Prioritization and Pruning: Efficient Inference with Weighted Context-Free Grammars

Nathan Matthew Bodenstab
B.A., George Fox University, 2004
M.S., Oregon Health & Science University, 2006

Presented to the Center for Spoken Language Understanding
and the Oregon Health & Science University
School of Medicine
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
in
Computer Science & Engineering

July 2012

Center for Spoken Language Understanding
School of Medicine
Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is to certify that the Ph.D. dissertation of
Nathan Matthew Bodenstab
has been approved.

Brian Roark, Thesis Advisor
Associate Professor, OHSU

Richard Sproat
Professor and Associate Director, OHSU

Peter Heeman
Research Associate Professor, OHSU

Izhak Shafran
Assistant Professor, OHSU

Keith Hall
Research Scientist, Google Inc.

Dedication

To my family

Acknowledgments

Obviously I need to lead off with thanking my advisor, Brian Roark. This thesis would not be possible without him. He has given me excellent guidance throughout my years at OGI/OHSU and it has been an honor to study under him. Brian struck the perfect balance of giving me the freedom to follow my own research interests, yet reining me in at times to ensure I had concrete and attainable goals. He taught me to think critically, endeared me to parsing, and filled my brain with more facts about the Portland Trail Blazers than I ever wanted to know. It was a fun few years, and I am deeply appreciative to all that he taught me. Thanks, Brian.

Second, I am forever indebted to my undergraduate advisor, David Hansen. David spent countless hours (and green pens) pouring over my code and teaching me to be a computer scientist. His encouragement was the only reason I pursued grad school. When I'm old, I'll reminisce about Dr. Hansen and think "he was the teacher that changed the coarse of my academic life."

Third, I am grateful to my thesis committee – Richard Sproat, Peter Heeman, Zak Shafran, and Keith Hall. Their thoughtful questions and comments have vastly improved the quality of this thesis. Conversations with each of my committee members have been both enlightening and challenging, and I thank them for that.

Of course most of my day-to-day research time was spent with the colorful students of CSLU. I want to especially thank my close collaborators Aaron Dunlop and Kristy Hollingshead. Kristy blazed the trail into Ph.D. land for which I am most grateful. Without her vast knowledge of LaTeX and the Charniak parser, I would have been at school for at least another nine months. Aaron's tutelage of best practices in software engineering, version control, and the intricacies of the Java virtual machine were fended off for as long as possible. But after succumbing to the unsurmountable pressure, I followed suit and

soon came to realize that he may have taught me some of the most practical CS knowledge yet. More importantly, Aaron's keen insight into all things NLP has greatly increased the quality of my research and I am truly grateful for all of our discussions over the years.

To the rest of my friends at CSLU, thank you for making grad school fun. Thanks Rachel for the movie nights, motorcycle rides, and dance party accessories you let me borrow; thanks Fran for the top-bunk discussions on soccer, lady-trees, and text-to-speech evaluations; thanks Akiko for the haircuts and lessons on proper sushi etiquette; thanks Umut for the untold hours of Dr. Mario amidst optimization homework; thanks Katja for the enriching discussions on science, religion, and shrimps; thanks Jim for sharing your knowledge of ray tracing, telescopes, and cigar box guitars (as well as your love of garage sales); thanks Nuance and Google for all the free pizza; thanks Pat for taking the brunt of the red tape; thanks Emily, Meg, Andrew, Russ, Mahsa, Masoud, Steven, and Christian for enlightening me on many interesting topics outside my small world of parsing.

And finally, thanks to my family. My mom and dad who taught me how to work hard and think for myself. You're the best parents a guy could ask for. Thanks to my siblings Nikki, Laura, and Jake for all your snide comments and hilarious jokes. And saving the best for last, I never would have finished without my beautiful wife Sharon. Your encouragement and patients were never ending. For that (and countless other things) you have my love and my thanks.

Contents

List of Tables

List of Figures

Abstract

Efficient Inference with Weighted Context-Free Grammars

Nathan Matthew Bodenstab

Doctor of Philosophy
the Center for Spoken Language Understanding
and the Oregon Health & Science University
School of Medicine

July 2012
Thesis Advisor: Brian Roark

This thesis addresses the problem of efficient search within the exponential space of possible parse trees generated by a weighted context-free grammar. Four novel probabilistic methods are presented to prioritize and prune the search space, all of which are agnostic to the structure and linguistic annotations of the underlying grammar. Each of the four models are conditioned on lexical information from the input sentence, and we argue that these cues are vital to effectively guide the search through the large solution space. We present empirical results for each of the four efficient search methods applied to multiple parsing architectures, including exact search with the CKY dynamic programming algorithm, globally prioritized search with a best-first graph-based algorithm, and inexact pruned beam-search. The combination of all four efficient search methods results in the fastest reported parsing time for high-accuracy English, Chinese, and German constituent parsing, at over 1,500 words per second with no F_1 labeled accuracy loss relative to the

maximum likelihood solution. Observed run-time complexity for a sentence of length N is reduced from $O(N^3)$ to $O(N^{1.5})$ and our efficient search methods are over an order of magnitude faster than multi-level coarse-to-fine pruning.

Chapter 1

Introduction

Building machines that can understand spoken and written natural language remains a hallmark goal of artificial intelligence. Science fiction characters from HAL 9000 to Star Wars' C3PO have depicted a future in which machine-human communication is not only possible, but such interactions are as natural as speaking with another human. For decades, a vast body of research spanning multiple disciplines is steadily turning these science fiction dreams into reality. This thesis is one step towards that goal.

Natural language processing (NLP) and natural language understanding (NLU) are two subfields of artificial intelligence where the research of this thesis is situated. Initial work in these areas was popularized in the 1950s following Alan Turing's challenge [172] to create a computer program that can successfully impersonate a human during a real-time conversation. Some progress has been made in the past 60 years towards solving the Turing Test with programs such as ELIZA [177], Jabberwacky [27], and A.L.I.C.E [175], but such programs achieve their success in superficial ways with very little "understanding" of human language. Computational models to understand and generate natural language as well as humans still remains an open problem, and introducing syntax may provide a vital piece of information to better assist machines in conquering language.

With the advent of the Internet and the availability of large corpora and computing resources in the 1990s, many in the NLP community have migrated away from hand-crafted linguistic models of language to statistical models induced from data. Under this paradigm, model parameters are learned from manually annotated data sets, which (when made public) are ideal for comparing the utility of competing research methods.

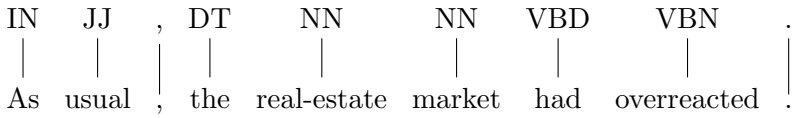
Statistical models trained from large corpora can also decrease development cost: hand-crafted grammar development can take years to perfect (20 years in the case of Riezler et al. [142]) compared to just minutes when induced from data. Although corpus creation is also very resource intensive, once created it provides an experimental spring-board, where evaluating the utility of a novel idea can be done quickly and efficiently. Statistical models of natural language have also produced more accurate and robust algorithms for many NLP applications, including speech recognition [138, 90], machine translation [163, 22], and question-answering [59]. In this thesis, all of the models we present are trained from data.

The standard for which a computer is deemed "language proficient" is difficult to define and often ignored in the NLP community in favor of more attainable goals, such as accuracy on standardized tasks. Some of these NLP tasks are end-to-end systems, such as *machine translation* (translating text from one language to another) and *question answering* (answering questions communicated in natural language), while other NLP tasks are modular in nature and used in a processing pipeline to accumulate disparate linguistic meta-information of the natural language input for down-stream processing. Canonical examples of these latter NLP tasks can be seen in Figure 1.1, and include *part-of-speech tagging*, *noun-phrase chunking*, *dependency parsing*, and *constituent parsing*. Each task predicts a form of syntactic structure of the input sentence.

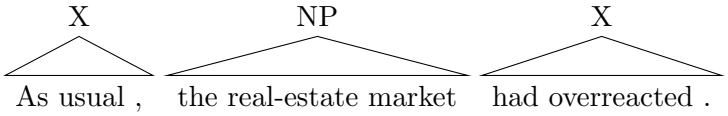
The first task, depicted in Figure 1.1a, is part-of-speech (POS) tagging, which assigns a single POS tag to each word from a set of pre-defined tags.[1] These tags disambiguate the syntactic and/or morphological properties of each word as many words may serve different functions in different contexts. For example, the word "box" can be a noun or a verb depending on how it is used in a sentence. Labeling this information explicitly with POS tagging can aid other NLP components in processing the text more accurately.

Figure 1.1b shows an example of the NLP task noun-phrase chunking (a form of shallow parsing), which identifies selected constituent groups but does not specify their internal structure or relation with other constituents within the sentence. The goal of this task is

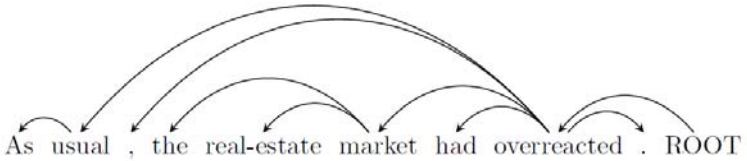
[1] Throughout this thesis will be using the POS and phrase-level tags from the Penn Treebank [114]. A complete list of tags and tag descriptions can be found in Appendix A



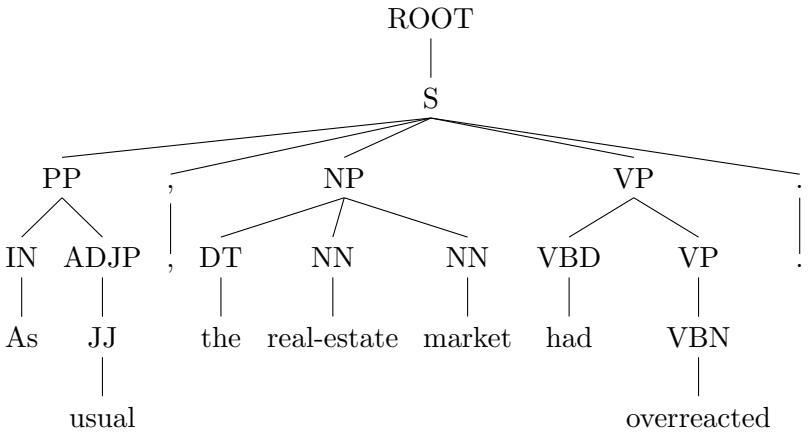
(a) Part-of-speech tagging



(b) Noun-phrase chunking



(c) Dependency parsing



(d) Constituent parsing

Figure 1.1: Canonical NLP tasks incorporating syntax.

to extract all noun-phrases from a given sentence, which, like POS tagging, is often ambiguous given the nature of natural language. Correctly identifying noun phrases can be critical for larger NLP tasks such as question answering, coreference resolution, and document summarization. As we will see in Chapter 5, similar methods have been used in a parsing pipeline to prune the search for full context-free grammar parsing. Such a pipeline architecture is appealing because shallow chunking can be performed accurately and efficiently in linear time, and provide structural cues to prune the more computationally intensive downstream constituent parsing task.

Figure 1.1c depicts the dependency structure for the example sentence. Dependency parsing is a form of syntactic analysis that labels directed relationships between two words in a sentence. The head of the dependency arc determines the syntactic category of the child; the closer a word is to the root of the tree, the more syntactically important it typically is. For example, the word "overreacted" is the main verb of the sentence and drives the subcategorization structure. The bi-lexical arcs in Figure 1.1c can optionally be labeled with a syntactic function, such as "noun modifier" or "verb's subject", but unlabeled dependency parsing is also a well-studied task as the dependency arcs alone provide significant syntactic disambiguation.[2]

Finally, in Figure 1.1d we see the constituent parsing structure for our example sentence. Here, our task is to hierarchically label constituents (groups of words) with their syntactic function. Prior work incorporating parse structure into machine translation [37] and semantic role labeling [169, 137] indicate that such hierarchical structure can have great benefit over shallow labeling techniques like chunking and part-of-speech tagging. Constituent parsing is also valuable because once a parse tree is established, a number of alternative linguistic forms can be derived: POS tags, noun-phrase chunks, and dependency relations can all be extracted from a constituent parse tree with minimal effort.[3] In essence, solving the constituent parsing problem simultaneously solves a number of related

[2]Given the unlabeled dependency arcs and POS tags, labeling the dependency graph with syntactic relationships is straight-forward and can be done with high accuracy. This is done as a post-processing step in many dependency parsers [120].

[3]Projective dependency structure can be extracted from a constituent tree by way of head-percolation rules as a post-processing step. For example, see [85].

NLP tasks. What makes constituent parsing formidable is that the number of possible derivations is exponential in the length of the sentence. *Efficiently* finding the most likely constituent parse tree from this exponential set is the central theme of this thesis.

The algorithms we investigate are of wide utility to a range of important NLP applications. For example, machine translation decoding with hierarchical models centrally involves parsing the source language string with a synchronous context-free grammar [183, 52]. Context-free parsing is also used as a backbone to prune higher-complexity grammar formalisms, such as tree adjoining grammar [75] and head driven phrase structure grammar [168][4]. Within the biological literature, weighted grammars have been used to successfully model the hierarchical folding patterns of protein [79]. The algorithms we present in this thesis will lead to a direct efficiency improvement in each of these applications. More generally, any NLP pipeline where context-free parsing is the computational bottleneck stands to gain significant improvement in the overall runtime of the system. As we will discuss in Section 1.1 and demonstrate in Chapter 4, exhaustive parsing with large grammars is currently prohibitive for real-time applications, such as re-ranking automatic speech recognition results, or sentence boundary detection and punctuation prediction over large word lattices. The methods we will present in this thesis increase the efficiency of high-accuracy context-free parsing by up to three orders of magnitude over the CYK algorithm, and are over an order of magnitude faster than state-of-the-art efficient search methods, with no loss in accuracy, allowing rich syntactic information to be incorporated into a number of NLP applications that were previously limited by computational speed.

1.1 Problem Statement

Since the inception of computational approaches to constituent-based parsing, both grammars and search methods to find accurate parse trees have rapidly co-evolved. When parsing with relatively small grammars, chart parsing with top-down filtering (as in the *P*earl parser [110]), or pruned shift-reduce parsing (as in the Marcus parser [113], the

[4]See Appendix B for more detail about these grammar formalisms and their relation to context-free grammars.

Core Language Engine [3], or the Fiddich parser [78]), were sufficient to retrieve syntactic analyses in a timely fashion. But within the past 15 years, probabilistic context-free grammars have grown to become more accurate and robust, increasing from a few thousand productions to several million (or larger) via linguistically inspired nonterminal annotation [86, 102], lexicalization [46, 32], and latent-variable techniques [116, 130] (we discuss the evolution of context-free grammars in detail in Section 2.9). The resulting distribution over these large grammars is more sensitive to contextual factors, increasing accuracy from approximately 70% to 90% on standard newswire parsing tasks. Unfortunately, this gain in accuracy comes at the cost of parsing speed. Standard dynamic programming algorithms, such as the CYK algorithm [43], reduce the search from exponential to polynomial complexity, but even with this reduction in complexity, parsing is still too slow, requiring as much as one minute per sentence [17]. Deterministic algorithms for dependency parsing exist that can extract syntactic dependency structure very quickly [124], but this approach is often undesirable as constituent parsers have been found to be more accurate and more adaptable to new domains [131]. Furthermore, some applications, like machine translation, rely on substring-oriented processing, which is not available with dependency parse structure.

In order to apply constituent parsing to real-time applications or web-scale text with high accuracy, efficient search though the solution space created by these large grammars is vital. Present-day context-free parsers, such as the parsers of Collins [46], Charniak [32], and Petrov and Klein [132], all employ approximate inference techniques – such as best-first, beam, and coarse-to-fine search – to find reasonable solutions quickly (we discuss current methods for efficient inference in Chapter 3). Efficient search methods have typically played second fiddle to accuracy gains; we observe that the prevailing paradigm for context-free parsing research is to (1) create a high-accuracy grammar, and then (2) apply algorithms to search the solution space efficiently in hopes that accuracy isn't adversely affected.[5] Leveraging the efficient search methods listed above, these three parsers perform inference at between 20 and 100 words per second. Although much faster than

[5]Step ($1\frac{1}{2}$) is to gloat about your 0.2 accuracy improvement over prior work at ACL.

pure dynamic programming, these speeds are orders of magnitude slower than deterministic shift-reduce parsers for dependency or low-accuracy constituent parsing [93, 124], and we believe that inference for high-accuracy constituent parsing has significant room for improvement.

Furthermore, current methods for efficient inference are sometimes approached in a grammar-specific way. For example, coarse-to-fine pruning requires crafting a coarse grammar from the target (fine) grammar such that it is both fast to parse with and effective for pruning. Creating such a grammar is a non-trivial problem, driven in past work by grammar knowledge and implementation convenience. Goodman [70] suggests using a coarse grammar consisting of treebank non-terminals, such as NP and VP, and then non-terminals augmented with head-word information for the more accurate second-pass target grammar. A similar approach is followed by Charniak [32] and Petrov and Klein [132], but customized to their respective grammar representations. We find that grammar-specific efficient search methods require unnecessary domain-specific or grammar-specific knowledge and we aim in this thesis to propose new methods that are entirely grammar agnostic and applicable to any weighted context-free grammar without customization.

A third problem we note is that context-free models have largely improved due to lexical (or lexical class) conditioning information, yet efficient search methods rarely leverage this resource. Prior pruning work has instead relied on comparing local competitors (beam-search) or the accuracy of an impoverished PCFG model (coarse-to-fine). Neither of these techniques take advantage of the lexical information present in the input sentence, such as subcategorization preferences or phrasal boundary cues, which can give vital indications about the final parse structure. It is our hypothesis that leveraging this lexical information prior-to and during the search for the optimal parse tree will add a new dimension of knowledge to the search, allowing portions of the search space to be pruned with high confidence. In addition, such methods would be orthogonal to prior pruning work and, thus, efficiency gains should be additive.

In summary, this thesis will present novel efficient inference algorithms that both prune and prioritize the search space, such that high-accuracy constituent parsing is an order of magnitude faster than the current state-of-the art search algorithms, and adhering to the

following aims:

- Applicable to all PCFG models, without grammar-specific knowledge;
- Directly condition the search on long-distance lexical information, opposed to inheriting it through the grammar;
- Present orthogonal approaches to current methods for cumulative efficiency gains;
- Efficiency is increased without compromising accuracy.

1.2 Thesis Overview

We address each of these above-mentioned aims in the remainder of this thesis. In Chapter 2 we introduce notation, define the context-free grammar formalism, and discuss its strengths and weaknesses compared to alternative grammar formalisms. We also discuss the practical aspects of constituent parsing, including induction of a grammar from a treebank, model smoothing, and the evolution of grammar induction methods over the past two decades. In Chapter 3 we review current methods for efficient inference, including graph-based search methods, coarse-to-fine search, and pipeline-constrained search, and how they each relate to the efficient inference methods we present in this thesis. Metrics for evaluating efficient search are also discussed, laying the foundation for comparisons of prior work with the methods we present.

Chapters 4 through 7 encapsulate the novel contributions of this thesis. Chapter 4 presents multiple strategies for prioritizing the search space within graph-based parsing architectures, building on the prior work of Charniak et al. [33]. We present an extension of this prior work in addition to a novel prioritization method conditioned on expected lexical boundary cues surrounding each constituent, and compare the effectiveness of each within best-first and beam-search parsers. Chapter 5 extends work of Roark and Hollingshead [145] to close portions of the dynamic programming chart using finite-state tagging models. We complement this work by restricting the population of span-1 chart cells, providing a framework for which the entire chart can be constrained. Results applying such constraints on left- and right-binarized grammars for English and Chinese are also presented. Chapter

6 is inspired by the prioritization methods of Chapter 4 and chart-constraining methods of Chapter 5. Here, we introduce two classifications methods to constrain individual cells in the dynamic programming chart, opposed to closing all cells starting or ending at a word position (i.e., closing full diagonals in the chart) as is done in [145]. Our approach to constraining the dynamic programming chart can alternatively be seen as predicting the unlabeled constituent structure of the final parse tree. We conclude Chapter 6 assuming a parsing architecture where constituents are prioritized at the span level and pruned with a beam-search, and present a novel method to adapt the beam-search tuning parameters to span-specific thresholds.

Chapter 7 unites the efficient search methods presented in Chapters 4, 5, and 6 and evaluates their additive gains. Results are presented for community-standard parsing tasks in English, Chinese, and German with decoding times over 1,500 words per second — over 12x faster than multi-level coarse to fine — at state-of-the-art accuracy levels. The source code for all efficient search methods we present is publicly available to facilitate comparison and replication of results. We conclude Chapter 7 with a synopsis of the our results and a discussion of the additional efficient parsing optimizations present in our parser.

This thesis is a culmination and extension of previously published work. The relevant publications by the author in collaboration with colleagues can be found in [16, 17, 18, 56, 57, 144].

1.3 Thesis Contributions

The seminal contributions of this thesis include four grammar-agnostic methods to significantly increase parsing efficiency with large context-free grammars. These methods include (1) extensions of constituent boundary prioritization, (2) finite-state unary constraints, (3) unlabeled constituent prediction, and (4) beam-width prediction. To our knowledge, these methods have produced the fastest high-accuracy context-free parsing speeds by over a factor of ten on standard English, Chinese, and German corpora. Specific contributions from these methods include:

1. The use of best-first parsing techniques with very large grammars in a way that

eliminates edge comparability issues by comparing only edges of the same span;

2. A novel decomposition of the Charniak and Caraballo boundary figure-of-merit [25] for efficient estimation of the outside probability heuristic;

3. The first comparison of best-first and beam-search parsing across numerous grammars and prioritization functions;

4. A finite-state tagging approach to prune unary production in span-1 chart cells, complementing prior work by Roark and Hollingshead [145] to prune all cells in the dynamic programming chart;

5. The first extensive evaluation of chart constraints to both left- and right-binarized grammars, parsing Chinese, and integration with other efficient inference methods such as best-first and coarse-to-fine pruning;

6. A generalization of chart constraints [145] to prune individual chart cells, opposed to diagonals in the chart;

7. An adaptive algorithm to predict the beam-width during bottom-up beam-search chart parsing, incorporating an asymmetric loss function to control for the imbalance between over- and under-predicting the true value;

8. Introduction of efficient inference methods that do not rely on reference transcriptions, providing a framework for optimal accuracy/efficiency tuning on domains with no labeled corpora;

9. The fastest reported parsing time for high-accuracy English, Chinese, and German constituent parsing, at over 1,500 words per second with no accuracy degradation compared to the maximum likelihood solution, and over an order of magnitude faster than multi-level coarse-to-fine pruning [132].

Chapter 2

Weighted Context-Free Grammars and Parsing Preliminaries

People are remarkable at distinguishing syntactically valid sentences from those that are not. For example, consider the following:

1. The Swiss Alps are not for the faint of heart.
2. The Swiss Alps not are the for faint heart of.
3. Traveling Zürich by train is.
4. Warm hats is wonderful.

Only the first sentence above is grammatical. Even without formal training, nearly all English speakers know something is amiss when a sentence incorrectly ends on a linking verb or has improper agreement. How the human mind determines the grammaticality of language continues to be an open question, and also a topic beyond the scope of this thesis. What we are concerned with here are formal systems of linguistic theory and their relevance to NLP, including representative power, complexity of inference, model construction effort, and parsing performance on well-defined tasks. Grammar formalisms provide a basis for empirical evaluation of competing linguistic theories, each with their own strengths and weaknesses for specific NLP objectives.

2.1 The Chomsky Language Hierarchy

In 1956, Noam Chomsky presented a hierarchy of formal languages, describing generative power of each [38]. Levels of the hierarchy are characterized by a single rewrite rule, specifying the set of allowed terminals and non-terminals on the rule's left-hand and right-hand side. A language $\mathcal{L}$ is said to be of type X (i.e., context-free) if all strings in the language can be generated by rewrite rules of type X. Each level of the hierarchy is a proper subset of the next highest, more expressive level (i.e., all regular languages are context-free, but not all context-free languages are regular).

The Chomsky Hierarchy is summarized in Table 2.1, where A is a single non-terminal, B is 0 or 1 non-terminals, τ is a sequence of 0 or more terminals, α and β are sequences of 0 or more terminals and non-terminals, and γ is a sequence of 1 or more terminals and non-terminals. We also note the complexity of inference for each formalism with respect to the length of the string, N, and recognize that the gains of additional expressive power comes with the cost of a significant increase in worst-case efficiency.

Language	**Rewrite**	**Inference**
Regular	$A \rightarrow \tau B$	$O(N)$
Context-Free	$A \rightarrow \beta$	$O(N^3)$
Context-Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$	NP-Complete
Recursively Enumerable	$\alpha \rightarrow \beta$	Undecidable

Table 2.1: The Chomsky Language Hierarchy.

In 1985, Aravind Joshi introduced the notion of *mildly context-sensitive languages* [89]. As the name implies, this formalism lives between context-free and context-sensitive languages in the original Chomsky hierarchy. Mildly context-sensitive languages are motivated by favorable linguistic and computational properties, and are defined as:

- Containing all context-free languages;
- Admitting limited cross-serial dependencies;
- All languages are parsable in polynomial time;

- All languages have constant growth.

Grammar formalisms such as Tree Adjoining Grammar and Combinatorial Categorial Grammar are known to be mildly context-sensitive, while Lexical Functional Grammar and Head-driven Phrase Structure Grammar are fully context-sensitive. We discuss the modeling advantages of mildly and fully context-sensitive languages in Section 2.3.

2.2 Context-Free Grammar

In this thesis we focus on the Context-Free Grammar (CFG) formalism.[1] CFGs are a natural formalism to represent the constituency behavior of language, where a constituent is defined as a sequential group of words functioning as a unit. For example, consider the sentence in Figure 1.1d. The string "the real-estate market" functions as a complete unit in the sense that it must remain together if moved in the sentence (i.e., "The real-estate market, as usual, had overreacted.") and the sentence is still syntactically valid if the phrase is replaced by a alternative, but syntactically equivalent unit (i.e., "As usual, the flesh-eating robot cyclops had overreacted."). Within the formalisms we consider in this thesis, constituents are hierarchically organized and labeled with their syntactic function, and, as discussed in Chapter 1, are useful to many NLP applications.

Context-free grammars encode the set of grammatically valid constituent structures. For example

$$
\begin{aligned}
&\text{S} \rightarrow \text{NP} \quad \text{VP} \\
&\text{NP} \rightarrow \text{DT} \quad \text{NN} \quad \text{NN} \\
&\text{PP} \rightarrow \text{IN} \quad \text{ADJP} \\
&\text{DT} \rightarrow \textit{the} \\
&\text{NN} \rightarrow \textit{market} \\
&\ldots
\end{aligned}
$$

where the production NP $\rightarrow$ DT NN NN indicates that a determiner (DT) followed by two nouns (NN) can be grouped together to form a noun phrase (NP). The remainder of this

[1] Context-Free Grammars are also known as Phrase Structure Grammars or Backus-Naur Form, after the two scientists who independently discovered the formalisms soon after Noam Chomsky [5].

chapter will present a formal definition of CFGs and probabilistic CFGs, and situate the representative power of CFGs with respect to other formal language families.

A context-free grammar (CFG) is a set of production rules[2] that describe how to form syntactically valid sentences in a language. It is formally defined by a quadtuple $G = (V, T, S^\dagger, P)$ where:

- V is a finite set of non-terminals, each representing a different type of phrase or clause in the sentence (e.g., NP, VP, ...);
- T is a finite set of terminals, disjoint from V. This is the lexicon of the language (e.g., *market*, *the*, *usual*, ...);
- $S^\dagger$ is a unique start symbol from V, of which all syntactically valid sentences are derived. In this thesis we use the symbol ROOT;
- P is a set of productions of the form $A \to \beta$, where $A \in V$ and $\beta \in (V \cup T)^*$. Note that β may be the empty set ($\emptyset$).

In addition, we define a *derivation* $A \Rightarrow \beta$ as a sequence of one or more applications of context-free rules, starting at non-terminals A and ending with a string (order lists of of terminals), β, where context-free rules are applied by replacing the symbol on the left-hand side of the production with those on the right-hand side. A language $\mathcal{L}$ is said to be context-free if and only if all (possibly infinite) sentences in $\mathcal{L}$ can be generated by a context-free grammar G from the start symbol $S^\dagger$:

$$\mathcal{L} = \{\tau | \tau \in T^* \text{ and } S^\dagger \Rightarrow \tau\} \tag{2.1}$$

A CFG is defined as *proper* if the following conditions hold:

- All non-terminals $A \in V$ are derivable from $S^\dagger$;
- For all non-terminals $A \in V$, there exists some terminal $t \in T$ such that A derives t.

[2] Also known as rewrite rules, grammar rules, derivation rules, or simply productions.

All CFGs used in this thesis are induced from treebanks. Because each non-terminal in the treebank is observed in a tree rooted by $S^{\dagger}$ and derives some terminal symbol $t \in T$, all CFGs we consider are proper [36].

2.3 Limitations of CFGs

There are a number of limitations to the CFG formalism described above for capturing the syntax of language. One shortcoming is that CFGs are not lexicalized. Pertinent syntactic information, such as number agreement or subcategorization is dependent on the specific lexical items of the sentence, but this information is not explicitly captured by the rewrite rules of the grammar. For example, consider the following context-free grammar:

$$\begin{aligned}
&\text{S} \rightarrow \text{NP} \quad \text{VP} \quad \text{NP} \\
&\text{NP} \rightarrow \textit{Germans} \\
&\text{NP} \rightarrow \textit{bratwurst} \\
&\text{VP} \rightarrow \textit{love} \\
&\text{VP} \rightarrow \textit{put}
\end{aligned}$$

The grammar above derives both "Germans love bratwurst" and "Germans put bratwurst", considering them equally valid sentence in the language. Although transformations exist to incorporate lexical information into CFGs (see Section 2.9), alternative grammar formalisms such as lexical functional grammars and head-drive phrase structure grammars account for lexicalization directly.

Another concern when considering probabilistic CFGs (PCFGs; see Section 2.4 for a formal definition) is that, when they are used as a basis for a probabilistic grammar, the formalism encodes a bias towards smaller trees. This is because the probability of a tree is the product of all production rules used to generate the tree. As all production probabilities are less than one, the greater the number of production rules required to derive a sentence, the less probable the overall parse tree is given the model. In addition, it is irrelevant how the productions are ordered in the tree: the derivation process does not affect the probability of the parse tree, only the set of productions used. But in natural language there is no direct correlation between the size of the parse tree and the

grammaticallity of the sentence. Hence, it has been argued that PCFGs are lacking when used as the basis for a probabilistic model natural language syntax [112].

Perhaps the most limiting attribute of the CFG formalism from a linguistic perspective is that they are unable to encode aspects of natural languages that are known to be non-context-free, such as as cross-serial dependencies in Dutch and Swiss-German [159] and reduplication in Bambara [50]. This finding, coupled with other limitations discussed above, has caused some computational linguists to argue that context-free grammars are not the proper formalism to capture the syntax of language [2]. Alternative grammar formalisms have been proposed to specifically address these concerns, including Tree Adjoining Grammar, Combinatorial Categorial Grammar, Lexical Functional Grammar, and Head-driven Phrase Structure Grammar. Table 2.2 summarize two attributes of each formalism: its language class, which determines the expressive power of the formalisms, and the upper bound on exact inference. We describe each of these methods in Appendix A, including a brief discussion of their relationship to our work.

Grammar Formalism	**Language Class**	**Inference**
Constituent Phrase Structure	Context-Free	$O(N^3)$
Tree Adjoining Grammar	Mildly Context-Sensitive	$O(N^5)$[1]
Combinatorial Categorial Grammar	Mildly Context-Sensitive	$O(N^6)$
Lexical Functional Grammar	Context-Sensitive	$O(2^N)$
Head-driven Phrase Structure Grammar	Context-Sensitive	$O(2^N)$

Table 2.2: A summary of grammar formalisms.

In this thesis we study the efficiency of inference with context-free grammars, as opposed to mildly or fully context-sensitive languages, for three reasons. First, there are well-known $O(N^3)$ parsing algorithms for context-free grammars, providing very efficient baseline performance when using this formalisms. Even with these efficiencies, $O(N^3)$ parsing is still slow for long sentences and complexity of $O(N^5)$ or higher with context-sensitive grammars presents significant problems. Second, the advantages of context-sensitive formalisms, such as the ability to represent cross-serial dependencies, are not of

[1]Tree Adjoining Grammar can be parsed in $O(N^4)$ if the grammar is splittable [61].

large practical benefit. These constructions are often rare or nonexistent depending on the language or domain being processed, and modifications exist to augment CFG parsing algorithms to handle a subset of cross-serial dependencies, which do not increase the worst-case parsing complexity [174]. Finally, since the development of large treebanks (i.e., The Penn English and Chinese treebanks [114, 180]), PCFG models have frequently outperformed alternative formalisms on standardized constituent parsing tasks, despite shortcoming such as the bias towards shorter derivations. Indeed, the Collins [46], Charniak [32], and Petrov [132] parsers, well known for high-accuracy inference, have all been based on context-free grammars. Notable exceptions include Combinatorial Categorial Grammar parsing by [41], and more recently, splittable Tree Adjoining Grammar parsing by Carreras et al. [28]. The accuracy of the latter approach is very competitive with state-of-the-art PCFG parsers, although the upper bound on inference is still $O(N^4)$.

We also believe that less expressive formalisms, such as finite-state approximations of phrase structure grammars [127], or deterministic classification parsing algorithms [150] – both which run in linear time – have serious promise for future research in efficient constituent parsing. But for this present work, we concentrate on PCFG models due to their superior accuracy and adaptability to new domains [131].

2.4 Probabilistic Context-Free Grammar

An unweighted context-free grammar is able to distinguish the grammaticality of a sentence: a boolean value indicating if the sentence is derivable from the start symbol by the CFG production rules. In practice, there is often more than one valid derivation according to the grammar. In fact, the number of possible derivations is exponential in the length of the string. In these cases, it is beneficial to know which derivation is most likely.

A probabilistic context-free grammar (PCFG) can be used to assign a probability to each derivation (parse tree) in the language. A PCFG is defined as the tuple $(V, T, S^{\dagger}, P, \rho)$, where $V, T, S\dagger$, and P follow the CFG formalism of Section 2.2, and ρ assigns a probability to each production in P. Probabilities from ρ are often stored with the productions themselves when the grammar is explicit, as can been seen in Table 2.3. Note that this

1.0	S $\rightarrow$ NP VP NP
0.2	NP $\rightarrow$ *Germans*
0.8	NP $\rightarrow$ *bratwurst*
0.5	VP $\rightarrow$ *love*
0.5	VP $\rightarrow$ *put*

Table 2.3: Toy example of a probabilistic context-free grammar.

example grammar is a PCFG as it is conditionally normalized such that $\sum_{\beta} \rho(A \rightarrow \beta) = 1$, but such normalization is not required by definition for arbitrary weighted context-free grammars.

Given sentence s and derivation t produced with a PCFG, we compute the joint probability of the derivation and the sentence by multiplying the set of production probabilities:

$$\mathrm{P}(t, s) = \prod_{A \rightarrow \beta \in t} \rho(A \rightarrow \beta) \tag{2.2}$$

Applying Bayes' Rule and noting that $\mathrm{P}(s|t) = 1$ because a parse tree t determines the sentence s, we see that the joint probability of Equation 2.2 is identical to the probability of the tree t:

$$\mathrm{P}(t, s) = \mathrm{P}(s|t)\mathrm{P}(t) = \mathrm{P}(t) \tag{2.3}$$

Using the grammar of Table 2.3, the probability of the (only) derivation for "Germans love bratwurst" is 1.0 x 0.2 x 0.5 x 0.8 = 0.08. Note that the order of the productions is not considered in this equation. Two derivations containing the same productions but in a different configuration – such as "Germans love bratwurst" and "bratwurst loves Germans" will have the same probability.

2.5 Treebanks and PCFG Induction

Treebanks have revolutionized computational approaches to syntax. Before the availability of these large, annotated corpora, linguists built grammars by hand requiring months – sometimes years – of labor [44]. Due to the required effort, hand-crafted grammars were

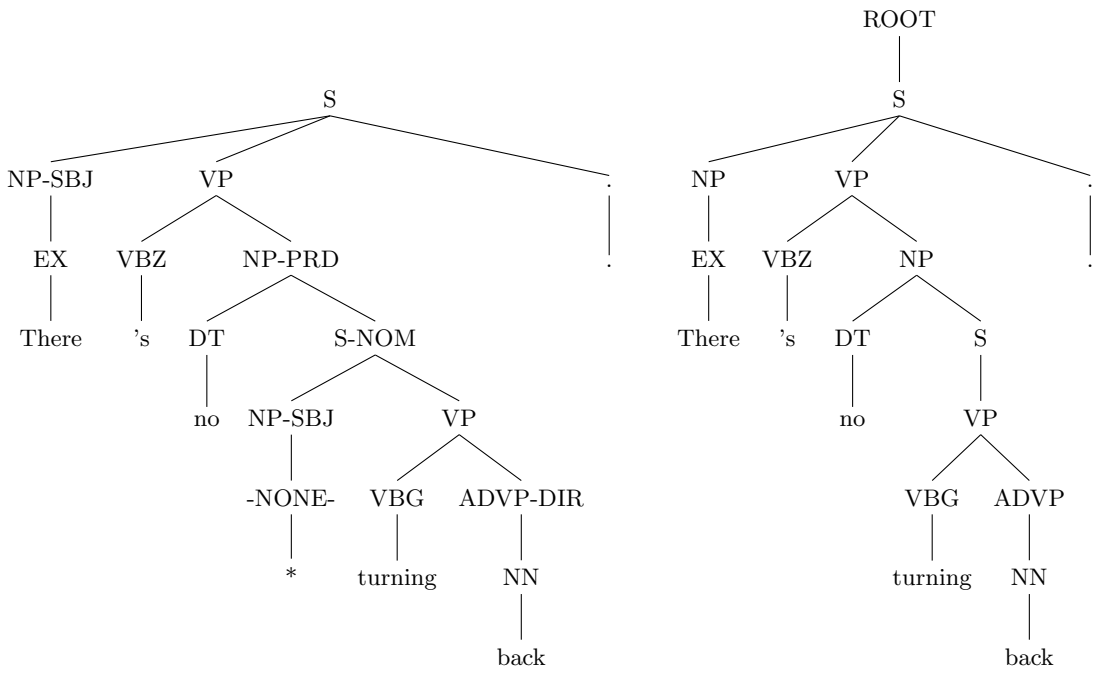
often tailored to specific domains and unable to handle ill-formed or unexpected strings. For example, Black et. al developed a grammar over three years for sentences from a computer manual domain and restricted to a vocabulary of 3000 words, but were still unable to analyze 4% of unseen sentences in the same domain [11]. With the recent ability to induce a grammar from data, advancements in PCFG modeling has rapidly accelerated.

Assigning a probability or score to each production in a CFG has been done in a variety of ways including maximum likelihood estimation [30], maximum-entropy estimation [136], neural networks [76], and discriminative training [134]. The equation for maximum likelihood estimation is defined below. Given all productions extracted from a treebank, the probability of non-terminal A producing β is equal to the normalized frequency of occurrence:

$$\mathrm{P}(A \to \beta) = \frac{\mathrm{count}(A \to \beta)}{\sum_{\gamma} \mathrm{count}(A \to \gamma)} = \frac{\mathrm{count}(A \to \beta)}{\mathrm{count}(A)} \tag{2.4}$$

The process of treebank creation is itself very time consuming and error prone. Consistency is not always maintained across annotaters or across time, but grammar induction over a large corpus helps alleviate outlying conditions. Those creating treebanks must also decide what syntactic information to encode, whether it be constituent or dependency structure, specific grammatical categories such as tense, aspect, or gender, or subcategorization information. The second and third releases of the English Penn Treebank [115] annotate forms of "deep" structure such as predicate/argument information. But in practice, much of the "deep" linguistic information embedded in the Penn treebank is removed before training or testing with CFG constituent parsers. The following is a list of modifications commonly applied before grammar induction, although this process varies among researchers:

1. Removal of empty nodes;
2. Striping functional tags and cross-referencing annotations;
3. Affixing a ROOT unary production to the root symbol of the original tree;



(a) Original Penn treebank parse tree

(b) Transformed parse tree

Figure 2.1: A parse with (a) the original WSJ treebank annotations and structure, and (b) the same tree after processing.

4. Removing $X \rightarrow X$ unary productions for all non-terminals X;

5. Collapsing unary chains to a single (possibly composite) unary production [99];

6. Introducing new categories such as AUX [31];

7. Collapsing of categories such as PRT and ADVP [46];

8. Removal of quotation marks [14].

One motivation for the removal of empty nodes is that bottom-up parsers – a popular parsing framework – do not easily support arbitrary insertion. Function tag removal or collapsing multiple categories can be seen as a form of smoothing, while cross-referencing annotations are sentence specific and have no generalization power without additional processing. Unary transforms, such as (3) and (4) are primarily motivated by efficient

processing and implementation simplicity. Because empty nodes and function tags are not incorporated into the PCFG, parsing with grammars induced from these modified treebanks does not provide this deeper level of syntactic analysis. Some research has been done in past 10 years to recover empty categories, traces, and function tags from the output of bottom-up CFG parsers [87, 54, 24], and such processes are equally applicable to the output of the efficient inference methods we will present in Chapters 4 through 6.

In this thesis we apply transforms 1-3 above and otherwise leave the treebank in its original form. An example of applying transforms 1-3 to a treebank tree can be seen in Figure 2.1. These tree transformations have a large impact on the number and type of productions in the treebank, especially unary productions, which we discuss more in Chapter 5.

2.6 Dynamic Programming and CYK Parsing

In this section we examine the CYK dynamic programming algorithm, which will serve as the baseline for efficient parsing experiments throughout this thesis. The CYK algorithm requires a binarized grammar, which can either be done as a pre-processing step (Chomsky Normal Form) or as an on-line process (dotted rules). We discuss binarization in Section 2.7.

Formally defined, the optimization goal of constituent parsing is, given an input sentence s and a model θ over all possible syntactic trees, find the optimal parse tree t^* from among the set of all trees T:

$$t^* = \underset{t \in T}{\operatorname{argmax}}\, P(t|s;\theta) \tag{2.5}$$

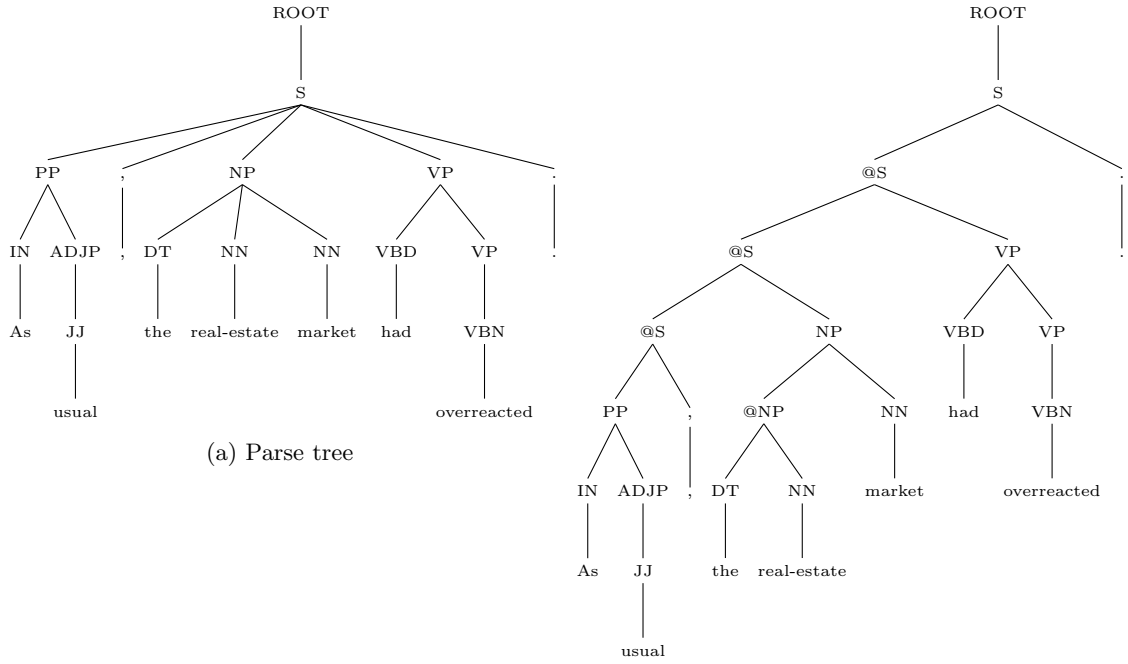
If the model θ is a PCFG, we can rewrite equation 2.5 using Bayes' Rule, Equation 2.3, and the observation that P(s) is constant for a given sentence and safely ignored during the argmax operation:

$$
\begin{aligned}
t^* &= \operatorname*{argmax}_{t \in T} \mathrm{P}(t|s) && (2.6)\\
&= \operatorname*{argmax}_{t \in T} \frac{\mathrm{P}(t,s)}{\mathrm{P}(s)} && (2.7)\\
&= \operatorname*{argmax}_{t \in T} \mathrm{P}(t,s) && (2.8)\\
&= \operatorname*{argmax}_{t \in T} \mathrm{P}(t) && (2.9)\\
&= \operatorname*{argmax}_{t \in T} \prod_{A \to \beta \in t} \rho(A \to \beta) && (2.10)
\end{aligned}
$$

Finding t^* is a non-trivial problem as the number of possible trees is exponential in the size of the sentence, and modern high-accuracy grammars induced from treebanks often contain tens of thousands – even millions – of productions. In order to parse sentences of reasonable length, efficient search through this space of exponential tree is essential. Alternative optimization objectives have also been applied, such as Goodman's max-recall decoding [71] and Petrov's max-rule decoding [132] that more directly optimize for the common evaluation criteria of labeled F_1. Both of these methods are run during a third pass over the dynamic programming chart (following the standard inside/outside passes). Although an analysis of various decoding methods is outside the scope of this thesis, we do note that the efficient search techniques we will present improve the efficiency of computing the initial inside scores of the packed forest. This indirectly prunes the subsequent outside and final decoding passes and will thus improve the overall efficiency of context-free parsing despite the optimization criteria used.

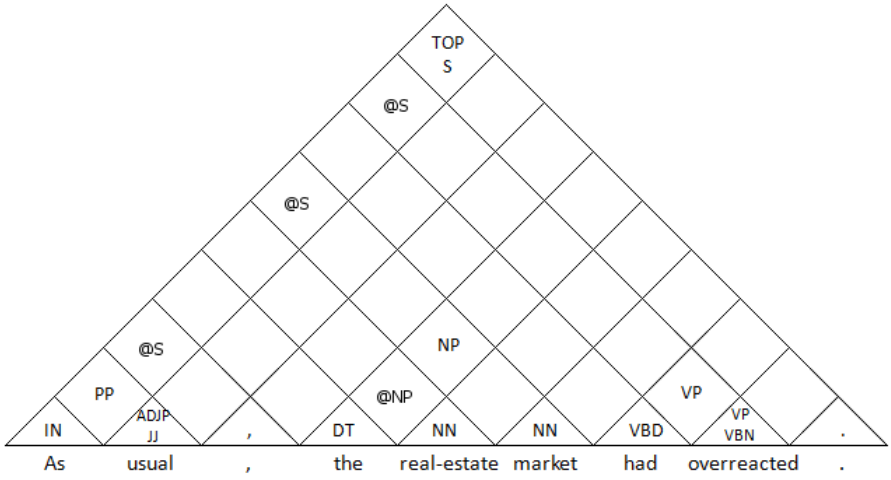
Dynamic programming methods to perform exact inference, such as the CKY algorithm [43, 95, 182], take advantage of the large overlap in sub-structure shared between the exponential number of solutions. A key efficiency of these algorithms rests on the assumption that every grammar production has at most two children on the right-hand side. It has been shown that any CFG can go through a process known as *binarization* to transform the original grammar into a grammar with at most two right-hand side children, while still being strongly equivalent (generating the same grammatical structure) [39, 83].

Given that the number of possible deviations (parse trees) is exponential in the sentence length, finding the highest scoring derivation by enumerating each possible tree is not



(a) Parse tree

(b) Left-binarized parse tree



(c) Dynamic programming chart with binarized constituents

Figure 2.2: Parse tree and corresponding dynamic programming chart. Non-terminals preceded with the symbol '@' are created through binarization. Each cell in the chart spans a unique substring of the input sentence.

practical. Doing this efficiently requires two key observations. First, although no one derivation will be identical to another, many derivations will share identical sub-structure.

For example, two parse trees may differ on prepositional phrase attachment, but contain the same internal noun-phrase structure. The second observation is that probability of the derivation can be decomposed into a product of its parts. Therefore, if multiple derivations share identical sub-structure we only need to compute the probability of each sub-structure once to be used by multiple trees. Because the complex problem of scoring all possible trees can be broken down into smaller subtasks, context-free parsing can leverage dynamic programming to find the highest scoring tree efficiency.

Dynamic programming for context-free inference generally makes use of a chart structure, as shown in Figure 2.2c. Each word $w_1 \dots w_N$ of the length-N input sentence forms the base of the chart. Each cell in the chart represents a collection of possible constituents covering a specific substring, which is identified by the indices of the first and last words of the substring. Thus, the cell identified with (b, e) will contain possible constituents spanning the substring $w_b \dots w_e$, where the *span* of a constituent or a chart cell is defined as the number of words it covers, or $e - b + 1$. In this thesis we will refer to span-1 chart cells, meaning all chart cells covering a single word (the bottom-most row of cells in Figure 2.2c). Context-free inference has cubic complexity in the length of the string N, due to the $O(N^2)$ number of chart cells and $O(N)$ possible child configurations at each cell. As an example, a cell spanning (b, e) must consider all possible configurations of two child constituents (child cells) that span a proper prefix (b, m) and a proper suffix (m+1, e), where $b \leq m < e$ leading to $O(N)$ possible midpoints.

Context-free inference can be preformed bottom-up (building constituents from words to the start symbol $S^\dagger$), top-down (expanding constituents from $S^\dagger$ until they derive the target sentence), or some combination thereof (e.g., left-corner [143]). The first two traversal methods restrict the search in opposing ways by initially enforcing possible derivations to conform to the lexical input (bottom-up) or global phrase structure (top-down). Note that this traversal is independent of a depth-first verses breadth-first search order; we can use either search strategy with bottom-up or top-down parsing [118]. For simplicity, we will assume a bottom-up breadth-first derivation unless otherwise stated.

Given this assumption, the intuition of the CYK algorithm is that it builds longer-span constituents by combining smaller span constituents, as allowed by productions in a

Algorithm 1 CYK($w_1 \dots w_n,\ G = (V, T, S^\dagger, P, \rho)$)

Input:

$w_1 \dots w_n$: Input sentence

G: PCFG in Chomsky Normal Form

Output:

α: Viterbi-max score for all non-terminals over every span

1: **for** $s = 1$ to n **do** ▷ Span width: bottom-up traversal
2: **for** $b = 1$ to $n - s + 1$ **do** ▷ Begin word position
3: $e \leftarrow b{+}s{-}1$ ▷ End word position
4: **for** $A_i \in V$ **do**
5: **if** $s = 1$ **then** ▷ Special case for lexical productions
6: $\alpha_i(b, e) \leftarrow \rho(A_i \rightarrow w_b)$
7: **else**
8: $\alpha_i(b, e) \leftarrow \max_{b \le m < e} \left(\max_{j,k} \ \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, m)\ \alpha_k(m + 1, e) \right)$
9: **for** $A_i \in V$ **do** ▷ Add unary productions
10: $v_i(b, e) \leftarrow \max \left(\alpha_i(b, e)\ ,\ \max_j \ \rho(A_i \rightarrow A_j)\ \alpha_j(b, e) \right)$
11: $\alpha(b, e) \leftarrow v(b, e)$
12: **return** α

context-free grammar. If we have a grammar production $A \rightarrow B\ C \in P$, a completed B entry in chart cell (b, m) and a completed C entry in chart cell $(m{+}1, e)$, we can place a completed A entry in chart cell (b, e). Such a chart cell entry is sometimes called an *edge* and can be represented by the tuple $\langle A \rightarrow B\ C, b, m, e \rangle$.

Algorithm 1 contains pseudocode for the CYK algorithm, where the context-free grammar G is assumed to be binarized. The function ρ maps each grammar production in P to a probability. Lines 1-3 iterate over all $O(N^2)$ chart cells in a bottom-up traversal. Line 6 initializes span-1 cells with all possible part-of-speech tags, and line 8 introduces the cubic complexity of the algorithm: maximization over all midpoints m, which is $O(N)$. The variable α stores the Viterbi-max score for each non-terminal $A_i \in V$ at each cell (b, e), representing the span's most probable derivation rooted in A_i. Backpointers indicating which argument(s) maximize each $\alpha_i(b, e)$ can be optionally recorded to efficiently extract the maximum likelihood solution at the end of inference, but we omit these for clarity; they are easily recoverable by storing the *argmax* for each *max* in lines 8 and 10.

We have also included pseudocode in Algorithm 1 to process unary productions in lines 9-10. Unary processing is a necessary step to recover the gold standard trees of the Penn treebanks, but it is often ignored in the presentation of the CYK algorithm. Due to this, implementation details sometimes differ from parser to parser. In Algorithm 1 we present a simplified version of unary processing that only allows unary chains of length one per span (excluding lexical productions). This approach is efficient and representative of the observed unary productions in the treebank, and can be iterated for longer unary chains as needed.[3] Note that line 10 uses the temporary variable v to store the accumulated Viterbi-max scores α. This temporary variable is necessary due to the iterative nature in which we update α. If we were to write the result of line 10 directly to $\alpha_i(b, e)$, then the subsequent maximization of $\alpha_{i+1}(b, e)$ would use an unstable version of α, some of which would already be updated with unary productions, and some which would not.

Computing the Viterbi-max scores (α) is sufficient when we are interested in the 1-best derivation, but Algorithm 1 can be easily modified to compute the inside probability if needed. The inside pass of the inside/outside algorithm is identical to the CYK algorithm we have presented modulo changing the "max" operators in lines 8 and 10 to a sum. This change accumulates all probability mass for all possible derivations of non-terminal A_i covering span (b, e). When combined with the outside algorithm, the inside/outside scores are useful for a number of tasks, including unsupervised grammar induction, pruning (i.e., coarse-to-fine pruning) [70], or minimum Bayes risk decoding [71, 167].

2.7 Chomsky Normal Form and Dotted Rules

A key feature to the efficiency of the CYK algorithm is that all productions in the grammar are assumed to have no more than two right-hand side children. Rather than trying to combine an arbitrary number of smaller substrings (child cells), the CYK algorithm exploits shared structure between rules and only needs to consider pairwise combination. To conform to this requirement, *incomplete* edges are needed to represent that further

[3] The beam-search and agenda-based parsers presented in Chapters 4-6 allow unary chains of arbitrary length.

combination is required to achieve a complete edge. This can either be performed in advance, e.g., by transforming a grammar into Chomsky Normal Form with "incomplete" non-terminals, or performed on the fly by creating incomplete edges can be represented through so-called dotted rules. Binarization via dotted rules is exemplified in the Earley algorithm [58]. For example, if we have a rule production $A \rightarrow B\ C\ D \in P$, a completed B entry in chart cell (b, m_1) and a completed C entry in chart cell $(m_1{+}1, m_2)$, then we can place an *incomplete* edge $A \rightarrow B\ C \cdot D$ in chart cell (b, m_2). The dot signifies the division between what has already been combined (left of the dot), and what remains to be combined. Then, if we have an incomplete edge $A \rightarrow B\ C \cdot D$ in chart cell (b, m_2) and a complete D in cell $(m_2{+}1, e)$, we can place a completed A entry in chart cell (b, e).

The second way to binarize the grammar is the off-line transformation into Chomsky Normal Form (CNF), which converts rules with more than two children on the right-hand side into multiple binary rules. Formally defined, a context-free grammar is in Chomsky Normal form if and only if all productions $p \in P$ conform to one of the following:

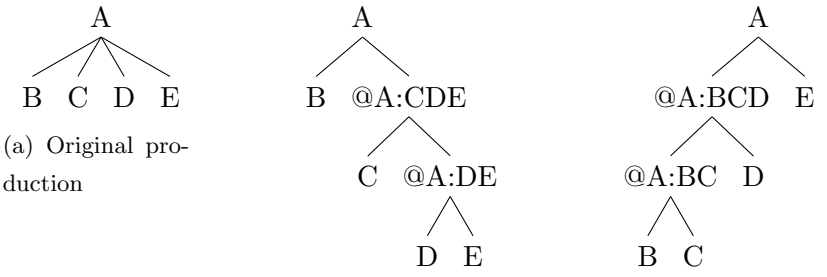
- $A \rightarrow B\ C$
- $A \rightarrow a$

where A,B,and C are non-terminals in V and a is a terminal in T.

When transforming a grammar into CNF, composite non-terminals are created to represent incomplete constituents, i.e., those edges that require further combination to be made complete.[4] For example, if we have the production $A \rightarrow B\ C\ D$ in the context-free grammar, then a new composite non-terminal would be created, e.g., $@A{:}BC$, and two binary rules would replace the previous ternary rule: $A \rightarrow @A{:}BC\ D$ and $@A{:}BC \rightarrow B\ C$. The $@A{:}BC$ non-terminal represents part of a rule expansion that needs to be combined with something else to produce a complete non-terminal from the original set of non-terminals.[5] A visual example can be seen in the following figure.

[4] In this section we assume that edges are extended from left-to-right, which requires a left-binarization of the grammar, but everything carries over straightforwardly to the right-binarized case.

[5] In this example, the symbol '@' indicates that the new non-terminal is incomplete, and the symbol ':' separates the original parent non-terminal from the children.



(a) Original production

(b) Right binarization

(c) Left binarization

Alternative binarization methods, such as head binarization [102], compact binarization [157], and learned (or "better") binarization [164] have shown efficiency advantages over standard left or right methods. In this thesis we focus on left and right binarization only. It is our hypothesis that these alterative binarization methods see their largest efficiency gains when applied to exhaustive CYK parsing, and the relative speedup when combined with the inference techniques we present in this thesis would be marginal.

We note that it is improper to use the term "Chomsky Normal Form grammar" as we explicitly discuss unary productions of type $A \rightarrow B$ where B is a non-terminal, and include these productions in our grammar. These productions violate the definition of a Chomsky Normal Form grammar, and therefore we will use the term 'binarized grammar' for the remainder of the thesis to indicate a grammar in CNF with the addition of unary productions. We will assume that all of our grammars have been previously binarized and we define V' to be the set of non-terminals that are created through binarization, and denote edges where $A \in V'$ as *incomplete edges*. Note that categories $A \in V'$ are only used in binary productions, not unary productions, a consideration that will be used in some of our efficient parsing algorithm.

2.8 PCFG Smoothing

Large grammars not only increase computational search requirements, but are also prone to sparse data problems as probability estimates quickly become unreliable. If a grammar rule is never observed in the training data, it receives zero probability. As a result, many

Symbols on RHS	Production Count	Most Frequent Production
1	1134416	DT → *the*
2	399840	PP → IN NP
3	125963	S → NP VP .
4	39758	NP → NP , NP ,
5	19463	S → PP , NP VP .
6	5764	S → NP , PP , VP .
7	3727	SINV → “ S , ” VP NP .
8	618	NP → NP , NP , NP , CC NP
9	320	NP → NP , NP , NP , NP CC NP
10+	312	NP → NP , NP , NP , NP , NP ,

Table 2.4: Number of children per constituent for all constituents in WSJ Section 02-21. Also included in the most frequent constituent for the indicated number of right-hand-side (RHS) children (excluding productions with only punctuation).

unseen sentences may fail to find even one valid parse tree – even if the sentence is well-formed – since a required production may not exist in the grammar.

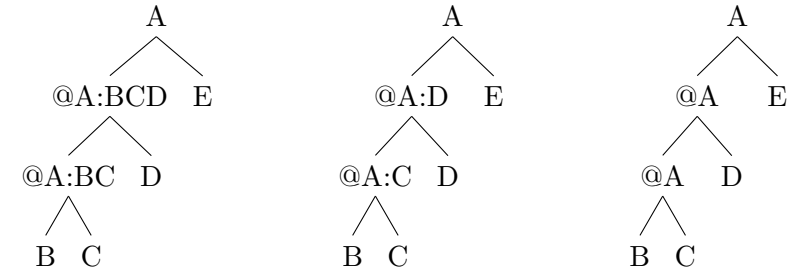
Sparse data problems are typically addressed by *smoothing* the distribution of children for each non-terminal [35]. Smoothing robs a small amount of probability from observed grammar rules, and uniformly distributes that probability over unobserved rules, allowing a class of unobserved productions to have some probability. PCFG smoothing can be done in many ways; Charniak applies a backoff model to smooth bi-lexical conditioning information [32], while Petrov et al. employ parameter tying in their latent-variable PCFG [130].

A common smoothing approach applied during binarization is to create a Markov model of right-hand side dependencies with each newly created binarized production. The Penn treebank annotation guidelines encourage flat phrase structure, and hence the number of right-hand side children observed per constituent is large, if reading productions straight from the treebank. In fact, we see in Table 2.4 that 312 productions extracted from the treebank are observed with 10 or more children on their right-hand side, one

of which is a noun phrase spanning a list of 26 items each separated by commas for a grand total of 51 children! One way to smooth the grammar so that we can generalize to e.g., lists of 22 items, is to reduce the number of child categories annotated on our new composite non-terminals introduced by binarization. For example, if instead of $@A{:}BCD$ we assign the label $@A{:}B$ to our new non-terminal – with the semantics "an incomplete A constituent with leftmost child B" – then the three rules that result from binarization are:

$$A \rightarrow @A{:}D\ E$$
$$@A{:}D \rightarrow @A{:}C\ D$$
$$@A{:}C \rightarrow B\ C$$

This is known as Markov order-1 smoothing since a history of one non-terminals is retained. Probabilistically, the result of this transformation is that B is conditionally independent from D and E given A. This approach will provide probability mass to combinations of children of the original category A that may not have been observed together, hence should be seen as a form of smoothing. One can go further and remove all child categories from the new non-terminals, i.e., replacing $@A{:}BCD$ with just $@A$. This is known as Markov order-0 smoothing. In each of these cases, new derivations become possible that would not have been possible with the original unsmoothed grammar. Examples of three levels of smoothing during binarization can be seen in the following figure.



(a) Unsmoothed left binarization

(b) Markov order-1 smoothing

(c) Markov order-0 smoothing

When parsing open-vocabulary domains, smoothing lexical production is also a good

idea. Considering again the WSJ corpus, nearly 2.5% of the words in Section 23 are never observed in all of Section 02-21 (including "Managed", "blood-letting", "254,280", and "snowball"). Without smoothing, sentences containing even a single new word would not be derivable given the context-free model. Common heuristics used to smooth lexical productions include text normalization (lower-casing, mapping numbers to a common symbol, removing hyphens, etc.) and estimating the probability of unknown-word classes given language-specific lexical cues. For example, if an unseen words end in the suffix "ly" then we can leverage the distribution of all productions deriving words in the training data that also end in "ly". All grammars in our experiments have been smoothed using both text-normalization and language-specific lexical cues.

2.9 The Evolution of PCFGs

As discussed in Section 2.3, modeling the context-sensitive and lexically-driven syntax of language with a context-free grammar has its drawbacks. In this section we give a brief overview of grammar induction techniques that overcome some of these issues while still maintaining polynomial parsing complexity. These techniques alter PCFG probability estimation by splitting non-terminals into smaller classes and refining the conditional probability distributions over these new grammar productions to be more context-sensitive. These new grammars can often be enumerated explicitly, as is seen in Table 2.3, but when the space of possible conditioning information is too large (as it sometimes is for lexicalized grammars) then the conditional probabilities must be computed on the fly during inference.

Following standard treebank pre-processing, as discussed in Section 2.5, we can induce a grammar by extracting all productions observed in the treebank and assigning each the maximum likelihood probability described in equation 2.4. Inducing such a grammar from 40,000 Penn WSJ treebank sentences[6] creates a PCFG containing nearly 67,000 productions, 51,000 of which are *lexical productions* of the form $A \rightarrow \tau$ where τ is a terminal, and 16,000 *phrase-level productions*, where the right-hand side is one or more non-terminals.

[6] All sentences in Sections 02-21.

Because lexical productions are restricted to span-1 chart cells and not considered during the $O(N^3)$ "inner loop" of CYK parsing, we find it useful to distinguish the respective sizes of these sets by $|G_{lex}|$ and $|G_{phrase}|$. Following the extraction of productions from the corpus, right-binarization and Markov order-2 smoothing (see Sections 2.7 and 2.8) decrease the size of $|G_{phrase}|$ from 16,000 to 13,500 productions. Coupled with simple unknown word smoothing, this is a particularly straight-forward grammar to induce from a treebank. We call this a *Vanilla* PCFG and with exact inference methods, it achieves labeled precision and recall scores of 74.7% and 69.0% respectively on Section 23. The vanilla model can be written in terms of the conditioning information on production rules:

$$P_{vanilla}(A \rightarrow \beta) = P(\beta|A) \tag{2.11}$$

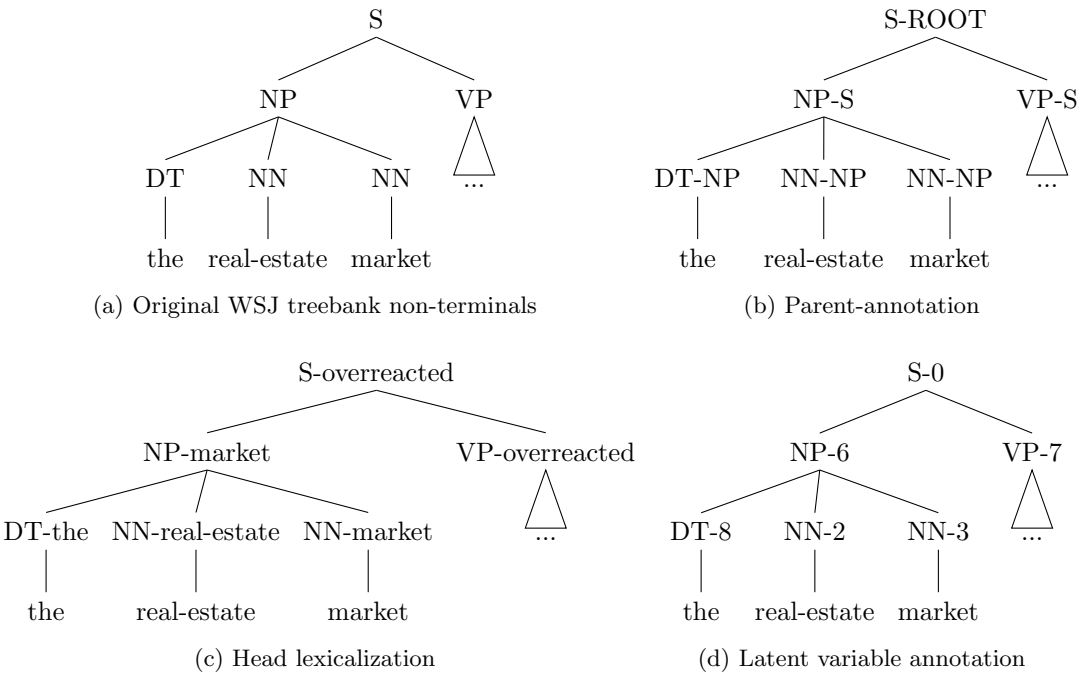
To increase accuracy over the vanilla model, one popular modification to the original treebank is *parent annotation* [86]. Johnson reports, for example, that the distribution of non-terminals on the right-hand side under a verb phrase is highly influenced by the parent node of the verb phrase. Standard PCFG induction from the WSJ Treebank disregards parent information; only one distribution over children is estimated for each non-terminal. To address this problem, Johnson suggests the following:

1. Replace all non-terminals in the training data with new non-terminals that explicitly include the direct parent non-terminal,
2. Induce a PCFG from this context-enriched data; probabilities are now conditioned on additional parent information:

$$P_{parent}(A \rightarrow \beta) = P(\beta|A, parent = X) \tag{2.12}$$

3. Perform inference with this new model,
4. For evaluation, map the parent-specific non-terminals in the parse tree back to the original non-terminal set and compute labeled precision and recall.

An example of a parse tree with parent-annotated non-terminals can be seen in Figure 2.3b. Adding the context of a parent label to each non-terminal potentially squares the



(a) Original WSJ treebank non-terminals

(b) Parent-annotation

(c) Head lexicalization

(d) Latent variable annotation

Figure 2.3: Three annotation schemes used to increase the accuracy of context-free parsing.

number of non-terminals in the new grammar and significantly increases the number of grammar rules. This idea can be pursued even further to include not just the constituent's parent information, but also grandparent or even great-grandparent when estimating production probabilities. Although the complexity of exact inference remains at $O(N^3)$, the run-time performance is severely impacted due to the true operating cost of $O(|G|N^3)$, where $|G|$ is the (constant) size of the grammar. Increasing the size of the grammar with parent (or grandparent) annotation improves accuracy, but at the cost of speed.

As mentioned in Section 2.3, another modeling deficiency of vanilla PCFGs is their independence from lexical and subcategorization information. Because the structure of constituents is often governed by specific words (or word classes), conditioning the probability of productions on this information may also be helpful [10]. But conditioning on all words in the phrase would lead to significant sparsity problems; plus, some words in a phrase have more syntactic relevance than others. To combat both these issues, one can condition on the *head* word (or word class) in a phrase, where the head is defined as the

lexical item driving the syntactic structure [46]. In practice, the phrasal head is found by way of if-then rules such as "If the constituent is a Verb Phrase then the head is the Verb" and "If the constituent is a Noun Phrase then the head is the right-most noun". Similar methods are also used to convert constituent trees into dependency structure [85].

Applying the intuition above, we can induce a new grammar with productions conditioned on the POS tag of the head word. This approach gleans the context of the syntactic heads without incurring data sparsity problems found by conditioning on lexical items directly. Such a model has the form

$$P_{headPOS}(A \rightarrow \beta) = P(\beta|A, headPOS = X) \tag{2.13}$$

We can subsequently combine head-word conditioning with parent annotation:

$$P_{headPOS+parent}(A \rightarrow \beta) = P(\beta|A, headPOS = X, parent = Y) \tag{2.14}$$

In [102], Klein and Manning investigate 17 linguistically motivated non-terminal annotations beyond the influence of constituent parents and head-words, in each case attempting to balance the addition of contextual information with over-splitting the training data. In an effort to keep their grammar small, no open-class lexical items were used for conditioning and only non-terminals benefitting from a specific refinement were kept, opposed to the universal application style of parent annotation. With careful engineering, their grammar achieves an impressive 85.7 F_1 on WSJ Section 23 with a grammar size under 52,000 productions – orders of magnitude smaller than lexicalized grammars. Although accuracy is not as high as the models of Charniak or Collins, Klein and Manning provide an interesting data point for which to compare hand-crafted refinements of standard PCFG induction – most likely an upper bound on what humans can do with manual non-terminal splits.

In the late 1990s, the constituent parsing community saw rapid evolution (and competition) of lexicalized PCFGs, achieving new levels of accuracy. Collins' generative lexicalized parsing [45] and head-driven models [47], Goodman's probabilistic feature grammar [70], Charniak's parsing with word statistics [31] and maximum-entropy inspired parser [32] all

leveraged specific lexical items during grammar induction. An example of a pre-processed tree to include head lexiclaization annotations can be seen in Figure 2.3c. Because the feature space becomes so large with a lexicalized grammar, every model was heavily smoothed so that generalizability could be maintained. In addition, searching for the head words within the left and right child cells of Charniak and Collins bi-lexical models requires an additional $O(N^2)$ processing component, increasing overall complexity to $O(N^5)$ (although Eisner and Satta [60] provide alternative inference methods with $O(N^4)$ or, in special cases, $O(N^3)$). Even with these optimization, inference for each of the above lexicalized models required heavy pruning of the search space to maintain practical parsing efficiency as exact inference was no longer possible.

More recently, Matsuzaki et al. [116] and Petrov et al. [130, 132] have introduced latent variable grammars. Instead of manually refining the conditioning information of grammar productions as was done previously, each non-terminal is annotated with a set of latent variables (i.e., NP-0, NP-1, NP-2, ...) and their distributions are estimated via expectation maximization. Matsuzaki et al. set a fixed number of latent variables for all non-terminals, achieving their best results with 16 latent annotations per non-terminal.[7] Petrov et al. allow the number of latent annotations to fluctuate on a per non-terminal basis, using a split-and-merge technique, achieving significantly better parsing performance. Exact inference with latent-variable grammars is NP-hard [116], but $O(N^3)$ approximations such as Viterbi decoding, max-rule decoding [132], or variational approximation of the posterior forest [116] exist that perform well in practice. Although these latent variable grammars are large, they can be explicitly enumerated, facilitating easy distribution to others in the research community. In this thesis we use the latent variable grammar of Petrov et al. to perform high-accuracy parsing, allowing us to compare our efficient inference methods with those of the Berkeley parser (which is multi-level coarse-to-fine pruning).

Table 2.5 summarizes the evolution of PCFGs over the past 15 years. The table is formatted in three groups: grammars with manually-inspired annotations, lexicalized grammars, and latent-variable grammars. Parsing performance is reported on Section

[7] Computational resources were exhausted at 16 latent annotations, inhibiting larger experiments.

	Grammar	$\|G_{phrase}\|$	LP	LR	F_1
Manual	Vanilla	13,500	74.7	69.0	71.5
	Johnson 1998 [86]	30,000	-	-	78.6
	Parent + Head POS	56,000	-	-	80.8
	Klein et al. 2003 [102]	52,000	86.3	85.1	**85.7**
Lexical	Collins 1996 [45]	-	85.3	85.7	85.5
	Goodman 1997 [70]	-	85.3	84.8	85.0[1]
	Charniak 1997 [31]	-	86.7	86.6	86.6
	Collins 1999 [47]	-	88.1	88.3	88.2
	Ratnaparkhi 1999 [141]	-	87.5	86.3	86.9
	Charniak 2000 [32]	-	89.6	89.5	**89.5**
	Gildea 2001 [68]	-	86.6	86.1	86.3
Latent	Matsuzaki et al. 2005 [116]	-	86.1	86.0	86.1
	Petrov et al. 2006 [130]	1,841,000	89.8	89.6	89.7
	Petrov et al. 2007 [132]	1,841,000	90.2	89.9	**90.0**[2]

Table 2.5: A comparison of the grammar size ($|G_{phrase}|$), labeled precision (LP), labeled recall (LR) and F_1 accuracy on Section 23 of the Penn WSJ treebank for various PCFG models.

23 of the Penn WSJ treebank and the size of the grammar is documented if explicitly enumerable. The general trend of Table 2.5 is that modest gains in accuracy require an increase in grammar size and, consequently, a decrease in parsing efficiency. Unfortunately, the true modeling power of the lexicalized and latent-variable grammars are difficult to determine since exact inference is not performed (or possible), but the results in Table 2.5 show an interesting trend. Efficient PCFG parsing with high-accuracy grammars, such as those listed here, is the primary concern of this thesis The next chapter outlines current methods used for efficient inference, many of which are used by the parsers in Table 2.5.

[1] Goodman tested on sentences of length ≤ 40 words.

[2] Petrov et al. (2007) parse with the same latent-variable grammar as 2006, but improve the approximate inference technique.

Chapter 3

Current Methods for Efficient Inference

Even with the computational advantages of dynamic programming, exhaustive exploration of the entire search space may be infeasible when parsing with large grammars. For example, CYK parsing with the latent-variable grammar of Petrov et al. [130] requires over one minute of processing and 2GB of memory per sentence. Many methods for efficient inference have been proposed in recent years to improve upon the baseline efficiency of CYK parsing. We classify each method along two dimensions: exact versus approximate inference, and pruning versus prioritization.

Exact inference methods find the globally optimal solution given a grammar. The CYK algorithm is a classic example of exact inference as it computes the derivation probability for all possible trees, returning only the globally optimal solution. But exhaustive search over the entire solution space is not required for exact inference. As we will see later in this chapter, admissible prioritization methods such as A* or restricted classes of pruning can guarantee to find the exact inference solution without searching the entire space. More recently, Lagrangian relaxation methods have been applied to NLP problems, including parsing, with the ability to guarantee when the globally optimal solution is found [148, 29]. This does not guarantee that the exact solution will always be found, but in experiments with machine translation and joint dependency/constituency parsing, exact solutions are found on most data sets for over 98% of sentences [108, 147].

In contrast, *approximate inference* methods share no such formal guarantees. These methods make hard search decisions often based on local information to constrain the search space. The majority of efficient search algorithms we present in this thesis are approximate inference methods due to their typically superior performance on standardized

tasks. But finding the globally optimal solution — as is guaranteed by exact inference algorithms — has interesting theoretical advantages, such as the true modeling power of the grammar or the density of the solution space around the globally optimal solution. For example, the lexicalized grammars of Charniak and Collins are so large that encoding the grammar rules is done implicitly and exact inference becomes infeasible (see Section 2.9). Unable to find the exact inference solution, it is impossible to know how well their grammars truly model syntax or how their respective pruning strategies affect the search for this globally optimal solution.

The second dimension we classify efficient parsing methods along is pruning verses prioritization. *Pruning methods* make hard decisions during inference to exclude portions of the search space entirely. Once a derivation has been pruned, no paths from the pruned derivation will be extended for the remainder of inference. Pruning can either be integrated directly into the search, eliminating paths as they are expanded (i.e., beam-search), or pruning can be done by imposing constraints on the final search space via earlier stages of a pipeline (i.e., coarse-to-fine pruning or POS tagging). *Prioritization methods* never make hard, irreversible decisions to constrain the search space. Instead, they prioritize the order of search so that promising derivations are pursued first. In the worst case, all search paths may still be expanded before inference is complete, but given an accurate prioritization function, much of the search space may safely be ignored and significantly increase parsing efficiency.

In the following sections we discuss several influential methods for efficient parsing with PCFGs and situate them in the taxonomy we have just described. We then discuss the strengths and weaknesses of each search strategy and how our work in Chapters 4 - 6 address these problems.

3.1 Graph-Based Search

3.1.1 Best-First Parsing

Best-first parsers – also called agenda-based parsers – maintain a global agenda of edges and explore the search space in a best-first traversal. At each iteration, the highest-scoring

edge is popped off of the agenda and added to the chart. Edges are scored with a heuristic function $f(\cdot)$ comprised of two components: a known cost $g(\cdot)$ from the start node to the current position, and an unknown heuristic cost $h(\cdot)$ representing the remaining distance from the current node to the goal. We write this formally with respect to the non-terminal node A and the words $w_b \dots w_e$ derived by A in the sentence:

$$f(A, w_b \dots w_e) = g(A, w_b \dots w_e) + h(A, w_b \dots w_e) \tag{3.1}$$

With respect to PCFG parsing, $g(\cdot)$ is the accumulated Viterbi-max or inside score readily available during bottom-up parsing, while the heuristic function $h(\cdot)$ is more open ended, often crafted in a number of ways to suit domain-specific objectives. The most basic heuristic function assigns each edge a score of zero, forcing edges prioritized by $f(\cdot)$ to rely only on the inside score $g(\cdot)$. On the other hand, if $h(\cdot)$ assigned the oracle remaining derivation for each edge, parsing would be extremely efficient and the best-first parser would only consider the edges in the optimal parse tree. But knowing all oracle derivations requires exhaustive parsing, which is exactly what we are trying to avoid.

Crafting the heuristic function is an art that often relies on grammar or problem-specific knowledge. Two classes of heuristic functions exist: admissible and inadmissible. Admissible heuristics guarantee that $h(\cdot)$ never overestimates the true distance to the goal. From this restriction, it can be shown that best-first traversal with an admissible heuristic will always find the globally optimal solution [149, 112]. Admissible best-first search has been applied to constituent parsing and a number of heuristic functions proposed to ensure optimal search [101, 126].

In practice, heuristic functions that never overestimate the true distance to the goal and simultaneously provides effective ranking are difficult to engineer. An alternative approach is to relax the admissibility criteria, allowing the heuristic function to more closely approximate the true distance to the goal – sometimes underestimating and other times overestimating. This is the approach taken by many best-first parsers, including that of Kay [98], Shieber et al. [160] Bobrow [13], Magerman and Weir [111], Caraballo and Charniak [25], Charniak et al. [33], and Hall [73].

In Chapter 4 we discuss the best-first parsing algorithm in detail, including practical

modifications such as "over-parsing" [74] (the continuation of parsing after the goal node is discovered to increase the density of the chart for n-best extraction or to find a more optimal solution give an inadmissible heuristic) and memoization (recording agenda entries for fast exclusion). Even with such modifications, best-first parsing can be relatively inefficient due to the overhead of maintaining a sorted agenda and the cost of computing complex prioritization values for each of an exponential number of constituents. Furthermore, as discussed in Blaheta and Charniak [12], normalization of the prioritization function for constituents spanning differing substrings is problematic within the global agenda of best-first parsers.

In Chapter 4 we present solutions to the above problems and support our hypotheses with empirical results. Specifically, we reduce the complexity of computing the boundary tri-gram heuristic [25] from $O(M^2)$ to $O(1)$ where M is the number of POS tags by introducing a novel decomposition of the heuristic function. We also demonstrate the problems with proper normalization of multiple prioritization functions and contrast that to pruning methods such as beam-search where such normalization is irrelevant.

3.1.2 Beam-Search Parsing

Another popular efficient search method for PCFG parsing is beam-search. Beam-search is a standard graph-based search algorithm that makes hard pruning decisions at each span. In a bottom-up breadth-first traversal, all candidate derivations are sorted by a heuristic scoring function and only the most promising candidates are retained. The size of this set is determined by the *beam-width*, which has been defined in two ways [47, 185]:

1. A constant value K such that the K-best candidates are retained;

2. A constant threshold R between 0 and 1 such that all candidates with a heuristic score greater than $R\hat{\pi}$ are retained, where $\hat{\pi}$ is the highest scoring local derivation.

There are two key points we wish to highlight when comparing the best-first and beam-search algorithms. First, both algorithms sort edges in a priority queue within the inner-most loop, increasing the complexity of both best-first and beam-search from $O(N^3)$ to $O(N^3 \log N)$. This increases the parsing complexity over the CYK algorithm, but in

practice often decreases actual parsing time due to pruning or prioritization of the search space. Within a best-first architecture, the global agenda potentially maintains a sorted list of all $O(N^3|G|)$ edges, while beam-search parsing sorts at most $O(N|G|)$ edges. As we will see in Chapter 4, this has practical implications on the time and memory of parsing.

Second, it is clear that the same heuristic ranking functions can be used either within the global agenda (best-first) or within the span-level agenda (beam-search). Although heuristic ranking with beam-search is popular in speech-recognition [109, 123] and machine translation [105, 125], it is much more common in the parsing community to apply heuristic functions to best-first parsing. The utility of these heuristic functions to prune at a local level has not been fully explored and many state-of-the-art parsing systems only use simple heuristics when pruning at this level [70, 47]. Furthermore, when sophisticated techniques have been used within the beam-search framework [179] it is unclear how they perform relative to best-first architectures or if performance gains are tied to a particular grammar.

In Chapter 4 we present pseudo code for the beam-search algorithm and discuss it in greater detail, especially with respect to the exhaustive CYK algorithm and best-first discussed in the previous section. In We also empirically compare beam-search with best-first search across many different heuristic ranking functions and grammars and discuss their strengths and weaknesses with respect to PCFG parsing. To our knowledge, this evaluation is the first extensive comparison of these two approaches for PCFG parsing.

3.2 Coarse-to-Fine Parsing

In addition to graph-based search algorithms, hierarchical search and pruning methods for syntactic parsing, such as coarse-to-fine pruning, have also shown significant promise. Coarse-to-fine parsing [70] utilizes the idea of first quickly searching with an impoverished model, then using the posterior scores from preliminary search to prune (or guide) the subsequent search in the full model space. Hierarchical search algorithms have proven to be effective in many domains, including image processing [62], speech recognition [53, 165], and route planning [153]. Within context-free parsing, coarse-to-fine search is accomplished by projecting the target grammar onto a reduced grammar space, collapsing context-specific

rules into a less accurate model that is much faster to search. The scores of the projected grammar can be created such that the search with the final model is still globally optimal [15, 126], but this approach is often less effective than non-exact coarse-to-fine search. In most cases, coarse-to-fine is used for approximate inference to favor efficiency over optimality [70, 34]. Coarse-to-fine inference can also be applied iteratively with multiple grammars of increasing complexity [132].

Building a coarse grammar from a fine grammar is a non-trivial problem, and most often approached with detailed knowledge of the fine grammar being used. For example, Goodman [70] suggests using a coarse grammar consisting of treebank non-terminals, such as NP and VP, and then non-terminals augmented with head-word information for the more accurate second-pass target grammar. Such an approach is followed by Charniak [32] as well. Petrov and Klein [132] derives coarse grammars in a more statistically principled way, although the technique is closely tied to their latent variable grammar representation.

The efficient optimization methods we present in Chapters 5 and 6 are specifically designed to require no knowledge of the underlying grammar. This allows our algorithms to be easily adapted to new languages or domains without the need to derive a suitable coarse grammar. But if a coarse grammar exists, or a natural projection to a coarse grammar space is available, the efficient search methods we will present later in this thesis prune the search space in orthogonal ways compared to coarse-to-fine pruning and we expect the efficiency gains to be additive.

As both the best-fist and beam-search algorithms are presented in Chapter 4, we have also included pseudocode for two-level coarse-to-fine parsing in Appendix C for completeness.

3.3 Pipeline Systems

Pipeline systems, in which data is processed in stages with the output of one stage providing input to the next [80], have also been applied to syntactic parsing. Coarse-to-fine pruning – when seen as a chain of increasingly complex grammars – is an instance of pipeline parsing, although traditional pipeline systems are often composed of significantly different

architectures (i.e., finite-state tagger to context-free parser). Two common pipeline systems to increase the efficiency of PCFG parsing are (1) part-of-speech tagging to reduce the number of paths from each start node [141], and (2) shallow chunking to constrain constituent spans [69]. For example, consider shallow NP chunking as a pre-processing step to parsing. Given the constraints of where noun phrases can begin and end in the dynamic programming chart, not only does this eliminate alternative NP structure outside these constraints, it also disallows all other constituent structure crossing brackets with the proposed NP structure.

Supertagging is another pipeline-pruning method that has gained popularity to decrease the typical-case runtime of LTAG parsing [7, 155] and CCG parsing [40]. Given the surrounding context of each lexical item, supertagging limits the ambiguity of elementary trees by pruning those that are unlikely to participate in a full derivation. This approach is similar to pre-tagging the input with POS tags (1-best or n-best) to constrain the downstream parser, but operates on the larger set of possible elementary trees for each lexical item instead of the set of possible POS tags.

In Chapter 5 we present an extension of the pipeline system of Roark and Hollingshead [145, 146] to constrain context-free parsing by limiting all constituent beginning and ending positions, as well as possible locations for unary productions. Our technique is inspired by previous work to constrain portions of the dynamic programming chart via NP-chunk constraints, but generalized to limit the possible location of any multi-word constituent. In Chapter 6 we continue with the theme of constraining cells in the dynamic programming chart, but instead of restricting multi-word constituent start and end locations (diagonals in the dynamic programming chart), we predict the unlabeled constituent structure by classifying individual chart cells as open or closed to possible constituents.

Each of the pipeline systems discussed above are approximate inference search methods as there is no guarantee that the globally optimal parse tree will not be pruned given the pre-processing constraints. But as we will demonstrate in Chapters 5 and 6, some forms of pruning may actually increase the accuracy of other objectives, such as constituent precision and recall compared to a gold standard, if the pruning leverages additional information not present in the grammar. In addition, we will show that our pipeline

systems provide orthogonal efficiency gains to graph-based and coarse-to-fine pruning, allowing significant inference speedups when these search techniques are combined [17].

3.4 Discriminative Parsing and Re-Ranking

In Section 2.9 we examined the evolution of PCFGs and the inclusion of additional conditioning information within derivation rules. Although methods such as lexicalization allow the grammar to be sensitive to subcategorization preferences, possible conditioning information is limited to local features if we wish to take advantage of dynamic programming. In order to score parse trees using non-local features – indeed, any arbitrary feature of the input or parse forest – one can more naturally train a discriminative model to estimate $P(t|S)$ opposed to the joint distribution $P(t, S)$. Unfortunately, the power of dynamic programming is marginalized or often lost under this framework, and parsing efficiency dramatically decreases due to the inability to decompose the total score of the tree into the sum of its parts.

Approaches to handle the compromise between efficient training and inference with possible feature representation fall into three categories. The first field of research, exemplified by [166, 170], resign themselves to the limitation of efficiency and focuses on discriminative parsing with short sentences (often less than 15 words). Even with state-of-the-art accuracy on this sub-domain, parsing time is impractical for real-time applications or web-scale corpora.

On the other extreme, only local features encapsulated within a PCFG are used to maintain efficient training and inference, but the scores of each PCFG production are trained discriminatively, maximizing the conditional likelihood of the data [100, 76, 134]. This approach is unable to incorporate additional non-local features into the grammar, but the success of discriminative over generative training in other NLP tasks suggests that accuracy may improve. Although training is still slow compared to generative approaches (requiring expectations over all possible derivations in the training data), decoding remains efficient as standard $O(N^3)$ algorithms can still be applied. Therefore, all efficient inference methods we present in this thesis are applicable to parsing with these discriminatively

Parser	**LP**	**LR**	**F_1**
Taskar et al. [166] (2004)	89.1	89.1	89.1[1]
Turian and Melamed [171] (2006)	89.6	89.3	89.4[1]
Collins [48] (2000)	-	-	89.7
Charniak and Johnson [34] (2005)	-	-	91.4
Huang [84] (2008)	-	-	91.7
Henderson [76] (2004)	-	-	90.1
Petrov and Klein [134] (2008)	-	-	88.3

Table 3.1: A comparison of the accuracy of three methods on WSJ Section 23: discriminative parsing, discriminative re-ranking, and discriminative training of PCFG models.

trained PCFGs.

The third approach to discriminative parsing is to extract the k-best parse trees from a generative parser and use discriminative re-ranking to re-score each of the k trees individually [48, 34]. Because k is relatively small (often 50 or 100 trees), re-ranking is much faster than full discriminative parsing and parsing time is dominated by the first-pass generative phase. Huang [84] presents an approximate algorithm to re-rank the entire forest (i.e., set of all possible derivations) by factoring features into local and non-local sets. By computing all local scores with dynamic programming and non-local scores on-the-fly, he is able to discriminatively train his model using the entire treebank, achieving impressive accuracy on the standard WSJ test set. The efficient parsing methods we present in Chapters 4-6 are not in conflict with discriminative re-ranking paradigms; one can prune the search space and still extract k-best trees or re-rank the entire forest.

Table 3.1 reports the accuracy of canonical examples of each of the three discriminative parsing approaches on WSJ Section 23. Accuracy results are directly comparable for the re-ranking and discriminative training of PCFG paradigms, but the test set for full discriminative parsing is reduced to only include sentences of length less than 15 words. Although we limit the application of our efficient optimization methods to generative

[1]Tested on sentences less than 15 words.

models in the remainder of this thesis, we emphasize that such methods can also be used to improve the efficiency of n-best extraction for discriminative re-ranking as well as parsing with discriminatively trained PCFGs.

3.5 Search as Classification

The efficient inference methods discussed in this chapter – and the new methods we will present later in this thesis – are predicated on the assumption that the grammaticality of language is modeled by a grammar formalism (PCFG, TAG, CCG, etc.) and we are searching for the optimal parse tree given this model. Instead of inducing a grammar from a treebank, an alternative approach to constituent parsing is to model the derivation process explicitly. Here, the probability of a parse tree is the accumulation of parser actions given the respective state:

$$P(t|S) = \prod P(a_i|s_i) \tag{3.2}$$

where t is a parse tree, S is input sentence, and a_i is the action taken by the parser at state s_i. Each state s_i can incorporate any number of previous parser actions or elements of the input sentence. One notable difference between PCFG parsing and classification-based parsing is that of consistency. With a classification-based parser, there may be multiple derivation sequences that derive the same final parse structure, but in no way are they required to produce the same score. A PCFG model, on the other hand, will consistently assign the same score to identical parse trees, even if how the parse trees are constructed differs.

The majority of parsing by classification work follows a shift-reduce parsing paradigm [113], where at each state, the parser chooses to push a new input word onto the stack (shift), or reduce elements at the top of the stack according to rules in the grammar. Predicting which action(s) to pursue is the learning goal, accomplished in the past by a host of machine learning tools including neural networks [161], decision trees [77, 178, 93], and support vector machines [150]. Although accuracy has generally been lower with classification-based parsers as compared to PCFG models on standardized tasks, the

English WSJ Treebank				
Parser	**LP**	**LR**	**F_1**	**Words/Sec**
Simmons and Yu [161] (1990)	-	-	-	-
Ratnaparkhi [140] (1997)	87.5	86.3	86.9	30
Hermjakob and Mooney [77] (1997)	84.9[1]	84.6[1]	84.8[1]	-
Kalt [93] (2004)	77.2	75.9	76.5	33,560
Sagae and Lavie [151] (2006) Accurate	88.1	87.8	87.9	55
Sagae and Lavie [151] (2006) Fast	85.4	84.8	85.1	1,000+
Chinese Treebank				
Parser	**LP**	**LR**	**F_1**	**Words/Sec**
Wong [178] (1999)	77.7	78.9	78.3	500

Table 3.2: Accuracy and timing comparison of classification-based shift-reduce parsers. Timing results do not control for computer hardware or implementation details (including programming language) and are only meant to give a general sense of parsing efficiency.

framework allows for very efficient $O(N)$ runtime performance, and Kalt (2004) reports parsing speeds of over 30,000 words per second [140, 93, 150]. The same framework has also become popular for dependency parsing [181, 124], primarily due to the same $O(N)$ efficient decoding algorithms providing parsing speeds of over 10,000 words per second.

In Table 3.2 we list a number of prominent shift-reduce parsers that search via classification. When reported, we also list the labeled precision (LP), labeled recall (LR), and words per second on WSJ Section 23, with the exception of Hermjakob and Mooney [77] who evaluated on a smaller subset of the WSJ test set, and Wong [178] who report results on the Chinese treebank. Timing results do not control for computer hardware or implementation details (including programming language) and are only meant to give a general sense of parsing efficiency.

We believe that deterministic parsing by classification algorithms such as those listed in Table 3.2, as well as less expressive formalisms, such as finite-state approximations of

[1]Hermjakob uses non-standard subsets of the WSJ treebank for training and testing, with shorter sentences on average.

phrase structure grammars [127] – both which run in linear time – have serious promise for future research in efficient constituent parsing. But for this present work, we focus on PCFG models due to their superior accuracy and adaptability to new domains [131].

3.6 Evaluation of Efficient Parsing Methods

In the following chapters, we compare our proposed efficient parsing methods with a selection of those described in the previous sections. Where possible, we compare competing methods by running tests under identical conditions – using the same grammar models, computer hardware, and programming language – such that we can isolate the contribution of the proposed algorithms in isolation. In some cases this is not practical and we will match as many conditions as possible to make the comparison meaningful.[1] We will also apply our efficient search methods to PCFG decoding with grammars from multiple languages in order to demonstrate that our algorithms are truly adaptable. Additionally, we will also show that unlabeled constituent and beam-width prediction (Chapter 6) can be adapted to a target domain even when labeled data is not present.

3.6.1 Languages, Grammars, and Data Sets

In this thesis we apply our efficient search algorithms to three languages: English, German, and Chinese. Each of these languages has established treebanks from which it is common to report parsing accuracy and efficiency statistics to facilitate comparison. In particular, Petrov [129] has made high-accuracy latent-variable grammars publicity available for these languages which allows direct comparisons between coarse-to-fine pruning and our methods using the same grammar. Because our efficient search methods are entirely data driven, in one respect it is irrelevant what languages we apply our methods to – our algorithms are simply trying to search the exponential hyper-graph space in the most efficient manner. But due to the extent that lexical information is used to direct the search, we hypothesize that some languages may benefit more than others – both with respect to efficiency and

[1] For example, comparison with the Charniak parser and grammar model would require either (1) integration of our efficient search methods into the existing Charniak parser codebase or (2) re-implementation of our BUBS parser in C++ with the same implicit grammar model.

					Sent Length	
Treebank		**Partition**	**Sent**	**Words**	**Mean**	**SD**
English WSJ [115]	Train	Sections 2-21	39,832	950,028	23.8	11.2
	Dev	Section 22	1,700	40,117	23.6	11.0
	Test	Section 23	2,416	56,684	23.5	11.1
Chinese [180]	Train	Articles 1-270,400-1151	18,086	493,708	27.3	19.4
	Dev	Articles 301-325	350	6,801	19.4	13.6
	Test	Articles 271-300	348	8,008	23.0	18.5
German [162]	Train	Sentences 1-18,602	18,602	328,418	17.7	11.2
	Dev	Sentences 18,603-19,602	1,000	17,165	17.2	11.4
	Test	Sentences 19,603-20,602	1,000	17,787	17.9	10.4

Table 3.3: Data sets used in this thesis

accuracy – due to the predictability of constituent boundaries within the language.

In Table 3.3 we list the treebanks and training, development, and testing set divisions for each. We use these data sets and divisions for the remainder of the thesis and will refer the reader back to this subsection at the appropriate time for clarification. Statistics for each division of the data are also reported. The count, mean, and standard deviation are all computed over tokens in the treebank, where a token is defined by the treebank guidelines. In the case of English and German, tokens are closely related to white-space delimited text with a few exceptions such as contractions and parenthesis [115]. In Chinese, where word segmentation is unnatural due to the lack of word delimiters and infectional morphology, tokens in the treebank are annotated based on syntactic independence (called a syntactic atom) which include both simple words and compounds [180]. The listed token statistics show inter-language and intra-language differences, which should be kept in mind when comparing efficiency results and parsing complexity is polynomial in the sentence length (number of tokens).

In Table 3.4 we list the grammar induction methods used throughout this thesis, as well a short-hand label for each. Details on grammar binarization and markovization can

Label	Grammar Description
R0	Right-binarized Markov-order 0 grammar
R2	Right-binarized Markov-order 2 grammar
R2P1	Right-binarized Markov-order 2 grammar with Parent annotation [86]
L0	Left-binarized Markov-order 0 grammar
L2	Left-binarized Markov-order 2 grammar
L2P1	Left-binarized Markov-order 2 grammar with Parent annotation [86]
Latent	Latent variable grammars trained with six split/merge cycles for English and five for non-English. [133]

Table 3.4: List of grammars and short-hand label used in this thesis

be found in Section 2.7 and 2.8 respectively. The main impetus of evaluating our efficient search methods across a variety of grammars is to measure the effects of grammar size and structure (including modeling accuracy) within our efficient search framework. Questions we will address are:

1. How are efficiency gains affected with respect to grammar size (number of productions)?

2. Can pruning or prioritizing the search directly on lexical information improve accuracy of the final F_1 objective over the maximum likelihood solution given the grammar?

3. When lexical information is already present in the PCFG, can our efficient search methods still improve accuracy?

3.6.2 Accuracy Evaluation

Parsing accuracy is reported using the standard PARSEVAL [1], which computes the labeled precision and recall of constituents (also called brackets) between test and gold parse trees. A constituent here is defined as a tuple (b, e, x) where b is the index of the left-most word covered by the constituent, e is the right-most word covered by the constituent, and x is the constituent's label (i.e., NP). To condense the accuracy into a single number,

we also report F_1, which is the harmonic mean between precision and recall. Note that both POS constituent and the unary production with $S^\dagger$ on the left-hand side are ignored in this evaluation; POS accuracy is computed separately and all trees are labeled with a root node, so using it during evaluation would artificially inflate the actual accuracy of prediction. We also follow Collins [46] and collapse PRT to ADVP during evaluation, but retain this distinction during modeling and decoding.

Recall measures the number of constituents in the gold tree that were correctly identified, and precision measures the fraction of hypothesized constituents that were correct. Assuming t is the hypothesized tree and t^* is the gold tree, we compute precision, recall, and F_1 as follows:

$$Precision(t, t^*) = \frac{\text{Number correct constituents in } t}{\text{Number constituents in } t} \tag{3.3}$$

$$Recall(t, t^*) = \frac{\text{Number correct contituents in } t}{\text{Number constituents in } t^*} \tag{3.4}$$

$$\mathrm{F}_1(t, t^*) = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.5}$$

3.6.3 Efficiency Evaluation

We measure the efficiency of each method by reporting the number of words parsed per second. All experiments are run on an Intel 3.00GHz processor with 6MB of cache and 16GB of memory. Multiple trials are timed over the same data set, and the trial with the fastest recorded time (highest words per second) is retained. We keep the fastest time — opposed to, say, the average time — because any increase in latency is not caused by the algorithm, but rather the underlying state of the operating system, which we do not have full control over. The fastest time represents the preferred state of the computer and ignores inflated timings due to unrelated background processes or non-optimal cache content.

We note that reporting words per second is not ideal due to the per-sentence processing

of constituent parsing. Fifty one-word sentences will parse much faster than a single fifty-word sentence due to the cubic complexity of the CYK algorithm. Because of this, we also report the mean and standard deviation of sentence lengths for each data set in Table 3.3, and plot the parsing speed as a function of the sentence length in many of our experiments. We feel that modifying our efficiency measure to account for the length of each sentence (such as normalizing by the sentence length) would unnecessarily obfuscate the data. In addition, others in the parsing community (especially dependency parsing) also report words per second on the same data set, thus making a direct comparison possible.

Chapter 4

Prioritization for Efficient Search

4.1 Introduction

We discussed in Chapter 3 how both the best-first and beam-search parsing architectures can leverage a prioritization function $f(\cdot)$ to guide the local search towards paths that are more globally optimal by considering information outside the constituent's span. It has been shown multiple times that a good prioritization function can significantly increase parsing efficiency [104, 12, 33, 101, 126], but the application of these metrics are primarily used in a best-first parsing architecture (also called agenda-based parsing). The transfer of these techniques to beam-search inference is popular in fields such as machine translation [105, 106], but has not "caught on" to the same extent for context-free parsing, despite past work [70, 179], most likely due to a lack of empirical evidence supporting one method over another. Furthermore, Caraballo and Charniak [25] note that comparing edges covering different spans (as is done in a global agenda) is not always appropriate because it is unclear that, e.g., non-terminal A_1 spanning words $w_1 \dots w_4$ should be ranked higher or lower than non-terminal A_2 covering words $w_3 \dots w_{16}$, as each may be equally appropriate given their respective context. Within a beam-search architecture, only constituents covering the same substring (span) are compared, eliminating the problem of comparability inherent in best-first parsers.[1]

In this chapter we investigate a representative set of prioritization methods and apply

[1] This is analogous to the problem solved by multi-stack decoder in the machine translation literature [67].

each to best-first and beam-search context-free parsing with grammars of different composition. We focus on two key contributions. First, we apply best-first prioritization to beam-search parsing within a controlled environment and analyze the respective strengths and weaknesses of each architecture. To our knowledge, this is the first extensive analysis of these competing efficient search strategies for context-free parsing. Second, we introduce two prioritization functions: the Constituent Boundary POS model and the Constituent Boundary Lexical model. The former is an extension of Caraballo and Charniak [25] that does not require gold part-of-speech tags. Instead, we follow [74] and apply the forward-backward algorithm to compute probabilities over the ambiguous POS sequence and efficiently store the boundary scores such that prioritization access is $O(1)$. The latter method computes transition probabilities from lexical items directly, which alleviates the non-trivial forward-backward computation, but can encounter sparse-data problems with large grammars. We find that clustering lexical items is an effective solution to alleviate the large parameter space, increasing the accuracy when parsing with the latent variable grammar so long as the number of clusters is sufficiently large (greater than 64 in our experiments).

4.2 Prioritization Functions

We can prioritize best-first search given some function $f(\cdot)$ that assigns a score to any non-terminal covering a substring of the sentence. Because sub-derivation scores are tabulated while processing the CYK algorithm, partitioning $f(\cdot)$ into two components is a natural decomposition, where $g(\cdot)$ is the exact model score from the initial-to-current position in the derivation, and $h(\cdot)$ is a heuristic estimate of the remaining distance (i.e., model score) from the current position to the end goal. Representing non-terminal A spanning words $w_b \dots w_e$ as $A_{b,e}$, we can write:

$$f(A_{b,e}) = g(A_{b,e})h(A_{b,e}) \tag{4.1}$$

The true value of $h(\cdot)$ can be computed in $O(N^3)$ time using the outside algorithm [6], but requires exhaustive inside/outside parsing which is exactly what we are trying to

avoid during efficient inference. Instead, we wish to estimate the true outside probability of constituents so that prioritization is both effective and efficient. Because the functions $g(\cdot)$ and $h(\cdot)$ aim to represent the true inside and outside scores of a derivation in PCFG parsing, we will interchangeably use $g(\cdot)$ with α and $h(\cdot)$ with β, and can rewrite Equation 4.1 as follows, where $\hat{\alpha}$ represents an estimate of the true value α and likewise for β:

$$f(A_{b,e}) = \hat{\alpha}(A_{b,e})\hat{\beta}(A_{b,e}) \tag{4.2}$$

In what follows, we first introduce four distinct methods to estimate the outside probability β, and then proceed in Section 4.3 through 4.5 to analyze the merit of each within a best-first and a beam-search framework.

4.2.1 Inside

The simplest method to prioritize constituents is to ignore the outside contribution entirely. We call this method *inside* prioritization:

$$f_{inside}(A_{b,e}) = \alpha(A_{b,e}) \tag{4.3}$$

Note that the inside probability α is the exact inside score – not an estimate – as it is naturally tabulated during bottom-up parsing. Because the outside contribution β is constant, this prioritization function does not consider how well the constituent may fit into the global parse structure required for a full derivation. The magnitude of the inside score is also relative to the number of grammar productions in its derivation. Within a best-first parser, this will bias the search to explore short substrings (even those with low probability) before large substrings.

4.2.2 Normalized Inside

To combat the bias of short versus long derivations when prioritizing by the inside probability, we can normalize the inside score by the length of the substring. Caraballo and Charniak [25] estimate the per-word inside probability by computing the geometric mean of the original results:

$$f_{geometric}(A_{b,e}) = \sqrt[e-b+1]{\alpha(A_{b,e})} \tag{4.4}$$

We find that computing the geometric mean skews the distribution of constituent scores so that short constituents have a much larger dynamic range compared to longer constituents. Said another way, the absolute difference between the largest and smallest $f_{geometric}$ scores for short spans is a much bigger than for long spans (see Figure 4.1b for a visual example). When competing in a single global agenda, low scoring short constituents are very unlikely to be placed in the chart without exhaustive parsing. This normalization method thus has the opposite effect from the unnormalized inside score in that longer constituents are preferred over shorter ones. To better control the bias between constituents spanning shorter or longer substrings, we instead normalize the inside probability as follows:

$$f^{\lambda}_{prod}(A_{b,e}) = \alpha(A_{b,e})e^{\lambda(e-b+1)} \tag{4.5}$$

We refer to this method as the Normalized Inside Product (due to its representation in the log domain where we often operate in practice). Here, the tuning parameter λ allows us to bias the agenda parser towards favoring shorter or longer constituents, while maintaining the dynamic range of inside scores across all spans (Figure 4.1 visually depicts how these two methods differ, which we will discuss at the end of this section). The exponentiation of the span length is motivated by the nature of the bottom-up parsing traversal. In essence, the dynamic range of the normalized scores are reversed compared to geometric mean normalization — short spans now cover a smaller absolute range compared to longer spans. But this is preferred in bottom up parsing as shorter spans must be added to the chart first before they expand the frontier to larger spans. As we will later show, this (tuneable) bias towards shorter spans leads to superior empirical results relative to geometric mean normalization when used for best-first parsing.

A third method we investigate to normalize the inside score is the use of an n-gram model to estimate the probability of the substring spanned by the constituent given the remainder of the sentence, namely $P(w_b \dots w_{e-1} | w_1 \dots w_{b-1}, w_e \dots w_n)$. We compute the

Normalized Inside N-Gram estimate as follows:

$$f_{ngram}(A_{b,e}) = \frac{\alpha(A_{b,e})}{\prod_{i \in b \ldots e-1} P(w_i | w_{i-1} \ldots w_0)} \tag{4.6}$$

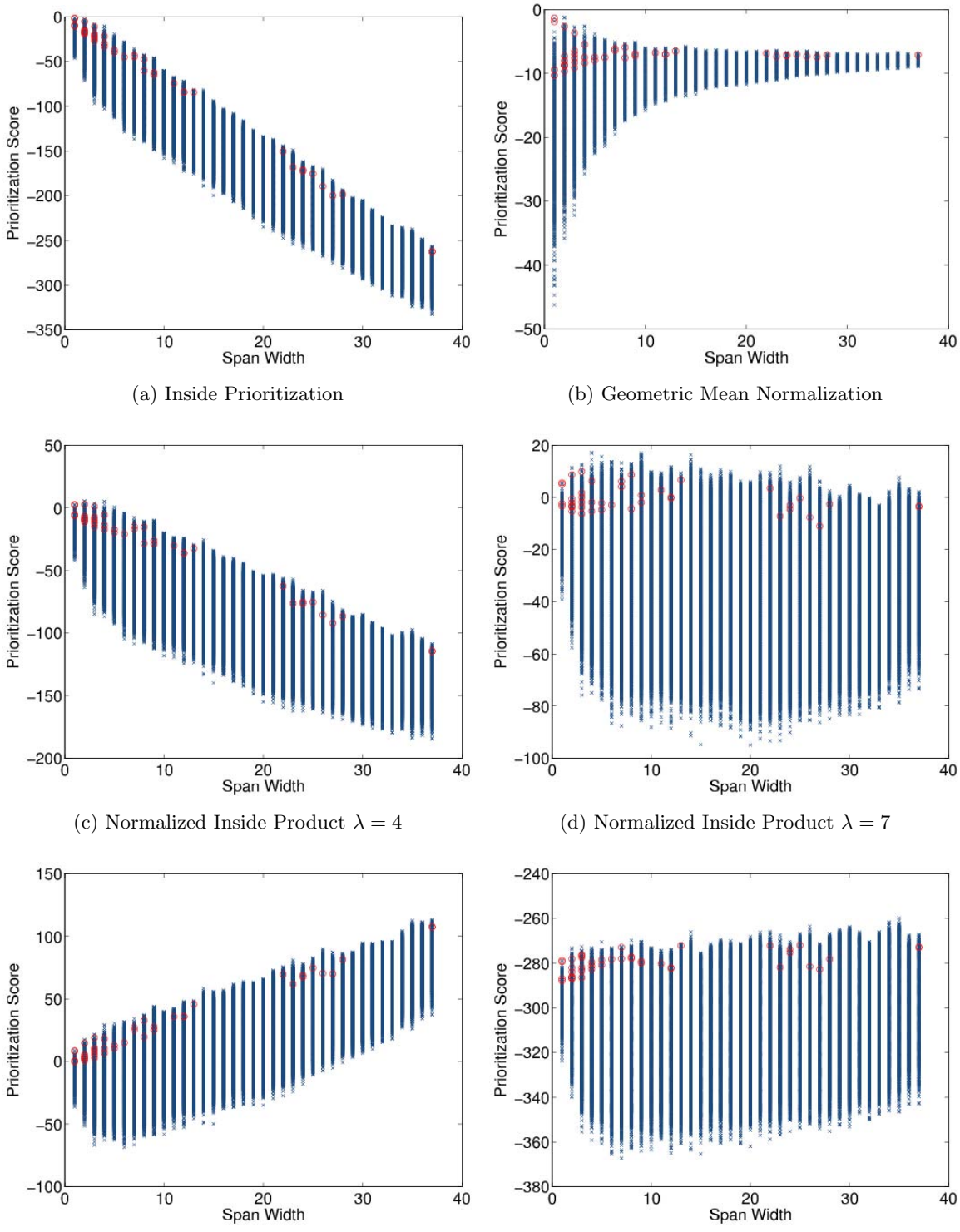
As usual, we approximate the true sequence probability with an N^{th}-order Markov approximation. Through experimentation, we found that higher order n-gram models do not improve prioritization performance. Therefore, we use a unigram model in all reported experiments due to its efficiency.

These three inside-score normalization methods are only useful within best-first parsers as constituents covering all spans are prioritized together in a single global agenda. With beam-search parsing, constituents are only compared locally, within a span-specific agenda. Because the span length $e - b + 1$ and spanned substring $w_b \ldots w_e$ are identical for all constituents within a given chart cell, the prioritization order will remain identical and applying the normalized inside prioritization function to beam-search parsing will have no effect compared to using the inside score alone.

Figure 4.1 displays a visual representation of each inside normalization method. We exhaustively parse a single sentence and plot the normalized inside score for each constituent given its span width. The constituents from the gold tree are drawn as red circles; all other constituents are each a blue "x". Constituents covering different substrings of the same span-width are plotted together in these figures (for additional caveats, see Section 4.2.7).

It may be said that for each of these methods, we would expect to see more data points in the lower right of each plot — poorer scores for derivations that span a larger portion of the input string, due to being composed of other poorer sub-derivations. If we were to plot *all* derivations possible given the input string and grammar, this indeed would be true. But since our search of possible derivations uses dynamic programming, only the highest scoring local derivation is retained for each non-terminal. This leads to each derivation necessarily being composed of only locally optimal derivations of smaller spans, hence we do not see the accumulation of poor derivations as these are, in essence, pruned.

Because a best-first agenda parser pursues frontier entries with the highest prioritization score, the absolute score for each constituent is important, regardless of its span



(a) Inside Prioritization

(b) Geometric Mean Normalization

(c) Normalized Inside Product $\lambda = 4$

(d) Normalized Inside Product $\lambda = 7$

(e) Normalized Inside Product $\lambda = 10$

(f) Normalized Inside N-Gram

Figure 4.1: Prioritization scores per span-width of all constituents during exhaustive parsing of sentence 1 in WSJ Section 22 with the R2 grammar. Red circles indicate constituent from gold parse tree.

length. We can see in Figure 4.3a that the unnormalized inside prioritization function is far from ideal. The final gold constituent covering the entire string has a (log) prioritization score of approximately -250. In order to pop this constituent from the global agenda, all constituents with a higher prioritization scores must necessarily be processed first. This requires processing all constituents with a span length less than 20, and the majority of those over length 20. Although fewer constituents will be processed compared to exhaustive CYK search, the total time-to-parse will be much greater due to the required sorting of constituents in the global agenda.

In contrast to the unnormalized inside score, we see that the Geometric Mean, the Normalized Inside Product, and the Normalized Inside N-Gram methods all alter the absolute prioritization score of each constituent such that the bias towards shorter derivations is lessened. The Normalized Inside Product method is also depicted at three different values of λ, showing how the bias towards long or short spans can be controlled. But even in the ideal case where the bias towards shorter spans could be ignored, the relative ranking of the gold constituents (red circles) will never change within a chart cell for any of these methods, and may only change across spans of the same width for the Inside N-Gram method. In general, this leads to a poor local prioritization of constituents for both the normalized or unnormalized inside prioritization functions. It is not until we supplement the inside constituent score with a more sophisticated outside estimate that we will begin to see significant improvements in local constituent ranking. The next three sections discuss such methods.

4.2.3 Constituent Prior

The outside probability β represents the likelihood of non-terminal $A_{b,e}$ participating in the remaining parse structure covering words $w_1 \dots w_{b-1}$ and words $w_{e+1} \dots w_n$. Formally,

$$\beta(A_{b,e}) = \mathrm{P}(w_{1\dots b}, A_{b,e}, w_{e\dots n}) \tag{4.7}$$

If we make the simplifying assumption that nonterminal A is independent of the words in the sentence, we can reduce the outside probability β to the prior over non-terminal

A participating in any parse tree [72]. We call this the *constituent prior* prioritization function:

$$f_{prior}(A_{b,e}) = \frac{\alpha(A_{b,e})\mathrm{P}(A)}{\prod_{i \in b \ldots e-1} P(w_i | w_{i-1} \ldots w_0)} \tag{4.8}$$

Although the assumptions present in this model are far from realistic, computing the prior probability over each non-terminal $\mathrm{P}(A) = \frac{count(A)}{count(*)}$, where $count(*)$ is the total number of non-terminal instances in the training data, is trivial to estimate and can be seen as a practical compromise between using the inside probability alone and the more complex prioritization models we present in the following sections. Surprisingly enough, we will see later in this chapter that the Constituent Prior model performs nearly as well as its more complex counterparts when performing beam-search with small grammars. This model provides significant computational advantages over the inside probability alone, with very little added complexity.

4.2.4 Constituent POS Boundary

We can improve the outside estimate β to condition its estimate on more than the prior probability over non-terminals. Caraballo and Charniak [25] (C&C) introduce the boundary n-gram prioritization function (also known as a figure-of-merit), which relies on two key factors: (1) an n-gram estimate of the probability of the unambiguous POS tag sequence $\mathrm{P}(t_1 \ldots t_n)$, and (2) the probability that the non-terminal A will be situated in the parse tree with POS tag t_b to the left of its span and t_e to the right. Given unambiguous POS tags $t_1 \ldots t_n$, we write the C&C estimate as

$$f_{pos}(A_{b,e}) = \frac{\alpha(A_{b,e})\mathrm{P}(A|t_{b-1})\mathrm{P}(t_e|A)}{\prod_{i \in b \ldots e-1} \mathrm{P}(t_i | t_{b-1} \ldots t_0)} \tag{4.9}$$

C&C assume gold POS tags in their experiments, which simplifies the calculation of their estimate. To accommodate parsing from lexical items without gold POS tags, we can either tag the input sentence as a pre-processing step or leave the sequence ambiguous and compute the forward-backward probability of each tag at each position, and directly incorporate this information into the prioritization model. We note that Magerman and

Weir [111], Collins [46], and Charniak [31] do not parse from pre-processed 1-best POS tags due to significant F_1 degradation (0.6 absolute from Collins). Instead, we follow Hall and Johnson [74] and incorporate the forward-backward POS probabilities as follows:

$$f_{pos}(A_{b,e}) = \max_{i,j}[\frac{\alpha(A_{b,e})\mathrm{P}(A|t^i_{b-1})\mathrm{P}(t^j_e|A)}{fwd(t^i_{b-1}\dots t^j_e)bkwd(t^i_{b-1}\dots t^j_e)}] \tag{4.10}$$

Instead of dividing the the forward-backward scores of the tag sequence spanned by $A_{b,e}$, we can instead multiply by its complement – the forward score from the beginning of the sentence to the beginning of the span, and the backward score from the end of the span to the end of the sentence. A similar approach is taken by [74], but here we also decompose β into a left and right component which can be independently computed:

$$\begin{aligned}
\beta_{left}(A_{b,e}) &= \max_i[fwd(t_0\dots t^i_{b-1})\mathrm{P}(A|t^i_{b-1})] &(4.11)\\
\beta_{right}(A_{b,e}) &= \max_j[\mathrm{P}(t^j_e|A)bkwd(t^j_{e+1}\dots t_{n+1})] &(4.12)\\
f_{pos}(A_{b,e}) &= \alpha(A_{b,e})\beta_{left}(A_{b,e})\beta_{right}(A_{b,e}) &(4.13)
\end{aligned}$$

where $fwd(t_0\dots t^i_{b-1})$ computes the forward probability from a special start POS symbol t_0 to the tag t^i at position $b-1$, and $bkwd(\cdot)$ is computed similarly. Boundary transition probabilities $\mathrm{P}(A|t^i_{b-1})$ and $\mathrm{P}(t^j_e|A)$ are computed from labeled data using maximum likelihood estimation and smoothing with a uniform prior. Note that by decoupling the outside estimate into left and right components, we can efficiently pre-compute the partial boundary estimates $\beta_{left}(A^k_b)$ and $\beta_{right}(A^k_e)$ for all non-terminals A^k at all word positions b and e, resulting in a constant-time lookup during parsing. To our knowledge, this is the first time this decomposition has been presented, reducing the complexity of computing $f_{pos}(A_{b,e})$ on-the-fly from $O(M^2)$ to $O(1)$, where M is number of POS symbols in the grammar. Because this function is evaluated millions of times for a typical sentence, our decomposition has great practical benefit.

4.2.5 Constituent Lexical Boundary

As it has become standard in the parsing community to parse directly from lexical items, we should not assume that gold POS tags will be given as the input to our parser. The

Constituent POS Boundary prioritization function discussed above can straight-forwardly be adapted to a lexical boundary probability model by learning constituent boundary co-occurrence statistics with lexical items instead of POS tags (for example, see the work of Hall [73]). The advantage of using lexical items directly is that they are unambiguous, thus alleviating the need to compute the non-trivial forward-backward probabilities over all possible POS sequences given the input sentence.[2] We define the Constituent Lexical Boundary model as follows, where non-terminal A spans words $w_b \dots w_e$, and the sentence $w_0 \dots w_n$ is padded by special begin and end symbols:

$$f_{lex}(A_{b,e}) = \frac{\alpha(A_{b,e})\mathrm{P}(A|w_{b-1})\mathrm{P}(w_{e+1}|A)}{\prod_{i \in b \dots e-1} P(w_i|w_{i-1} \dots w_0)} \tag{4.14}$$

The condition probability $\mathrm{P}(A|w_{b-1})$ is computed using the maximum likelihood estimate $\frac{c(A,w_{b-1})}{\sum_{v \in V} c(v,w_{b-1})}$, where $c(\cdot,\cdot)$ is the observed co-occurrence counts in the corpus (from gold labeled or maximum likelihood trees), and $\mathrm{P}(w_{e+1}|A)$ is computed likewise.

The disadvantage of computing lexical-to-non-terminal transition probabilities (opposed to POS-to-non-terminal transition probabilities) is the possibility of poor parameter estimation due to sparse data. With over 44,000 words in our vocabulary and over 1,000 non-terminals in the latent variable grammar, robust estimation of the left and right boundary scores may be problematic given the size of our training corpus. In order to more robustly estimate the boundary transition probabilities, we follow Koo et al. [107] and cluster our vocabulary using Brown clustering algorithm [21]. The Brown clustering algorithm is a bottom-up agglomerative clustering method which merges clusters that minimally increase the class bi-gram likelihood of the training data. The objective function is $\prod_{i \in 1 \dots n} \mathrm{P}(w_i|\Phi(w_i))\mathrm{P}(\Phi(w_i)|\Phi(w_{i-1}))$ where i iterates over all n words in the corpus and $\Phi(\cdot)$ is a mapping of words to clusters. The cluster-merging process from one-cluster-per-word to all-words-in-one-cluster can be efficiently represented as a binary tree where a depth of D represents the (at most) 2^D word clusters. In practice, there are often fewer instantiated clusters than possible clusters. Table 4.1 shows the correspondence between D, the number of possible clusters, and the number of observed clusters in our data.

[2] In our experiments, computing the forward-backward scores over all POS tags takes 15-30% of the total runtime, depending on the sentence length. See Figure 6.6 for details.

D	**Possible**	**Observed**
4	16	16
6	64	54
8	256	166
10	1024	401
12	4096	725
18	262144	1000

Table 4.1: Number of possible and observed lexical clusters using the Brown clustering algorithm [21] for a tree of depth D.

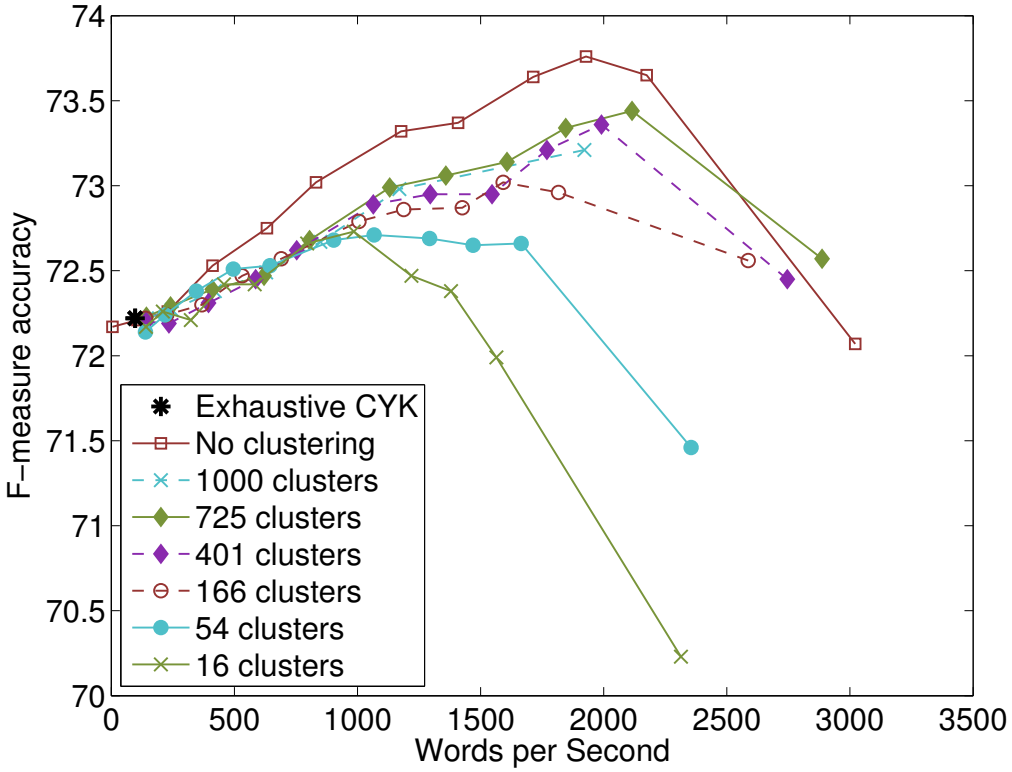
Given a mapping Φ_D which maps a lexical item w_i to a cluster c_j, we can rewrite the Constituent Lexical Boundary prioritization function as:

$$f^{\Phi}_{lex}(A_{b,e}) = \frac{\alpha(A_{b,e})\mathrm{P}(A|\Phi(w_{b-1}))\mathrm{P}(\Phi(w_{e+1})|A)}{\prod_{i\in b\ldots e-1} P(\Phi(w_i)|\Phi(w_{i-1})\ldots\Phi(w_0))} \tag{4.15}$$

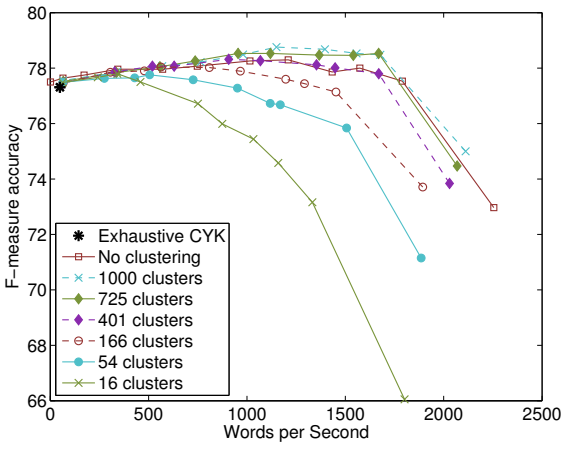
The off-line computation of lexical clusters for our domain will tie parameters of similar functioning lexical items, but making hard clustering decisions could have an adverse affect on parsing performance compared to the soft constraints of the Constituent POS Boundary if the clusters are either too coarse (providing no useful information about the boundary context) or too fine (overfitting to peculiarities of the training set). As this an empirical question, we turn to Figure 4.2, which shows beam-search parsing results with the Constituent Lexical Boundary model for multiple grammars and multiple clustering granularities (i.e., values of D). For each cluster size, the beam-search pruning parameters are varied from very loose (performance similar to CYK) to very restrictive such that performance severely degrades.

Looking again at Figure 4.2, we see that the optimal number of clusters is dependent on the grammar. For the R2 grammar (right binarized Markov order-2), the trend is straightforward – the more individual lexical items are clustered, the worse parsing performs, especially at the most restrictive beam-search settings. The R2P1[3] and latent grammars

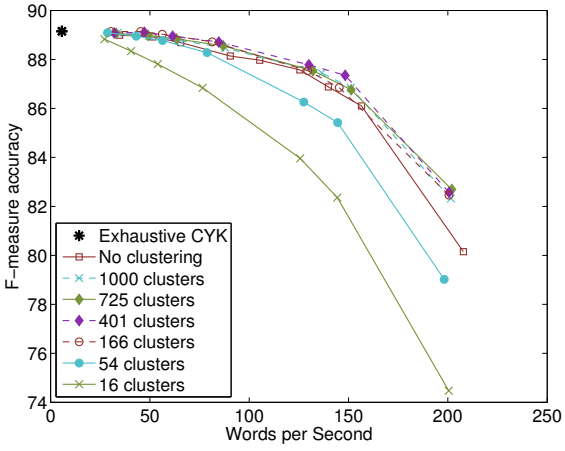
[3]R2P1 is a right binarized Markov order-2 grammar with parent annotation. See Section 3.6.1 for the complete list.



(a) R2 grammar



(b) R2P1 grammar



(c) Latent grammar

Figure 4.2: Results for Constituent Lexical Boundary prioritization function with different cluster sizes.

also see substantial degradation in performance when the number of clusters is lower than 100. This should be expected as limiting the number of possible lexical contexts restricts the power of the model to distinguish probable from improbable contexts. What makes the results different for these two grammars compared to the R2 grammar is that we see a performance gain when 725 or 1000 clusters are used, over estimating the model from lexical items directly. This confirms our hypothesis that smoothing the model by tying similar functioning lexical items is effective when the parameter space is large.

4.2.6 Prioritization Models with Unlabeled Data

Standard estimation of the prior probability for the Constituent Prior model or the boundary transition probabilities for either the POS or Lexical Boundary model is computed via maximum likelihood counts from an annotated treebank. For example, the prior probability $P(A_i)$ of observing non-terminal A_i would be the frequency of A_i observed in the treebank divided by the total observed non-terminals. For the R2 or R2P1 grammars, the Markovization and parent annotation is a deterministic grammar transform of the gold treebank, and these modified non-terminal counts can be extracted after transformation. But with a grammar such as Petrov and Klein [133] latent variable grammar, which is estimated via expectation maximization, or the "unlexicalized" grammar of Klein and Manning [102], which is non-trivial to reproduce, no such gold treebank is available with the same non-terminal annotations for training prioritization models. More generally, if we wish to train a prioritization model given any arbitrary PCFG – even a hand constructed grammar – we cannot assume to have access to a treebank annotated under the same philosophy.

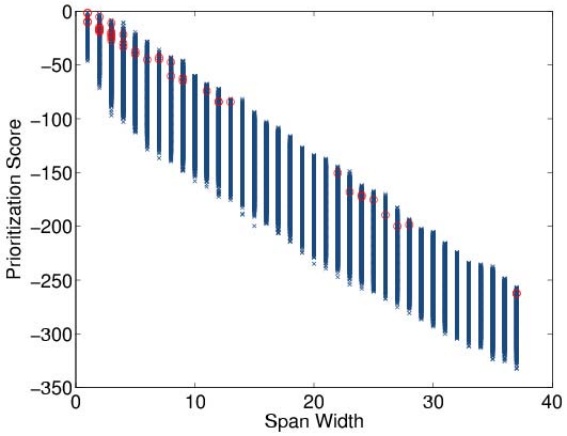
Given this scenario, we consider two options to automatically induce a labeled treebank for training prioritization models. First, we can map each non-terminal in the target grammar to corresponding non-terminals in the original treebank. This should not require significant effort as such a mapping must already exist for evaluation purposes. A Prior or Boundary prioritization model could then be estimated form the original treebank and applied to the target grammar, effectively tying the parameter space of all non-terminals mapped to a single treebank label (i.e., NP). Our second option is to parse an arbitrary

amount of text in a given domain with the target grammar and estimate the prioritization model parameters from this generated treebank. This is similar in spirit to self-training [119] – and to a lesser extent, co-training [154] – which both leverage unlabeled data to improve the underlying PCFG. In our case, we do not alter the grammar, but instead train a prioritization function to efficiently search the space created by the grammar. Under this scenario, we should not expect the prioritization model to improve parse accuracy above the maximum likelihood PCFG solution, as that is exactly what we are using to train the model.

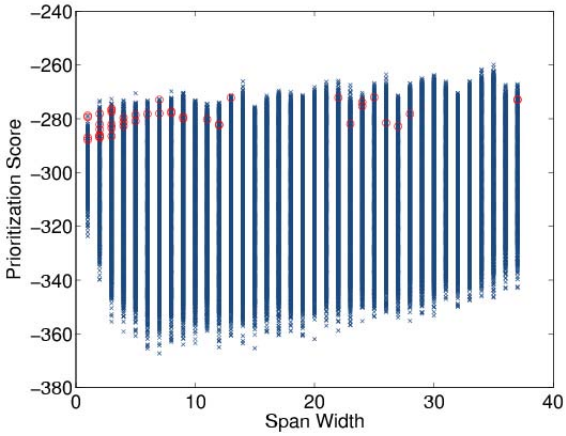
Training from unlabeled data has two advantages over mapping non-terminals to the original treebank set. First, we can estimate prioritization parameters for all non-terminals in the target grammar, opposed to the smaller set of those in the original treebank. This allows our prioritization methods to better define the distribution of probable and improbable outside contexts for each of the refined non-terminals. Second, it is well-known that a PCFG trained from the WSJ treebank often overfits the data and does not generalize well to new domains [68], and there is no reason to believe prioritization models trained on the same data will not suffer from the same problem. With the ability to train from unlabeled corpora, we can now adapt our prioritization functions to arbitrary domains such that best-first and beam-search parsing can search the solution space effectively when not parsing news-wire text. Given these advantages, we train the Constituent Prior, POS Boundary, and Lexical Boundary prioritization models from maximum likelihood trees when parsing with the latent variable grammar.

4.2.7 Analysis and Discussion

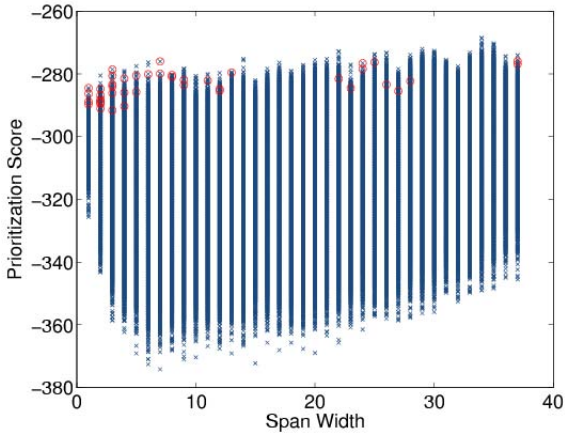
The true effectiveness of each prioritization function is how well it guides the search during parsing, and ultimately, the possible accuracy/efficiency operatizing points when parsing with the model. Before presenting such results, we can get a more comprehensive view of how each function is affecting the search by looking at the five sub-figures of Figure 4.3. These figures plots the prioritization score of every constituent while parsing a single



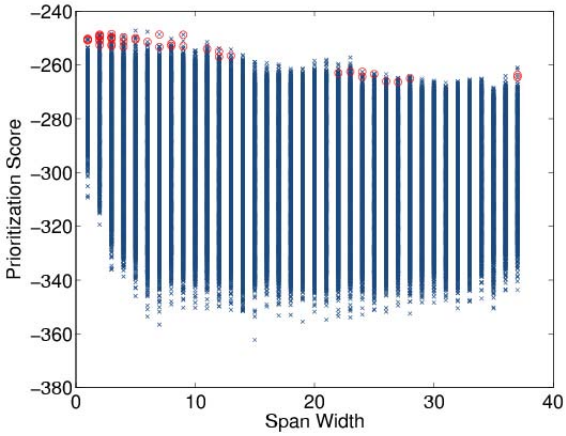
(a) Inside Prioritization



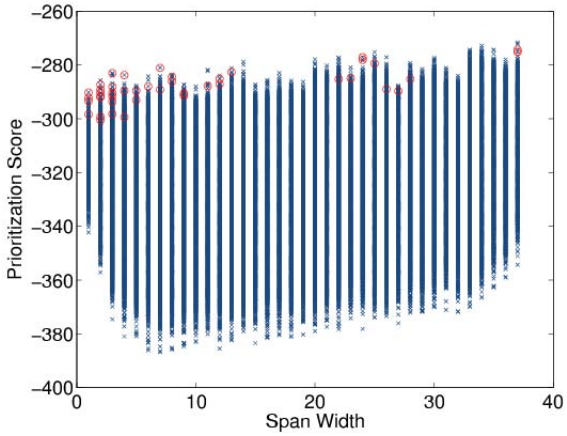
(b) N-Gram Normalized Inside



(c) Constituent Prior



(d) Constituent POS Boundary



(e) Constituent Lexical Boundary (no clustering)

Figure 4.3: Prioritization scores per span-width of all constituents during exhaustive parsing of sentence 1 in WSJ Section 22 with the R2 grammar. Red circles indicate constituent from gold parse tree. Sub-figures 4.3a and 4.3b are repeated from Figure 4.1 for comparison with the Constituent Prior and Boundary methods.

example sentence;[4] no pruning is applied. Each blue "x" is an individual constituent and the red circles are the correct constituents given by the gold tree. An idealized prioritization function would push all red circles towards positive infinity, while pushing everything else towards negative infinity.

Consider, for example, a best-first parsing architecture with a global agenda. Under this assumption, the constituent with the highest prioritization score is popped from the agenda and placed in the chart. This is analogous to moving a horizontal line from the top of each plot in Figure 4.3 to the bottom, and processing each red circle or blue "x" that crosses the line until a full parse tree has been discovered (minimally when all red circles have been popped from the agenda).[5] Given this analogy, we can see how using the inside score alone in Figure 4.3a to prioritize constituents is far from ideal. Because the inside score is cumulative (in the log domain), there is an indirect correlation between the span length and the inside score. Thus, the first constituent to span the entire input sentence has a prioritization score of nearly -250, by which time all constituents of length 1 to 20 will have already been processed, whether or not they are likely given their context (see further discussion in Section 4.2.2).

The remaining plots in Figure 4.3 show the prioritization of every constituent for four different outside heuristics: N-gram normalized inside probability, the Constituent Prior model, the Constituent POS Boundary model, and the Constituent Lexical Boundary model (no word clustering is used when parsing with the Lexical Boundary prioritization function in this example). Remember that these figures only plot the prioritization scores of a single sentence and may not be representative of the behavior over the entire corpus. Keeping this in mind, we see that the POS and Lexical Boundary prioritization functions rank the gold constituents (red circles) higher than either the normalized inside or Prior outside models. In particular, the Lexical Boundary prioritization function ranks the gold constituents spanning the entire sentence (span-width of 37) very favorably compared to all other constituents in the agenda. But the POS Boundary prioritization function will

[4]Sentence 1 in WSJ Section 22

[5]This analogy is not entirely correct since a constituent is only traversable once its children have been discovered and although a parent constituent's *inside* score is always less than its children, that is not necessarily true for a parent's *prioritization* score given an arbitrary prioritization function.

require the least number of agenda pops to find the gold tree, primarily due to its superior ranking performance on constituents spanning shorter substrings (length 1-10). We will present performance statistics over the entire development set in the next section.

The plots of Figure 4.3 are informative when performing best-first parsing and all constituents are prioritized together in a global agenda. But when a beam-search is employed, such a global comparison is not helpful as constituents are compared only against others spanning the same substring. In order to visualize the effectiveness of each prioritization function within a beam-search parser, we turn to Figures 4.4 and 4.5. Here, we plot a histogram of the rank of all gold constituents within each local agenda when performing an unpruned beam-search (i.e., a beam-width of infinity). For example, if a constituent from the gold parse tree spans substring $w_b \dots w_e$ and its rank in the local agenda for chart cell $C_{b,e}$ is four, then one instance would be observed in bin four of the plot. Chart cells containing no gold constituents contribute no information to these figures. Aggregating the results across all sentence in the development set, we can see the percentage of gold constituents which are ranked first, second, third, etc. in the local agenda. The bin labeled "10+" in each plot represent any rank 10 or greater such that the cumulative probability distribution – represented by the red dotted line – of all bins is 100 percent.

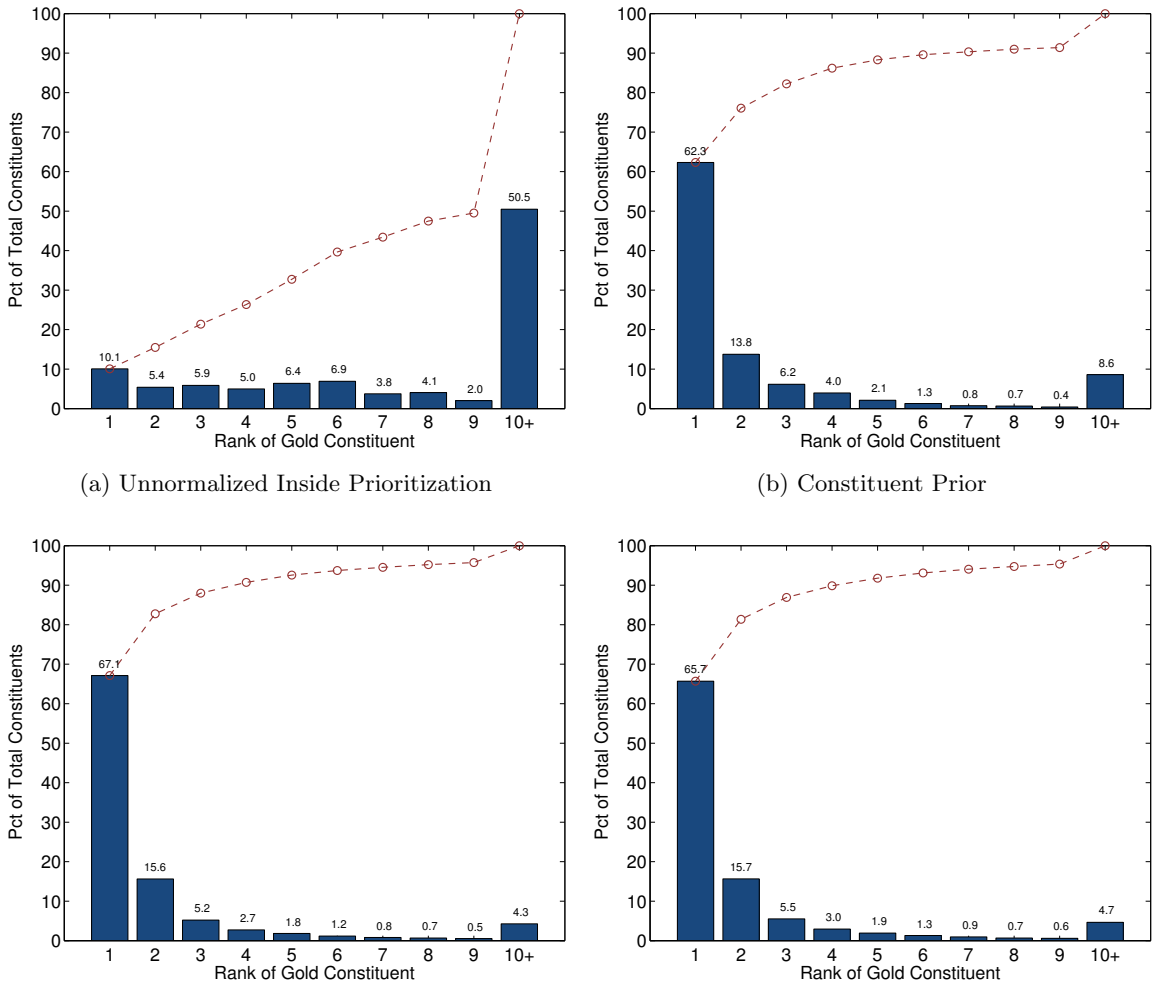
Setting a hard K-best beam-width of K=9 necessarily eliminates all constituents with a local rank greater than nine. When prioritizing by the inside constituent scores using a right-binarized Markov order-2 grammar (R2) as in Figure 4.4a, this would eliminate approximately 50.5% of gold constituents from the search, significantly degrading accuracy. The Prior, POS Boundary, and Lexical Boundary prioritization functions all fare much better, pruning approximately 8.6%, 4.3%, and 4.7% of gold constituents respectively at the K=9 threshold.[6] Although the Constituent Prior prioritization function would prune

[6]We say "approximately" because only derivable constituents are added to the local agenda. When enforcing a beam-width of K=9, this severely limits the number of derivable constituents and many scenarios exist to either move the rank of the gold constituent closer to one (high-scoring non-gold edges are no longer derivable) or eliminate the gold constituent from consideration (it is no longer derivable given the pruned search space). Note that the rank of the gold constituent will never decrease when using hard constraints compared to exhaustive beam-search, which is presented here. The true number of pruned gold constituents can only be found via decoding with the specified parameters, and we report the accuracy and efficiency tradeoff of each prioritization function in the next section.

nearly twice as many gold constituents as the either of the Boundary models at this threshold, it is surprisingly effective at ranking constituents compared to the inside score alone given that only the prior probability for each non-terminal is used for the heuristic. But as we will see next, the utility of the Constituent Prior model diminishes as the size of the grammar – and the number of non-terminals – increases.

Figure 4.5 parallels Figure 4.4 by parsing with the same four prioritization functions, but using the R2P1 grammar (right-binarized Markov order-2 parent annotation) instead of the R2 grammar. The one exception to the direct comparison is that the Constituent Lexical Boundary prioritization function does not apply word clustering with the R2 grammar, but does when used with the R2P1 grammar due to improved performance on the development set. Because the R2P1 grammar encodes additional contextual information into each phrase-level non-terminal, the grammar contains more than twice the number of productions as the R2 grammar and we do not expect the prioritization functions to be as effective given the added ambiguity. Comparing parallel results from Figure 4.4 and 4.5, we see that all four prioritization functions are less effective at correctly ranking the gold constituent in the local agenda when using the R2P1 grammar. The number of gold constituents ranked first drops from 10.1% to 4.2% when using the inside score alone; 62.3% to 39.9% with the Constituent Prior function; 67.1% to 60.0% with the POS Boundary function; and 65.7% to 60.7% with the Lexical Boundary function. The Constituent Prior model suffers the biggest loss with the increased grammar size. This is not surprising as the model is no longer relying on the prior probability difference between, say, a noun-phrase and a coordinating conjunction in the R2 grammar, but instead is predicting the non-terminal *and* non-terminal parent without any contextual information.

Also interesting is that the Lexical Boundary prioritization function outperforms the POS Boundary model at ranking the gold constituent first in the local agenda when parsing with the R2L1 grammar. It is our hypothesis that the Lexical Boundary function, with the ability to condition its outside estimate on lexical items directly, has better discriminative power than the POS Boundary model for lexical-to-non-terminal transitions that defy their POS-to-non-terminal generalization. Conversely, the Boundary POS model should perform better, on average, over all contexts given that the transition probabilities can



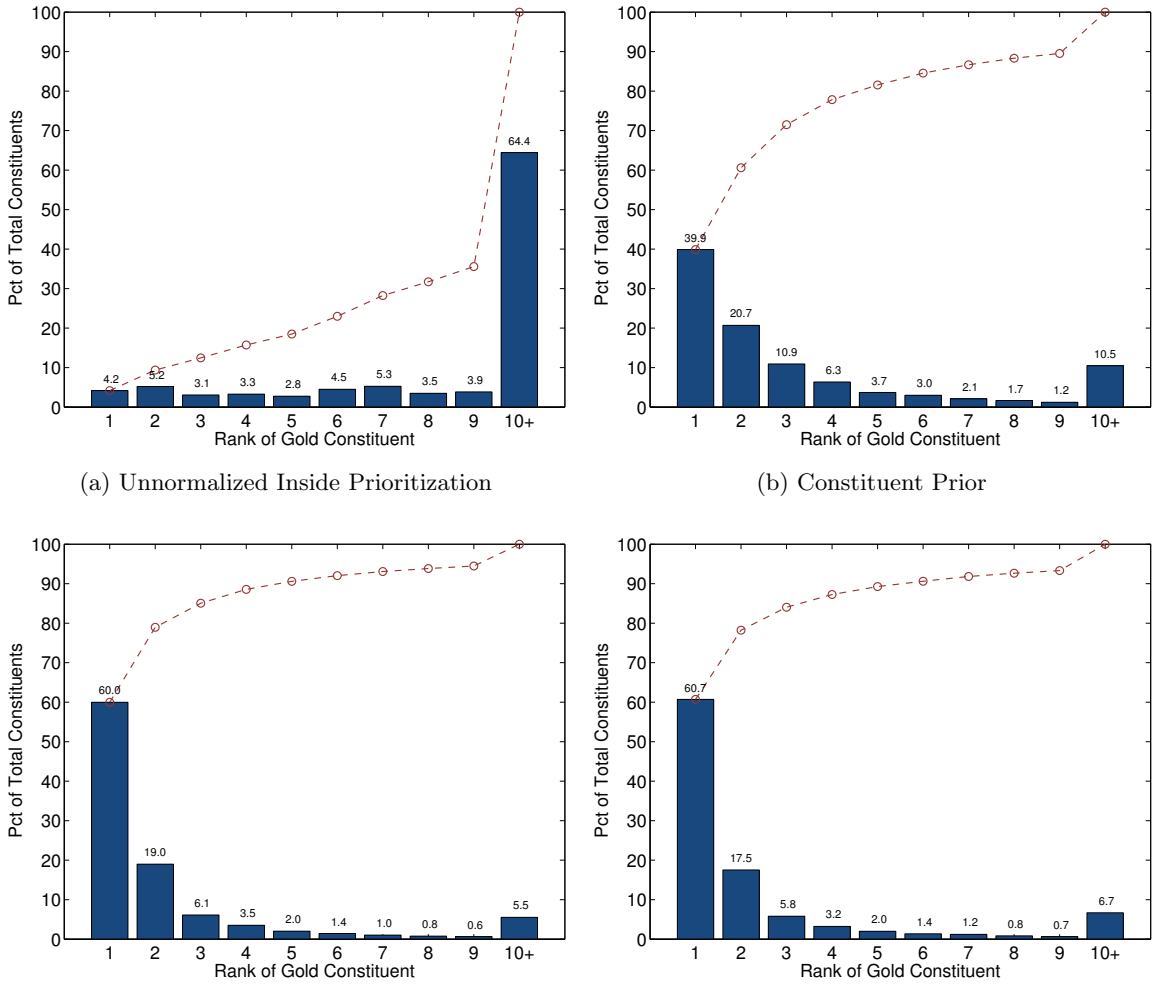
(a) Unnormalized Inside Prioritization

(b) Constituent Prior

(c) Constituent POS Boundary

(d) Constituent Lexical Boundary (no clustering)

Figure 4.4: Histogram of the rank of all gold constituent in each span-specific agenda during beam-search parsing with the **R2 grammar**. The red line is the cumulative distribution function. No pruning was applied during search (i.e., the beam-width thresholds were infinity). No clustering was applied to (d) as this leads to the best results with the R2 grammar.



(a) Unnormalized Inside Prioritization

(b) Constituent Prior

(c) Constituent POS Boundary

(d) Constituent Lexical Boundary (725 clusters)

Figure 4.5: Histogram of the rank of all gold constituent in each span-specific agenda during beam-search parsing with the **R2P1 grammar**. The red line is the cumulative distribution function. No pruning was applied during search (i.e., the beam-width thresholds were infinity)

be estimated robustly given fewer model parameters. This is confirmed in the data as we see the cumulative percentage of gold constituents with rank $K>1$ is larger for the POS Boundary model as compared to the Lexical Boundary model.

Extension of this analysis to the latent variable grammar would be interesting as the number of non-terminals and productions is significantly larger. Unfortunately, such experiments can not be performed under the same conditions as we do not have gold parse trees labeled with the appropriate latent states (see discussion in Section 4.2.6). Instead, we will compare the utility of each prioritization function by parsing unseen data and analyzing the accuracy/efficiency trade-off of each, which is the most practical objective when using syntactic structure for down-stream NLP applications.

4.3 Best-First Parsing

We begin our analysis of prioritization by evaluating each method within a best-first parsing architecture. Algorithm 2 contains pseudocode for the best-first parsing algorithm. A global priority queue, p, is initialized in line 1 by the target prioritization function, by which will sort constituents. The agenda is first populated with lexical productions $A_i \rightarrow w_j$ where A_i is a POS tag. The algorithm then continues by popping the highest scoring constituent from the agenda, adding it to the chart (α), and expanding the frontier of possible constituents given this new entry. Parsing continues until the goal node is found or the agenda is exhausted, indicating that no valid parse trees exist given the grammar. Unary productions are included in the pseudocode of Algorithm 2, which allows arbitrary length unary chains. Infinite-length chains are not a concern since α only stores the highest-scoring entry for each non-terminal A_i.

With a good prioritization function, best-first parsing with a global agenda can significantly decrease the number of constituents explored during search compared to exhaustive parsing. Fewer processed constituents, though, does not imply faster parsing speed due to the overhead of sorting all constituents by their respective prioritization score. Pushing a new frontier item onto the agenda is an $O(\log M)$ operation, where M is the size of the agenda – potentially tens of millions of entries.

Algorithm 2 BestFirstParser

Pseudocode for the best-first parsing algorithm. Unary productions are included and the Viterbi decoding is performed. Backpointer storage, memoization, and over-parsing are omitted.

Input:

$w_1 \dots w_n$: Input sentence

G: Binarized PCFG

f: Prioritization function

Output:

α: Viterbi-max scores for all non-terminals over every span

BestFirstParser$(w_1 \dots w_n,\ G = (V, T, S^\dagger, P, \rho), f)$

$p \leftarrow PriorityQueue(f)$ ▷ The agenda prioritized by f

for $b = 1$ to n **do** ▷ Add lexical productions to agenda

  for $A_i \in V$ **do**

    $p.push(\{A_i \rightarrow w_b, b, \emptyset, b+1\})$

while $p.size() > 0$ and $\alpha_{S^\dagger}(1, n) = 0$ **do**

  $\{A_i \rightarrow \beta, b, m, e\} \leftarrow p.pop()$ ▷ Remove highest-priority edge

  if $m = \emptyset$ **then** ▷ If unary production

    $\alpha_i(b, e) \leftarrow \rho(A_i \rightarrow A_j)\ \alpha_j(b, e)$

  else

    $\alpha_i(b, e) \leftarrow \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, m-1)\ \alpha_k(m, e)$

  for $A_x \in V$ **do** ▷ Possible unary expansions: $X \rightarrow A_i$

    $p.push(\{A_x \rightarrow A_i, b, \emptyset, e\})$

  $m \leftarrow b$

  for $b_2 = 1$ to m **do** ▷ Possible binary expansions: $X \rightarrow Y\ A_i$

    for $A_x \in V$ **do**

      $y \leftarrow \text{argmax}_y[\rho(A_x \rightarrow A_y\ A_i)\ \alpha_y(b_2, m)\ \alpha_i(m+1, e)]$

      $p.push(\{A_x \rightarrow A_y\ A_i, b_2, m, e\})$

  $m \leftarrow e + 1$

  for $e_2 = m$ to n **do** ▷ Possible binary expansions: $X \rightarrow A_i\ Z$

    for $A_x \in V$ **do**

      $y \leftarrow \text{argmax}_y[\rho(A_x \rightarrow A_i\ A_y)\ \alpha_i(b, m)\ \alpha_y(m+1, e_2)]$

      $p.push(\{A_x \rightarrow A_i\ A_y, b, m, e_2\})$

return α

4.3.1 Agenda Memoization

To parse efficiently, we must aim to minimize the number of agenda push and pop operations. The prioritization functions presented in the previous section are one step towards this goal, but even applying these to best-first search in a naïve fashion can be costly. For example, when a new frontier edge $(A_i \rightarrow A_j\ A_k, b, m, e)$ is discovered (but not yet added to the agenda) with an inside score α^* and prioritization score π^*, we can first verify that $\alpha^* > \alpha_i(b, e)$ in the dynamic programming chart before adding it to the global agenda, where α is the current Viterbi-max scores recorded in the dynamic programming chart (see Algorithm 2). This is a straight-forward verification to eliminate a number of unnecessary operations, but is limited to the current state of α at the time of frontier expansion. What if a constituent $(A_i \rightarrow A_x\ A_y, b, m_2, e)$ exist in the agenda with an inside score α' and prioritization score π', but has not yet been added to the chart? Because the outside estimate of each of the prioritization functions we consider is agnostic to the internal structure of the partial derivation (i.e., the constituent's child non-terminals or midpoint), then it must be true that if $\alpha^* < \alpha'$ then $\pi^* < \pi'$ and we can be assured that withholding this constituent from the agenda will have no effect on the final derivation as it will never be added to the chart.

Searching the agenda before each push is not practical as it would requires a linear traversal of the items. Instead, we memoize (record) the inside score of each pushed edge in $\mathcal{M}_i(b, e)$ for comparison with each frontier edge before pushing onto the global agenda. This optimization provides large efficiency benefits, especially when the prioritization function is poor, at the cost of potentially doubling the memory requirements (we are essentially keeping two dynamic programming charts). This can been seen in Table 4.2 where we parse the development set with the R2 grammar both with and without memoization. Results for the CYK algorithm are also presented for comparison. Note that all three algorithms, when using the Inside prioritization function, optimize the search function exactly finding the parse tree t that maximize $P(t|w)$. The F_1 score is slightly different between the methods due to tie resolution differences based on traversal order.

	Inside				POS Boundary			
	F_1	**W/S**	**Pushes**	**Pops**	**F_1**	**W/S**	**Pushes**	**Pops**
CYK	72.2	97.1	14899833		72.2	97.1	14899833	
Agenda	71.9	1.8	4296694	3974561	72.5	286.7	157996	6013
Memoization	71.7	13.5	397470	386889	72.6	335.9	40490	2248

Table 4.2: Comparison of accuracy, runtime, and constituents processed for a standard best-first agenda parser (Agenda) and an agenda parser with memoization using the R2 grammar. "W/S" is words per second; "Pushes" and "Pops" are the average number of agenda operations per sentence over the entire development set. Results from the CYK algorithm are presented for comparison, where edges added to the chart are reported (no agenda is used during CYK parsing).

We see in Table 4.2 that the naïve best-first agenda parser using the Inside prioritization function reduces the number of processed constituents by 71% compared to the exhaustive CYK algorithm, yet the time-to-parse is much worse (1.8 words per second versus 97.1 words per second). Pruning constituents using memoization provides an additional 10-fold reduction to the search space, but both methods still see a run-time slower than CYK parsing. Applying a more accurate prioritization function, such as the POS Boundary model, we see similar trends but to a lesser degree. Using memoization decreases the number of agenda pushes and pops by nearly 75%, translating into a 17% increase in words per second. For the remainder of this thesis, all best-first results assume a global agenda framework with memoization.

4.3.2 Over-parsing

Given an admissible prioritization heuristic, such as those found in A* search [126], one can guarantee that the first parse tree discovered spanning the entire sentence will be the maximum likelihood solution [149]. Each of the prioritization functions discussed in Section 4.2 are inadmissible, meaning there are no constraints on how priority is assigned and the first complete parse tree discovered may not be the optimal solution according to the PCFG. Inadmissible heuristics are often used in NLP applications such as parsing [25] and machine translation [176] in order to optimize efficiency over accuracy, but as we've

already seen in Table 4.2, an inadmissible heuristic – such as the Boundary POS model – can actually lead to more accurate solutions given a non-likelihood objective such as F_1 or labeled bracket match. This is due to the complementary nature of the prioritization function leveraging contextual information that is not directly encoded in the grammar.

In cases where our best-first search is guided by an inadmissible heuristic and we prefer to find the maximum likelihood parse tree, we can employ a technique known as over-parsing, which continues to add constituents from the agenda to the dynamic programming chart after the first solution has been found [73]. A single parameter, η, controls the degree of additional search such that if p constituents have been popped from the agenda when the first complete parse tree is discovered, we do not terminate the search until a total of $\eta \cdot p$ constituents have been popped. A value of η=1 returns the first parse tree discovered with no additional processing; a value of positive infinity will continue to parse until the agenda is empty.

Table 4.3 contains results showing the effect of over-parsing for three different Inside normalization functions discussed in Section 4.2. Looking at the performance for η=1, we see that each normalization function has a significantly different accuracy/efficiency operating points for the first full parse tree returned. As the over-parsing value η increases, we see that each method adds additional constituents to the chart and steadily approaches the maximum likelihood solution. The number of pops for, say, η=4 is not exactly four-times the value of η=1 due to the sentence-specific nature of over-parsing. If the prioritization method is particularly poor for an individual sentence and a full parse is not found until 85% of the agenda has been exhausted, doubling (or quadrupling) the target number of pops has little effect given that parsing necessarily terminates once the agenda is empty.

In Table 4.4 we apply the more complex Boundary POS model to best-first parsing with the R2 and Latent grammars. Here we see that the first parse tree returned with the prioritization function and the R2 grammar has a higher F_1 than the maximum likelihood solution obtained with exhaustive search (72.6 compared to 72.2). If we apply over-parsing and continue to add constituents to the chart, F_1 decreases while likelihood increases until the optimal solution according to the PCFG is found. In this situation, it is better to

	Product λ=6			**Geometric Mean**			**N-Gram**		
η	**F_1**	**W/S**	**Pops**	**F_1**	**W/S**	**Pops**	**F_1**	**W/S**	**Pops**
1	67.8	29.1	152757	37.4	248.7	24027	56.9	221.7	26334
2	70.5	21.0	243451	41.9	143.8	48054	63.9	117.6	52669
4	71.2	18.5	290173	46.4	78.7	95971	67.9	58.2	101682
8	71.5	17.5	312506	52.6	41.5	189761	70.5	33.5	172889
16	71.7	17.0	324333	59.1	21.8	346013	71.4	22.9	252126
32	71.7	16.8	329251	64.9	13.2	541215	71.7	17.5	320037

Table 4.3: Over-parsing with three inside probability normalization methods using the R2 grammar. "W/S" is words per second and "Pops" is the average number of agenda pops per sentence over the entire development set.

terminate the search early and rely heavily on the prioritization method more than the grammar if F_1 is our true objective.

When parsing with the latent variable grammar in Table 4.4, we see that maximizing the likelihood of the PCFG is more closely aligned with F_1 than the first parse tree returned by the prioritization function. Over-parsing by η=4 is necessary to achieve the accuracy of exhaustive search, although this only increases the number of agenda pushes by roughly 50%. Note that the number of agenda operations is not proportional to the true parsing speed (words per second); although the number of pushes is commensurate between the two grammars, over an order of magnitude difference exists between the speed of parsing. This is due to the difference in grammar size. When a constituent is popped from the agenda and placed in the chart, all possible frontier expansions are considered according to the grammar and the current chart population. But most candidates are not pushed onto the agenda due to restrictions already in the chart (a constituent with higher likelihood already exists) or is already in the agenda (see previous section on agenda memoization). The iteration over this set of possible candidates is nearly two orders of magnitude larger for the Latent grammar compared to the R2 grammar, hence the difference in parsing speed. This is a key example of why we consider runtime performance a critical metric when comparing efficient parse strategies.

	R2 + POS Boundary				Latent + POS Boundary			
η	**F_1**	**W/S**	**Pushes**	**Pops**	**F_1**	**W/S**	**Pushes**	**Pops**
1	72.6	336.7	40490	2248	87.7	15.7	52059	425
2	72.5	212.5	62232	4497	88.8	13.9	61937	851
4	72.2	138.4	90176	8994	89.2	11.4	77278	1703
8	72.1	90.9	122822	17988	89.2	8.6	101262	3406
16	72.1	63.3	158002	33798	89.2	6.1	138030	6812

Table 4.4: Over-parsing with the POS Boundary prioritization methods using the R2 and Latent grammars. "W/S" is words per second; "Pushes" and "Pops" are the average number of agenda operations per sentence over the entire development set.

4.4 Beam-Search Parsing

Beam-search is a standard graph-based search algorithm that makes hard pruning decisions at each span. In a bottom-up breadth-first traversal, all candidate derivations are sorted by a prioritization function and only the most promising candidates are retained. The size of this set is determined by two tuning parameters, K and R, which we discuss later in this section.

In Algorithm 3 we present pseudocode of a beam-search parser leveraging both pruning parmeters. We see that the algorithm is very similar to exhaustive CYK parsing, with the exception of keeping a local priority queue for each span (line 4). Each constituent with the highest derivational score for each non-terminal A_i is added to the local priority queue (lines 5-10) but only those edges satisfying the beam-search criteria are added to the chart (i.e., stored in α). We also include pseudocode to process unary edges in lines 19-20. Unlike the implementation of unary processing for the CYK algorithm (see Alg. 1), this beam-search implementation can construct long unary chains within a single chart cell. This is because unary edges are pushed to the same local agenda as binary edges, and forced to compete under the same beam-search constraints. In theory, after a single binary edge is added to the chart, a chain of $K-1$ unary edges may determine the optimal derivation for the span, where K is the maximum number of constituents permitted in the chart cell according to the pruning parameters.

Algorithm 3 BeamSearchParser

Pseudocode for the beam-search parsing algorithm. Unary productions are included and Viterbi decoding is preformed. Backpointer storage, failure recovery, and online pruning has been omitted.

Input:

$w_1 \dots w_n$: Input sentence

G: Binarized PCFG

f: Prioritization function

K: Beam-width of at most K derivations per chart cell; $K > 0$

R: Maximum score differential from best local derivation; $0 < R \leq 1$

Output:

α: Viterbi-max scores for all non-terminals over every span

BeamSearchParser$(w_1 \dots w_n,\ G = (V, T, S^{\dagger}, P, \rho),\ f,\ K,\ R)$

for $s = 1$ to n **do** ▷ Span width
  for $b = 1$ to $n - s + 1$ **do** ▷ Begin word position
    $e \leftarrow b+s-1$ ▷ End word position
    $p \leftarrow PriorityQueue(f)$ ▷ Local agenda prioritized by f
    for $A_i \in V$ **do**
      if $s = 1$ **then** ▷ Special case for lexical productions
        $p.push(\{A_i \rightarrow w_b, \emptyset\})$
      else
        $j, k, m \leftarrow \underset{b \leq m < e}{\operatorname{argmax}} \left(\underset{j,k}{\operatorname{argmax}}\ \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, m)\ \alpha_k(m+1, e) \right)$
        $p.push(\{A_i \rightarrow A_j\ A_k, m\})$
    $\{A_i \rightarrow \beta, m\} \leftarrow p.pop()$
    $\pi \leftarrow f(A_i, b, e)$
    $R_{b,e} \leftarrow \pi * R$ ▷ Lower bound on priority score
    while $p.totalPops() < K$ and $\pi > R_{b,e}$ **do**
      if $m = \emptyset$ **then**
        $\alpha_i(b, e) \leftarrow \rho(A_i \rightarrow A_j)\ \alpha_j(b, e)$
      else
        $\alpha_i(b, e) \leftarrow \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, m)\ \alpha_k(m+1, e)$
      for $A_x \in V$ **do** ▷ Add unary edges
        $p.push(\{A_x \rightarrow A_i, \emptyset\})$
      $\{A_i \rightarrow \beta, m\} \leftarrow p.pop()$
      $\pi \leftarrow f(A_i, b, e)$
return α

There are two key points we wish to highlight when comparing the best-first and beam-search algorithms. First, both algorithms sort edges in a priority queue within the inner loop of the algorithm, increasing the worst-case complexity of both algorithms from $O(N^3)$ to $O(N^3 \log N)$. This increases the parsing complexity over the CYK algorithm, but in practice often decreases parsing time due to pruning or prioritization of the search space. Within a best-first architecture, the global agenda potentially maintains a sorted list of all $O(N^3|G|)$ edges, while beam-search parsing sorts at most $O(N|G|)$ constituents, where $|G|$ is the size of the grammar.

Second, because the agenda in a beam-search parser only compares constituents spanning the same substring, the normalization component of each of the prioritization functions is unnecessary. Furthermore, the difficulties of comparing constituents spanning different substrings, as is done in the global agenda of a best-first parser, is eliminated. As we will see in the next section, this allows us to maximize the benefit of each prioritization function and tighten the pruning parameters quite severely before accuracy degrades.

4.4.1 Relative and K-Best Beam-Width

The number of derivations retained in each cell during beam-search parsing has significant impact on parsing performance, as cells spanning larger substrings need only consider those entries that remain. Prior work has often focused on tuning a single beam-width parameter — a constant number of entries per span [9, 128], or a constant probability threshold from the highest-scoring local candidate [46, 143] — to optimize efficiency and accuracy. We refer to these methods as *beam-K* and *beam-R* respectively. Beam-K is guaranteed a fixed amount of work no matter the distribution of local constituents; even if one constituent has far superior merit, that K highest ranking entries will be retained. Beam-R only retains local candidates within a fixed-width probability window, where the number of qualifying candidates may be as small as one, or as large as hundreds.

In order to compare the individual merit of each beam-search pruning method, we run empirical trials with multiple grammars and prioritization functions. In Figure 4.6 we see an individual subplot for each of the R2, R2P1, and Latent grammars with both the Inside and POS Boundary prioritization functions. The values of K range from 2 to 100; the

values of R range from 4 to 10 (in the log domain). The first column of subplots — Figures (a), (c), and (e) — parse with the Inside prioritization function. Here we see that for all grammars, accuracy quickly degrades with both the beam-K and beam-R methods. In fact, no pruning configuration can both speed up parsing and maintain accuracy compared to the exhaustive chart parsing, except the largest values of beam-K with the R2 grammar.

The second column of subplots applies the POS Boundary prioritization function to parsing with each grammar, resulting in much more favorable accuracy/efficieny curves. We see that both the beam-K and beam-R methods can independently increase the words-per-seconds parsed by an order of magnitude over exhaustive parsing without compromising accuracy. And as we saw with best-first search, F_1 accuracy increases with the R2 grammar over the maximum likelihood solution given the grammar. For both Inside and Boundary POS ranking functions, the beam-R method degrades more gracefully than beam-K. This should be expected as beam-R pruning takes the distribution of local candidates into account, and will retain a potentially large set of competitors even if the pruning parameters are overly aggressive. But if F_1 accuracy cannot be compromised, then both pruning methods are equally effective at pruning the search space under this constraint.

Unfortunately, the accuracy versus efficiency plots of Figure 4.6 do not tell the entire story when using the Inside prioritization function. Because it is standard to ignore failed parses when computing accuracy over a test set (and the default behavior of EVALB [1]), F_1 accuracy alone does not give a sense of parse failure rate with these methods. In Table 4.5, we report the tuning value and number of failed parses along with the F_1 and words-per-second and from Figure 4.6. The tuning parameter for beam-K is simply the number of local-agenda pops per chart cell. The tuning parameter for beam-R is the score difference between constituents in the log domain. With this new information, it is clear that a constant tuning parameter for either beam-K or beam-R is insufficient to find valid parse trees effectively when prioritized by the inside probability. For example, a beam-K value of K=50 with the R2 grammar searches the space well for many sentences, achieving an F_1 score of 72.5, but even with at this (relatively generous) pruning threshold, 27% of the input sentences (464 of 1700) fail to find a single valid parse. With the Boundary

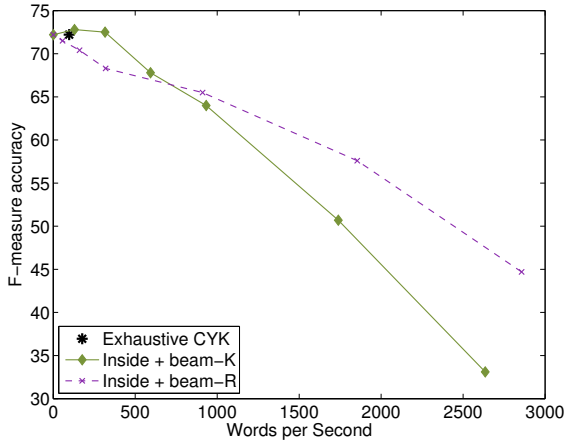
	R2			R2P1			Latent		
Beam-K	**F_1**	**W/S**	**Fail**	**F_1**	**W/S**	**Fail**	**F_1**	**W/S**	**Fail**
5	33.1	2,635.1	1445	15.8	2,244.2	1498	72.2	164.8	727
10	50.7	1,738.8	1160	26.8	1,369.9	718	79.8	108.0	285
20	64.0	932.9	968	43.1	651.6	361	84.1	61.7	104
30	67.8	593.4	716	55.7	367.5	123	85.8	41.7	52
50	72.5	316.1	464	66.4	181.1	48	87.3	23.6	23
100	72.8	129.5	157	74.1	67.2	23	88.6	10.3	11
Beam-R									
4	44.7	2,857.1	1418	42.3	1,403.1	1469	85.8	133.3	293
5	57.6	1,854.2	1115	55.3	689.8	1137	87.1	79.2	142
6	65.5	991.1	678	66.2	252.5	643	88.1	45.0	50
7	68.3	319.6	262	71.7	100.0	278	88.6	24.7	25
8	70.4	160.5	122	74.6	48.2	107	88.9	10.7	7
10	71.5	56.9	36	77.1	17.1	29	89.0	4.3	2

Table 4.5: Beam-K and beam-R pruning with Inside prioritization. The beam-K tuning parameter is the number of local-agenda pops per chart cell; the beam-R tuning parameter is the prioritization score difference in the log domain.

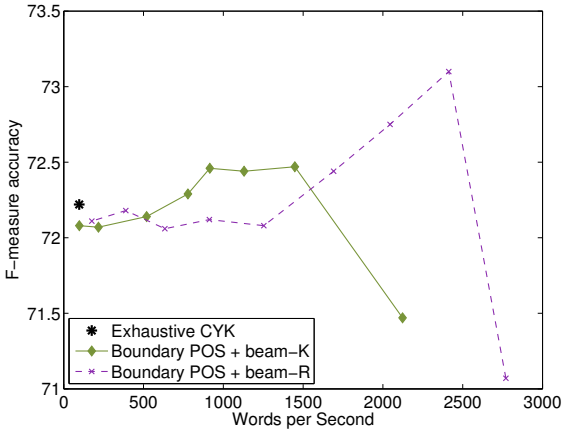
POS prioritization function, every sentence in the development set receives a parse with all three grammars, and therefore the graphs of Figure 4.6 are an accurate portrayal of parsing performance.

4.4.2 Online Pruning and Bounded Priority Queues

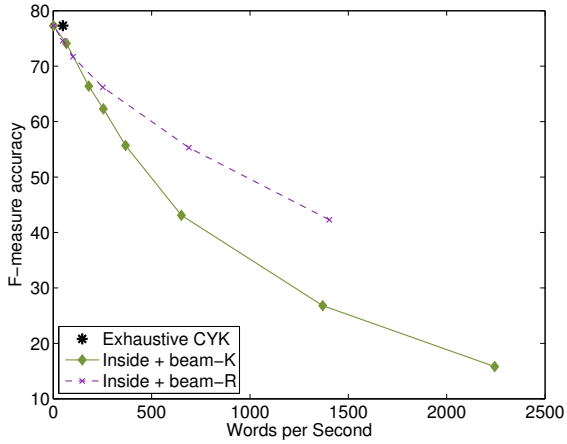
We add two optimizations to beam-search parsing in Algorithm 3. The key motivation stems from line 10, where the highest scoring backpointer for each non-terminal A_i is pushed onto the local agenda with its prioritization score. Given fixed beam-K and beam-R constraints, we can prune many of the entries before adding them to the local agenda, saving the $O(\log N)$ operation. The first method we call *online pruning*, the second leverages a *bounded priority queue*. Neither of these optimizations alter the complexity of the algorithm, but both methods decrease parsing time in practice.



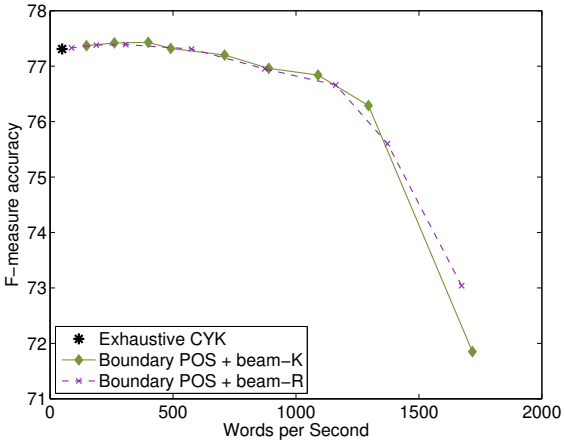
(a) R2 grammar, Inside priority



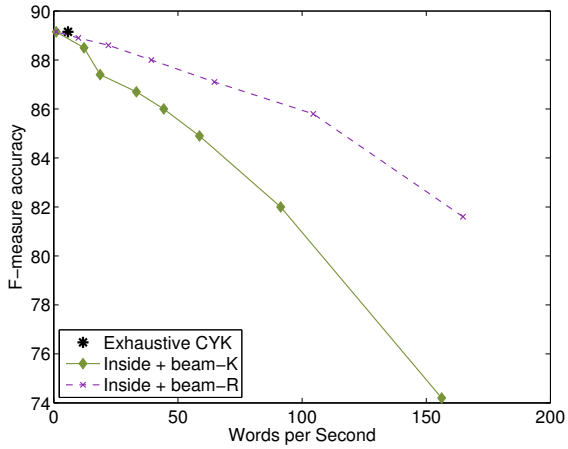
(b) R2 grammar, Boundary POS priority



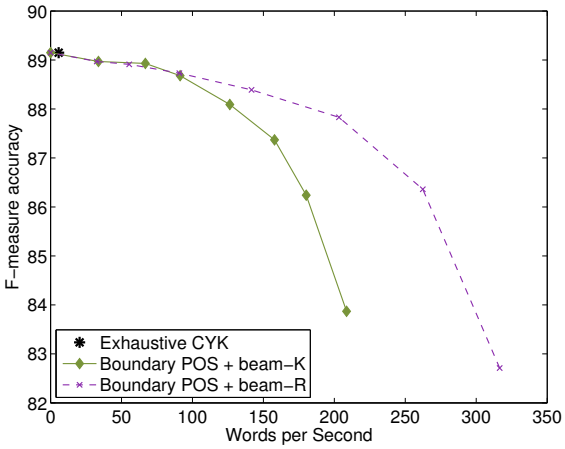
(c) R2P1 grammar, Inside priority



(d) R2P1 grammar, Boundary POS priority



(e) Latent grammar, Inside priority



(f) Latent grammar, Boundary POS priority

Figure 4.6: Adjusting the k-best beam-width (beam-K) and relative beam-width (beam-R) independently using two prioritization functions with three different grammars.

In line 9 of Algorithm 3 we find the highest scoring derivation rooted at A_i for all possible child non-terminals and midpoints. This maximization is necessary to ensure we find the optimal derivation, but the highest scoring derivation $\{A_i \rightarrow A_j\ A_k, m\}$ should only be added to the local agenda if its prioritization score satisfies the beam-K and beam-R constraints, namely:

$$f(A_i \rightarrow A_j\ A_k, b, m, e) \geq R \cdot \max_{x,y,z,m_2} f(A_x \rightarrow A_y\ A_z, b, m_2, e) \tag{4.16}$$

Unfortunately, we do not know the maximum prioritization value over all non-terminals A_i until each has been processed. What is known at time i is the maximum prioritization score for all non-terminals $A_0 \ldots A_{i-1}$, which can provide a monotonically increasing bound on acceptable prioritization scores. This bound will never overestimate the true maximum value of $f(\cdot)$ for all A_i and we can safely eliminate unnecessary candidates that fall below the bound before adding them to the priority queue.

The second optimization we employ is replacing the priority queue in Algorithm 3 with a bounded priority queue, a data structure with an upper bound on the number of queue elements. Given the beam-K constraint, we know that at most K constituents will be added to the chart, and we can therefore discard all constituents with a priority score below the K^{th}-ranked element. As with our online pruning criteria for beam-R, we do not know the highest scoring K constituents *a priori*. Instead, our implementation of the bounded priority queue continues to add entries into the agenda until K elements exist. When subsequent elements are pushed, its priority is compared to the lowest scoring entry in the agenda. If its priority score is lower, then the entry is discarded. If it is higher, then the element is sorted, the lowest scoring entry is discarded, and a pointer to the new lowest scoring entry is reassigned to the correct queue element. As discussed in [26], we also find that using the bubble-sort algorithm [103] to order the elements is superior to a tree-based bounded priority queue or a min-max heap [4] due to the small value of K in our experiments.

Unless otherwise stated, all beam-search results in this thesis use both the online pruning and bounded priority queue optimizations.

4.4.3 Failure Recovery Strategy

Unlike best-first parsing, beam-search makes hard pruning decisions at a local level that can not be reversed. If a sentence requires an atypical amount of the search space to be explored before returning a valid parse tree, the best-first parser will continue to pop constituents from the agenda until it is found. Within a beam-search framework, it is possible for no valid solution to be found spanning the entire sentence, depending on the beam pruning parameters.

Instead of returning a parse failure during beam-search parsing, we choose to automatically loosen the pruning parameters and re-parse the input string. This allows the beam-K and beam-R values to be set effectively for the majority of the development corpus, yet if a small number of sentence fail to parse with the initial settings, they will be re-parsed under a more lenient pruning paradigm, without forcing the remainder of the corpus to follow suit. For all our experiments, when a parse failure is returned, we double the beam-K and beam-R values and re-parse. If a failure still occurs, we again double the parameters and re-parse. This continues until a valid parse tree is found, or a tunable number of re-parses have been attempted (in case no valid parse tree can be found given the grammar).

Table 4.6 reports parsing statistics on the development set for re-parsing failure recovery. We adjust the beam-K parameter with the R2 grammar and Inside prioritization function, which we showed earlier in this section to have a very high failure rate. After re-parsing, we see that all sentences return a valid parse tree (no failures), and as expected, the average number of required parses per sentence decreases as the initial beam-K value increases. We also observe that the first parse tree found is not necessarily accurate under the F_1 objective, yet even a partially correct syntactic analysis can be beneficial, depending on the downstream NLP application.

The reported "words per second" parse time in Table 4.6 accumulates the total time from all failed and successful parse attempts. This correctly give an accurate description of the true time-to parse under a re-parsing framework. Interestingly enough, the words-per-second result is roughly constant when beam-K ranges from 5 to 20, and only moderately

	No recovery			Re-parsing			
beam-K	**F_1**	**W/S**	**Fail**	**F_1**	**W/S**	**Fail**	**Parses/Sent**
5	33.1	2,635.1	1,445	58.1	116.6	0	3.14
10	50.7	1,738.8	1,160	62.2	110.6	0	2.46
20	64.0	932.9	968	67.2	100.5	0	1.98
50	72.5	316.1	464	70.7	87.7	0	1.40
100	72.8	129.5	157	71.7	74.5	0	1.13
200	72.6	57.6	58	71.9	48.7	0	1.04

Table 4.6: Parse recovery with failed beam-search parsing, using the R2 grammar and Inside prioritization function on the development set of 1700 sentences. "Parses/Sent" is the average number of times each sentence was parsed due to parse failures.

increases for larger pruning parameters. With an average number of parse attempts at 3.14 for K=5, this means the average sentence is parsed once at K=5, again at K=10, and a third time at K=20. In the end, initializing K to 20 provides similar efficiency results but with higher F_1 due to the less restrictive initial pruning parameters.

4.5 Prioritization Results

We bring together all of the methods we have discussed in this chapter as compare best-first and beam-search parsing frameworks across multiple prioritization functions in Table 4.7. Tuning parameters have been optimized for each test such that efficiency was maximized without significant loss in F_1, compared to the maximum likelihood solution. Accuracy is reported using labeled precision (LP), labeled recall (LR), and F_1; efficiency is reported in words per second (W/S). We also include results using the smaller R2 grammar as well as the high-accuracy latent variable grammar to see the overall effects of grammar size.

As one might expect, best-first search with the Inside prioritization function is much slower than even exhaustive CYK parsing, due to the overhead of sorting constituents in the global agenda. Even a crude estimate of the outside score using the prior probability of each non-terminal is not sufficient to out-perform exhaustive search when best-first parsing. With the Lexical and POS Boundary prioritization functions, though, we do

Alg	Grm	Priority	Param	LP	LR	F_1	W/S
CYK	R2	-	-	74.8	69.6	72.1	82.5
BF	R2	Inside	-	74.4	69.2	71.7	13.5
BF	R2	Prior	OP=4	74.4	69.3	71.7	56.8
BF	R2	POS Boundary	OP=1	75.1	70.2	72.6	336.7
BF	R2	Lex Boundary	OP=1 Clust=None	74.3	69.2	71.6	522.3
Beam	R2	Inside	K=100 R=10	74.3	69.2	71.7	74.5
Beam	R2	Prior	K=15 R=7	75.1	69.9	72.4	589.3
Beam	R2	POS Boundary	K=5 R=2	74.8	70.3	72.5	1,448.2
Beam	R2	Lex Boundary	K=5 R=3 Clust=None	75.8	70.8	73.2	1,921.4
CYK	Latent	-	-	89.1	89.2	89.1	5.7
BF	Latent	Inside	-	89.1	89.2	89.1	0.3
BF	Latent	Prior	OP=8	89.2	89.1	89.1	1.9
BF	Latent	POS Boundary	OP=4	89.2	89.2	89.2	11.4
BF	Latent	Lex Boundary	OP=2 Clust=8	89.3	89.0	89.1	13.6
Beam	Latent	Inside	K=100 R=12	88.6	88.7	88.6	10.3
Beam	Latent	Prior	K=50 R=8	89.0	88.9	88.9	15.9
Beam	Latent	POS Boundary	K=30 R=8	89.0	88.9	89.0	33.8
Beam	Latent	Lex Boundary	K=20 R=7 Clust=8	89.2	89.1	89.1	45.4

Table 4.7: All prioritization functions are normalized with the n-gram outside score for best-first parsing. “OP” is over parsing; “K” and “R” are beam-K and beam-R tuning parameters; “Clust” is the tree depth using Brown clustering for the Constituent Lexical Boundary prioritization function.

see a significant increase in parsing speed without accuracy loss for both the R2 and latent variable grammars. Therefore, we find that best-first parsing only becomes a viable alternative to exhaustive CYK parsing given a sufficiently accurate prioritization function. This is in contrast to beam-search parsing, which sees roughly similar performance to CKY parsing when using the inside prioritization function, and superior performance with the Prior model.

We also see that the accuracy of the prioritization function has a direct affect on the required tuning parameters. With best-first parsing, the more accurate the prioritization function, the less over-parsing is required to achieve target F_1 accuracy. Likewise with beam-search, the more accurate the prioritization function, the more restrictive we can be with the pruning parameters, decreasing K to as little as five-constituents per cell for the R2 grammar.

Comparing parallel results for best-first and beam-search given each prioritization function, it is always the case the beam-search is more efficient, and nearly always the case that it is more accurate. Although parsing complexity remains the same, the general trend of a three-fold increase in words-per-second has real-world implications when parsing web-scale corpora or applying constituent parsing to real-time applications.

4.6 Conclusion

We have investigated the effects of multiple prioritization functions for context-free chart parsing, and analyzed the empirical merit of each within a best-first and beam-search framework. We have also introduced a novel prioritization function, the Constituent Lexical Boundary model, which has been shown to effectively guide the search though large model spaces, and with impoverished grammars, to even improve F_1 accuracy. The Constituent Boundary POS model of Caraballo and Charniak [25] has been adapted to the typical-case scenario of non-gold POS tags, such that prioritization look-up scores are $O(1)$, opposed to $O(N)$, where N is the length of the string.

We have also shown that under identical operating conditions, normalizing for computer hardware, implementation language, grammar and prioritization models, we see

that local prioritization with a beam-search parser outperforms best-first parsing with a global agenda in nearly all trials. This is due to two key factors. First, when parsing with large PCFGs, the size of the global agenda in a best-first parser grows so large, that the benefit it provides is marginalized, especially with poor prioritization functions. Beam-search, on the other hand, only maintains a local agenda of sorted entries and so long as the hard pruning decisions are sufficient for the domain, leads to more accurate and efficient search results. Second, as discussed by [12], normalization of the prioritization function for constituents spanning differing substrings is problematic within a global agenda. The same problem is present within decoding systems for machine translation [67]. The prioritization methods we have presented in this chapter, as well as those by Caraballo and Charniak [25], each normalize the prioritization scores using heuristic metrics and, as shown by empirical trials, require dampening the effect of normalization or sufficiently over-parsing in order to maintain baseline accuracy levels. Beam-search, on the other hand, alleviates this global normalization problem completely by only performing local comparisons, leading to superior parsing performance.

When the parameter space of the prioritization model is sparse, such as with the latent variable grammar, we have shown that the Lexical Boundary transition probabilities can be estimated more robustly with a bottom-up clustering algorithm by tying parameter of multiple lexical items. In this way, the prioritization function can be adapted to any PCFG regardless of its size. In addition, when a supervised training corpus does not exist labeled with trees from the grammar, we have shown that using maximum likelihood trees to train the prioritization function is still effective at guiding the search in the target grammar space.

Most importantly, we have the first empirical evidence to corroborate our thesis that we can train models to prioritize and prune the solution space effectively given an arbitrary PCFG by leveraging information from the unambiguous input sentence. Beam-search parsing with the Constituent Lexical Boundary model is nearly 8x the speed of CYK parsing with the latent variable grammar, and over 20x the speed with the R2 grammar, each without any loss in F_1 accuracy.

Chapter 5

Pruning with Finite-State Chart Constraints

5.1 Introduction

Finite-state pre-processing for context-free parsing is very common as a means of reducing the amount of search required during full inference. The Ratnaparkhi pipeline [141] used a finite-state POS-tagger and a finite-state NP-chunker to reduce the search space at the parsing stage, achieving linear observed-time performance. Other recent examples of the utility of finite-state constraints for parsing pipelines include Glaysher and Moldovan [69], Djordjevic et al. [55], and Hollingshead and Roark [82]. Note that by making use of pre-processing constraints, such approaches are no longer performing full exact inference — these are approximate inference methods, as are the methods presented in this chapter.

Using finite-state chunkers early in a syntactic parsing pipeline has shown both an efficiency [69] and an accuracy [82] benefit for parsing systems. Glaysher and Moldovan [69] demonstrated an efficiency gain by explicitly disallowing constituents that cross chunk boundaries. Hollingshead and Roark [82] demonstrated that high precision constraints on early stages of the Charniak and Johnson [34] pipeline achieved significant accuracy improvements, by moving the pipeline search away from unlikely areas of the search space. All of these approaches achieve improvements by ruling out parts of the search space, and the gain can either be realized in efficiency (same accuracy, less time) and/or accuracy (same time, greater accuracy).

Rather than extracting constraints from taggers or chunkers built for different purposes, in this chapter we train prediction models to more directly reduce the number of entries stored in cells of a dynamic programming chart during parsing — even to the point of "closing" chart cells to all entries. We demonstrate results using three finite-state taggers that assign each word position in the sequence with a binary class label. The first tagger decides if the word can *begin* a constituent of span greater than one word; the second tagger decides if the word can *end* a constituent of span greater than one word; and the third tagger decides if a chart cell spanning a single word should contain phrase-level non-terminals, or only part-of-speech tags. Following the prediction of each word, chart cells spanning multiple words can be completely closed as follows: Given a chart cell (b, e) spanning words $w_b \dots w_e$, we can "close" cell (b, e) if the first tagger decides that w_b cannot be the first word of a multi-word constituent or if the second tagger decides that w_e cannot be the last word in a multi-word constituent. Roark and Hollingshead [145] have shown in related work that completely closing a sufficient number of chart cells allows us to impose worst-case complexity bounds on the overall pipeline, a bound that none of the other above-mentioned methods for finite-state preprocessing can guarantee.

To complement closing multi-word constituent chart cells, our third tagger restricts the population of span-1 chart cells. We note that all span-1 chart cells must contain at least one part-of-speech (POS) tag and can therefore never be closed completely. Instead, our tagger restricts unary productions with POS tags on their right-hand side that span a single word. We term these single word constituents. Disallowing such constituents alters span-1 cell population from potentially containing all non-terminals to just POS non-terminals. In practice, this decreases the number of entries in span-1 chart cells by 70% during exhaustive parsing, significantly reducing the number of allowable constituents in larger spans [18]. Because span-1 chart cells are the most frequently queried cells in the CKY algorithm, minimizing the number of entries therein will lead directly to more efficient context-free parsing.

The pre-processing framework we have outlined above is straightforward to incorporate into most existing context-free constituent parsers, a task we have already done for several state-of-the art parsers. In the following sections we formally define our approach

to finite-state chart constraints and analyze the accuracy of each of the three taggers and their impact on parsing efficiency and accuracy when used to prune the search space of a constituent parser. We apply our methods to exhaustive CYK parsing with simple grammars, as well as to high-accuracy parsing approaches such as the Charniak and Johnson [34] parsing pipeline and the coarse-to-fine parser of Petrov and Klein [132, 133]. Various methods for applying finite-state chart constraints are investigated, including methods that guarantee quadratic or linear complexity of the context-free parser.

This chapter is an extension of previous work by Roark and Hollingshead [145, 146], with additional original work for constraining unary productions in span-1 chart cells, complementing finite-state constraints for cells spanning multiple words [18]. Some of the results in this chapter have been previous published in [144]. In addition to constraining unary productions in span-1 chart cells, the original contributions of this chapter include (1) the first analysis of chart constraints on left- and right-binarized grammars; (2) the introduction of *global* high-precision pruning with demonstrable accuracy gains over previous work; (3) combination of chart constraints with beam-search and coarse-to-fine pruning, which provides additive efficiency gains; and (4) the first application of chart constraints to Chinese constituent parsing, which, to our knowledge, achieves the highest labeled F_1 score on the Chinese Penn Treebank [180] to date.

5.2 Finite-State Chart Constraints

In this section, we will explicitly define our chart constraints, and present methods for using the constraints within a context-free parser. We begin with constraints on beginning or ending multi-word constituents, then move to constraining span-1 chart cells.

5.2.1 Constituent Begin and End Constraints

Our task is to learn which words (in the appropriate context) can begin (B) or end (E) multi-word constituents. We will treat this as a pre-processing step to parsing and use these constraints to either completely or partially close chart cells during execution of the CYK algorithm.

First, let us introduce notation. Given a set of labeled pairs (S, T) where S is a string of n words $w_1 \dots w_n$ and T is the target constituent parse tree for S, we say that word $w_b \in B$ if there is a constituent spanning $w_b \dots w_e$ for some $e > b$ and $w_b \in \overline{B}$ otherwise. Similarly, we say that word $w_e \in E$ if there is a constituent spanning $w_b \dots w_e$ for some $b < e$ and $w_e \in \overline{E}$ otherwise. Recovery of these labels will be treated as two separate binary tagging tasks ($B/\overline{B}$ and $E/\overline{E}$).

While it may be obvious that we can rule out multi-word constituents with particular begin and end positions, there may be *incomplete* structures within the parser that should not be ruled out by these same constraints. Hence the notion of "closing" a chart cell is slightly more complicated than it may initially seem (which accounts for our use of quotes around the term). Consider the chart representation in Figure 5.1 with the following constraints, where $\overline{B}$ is the set of words disallowed from beginning a multi-word constituent and $\overline{E}$ is the set of words disallowed from ending a multi-word constituent[1]

$\overline{B}$: {"*usual*", ",", "*real-estate*", "*market*", "*overreacted*"}
$\overline{E}$: {",", "*the*", "*real-estate*", "*had*"}

Given the constraints above, suppose that w_b is in class $\overline{B}$ and w_e is in class $\overline{E}$, for $b < e$. We can "close" all cells (b, i) such that $i > b$ and all cells (j, e) such that $j < e$, based on the fact that multi-word constituents cannot begin with word w_b and cannot end with w_e. In Figure 5.1 this is depicted by the black and gray diagonals through the chart, "closing" those chart cells.

If a chart cell (b, e) has been "closed" due to begin or end constraints then it is clear that complete edges should not be permitted in the cell since these represent precisely the multi-word constituents that are being ruled out. But what about incomplete edges that are introduced through grammar binarization or dotted rule parsing? To the extent that an incomplete edge can be extended to a valid complete edge, it must be allowed. There are two cases where this is possible. If $w_b \in \overline{B}$, then under the assumption that incomplete

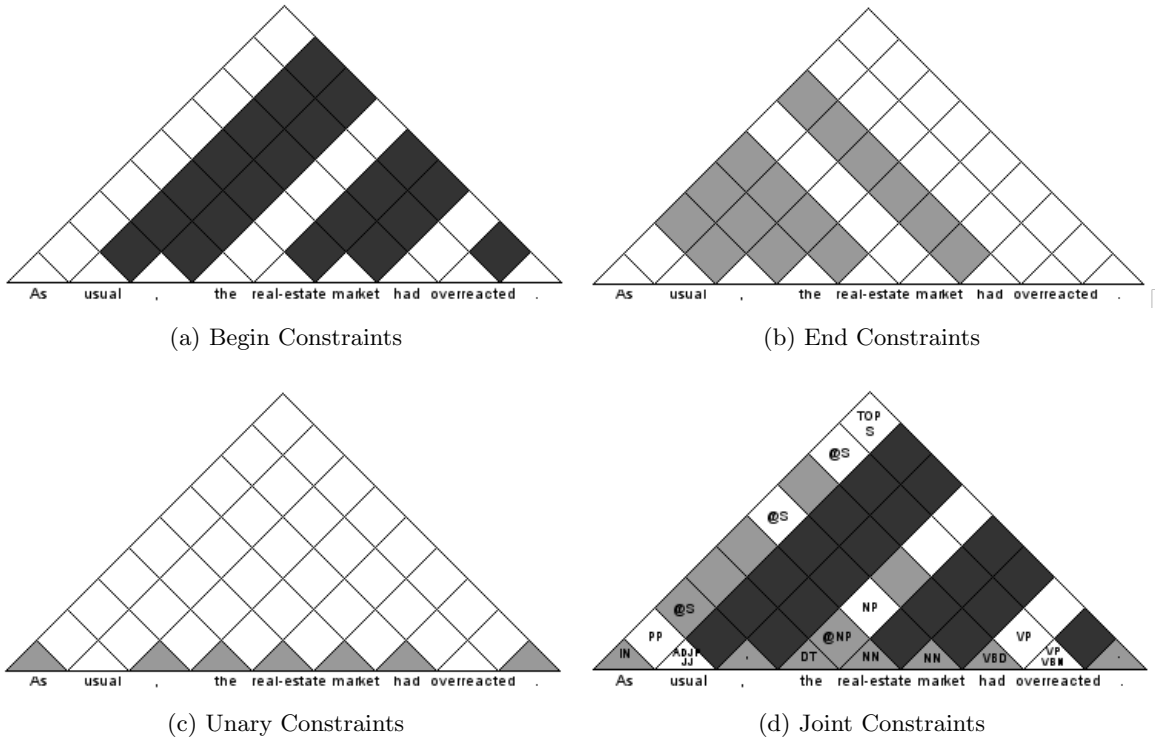
[1] In this example we list the actual words from the sentence for clarity with Figure 5.1, but in practice we classify word positions as a specific word may occur multiple times in the same sentence with potentially different B or E labels.

edges are extended from left-to-right, the incomplete edge should be discarded, because any completed edges that could result from extending that incomplete edge would have the same begin position. Stated another way, if $w_b \in \overline{B}$ for chart cell (b, e) then all chart cells (b, i) for $i > b$ must also be closed. For example, in Figure 5.1, the cell associated with the two-word substring *"real-estate market"* can be closed to both complete and incomplete edges, since *"real-estate"* $\in \overline{B}$, and any complete edge built from entries in that cell would also have to start with the same word and hence would be discarded. Thus, the whole diagonal is closed. However, if $w_b \in B$ and $w_e \in \overline{E}$, such as the cell associated with the two-word substring *"the real-estate"* in Figure 5.1, a complete edge — achieved by extending the incomplete edge — may end at w_i for $i > e$, and cell (b, i) may be open (*"the real-estate market"*). Therefore, incomplete edges must be allowed in cell (b, e).

5.2.2 Unary Constraints

In addition to begin and end constraints, we also introduce unary constraints in span-1 cells. While we cannot close span-1 cells entirely because each of these cells must contain at least one POS tag, we can reduce the population of these cells by restricting the type of constituents they contain. We define a single-word constituent (SWC) as any unary production $A \rightarrow B$ in the grammar such that B is a non-terminal (not a lexicon entry) and the production spans a single word. The productions ADJP $\rightarrow$ JJ and VP $\rightarrow$ VBN in Figure 2.2a are examples of SWCs. Note that TOP $\rightarrow$ S and JJ $\rightarrow$"usual" in Figure 5.2 are also unary productions, but by definition they are not SWC unary productions. We train a distinct tagger, as is done for B and E constraints (see Section 5.3), to label each word position as either in U or $\overline{U}$, indicating that the word position may or may not be extended by a SWC, respectively.

Because the search over possible grammar extensions from two child cells in the CYK algorithm is analogous to a database JOIN operation, the efficiency of this cross-product hinges on the population of the two child cells that are intersected. We focus on constraining the population of span-1 chart cells for three reasons. First, the begin and end constituent constraints only affect chart cells spanning more than one word and leave span-1 chart cells completely unpruned. By pruning entries in these span-1 cells, we



(a) Begin Constraints

(b) End Constraints

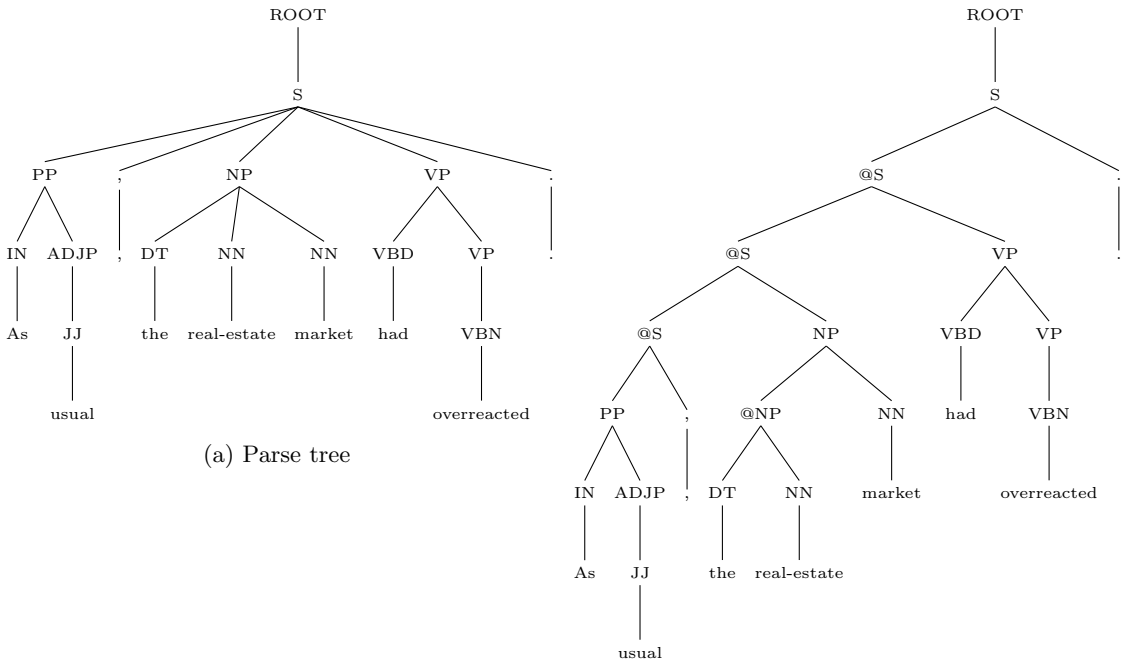
(c) Unary Constraints

(d) Joint Constraints

Figure 5.1: Constraints for the dynamic programming chart used to parse "As usual , the real-estate market had overreacted ." with a left-binarized grammar (See Figure 2.2). Black denotes "closed" cells, white cells are "open," and gray cells are only open to a restricted population (gray cells closed by $\overline{E}$ constraints only allow incomplete edges; gray cells closed by $\overline{U}$ constraints only allow POS tags).

complement multi-word constituent pruning so that all chart cells are now candidates for finite-state tagging constraints. Second, span-1 chart cells are the most frequently queried cells in the CYK algorithm. The search over possible midpoints will always include two cells spanning a single word — one as the first left child and one as the last right child. It is therefore important that the number of entries in these cells be minimized to make bottom-up CYK processing efficient. Finally, as we will show in Table 5.1, only 11.2% of words in the WSJ treebank are labeled with SWC productions. With oracle unary constraints, the possibility of constraining nearly 90% of span-1 chart cells has promising efficiency benefits to downstream processing.

To reiterate, unary constraints in span-1 chart cells never close the cell completely



(a) Parse tree

(b) Left-binarized parse tree

Figure 5.2: Unbinarized and left-binarized parse tree. Non-terminals preceded with the symbol '@' are created through binarization.

since each span-1 cell must contain at least one POS non-terminal. Instead, if $w_i \in \overline{U}$ then we simply do not add phrase-level non-terminals to chart cell (i, i). Similar to chart cells spanning multiple words that cannot be closed completely, these span-1 chart cells partially restrict the population of the cell, which we will empirically show to reduce processing time over unconstrained CYK parsing. These partially closed chart cells are represented as gray in Figure 5.1.

5.2.3 The Constrained CYK Algorithm

Our discussion of cell closing in Section 5.2.1 highlighted different conditions for closing cells for complete and incomplete edges. We will refer to specific conditions in the pseudocode, which we enumerate here. The constraints determine three possible conditions for cell (b, e) spanning multiple words:

1. $w_b \in \overline{B}$: cell is closed to *all* constituents, both complete and incomplete

2. $w_b \in B$ and $w_e \in \overline{E}$: cell is closed to complete constituents
3. $w_b \in B$ and $w_e \in E$: cell is open to all constituents

and two possible conditions for cell (i, i) spanning a single word:

4. $w_i \in \overline{U}$: cell is closed to unary phrase-level constituents
5. $w_i \in U$: cell is open to all constituents

We will refer to a chart cell (b, e) that matches the criteria above as "case 1 cells", "case 2 cells", etc. throughout the remainder of this chapter. Figure 5.1 represents these five cases pictorially, where case 1 cells are black, case 2 and 4 cells are gray, and case 3 and 5 cells are white.

Algorithm 4 contains pseudocode of our modified CYK algorithm that takes into account $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints. Line 4 of Algorithm 4 is the first modification from the standard CYK processing as we now consider only words in set B to begin multi-word constituents (for comparison, see Algorithm 1). Chart cells excluded from this criteria fall into case 1 and require no work. Line 5 determines if chart cell (b, e) is in case 2 (partially open) or case 3 (completely open). If $w_e \in \overline{E}$, then we skip to lines 16-17 and only incomplete edges are permitted. Note that there is only one possible midpoint for case 2 chart cells, which results in constant-time work (see proof in [144]) and unary productions are not considered since all entries in the cell are incomplete constituents, which only participate in binary productions. Otherwise, if $w_e \in E$ on line 5, then the cell is open to all constituents and processing occurs as in the standard CYK algorithm (lines 6-10 of Algorithm 4 are identical to lines 4-8 of Algorithm 1). Finally, unary productions are added in lines 12-14, but restricted to multi-word spanning chart cells, or span-1 cells where $w_b \in U$.

5.3 Tagging Chart Constraints

To better understand the proposed tagging tasks and their likely utility, we will first look at the distribution of classes and our ability to automatically assign them correctly. Note that we do not consider the first word w_1 and the last word w_n during the begin-constituent and end-constituent prediction tasks because they are unambiguous in terms

Algorithm 4 ConstrainedCyk

Pseudocode of a modified CYK algorithm with constituent begin, end and unary constraints. Unary processing is simplified to allow only chains of length one (excluding lexical unary productions). Backpointer storage is omitted.

Input:

$w_1 \dots w_n$: Input sentence

G: Left-binarized PCFG

V': Set of binarized non-terminals from V

B, E, U: Begin, End, and Unary chart constraints

Output:

α: Viterbi-max scores for all non-terminals over every span

ConstrainedCyk$(w_1 \dots w_n,\ G = (V, T, S^{\dagger}, P, \rho), V', B, E, U)$

for $s = 1$ to n **do** ▷ Span width: bottom-up traversal
  for $b = 1$ to $n{-}s{+}1$ **do** ▷ Begin word position
    $e \leftarrow b{+}s{-}1$
    if $w_b \in B$ or $s = 1$ **then** ▷ Case 1 cells excluded
      if $w_e \in E$ or $s = 1$ **then**
        for $A_i \in V$ **do**
          if $s = 1$ **then** ▷ Add lexical productions
            $\alpha_i(b, b) \leftarrow \rho(A_i \rightarrow w_b)$
          else ▷ Case 3: cell open
            $\alpha_i(b, e) \leftarrow \max_{b \le m < e} \left(\max_{j,k} \ \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, m)\ \alpha_k(m+1, e) \right)$
        if $s > 1$ or $w_b \in U$ **then** ▷ Case 5 for span-1 cells
          for $A_i \in V$ **do** ▷ Add unary productions
            $\upsilon_i(b, e) \leftarrow \max \left(\alpha_i(b, e)\ ,\ \max_{j} \ \rho(A_i \rightarrow A_j)\ \alpha_j(b, e) \right)$
          $\alpha(b, e) \leftarrow \upsilon(b, e)$
      else ▷ Case 2: closed to complete constituents
        for $A_i \in V'$ **do** ▷ Only consider binarized non-terminals
          $\alpha_i(b, e) \leftarrow \max_{j,k} \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, e{-}1)\ \alpha_k(e, e)$
return α

	Corpus totals		Begin class		End class		Unary class	
	Strings	Words	B	$\overline{B}$	E	$\overline{E}$	U	$\overline{U}$
English								
Count	39,832	950,028	430,841	439,558	223,544	646,855	105,973	844,055
Percent			49.5	50.5	25.7	74.3	11.2	88.8
Chinese								
Count	18,086	493,708	188,612	269,000	165,591	292,021	196,732	296,976
Percent			41.2	58.8	36.2	63.8	39.9	60.1

Table 5.1: Statistics on extracted word classes for English (Sections 2-21 of the Penn WSJ treebank) and Chinese (articles 1-270 and 400-1151 of the Penn Chinese treebank).

of whether they begin or end constituents of span greater than one. The first word w_1 must begin a constituent spanning the whole string, and the last word w_n must end that same constituent. The first word w_1 cannot end a constituent of length greater than 1; similarly, the last word w_n cannot begin a constituent of length greater than 1. We therefore omit $\overline{B}$ and $\overline{E}$ at these two word positions from prediction, leading to $n-2$ begin-constituent and $n-2$ end-constituent ambiguous predictions for a string of length n.

Table 5.1 displays word statistics from the training sections of the Penn English treebank [114] and the Penn Chinese treebank [180], and the positive and negative class label frequency of all words in the data. From the nearly 1 million words in the English corpus, just over 870 thousand are neither the first nor the last word in the string, therefore possible members of the sets $\overline{B}$ or $\overline{E}$, i.e., neither beginning a multi-word constituent ($\overline{B}$) nor ending a multi-word constituent ($\overline{E}$). Of these 870 thousand words, over half (50.5%) do not begin multi-word constituents, and nearly three quarters (74.3%) do not end multi-word constituents. The skewed distribution of E to $\overline{E}$ reflects the right-branching structure of English. Finally, almost 90% of words are not labeled with a single-word constituent, demonstrating the infrequency at which these productions occur in the English treebank.

The Chinese treebank is approximately half of the size of the English treebank in terms of the number of sentences and word count. Again, we see a bias towards right-branching trees ($|B| > |E|$), but the skew of the distributions is much smaller than it is for English.

	LEX	ORTHO	POS
τ_i	τ_i, w_i	$\tau_i, w_i[0]$	τ_i, φ_i
τ_{i-1}, τ_i	τ_i, w_{i-1}	$\tau_i, w_i[0..1]$	τ_i, φ_{i-1}
τ_{i-2}, τ_i	τ_i, w_{i+1}	$\tau_i, w_i[0..2]$	$\tau_i, \varphi_{i-1}, \varphi_i$
$\tau_{i-2}, \tau_{i-1}, \tau_i$	τ_i, w_{i-2}	$\tau_i, w_i[0..3]$	τ_i, φ_{i+1}
	τ_i, w_{i+2}	$\tau_i, w_i[n]$	$\tau_i, \varphi_i, \varphi_{i+1}$
	τ_i, w_{i-1}, w_i	$\tau_i, w_i[n\text{-}1..n]$	$\tau_i, \varphi_{i-1}, \varphi_i, \varphi_{i+1}$
	τ_i, w_i, w_{i+1}	$\tau_i, w_i[n\text{-}2..n]$	τ_i, φ_{i-2}
		$\tau_i, w_i[n\text{-}3..n]$	$\tau_i, \varphi_{i-2}, \varphi_{i-1}$
		$\tau_i, w_i \subseteq$ Digit	$\tau_i, \varphi_{i-2}, \varphi_{i-1}, \varphi_i$
		$\tau_i, w_i \subseteq$ UpperCase	τ_i, φ_{i+2}
		$\tau_i, w_i \subseteq$ Hyphen	$\tau_i, \varphi_{i+1}, \varphi_{i+2}$
			$\tau_i, \varphi_i, \varphi_{i+1}, \varphi_{i+2}$

Table 5.2: Tagger features for $\overline{B}$, $\overline{E}$, and $\overline{U}$. All lexical (LEX), orthographic (ORTHO), and part-of-speech (POS) features are duplicated to also occur with τ_{i-1}; e.g., $\{\tau_{i-1}, \tau_i, w_i\}$ as a LEX feature.

The most significant difference between the two corpora is the percentage of single-word constituents in Chinese compared to English. Due to the annotation guidelines of the Chinese treebank, nearly 40% of words contain single-word constituent unary productions. Because these occur with such high frequency, we can assume that unary constraints may have a smaller impact on efficiency for Chinese than for English, since an accurate tagger will constrain fewer cells. We still have the expectation, though, that accurate tagging of SWC productions will increase parsing accuracy for both English and Chinese.

To automatically predict the class of each word position, we train a binary predictor from supervised data for each language/word-class pair, tuning performance on the respective development sets (WSJ Section 24 for English and PCTB articles 301-325 for Chinese). Word classes are extracted from the treebank trees by observing constituents beginning or ending at each word position, or by observing single word constituents. We use the tagger from [81] to train six log-linear models (three for English, three for Chinese) with the averaged perceptron algorithm [49].

Table 5.2 summarizes the features implemented in our tagger for $\overline{B}$, $\overline{E}$, and $\overline{U}$ identification. In the table, the φ features are instantiated as POS-tags (provided by a separately

	Markov order		
Tagging Task	**0**	**1**	**2**
English			
$\overline{B}$ (no multi-word constituent begin)	96.7	96.9	96.9
$\overline{E}$ (no multi-word constituent end)	97.3	97.3	97.3
$\overline{U}$ (no span-1 unary constituent)	98.3	98.3	98.3
Chinese			
$\overline{B}$ (no multi-word constituent begin)	94.8	95.4	95.2
$\overline{E}$ (no multi-word constituent end)	96.2	96.4	96.6
$\overline{U}$ (no span-1 unary constituent)	95.9	96.2	96.3

Table 5.3: Tagging accuracy on the respective development sets (WSJ Section 24 for English and PCTB articles 301-325 for Chinese) for binary classes $\overline{B}$, $\overline{E}$, and $\overline{U}$, for various Markov orders.

trained log-linear POS-tagger) and τ features are instantiated as $\overline{B}$, $\overline{E}$, and $\overline{U}$ class labels. The feature set used in the tagger includes the n-grams of surrounding words, the n-grams of surrounding POS-tags, and the B, E, or U tags of the preceding words. The n-gram features are represented by the words within a five-word window of the current word. The tag features are represented as unigram, bigram, and trigram tags (i.e., constituent tags from the current and two previous words). These features are based on the feature set implemented by [158] for NP chunking and can be extracted in constant time. Additional orthographical features are used for unknown and rare words (words that occur fewer than 5 times in the training data), such as the prefixes and suffixes of the word (up to the first and last four characters of the word), the presence of a hyphen, a digit, or a capitalized letter, following the features implemented by [141]. Note that the orthographic feature templates, including the prefix (e.g., $w_i[0..1]$) and suffix (e.g., $w_i[n\text{-}2..n]$) templates, are only activated for unknown and rare words. When applying our tagging model to Chinese data, all feature functions were left in the model as-is, and not tailored to the specifics of the language.

We ran various tagging experiments on the development set and report accuracy results

in Table 5.3 for all three predictions tasks, using Viterbi decoding. We trained models with Markov order-0 (constituent tags predicted for each word independently), order-1 (features with constituent tag pairs) and order-2 (features with constituent tag triples). In general, tagging accuracy for English is higher than for Chinese, especially for the $\overline{U}$ and $\overline{B}$ tasks. Given the consistent improvement from Markov order-0 to Markov order-1 (particularly on the Chinese data), and for the sake of consistency, we have chosen to perform Markov order-1 prediction for all results in the remainder of this paper.

5.4 Experimental Setup

In the sections that follow, we present empirical trials to examine the behavior of chart constraints under a variety of conditions. First, we detail the data and parsers used in these experiments.

5.4.1 Datasets

For English, all stochastic grammars are induced from the Penn WSJ Treebank [114]. Sections 2-21 of the treebank are used as training, Section 00 as held-out (for determining stopping criteria during training and some parameter tuning), Section 24 as development, and Section 23 as test set. For Chinese, we use the Penn Chinese Treebank [180]. Articles 1-270 and 400-1151 are used for training, articles 301-325 for both held-out and development, and articles 271-300 for testing. Supervised class labels are extracted from the non-binarized treebank trees for B, E, and U (as well as their complements).

All results report F_1 labeled bracketing accuracy (harmonic mean of labeled precision and labeled recall) for all sentences in the data set. Specifications for the computer hardware used in the following experiments are consistent with all experiments in this thesis and can be found, along with additional statistics for each corpus, in Section 5.4.

5.4.2 Tagging Methods and Closing Chart Cells

We have three separate tagging tasks, each with two possible tags for every word w_i in the input string: (1) B or $\overline{B}$; (2) E or $\overline{E}$; and (3) U or $\overline{U}$. Our taggers are as described

in Section 5.3.

Within a pipeline system that leverages hard constraints, one may want to choose a tagger operating point that favors precision of constraints over recall to avoid over-constraining the downstream parser. We have two methods for trading recall for precision that will be detailed later in this section, both relying on calculating the cumulative score S_i for each of the binary tags at each word position w_i, i.e., (using B as the example tag):

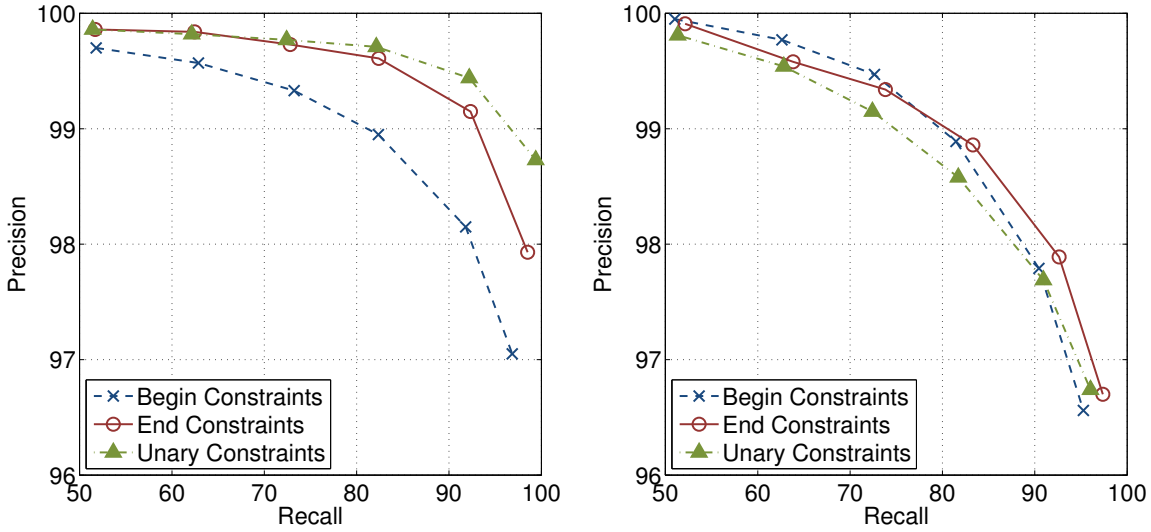
$$S_i(B \mid w_1 \dots w_n) \quad = \quad \log \sum_{\tau_1 \dots \tau_n} \delta(\tau_i, B)\ e^{\Phi(w_1 \dots w_n, \tau_1 \dots \tau_n) \cdot \mathbf{w}} \tag{5.1}$$

where $\sum$ sums over all possible tag sequences for sentence $w_1 \dots w_n$; $\delta(\tau_i, B) = 1$ if τ_i=B and 0 otherwise; $\Phi(w_1 \dots w_n, \tau_1 \dots \tau_n)$ maps the word string and particular tag string to a d-dimensional (global) feature vector; and $\mathbf{w}$ is the d-dimensional parameter vector estimated by the averaged perceptron algorithm. Note that this cumulative score over all tag sequences that have B in position i can be calculated efficiently using the forward-backward algorithm [90]. We can compare the cumulative scores $S_i(B)$ and $S_i(\overline{B})$ to decide how to tag word w_i, and define the cumulative score ratio (CSR) as follows:

$$\mathrm{CSR}(w_i) = S_i(\overline{B}) - S_i(B) \tag{5.2}$$

If we want to tag $\overline{B}$ with high precision, and thus avoid over-constraining the parser, we can change our decision criterion to produce fewer such tags. We present two different selection criteria in the next two subsections.

To show the effect of precision-oriented decision criteria, Figure 5.3 shows the precision/recall tradeoff at various operating points, using the global high-precision method detailed below, for all three tags on both the English and the Chinese development sets. As expected, we see that that the English $\overline{B}$ curve is significantly lower than the English $\overline{E}$ and $\overline{U}$ curves. This is due to the near-uniform prior on $\overline{B}$ in the data ($\overline{E}$ and $\overline{U}$ are much higher-frequency classes). Still, we can achieve 99% precision for $\overline{B}$ with recall above 70%. We do much better with $\overline{E}$ and $\overline{U}$ and see that when precision is 99% for these two tags, recall does not drop below 90%. For the Chinese tagging task, $\overline{B}$, $\overline{E}$ and $\overline{U}$ all have



(a) English (b) Chinese

Figure 5.3: Tagger precision/recall tradeoff of $\overline{B}$, $\overline{E}$, and $\overline{U}$ on the development set for English (a) and Chinese (b).

similar performance as we trade precision for recall. Here, as with the English $\overline{B}$ tag, we achieve 99% precision with recall still above 70%.

We can see from these results that our finite-state tagging approach yields very high accuracy on these tasks, as well as the ability to provide high precision (above 99%) operating points with a tolerable loss in recall. In what follows, we present two approaches to adjusting precision: first by adjusting the overall precision and recall of the tagger directly, as shown above; and second, by adjusting the precision and recall of tagging results on a per-sentence basis. We then discuss how complexity-bounded constraints are implemented in our experiments. In Section 5.5.1 we discuss empirical results showing how adjusting the tagger in these ways affects parsing performance.

Global High Precision

Global high precision constraints (GHP) are implemented using the cumulative score ratios directly to determine if $w_b \in B$ or $\overline{B}$.[2] Given the cumulative score ratio as defined above,

[2]In our experiments these scores range between $\pm 10^3$.

we define the set $\overline{B}$ under global high precision constraints as:

$$\text{GHP}_\lambda(w_i \dots w_n) = \{w_i \in \overline{B} : \text{CSR}(w_i) > \lambda\} \tag{5.3}$$

The value λ=0 provides the highest tagging accuracy, but as mentioned in Section 5.4.2 we may want to increase the precision of the tagger so that the subsequent parser is not over constrained. To do this, we increase the threshold parameter λ towards positive infinity, potentially moving some words in the target sentence from $\overline{B}$ to B. The same procedure is applied to E and U tags.

Sentence-Level High Precision

Global high precision constraints, as defined above, affect the precision of the tagger over an entire corpus, but may or may not affect individual sentences. Because parsing operates on a sentence-by-sentence basis, it may be beneficial to modify the precision of the tagger based on the current sentence being processed. We call this approach sentence-level high precision (HP).[3]

Intuitively, we close the k highest scoring word positions to begin or end multi-word constituents, where k is determined relative to the number of word positions that remain open given the global decision boundary when the posterior score of the tagger is zero. More precisely, to increase tagging precision at the sentence level, we compute the cumulative scores S_i and the cumulative scores ratio (CSR) at all word positions as described above. We next tag all word positions with a CSR(w_i) score less than zero with B, and rank the remaining word positions according to their CSR score. The lowest-ranking λ fraction of this sorted list are tagged as $\overline{B}$, while the rest default to B. When $\lambda = 0$, zero percent of words are tagged with $\overline{B}$, i.e., no constraints are applied; and when $\lambda = 1$, 100 percent of the words with CSR(w_i) > 0 are tagged with $\overline{B}$. This ensures that the high-precision threshold is adapted to each sentence, even if its absolute CSR scores are not similar to others in the corpus.

[3]Roark et al. referred to this approach as simply "high precision constraints" in [145, 146].

Quadratic Bounds

Sentence-level high precision, quadratic bounds, and linear bounds are methods of adding word positions to the set $\overline{B}$ and $\overline{E}$ originally introduced in [145, 146]. We include a description of these methods here for completeness and to give context to the independent contributions we present in this chapter. In [145], Roark et al. impose quadratic bounds on context-free parsing by restricting the number of open cells (case 3 in Section 5.2.3) to be less than or equal to λN for some tuning parameter λ and sentence length N. They show that this constraint provably reduces the complexity of ConstrainedCyk to $O(N^2)$. Imposing quadratic bounds only involves the sets B and E, as unary constraints are restricted to span-1 cells. Given gold parse trees t^* and a function $T(B, E)$ to generate the set of all valid parse trees satisfying constraints B and E, the optimal quadratically bounded constraints are:

$$\text{Quad}_\lambda(w_i \dots w_N) = \underset{B,E}{\text{argmax}}(\max_{t \in T(B,E)} F_1(t^*, t) \text{ s.t. } |\mathcal{C}_3| \leq \lambda N) \tag{5.4}$$

i.e., the set of B and E constraints that maximize F_1 under the limitation that the number of $\mathcal{C}_3$ cells is less than λN. This equation is an instance of combinatorial optimization and solving it exactly is NP-complete. Furthermore, access to gold parse trees t^* is not possible during parsing.

Instead, Roark and Hollingshead [145] rely on the posterior scores from the tagger as a measure of tagging confidence, which they assume to be correlated with the true F_1 accuracy, and use a greedy algorithm to approximate Equation 5.4. For every sentence, they first sort both the B and E CSR scores for each word position into a single list. Next, they assume all word positions are in $\overline{B}$ and $\overline{E}$, then starting from the top of the sorted list (highest posterior probability for set inclusion), they continue to add word positions to their respective open set and compute the number of open cells with the given constraints while $|\mathcal{C}_3| < \lambda N$. By doing this, they guarantee that only a linear number of case 3 cells are open in the chart, which leads to quadratic worst-case parsing complexity.

Linear Bounds

Imposing $O(N)$ complexity bounds requires constraining the size of the set B such that $|B| \leq \lambda$ for some constant λ (see proof in [144]). As with quadratic complexity bounds, we wish to find the optimal set B to fulfill the following requirements:

$$\text{Linear}_\lambda(w_i \dots w_N) = \underset{B}{\text{argmax}}(\max_{t \in T(B)} F_1(t^*, t) \text{ s.t. } |B| \leq \lambda) \tag{5.5}$$

Roark et al. again resort to a greedy method to determine the set B, as this optimization problem is also NP-complete and gold trees are not available. B is constructed by sorting all word positions by their CSR scores for B, and then adding only the highest-scoring λ entries to the inclusion set. All other word positions are closed and in the set $\overline{B}$. Because this method does not consider the set E to impose limits on processing, it will be shown in Section 5.5.2 that $O(N)$ complexity bounding is not as effective (in isolation) as compared to the quadratic complexity or high precision constraints presented above.

5.4.3 Parsers

We will present results constraining several different parsers. We first analyze exhaustive parsing using both the Cyk and the ConstrainedCyk algorithms. We use a basic CYK exhaustive parser, termed the BUBS parser,[4] to parse with a simple PCFG model that uses non-terminal node-labels as provided by the Penn Treebank after removing empty nodes, node indices, and function tags. The results presented here replicate and extend the results presented in [146], using a different CYK parser.[5] The BUBS parser is an open-source high-efficiency parser that is grammar agnostic and can be run in either exhaustive mode or with various approximate inference options (detailed more fully in Section 5.6.1). It has been shown to parse exhaustively with very competitive or superior efficiency compared with other highly optimized CYK parsers [57].

[4] http://code.google.com/p/bubs-parser

[5] Note that there are several differences between the two parsers, including the way in which grammars are induced, leading to different baseline accuracies and parsing times. Most notably, the parser from [146] relied upon POS-tagger output, and collapsed unary productions in a way that effectively led to parent annotation in certain unary chain constructions. The current results do not exploit those additional annotations, hence the baseline F_1 accuracy is slightly lower.

In what follows, we present results with both left- and right-binarized PCFGs induced using a Markov order-2 transform (L2 and R2, respectively)[6], and also present results for parsing Chinese. We will then present results applying finite-state chart constraints to state-of-the-art parsers, and evaluate the additional efficiency gains these constraints provide, even when these parsers are already heavily pruned. For simplicity, Section 5.5.1 focuses on parsing with the simple Markov order-2 grammars, and we then move to higher accuracy parsers in Section 5.6.

5.5 CYK Parsing with Finite-State Chart Constraints

5.5.1 High-Precision Constraints

As mentioned in Section 5.4.2, we can adjust the severity of pruning to favor precision over recall. Figure 5.4a shows parse time versus F_1 labeled parse accuracy on the English development set for the baseline (unconstrained) exact-inference CYK parser, and for various parameterizations of global high precision (GHP), sentence-level high precision (HP), and unary constraints. Note that we see accuracy increasing over the baseline in Figure 5.4a with the imposition of these constraints at some operating points. This is not surprising, though, as the finite-state tagger makes use of lexical information that the simple PCFG does not, hence there is complementary information added that improves the model. The best operating points — fast parsing and relatively high accuracy — are achieved for GHP constraints at λ=40, for sentence-level HP at λ=0.95, and for unary constraints at λ=40. These high-precision parameterizations achieve roughly an order of magnitude speedup and between 0.2 absolute (for unary constraints) and 3.7 absolute (for high-precision constraints) F_1 improvement over the baseline unconstrained parser.

In order to analyze how these constraints affect accuracy with respect to sentence length, we turn to Figure 5.4b. In this plot, F_1 accuracy is computed over binned sentence lengths in increments of ten. All sentences of length greater than 60 are included in the final bin. As one might expect, accuracy declines with sentence length for all models since the number of possible trees is exponential in the sentence length and errors in one portion

[6]Grammar induction is detailed in Section 2.7

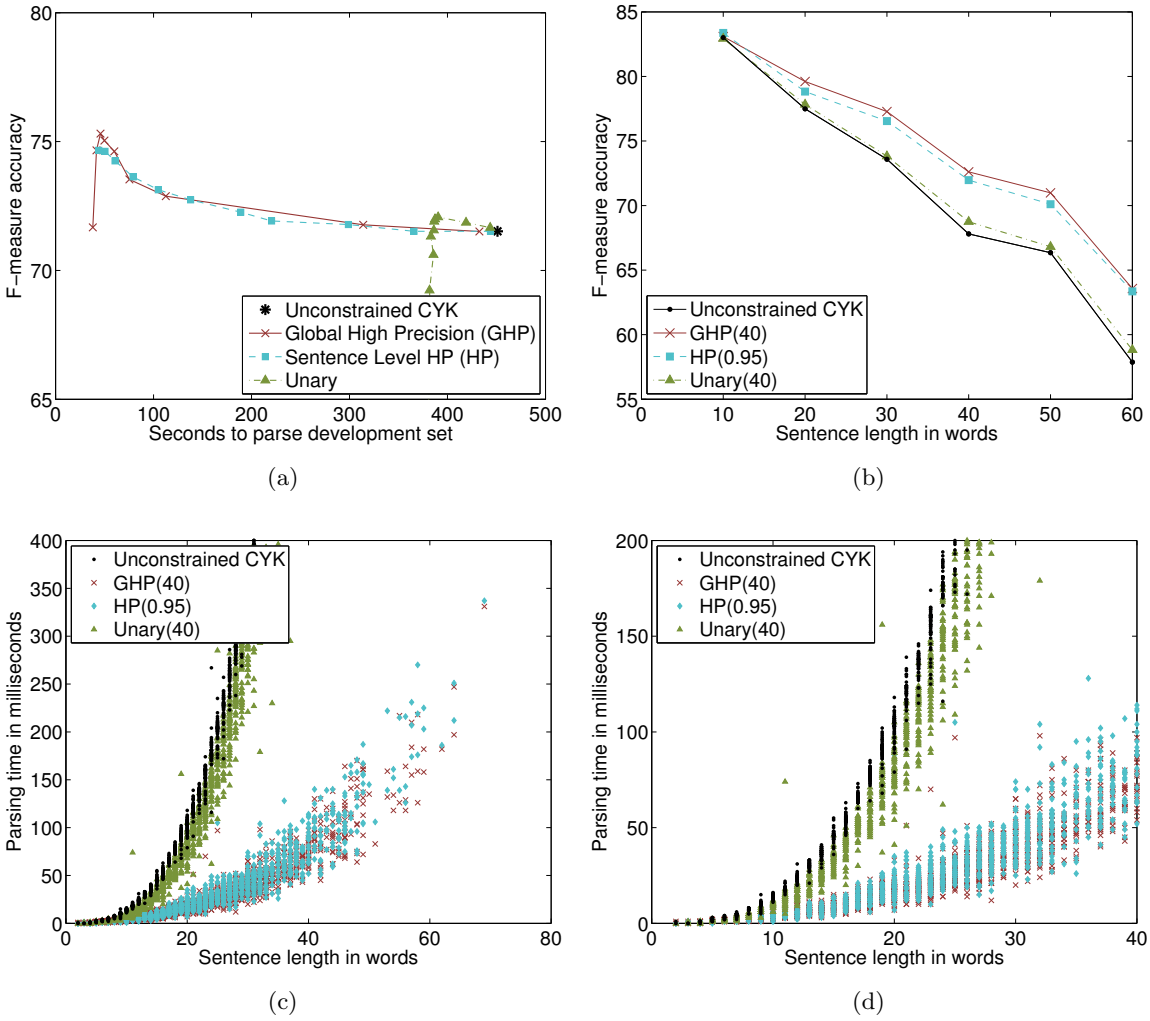


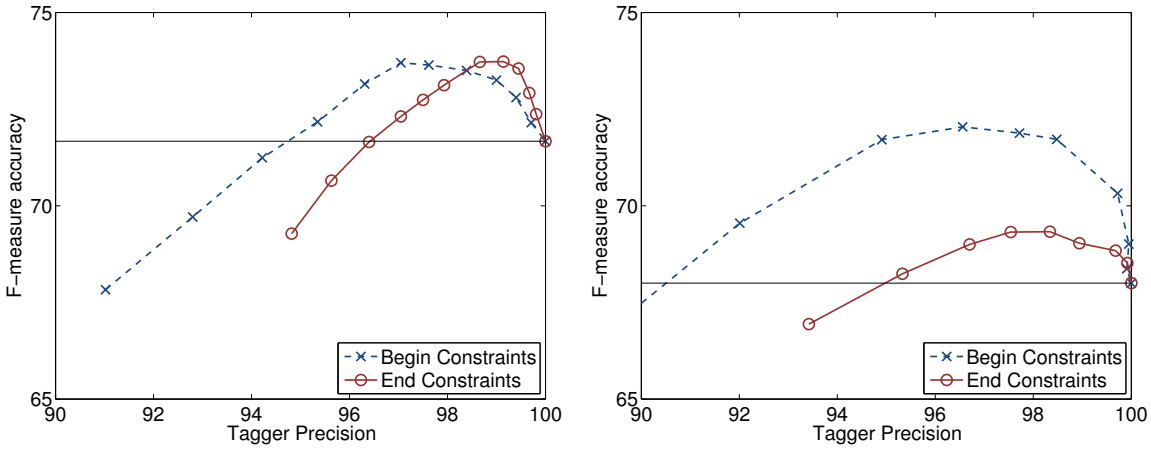
Figure 5.4: English development set results (WSJ Section 24) applying global high precision (GHP), sentence-level high precision (HP), and unary constraints with the ConstrainedCyk algorithm. We sweep over multiple values of λ in (a) and plot results in (b), (c), and (d) with the optimal value for each constraint found in (a). F_1 accuracy in (b) is computed over binned sentence lengths. Figure (d) plots the same data as (c), but zoomed in.

of the tree can adversely affect prediction of other constituents in nearby structure. We see that all constraints provide accuracy gains over the baseline at all sentence lengths, but point out that the gains by GHP B and E constraints are larger for longer sentences. As stated earlier, this is due to the model-correcting behavior of cell constraints: lexical features are leveraged to prune trees that the PCFG may favor but are not syntactically correct with respect to the entire sentence. Because longer sentences are poorly modeled by the PCFG, cell constraints play a larger role in restricting the space of possible trees considered and correcting modeling errors.

We can get a different perspective of how these constraints affect parsing time by considering the scatter plots in Figure 5.4c and 5.4d, which plot each sentence according to its length and parsing time at four operating points: baseline (unconstrained); global high precision at λ=40; sentence-level high precision at λ=0.95; and unary at λ=40. Figure 5.4c shows data points with up to 80 words and 400 milliseconds of parsing time. Figure 5.4d zooms in to under 200 milliseconds and up to 40 words. It can be seen in each graph that unconstrained CYK parsing quickly leaves the plotted area via a steep cubic curve (least-squares fit is $N^{2.9}$). Unary constraints operate at the same cubic complexity as the baseline, but with a constant factor decrease in parsing time (least-squares fit is also $N^{2.9}$). The plots for GHP and HP show dramatic decreases in typical-case runtime compared to the baseline. We again run a least-squares fit to the data and find that both GHP and HP constraints follow a $N^{2.5}$ trajectory.

The empirical complexity and run-time performance of GHP and HP is nearly identical relative to all other chart constraints. But looking a little closer, we see in Figures 5.4c and 5.4d that that GHP constraints are slightly faster than HP for some sentences, and accumulated over the entire development set, this leads to both higher accuracy (75.1 vs. 74.6 F_1) and faster parsing time (46 vs. 50 seconds).[7] This leads to the conclusion that when optimizing parameters for high-precision constraints, we can better tune the overall pipeline by choosing an operating point based on the corpus-level tagger performance as opposed to tuning for sentence-specific precision goals. Consequently, other than Table 5.4,

[7]See Table 5.4 for additional performance comparisons.



(a) English (b) Chinese

Figure 5.5: The effects of tagger precision on parsing F_1. B and E constraints are applied in isolation; no other constraints are used during parsing. Results on the English development set in (a) and Chinese in (b). Baseline unconstrained F_1 accuracy is indicated with the horizontal black line.

we only apply GHP constraints on test set results in the remainder of this chapter.

Although Figure 5.4a displays the constrained F_1 parse accuracy as a function of parsing time, one may also be interested in how tagger precision directly affects parse accuracy. To answer this question, we apply high precision constraints to sets B and E in isolation and plot results in Figure 5.5. Note that when only constraints on E are applied, no chart cells can be completely closed and parsing time does not significantly decrease. Likewise, when we apply constraints on B, a chart cell is either open to all constituents (case 3) or closed to all constituents (case 1). When constraining either B or E, we are predicting the structure of the final parse tree, which — as can be seen in Figure 5.5 — has a large impact on parse F_1 accuracy.

We point out in Figure 5.5 that to achieve optimal parsing F_1, tagger precision must be above 97% for both English and Chinese. For both languages, B constraints are more forgiving and do not adversely affect parsing F_1 as quickly as E constraints. These results may lead one to tune two separate high-precision parameters – one to constrain B and one to constrain E. We ran such experiments but found no significant gains compared to

tying these parameters together.

5.5.2 Complexity-Bounded Constraints

Figure 5.6a plots F_1 accuracy versus time to parse the entire development set for two complexity bounded constraints: $O(N^2)$ and $O(N)$. We also include the previously plotted global high-precision constraints for comparison. The large asterisk in the plot depicts the baseline accuracy and efficiency of standard CYK parsing without constraints. We sweep over various parameterizations for each method, from very lightly constrained to very heavily constrained. The complexity-bounded constraints are not combined with the high-precision constraints for this plot (but are later in the chapter).

As can be seen in Figure 5.6a, the linear-bounded method does not, as applied, achieve a favorable accuracy/efficiency tradeoff curve compared to the quadratic bound or high-precision constraints. This is not surprising, given that no words are excluded from the set E for this method, hence far fewer constraints overall are applied with the linear-bounded constraints. In addition, we see in Figure 5.6b that unlike high precision and quadratic constraints, the linear method decreases F_1 accuracy on longer sentences over the baseline unconstrained algorithm, notably for sentences greater than 50 words. We attribute this to the fact that for longer sentences, say length 65, only λ=16 words are allowed in B for linear constraints, which severely limits the search space for these sentences to the point of pruning gold constituents and decreasing overall accuracy.

Next we turn to the scatter plots of Figures 5.6c and 5.6d. Fitting an exponential curve to the data for each constraint via least-squares, we find that global high-precision constraints follow a $N^{2.5}$ trajectory, quadratic at $N^{1.6}$, and linear at $N^{1.4}$. It is interesting that quadratic constraints actually perform at sub-quadratic run-time complexity. This is because the quadric complexity proof assumes that the linear number of open cells each process $O(N)$ midpoints. But in practice, many midpoints are not considered due to the heavily constrained chart, decreasing the average-case runtime of ConstrainedCyk with quadratically-bounded constraints.

Also interesting is that linear constraints perform worse than $O(N)$ at $N^{1.4}$. We attribute this to the nature of the data set. When parsing with linear constraints, we

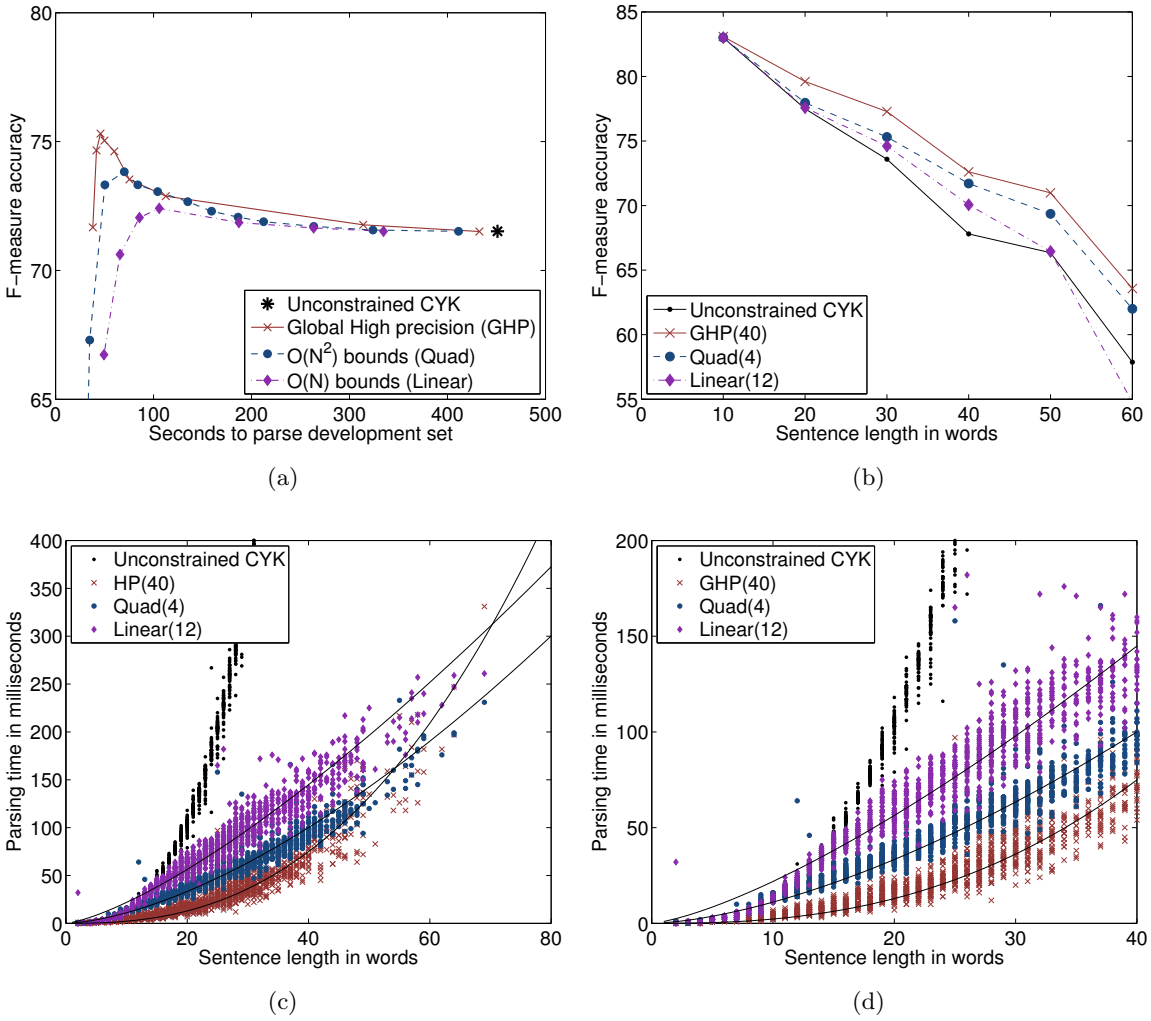


Figure 5.6: English development set results (WSJ Section 24), applying complexity-bounding constraints with the ConstrainedCyk algorithm. We sweep over multiple values of λ in (a) and plot results in (b), (c), and (d) with the optimal value for each constraint found in (a). F_1 accuracy in (b) is computed over binned sentence lengths. Figure (d) plots the same data as (c), but zoomed in.

see that for short sentences parsing complexity is actually cubic. Because we allow a constant number of word positions in the open set B, sentences with length less than λ are completely unconstrained and all chart cells remain open. In Figure 5.6d it looks as if parsing becomes linear after the sentence length notably exceeds λ, around N=20. Assuming our corpus contained a disproportionate number of long sentences, empirical complexity of linear constraints should approach $O(N)$, but due to the large number of short sentences, we see an empirical complexity greater than $O(N)$.

Three important points can be taken away from Figure 5.6. First, although we can provably constrain parsing to linear speeds, in practice this method is inferior to both quadratically-bounded constraints and high-precision constraints. Second, high-precision constraints are more accurate (see Figure 5.6b) and more efficient (see Figure 5.6d) for shorter strings than the quadratically-bound constraints; yet with longer strings the quadratic constraints better control parsing time than the high-precision constraints (see Figure 5.6c). Finally, at the "crossover" point, where quadratic constraints start becoming more efficient than high-precision constraints (roughly 50-60 words, see Figure 5.4c), there is a larger variance in the parsing time with high-precision constraints versus those with quadratic bounds. This illustrates the difference between the two methods of selecting constraints: high-precision constraints can provide very strong typical-case gains, but there is no guarantee of worst-case performance. In this way, the high-precision constraints are similar to other tagging-derived constraints like POS-tags or chunks.

5.5.3 Combining Constraints

Depending on the length of the string, the complexity-bound constraints may close more or fewer chart cells than the high-precision constraints — more for long strings, fewer for short strings. We can achieve worst-case bounds along with superior typical-case speedups by combining both methods. This is accomplished by taking the union of high-precision $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints with their respective complexity-bounded sets.

When combining complexity-bound constraints with high-precision constraints, we first chose operating parameters for each complexity-bounded method at the point where efficiency is greatest and accuracy has yet to decline. These operating points can be seen as

	Constraints	F_1	LP	LR	Seconds	Speedup
R2 Grammar	None (baseline CYK)	71.5	74.5	68.8	451	
	GHP(40)	75.3	78.6	72.3	46	9.8x
	HP(0.95)	74.6	77.8	71.7	50	8.9x
	Quad(4)	73.8	77.0	70.9	70	6.4x
	Linear(12)	72.4	75.4	69.6	106	4.3x
	Unary(40)	72.0	75.4	68.9	386	1.2x
	HP(0.95) + Quad(4)	74.6	77.8	71.7	48	9.5x
	HP(0.95) + Linear(12)	74.4	77.6	71.5	48	9.5x
	GHP(40) + Quad(4)	75.3	78.5	72.3	45	10.0x
	GHP(40) + Linear(12)	75.1	78.4	72.1	44	10.2x
	GHP(40) + Unary(40)	75.7	79.4	72.4	34	13.4x
	GHP(40) + Unary(40) + Quad(4)	75.8	79.5	72.4	33	13.9x
L2 Grammar	None (baseline CYK)	71.7	74.5	69.1	774	
	GHP(40)	75.4	78.5	72.5	66	11.2x
	HP(0.95)	74.9	77.9	72.1	75	10.3x
	Quad(4)	74.0	77.0	71.2	99	7.8x
	Linear(24)	71.7	74.5	69.1	448	1.7x
	Unary(40)	71.9	75.2	69.0	607	1.3x
	HP(0.95) + Quad(4)	74.9	78.0	72.1	69	11.2x
	HP(0.95) + Linear(24)	74.7	77.8	71.9	71	11.0x
	GHP(40) + Quad(4)	75.4	78.5	72.5	62	12.6x
	GHP(40) + Linear(24)	75.2	78.4	72.3	62	12.6x
	GHP(40) + Unary(40)	75.7	79.3	72.5	37	20.9x
	GHP(40) + Unary(40) + Quad(4)	75.7	79.3	72.5	37	20.9x

Table 5.4: English development set results (WSJ Section 24) for the ConstrainedCyk algorithm with both left- and right-binarized Markov order-2 grammars (L2 and R2, respectively) under various individual and combined constraints.

the 'knee' of the curves in Figure 5.6a. For the quadratic complexity method, we set λ=4, limiting the number of open cells to $4N$. For the linear complexity method, we set λ=12, limiting the number of word positions in B to a maximum of 12 members.

Table 5.4 displays F_1 accuracy and parsing time in seconds for many individual and combined constraints on the development set: unconstrained CYK parsing; unary constraints; global high-precision (GHP) constraints; sentence-level high-precision (HP) constraints; $O(N^2)$ and $O(N)$ complexity-bounded constraints (Quad and Linear, respectively). We present all parsing results in Table 5.4 using both a left- and right-binarized Markov order-2 grammar so that the effects of grammar binarization on finite-state constraints can be evaluated. Pre-processing the grammar with a right or left binarization alters the nature and distribution of child non-terminals for grammar productions. Because $\overline{B}$ and $\overline{E}$ constraints prune the chart differently depending on the grammar binarization, we suspect that one method may outperform the other due to the branching bias of the language being parsed. Note that the examples in this chapter have assumed a left-binarized grammar. Applying chart constraints to a right-binarized grammar is identical modulo the reversal of the $\overline{B}$ and $\overline{E}$ classes.

We find three general trends in Table 5.4. First, the efficiency benefits of combining constraints are relatively small. We suspect this is because the dataset contains mostly shorter sentences. Global high-precision constraints outperform the complexity bounded constraints on sentences of length 10 to 50, which makes up the majority of the development set. It is not until we parse longer sentences that the trends start to differ and exhibit characteristics of the complexity bounds. Thus by combining high-precision and complexity constraints, we attain the typical-case efficiency benefits of high-precision constraints with worst-case complexity bounds.

Second, we see that the efficiency gain combining unary and high-precision constraints is more than additive. Unary constraints alone for the right-binarized grammar decreased parsing time by 1.3x, but in conjunction with high-precision constraints, parsing time is decreased from 66 seconds to 37 seconds, an additional 1.8x speedup. We suspect that this additional gain comes from cache efficiencies — due to the heavily pruned nature of the chart, the population of commonly-queried span-1 chart cells have a higher likelihood of

remaining in high-speed cache, decreasing overall parsing time — but we leave empirical verification of this hypothesis to future work.

The third trend we see in Table 5.4 is that there are significant efficiency differences when parsing with a right- or left-binarized grammar. The difference in baseline performance has been previously studied [164, 56], and our results confirm that a right-binarized grammar has superior efficiency for parsing the WSJ treebank due to the right-branching bias of parse trees in this corpus. Furthermore, linear constraints are far less effective with a left-binarized grammar, requiring a tuning parameter of λ=24 such that accuracy was not adversely affected (compare with λ=12 for right-binarized results). This is also caused by the branching bias inherent in the treebank. For example, consider a left-binarized grammar and linear constraints; in the extreme case where λ=0, only w_1 will be in the open set B, forcing all constituents in the final tree to start at w_1. This results in a completely left-branching tree. With a right-binarized grammar, only the last word position will be in E, resulting in a completely right-branching tree. Thus, an overly-constrained linear-bounded parse will favor one branching direction over the other. Because the treebank is biased towards right-branching trees, a right-binarized grammar is more favorable when linear constraints are applied.

Finally, we note that after all constraints have been applied, the accuracy and efficiency differences between the two binarization strategies nearly disappear.

To validate the selected operating points on unseen data, we present results on the test sets for English in Table 5.5 and Chinese in Table 5.6. We parse individually with global high-precision and unary constraints, then present the combined result as this gave the best performance on the development set. In all cases, we see efficiency improvements greater than 10-fold and accuracy improvements of 4.4 absolute F_1 for English, and 8.4 absolute F_1 for Chinese. Note that these efficiency improvements are achieved with no additional techniques for speeding up search. Other than our imposed chart constraints, the CYK parsing is exhaustive and explores all possible category combinations from all open child cells. Techniques such as coarse-to-fine, A* parsing, or beam-search are all orthogonal to the current approach, and could be applied in conjunction to achieve additional speedups.

In the next section, we investigate the use of chart constraints with a number of

	Constraints	F_1	LP	LR	Seconds	Speedup
R2 Grammar	None (baseline CYK)	71.7	74.6	69.0	628	
	Unary(40)	72.1	75.5	69.0	525	1.2x
	GHP(40)	75.8	79.0	72.8	75	8.4x
	GHP(40) + Unary(40)	76.1	79.8	72.7	57	11.0x
L2 Grammar	None (baseline CYK)	72.0	74.8	69.4	1,063	
	Unary(40)	72.4	75.8	69.4	910	1.2x
	GHP(40)	76.1	79.3	73.2	106	10.1x
	GHP(40) + Unary(40)	76.4	80.0	73.1	60	17.9x

Table 5.5: English test set results (WSJ Section 23) for the ConstrainedCyk algorithm with both left- and right-binarized Markov order-2 grammars.

	Constraints	F_1	LP	LR	Seconds	Speedup
R2 Grammar	None (baseline CYK)	60.5	64.6	56.9	157	
	Unary(20)	62.3	67.6	57.8	141	1.1x
	GHP(20)	66.9	71.4	63.0	17	9.1x
	GHP(20) + Unary(20)	68.9	74.4	64.1	15	10.2x
L2 Grammar	None (baseline CYK)	60.4	64.5	56.9	269	
	Unary(20)	62.1	67.2	57.8	234	1.2x
	GHP(20)	66.0	70.5	62.1	33	8.2x
	GHP(20) + Unary(20)	68.0	73.5	63.2	23	11.7x

Table 5.6: Chinese test set results (PCTB Sections 271-300) for the ConstrainedCyk algorithm with both left- and right-binarized Markov order-2 grammars.

high-accuracy parsers, and empirically evaluate the combination of our chart constraint methods with popular heuristic search methods for parsing. These parsers include the Charniak parser, the Berkeley parser, and the BUBS parser.

5.6 High-Accuracy Parsing with Finite-State Chart Constraints

In this section we evaluate the additive efficiency gains provided by finite-state constraints to state-of-the-art parsers. We apply $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints to three parsers, all of which already prune the search space to make decoding time with large grammars more practical. Each high-accuracy parser that we evaluate also prunes the search space in a different way: best-first search, coarse-to-fine pruning, and beam-search. Applying finite-state constraints to these three distinct parsers demonstrates how our constraints interact with other efficient search algorithms we have discussed in this thesis. In what follows we describe the pruning inherent in each of the three parsers and how we apply finite-state constraints within each framework.

5.6.1 Parsers

Charniak parser: The Charniak [32] parser is a multi-stage, best-first parser that can be constrained by pruning edges before they are placed on the agenda. The first stage of the Charniak parser uses a global agenda, as described in Section 3.1.1, and a simple PCFG to build a sparse chart, which is subsequently used to limit the search in later stages with the bi-lexicalized PCFG model. We focus our description of the Charniak parser on this first stage, since it is here that we will apply finite-state chart constraints. The edges on the agenda and in the chart are dotted rules, as described in Section 2.7. When edges are created, they are pushed onto the agenda. Edges that are popped from the agenda are placed in the chart, and then combined with other chart entries to create new edges that are pushed onto the agenda. Once a complete edge spanning the whole string is placed in the chart, at least one full solution must exist. Instead of terminating the initial chart population at this point, over-parsing is employed and edges continue to be added to the chart (and agenda) until a parameterized number of additional edges have been added (see Section 4.3.2). A small over-parsing value will heavily constrain the search space of the later stages within the pipeline, while a large value will often increase accuracy at the expense of efficiency. Upon reaching the desired number of edges, the next stage of the

pipeline receives the chart as input and any edges remaining on the agenda are discarded. See Section 3.1.1 for more details on over-parsing.

We constrain the first stage of the Charniak parser by restricting edges that can be added to the global agenda. When an edge is created for cell (b, e), it is not placed on the agenda if either of the following two conditions hold: 1) $w_b \in \overline{B}$; or 2) the edge is complete and $w_e \in \overline{E}$. With these constraints, a large number of edges that would have previously been considered in the first stage of this pipeline will now be ignored. This allows us to either reduce the amount of over-parsing, which will increase efficiency, or maintain the over-parsing threshold and expand the search space in more promising directions according to the chart constraints. In this chapter we have chosen to do the latter. Note that speedups are still observed, presumably due to the parser finding a complete edge spanning the whole sentence more quickly, thus leading to slight reductions in total edges added to the chart.

Berkeley parser: The Berkeley parser [132] is a multi-level coarse-to-fine parser that operates over a set of coarse-to-fine grammars, $G_1 \ldots G_t$. At each grammar level, the inside and outside constituent probabilities are computed with coarse grammar G_i and used to prune the subsequent search within G_{i+1}. This continues until the sentence is parsed with the target grammar G_t. The initial grammar G_1 is a Markov order-0 grammar, and the target grammar G_t is a latent-variable grammar induced through multiple iterations of splitting and merging non-terminals to maximize the likelihood of the training data [133]. This explicit target grammar is very large, consisting of 4.3 million productions, 2.4 million of which are lexical productions. We refer to G_t as the Berkeley grammar.

We apply chart constraints to the Berkeley parser during the initial inside pass with grammar G_1. Inside scores are computed via the ConstrainedCyk algorithm of Algorithm 4 modulo the fact that the standard inside-outside sum over scores is used in lines 10, 13, and 17 instead of the Viterbi-max. It is unnecessary to constrain either the subsequent outside pass or the search with the following grammars $G_2 \ldots G_t$, as the coarse-to-fine algorithm only considers constituents remaining in the chart from the previous coarse grammar. Reducing the population of chart cells during the initial G_1 inside

pass consequently prunes the search space for all levels of the coarse-to-fine search. We also note that even though there are t=6 grammar levels in our experiments with the Berkeley parser, exhaustive parsing with G_1 consumes nearly 50% of the total parse time (Petrov, p.c.). Applying chart constraints during this initial pass is where we see the largest efficiency gain — much more than when chart constraints supplement coarse-to-fine pruning in subsequent passes.

BUBS parser: The bottom-up beam-search parser (BUBS) is a variation of the CYK algorithm where, at each chart cell, all possible edges are sorted by a prioritization function and only the edges satisfying the k-best and a relative probability difference constraints are retained [17]. Pseudocode and analysis of beam-search parsing can be found in Section 4.4. The following experiments use the Constituent POS Boundary prioritization function detailed in Section 4.2. The BUBS parser is grammar agnostic, so to achieve high accuracy we parse with the Berkeley latent variable grammar (G_t described in the previous subsection), yet only require a single pass over the chart. The BUBS parser performs Viterbi decoding and does not marginalize over the latent variables or compute the max-rule solution as is done in the Berkeley parser. This leads to a lower F_1 score in the final results even though both parsers use the same grammar.

In this chapter, we apply finite-state constraints to the BUBS parser in a fashion almost identical to the ConstrainedCyk algorithm. Because the BUBS parser is a beam-search parser, the difference is that instead of retaining the max score for all non-terminals A_i at each chart cell, we only retain the max score for those edges satisfying the beam-search pruning criteria. Otherwise, $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints are used to prune the search space in the same way.

5.6.2 High-Accuracy Parsing Results

Figure 5.7 displays accuracy and efficiency results from applying three independent constraints to the three parsers: high precision, quadratically bounded, and unary constraints. We sweep over possible tuning parameters from unconstrained (baseline asterisk) to overly

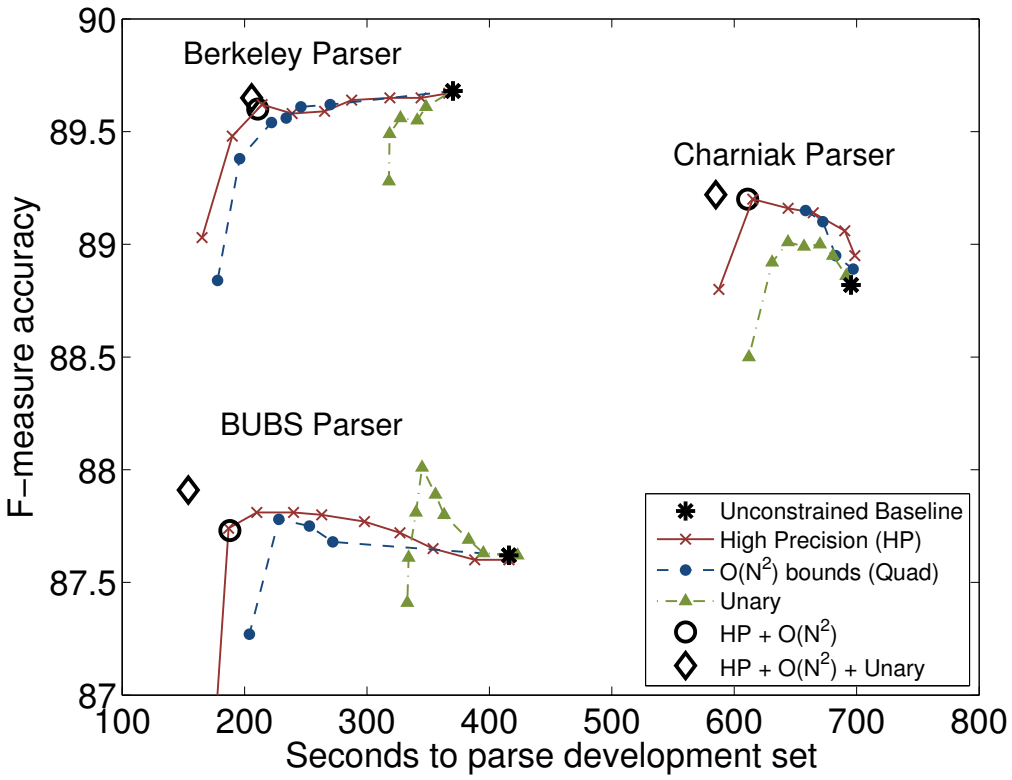


Figure 5.7: English accuracy and efficiency results of applying high precision, quadratic, and unary constraints at multiple values of λ to the Charniak, Berkeley, and BUBS parsers, all of which already heavily prune the search space. For implementation reasons, we apply HP constraints here instead of GHP.

constrained such that F_1 accuracy is adversely affected. We also plot the optimal combination of high precision and quadratic constraints (diamond) and the combination of all three constraints (circle) for each parser which was computed via a grid search over the joint parameter space.

There are many interesting patterns in Figure 5.7. First, all three constraints independently improve the accuracy and efficiency of all three parsers, with the exception of accuracy in the Berkeley parser. This is a significant result considering each of these parsers is simultaneously performing various alternative forms of pruning, which were (presumably) tuned for optimal accuracy and efficiency on this same data set. We also note that the efficiency gains from all three constraints are not identical. In particular, high

precision and quadratic constraints outperforms unary constraints in isolation. But this should be expected as unary constraints only partially close $O(N)$ chart cells while both high precision and quadratic constraints affect $O(N^2)$ chart cells. Nevertheless, looking at the optimal point combining all three constraints, we see that adding unary constraints to begin/end constraints does provide additional gains (in both accuracy and efficiency) for the BUBS and Charniak parsers.

The Berkeley parser realizes efficiency benefit from $\overline{B}$ and $\overline{E}$ constraints, but sees almost no gain from unary constraints. The reason for this has to do with the implementation details of combining (joining) two child cells within the inner loop of the CYK algorithm (line 8 in Algorithm 1). In bottom-up CYK parsing, to extend derivations of adjacent substrings into new constituents spanning the combined string, one can either iterate over all binary productions in the grammar and test if the new derivation is valid (we call this "grammar loop"), or one can take the cross-product of active entries in the cells spanning the substrings and poll the grammar for possible derivations (we call this "cross-product"). With the cross-product approach, fewer active entries in either child cell leads to fewer grammar access operations. Thus, pruning constituents in smaller-span cells directly affects the overall efficiency of parsing. On the other hand, with the grammar loop method there are a constant number of grammar access operations (i.e., the number of grammar productions) and the number of active entries in each child cell has little impact on efficiency. Therefore, with the grammar loop implementation of the CYK algorithm, pruning techniques such as unary constraints will have very little impact on the final run-time efficiency of the parser unless the list of grammar productions is modified given the constraints. For example, alternate iterators over grammar productions could be created such that they only consider a subset of all possible productions. If the left child is a span-1 chart cell in $\overline{U}$, then only grammar productions with a POS tag as the left child need to be considered. Looping over this smaller list, opposed to all possible grammar productions, can reduce the overall runtime. The Berkeley parser contains the grammar-loop implementation of the CYK algorithm. Although all grammar productions are iterated over at each cell, the parser maintains meta-information indicating where non-terminals have been placed in the chart, allowing it to quickly skip over a subset of

the productions that are incompatible with state of the chart. This optimization improves efficiency, but does not take full advantage of restricted cell population imposed by unary constraints, and thus does not benefit as greatly when compared to the BUBS or Charniak parsers.

The second trend we see in Figure 5.7 is that F_1 accuracy improves with additional constraints. This is expected with the Charniak parser as we keep the amount of search space fixed in hopes of pursuing more accurate global paths, but both the Berkeley and BUBS parsers are simply eliminating search paths they would have otherwise considered. Although it is unusual for pruning methods to lead to higher accuracy during search, it is not wholly unexpected here as our finite-state tagger makes use of lexical relationships that the PCFG does not (i.e., lexical relationships based on the linear string rather than over the syntactic structure). By leveraging this new information to constrain the search space, we are indirectly improving the quality of the model. We also suspect that the Berkeley parser sees less of an accuracy improvement than the BUBS parsers because the coarse-to-fine pruning within the Berkeley parser is more "globally informed" than the beam-search within the BUBS parser. By leveraging the coarse-grained inside/outside distribution of trees over the input sentence, the Berkeley parser can more intelligently prune the search space with respect to the target grammar and may not benefit from the additional information inherent in the finite-state tagging model.

The third observation we point out in Figure 5.7 is that we see no additional gains from combining high-precision constraints with quadratic complexity constraints. With all three parsers, high-precision constraints are empirically superior to quadratic constraints, even though high-precision constraints come with no guarantee on worst-case complexity reduction. It is our hypothesis that the additional pruning provided by quadratic constraints for exhaustive CYK parsing is already removed by the internal pruning of each of the three high-accuracy parsers. We therefore report testing results using only high-precision and unary constraints for these high-accuracy parsers.

We apply models tuned on the development set to unseen English test data (WSJ Section 23) in Table 5.7, and Chinese test data (PCTB articles 271-300) in Table 5.8. For English,

Parser	F_1	LP	LR	Seconds	Speedup
BUBS (2010)	88.4	88.5	88.3	586	
+ Unary(100)	88.5	88.7	88.3	486	1.2x
+ HP(0.9)	88.7	88.9	88.6	349	1.7x
+ HP(0.9) + Unary(100)	88.7	89.0	88.4	283	2.1x
Charniak (2000)	89.7	89.7	89.6	1,116	
+ Unary(100)	89.8	89.8	89.7	900	1.2x
+ HP(0.8)	89.8	90.0	89.6	716	1.6x
+ HP(0.8) + Unary(100)	89.7	90.0	89.5	679	1.6x
Berkeley (2007)	90.2	90.3	90.0	564	
+ Unary(125)	90.1	90.3	89.9	495	1.1x
+ HP(0.7)	90.2	90.4	90.0	320	1.8x
+ HP(0.7) + Unary(125)	90.2	90.4	89.9	289	2.0x

Table 5.7: English test set results (WSJ Section 23) applying sentence-level high precision and unary constraints to three parsers with parameter settings tuned on development data. For implementation reasons, we apply HP constraints here instead of GHP.

we see similar trends as we did on the development set results: decoding time is nearly halved when chart constraints are applied to these already heavily-constrained parsers, without any loss in accuracy. We also see independent gains from both unary and high-precision constraints, and additive efficiency gains when combined.

Applying chart constraints to Chinese parsing in Table 5.8 gives substantially larger accuracy and efficiency gains than English for both the BUBS and Berkeley parser. In particular, the accuracy of the BUBS parser increases by 2.3 points absolute (p=0.0002), and the Berkeley parser increases by 0.8 points absolute to 84.7 (p=0.0119), the highest accuracy we are aware of for an individual model on this data set.[8] These increases relative to English may be surprising as chart constraint tagging accuracy for Chinese is worse

[8]Significance was tested using stratified shuffling. Zhang et al. [184] reports an F_1 of 85.5 with a k-best combination of parsers, and Burkett et al. [23] reports an F_1 of 86.0 by leveraging parallel English data for training, but our model is trained from the Chinese treebank alone and is integrated into the Berkeley parser, making it very efficient.

Parser	F_1	LP	LR	Seconds	Speedup
BUBS (2010)	79.3	79.5	79.1	169	
+ Unary(50)	80.7	82.1	79.4	153	1.1x
+ HP(0.8)	81.1	81.5	80.7	75	2.3x
+ HP(0.8) + Unary(50)	81.8	83.1	80.5	44	3.8x
Berkeley (2007)	83.9	84.5	83.3	141	
+ Unary(50)	84.5	85.9	83.0	125	1.1x
+ HP(0.7)	84.5	85.1	83.8	64	2.2x
+ HP(0.7) + Unary(50)	84.7	86.1	83.4	57	2.5x

Table 5.8: Chinese test set results (PCTB articles 271-300) applying sentence-level high-precision and unary constraints to two parsers with parameter settings tuned on development data.

than English (see Table 5.3). We attribute this large gain to the lower baseline accuracy of parsing with the Chinese treebank, allowing our method to contribute additional syntactic constraints that were otherwise unmodeled by the PCFG.

5.7 Conclusion

We have presented a comprehensive analysis of the finite-state pre-processing methods in [145, 146] to constrain context-free parsing that reduce both the worst-case complexity and overall run time. In this chapter we have extended prior work by constraining unary constituents in span-1 chart cells, which demonstrably improves parsing efficiency and presents a cohesive paradigm to constrain all cells in the dynamic programming chart. In Section 5.4.2 we introduced global high-precision constraints, which provided superior performance in empirical trials over sentence-level high-precision constraints presented in [145]. Together, applying begin, end, and unary constraints to context-free parsing increases efficiency by over 20-times for exhaustive CYK parsing, and nearly doubles the speed of the Charniak and Berkeley parsers — both of which have been previously tuned for optimal accuracy/efficiency performance. The combination of chart constraints with coarse-to-fine pruning also produces the highest labeled F_1 accuracy to date on the Penn

Chinese Treebank [180]. We have shown that our methods generalize across multiple grammars and languages, and that constraining both multi-word chart cells and single-word chart cells produces additive efficiency gains.

In the next chapter we present additional methods to constrain the dynamic programming chart by taking into account the population and structure of chart cells at a finer level. For example, the B and E constraints presented here do not consider the height of the constituent they are predicting, but instead open or close entire diagonals in the dynamic programming chart. We will show that opening and closing chart cells on a cell-by-cell basis — in effect, predicting the unlabeled constituent structure — can provide even further stand-alone efficiency benefit for context-free parsing than what we have presented above. The combination of the two methods, which we demonstrate in Chapter 7, has complementary pruning performance and the methods together provide superior efficiency gains relative to each in isolation.

Chapter 6

Unlabeled Constituent and Beam-Width Prediction

6.1 Introduction

Using a tagger within a parsing pipeline to prune the the search space of downstream context-free inference is an effective approach to efficient parsing. In Chapter 5 we presented finite-state pre-processing methods to close portions of the dynamic programming chart during parsing. This was accomplished via three $O(N)$ taggers to classify each word in the input string as either beginning or ending a multi-word constituent, or allowing unary productions in span-1 chart cells. By restricting an individual word position to begin or end any constituent, finite-state chart constraints are effectively closing diagonals in the dynamic programming chart cell. Put another way, if no constituent is allowed to begin at word w_b, then all chart cells (b, e) will be closed for values $e > b$.

In the Ratnaparkhi pipeline [141], an NP-chunker was used to identify continuous noun phrases within the input sentence as a pre-processing step to full context-free parsing. Given the identified noun phrases, the exact chart cells (i.e., substring spans) are identified where NP constituents may be placed and only grammar productions headed by NP need to be considered. Furthermore, assuming an NP constituent spans the substring $w_b \dots w_e$, all chart cells (i, j) and (j, k) can be closed where $i < b < j < e < k$. These are the chart cells that would cross brackets with the identified noun phrase, and since the final parse tree must conform to the pre-processing constraints including the location of the identified noun-phrase, then any tree with constituents in these cells can be pruned.

The chart constraints method we presented in the previous chapter is a generalization from NP phrases to all unlabeled constituents in the final parse tree. But by classifying possible beginning and ending positions independently, we can not prune as much of the chart space as if we knew the exact substrings spanned by possible constituents (i.e., as is known for noun phrases in the Ratnaparkhi pipeline). For example, consider Figure 6.1, where chart cells have been fully (black) and partially (gray) closed with finite-state chart constraints. If we can predict that a constituent spans "the new savings-and-loan bailout agency", then all spans that cross brackets with this substring can be closed (since constituent parse structure can not have overlapping spans). We indicate these cells that can be safely closed with red, many of which are open or partially open given finite state chart constraints. Furthermore, we observe that sentence-final punctuation most often attaches directly to the constituent spanning the entire sentence. Because this lexical item necessarily ends a constituent, under the chart constraints paradigm all chart cells spanning (i, n) for $i < n$ are open to ending a constituent (some may be closed if not allowed to being a constituent). If we instead predict where the sentence-final punctuation will attach, we can close all other cells along this diagonal.

Zhang et al. [185] independently recognized this problem and proposed a solution called "level tagging." Instead of predicting a binary class label to determine if a constituent may begin or end at a given word position, level tagging uses a multi-class classifier to predict the span width of a constituent beginning or ending at a given word position, where a span-width of zero restricts constituents of any length. In practice, the number of classes was restricted to K=4 and only a marginal efficiency improvement (from 33.9 to 34.9 sentences per second) was observed over binary classification within the CCG parser of Clark and Curran [42].

In this chapter, we introduce two novel methods to classify each chart cell as either open or closed to all, or a restricted set of constituents. Given features extracted from the chart cell context — e.g., span width, POS-tags and words surrounding the boundary of the cell — we train a log linear model to, in essence, predict the unlabeled constituent structure of the final parse tree. Given these constraints, we can restrict context-free inference to a much smaller portion of the dynamic programming chart, greatly increasing parsing

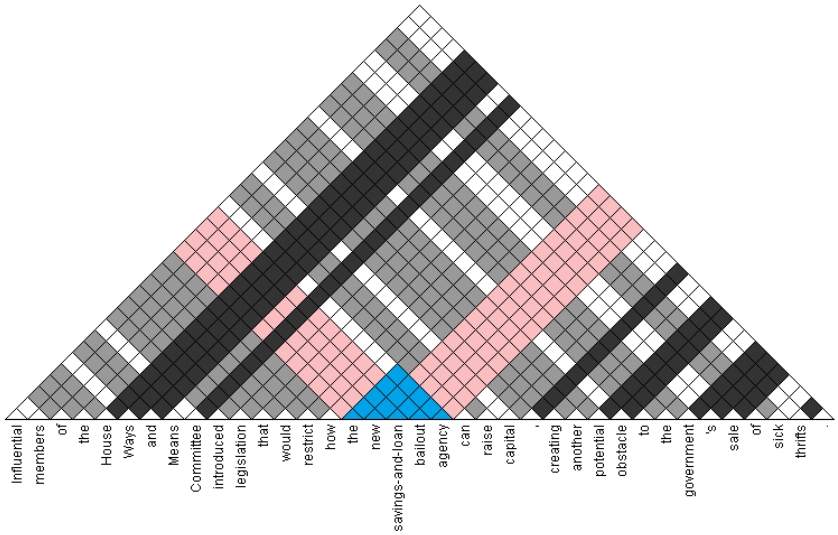


Figure 6.1: Example of finite-state chart constraints restricting the population of chart cells. Black cells are completely closed, gray cells are partially closed. Blue cells highlight the span of the proposed noun phrase; red cells highlight chart cells that can be closed assuming this noun phrase exists, many of which are open or partially open under chart constraints.

efficiency. This approach can be seen as a generalization of finite-state chart constraints presented in Chapter 5, as well as the work by Roark and Hollingshead [145, 146], where $O(N^2)$ predictions are made to classify each chart cell individually, as opposed to $O(N)$ classifications of each word position. As we will show in Section 6.4, cell classification time is dwarfed by the later $O(N^3)$ context-free inference and the additional constraints provided by cell-by-cell classification is empirically justified.

The second major contribution of this chapter is beam-width prediction. As discussed in Chapter 4, beam-search parsing relies on two tunable parameters to control the amount of search space pruned in each chart cell, which we call beam-K and beam-R. Traditionally, these parameters are tuned to a single constant that optimizes accuracy and efficiency performance over a given development corpus [46, 143, 9, 128]. But as we will see in Section 6.3, a constant value for each beam-width parameter is necessarily set conservatively to ensure that outlier sentences — sentences where the prioritization function is less accurate

– are not pruned too severely. If we instead adapt these beam-search parameters to each sentence, or even each chart cell, we can significantly improve the efficiency of context-free inference without compromising accuracy.

We combine cell closure prediction and beam-width prediction into a single model where a beam-width of zero indicates that the cell is closed, otherwise the population of the cell is restricted by the predicted beam-width. As with finite-state chart constraints, high-precision classification for cell closure, as well as a measured bias towards over-predicting the beam-width is desirable so that context-free inference is not overly constrained to the point of accuracy degradation. To accomplish this, we introduce an asymmetric loss function during optimization to penalize under-predicting the beam-width more severely than over-predicting.

Another key feature of our approach is that beam-width prediction does not rely upon reference syntactic annotations when learning to search. Rather, the model is trained to learn the rank of constituents in the maximum likelihood trees.[1] We will illustrate this by presenting results using a latent-variable grammar, for which there is no "true" reference latent variable parse trees. We simply parse unlabeled sentences from the target domain and train our search models from the output of these trees, with no prior knowledge of the non-terminal set or other grammar characteristics to guide the process. Hence, this approach is broadly applicable to a wide range of scenarios, including tuning the search to new domains where domain mismatch may yield very different efficiency/accuracy operating points.

6.2 Open/Closed Cell Classification

6.2.1 Constituent Closure

We first look at the binary classification of chart cells as either open or closed to full constituents, and predict this value from lexical and syntactic features of the input sentence.

[1]Note that we do not call this method "unsupervised" because all grammars used in this paper are induced from supervised data, although our framework can also accommodate unsupervised grammars. We emphasize that we are learning to search using only maximum likelihood trees, not that we are doing unsupervised parsing.

This is the same problem we addressed in Chapter 5 with chart-constraints; however, where there we classify lexical items as either beginning or ending a constituent, here we classify individual chart cells as open or closed, an approach we call *Constituent Closure*. Although the number of classifications scales quadratically with our approach, the total parse time is still dominated by the $O(N^3|G|)$ parsing complexity and we find that the added level of specificity reduces the search space significantly.

To learn to classify a chart cell spanning words $w_{i+1} \dots w_j$ of a sentence S as open or closed to full constituents, we first map cells in the training corpus to tuples:

$$\Phi(S, i, j) = (\boldsymbol{x}, y) \tag{6.1}$$

where $\boldsymbol{x}$ is a feature-vector representation of the chart cell and y is the target class 1 if the cell contains an edge from the maximum likelihood parse tree, 0 otherwise. The feature vector $\boldsymbol{x}$ is encoded with the chart cell's absolute and relative span width, as well as unigram and bigram lexical and part-of-speech tag items from a four-word window (one word position to the left to two words positions to the right). POS features are computed using the 1-best path from an order-2 hidden Markov model. The complete set of features is listed in Table 6.1.

Given feature/target tuples $(\boldsymbol{x}, y)$ for every chart cell in every sentence of a training corpus τ, we train a weight vector θ using the averaged perceptron algorithm [49] to learn a binary decision boundary to classify each chart cell as open or closed:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{(\boldsymbol{x},y)\in\Phi(\tau)} L_\lambda(H(\theta \cdot \boldsymbol{x}), y) \tag{6.2}$$

where $H(\cdot)$ is the unit step function: 1 if the inner product $\theta \cdot \boldsymbol{x} > 0$, and 0 otherwise; and $L_\lambda(\cdot, \cdot)$ is an asymmetric loss function, defined below.

When predicting cell closure, all misclassifications are not equal. If we leave open a cell which contains no edges in the maximum likelihood (ML) parse, we incur the cost of additional processing, but are still able to recover the ML parse tree. However, if we close a chart cell which contains an edge from the ML Tree, we will no longer be able to recover the target parse due to search errors. To deal with this imbalance, we introduce an

Feature	Description
t_{i-1}	One left of outside left boundary POS tag
t_i	Outside left boundary POS tag
t_{i+1}	Inside left boundary POS tag
t_{i+2}	One right of inside left boundary POS tag
$t_i\ t_{i+1}$	Joint left boundary POS tags
t_{j-1}	One left of inside right boundary POS tag
t_j	Inside right boundary POS tag
t_{j+1}	Outside right boundary POS tag
t_{j+2}	One right of outside right boundary POS tag
$t_j\ t_{j+1}$	Joint right boundary POS tags
w_i	Outside left boundary lexical entry
w_{i+1}	Inside left boundary lexical entry
$w_i\ w_i + 1$	Joint left boundary lexical entries
w_j	Inside right boundary lexical entry
w_{j+1}	Outside right boundary lexical entry
$w_j\ w_{j+1}$	Joint right boundary lexical entries
$\Phi(j-i)$	Absolute span-width bins of 2, 3, 4, 5, 10, 20, 30, 40, 50
$\Psi((j-i)/n)$	Relative span-width bins of 0.2, 0.4, 0.6, 0.8, 1.0

Table 6.1: Features used for classification of chart cell spanning $w_i \ldots w_j$. Entries t and w are POS tags and words, respectively; Φ and Ψ bins are triggered accumulatively, meaning a span-width of 7 has a binary feature of one for span-width bins 2,3,4, and 5 and zero for all others.

asymmetric loss function $L_\lambda(\cdot,\cdot)$ to penalize false-negatives more severely during training.

$$L_\lambda(h, y) = \begin{cases} 0 & \text{if } h = y \\ 1 & \text{if } h > y \\ \lambda & \text{if } h < y \end{cases} \tag{6.3}$$

We found the value $\lambda = 10^2$ to give the best performance on our development set, and we use this value in all of our experiments.

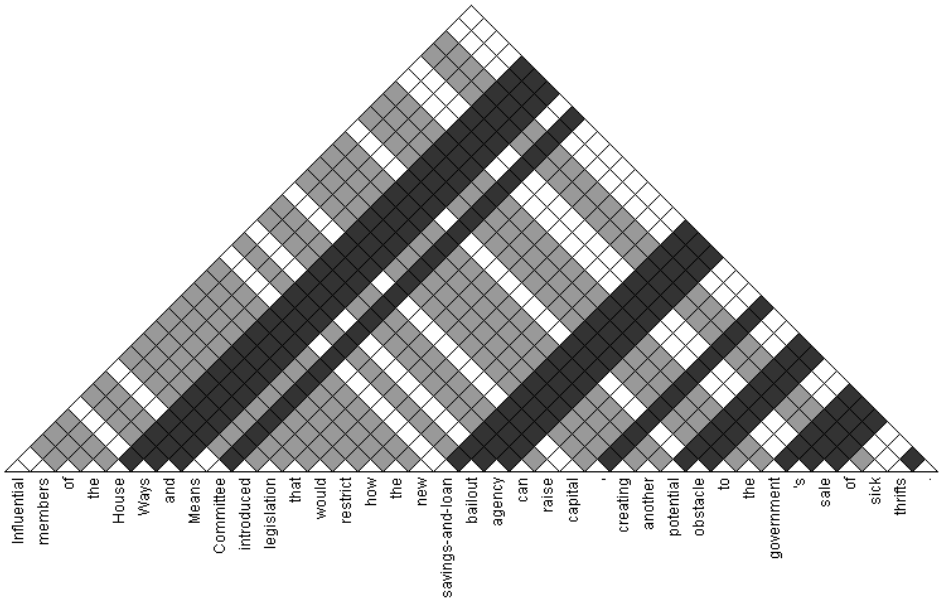
Figures 6.2a and 6.2b compare the pruned charts of Chart Constraints and Constituent

Closure for a single sentence in the development set. Note that both of these methods are predicting where a complete constituent may be located in the chart, not partial constituents headed by factored nonterminals within a binarized grammar. Depending on the grammar binarization (right or left) we can infer chart cells that are restricted to only edges with a factored left-hand-side non-terminal. In Figure 6.2 these chart cells are colored gray. We see that constituent closure does indeed close or partially close many of the chart cells highlighted red in the example of Figure 6.1. Unfortunately, due to the right-branching nature of trees in the treebank, many chart cells spanning substrings ending with "thrift" in Figure 6.2b are classified as open, forcing most cells in the chart to be partially closed (only open to factored non-terminals) opposed to fully closed. In Section 6.4 we analyze the effects of constituent closure and chart constraints on downstream parsing speed and accuracy.

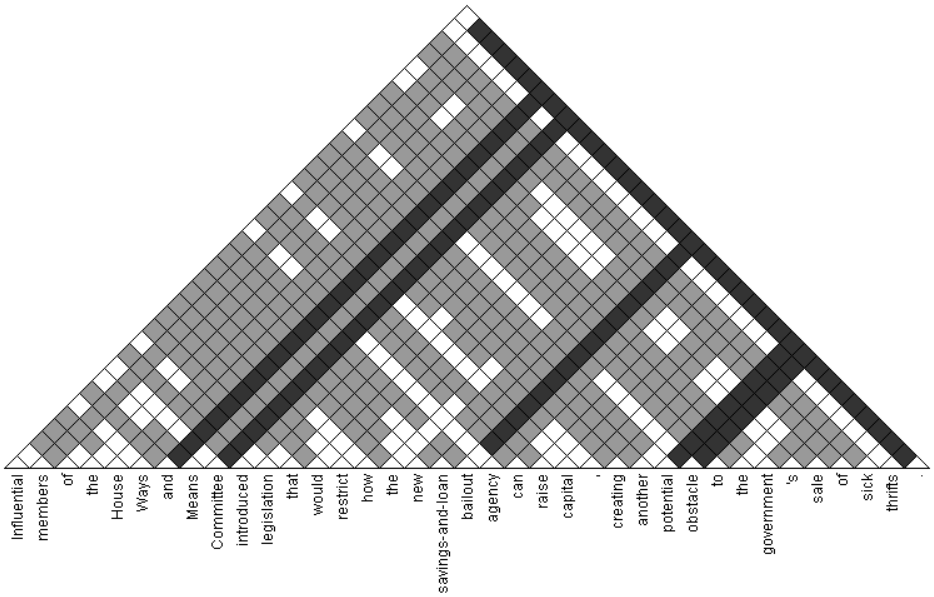
6.2.2 Complete Closure

Alternatively, we can predict whether a chart cell contains any constituent from the binarized target tree, either a partial or a full constituent. We call this approach *Complete Closure*. This is a more difficult classification problem as the constituent boundary context of partial constituents may be significantly different than that of full constituents. Nevertheless, predicting both complete and incomplete constituents directly allows us to close a large number of chart cells, since no additional cells need to be left open to partial constituents. The learning algorithm is identical to Equation 6.2, but training examples are now assigned a positive label if the chart cell contains any edge from the binarized maximum likelihood tree. Figure 6.3 gives a visual representation of complete closure for the same sentence; the number of completely open cells increases somewhat, but the total number of open cells (including those open to factored categories) is greatly reduced.

Table 6.2 records the percentage of open, closed, and partially closed cells over the entire development Section 22 for three chart-closing methods: chart constraints, constituent closure, and complete closure. The precision of each classifier was tuned by adjusting the classification boundary such that downstream parsing accuracy was maintained. For constituent and complete closure, we also tuned the loss function to $\lambda = 10^2$ on heldout data



(a) Finite-State Chart Constraints



(b) Constituent Closure

Figure 6.2: Comparison of Finite-State Chart Constraints (Chapter 5) with Constituent Closure for a single example sentence. Black cells are closed to all edges while grey cells only allow factored edges (incomplete constituents).

(Section 24). Note that 6.7% of all open cells in Table 6.2 are span-1 chart cells which we do not attempt to classify for any of the three methods.[2]

We see that chart constraints has a near-uniform distribution of cell types, with each receiving close to one-third. Constituent closure, although reducing the percentage of open cells from 33.5% to 28.5%, also decreases the total number of closed cells. It is unclear whether this shift to favor partially open chart cells will results in better downstream parsing. Since nearly half of the 1.7 million binary productions in the Berkeley latent variable grammar are headed by factored non-terminals, these "partially open" chart cells are far from empty. Yet, as was shown in Chapter 5, partially open cells only require finding the highest-scoring derivation given a single midpoint, which reduces the work required in these chart cells from $O(N)$ to $O(1)$, where n is the length of the sentence. Thus, the merit of each approach must be measured empirically, which we address in Section 6.4.

Complete closure, on the other hand, is most certainly superior to both chart constraints and constituent closure given the statistics of Table 6.2. The percentage of open cells increases 1.2% absolute from chart constraints, and 6.2% absolute from constituent closure, yet there are no partially open chart cells. A large portion of the cells previously classified as partially open have been completely closed, and because no work is required for closed chart cells, the number of chart cells requiring processing with complete closure is nearly half that of chart constraints and constituent closure.

Constraint Method	**Open**	**Partially Open**	**Closed**
Chart Constraints	33.5%	33.3%	33.2%
Constituent Closure	28.5%	44.5%	27.0%
Complete Closure	34.7%	0.0%	65.3%
Gold	4.5%	2.5%	93.0%

Table 6.2: Percent cells open, cells restricted to only incomplete constituents (partially open), and closed for all of WSJ Section 22 for three methods of cell closing.

[2]In Chapter 5 we consider restricting the population of span-1 unary productions, but this addition is not used for computing the values in Table 6.2.

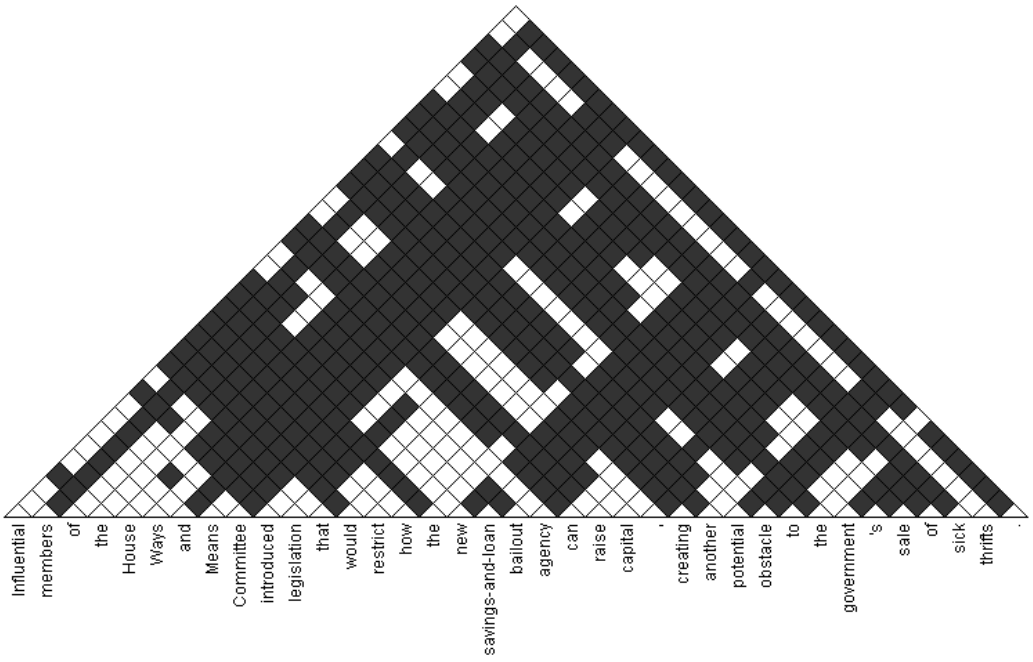


Figure 6.3: Visualization for Complete Closure for a single example sentence. Black cells are closed to all constituents; white cells are open to all constituents.

The three cell-closing methods we have discussed thus far — chart constraints in Chapter 5, constituent closure, and complete closure in the previous sections — each make binary decisions to either open or close portions of the dynamic programming chart. Because constraints are placed on the chart structure itself, these pruning methods can be used to improve the efficiency of any parsing architecture that leverages the same dynamic programming structure. This implies that not only can we speed up exhaustive CYK parsing, but these methods can also be combined with orthogonal pruning methods such as coarse-to-fine pruning, best-first search, and beam search. In the next section we discuss the combination of complete closure with beam-width prediction and show how these methods prune complementary portions of the search space.

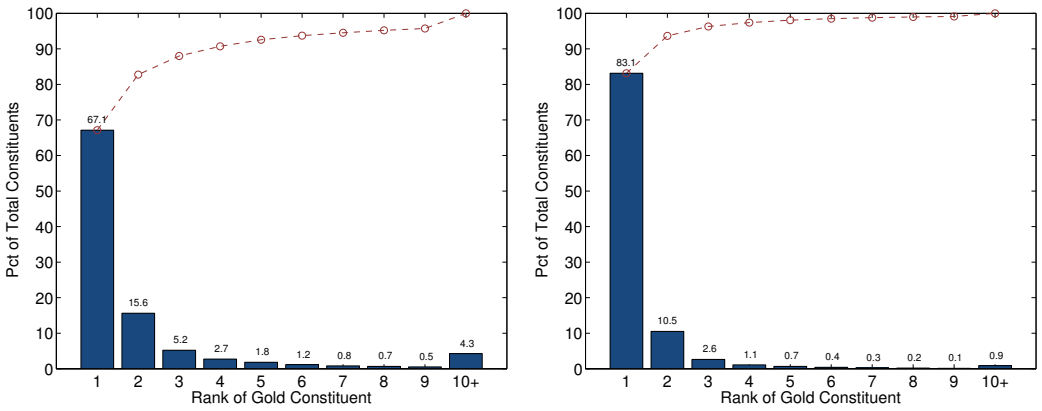
6.3 Beam-Width Prediction

6.3.1 Motivation

When parsing with a beam-search, finding the optimal beam-width threshold(s) to balance speed and accuracy is a necessary step. As mentioned in Section 4.4.1, two variations of the beam-width are often considered: a fixed number of k-best constituents per cell, or a relative probability difference from the highest scoring local edge. We showed in Chapter 4 that both methods perform similarly, and for the remainder of this chapter choose to fix the relative probability threshold for all experiments and focus on adapting the number of allowed constituents per cell. We will refer to this number-of-allowed-constituents value as the beam-width, notated by b.

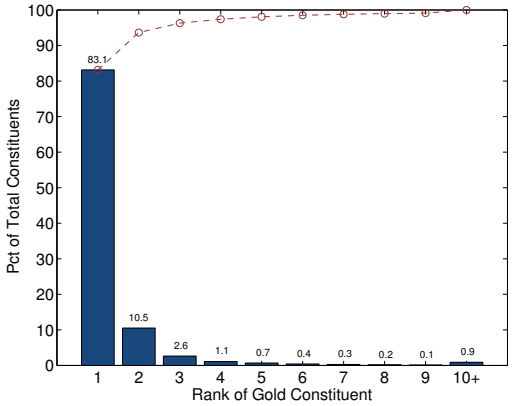
The standard way to tune the beam-width is a grid search over possible values until accuracy on a heldout data set starts to decline. The optimal point will necessarily be very conservative, allowing outliers (sentences or sub-phrases with above average ambiguity) to stay within the beam and produce valid parse trees. Many chart cells may require much fewer than b entries to find the maximum likelihood (ML) edge, yet, constrained by a constant beam-width, the cell will continue to be filled with unfruitful edges, exponentially increasing downstream computation.

To illustrate this point, we turn to Figure 6.4 which plots the rank of the target edge in each chart cell over the entire development set (Section 22). Results in Figure 6.4a are computed using the Constituent POS Boundary prioritization function and R2 grammar, for which the Markovization is a deterministic grammar transform of the gold treebank, and thus we can compute the appropriate chart cell containing each markovized, binarized constituent. But when parsing with i.e., a latent variable grammar estimated via expectation maximization, or the "unlexicalized" grammar of Klein and Manning [102], which is non-trivial to reproduce, no such gold treebank is available with the same non-terminal annotations for judging the utility of the prioritization function as is done in Figure 6.4a. In these situations, we resort to parsing the development set exhaustively to find the maximum likelihood (ML) solution, and then using it as a proxy to the gold tree. Figures 6.4b and 6.4c each plot the rank of the ML constituents in each chart cell



(a) R2 grammar, gold constituents

(b) R2 grammar, ML constituents



(c) Latent grammar, ML constituents

Figure 6.4: Histogram of the rank of target constituents in each span-specific agenda during beam-search parsing. The red line is the cumulative distribution function. No pruning was applied during search (i.e., the beam-width thresholds were infinity)

using the R2 and latent grammar respectively, as well as the Constituent POS Boundary prioritization function. Comparing the ranking of the R2 grammar with the gold and ML trees, we see a bias towards higher rankings with the ML trees. This should be expected because one significant component of the prioritization score of each constituent is the probability of the inside derivation given by the grammar.

Figure 6.4c results are computed using the POS Boundary prioritization function and the Berkeley latent variable grammar, for which we have no true reference parse trees.

Although no gold parse trees exits with latent annotations for which to compare the maximum likelihood results, it is still very useful to compare the rank of the ML constituents with the beam-width tuned to optimize accuracy. For example, a grid search of possible beam-width values in Chapter 4 indicates that we can reduce the global beam-width b to 15 constituents in each cell before accuracy starts to decline. However, from Figure 6.4c we find that 75.3% of the ML edges are ranked *first* in their respective cell and 94.4% are ranked in the top three. Thus, for nearly three-fourths of chart cells visited during parsing Section 22, an additional 14 constituents are unnecessarily added to the chart. Not only does this effect the efficient of the given cell, but also exponentially increases the number of possible constituents that may be built in longer spans. Clearly, it would be advantageous to adapt the beam-width such that it is restrictive when we are confident in the prioritization of constituents in a given cell and more forgiving in ambiguous contexts.

6.3.2 Prediction Model

To address this problem, we learn the optimal beam-width for each chart cell directly. We define $R_{i,j}$ as the rank of the maximum likelihood (ML) edge in the chart cell spanning $w_i \dots w_j$. If no ML edge exists in the cell, then $R_{i,j} = 0$, indicating that the cell should be closed. Given a global maximum beam-width b, we train b different binary classifiers, each using separate mapping functions Φ_k, where the target value y produced by Φ_k is 1 if $R_{i,j} > k$ and 0 otherwise.

The same asymmetry noted in Section 6.2 applies in this task as well. When in doubt, we prefer to over-predict the beam-width and risk an increase in processing time opposed to under-predicting at the expense of accuracy. Thus we use the same loss function L_λ as in Equation 6.2, this time training several classifiers:

$$\hat{\theta}_k = \operatorname*{argmin}_{\theta} \sum_{(\boldsymbol{x},y)\in\Phi_k(\tau)} L_\lambda(H(\theta \cdot \boldsymbol{x}), y) \tag{6.4}$$

Note that in Equation 6.4 when $k = 0$, we recover the open/closed cell classification of Equation 6.2, since a beam width of 0 indicates that the chart cell is completely closed. Hence, this can been seen as a generalization of the chart-closing methods of Chapter 5,

as well as prior work [144]

During inference, we assign the beam-width for chart cell spanning $w_{i+1} \dots w_j$ given models $\theta_0, \theta_1, ...\theta_{b-1}$ by finding the lowest value k such that the binary classifier θ_k classifies $R_{i,j} \leq k$. If no such k exists, $\hat{R}_{i,j}$ is set to the maximum beam-width value b:

$$\hat{R}_{i,j} = \underset{k}{\operatorname{argmin}}\ \theta_k \cdot x_i \leq 0 \tag{6.5}$$

In Equation 6.5 we assume there are b unique classifiers, one for each possible beam-width value between 0 and $b-1$, but this level of granularity is not required. Choosing the number of classification bins to minimize total parsing time is dependent on the prioritization function, the grammar, and properties of the input. Using the POS Boundary prioritization function, the latent variable grammar, and Section 22 of the WSJ corpus, 97.3% of the ML edges have a local rank less than five and we find that the added cost of computing b decision boundaries for each chart cell is not worth the added specificity. Results showed that training four classifiers with beam-width decision boundaries at 0, 1, 2, and 4 is faster than 15 individual classifiers and more memory efficient, since each model θ_k has over 800,000 parameters. All beam-width prediction results reported in this chapter use these settings.

Figure 6.5 is a visual representation of beam-width prediction on a single sentence of the development set using the latent-variable grammar and POS Boundary prioritization function. In this figure, the gray scale represents the relative size of the beam-width: white being the maximum beam-width value, b, shades of gray indicating a beam width of 4, 2, and 1, and black implying the chart cell is closed to all constituents. We can see from this figure that few chart cells are classified as needing the full 15 constituents, apart from span-1 cells which we do not classify.

6.4 Results

We run all experiments on the WSJ treebank [115] using the standard splits: Section 2-21 for training, Section 22 for development, and Section 23 for testing. We preprocess the treebank by removing empty nodes, temporal labels, and spurious unary productions

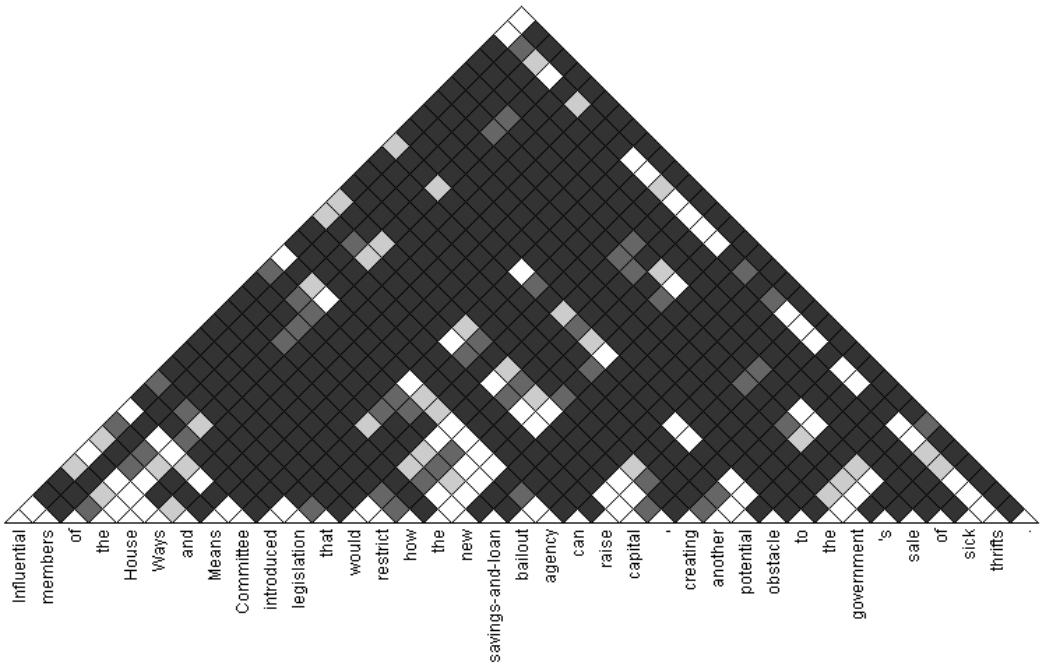


Figure 6.5: Visualization of Beam-Width Prediction for a single example sentence. The grey scale represents the size of the predicted beam-width: black is 0 (cell is skipped) and white is the maximum value b (b=15 in this example).

($X \rightarrow X$), as is standard in published works on syntactic parsing and detailed in Section 2.5.

The pruning methods we present in this chapter can be used to parse with any grammar. To achieve state-of-the-art accuracy levels, we parse with the Berkeley SM6 latent-variable grammar [133] where the original treebank non-terminals are automatically split into subclasses to optimize parsing accuracy. This is an explicit grammar consisting of 4.3 million productions, 2.4 million of which are lexical productions. Exhaustive CYK parsing with the grammar takes more than a minute per sentence.

Accuracy is computed from the 1-best Viterbi (max) tree extracted from the chart. We also ran experiments using the inside score (sum), but results were no better. Alternative decoding methods, such as marginalizing over the latent variables in the grammar or MaxRule decoding [132] are certainly possible in our framework, but it is unknown how

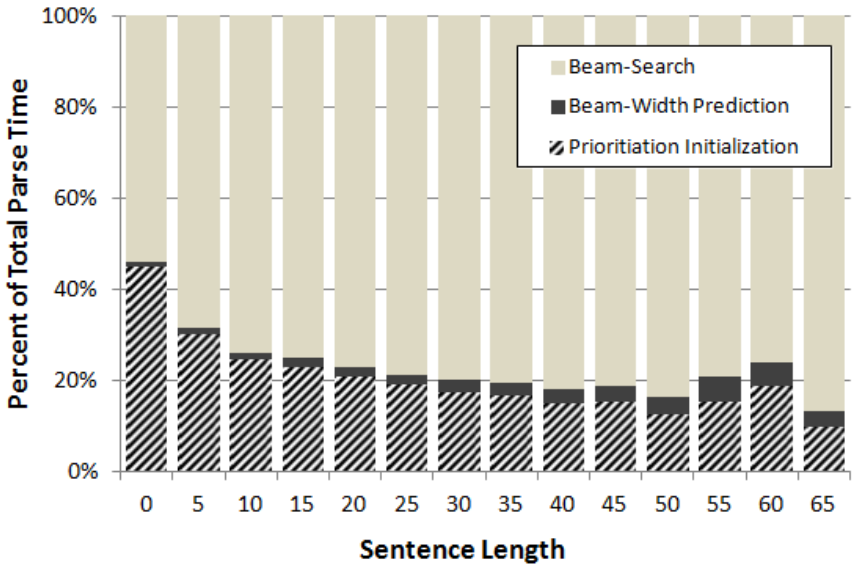


Figure 6.6: Timing breakdown by sentence length for major components of inference.

effective these methods will be given the heavily pruned nature of the chart. We leave investigation of this to future work. We compute the precision and recall of constituents from the 1-best Viterbi trees using the standard EVALB script [46], which ignores punctuation and the root symbol. Accuracy results are reported as F_1, the harmonic mean between precision and recall.

We ran all timing tests on an Intel 3.00GHz processor with 6MB of cache and 16GB of memory. Our parser is written in Java and publicly available at http://code.google.com/p/bubs-parser/. We discuss additional elements of the parser in Section 7.2.

We empirically demonstrate the advantages of our pruning methods by comparing the total parse time of each system, including all stages of the parsing pipeline. The one exception being the parse times reported for Chart Constraints do not include begin, end, or unary constraint tagging times as this was computed offline with a previous setup, but tagging all of Section 22 takes less than three seconds and we choose to ignore this contribution for simplicity.

To understand the processing time required by each component of our parser as a function of sentence length, we turn to Figure 6.6. Here we see a breakdown of the

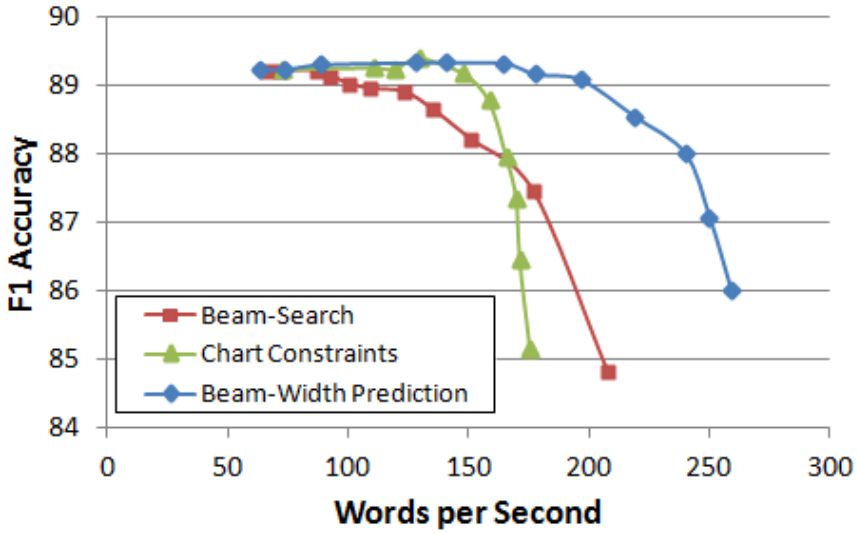


Figure 6.7: Time versus accuracy curves comparing beam-search, beam-search with chart constraints, and beam-search with beam-width prediction.

three components of our final parser: prioritization initialization (including the forward-backward algorithm over ambiguous part-of-speech tags), beam-width prediction, and the final beam-search, including 1-best extraction. We bin these relative times with respect to sentence length to see how each component scales with the number of input words. As expected, the $O(N^3|G|)$ beam-search begins to dominate as the sentence length grows, but the initialization of the prioritization model is not cheap, and absorbs, on average, 20% of the total parse time. Beam-width prediction, on the other hand, is almost negligible in terms of processing time even though it scales quadratically with the length of the sentence.

We compare the accuracy degradation of beam-width prediction and chart constraints in Figure 6.7 as we incrementally tighten their respective pruning parameters. We also include the baseline beam-search parser with POS Boundary prioritization function in this figure to demonstrate the possible accuracy/efficiency trade-off of adjusting a global beam-width alone. In this figure we see that the knee of the beam-width prediction curve (Beam-Width Prediction) extends substantially further to the right before accuracy declines, indicating that our pruning method is intelligently removing a significant portion of the search space that remains unpruned with Chart Constraints. Furthermore, using a

Parser	W/S	F_1
CYK	0.335	89.4
CYK + Constituent Closure	0.493	89.3
CYK + Complete Closure	0.723	89.3
Beam + Inside Prioritization (BI)	5.9	89.2
BI + Constituent Closure	11.6	89.2
BI + Complete Closure	15.0	89.3
BI + Beam-Predict	20.0	89.3
Beam + POS Boundary Priority (BB)	72.4	89.2
BB + Constituent Closure	84.6	89.2
BB + Complete Closure	118.6	89.3
BB + Beam-Predict	165.0	89.3

Table 6.3: Section 22 development set results for CYK and Beam-Search (Beam) parsing using the Berkeley latent-variable grammar.

beam-search alone can achieve efficiency levels similar to chart constraints if some degree of accuracy loss (here, 1-2% absolute) is tolerable.

In Table 6.3 we present the accuracy and parse time for three baseline parsers on the development set using the latent variable grammar: exhaustive CYK parsing, beam-search parsing using the Inside prioritization function, and beam-search using the POS Boundary prioritization function. We then apply our two cell-closing methods, Constituent Closure and Complete Closure, to all three baseline parsers. As expected, the relative speedup of these methods across the various baselines is similar since the open/closed cell classification does not change across parsers. We also see that Complete Closure is between 22% and 31% faster than Constituent Closure, indicating that the greater number of cells closed translates directly into a reduction in parse time. We can further apply beam-width prediction to the two beam-search baseline parsers in Table 6.3. Dynamically adjusting the beam-width for the remaining open cells decreases parse time by an additional 25% when using the Inside prioritization function, and 28% with the POS Boundary prioritization

Parser	W/S	F_1
CYK	0.363	88.7
Berkeley CTF MaxRule	110.1	90.2
Berkeley CTF Viterbi	112.8	88.8
Beam + POS Boundary Priority (BB)	70.2	88.6
BB + Chart Constraints	96.2	88.7
BB + Beam-Predict	187.7	88.7

Table 6.4: Section 23 test set results for multiple parsers using the Berkeley latent-variable grammar.

function.

We apply our best model to the test set and report results in Table 6.4. Beam-width prediction, again, outperforms the baseline of a constant beam-width by a factor of 2.7 and the open/closed classification of chart constraints by nearly doubling the efficiency while maintaining F_1. We also compare beam-width prediction to the Berkeley Coarse-to-Fine parser. Both our parser and the Berkeley parser are written in Java, both are run with Viterbi decoding, and both parse with the same grammar, so a direct comparison of speed and accuracy is fair.[3]

6.5 Conclusion

We have introduced three new pruning methods: (1) constituent closure, (2) complete closure, and (3) beam-width prediction, the best of which (beam-width prediction used in conjunction with beam-search and POS boundary prioritization) unites constituent prioritization metrics used heavily in agenda-based parsing, local pruning from beam-search parsing, and unlabeled constituent structure prediction from chart constraints. Furthermore, we have shown that pruning models trained using only maximum likelihood trees is possible, which allows tuning to specific domains without labeled data. Using

[3]We run the Berkeley parser with the default search parameterization to achieve the fastest possible parsing time. We note that 3 of 2416 sentences fail to parse under these settings. Using the '-accurate' option provides a valid parse for all sentences, but decreases parsing time of Section 23 to 80.1 words per second with no increase in F_1. We assume a back-off strategy for failed parses could be implemented to parse all sentences with a parsing time close to the default parameterization.

this framework, we have shown that parsing time can be reduced by 65% over a standard beam-search without any loss in accuracy, and our framework is significantly faster than both the Berkeley coarse-to-fine parser and a beam-search parser using chart constraints.

Interesting directions for future work include combining beam-width prediction with other forms of efficient search, such as coarse-to-fine pruning. The two methods prune the search space in very different ways and may prove to be complementary. On the other hand, our parser currently spends 20% of the total parse time computing forward-backward scores used as features for cell clarification, and adding additional preprocessing costs, such as parsing with a coarse grammar, may not outweigh the benefits gained in the final search. In the next chapter we investigate the integration of beam-width prediction with chart constraints of Chapter 5, which results in substantial additive efficiency gains over the respective methods in isolation.

Chapter 7

Putting It All Together

7.1 Combination of Efficient Search Methods

In this section we combine optimal models from each of the efficient search methods discussed in this thesis to analyze their potential in additive efficiency gains. Specifically, we apply the POS Boundary prioritization function from Chapter 4, begin, end, and unary chart constraints from Chapter 5, and beam-width prediction from Chapter 6. Recent work by Dunlop et al. [57] to efficiently store and access the grammar in a sparse matrix encoding has also been incorporated, increasing the baseline CYK parsing efficiency by nearly an order of magnitude. This optimization was not applied to the results in Chapters 4 through 6, hence, the following results will not be identical to those previously discussed. We restrict parsing to a single thread and do not parallelize it in any other way so that comparisons with other efficient parsing algorithms can be conducted, but such methods are indeed possible within our framework and prior work has found near optimal latency reductions as the number of cores increases [57]. We discuss other optimizations of our parser, including the sparse matrix encoding of our grammar, in the following section.

Table 7.1 contains parsing accuracy and efficiency results for English, computed on WSJ Section 23. We compare our parser (BUBS) with the Charniak [32] and Petrov and Klein [132] parsers — two well-known state-of-the-art parsers. A direct comparison of accuracy and efficiency with the Charniak parser is not valid as the Charniak parser searches the solution space provided by a different (lexicalized) grammar compared to the latent-variable grammar used by both the Petrov and BUBS parsers; is also is implemented in C++ opposed to Java, further obfuscating relative efficiency gains. Results from the

Charniak parser are only shown here to put the contributions of this thesis in perspective with more well-established research in the NLP community.

The first observation we point out in Table 7.1 is the efficiency difference between a typical CYK implementation that loops over a list of grammar rules during grammar intersection (BUBS CYK) and the matrix-encoded grammar encoding of Dunlop et al. [57] (BUBS Matrix-encoded CYK). We see over a 10x speedup by using this encoding, due primarily to two factors: (1) a reduction in the number of grammar access operations, and (2) more efficient usage of the cache. A detailed analysis of these efficiencies can be found in [56, 57]. Impresivly, this 10x speedup is carried throughout all results in Table 7.1, where we see nearly a 10x speedup of e.g., beam-width prediction coupled with the matrix-encoded grammar, compared to the beam-width prediction results in Chapter 6 which looped over the list of grammar rules. Although the labeled precision and recall are identical for the BUBS CYK and BUBS Matrix-encoded CYK solutions — both of which find the maximum likelihood 1-best Viterbi parse tree — this is not necessarily the case as multiple derivations may have identical likelihood scores and differences in tie-breaking may lead to differences in 1-best solutions between the two parsers.

We apply each of our efficient search methods, in turn, to incrementally improve parsing speed. First, a standard fixed-beam-width beam-search is applied with local prioritization using the POS Boundary function, with a beam-K value of 30 (maximum number of constituents) and a beam-R value of 8 (delta log score from best scoring local candidate). We see a slight degradation (0.3 absolute) in F_1 here, but of course the beam-width can be adjusted to favor accuracy over efficiency if desired. Next, we apply begin-constituent, end-constituent, and unary chart constraints. As observed in Chapter 5, we also see an increase in parsing accuracy due to the elimination of globally poor candidates in the search space. Chart constraints provides a sizable efficiency gain when applied to beam-search parsing, increasing parsing speed from 279 words per second to 753. But when beam-width prediction is used instead of chart-constraints, we see an even larger gain from 279 words per second to 1,131.

Most impressively, when all three methods are combined, parsing efficiency increases to 1,581 words per second. This is a 14x increase over the coarse-to-fine pruned parser of

Petrov and Klein, with identical Viterbi F_1 accuracy. One may be surprised by the additional gains when combining chart constraints with beam-width prediction, as both close portions of the dynamic programming chart, especially since beam-width prediction does so at a finer granularity than chart constraints — closing individual chart cells instead of diagonals. Chart constraints, despite lacking the sensitivity to span-specific classification decisions, does provide three additional modes of pruning that beam-width prediction does not. First, unary productions in span-1 cells can be pruned with high precision, whereas beam-width prediction does not distinguish between lexical and single-word constituent unary productions. Second, because chart constraints close entire diagonals in the chart, it is provable that only a single midpoint needs to be considered for case-2 cells and that the set of productions to consider can be restricted to those with a binarized non-terminal on the right hand side (see Section 5.2.3 for details). In contrast, beam-width prediction iterates over all midpoints where both child cells remain open, and does not distinguish between binarized and non-binarized productions. Finally, the finite-state tagging model used to label each word in the input sentence as in B or $\overline{B}$, E or $\overline{E}$, and U or $\overline{U}$ computes the forward-backward posterior scores over all possible tagging paths in the input. These scores essentially condition the classification of each tag to be globally plausible with respect to the distribution of other tags in the sequence. Beam-width prediction (as well as the constituent and complete closure methods of Chapter 6) classify individual chart cells as open or closed in isolation, with no explicit enforcement of global consistency. Given these differences, it is not surprising that we see additional efficiency when chart constraints and beam-width prediction are combined.

In Figure 7.1 we plot the total parse time per sentence as a function of sentence length for the fastest parser in Table 7.1 (beam-search, POS Boundary prioritization, chart constraints, and beam-width prediction). We roughly see a one-word-per-millisecond ratio of sentence length to parse time as the sentence length increases. Fitting an exponential curve to the data via least squares results in a $N^{1.5}$ trajectory (plotted), which is far

[1]The Charniak parser searches the solution space provided by a different grammar compared to the Berkeley and BUBS parsers; is also is implemented in C++ opposed to Java, so a direct comparison of accuracy or efficiency is not applicable. Results with the Charniak parser are shown here to put the contributions of this thesis in perspective with more well-established research in the NLP community.

	$\mathbf{F_1}$	LP	LR	W/S
Charniak [32]	89.7†	89.7	89.6	50.8†
Petrov and Klein [132] Coarse-to-Fine MaxRule	90.2	90.3	90.0	110.1
Petrov and Klein [132] Coarse-to-Fine Viterbi	88.8	88.9	88.8	112.8
BUBS CYK Viterbi	88.7	88.8	88.5	0.4
BUBS Matrix-encoded CYK Viterbi [57]	88.7	88.8	88.5	5.8
+ beam-search + POS Boundary prioritization	88.4	88.5	88.3	279.2
+ chart constraints	88.6	88.8	88.3	753.3
+ beam-width prediction	88.6	88.7	88.6	1,331.4
+ beam-width prediction + chart constraints	88.8	89.0	88.6	1,581.0

Table 7.1: English test set results (WSJ Section 23) for all efficient parsing methods discussed in this thesis in conjunction with matrix-encoded grammars [57].

superior to the cubic complexity of CYK or even the $N^{2.6}$ and $N^{1.6}$ observed respective complexity of high-precision and quadratic chart constraints discussed in Section 5.5.2.

In Tables 7.2 and 7.3 we apply each of our efficient search methods to the task of parsing Chinese and German. As with English, we see consistent independent and combined efficiency gains with all three methods. Here, we include only the parsing results of Petrov and Klein's coarse-to-fine parser and reiterate that both their parser and the BUBS parser are decoding with identical grammars (i.e., searching for the shortest path within the same implicit hypergraph). Comparing absolute efficiency figures across languages, we note that parsing the German NEGRA corpus [162] is significantly faster than both the English and Chinese corpora. This can be attributed to two facts. First, the average sentence length is the German test set is 17.9 words per sentence compared to 23.0 and 23.5 word per second for the Chinese and English tests sets respectively. Given that our observed complexity is $O(N^{1.5})$, shorter sentences will produce faster efficiency statistics when measured in words per second. Second, the size of the grammar, and consequently the size of the search space, is much smaller for German in our experiments. The German latent variable grammar contains approximately 0.6 million phrase-level productions, while the Chinese

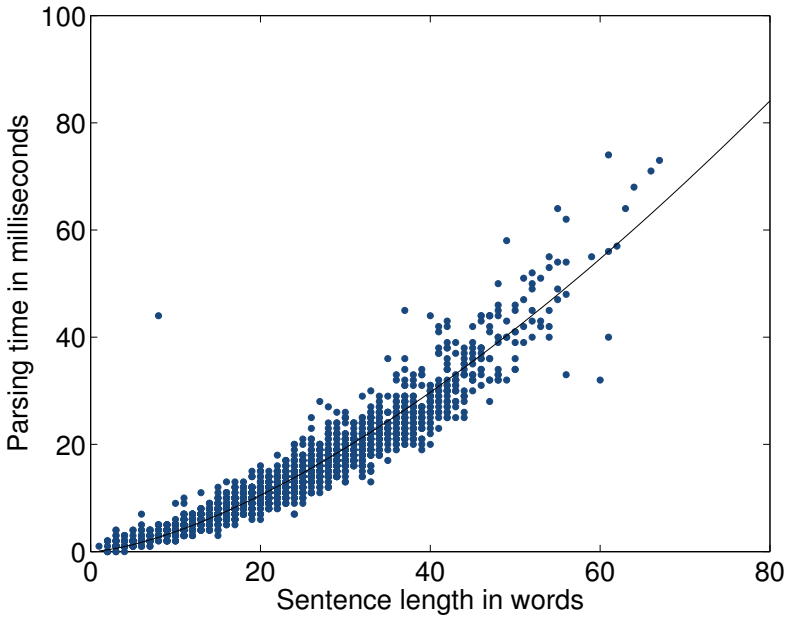


Figure 7.1: Scatter plot of parse time per sentence in milliseconds as sentence length increases for all sentences in WSJ Section 23.

and English latent variable grammars contain 1.1 and 1.8 million productions respectively. As was demonstrated in Chapter 4, the size of the grammar has a direct impact on parsing speed. Consequently, parsing this German test set of shorter sentences with a smaller grammar leads to faster context-free parsing. We also note that we were unable to apply chart constraints to parsing the German corpus for reasons beyond our control, but assume that similar efficiency gains combining all of the efficient search methods we present would be seen on this corpus as well.

Apart from the average words per sentence difference between English, Chinese, and German, there are other notable differences between these languages that would provide accuracy and efficiency differences in parsing performance. We observe in Chapter 5 that the distribution of production types — in particular, unary productions in span-1 chart cells — is significantly different. Sentences from the Chinese treebank contained span-1 unary productions in nearly 40% of span-1 cells; the English treebank contained only 11%. It is difficult to tease apart the exact factors that lead to this difference. Much of it can be attributed to syntactic differences between the languages, where in Chinese a single

	$\mathbf{F_1}$	LP	LR	W/S
Petrov and Klein [132] Coarse-to-Fine MaxRule	83.9	84.5	83.3	56.8
Petrov and Klein [132] Coarse-to-Fine Viterbi	80.7	81.0	80.3	58.2
BUBS CYK Viterbi	79.3	79.5	79.1	0.2
BUBS Matrix-encoded CYK Viterbi [57]	79.3	79.5	79.1	14.2
+ beam-search + POS Boundary prioritization	80.1	80.1	80.1	203.2
+ chart constraints	81.0	82.3	79.8	776.0
+ beam-width prediction	80.0	79.9	80.0	1,124.9
+ beam-width prediction + chart constraints	81.1	82.3	80.0	1,168.9

Table 7.2: Chinese test set results for all efficient parsing methods discussed in this thesis in conjunction with matrix-encoded grammars [57].

token is more likely to represent a stand-alone constituent than in English. Of course, assumptions were made in the tokenization of this treebank and as such the annotation guidelines also play a part, hence the difficulty in separating one factor from the other.

Comparing our optimal configuration with that of the Petrov and Klein parser, we see a 20-fold increase in words-per-second for Chinese over the coarse-to-fine Viterbi decoding baseline, and a 0.4 absolute increase in F_1. When parsing German, we observe a 12-fold increase in parsing efficiency, with a degradation of 1.4 absolute F_1.

All parsing results for the Petrov and Klein parser have been reproduced using the publicity available code at http://code.google.com/p/berkeleyparser/. Accuracy results may differ slightly from previously published works due to discrepancies in treebank pre-processing, which, in turn, leads to differences in labeled precision and recall evaluation. We report our in-house accuracy results here for both parsers to ensure that our evaluation methods are consistent across all algorithms.

7.2 The BUBS Parser

The BUBS parser is a collaborative parsing framework incorporating all of the efficiencies discussed in this thesis, as well as matrix-encoded grammars, low-level parallelization, and

	$\mathbf{F_1}$	LP	LR	W/S
Petrov and Klein [132] Coarse-to-Fine MaxRule	80.1	80.0	80.2	189.2
Petrov and Klein [132] Coarse-to-Fine Viterbi	78.4	78.1	78.7	198.7
BUBS CYK Viterbi	77.3	77.0	77.7	3.2
BUBS Matrix-encoded CYK Viterbi [57]	77.3	77.0	77.7	36.8
+ beam-search + POS Boundary prioritization	77.0	76.3	77.4	751.6
+ beam-width prediction	77.2	77.0	77.3	2,423.6

Table 7.3: German test set results for all efficient parsing methods discussed in this thesis in conjunction with matrix-encoded grammars [57]. Chart constraints were not applied.

other research conducted by the Natural Language Processing group at Oregon Health and Science University. The parser is available to the public at http://code.google.com/p/bubs-parser/ under the GNU Affero General Public License.

Recent work by Dunlop et al. [57] has shown many advantageous properties of encoding a PCFG in sparse matrix form for constituent parsing, where rows are index by each parent non-terminal and columns are indexed by the tuple of the two child non-terminals. Such a representation allows refactoring the CYK algorithm with two beneficial properties: (1) the number of expensive grammar intersection operations is reduced from $O(N^3)$ to $O(N^2)$; and (2) since grammar intersection is reduced to a set of matrix operations, the resulting algorithm is amenable to fine-grained parallelization. This grammar refactoring method, as well as prior work encoding the grammar as a finite state automaton [99] and prefix compacted tries [122] all demonstrate that model encoding can lead to significant efficiency gains in parsing. No approximations are made with these alternate grammar encodings and efficiency is gained without compromising search.

The BUBS parser contains multiple grammar storage and access methods, including the sparse-matrix encoding discussed above. In Section 7.1 we saw that this grammar encoding strategy alone can provide up to a 10x reduction in parsing time over alternative grammar encoding methods. Coupled with the efficient search techniques discussed in this thesis, the BUBS parser is the fastest high-accuracy constituent parser to date, and an ideal tool for both commercial applications and future research in the syntactic analysis

of natural language.

7.3 Conclusion and Future Work

In this this thesis, we have investigated multiple methods to improve the efficiency of constituent parsing with large context-free grammars. All of the models we have presented are grammar agnostic, requiring no knowledge of the underlying linguistic principals governing either treebank generation or non-terminal annotation. Our methods are completely data driven and we have shown that they are effective across multiple languages and can be trained from maximum likelihood parse trees with minimal loss in F_1 accuracy compared to hand labeled treebanks. Furthermore, each of the models we have presented rely on direct lexical cues from the input sentences to prioritize or prune the search space – information that is not present in unlexicalized grammars and only partially available in bi-lexical grammars conditioned on constituent head words.

In Chapter 4 we discussed models to prioritize constituents at a local level in a beam-search parser, as well as global prioritization within a best-first parser. We introduced the Lexical Boundary prioritization function and showed how sparse data problems estimating lexical-to-non-terminal transition probabilities for large grammars can be alleviated with agglomerative clustering in the lexical space. We also discussed the disadvantages of a best-first parsing architecture, namely the inability to compare constituents spanning different substrings and the prohibitive overhead of sorting a large number of constituents in a global agenda. Beam-search, on the other hand, has the disadvantage of making local pruning decisions that are irreversible, but in empirical trials performed superior to best-first search with multiple grammars and prioritization functions, providing between an 8x and 23x speed improvement over exhaustive parsing depending on the size of the grammar.

In Chapter 5, we extended prior work by Roark and Hollingshead [145] to prune unary productions in span-1 chart cells, providing a cohesive framework for which to prune all spans in the dynamic programming chart under a single finite-state pre-processing

paradigm. Additionally, we presented the first results comparing multi-word and single-word chart constraints to left- and right-binarized grammars. Our results confirmed past work on the efficiency advantages of right-binarized grammars for parsing language with a bias towards right-branching syntactic trees [56, 164], but even more interesting was that the application of chart constraints eliminated these efficiency differences due to pruning portions of the chart that may be locally productive according to the grammar, but globally inconsistent according to the input sentence. Chapter 5 also contained the first analysis of parsing Chinese with chart constraints, as well as the integration of chart constraints with coarse-to-fine pruning, which produced the highest labeled F_1 results to date on the Chinese Penn Treebank. It was shown that constraining multi-word constituents with begin and end constraints increased parsing speeds by a factor of 10x, and the addition of unary constraints increased this to a factor of 20x over unconstrained CYK parsing. When applied to the Charniak, Petrov, and BUBS parsers — all of which are already heavily pruned — we see an addition 2x reduction in latency.

Chapter 6 introduces three separate models: complete and constituent closure, which classify individual chart cells as either open or closed to a set of constituents, and beam-width prediction, which adapts the pruning parameters of beam-search at a cell-by-cell level. It was shown that opening and closing chart cells individually, opposed to allowing word positions to begin or end a constituent as is done in chart constraints, prunes a larger portion of the search space without compromising accuracy. Specifically, the complete closure model was able to completely close 65% of spans compared to the 33% closed by begin and end chart constraints. When we prune the search space with a beam-search and adapt the beam-search parameters with our beam-width prediction model, we see nearly a 500x decrease in parsing time compared to exhaustive CYK parsing with the same grammar.

Results combining each of the methods in Chapters 4 through 6 to the task of parsing English, Chinese, and German were presented in Chapter 7. There we saw large additive efficiency gains for all three languages. With the additional optimization of sparse-matrix encoded grammars, our final efficiency results are over and order of magnitude faster than the coarse-to-fine parser of Petrov and Klein when parsing with the same latent variable

grammar, and maintain — and in the case of Chinese, improve — the labeled F_1 accuracy over the maximum likelihood solution. The combined parsing algorithm has an empirical complexity of $O(N^{1.5})$, providing superior performance for both long and typical-length sentences.

There are many interesting avenues for potential future work. First, both the constituent and complete closure methods of Chapter 6 predict the unlabeled constituent structure of the final parse tree, but these methods make the false independence assumption that each constituent can be predicted in isolation. Future work will explore the possibility of predicting this structure satisfying global conditions that the chart cells open to constituents must be combined to form a valid tree structure. One method of incorporating this information would be to use the output of a dependency parser in a multi-stage pipeline to constrain the structure of the dynamic programming chart. Similar work has recently been done using dependency parsing to constrain decoding with Tree Adjoining Grammars [28] and we believe applying this method to constituent parsing will provide additional efficiency gains.

Second, the size and structure of the grammar is the single largest contributor to parse efficiency. In contrast to the current paradigm of designing a context-free grammar for optimal accuracy and then applying methods to effectively search the space created by the grammar, we plan to investigate new algorithms that jointly optimize accuracy and efficiency during grammar induction, leading to more efficient decoding. Furthermore, labeling a sentence with its constituent structure is rarely an end goal; it is used to provide syntactic information to downstream NLP applications. When the end goal is known, we can optimize the accuracy and efficiency of the grammar directly on the target task, which has the potential for dramatic efficiency gains over optimization of labeled constituent precision and recall on newswire text. For example, consider the latent variable grammar of [133] where groups of non-terminals are split and merged. Under this framework we may need to only refine a small subset of the original treebank non-terminals — and possibly collapse similar functioning treebank tags, such as the standard, comparative, and superlative adjectives (JJ, JJR and JJS) — without reducing accuracy on the end task. Such a reduction in grammar size may lead directly to substantial efficiency gains.

Finally, we suspect the efficiency gains for bottom-up context-free parsing with large grammars may begin to saturate as pruning or prioritization methods improve over those presented in the thesis. In particular, the search space is drastically pruned by beam-width prediction, often leaving only a single binary constituent in each span. Without inducing grammars that are optimized for efficiency, as was previously discussed, we see significant potential in the greedy shift-reduce architectures used for both dependency [181, 124] and constituent [93, 150] parsing, with linear complexity and reported speeds of over 30,000 words per second. Indeed, the space of possible parse trees explodes with the array of non-terminal annotations present in many of the high-accuracy context-free grammars, all of which are removed for evaluation. We believe that the upper bound on efficient parsing with such shift-reduce or state transition parsers may be much higher than that of parsing with large context free grammars, although it has yet to be shown that the accuracy or domain-adaptability of these models can reach the same levels of context-free grammar parsing.

Appendix A

Penn Treebank POS and Phrase-level Tags

POS Tag	Description	POS Tag	Description
CC	Coordinating conjunction	TO	to
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential there	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund/present participle
IN	Preposition/subord conjunction	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd ps. sing. present
JJR	Adjective, comparative	VBZ	Verb, 3rd ps. sing. present
JJS	Adjective, superlative	WDT	wh-determiner
LS	List item marker	WP	wh-pronoun
MD	Modal	WP	Possessive wh-pronoun
NN	Noun, singular or mass	WRB	wh-adverb
NNS	Noun, plural	#	Pound sign
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PP	Possessive pronoun	)	Right bracket character
RB	Adverb	”	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	”	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol	”	Right close double quote

Table A.1: The Penn Treebank POS tagset

Syntactic tag	Description
ADJP	Adjective phrase
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
S	Simple declarative clause
SBAR	Clause introduced by subordinating conjunction or 0
SBARQ	Direct question introduced by wh-word or wh-phrase
SINV	Declarative sentence with subject-aux inversion
SQ	Subconstituent of SBARQ excluding wh-word or wh-phrase
VP	Verb phrase
WHADVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
X	Constituent of unknown or uncertain category

Table A.2: The Penn Treebank syntactic tagset

Null tag	Description
*	"Understood" subject of infinitive or imperative
0	Zero variant of that in subordinate clauses
T	Trace—marks position where moved wh-constituent is interpreted
NIL	Marks position where preposition is interpreted in pied-piping contexts

Table A.3: The Penn Treebank Null tagset

Appendix B

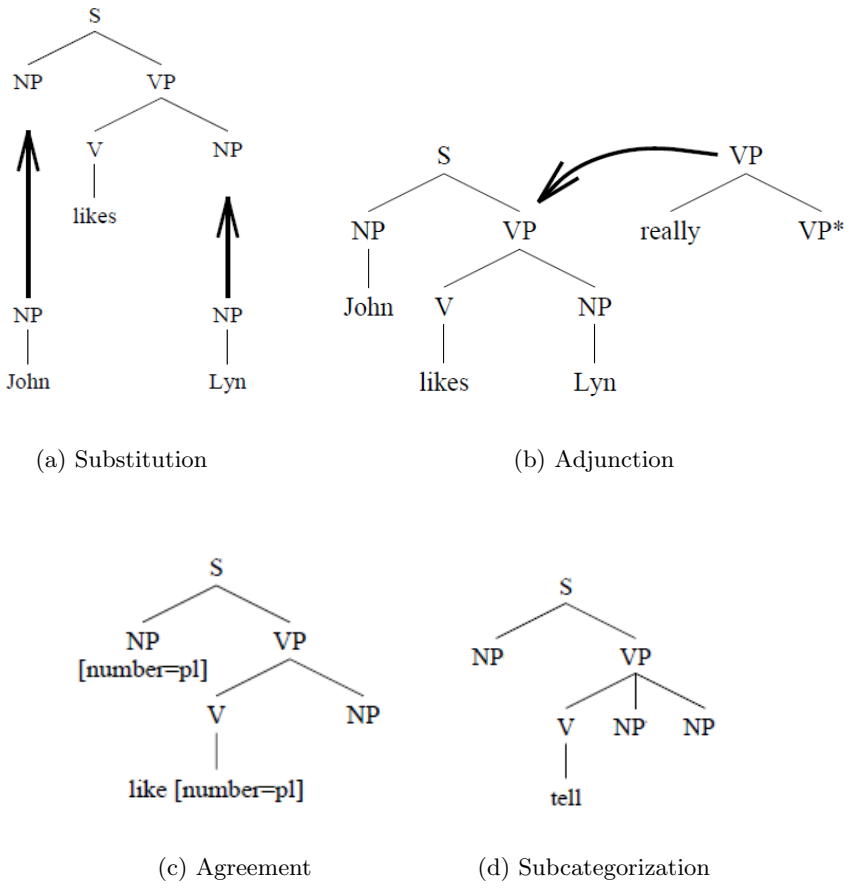
Alternative Grammar Formalisms

B.1 Tree Adjoining Grammar

Tree Adjoining Grammar (TAG) was formalized in 1975 by Joshi et al. [88]. The primary data structure of TAG is the tree, rather than the rule, as is used in context-free grammars, and therefore considered a structure binding formalism opposed to structure changing formalism [90]. There are two types of trees in TAG: *initial trees* representing simple sentential structures, and *auxiliary trees* used to add recursion into the formalism. In addition, two operations are used to combine these tree structure into a final derivation: *substitution* and *adjunction*. An example of two trees joined via substitution can be seen in Figure B.1a, where the parent non-terminal of one tree is substituted for the equivalent child non-terminal in a second tree. The process of adjunction can be seen in Figure B.1b, and joins the two trees into a new derivation for the sentence "John really likes Lyn".

The adjunction operation allows TAG to become mildly context-sensitive; opposed to context-free rules $A \rightarrow \beta$, TAG supports auxiliary trees that specify a right and/or left context: $xAy \rightarrow x\beta y$. Furthermore, agrement can be enforced through the specification of feature-rich initial trees (see Figure B.1c) and subcategorization can be implemented via trees anchored in a single lexical item and containing appropriate frames. For example, in Figure B.1d we see that the word "tell" requires a subject, indirect object, and direct object noun phrases before participating in a full derivation.

When elementary trees are associated with lexical items, as in the example above, the grammar formalisms is referred to as Lexicalized Tree Adjoining Grammar (LTAG). LTAG parsing naturally proceeds in two steps: first selecting the set of lexical structure associated



(a) Substitution (b) Adjunction

(c) Agreement (d) Subcategorization

Figure B.1: Examples of substitution and adjunction operations, and agreement and subcategorization structure for Tree Adjoining Grammar. Figure adapted from [139].

with the input, and second, applying standard polynomial parsing algorithms resolve the remaining ambiguity [156]. CYK dynamic programming and Early-style top-down filtering approaches can both been used for LTAG parsing [156], but a straightforward implementation is $O(N^6)$, where N is the length of the sentence. Harbusch introduces an $O(N^4 \log N)$ approach by first parsing with a projected context-free grammar to constrain subsequent TAG operations, while still performing exact inference [75]. More recently, supertagging has become a widely used pre-processing step to decrease the typical-case runtime of LTAG parsing [7, 155]. Given the surrounding context of each lexical item, supertagging limits the ambiguity of elementary trees by pruning those that are unlikely

to participate in a full derivation. This approach is similar to pre-tagging the input with POS tags (1-best or n-best) to constrain the downstream parser, but operates on the larger set of possible elementary trees for each lexical item instead of the set of possible POS tags.

B.2 Combinatorial Categorial Grammar

Combinatorial categorial grammar (CCG) is a proof-theoretic formalism that encodes possible grammatical derivations as attributes of the lexicon. Binary rules such as *application*, *composition*, and *substitution* determine how these lexical categories combine to form a parse tree. For example, one form of the transitive verb "loves" is represented in the lexicon as

$$\frac{loves}{(S\backslash NP)/NP}$$

indicating that a sentence (S) can be formed if one noun phrase (NP) is found to the left of the word, and one to the right (the direction of the slash represents the direction of the required argument). Provided with noun phrases to the left and right, we join lexical items into phrases through iterative functional evaluation:

$$\dfrac{\dfrac{John}{NP} \quad \dfrac{\dfrac{loves}{(S\backslash NP)/NP} \quad \dfrac{Mary}{NP}}{S\backslash NP}\,>}{S}\,<$$

In [173], Vijay-Shanker and Weir proved that CCGs generate exactly the same class of strings as Tree Adjoining Grammars, and are therefore weakly equivalent. Both formalisms are mildly context-sensitive, with the ability to represent cross-serial dependencies, distinguishing them from context-free grammars. But in [63], Fowler and Penn show that a number of CCGs used in practice, including that of Clark and Curran [42] are actually strongly context-free. Nevertheless, all CCGs – including those outside the definition of

context-free – are parsable in polynomial time, and due to the binary nature of derivation rules, easily amenable to chart-based parsing techniques such as the CYK algorithm.

As with TAG, the efficiency of CCG parsing has recently been improved with an adapted version of supertagging [40, 41], a preprocessing step to limit the amount of ambiguity in lexical items given the context of the sentence.[1] In 2011, Zhang and Clark showed that recent efficiency benefits from dependency parsing via a shift-reduce framework could also be adapted to CCG parsing with measurable success [186]. Although accuracy parsing standard corpora with CCGs has not yet attained the level of lexicalized or latent-annotated context-free grammars [63], it remains a viable and interesting grammar formalism for computational linguists.

B.3 Lexical Functional Grammar

Lexical functional grammars (LFG) are an extension of unification theory, developed by Bresnan and Kaplan in the 1970s [94, 20]. LFG structure is composed of two independent syntactic components, each with its own rules: *feature structure*, which represents the organizational structure of the sentence, including subject-object relations and predicate-argument structure; and *constituent structure*, the hierarchical structuring of words into phrases [51]. This decomposition into feature and constituent structure differs from transformational grammar formalisms by focusing on the structure and function of the language independently instead of one producing the other. LFG also differentiates itself from Chomskyan grammar traditions by encoding phenomena, such as passivization, in the lexicon rather than treating it as a transformation [44].

The constituent structure of LFG has the form of context-free phrase structure trees, while feature structures are sets of attribute/value pairs. Attributes may be features, such as tense and gender, or functions, such as subject and object. The constituent and feature structure can be encoded simultaneously in a PCFG where $\uparrow$ and $\downarrow$ are commonly used to denote the feature structure of the parent/child relationship. For example, in Figure B.2,

[1] Supertagging was originally proposed for lexicalized tree adjoining grammars by Bangalore and Joshi in [7], and adapted to CCGs by Clark and Curran.

↑ SUBJ =↓ indicates that the subject of the parent (S) is the child.

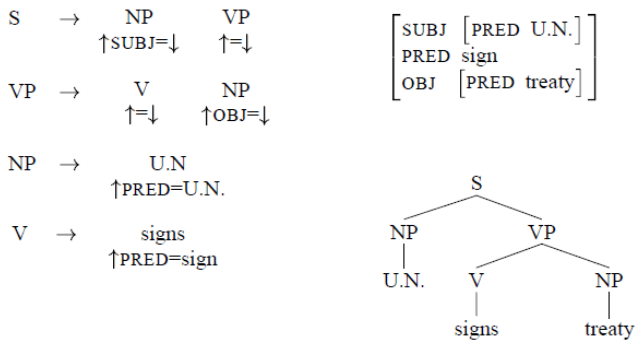


Figure B.2: Parsing "U.N. sign treaty" with the LFG above produces both the feature structure and constituent structure pictured. Figure adapted from [92].

The constituent and feature structure of the grammar provide complementary constraints. Parsing with such a grammar requires satisfying both sets of constraints, but can be implemented in two distinct steps. For example, one may first find all possible constituent structures via the CYK chart-parsing algorithm, and then resolve the feature structure constraints given this reduced set. Resolving feature structure constraints is an exponential operation, but efficiencies such as packed feature representations [117], dynamic programming [66], and lazy evaluation [19] have been presented to decrease the run-time in practice.

LFG can encode long-distance dependencies through reentrancies in its feature structure, and unlike TAG and CCG, can also represent cross-serial dependencies, such as those found in Dutch [20]. This representative power makes LFG a fully context-sensitive formalism.

B.4 Head-driven Phrase Structure Grammar

Head-driven Phrase Structure Grammar (HPSG) is another unification-based formalisms that has recently been adopted by computational linguists [135]. The phrasal head, is considered the most important element in the phrase and other elements inherit its properties during unification. For example, the verb is often considered the head of a sentence

as it drives much of the grammatical structure. HPSG builds on the previous work of Generalized Phrase Structure Grammar [65] and is, in turn, a non-derivational formalism, meaning that one form of structure or representation can not be induced from (or implied by) another. Instead, HPSG imposes a set of constraints on the input sentence which must be satisfied through unification, similar to CCG.

As with LFG, HPSG extends unification-based grammars by enforcing typed feature-structure such that features and values assigned to a word are appropriate given its syntactic function [90]. For example, the GENDER attribute may not be assigned the nonsensical value THREE. But unlike LFG (and TAG), HPSG incorporates the phrase-structure information directly into the feature terms.

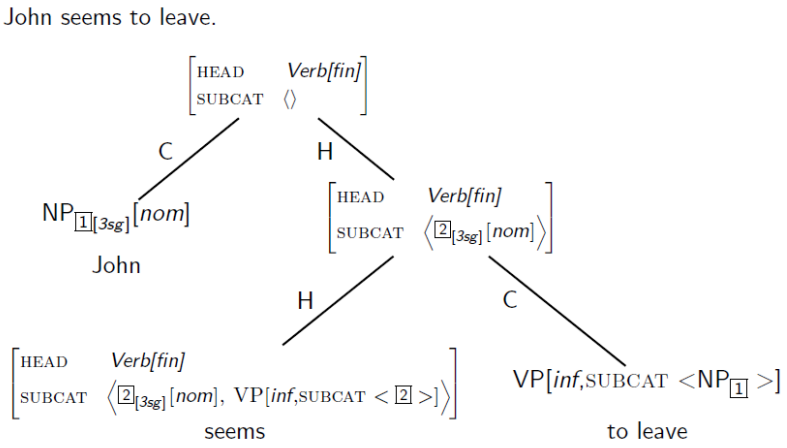


Figure B.3: HPSG tree structure for the sentence "John seems to leave". Figure adapted from [91].

An example of an HPSG tree structure can be see in Figure B.3. The HEAD attribute represents the syntactic features of the head word (similar to a POS tag), while SUBCAT encodes the subcategorization information. Boxed numbers are used for feature sharing (i.e., pointers) and the H and C labels on arcs indicate which structure is the head, and which are the complement(s).

Similar to LFG, HPSG is fully context-sensitive and requires exponential time to parse. To increase the efficiency of HPSG parsing, Kasper et al. proposed a compilation algorithm to convert HPSG to LTAG. The precompilation process identifies substructure of

the HPSG that is not context-dependent. This substructure can be parsed with standard chart-based algorithms and used as a backbone to the remaining context-sensitive disambiguation [96, 8]. Similar methods, such as CFG filtering, have been proposed converting HPSG to CFG to constrain the search space of possible HPSG unifications [168]. Note that all of these methods still perform exact HPSG unification – only ungrammatical structure is eliminated during pruning.

B.5 Dependency Grammar

Dependency parsing is a form of syntactic analysis that labels directed relationships between two words in a sentence (bi-lexical relationships). The head of the dependency arc determines the syntactic category of the child; the closer a word is to the root of the tree, the more syntactically important it is. Figure B.4 depicts the dependency structure for the example sentence where the word "overreacted" is the main verb of the sentence and drives the subcategorization structure. The bi-lexical arcs in Figure B.4 can optionally be labeled with a syntactic function, such as "noun modifier" or "verb's subject", but unlabeled dependency parsing is also a well-studied task as the dependency arcs alone provide significant syntactic disambiguation. Furthermore, given the unlabeled dependency arcs and POS tags, labeling the dependency graph with syntactic relationships is straight-forward and can be done with high accuracy. This is done as a post-processing step in many dependency parsers, such as McDonald [120].

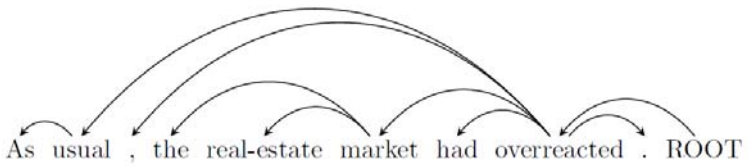


Figure B.4: Unlabeled dependency grammar structure for an example sentence.

Dependency parsing has recently received significant attention, most notably due to the efficient decoding algorithms of Yamada and Matsumoto [181], Nivre [124], and McDonald [120] and their utility within NLP applications. Nivre [124]'s word investigated the impact

of various constraints on the space of possible dependency grams – requiring each word to have a single head or that the graph be acyclic – and how these constraints affected coverage and the number of candidate edges. His results found that constraining the degree of allowed non-projectivity can greatly reduce the number of arcs that must be considered during search, and, as long as some degree of non-projectivity is allowed, coverage is minimally impacted. In the work of McDonald et al. [121] it was shown the total absence of projectivity constraints allows for the use of spanning tree algorithms that can be quadratic complexity for certain classes of statistical models, so the ultimate utility of such constraints varies depending on the model being used.

Dependency parsing has been shown to increase the accuracy of machine translation by transferring lexical relationships from the source language to the target language, especially for translation between English and Subject-Object-Verb languages such as Japanese and Turkish [97, 64]. Research has also been done using dependency structure to constrain higher-complexity parsing within a pipeline, such as with the HPSG and TAG formalisms [152, 28]. With these recent successes to NLP applications and the development of $O(N)$ decoding algorithms, dependency parsing is an attractive alternative to context-free parsing as it can be very efficient – similar to speeds of finite-state tagging algorithms. It has been shown that dependency structure extracted from context-free parses with PCFGs produces superior accuracy and adaptability to new domains, compared to the $O(N)$ or $O(N^2)$ algorithms of Nivre [124] and McDonald [120], but it remains an open question as to how such a difference in accuracy or adaptability affects downstream NLP applications leveraging such syntactic information.

Appendix C

Coarse-to-Fine Parsing Algorithm

In Algorithm 5 we present pseudocode for two-level coarse-to-fine parsing. Given a pre-defined mapping of fine-to-coarse non-terminals Φ, we first compute the inside/outside scores of each constituent over every span given the coarse grammar.[1] We assume these values are normalized by the probability of the sentence. Next, we proceed exactly as done in the CYK algorithm – iteration over each nonterminal in the target grammar at every span – with the exception of skipping non-terminals that, when projected to the coarse grammar space, have an inside/outside score less than some set threshold λ. The variable λ can be tuned on held-out data for optimal accuracy and/or efficiency performance. The presentation of coarse-to-fine in Algorithm 5 is an approximate inference algorithm as there is no guarantee that the projected grammar scores are an upper bound on the true target-grammar derivation. Although the inside/outside scores perform well in practice, we may inadvertently prune the globally optimal derivation during parsing.

[1] For a presentation of the inside/outside algorithm, see e.g. Manning and Schütze (1999) pg. 400 [112].

Algorithm 5 CoarseToFineParser($w_1 \dots w_n,\ G_{coarse},\ G_{fine} = (V, T, S^{\dagger}, P, \rho),\ \Phi,\ \lambda$)

Input:

$w_1 \dots w_n$: Input sentence

G_{coarse}: Binarized Coarse PCFG

G_{fine}: Binarized Target PCFG

Φ: Maps non-terminals from V_{fine} to V_{coarse}

λ: Coarse-To-Fine pruning threshold; $0 \leq \lambda < 1$

Output:

α: Viterbi-max scores for all non-terminals over every span

$\gamma^{coarse} \leftarrow$ InsideOutside($w_1 \dots w_n,\ G_{coarse}$)
for $s = 1$ to n **do** ▷ Span width: bottom-up traversal
  for $b = 1$ to $n - s + 1$ **do** ▷ Begin word position
    $e \leftarrow b + s - 1$ ▷ End word position
    for $A_i \in V_{fine}$ **do**
      if $\gamma^{coarse}_{\Phi(i)}(b, e) > \lambda$ **then**
        if $s = 1$ **then** ▷ Special case for lexical productions
          $\alpha_i(b, e) \leftarrow \rho(A_i \rightarrow w_b)$
        else
          $\alpha_i(b, e) \leftarrow \max_{b \leq m < e} \left(\max_{j,k} \ \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, m)\ \alpha_k(m + 1, e) \right)$
    for $A_i \in V$ **do** ▷ Add unary productions
      if $\gamma^{coarse}_{\Phi(i)}(b, e) > \lambda$ **then**
        $\upsilon_i(b, e) \leftarrow \max \left(\alpha_i(b, e)\ ,\ \max_{j} \ \rho(A_i \rightarrow A_j)\ \alpha_j(b, e) \right)$
    $\alpha(b, e) \leftarrow \upsilon(b, e)$
return α

Bibliography

[1] S. Abney, S. Flickenger, C. Gdaniec, C. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. Procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the workshop on Speech and Natural Language*, HLT '91, pages 306–311, Association for Computational Linguistics, Stroudsburg, PA, USA.

[2] J. Allen. 1995. *Natural language understanding.* Benjamin/Cummings series in computer science. Benjamin/Cummings Pub. Co.

[3] Hiyan Alshawi, editor. 1992. *The Core Language Engine.* MIT Press, Cambridge, Massachusetts.

[4] M. D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte. 1986. Min-max heaps and generalized priority queues. *Commun. ACM*, 29:996–1000.

[5] John W. Backus. 1959. The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. In *IFIP Congress*, pages 125–131.

[6] James K. Baker. 1979. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550.

[7] Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25:237–265.

[8] Tilman Becker and Patrice Lopez. 2000. Adapting hpsg-to-tag compilation to wide-coverage grammars. In *In Proc. of TAG+5*, pages 47–54.

[9] Steven Bird. 2002. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics.*

[10] Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *In Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 31–37.

[11] Ezra Black, John Lafferty, and Salira Roukos. 1992. Development and evaluation of a broad-coverage probabilistic grammar of english-language computer manuals. In *In Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 185–192.

[12] Don Blaheta and Eugene Charniak. 1999. Automatic compensation for parser figure-of-merit flaws. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 513–518, Association for Computational Linguistics, Stroudsburg, PA, USA.

[13] Robert J. Bobrow. 1990. Statistical agenda parsing. In *DARPA Speech and Language Workshop*, pages 222–224.

[14] Rens Bod. 2003. An efficient implementation of a new dop model. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 19–26, Association for Computational Linguistics, Stroudsburg, PA, USA.

[15] Nathan Bodenstab. 2006. Coarse-to-fine efficient viterbi parsing. Research Proficiency Exam, Oregon Health and Science University.

[16] Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. 2010. Exponential decay pruning for bottom-up beam-search parsing. In *Pacific Northwest Regional NLP Workshop (NW-NLP 2010)*.

[17] Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. 2011. Adaptive beam-width prediction for efficient context-free parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Portland, Oregon.

[18] Nathan Bodenstab, Kristy Hollingshead, and Brian Roark. 2011. Unary constraints for efficient context-free parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Portland, Oregon.

[19] Pierre Boullier and Benoît Sagot. 2005. Efficient and robust lfg parsing: Sxlfg. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 1–10, Association for Computational Linguistics, Stroudsburg, PA, USA.

[20] Joan Bresnan. 2000. *Lexical-Functional Syntax*. Blackwell Textbooks in Linguistics. Blackwell Publishers.

[21] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Comput. Linguist.*, 18:467–479.

[22] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19:263–311.

[23] David Burkett, John Blitzer, and Dan Klein. 2010. 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*, HLT '10, pages 127–135, Association for Computational Linguistics, Stroudsburg, PA, USA.

[24] Richard Campbell. 2004. Using linguistic principles to recover empty categories. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 645–652, Barcelona, Spain.

[25] Sharon A Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24:275–298.

[26] Bob Carpenter. 2009. Atkinson, sack, santoro, and strothotte (1986) min-max heaps and generalized priority queues. cacm. http://lingpipe-blog.com/2009/02/27/atkinson-sack-santoro-and-strothotte-1986-min-max-heaps-and-generalized-priority-queues-cacm/.

[27] Rollo Carpenter. 1981. http://en.wikipedia.org/wiki/Jabberwacky.

[28] Xavier Carreras, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pages 9–16, Association for Computational Linguistics, Stroudsburg, PA, USA.

[29] Yin-Wen Chang and Michael Collins. 2011. Exact Decoding of Phrase-Based Translation Models through Lagrangian Relaxation. In *Proceedings of the 2011 Conference*

on Empirical Methods in Natural Language Processing, pages 26–37, Association for Computational Linguistics, Edinburgh, Scotland, UK.

[30] E. Charniak. 1993. *Statistical Language Learning.* MIT Press, Cambridge MA.

[31] Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, AAAI Press/MIT Press, Menlo Park, CA.

[32] Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139, Morgan Kaufmann Publishers Inc., Seattle, Washington.

[33] Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *In proceedings of the sixth workshop on very large corpora*, pages 127–133, Morgan Kaufmann.

[34] Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Ann Arbor, MI.

[35] Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, Association for Computational Linguistics, Stroudsburg, PA, USA.

[36] Zhiyi Chi and Stuart Geman. 1998. Estimation of probabilistic context-free grammars. *Comput. Linguist.*, 24:299–305.

[37] David Chiang. 2010. Learning to translate with source and target syntax. In *Proceedings of the 48rd Annual Meeting on Association for Computational Linguistics*, pages 1443–1452, Association for Computational Linguistics.

[38] Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

[39] Noam Chomsky. 1959. On certain formal properties of grammars. *Information and Control*, 2(2):137–167.

[40] Stephen Clark. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the International Workshop on Tree Adjoining Grammars*, pages 19–24, Venice, Italy.

[41] Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL)*, pages 103–110.

[42] Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33.

[43] John Cocke. 1969. *Programming languages and their compilers: Preliminary notes.* Courant Institute of Mathematical Sciences, New York University.

[44] Ronald A. Cole, In Chief, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, Giovanni Varile, Antonio Zampolli (eds.), Antonio Zampolli, Ron Cole, Victor Zue, and Victor Zue. 1995. Survey of the state of the art in human language technology.

[45] Michael Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 184–191, Association for Computational Linguistics, Santa Cruz, California.

[46] Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, Madrid, Spain.

[47] Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing.* PhD dissertation, University of Pennsylvania.

[48] Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, Stanford, CA.

[49] Michael Collins. 2002. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical Methods in Natural Language Processing*, volume 10, pages 1–8, Association for Computational Linguistics, Philadelphia.

[50] Christopher Culy. 1985. The complexity of the vocabulary of bambara. *Linguistics and Philosophy*, 8:345–351. 10.1007/BF00630918.

[51] M. Dalrymple. 2001. *Lexical functional grammar.* Syntax and semantics. Academic Press.

[52] John DeNero, Adam Pauls, and Dan Klein. 2009. Asynchronous binarization for synchronous grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 141–144, Association for Computational Linguistics, Suntec, Singapore.

[53] Neeraj Deshmukh, Aravind Ganapathiraju, and Joseph Picone. 1999. Hierarchical search for large vocabulary conversational speech recognition. *IEEE Signal Processing Magazine*, 16:84–107.

[54] Peter Dienes, P Eter Dienes, and Amit Dubey. 2003. Antecedent recovery: Experiments with a trace tagger. In *In Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 33–40.

[55] Bojan Djordjevic, James R. Curran, and Stephen Clark. 2007. Improving the efficiency of a wide-coverage CCG parser. In *Proceedings of the 10th International Workshop on Parsing Technologies (IWPT) at ACL*, Prague, Czech Republic.

[56] Aaron Dunlop, Nathan Bodenstab, and Brian Roark. 2010. Reducing the grammar constant: an analysis of CYK parsing efficiency. Technical report CSLU-2010-02, OHSU.

[57] Aaron Dunlop, Nathan Bodenstab, and Brian Roark. 2011. Efficient matrix-encoded grammars and low latency parallelization strategies for CYK. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*, pages 163–174.

[58] Jay Earley. 1970. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102.

[59] Abdessamad Echihabi and Daniel Marcu. 2003. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 16–23, Association for Computational Linguistics, Stroudsburg, PA, USA.

[60] Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 457–464, University of Maryland.

[61] Jason Eisner and Giorgio Satta. 2000. A faster parsing algorithm for lexicalized tree-adjoining grammars. In *Proceedings of the 5th Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+5)*, pages 14–19, Paris.

[62] Francois Fleuret and Donald Geman. 2001. Coarse-to-fine face detection. *International Journal of Computer Vision*, 41:85–107.

[63] Timothy A. D. Fowler and Gerald Penn. 2010. Accurate context-free parsing with combinatory categorial grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 335–344, Association for Computational Linguistics, Stroudsburg, PA, USA.

[64] Michel Galley and Christopher D. Manning. 2009. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 773–781, Association for Computational Linguistics, Stroudsburg, PA, USA.

[65] Gerald Gazdar. 1985. *Generalized phrase structure grammar*. Harvard University Press.

[66] Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 279–286.

[67] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 228–235, Association for Computational Linguistics, Stroudsburg, PA, USA.

[68] Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 167–202.

[69] Elliot Glaysher and Dan Moldovan. 2006. Speeding up full syntactic parsing by leveraging partial parsing decisions. In *Proceedings of the COLING/ACL on Main conference poster sessions*, COLING-ACL '06, pages 295–300, Association for Computational Linguistics, Stroudsburg, PA, USA.

[70] Joshua Goodman. 1997. Global thresholding and multiple-pass parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 11–25, Providence, Rhode Island.

[71] Joshua Goodman. 1998. *Parsing Inside-Out*. Ph.D. thesis, Harvard University, Harvard.

[72] Jan Hajič, Martin Čmejrek, Bonnie Dorr, Yuan Ding, Jason Eisner, Daniel Gildea, Terry Koo, Kristen Parton, Gerald Penn, Dragomir Radev, and Owen Rambow. 2004. Natural language generation in the context of machine translation. Technical report, Center for Language and Speech Processing, Johns Hopkins University, Baltimore. Final report from 2002 CLSP summer workshop (87 pages).

[73] Keith Hall. 2004. *Best-first word-lattice parsing: Techniques for integrated syntactic language modeling*. Ph.D. thesis, Brown University.

[74] Keith Hall and Mark Johnson. 2004. Attention shifting for parsing speech. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 40–46, Barcelona, Spain.

[75] Karin Harbusch. 1990. An efficient parsing algorithm for tree adjoining grammars. In *IN PROC. OF THE 28TH ACL*, pages 284–291.

[76] James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Association for Computational Linguistics, Stroudsburg, PA, USA.

[77] U. Hermjakob and R. J. Mooney. 1997. Learning Parse and Translation Decisions from Examples with Rich Context. 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*, ACL, Madrid, Spain.

[78] Donald Hindle. 1994. A parser for text corpora. In *Computational Approaches to the Lexicon*, pages 103–151, Oxford University Press, New York, NY, USA.

[79] Julia Hockenmaier, Aravind K. Joshi, and Ken A. Dill. 2007. Routes are trees: The parsing perspective on protein folding. *Proteins: Structure, Function, and Bioinformatics*, 66(1):1–15.

[80] Kristy Hollingshead. 2010. *Formalizing the use and characteristics of constraints in pipeline systems.* PhD dissertation, Oregon Health and Science University.

[81] Kristy Hollingshead, Seeger Fisher, and Brian Roark. 2005. Comparing and combining finite-state and context-free parsers. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 787–794, Vancouver, BC.

[82] Kristy Hollingshead and Brian Roark. 2007. Pipeline iteration. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 952–959, Prague, Czech Republic.

[83] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[84] Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594, Association for Computational Linguistics, Columbus, Ohio.

[85] Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for English. In *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia.

[86] Mark Johnson. 1998. Pcfg models of linguistic tree representations. *Comput. Linguist.*, 24:613–632.

[87] Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA, USA.

[88] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

[89] Aravind K. Joshi, Vijay K. Shanker, and David J. Weir. 1991. The convergence of mildly context-sensitive grammatical formalisms. In Peter Sells, Stuart Shieber, and Thomas Wasow, editors, *Foundational issues in natural language processing*. MIT Press, Cambridge, MA, pages 31–81.

[90] Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st edition. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[91] Laura Kallmeyer, Timm Lichte, and Wolfgang Maier. 2007. Grammar formalisms: Head-driven phrase structure grammar. University Lecture.

[92] Laura Kallmeyer, Timm Lichte, and Wolfgang Maier. 2007. Grammar formalisms: Lexical functional grammar. University Lecture.

[93] Tom Kalt. 2004. Induction of greedy controllers for deterministic treebank parsers. In *Proceedings of EMNLP 2004*, pages 17–24, Association for Computational Linguistics, Barcelona, Spain.

[94] R. M. Kaplan and J. Bresnan. 1995. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In M. Dalrymple, R. M. Kaplan, J. T. Maxwell, and A. Zaenen, editors, *Formal Issues in Lexical Functional Grammar*. Center for the Study of Language and Information, pages 29–130+.

[95] Tadao Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab, Bedford, MA.

[96] Robert Kasper, Bernd Kiefer, Klaus Netter, and K. Vijay-Shanker. 1995. Compilation of hpsg to tag. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, ACL '95, pages 92–99, Association for Computational Linguistics, Stroudsburg, PA, USA.

[97] Jason Katz-Brown, Slav Petrov, Ryan McDonald, Franz Och, David Talbot, Hiroshi Ichikawa, Masakazu Seno, and Hideto Kazawa. 2011. Training a parser for machine translation reordering. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 183–192, Association for Computational Linguistics, Edinburgh, Scotland, UK.

[98] M Kay. 1986. Algorithm schemata and data structures in syntactic processing. In *Readings in natural language processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pages 35–70.

[99] Dan Klein and Christopher D. Manning. 2001. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the penn treebank. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 338–345, Association for Computational Linguistics, Toulouse, France.

[100] Dan Klein and Christopher D. Manning. 2002. Conditional structure versus conditional estimation in nlp models. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, pages 9–16, Association for Computational Linguistics, Stroudsburg, PA, USA.

[101] Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *In proceedings of the Human Language Technology Conference and the*

North American Association for Computational Linguistics (HLT-NAACL, pages 119–126.

[102] Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, Sapporo, Japan.

[103] Donald E. Knuth. 1998. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

[104] Fred Kochman and Joseph Kupin. 1991. Calculating the probability of a partial parse of a sentence. In *Proceedings of the workshop on Speech and Natural Language*, HLT '91, pages 237–240, Association for Computational Linguistics, Stroudsburg, PA, USA.

[105] Philipp Koehn. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas (AMTA)*.

[106] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: open source toolkit for statistical machine translation. In *Proceedings of the ACL 2007 Demonstrations Session*, pages 177–180, Prague, Czech Republic.

[107] Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Association for Computational Linguistics, Columbus, Ohio.

[108] Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with nonprojective head automata. In *In Proc. of EMNLP*.

[109] Bruce Lowerre. 1990. The harpy speech understanding system. In Alex Waibel and Kai-Fu Lee, editors, *Readings in speech recognition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pages 576–586.

[110] David M. Magerman and Mitchell P. Marcus. 1991. Pearl: A probabilistic chart parser. In *Proceedings of the European ACL Conference*, pages 40–47.

[111] David M. Magerman and Carl Weir. 1992. Efficiency, robustness and accuracy in picky chart parsing. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, ACL '92, pages 40–47, Association for Computational Linguistics, Stroudsburg, PA, USA.

[112] Christopher D. Manning and Hinrich Schuetze. 1999. *Foundations of Statistical Natural Language Processing*, 1 edition. The MIT Press.

[113] Mitchell P. Marcus. 1980. *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, Ma.

[114] Mitchell P. Marcus, M.A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.

[115] Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. 1999. *Treebank-3*. Linguistic Data Consortium, Philadelphia.

[116] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics - ACL '05*, pages 75–82, Ann Arbor, Michigan.

[117] John T. Maxwell, III, Iii Ronald, and Ronald M. Kaplan. 1991. A method for disjunctive constraint satisfaction. In *In Proceedings of the International Workshop on Parsing Technologies*, pages 18–27, Kluwer Academic Publishers.

[118] Andrew McCallum. 2007. Context free grammars. University Lecture.

[119] David McClosky, Eugene Charniak, and Mark Johnson. 2006. Reranking and self-training for parser adaptation. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, Sydney, Australia.

[120] Ryan McDonald. 2006. *Discriminative learning and spanning tree algorithms for dependency parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

[121] Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Flexible text segmentation with structured multilabel classification. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, Vancouver, BC, Canada.

[122] Robert C. Moore. 2004. Improved left-corner chart parsing for large context-free grammars. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New developments in parsing technology*. Kluwer Academic Publishers, Norwell, MA, USA, pages 185–201.

[123] Hermann Ney and Stefan Ortmanns. 1997. Progress in dynamic programming search for lvcsr. In *Proceedings of the IEEE*, pages 88–8.

[124] Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Compututational Linguistics*, 34:513–553.

[125] Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30.

[126] Adam Pauls and Dan Klein. 2010. Hierarchical A* parsing with bridge outside scores. In *Proceedings of the ACL 2010 Conference Short Papers*, ACLShort '10, pages 348–352, Association for Computational Linguistics, Stroudsburg, PA, USA.

[127] Fernando Pereira and Rebecca N. Wright. 1997. Finite-state approximation of phrase-structure grammars. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Devices for Natural Language Processing*. MIT Press, Cambridge, MA, pages 149–174.

[128] Georgios Petasis, Georgios Paliouras, Vangelis Karkaletsis, Constantine Halatsis, and Constantine D. Spyropoulos. 2004. e-grids: Computationally efficient gramatical inference from positive examples. *Grammars*, 7:69–110.

[129] Slav Petrov. 2009. *Coarse-to-Fine Natural Language Processing*. Ph.D. thesis, University of California at Bekeley, Berkeley, CA, USA.

[130] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440, Association for Computational Linguistics, Sydney, Australia.

[131] Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 705–713, Association for Computational Linguistics, Cambridge, MA.

[132] Slav Petrov and Dan Klein. 2007. 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, Association for Computational Linguistics, Rochester, New York.

[133] Slav Petrov and Dan Klein. 2007. Learning and inference for hierarchically split PCFGs. In *AAAI 2007 (Nectar Track)*.

[134] Slav Petrov and Dan Klein. 2008. Discriminative log-linear grammars with latent variables. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1153–1160, MIT Press, Cambridge, MA.

[135] C.J. Pollard and I.A. Sag. 1994. *Head-driven phrase structure grammar*. Studies in contemporary linguistics. Center for the Study of Language and Information.

[136] Detlef Prescher. 2003. tutorial on expectation-maximization algorithm including maximum-likelihood estimation and em training of. In *Probabilistic Context-Free Grammars, 15 th European Summer School in Logic, Language and Information, University of*.

[137] Vasin Punyakanok, Dan Roth, and Wen tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

[138] Lawrence Rabiner and Biing-Hwang Juang. 1993. *Fundamentals of Speech Recognition*, united states ed edition. Prentice Hall.

[139] Owen Rambow. 2008. Tree adjoining grammar: Introduction. University Lecture.

[140] Adwait Ratnaparkhi. 1997. A Linear Observed Time Statistical Parser Based on Maximum Entropy Models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, Brown University, Providence, Rhode Island.

[141] Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.

[142] Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Iii Mark Johnson. 2002. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *IN PROCEEDINGS OF THE 40TH MEETING OF THE ACL*, pages 271–278.

[143] Brian Roark. 2001. *Robust probabilistic predictive syntactic processing: motivations, models, and applications*. Ph.D. thesis, Brown University, Providence, RI, USA. AAI3006783.

[144] Brian Roark, Nathan Bodenstab, and Kristy Hollingshead. 2012. Finite-state chart constraints for reduced complexity context-free parsing pipelines. *Computational Linguistics*.

[145] Brian Roark and Kristy Hollingshead. 2008. Classifying chart cells for quadratic complexity context-free inference. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 745–751, Association for Computational Linguistics, Stroudsburg, PA, USA.

[146] Brian Roark and Kristy Hollingshead. 2009. Linear complexity Context-Free parsing pipelines via chart constraints. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 647–655, Association for Computational Linguistics, Boulder, Colorado.

[147] Alexander M. Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through lagrangian relaxation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 72–82, Association for Computational Linguistics, Portland, Oregon, USA.

[148] Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *In Proc. EMNLP*.

[149] Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik, and Douglas D. Edwards. 1996. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[150] Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 125–132, Association for Computational Linguistics, Stroudsburg, PA, USA.

[151] Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL on Main conference poster sessions*, COLING-ACL '06, pages 691–698, Association for Computational Linguistics, Stroudsburg, PA, USA.

[152] Kenji Sagae, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Hpsg parsing with shallow dependency constraints. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 624–631, Prague, Czech Republic.

[153] Peter Sanders and Dominik Schultes. 2006. Engineering highway hierarchies. In *Proceedings of the 14th conference on Annual European Symposium - Volume 14*, pages 804–816, Springer-Verlag, London, UK.

[154] Anoop Sarkar. 2001. Applying co-training methods to statistical parsing. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, NAACL '01, pages 1–8, Association for Computational Linguistics, Stroudsburg, PA, USA.

[155] Anoop Sarkar. 2007. Combining supertagging and lexicalized tree-adjoining grammar parsing. In Srinivas Bangalore and Aravind Joshi, editors, *Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach*. MIT Press.

[156] Yves Schabes and Aravind K. Joshi. 1991. Parsing with lexicalized Tree Adjoining Grammar. In Masaru Tomita, editor, *Current Issues in Parsing Technologies*. Kluwer Academic Publishers, Norwell, MA, USA, chapter 3, pages 25–47.

[157] Helmut Schmid. 2004. Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proceedings of the 20th international conference on Computational Linguistics*, COLING '04, Association for Computational Linguistics, Stroudsburg, PA, USA.

[158] Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 134–141, Association for Computational Linguistics, Stroudsburg, PA, USA.

[159] Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343. 10.1007/BF00630917.

[160] Stuart M. Shieber, Yves Schabes, and O C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*.

[161] Robert F. Simmons and Yeong Yu. 1990. Training a neural network to be a context sensitive grammar,. Technical report, University of Texas at Austin, Austin, TX, USA.

[162] Wojciech Skut, Thorsten Brants, Brigitte Krenn, and Hans Uszkoreit. 1998. A linguistically interpreted corpus of german newspaper text. In *In Proceedings of the ESSLLI Workshop on Recent Advances in Corpus Annotation*, pages 705–711.

[163] Harold Somers. 1999. Review article: Example-based machine translation. *Machine Translation*, 14:113–157. 10.1023/A:1008109312730.

[164] Xinying Song, Shilin Ding, and Chin-Yew Lin. 2008. Better binarization for the CKY parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 167–176, Association for Computational Linguistics, Stroudsburg, PA, USA.

[165] Yun Tang, Wen-Ju Liu, Hua Zhang, Bo Xu, and Guo-Hong Ding. 2006. One-pass coarse-to-fine segmental speech decoding algorithm. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 1, page I.

[166] Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-Margin Parsing. In *Proceedings of EMNLP 2004*, pages 1–8, Association for Computational Linguistics, Barcelona, Spain.

[167] Ivan Titov and James Henderson. 2006. Loss minimization in parse reranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 560–567, Association for Computational Linguistics, Stroudsburg, PA, USA.

[168] Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun-Ichi Tsujii. 2000. An HPSG parser with CFG filtering. *Natural Language Engineering*, 6:63–80.

[169] Tzong-Han Tsai, Chia-Wei Wu, Yu-Chun Lin, and Wen-Lian Hsu. 2005. Exploiting full parsing information to label semantic roles using an ensemble of ME and SVM via integer linear programming. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, page 233236, Association for Computational Linguistics, Morristown, NJ, USA.

[170] Joseph Turian. 2007. *Constituent parsing by classification.* Ph.D. thesis, New York University, New York, NY, USA. AAI3296885.

[171] Joseph Turian and Dan I. Melamed. 2006. Advances in Discriminative Parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 873–880, Association for Computational Linguistics, Sydney, Australia.

[172] Alan M. Turing. 1950. Computing machinery and intelligence. *Mind*, LIX(236):433–460.

[173] K. Vijay-Shanker and David J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:27–511.

[174] Carl Vogel, Ulrike Hahn, and Holly Branigan. 1996. Cross-serial dependencies are not hard to process. In *Proceedings of the 16th conference on Computational linguistics - Volume 1*, COLING '96, pages 157–162, Association for Computational Linguistics, Stroudsburg, PA, USA.

[175] Richard Wallace. 1995. http://alicebot.blogspot.com/.

[176] Ye yi Wang and Alex Waibel. 1997. Decoding algorithm in statistical machine translation. In *In Proceedings of ACL/EACL'97*, pages 366–372.

[177] Joseph Weizenbaum. 1966. Eliza: a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9:36–45.

[178] W.H. Wong. 1999. *Learning a lightweight robust deterministic parser*. Hong Kong University of Science and Technology.

[179] Devi Xiong, Qun Liu, and Shouxun Lin. 2005. Lexicalized beam thresholding parsing with prior and boundary estimates. *Lecture Notes in Computer Science*, 3406:132141.

[180] Naiwen Xue, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. *Nat. Lang. Eng.*, 11:207–238.

[181] Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *In Proceedings of IWPT*, pages 195–206.

[182] Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189 – 208.

[183] Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 256–263, Association for Computational Linguistics, Stroudsburg, PA, USA.

[184] Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. 2009. K-best combination of syntactic parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, pages 1552–1560, Association for Computational Linguistics, Stroudsburg, PA, USA.

[185] Yue Zhang, Byung gyu Ahn, Stephen Clark, Curt Van Wyk, James R. Curran, and Laura Rimell. 2010. Chart pruning for fast Lexicalised-Grammar parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1472–1479, Beijing, China.

[186] Yue Zhang and Stephen Clark. 2011. Shift-reduce ccg parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 683–692, Association for Computational Linguistics, Stroudsburg, PA, USA.

Biographical Note

Nathan Matthew Bodenstab was born on March 28, 1982 in Medford Oregon. He received his Bachelor of Science degree in Computer Science and Mathematics, with a minor in Philosophy, from George Fox University in 2004, graduating summa cum laude. In 2006, he graduated with a Master of Science degree from Oregon Health and Science University, with an emphasis in Natural Language Processing and Machine Learning. After leaving academia for two years to work as a research scientist for Nuance Communications, he returned to OHSU and received his Ph.D. in Computer Science in 2012. Nathan's professional interests include syntactic parsing, language modeling, and speech recognition. He has co-authored four patents and several articles in peer-reviewed conferences and journals.