also for comparability's sake, we have presented the algorithm as computing the tree with the best product of exponentiated scores u, rather than the best sum of scores s. In practice, log-domain computations are used for this and later algorithms.

B.2 The Partition Function and Its Gradient

Searching for the best tree is common to linear learning frameworks, where it is also used during training, as well as parsing with log-linear models. The latter, however, require additional inferences for maximum likelihood (or minimum risk) training. In particular, we need to compute the normalizer, or partition function, Z_i for each input x_i and its gradient (A.21).[1]

Instead of the (score of the) best tree, the partition function requires the sum of scores of all trees. For grammars amenable to dynamic programming, this is precisely the quantity computed by the **inside algorithm**. Goodman (1999) presented the relationship between the maximization and summing algorithms as a change of **semiring**. In brief, we can derive the inside algorithm (algorithm B.2) by replacing the maximization operations in algorithm B.1 with summations.

Algorithm B.2 Find the total score of all projective trees with edge-factored weights

```
1:  function EFINSIDE($u, n$)                    ▷ Edge weight array $u$ and sentence length $n$
2:      $C[i,i,d,c] \leftarrow 0, \forall\, i \in \{0..n\}, d \in \{L,R\}, c \in \{0,1\}$
3:      for $width \leftarrow 1..n$ do
4:          for $i \leftarrow 0..n$ do
5:              $k \leftarrow i + width$
6:              if $k > n$ then break
7:              $C[i,k,L,0] \leftarrow \sum_{i \leq j < k}(C[i,j,R,1] \cdot C[j+1,k,L,1] \cdot u(k,i))$
8:              $C[i,k,R,0] \leftarrow \sum_{i \leq j < k}(C[i,j,R,1] \cdot C[j+1,k,L,1] \cdot u(i,k))$
9:              $C[i,k,L,1] \leftarrow \sum_{i \leq j < k}(C[i,j,L,1] \cdot C[j,k,L,0])$
10:             $C[i,k,R,1] \leftarrow \sum_{i < j \leq k}(C[i,j,R,0] \cdot C[j,k,R,1])$
11:         end for
12:     end for
13:     return $C$                                ▷ Total score is $C[0,n,R,1]$
14: end function
```

[1]For simple (stochastic) gradient descent algorithms (A.13), we in fact need only the gradients, which will be advantageous when the partition function and its gradient are approximated (§2.7).

Computing the expected counts of grammar rules or dependency links over all possible trees (A.21) is often performed by the **inside-outside algorithm** (Baker, 1979). A an alternative to deriving this specialized algorithm (by analogy to the forward-backward algorithm on trellises for HMM inference), we can perform automatic differentiation on the operations of the forward algorithm (Griewank, 1988; Griewank and Corliss, 1991; Pearlmutter and Siskind, 2008). In particular, we can differentiate in the reverse mode, also known as backpropagation, to get a cubic-time algorithm that computes the gradient of narrower spans from wider ones (algorithm B.3). Each of the products in B.2.7–10 becomes by the product rule two or three operations in the gradient algorithm, depending on the number of factors.

An alternative backpropagation strategy to this explicit program transformation involves the use of a "tape" to record operations on the forward pass, which is then run backwards to get the gradient (Eisner et al., 2005). If we only need the gradient with respect to a few parameters/rules/dependencies, it is more efficient to perform a single forward pass with the **expectation semiring** (Eisner, 2002).

B.3 Viterbi Outside Scores

A similar program transformation on the best-tree procedure allows us to compute the "Viterbi outside scores" or "max-marginals" (algorithm B.4). The product of the Viterbi outside score by the Viterbi inside score for a particular dependency, say $i \to j$, gives the

Algorithm B.3 Find the gradient of the partition function with respect to all edges

```
1: function EFINSIDEGRADIENT($u, C, n$)                          ▷ Chart $C$ from EFINSIDE
2:     $g(i,j) \leftarrow 0, \forall\, i,j \in \{0..n\}$
3:     $G[i,j,d,c] \leftarrow 0, \forall\, i,j \in \{0..n\}, d \in \{L,R\}, c \in \{0,1\}$
4:     $G[0,n,R,1] \leftarrow (C[0,n,R,1])^{-1}$
5:     for $width \leftarrow n..1$ do
6:         for $i \leftarrow 0..n$ do
7:             $k \leftarrow i + width$
8:             if $k > n$ then break
9:             for $j \leftarrow (i+1)..k$ do                               ▷ Backprop B.2.10
10:                $G[i,j,R,0] \leftarrow G[i,j,R,0] + G[i,k,R,1] \cdot C[j,k,R,1]$
11:                $G[j,k,R,1] \leftarrow G[j,k,R,1] + G[i,k,R,1] \cdot C[i,j,R,0]$
12:            end for
13:            for $j \leftarrow i..(k-1)$ do                                ▷ Backprop B.2.9
14:                $G[i,j,L,1] \leftarrow G[i,j,L,1] + G[i,k,L,1] \cdot C[j,k,L,0]$
15:                $G[j,k,L,0] \leftarrow G[j,k,L,0] + G[i,k,L,1] \cdot C[i,j,L,1]$
16:            end for
17:            for $j \leftarrow i..(k-1)$ do                                ▷ Backprop B.2.7
18:                $G[i,j,R,1] \leftarrow G[i,j,R,1] + G[i,k,L,0] \cdot C[j+1,k,L,1] \cdot u(k,i)$
19:                $G[j+1,k,L,1] \leftarrow G[j+1,k,L,1] + G[i,k,L,0] \cdot C[i,j,R,1] \cdot u(k,i)$
20:                $g(k,i) \leftarrow g(k,i) + G[i,k,L,0] \cdot C[i,j,R,1] \cdot C[j+1,k,L,1]$
21:            end for
22:            for $j \leftarrow i..(k-1)$ do                                ▷ Backprop B.2.8
23:                $G[i,j,R,1] \leftarrow G[i,j,R,1] + G[i,k,R,0] \cdot C[j+1,k,L,1] \cdot u(i,k)$
24:                $G[j+1,k,L,1] \leftarrow G[j+1,k,L,1] + G[i,k,R,0] \cdot C[i,j,R,1] \cdot u(i,k)$
25:                $g(i,k) \leftarrow g(i,k) + G[i,k,R,0] \cdot C[i,j,R,1] \cdot C[j+1,k,L,1]$
26:            end for
27:        end for
28:    end for
29:    return $g$
30: end function
```

score of the best tree that contains $i \to j$. We can use the outside scores of simpler models as figures of merit or A* estimates to speed best-first parsing (Klein and Manning, 2003); we can also use them as messages in max-product belief propagation with combinatorial factors (Duchi et al., 2006), which we explore further in chapter 2.

B.4 Entropy

To compute the entropy of the distribution over trees, we apply to the inside algorithm the transformation derived by Hwa (2000) for constituency parsing. As noted by Li and Eisner (2009), this forward-pass algorithm B.5, which depends on inside scores alone, is appropriate for calculating the scalar entropy.

We can see from (A.32), however, that the entropy can also be calculated from the feature expectations. For edge-factored dependency models, which have $O(n^2)$ scores $s(i, j)$, if we have already computed feature expectations using the inside-outside algorithm, it is therefore always more efficient to add up a quadratic number of scores than to run algorithm B.5 in cubic time. If we want the gradient of the entropy, however, it is more convenient to apply the gradient transformation to the forward-pass entropy algorithm.

In contrast, we do not need any new machinery to calculate the Rényi entropy and its gradient: we need only two calls to the inside-outside algorithm. We can see from (A.35) that we need to calculate $\log \sum_j e^{\alpha s_{ij}}$ and $\alpha \log Z_i$. For the former quantity, multiply all edge scores by α and then run sum all derivations; for the latter, compute Z_i as usual.

Algorithm B.4 Find the outside scores of the best tree

```
1: function EFMaxGradient($u, C, n$)                                  ▷ Chart $C$ from EFMax
2:     $g(i,j) \leftarrow 0, \forall\, i,j \in \{0..n\}$
3:     $G[i,j,d,c] \leftarrow 0, \forall\, i,j \in \{0..n\}, d \in \{L,R\}, c \in \{0,1\}$
4:     $G[0,n,R,1] \leftarrow 1$
5:     for $width \leftarrow n..1$ do
6:         for $i \leftarrow 0..n$ do
7:             $k \leftarrow i + width$
8:             if $k > n$ then break
9:             for $j \leftarrow (i+1)..k$ do
10:                $G[i,j,R,0] \leftarrow \max(G[i,j,R,0], G[i,k,R,1] \cdot C[j,k,R,1])$
11:                $G[j,k,R,1] \leftarrow \max(G[j,k,R,1], G[i,k,R,1] \cdot C[i,j,R,0])$
12:            end for
13:            for $j \leftarrow i..(k-1)$ do
14:                $G[i,j,L,1] \leftarrow \max(G[i,j,L,1], G[i,k,L,1] \cdot C[j,k,L,0])$
15:                $G[j,k,L,0] \leftarrow \max(G[j,k,L,0], G[i,k,L,1] \cdot C[i,j,L,1])$
16:            end for
17:            for $j \leftarrow i..(k-1)$ do
18:                $G[i,j,R,1] \leftarrow \max(G[i,j,R,1], G[i,k,L,0] \cdot C[j+1,k,L,1] \cdot u(k,i))$
19:                $G[j+1,k,L,1] \leftarrow \max(G[j+1,k,L,1],$
                                   $G[i,k,L,0] \cdot C[i,j,R,1] \cdot u(k,i))$
20:                $g(k,i) \leftarrow \max(g(k,i), G[i,k,L,0] \cdot C[i,j,R,1] \cdot C[j+1,k,L,1])$
21:            end for
22:            for $j \leftarrow i..(k-1)$ do
23:                $G[i,j,R,1] \leftarrow \max(G[i,j,R,1], G[i,k,R,0] \cdot C[j+1,k,L,1] \cdot u(i,k))$
24:                $G[j+1,k,L,1] \leftarrow \max(G[j+1,k,L,1],$
                                   $G[i,k,R,0] \cdot C[i,j,R,1] \cdot u(i,k))$
25:                $g(i,k) \leftarrow \max(g(i,k), G[i,k,R,0] \cdot C[i,j,R,1] \cdot C[j+1,k,L,1])$
26:            end for
27:        end for
28:    end for
29:    return $g$
30: end function
```

Algorithm B.5 Find the entropy of all projective trees with edge-factored weights

```
1: function EFENTROPY($u, C, n$)                         ▷ Chart $C$ form EFINSIDE
2:     $H[i, i, d, c] \leftarrow 0, \forall\, i \in \{0..n\}, d \in \{L, R\}, c \in \{0, 1\}$
3:     for $width \leftarrow 1..n$ do
4:         for $i \leftarrow 0..n$ do
5:             $k \leftarrow i + width$
6:             if $k > n$ then break
7:             $H[i, k, L, 0] \leftarrow \sum_{i \leq j < k}(H[i, j, R, 1] \cdot C[j + 1, k, L, 1] \cdot u(k, i)$
                       $+C[i, j, R, 1] \cdot H[j + 1, k, L, 1] \cdot u(k, i)$
                       $+C[i, j, R, 1] \cdot C[j + 1, k, L, 1] \cdot -u(k, i) \log u(k, i))$
8:             $H[i, k, R, 0] \leftarrow \sum_{i \leq j < k}(H[i, j, R, 1] \cdot C[j + 1, k, L, 1] \cdot u(i, k)$
                       $+C[i, j, R, 1] \cdot H[j + 1, k, L, 1] \cdot u(i, k)$
                       $+C[i, j, R, 1] \cdot C[j + 1, k, L, 1] \cdot -u(i, k) \log u(i, k))$
9:             $H[i, k, L, 1] \leftarrow \sum_{i \leq j < k}(H[i, j, L, 1] \cdot C[j, k, L, 0]$
                       $+C[i, j, L, 1] \cdot H[j, k, L, 0])$
10:            $H[i, k, R, 1] \leftarrow \sum_{i < j \leq k}(H[i, j, R, 0] \cdot C[j, k, R, 1]$
                       $+C[i, j, R, 0] \cdot H[j, k, R, 1])$
11:        end for
12:    end for
13:    return $H[0, n, R, 1]/C[0, n, R, 1] + \log C[0, n, R, 1]$
14: end function
```

B.5 Second-Order Gradients

Computing the gradient of scalar values that happen to use inside and outside costs is done fairly straightforwardly in cubic time. In some situations, however, we need the full Hessian.

If we apply the gradient transformation to algorithm B.3, we get a $O(n^5)$ algorithm to compute the Hessian. A tape algorithm to retrace that algorithm while maintaining the Hessian is also quintic since it needs to pass the $O(n^2)$ entries of the Hessian through each of $O(n^3)$ steps. In many applications, however, we only want $O(n)$-sized product of the Hessian times a vector, e.g., the gradient of risk. In contrast, §3.2.4 shows how to compute the Hessian for nonprojective edge-factored models in $O(n^4)$ time.

Bibliography

Steven Abney. Understanding the Yarowsky algorithm. *Computational Linguistics*, 30(3): 365–395, 2004.

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. Learning dependency translation models as collections of finite state head transducers. *Computational Linguistics*, 26(1): 45–60, 2000.

Cristoph Arndt. *Information Measures*. Springer, 2001.

L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer. A new algorithm for the estimation of hidden Markov model parameters. In *ICASSP*, pages 493–496, 1988.

James K. Baker. Trainable grammars for speech recognition. In *Proceedings of the Acoustical Society of America*, pages 547–550, 1979.

Peter Bartlett, Michael Collins, Ben Taskar, and David McAllester. Exponentiated gradient algorithms for large-margin structured classification. In *NIPS*, 2004.

Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI, 2003.

Kedar Bellare and Andrew McCallum. Generalized expectation criteria for bootstrapping extractors using record-text alignment. In *EMNLP*, pages 131–140, 2009.

Michael Bendersky, W. Bruce Croft, and David A. Smith. Structural annotation of search queries using pseudo-relevance feedback. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, 2010.

Christian Bessière and Jean-Charles Régin. Arc consistency for general constraint networks: preliminary results. In *IJCAI*, pages 398–404, 1997.

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. The PDT: A 3-level annotation scenario. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*, chapter 7. Kluwer, 2003.

Léon Bottou. Stochastic learning. In *Advanced Lectures in Machine Learning*, pages 146–168. Springer, 2003.

Alexandre Bouchard-Côté and Michael I. Jordan. Variational inference over combinatorial spaces. In *NIPS*, 2010.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER treebank. In *TLT*, 2002.

Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Frederik Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, 1990.

Matthias Buch-Kromann. *Discontinuous Grammar. A Model of Human Parsing and Language Acquisition"*. Dr.ling.merc. dissertation, Copenhagen Business School, 2006.

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *CoNLL*, pages 149–164, 2006.

Razvan Bunescu and Raymond Mooney. A shortest path dependency kernel for relation extraction. In *HLT-EMNLP*, pages 724–731, 2005.

David Burkett and Dan Klein. Two languages are better than one (for syntactic parsing). In *EMNLP*, pages 877–886, 2008.

David Burkett, John Blitzer, and Dan Klein. Joint parsing and alignment with weakly synchronized grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–135, 2010.

Paolo M. Camerini, Luigi Fratta, and Francesco Maffioli. The k-best spanning arborescences of a network. *Networks*, 10:91–110, 1980.

Sharon A. Caraballo and Eugene Charniak. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298, 1998.

Glenn Carroll and Eugene Charniak. Two experiments on learning probabilistic dependency grammars from corpora. Technical report, Brown University, 1992.

S. Chaiken. A combinatorial proof of the all minors matrix tree theorem. *SIAM Journal on Algebraic and Discrete Methods*, 3(3):319–329, 1982.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*, pages 173–180, 2005.

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. Multilevel coarse-to-fine PCFG parsing. In *HLT-NAACL*, pages 168–175, June 2006.

W.-K. Chen. Topological analysis for active networks. *IEEE Transactions on Circuit Theory*, 12(1):85–91, 1965.

Joan Chen-Main. *On the Generation and Linearization of Multi-Dominance Structures.* PhD thesis, Johns Hopkins University, 2006.

David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2): 201–228, 2007.

David Chiang and Daniel M. Bikel. Recovering latent information in treebanks. In *COLING 2002: The 17th International Conference on Computational Linguistics*, pages 183–189, Taipei, 2002.

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

Montserrat Civit Torruella and M^{a} Antònia Martí Antonín. Design principles for a Spanish treebank. In *TLT*, 2002.

Shay B. Cohen and Noah A. Smith. Joint morphological and syntactic disambiguation. In *EMNLP-CoNLL*, pages 208–217, 2007.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Madrid, 1997. Association for Computational Linguistics.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, July 2002.

Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL*, pages 263–270, 2002.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *JMLR*, 2003.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-agressive algorithms. In *JMLR*, 2006.

Gregory Crane. Generating and parsing classical Greek. *Literary and Linguistic Computing*, 6(4):243–245, 1991.

Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *ACL*, pages 423–429, 2004.

Dipanjan Das and Noah A. Smith. Paraphrase identification as probabilistic quasi-synchronous recognition. In *ACL*, pages 468–476, 2009.

Hal Daumé, III. Frustratingly easy domain adaptation. In *ACL*, pages 256–263, 2007.

Ralph Debusmann. *Extensible Dependency Grammar: A Modular Grammar Formalism Based on Multigraph Description*. PhD thesis, Universität des Saarlandes, 2006.

Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

Edsgar W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

Bonnie J. Dorr. Machine translation divergences: A formal description and proposed solution. *Computational Linguistics*, 20(4):597–633, 1994.

Mark Dredze, John Blitzer, Partha Pratim Talukdar, Kuzman Ganchev, João Graca, and Fernando Pereira. Frustratingly hard domain adaptation for dependency parsing. In

Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007, pages 1051–1055, 2007.

Markus Dreyer and Jason Eisner. Better informed training of latent syntactic features. In *EMNLP*, pages 317–326, 2006.

Gregory Druck and David A. Smith. Computing conditional feature covariance under non-projective tree conditional random fields. Technical Report UM-CS-2009-060, University of Massachusetts, 2009.

Gregory Druck, Gideon Mann, and Andrew McCallum. Semi-supervised learning of dependency parsers using generalized expectation criteria. In *ACL*, pages 360–368, 2009.

John Duchi, Daniel Tarlow, Gal Elidan, and Daphne Koller. Using combinatorial optimization within max-product belief propagation. In *NIPS*, 2006.

Denys Duchier. Configuration of labeled trees under lexicalized constraints and principles. *Research on Language and Computation*, 1:307–336, 2003.

Denys Duchier and Ralph Debusmann. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 180–187, Toulouse, 2001.

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

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *ACL*, pages 1–8, 2002.

Jason Eisner. Three new probabilistic models for dependency parsing: An exploration. In *COLING*, pages 340–345, 1996.

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 and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 457–480, University of Maryland, 1999.

Jason Eisner and Noah A. Smith. Parsing with soft and hard constraints on dependency length. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 30–41, Vancouver, British Columbia, October 2005. Association for Computational Linguistics.

Jason Eisner, Eric Goldlust, and Noah A. Smith. Compiling comp ling: Weighted dynamic programming and the Dyna language. In *HLT-EMNLP*, pages 281–290, 2005.

Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *UAI*, 2006.

Yehuda N. Falk. *Lexical-Functional Grammar: An Introduction to Parallel Constraint-Based Syntax*. CSLI, 2001.

Jenny Rose Finkel and Christopher D. Manning. Nested named entity recognition. In *EMNLP*, pages 141–150, 2009.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *ACL*, pages 363–370, 2005.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *ACL*, pages 959–967, 2008.

Kilian A. Foth, Tomas By, and Wolfgang Menzel. Guiding a constraint dependency parser with supertags. In *ACL*, pages 289–296, 2006.

Heidi Fox. Phrasal cohesion and statistical machine translation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 304–3111, July 2002.

Robert Frank. *Phrase Structure Composition and Syntactic Dependencies*. MIT Press, 2002.

William A. Gale, Kenneth W. Church, and David Yarowsky. One sense per discourse. In *HLT*, pages 233–237, 1992.

Michel Galley and Christopher D. Manning. Quadratic-time dependency parsing for machine translation. In *ACL*, pages 773–781, 2009.

Kuzman Ganchev and Mark Dredze. Small statistical models by random feature mixing. In *ACL Workshop on Mobile Language Processing*, pages 19–20, 2008.

Kuzman Ganchev, Jennifer Gillenwater, and Ben Taskar. Dependency grammar induction via bitext projection constraints. In *ACL*, pages 369–377, 2009.

Stuart Geman and Mark Johnson. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *ACL*, pages 279–286, 2002.

Leonidas Georgiadis. Arborescence optimization problems solvable by Edmonds' algorithm. *Theoretical Computer Science*, 301(1–3):427–437, 2003.

Daniel Gildea. Optimal parsing strategies for linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 769–776, 2010.

Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 512–520, Hong Kong, 2000.

Kevin Gimpel and Noah A. Smith. Feature-rich translation by quasi-synchronous lattice parsing. In *EMNLP*, pages 219–228, 2009.

Vaibbhava Goel and William J. Byrne. Minimum Bayes risk automatic speech recognition. *Computer Speech and Language*, 14(2):115–135, 2000.

Yoav Goldberg and Reut Tsarfaty. A single generative model for joint morphological segmentation and syntactic parsing. In *ACL*, pages 371–379, 2008.

Sharon Goldwater and Mark Johnson. Learning OT constraint rankings using a maximum entropy model. In *Proceedings of the Stockholm Workshop on Variation within Optimality Theory*, pages 111–120, 2003.

Joshua Goodman. Efficient parsing of DOP with PCFG-reductions. In Rens Bod, Remko Scha, and Khalil Sima'an, editors, *Data-Oriented Parsing*, chapter 8, pages 125–146. CSLI, Stanford, 2003.

Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–606, 1999.

Joshua T. Goodman. Parsing algorithms and metrics. In *ACL*, pages 177–183, 1996.

Andreas Griewank. On automatic differentiation. Technical Report ANL/MCS-P10-1088, Argonne National Laboratory, 1988.

Andreas Griewank and George F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, 1991.

Kadri Hacioglu. Semantic role labeling using dependency trees. In *COLING*, pages 1273–1276, 2004.

Jan Hajič, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. Prague Arabic dependency treebank: Development in data and tools. In *Proc. of the NEMLAR Intern. Conf. on Arabic Language Resources and Tools*, 2004.

Johan Hall, Joakim Nivre, and Jens Nilsson. Discriminative classifiers for deterministic dependency parsing. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 316–323, 2006.

Keith Hall. K-best spanning tree parsing. In *ACL*, pages 392–399, 2007.

Mary P. Harper, Randall A. Helzerman, Carla B. Zoltowski, Boon-Lock Yeo, Yin Chan, Todd Stewart, and Bryan L. Pellom. Implementation issues in the development of the PARSEC parser. *Software—Practice and Experience*, 25(8):831–862, August 1995.

Refael Hassin and Shlomi Rubenstein. Approximations for the maximum acyclic subgraph problem. *Information Processing Letters*, 51:133–140, 1994.

Bruce Hayes and Colin Wilson. A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*, 39:379–440, 2008.

Kristy Hollingshead and Brian Roark. Pipeline iteration. In *ACL*, pages 952–959, 2007.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594, 2008.

Liang Huang, Wenbin Jiang, and Qun Liu. Bilingually-constrained (monolingual) shift-reduce parsing. In *EMNLP*, pages 1222–1231, 2009.

Richard A. Hudson. *Language Networks: The New Word Grammar*. Oxford University Press, 2007.

Rebecca Hwa. Sample selection for statistical grammar induction. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 45–52, Hong Kong, China, October 2000. Association for Computational Linguistics.

Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. Bootstrapping parsers via syntactic projection across parallel texts. *Natural Language Engineering*, 11:311–325, 2005.

Neil Immerman. *Descriptive Complexity*. Springer, 1999.

T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *NIPS*, 1999.

Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford University Press, 2002.

Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. Estimators for stochastic 'unification-based' grammars. In *Proceedings of the 37th Annual Meeting of* *the Association for Computational Linguistics*, pages 535–549, University of Maryland, 1999.

Aravind Joshi, K. Vijay-Shanker, and David Weir. The convergence of mildly context-sensitive grammar formalisms. In Peter Sells, Stuart Shieber, and Thomas Wasow, editors, *Foundational Issues in Natural Language Processing*, pages 31–81. MIT Press, 1991.

Florent Jousse, Rémi Gilleron, Isabelle Tellier, and Marc Tommasi. Conditional random fields for XML trees. In *Proc. ECML Workshop on Mining and Learning in Graphs*, 2006.

Damianos Karakos, Jason Eisner, Sanjeev Khudanpur, and Carey Priebe. Cross-instance tuning of unsupervised document clustering algorithms. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 252–259, Rochester, New York, April 2007. Association for Computational Linguistics.

R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

Dan Klein and Christopher Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *ACL*, pages 478–485, 2004.

Dan Klein and Christopher D. Manning. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn Treebank. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 338–345, Toulouse, 2001.

Dan Klein and Christopher D. Manning. A generative constituent-context model for improved grammar induction. In *ACL*, pages 128–135, 2002.

Dan Klein and Christopher D. Manning. A* parsing: Fast exact Vitberi parse selection. In *HLT-NAACL*, pages 40–47, 2003.

Philipp Koehn. Europarl: A multilingual corpus for evaluation of machine translation. http://www.iccs.informatics.ed.ac.uk/~pkoehn/publications/europarl.ps, 2002.

Terry Koo and Michael Collins. Hidden-variable models for discriminative reranking. In *HLT-EMNLP*, pages 507–514, 2005.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. Structured prediction models via the matrix-tree theorem. In *EMNLP-CoNLL*, pages 141–150, 2007.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *EMNLP*, pages 1288–1298, 2010.

Matthias T. Kromann. The Danish dependency treebank and the underlying linguistic theory. In Joakim Nivre and Erhard Hinrichs, editors, *TLT*, 2003.

Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

Marco Kuhlmann and Mathias Möhl. Mildly context-sensitive dependency languages. In *ACL*, pages 160–167, 2007.

Marco Kuhlmann and Joakim Nivre. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 507–514, 2006.

Jonas Kuhn. *Optimality-theoretic Syntax: A Declarative Approach.* CSLI, 2003.

Jonas Kuhn. Experiments in parallel-text based grammar induction. In *ACL*, pages 470–477, 2004.

Shankar Kumar and William Byrne. Minimum Bayes-risk word alignments of bilingual texts. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 140–147, July 2002.

Shankar Kumar and William Byrne. Minimum Bayes-risk decoding for statistical machine translation. In *HLT-NAACL*, pages 169–176, 2004.

Shankar Kumar, Wolfgang Macherey, Chris Dyer, and Franz Och. Efficient minimum error rate training and minimum bayes-risk decoding for translation hypergraphs and lattices. In *ACL*, pages 163–171, 2009.

Simon Lacoste-Julien, Ben Taskar, Dan Klein, and Michael I. Jordan. Word alignment via quadratic assignment. In *HLT-NAACL*, pages 112–119, June 2006.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

Géraldine Legendre, Jane Grimshaw, and Sten Vikner, editors. *Optimality-theoretic Syntax*. MIT Press, 2001.

Zhifei Li and Jason Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *EMNLP*, pages 40–51, 2009.

Percy Liang, Ben Taskar, and Dan Klein. Alignment by agreement. In *HLT-NAACL*, pages 104–111, June 2006.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming B*, 45(3):503–528, 1989.

David MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge, 2003.

Alan Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

David M. Magerman. Statistical Decision-Tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, Mass, 1995.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

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.

Andre Martins, Noah Smith, and Eric Xing. Concise integer linear programming formulations for dependency parsing. In *ACL*, pages 342–350, 2009a.

André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. Stacking dependency parsers. In *EMNLP*, pages 157–166, 2008.

André F. T. Martins, Noah A. Smith, and Eric P. Xing. Polyhedral outer approximations with application to natural language parsing. In *ICML*, 2009b.

Hiroshi Maruyama. Structural disambiguation with constraint propagation. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 31–38, University of Pittsburgh, Pittsburgh, Pennsylvania, USA, 1990.

Takuya Matsuzaki, Yusuke Miyao, and Junichi Tsujii. Probabilistic CFG with latent annotations. In *ACL*, pages 75–82, 2005.

John T. Maxwell, III and Ronald M. Kaplan. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590, 1993.

David McAllester, Michael Collins, and Fernando Pereira. Case-factor diagrams for structured probabilistic modeling. In *UAI*, pages 382–391, 2004.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *ICML*, 2000.

David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *HLT-NAACL*, pages 152–159, June 2006.

Patrick McCrae, Kilian A. Foth, and Wolfgang Menzel. Modelling global phenomena with extended local constraints. In *Proceedings of the 5th International Workshop on Constraints and Language Processing*, Hamburg, Germany, August 2008.

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

Ryan McDonald and Giorgio Satta. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 121–132, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *ACL*, pages 91–98, 2005a.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *HLT-EMNLP*, pages 523–530, 2005b.

Igor A. Mel'čuk. *Dependency Syntax: Theory and Practice*. SUNY Press, 1988.

M. Minoux. A generalization of the all minors matrix tree theorem to semirings. *Discrete Mathematics*, 199:139–150, 1999.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT)*, pages 149–160, 2003.

Joakim Nivre and Ryan McDonald. Integrating graph-based and transition-based dependency parsers. In *ACL*, pages 950–958, 2008a.

Joakim Nivre and Ryan McDonald. Integrating graph-based and transition-based dependency parsers. In *ACL*, 2008b.

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 2007*, pages 915–932, 2007.

Franz Josef Och. Minimum error rate training in statistical machine translation. In *ACL*, pages 160–167, 2003.

Joe Pater, David A. Smith, Robert Staubs, Karen Jesney, and Ramgopal Mettu. Learning hidden structure with a log-linear model of grammar. In *Linguistic Society of America (LSA)*, Baltimore, January 2010.

Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems*, 30(2), 2008.

Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April 2007. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *ACL*, pages 433–440, 2006.

Christopher Potts and Geoffrey K. Pullum. Model theory and the content of OT constraints. *Phonology*, 19:361–393, 2002.

Detlef Prescher. Head-driven PCFGs with latent-head statistics. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 115–124, Vancouver, British Columbia, October 2005. Association for Computational Linguistics.

Alan Prince and Paul Smolensky. *Optimality Theory: Constraint Interaction in Generative Grammar*. Blackwell, 2004.

Geoffrey K. Pullum. The evolution of model-theoretic frameworks in linguistics. In *Model Theoretic Syntax at 10*, pages 1–10, 2007.

Geoffrey K. Pullum. On two recent attempts to show that English is not a CFL. *Computational Linguistics*, 10(3-4):182–186, 1984.

Chris Quirk, Arul Menezes, and Colin Cherry. Dependency treelet translation: Syntactically informed phrasal SMT. In *ACL*, pages 271–279, 2005.

Ajit Rao and Kenneth Rose. Deterministically annealed design of hidden Markov model speech recognizers. *IEEE Transactions on Speech and Audio Processing*, 9(2):111–126, 2001.

Adwait Ratnaparkhi, Salim Roukos, and R. Todd Ward. A maximum entropy model for parsing. In *ICSLP*, 1994.

Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. In *ACL*, pages 41–47, 2002.

Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI*, pages 362–367, 1994.

Sebastian Riedel and James Clarke. Incremental integer linear programming for non-projective dependency parsing. In *EMNLP*, pages 129–137, 2006.

Sebastian Riedel and David A. Smith. Relaxed marginal inference and its application to dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 760–768, 2010.

Sebastian Riedel, David A. Smith, and Andrew McCallum. Inference by minimizing size, divergence, or their sum. In *UAI*, 2010.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 480–487, Hong Kong, 2000.

Brian Roark. Markov parsing: Lattice rescoring with a statistical parser. In *ACL*, pages 287–294, 2002.

Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *ACL*, pages 47–54, 2004.

James Rogers. A Model-Theoretic framework for theories of syntax. In *ACL*, pages 10–16, 1996.

James Rogers. "Grammarless" phrase structure grammar. *Linguistics and Philosophy*, 20: 721–746, 1997.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *EMNLP*, pages 1–11, 2010.

Jerrold M. Sadock. *Autolexical Grammar: A Theory of Parallel Grammatical Representations.* University of Chicago Press, 1991.

Ivan A. Sag, Thomas Wasow, and Emily M. Bender. *Syntactic Theory: A Formal Introduction.* CSLI, 2004.

Kenji Sagae, Yusuke Miyao, and Jun'ichi Tsujii. HPSG parsing with shallow dependency constraints. In *ACL*, pages 624–631, 2007.

Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *NIPS*, pages 1185–1192. MIT Press, Cambridge, MA, 2005.

Jangwon Seo, W. Bruce Croft, and David A. Smith. Online community search using thread structure. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 1907–1910, 2009.

Libin Shen, Jinxi Xu, and Ralph Weischedel. A new string-to-dependency machine translation algorithm with a target dependency language model. In *ACL*, pages 577–585, 2008.

Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann. Math. Stat. 21*, 21:124–127, 1950.

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

Stuart M. Shieber and Yves Schabes. Synchronous tree-adjoining grammars. In *COLING*, volume 3, pages 253–258, 1990.

Daniel Sleator and Davy Temperley. Parsing English with a link grammar. In *IWPT*, pages 277–291, August 1993.

David A. Smith and Jason Eisner. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 787–794, 2006a.

David A. Smith and Jason Eisner. Bootstrapping feature-rich dependency parsers with entropic priors. In *EMNLP-CoNLL*, pages 667–677, 2007.

David A. Smith and Jason Eisner. Dependency parsing by belief propagation. In *EMNLP*, pages 145–156, 2008.

David A. Smith and Jason Eisner. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the HLT-NAACL Workshop on Statistical Machine Translation*, pages 23–30, 2006b.

David A. Smith and Noah A. Smith. Bilingual parsing with factored estimation: Using English to parse Korean. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 49–56, July 2004.

David A. Smith and Noah A. Smith. Probabilistic models of nonprojective dependency trees. In *EMNLP-CoNLL*, pages 132–140, 2007.

Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *ACL*, pages 354–362, 2005a.

Noah A. Smith and Jason Eisner. Guiding unsupervised grammar induction using contrastive estimation. In *International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Grammatical Inference Applications*, Edinburgh, July 2005b.

Noah A. Smith and Jason Eisner. Annealing structural bias in multilingual weighted grammar induction. In *ACL*, pages 569–576, 2006c.

Noah A. Smith and Mark Johnson. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4):477–491, 2007.

Noah A. Smith, David A. Smith, and Roy W. Tromble. Context-based morphological disambiguation with random fields. In *HLT-EMNLP*, pages 475–482, 2005.

Benjamin Snyder, Tahira Naseem, and Regina Barzilay. Unsupervised multilingual grammar induction. In *ACL*, pages 73–81, 2009.

Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

Charles Sutton and Andrew McCallum. Collective segmentation and labeling of distant entities in information extraction. In *ICML Workshop on Statistical Relational Learning*, 2004.

Charles Sutton and Andrew McCallum. Improved dynamic schedules for belief propagation. In *UAI*, 2007.

Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *ICML*, 2004.

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

Ben Taskar. *Learning Structured Prediction Models: A Large Margin Approach.* PhD thesis, Stanford, 2004.

Ben Taskar, Lacoste-Julien Simon, and Klein Dan. A discriminative matching approach to word alignment. In *HLT-EMNLP*, pages 73–80, 2005.

Lucien Tesnière. *Éléments de Syntaxe Structurale*. Klincksieck, 2. edition, 1969.

Naftali Tishby, Fernando Pereira, and William Bialek. The information bottleneck method. In *Proceedings of the 37th Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*, pages 173–180, 2003.

Kristina Toutanova, Aria Haghighi, and Christopher Manning. Joint learning improves semantic role labeling. In *ACL*, pages 589–596, 2005.

Roy Tromble and Jason Eisner. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *HLT-NAACL*, pages 423–430, June 2006.

Roy Tromble, Shankar Kumar, Franz Och, and Wolfgang Macherey. Lattice Minimum Bayes-Risk decoding for statistical machine translation. In *EMNLP*, pages 620–629, 2008.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In *ACL*, pages 477–485, 2009.

W. T. Tutte. *Graph Theory*. Addison-Wesley, Menlo Park, CA, 1984.

W. T. Tutte. The dissection of equilateral triangles into equilateral triangles. *Proceedings of the Cambridge Philosophical Society*, 44:463–482, 1948.

Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. The Alpino dependency treebank. In *CLIN*. Rodopi, 2002.

Paul Viola and Mukund Narasimhan. Learning to extract information from semi-structured text using a discriminative context free grammar. In *SIGIR*, pages 330–337, 2005.

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

Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *COLING*, pages 836–841, 1996.

Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. What is the Jeopardy model? a quasi-synchronous grammar for QA. In *EMNLP-CoNLL*, pages 22–32, 2007.

Wen Wang. *Statistical Parsing and Language Modeling Based on Constraint Dependency Grammar*. PhD thesis, Purdue University, 2003.

Wen Wang and Mary P. Harper. The superarv language model: Investigating the effectiveness of tightly integrating multiple knowledge sources. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 238–247, July 2002.

Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120, 2009.

Yair Weiss and William T. Freedman. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47, 2001.

Kristian Woodsend, Yansong Feng, and Mirella Lapata. Title generation with quasi-synchronous grammar. In *EMNLP*, pages 513–523, 2010.

Fei Xia, Owen Rambow, Rajesh Bhatt, Martha Palmer, and Dipti Misra Sharma. Towards a multi-representational treebank. In *International Workshop on Treebanks and Linguistic Theories (TLT)*, pages 159–170, 2009.

Xiaobing Xue, W. Bruce Croft, and David A. Smith. Query reformulation using query distributions. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, 2010.

Limin Yao, Sebastian Riedel, and Andrew McCallum. Collective cross-document relation extraction without labelled data. In *EMNLP*, pages 1013–1023, 2010.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Bethe free energy, Kikuchi approximations, and belief propagation algorithms. In *NIPS*, 2000.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief approximation algorithms. MERL TR2004-040, Mitsubishi Electric Research Laboratories, 2004.

Anssi Yli-Jyrä. Multiplanarity – a model for dependency structures in treebanks. In *Second Workshop on Treebanks and Linguistic Theories*, 2003.

Vita

David Arthur Smith received an A.B. *summa cum laude* in Classics (Greek) from Harvard University in 1994. After working as head programmer for the Perseus Digital Library Project at Tufts University, where he conducted research in information extraction and retrieval, he entered the Ph.D. program in Computer Science at Johns Hopkins University in 2002. Since 2008, he has been a Research Assistant Professor in the Department of Computer Science at the University of Massachusetts, Amherst.