© 2016 by Ryan Musa. All rights reserved.

IMPROVING A SUPERVISED CCG PARSER

BY

RYAN MUSA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Associate Professor Julia Hockenmaier

Abstract

The central topic of this thesis is the task of syntactic parsing with Combinatory Categorial Grammar (CCG). We focus on pipeline approaches that have allowed researchers to develop efficient and accurate parsers trained on articles taken from the Wall Street Journal (WSJ). We present three approaches to improving the state-of-the-art in CCG parsing. First, we test novel supertagger-parser combinations to identify the parsing models and algorithms that benefit the most from recent gains in supertagger accuracy. Second, we attempt to lessen the future burdens of assembling a state-of-the-art CCG parsing pipeline by showing that a part-of-speech (POS) tagger is not required to achieve optimal performance. Finally, we discuss the deficiencies of current parsing algorithms and propose a solution that promises improvements in accuracy – particularly for difficult dependencies – while preserving efficiency and optimality guarantees.

To my mom, who taught me to always do my best.

Acknowledgments

Grad school is a journey , and I am truly grateful to the many good people have helped me find my way towards this thesis.

This thesis would not be possible without the financial support from the National Science Foundation (CAREER Grant No. IIS-1053856) and the Defense Advanced Research Projects Agency (Communicating with Computers). I would also like to personally thank Andrew and Shana Laursen for their generous assistance.

I owe my chance to work on the frustrating and rewarding problems of natural language processing to the people I met through a fortunate series of happy accidents in my undergraduate career. Without the time and energy that Dragomir Radev and others have devoted to NACLO, I would not have known to look for research opportunities with Claire Cardie, John Hale, and Effi Georgala; along with Michel Louge, these mentors not only helped me broaden my technical abilities and demonstrated the qualities needed to pursue high-quality research, but also provided advice and continuing support even after I left Cornell.

If I only remember one thing about the computer science department at the University of Illinois, it will be the joy of working with some of the most intelligent, hard-working, and kind people in the world. I especially wish to thank the academic office for the mind-boggling amount of work they put into cultivating such a good environment (in geographic order): Kathy Runck, Candy Foster, Steve Herzog, Lenny Pitt, Holly Bagwell, Heather Zike, Kara MacGregor, Viveka Kudaligama, Mary Beth Kelley (whose extraordinary efforts allowed me to graduate on time), and Rhonda McElroy. Dawn Cheek-Wooten and Elaine Wilson have provided unreasonably reliable support.

I am grateful to have met Yonatan Bisk, Jason Rock, Alice Lai, Chris Cervantes, Joseph DeGol, Daniela Steidl, Brandon Norick, Stephen Mayhew, Daphne Tsatsoulis, Christos Christodoulopoulos, Kent Quanrud, Lucas Waye, Li Guo, and Kevins Shih & Karsch; they are all good friends and colleagues whose talents and technical expertise far exceed my own. Daphne single-handedly carried our cohort through qualifying exams. Yonatan has been an invaluable mentor and friend throughout my tenure at Illinois.

Ken Kochan, Maria Glenn, Jaquelyn Hines, Sam Macy, and Deidre Weathersby helped me develop the habits I needed to finish this thesis.

Too many friends outside of the department have helped make my time at Illinois more enjoyable, but the members of the University of Illinois Women's Hockey Team, Ben Brandsen, Sean Wilner, Lindsey Lanfersieck, Keith Cassidy, and Jon Durney share the bulk of the responsibility for any trouble I might have gotten into. Robert Deloatch has saved my life on multiple occasions and created more fond memories than I can count, and with the hard work of Nicole Brown he finished becoming a very good man. I thank them for sharing their family and wish them all the happiness in the world.

Over the past two years, Ariane Vartanian has demonstrated what it means to be a good friend. This thesis is as much a product of her profound support as it is of any technical achievement.

It would also not exist without the guidance of Julia Hockenmaier. Julia's patience over five years of false starts and setbacks have allowed me to develop as a researcher in ways that would not have been possible under any mortal advisor. I am forever indebted to her for her generous support and gifted technical intuition, and look forward to the opportunity to justify her faith in ther future.

Finally, I wish to thank my family: the extended collection of Musas, Morgans, and Hausers whose loving homes provided relief from tragedies both professional and personal; my sister, Devan, whose energy and enthusiasm are an inspiration; and my parents, Pat and Karlis, whose love, encouragement, and comfort have meant more than they may ever know. Thank you.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CCG	Combinatory Categorial Grammar
NLP	Natural Language Processing
POS	Part-of-Speech
CG	Categorial Grammar
C&C	The original CCG parser of Clark and Curran [1]
Java C&C	The updated C&C parser implemented in Java [47]
NF	Normal Form
CRF	Conditional Random Field
NN	Neural Net
RNN	Recurrent Neural Net
LSTM	Long Short-Term Memory
CYK	The Clark-Younger-Kasami chart-parsing algorithm
A*	A* search algorithm
NN	Neural Net
RNN	Recurrent Neural Net
LSTM	Long Short-Term Memory
F1	The harmonic mean of recall and precision
LF1	F1 evaluated over labeled, directed dependencies

Chapter 1

Introduction

What does it mean to "understand" a language like English? The very act of reading this sentence confirms your ability to understand the meaning of a sequence of written words. You might take that ability for granted, but the cognitive act of forming those twenty words into a complex, abstract idea is quite impressive. Humans excel at this type inference – every day we effortlessly comprehend the meaning of hundreds of sentences that we have never heard before. We can even recognize unfamiliar words and try to learn their meaning from context. [1] To a human, learning to understand language is literally child's play.

By what cognitive processes do we produce and understand language? Answering that question falls under the enterprise of linguistics, the field of science devoted to the study of human language. Researchers in linguistics and its related disciplines have devoted lifetimes to answering that question, and their progress towards understanding how language encodes meaning is the foundation on which the subfield of artificial intelligence known as *natural language processing* (NLP) is built.

NLP is concerned with a slightly different set of questions: foremost, can our ability to understand language be accurately simulated by a computer? Can the quality of the simulation be measured quantitatively, in a meaningful and scientifically reproducible way? What kinds of models are needed to reach an appropriate level of understanding, and is it feasible to learn and apply those models under the constraints of real-world computing resources? For decades, computer scientists have collaborated with linguists, psychologists, statisticians, mathematicians, and electrical and computer engineers in a collective attempt to define and answer those questions. This thesis is a very small contribution to that effort.

1.1 Thesis statement

The central topic of this thesis is the task of syntactic parsing with Combinatory Categorial Grammar (CCG). We focus on how pipeline approaches have allowed researchers to develop efficient and accurate parsers trained on articles taken from the Wall Street Journal (WSJ). We present three approaches to

[1] For instance, the reader might not yet know the meaning of technical terms like *supertagger*, *tri-training*, or *adaptive nanoparticle surface chemistry*, but still deduce that all of those terms are noun phrases that are related to science.

improving the state-of-the-art in CCG parsing. First, we test novel supertagger-parser combinations to identify the parsing models and algorithms that benefit the most from recent gains in supertagger accuracy. Second, we attempt to lessen the future burdens of assembling a state-of-the-art pipeline by showing that part-of-speech (POS) tagging is not required for optimal performance. Finally, we discuss the deficiencies of current parsing algorithms and propose a solution that promises improvements in accuracy – particularly for difficult dependencies – while preserving efficiency and optimality guarantees.

1.2 Contents

Chapter 2: We begin with an introduction to NLP, including an explanation of the role played by pipelines in the development of state-of-the-art systems. We then define the CCG parsing task by briefly summarizing the core elements of CCG and highlighting the main pieces of syntatic analysis provided by a CCG parse. We also introduce the basic components of the CCG parsing pipeline.

Chapter 3: We present the supertagging models that comprise the state-of-the-art over the last ten years, including recent work with bi-directional Long Short-Term Memory (LSTM) models that have dramatically improved performance. We show that these new supertaggers are not only more accurate than the previous state of the art, but also more "certain" in their predictions.

Chapter 4: We describe the algorithms and state-of-the-art statistical models that predict a parse for a supertagged sentence. We show that parsing accuracy can be preserved using tag feature templates based on the Viterbi supertag assignment instead of POS tags, suggesting that POS tagging may not be necessary for a state-of-the-art CCG parsing pipeline. We illustrate how the accuracy and certainty of LSTM supertaggers permit an extremely efficient non-parametric A* parsing algorithm to approach state-of-the-art performance. However, we show that existing parsing models surpass the A* baseline of even the most accurate supertagger, and conclude that additional signal is required to accurately model long-range dependencies.

Chapter 5: We conclude by discussing ideas for improving end-to-end performance by relaxing the constraints of the pipeline to allow joint inference, while limiting the costs to efficiency that existing approaches incur.

Chapter 2

The CCG Parsing Pipeline

2.1 Introduction

Natural language understanding is a challenging research problem. It is difficult to *solve*, in that despite the considerable resources devoted to the effort, there is no general-purpose NLP system that can approach human performance across the full array of linguistic tasks – many of which are so intuitive to us that we take our own facility for granted. It is also difficult to *define*, in that it is difficult for the community to agree on what it would mean to "understand" language in a general setting, or even how to determine when that holistic bar is surpassed.

Instead, the overarching problem can be thought of as the aggregate of the high-level linguistic goals that humans reason about: the ability to understand and answer a question, for instance, or to translate a letter from German to French. Each of these high-level goals can be broken up into a series of connected subtasks, each with a problem definition, resources, and evaluation metrics agreed upon by the interested research subcommunity. This division serves several practical purposes. First, research groups can focus their efforts on a tractable, reasonably well-defined problem and develop approaches to solving it that are informed by their particular expertise. Specialization within the community allows for focused progress and ideological diversity.

Second, "sharing" the pursuit of a task allows researchers (or funding institutions) to curate human-annotated data, which is typically expensive. The specific type of annotation depends on the goals of the task, but the intent is always for the resulting annotated resource to provide a meaningful metric for evaluating a system against a quantitative measure of human ability. By publishing and adhering to consistent standards for experimental conditions (e.g. train/test/development splits), different approaches can be fairly compared against each other and results can (in theory) be replicated and verified by other scientists. It also allows for a reasonable condition that a specific subtask is not "solved" until an automated system can at least match the level of inter-annotator agreement on the subtask's evaluation metric. This empirically-driven definition of scientific progress can make it difficult to compare different theoretical approaches (as one might do when designing an experiment in a field like linguistics), because the variance introduced by engineering and training any sufficiently complex NLP system can significantly obscure the correlation with the true (latent) linguistic processes the system is attempting to simulate.

Finally, treating subtasks independently allows researchers to more easily apply the progress of others towards solving their own task. When treating a problem as a sequence from low-level tasks to more complex high-level tasks, higher performance on earlier tasks can propagate and improve performance on later tasks; this approach is known as pipelining. An NLP pipeline refers to a modular sequence of component systems that each contribute some type of annotation to the original input (e.g. raw text). Components are tailored to a specific subtask, but components in the pipeline have access to the analyses produced by earlier, typically lower-level components; in many cases, the lower-level annotations are incorporated as useful features to improve the accuracy of the higher-level inference, or make inference more efficient by restricting the search space to a smaller feasible region. However, because the pipeline is one-way (results from earlier components are fixed), errors made by earlier components also propagate and may be difficult to recover from. In the ideal pipeline, different systems or components can be swapped in and out of the pipeline, to more easily build on the work of others. At a global planning level, subtasks can be prioritized according to difficulty and research interest; once the performance on an existing subtask reaches a sufficient level, more difficult/complex tasks can become a viable area of research (e.g. the progression of CoNLL shared tasks).

2.2 Goal

This thesis is preoccupied with the task of syntactic parsing with Combinatory Categorial Grammar (CCG). We will focus on how pipeline approaches have allowed researchers to create efficient and accurate parsers for a particular domain: articles from the Wall Street Journal. In the rest of this chapter, we will introduce the tasks that have been identified as useful intermediate goals in producing a final parse analysis. Later chapters will discuss the state-of-the-art systems that have been designed to solve each of these tasks, and analyze the benefits of the pipeline by testing novel combinations of subcomponents. Finally, we will discuss ideas for improving end-to-end performance by relaxing the constraints of the pipeline to allow joint inference, while limiting the costs to efficiency that existing approaches incur.

2.3 Syntactic parsing

The syntactic parsing task is to infer the correct grammatical analysis of a sentence, known as a *parse*. A parse specifies relations between the words in the sentence, though the exact form of the parse will vary depending on the choice of representation and grammar formalism used by the parser. Its purpose is to encode the underlying linguistic structure necessary to correctly deduce what the sentence says about the world (its meaning). As a classic example, compare the sentence *man bites dog* with the sentence *dog bites man*. They both consist of the same three word tokens, and all three words have the same *sense*, or individual semantic meaning, in both sentences. However, taken in their entirety the two sentences describe two very different scenarios with antithetical semantic meaning. Our ability to distinguish the meanings of these two superficially similar sentences is precisely what a *syntactic parser* is trying to simulate.

As humans, we typically never have to *think* about how to analyze sentences in this way – we just do it subconsciously. The syntactic parser is therefore one of the earlier components in the NLP pipeline. As a consequence, the input to the syntactic parser does not come with much extra information – typically POS tags, if that.

One of the main resources for evaluating syntactic parsers is the Wall Street Journal (WSJ) portion of the Penn Treebank [2]. In the 1990s, trained linguists annotated a large corpus of articles with POS tags and constituent parse analyses (first with a skeletal context-free bracketing, and later adding some predicate-argument dependency information). Performance on the parsing task is measured by comparing the predicted parse's agreement with these human annotations, either of the bracketing of constituents or the word-word dependencies. These evaluation metrics can be refined to measure *labeled* agreement, in which the predicted syntactic role of the constituent or dependency must also agree with the annotation to be correct.

2.4 Combinatory Categorial Grammar

Combinatory Categorial Grammar [3, 4] is a formalism developed by Steedman while collaborating with Szabolcsi [5,6] and Ades [7]. It is derived from Categorial Grammar [8,9], which provides the slash notation for compositional syntactic categories along with a mechanism for inferring complex syntactic types by function application. CCG expands the space of functions to include other syntactically-motivated combinators, which we we describe below. Among the formalism's many attractive properties is that a CCG analysis enforces a close relationship between the syntactic structure of a parse and the semantic meaning implied by the analysis's predicate-argument dependency structure [10].

2.4.1 CCG categories

In CCG, a constituent's category is a function that directly encodes significant information about its syntactic role. Rules in the grammar are obtained by applying combinators that determine how two categories combine. The grammar is effectively defined by its lexicon (the set of lexical categories permitted for each word in the language) and the set of allowed combinators.

CCG categories encode a substantial amount of syntactic information relative to the symbols in most other grammar formalisms, and can be used to derive semantic meaning using lambda calculus.

Atomic categories

Atomic categories indicate concrete syntactic roles and do not take any arguments themselves. We use four atomic categories: N (bare noun), NP (noun phrase), PP (prepositional phrase), and S (sentence), each of which represents a constituent of the appropriate type. The categories can be augmented with features that provide additional information, typically for different kinds of sentence categories like S[dcl] (declarative sentence), S[q] (yes-no question), S[wq] (*wh*-question), etc.

Atomic categories (including features, if applicable) form the basic building blocks of the complex functor categories.

Complex categories

Complex categories, or functors, take the form X/Y or X\Y (where X and Y may be either atomic or complex); both represent a function that takes an argument category Y and produces a result category X. The direction of the slash indicates whether the argument Y is absorbed from the right (/) or left (\) of the functor. For example, the determiner category NP/N takes a bare noun argument on the right and produces a noun phrase as a result; an intransitive declarative verb phrase S[dcl]\NP applied to a subject NP on its left results in a declarative sentence S[dcl]. Modifier categories X|X [1] can combine with an argument of type X regardless of the argument's feature, if any (e.g. the adverb category (S\NP)\(S\NP) is allowed to combine with both S[dcl]\NP, S[wq]\NP, etc.), but otherwise there must be feature agreement between functor and argument.

The arity of a complex category is defined as the number of arguments it takes; for example, the arity of the category $((\mathsf{S}\backslash\mathsf{NP}_1)\backslash(\mathsf{S}\backslash\mathsf{NP})_2)/\mathsf{NP}_3$ is three. The index of each argument is indicated by a subscript.

[1] | is shorthand and represents either / or \

2.4.2 CCG combinators

We briefly describe the set of CCG combinators used in our grammar; we recommend [4,11] and especially [3] for a more thorough treatment of the subject.

Application

The basic combinatory schema is function application, in which a functor $\mathsf{X|Y}$ combines with an adjacent argument Y in the appropriate direction to produce a result X. See Figure 2.1 for an example: each step in the derivation consists of either forward or backward application, indicated by $>$ and $<$, respectively.

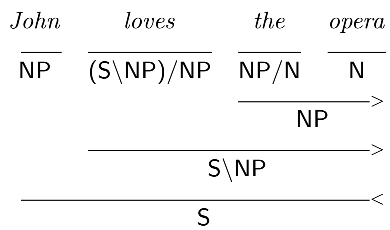

Figure 2.1: The derivation for the sentence *John loves the opera.*

CCG differentiates between head-argument and head-modifier relations; when modifier categories of the form $\mathsf{X|X}$) are applied to an argument X, the head indices of the resulting category will be the same as the modified argument.

Composition

The composition combinator operates on two functors; in the case of forward composition, $\mathsf{X/Y}$ and $\mathsf{Y|Z}$ combine to produce $\mathsf{X|Z}$. In backward composition, $\mathsf{Y|Z}$ and $\mathsf{X\backslash Y}$ combine to produce $\mathsf{X|Z}$. The degree of composition is defined to be the arity of Z. Under this convention, application is equivalent to composition with degree 0. In English, composition is necessary for modeling relative clauses and adverbial modifiers.

Coordination

In the sentence *John and Mary eat pizza*, both people are eating. In order to recover both dependencies, CCG allows conjuncts of the same syntactic type to combine into a single constituent via coordination. The resulting constituent has the same category as the conjuncts and combines their head information.

Type-raising

The unary type-raising schema allows a category Y to type-raise to a new functor $\mathsf{X/(X\backslash Y)}$ (forward type-raising) or $\mathsf{X\backslash(X/Y)}$(backward type-raising). Thus Y, which might have previously served as an argument to $\mathsf{X\backslash Y}$ using backward application, can instead type-raise to $\mathsf{X/(X\backslash Y)}$ and take $\mathsf{X\backslash Y}$ as an argument. Both analyses have the same result category X, which introduces *spurious ambiguity* that can harm the ability of a statistical parser to recover the correct parse. Spurious ambiguities can be prevented by applying normal-form constraints [12, 13]. The grammar may also contain unary type-changing rules, which simply replace category X with Y, leaving head information unchange. The most common of these converts atomic nouns to atomic noun phrases, but others are necessary to generate the WSJ as well.

2.5 Parsing with CCG

In this section, we describe the most popular resource available for the supervised CCG parsing task and introduce the three main elements of a CCG parse.

2.5.1 CCGbank

Just like the Penn Treebank from which it is derived, CCGbank [14,15] is a critical resource for developing, training, and evaluating statistical parsers in a rigorously reproducible setting. The first statistical models for wide-coverage CCG parsing were developed by Hockenmaier and Steedman [10,11] and Clark et al. [16]; the first work defines joint probability distributions factored over derivation structures (parse trees) and sentences, while the latter defines a conditional distribution under which the dependency parse, conditioned on the sentence. Together, they demonstrated that statistical parsers could recover the (unlabeled) word-word dependencies of Wall Street Journal text with accuracy comparable to that of state-of-the-art parsers at the time, such as Collins's lexicalized PCFG models [17–19]. The experiments performed later in the paper use CCGbank exclusively for training (Section 02-21), development (Section 00), and testing (Section 23). As a testament to the enduring popularity of the resource, other CCGbanks have been produced as well [20] and are used to train and evaluate CCG parsers in other languages and domains beyond the English WSJ.

2.5.2 Elements of a CCG parse

A CCG parse for a sentence consists of three main parts. First, every word in the sentence is associate with a pre-terminal (lexical) syntactic category know as a supertag. Supertags are significant because in (C)CG the lexicon encodes most of the syntactic information of a particular language. From these lexical categories the grammar licenses a rule-based parse tree that spans the sentence. Every node in the tree consists of a syntactic category and a set of one or more lexical heads. From this tree one can extract a set of labeled dependencies that indicate the predicate-argument structure encoded in the analysis.

2.5.3 Supertags

The pre-terminal categories in a CCG parse tree are known as *supertags* or *lexical categories*. We show a sample 1-best supertag sequence in Figure 2.2, where the lexical categories of the adverb *ardently* and adjective *classical* reflect their syntactic role as modifiers.

John	*ardently*	*loves*	*classical*	*opera*
NP	(S\NP)/(S\NP)	(S\NP)/NP	N/N	N

Figure 2.2: Example supertags for the sentence *John ardently loves classical opera.*

Supertags can be atomic or complex. Because of the size of the CCG lexicon, predicting the correct supertag for a particular word token may be difficult (and therefore, token- and sentence-based measure of supertagging accuracy correlate with parsing performance). However, successfully identifying the correct tag can greatly reduce the burden on the parsing model.

2.5.4 Derivations

A CCG derivation specifies how combinatory rules are applied to constituents to build a parse tree. Figure 2.3 shows the derivation for the sentence *John ardently loves classical opera.* This derivation indicates that the adverb attaches to the verb before the verb's direct object argument is filled, using composition (>B).

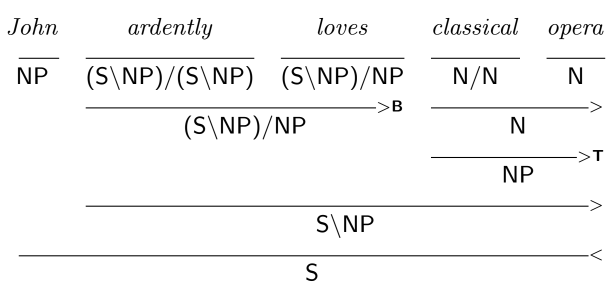

Figure 2.3: Example derivation for the sentence *John ardently loves classical opera.*

2.5.5 Predicate-argument dependencies

In CCGbank, the predicate-argument structure of each derivation is encoded in a set of bilexical (word-word) dependencies [3, 10]. For each head word assigned a complex lexical category in the derivation, the head is traced to its maximal projection (the point in the derivation where the head serves as an argument to another functor). Along this path, dependencies are generated whenever an argument slot in the initial lexical category is filled by combining with another constituent. Each of these dependencies is labeled with the head word (index), head lexical category, and the index of the filled argument slot, along with the head word (index) of the argument constituent. If the argument constituent has multiple heads (e.g. after coordination), a dependency is generated for each.

Dependent	**Head**	**Label**	**Slot**
John	loves	$(\mathsf{S}\backslash\mathsf{NP}_1)/\mathsf{NP}_2$	1
opera	loves	$(\mathsf{S}\backslash\mathsf{NP}_1)/\mathsf{NP}_2$	2
opera	the	NP/N_1	1

Figure 2.4: Labeled predicate-argument dependencies for the sentence *John loves the opera*.

The derivation for *John loves opera* from the previous example (Figure 2.1) produces the dependencies shown in Figure 2.4.

CCG dependency graphs are allowed to have non-projective (i.e. crossing) dependencies. Additionally, words can depend on multiple parents. For example, in the sentence *John loves and Mary hates the opera*, the word *opera* is the direct object of both verbs *love* and *hate*. Figure 2.5 provides the derivation for this sentence, while Figure 2.6 shows the predicate-argument dependencies. Due to coordination, the constituent with category $\mathsf{S}/(\mathsf{S}\backslash\mathsf{NP})$ spanning *John loves and Mary hates* has both *loves* and *hates* as heads, due to coordination. Thus, a CCG dependency graph is not necessarily a tree. Labeling dependencies based on lexical category helps avoid confusion due to spurious ambiguities and type-raising, as in the previous example.

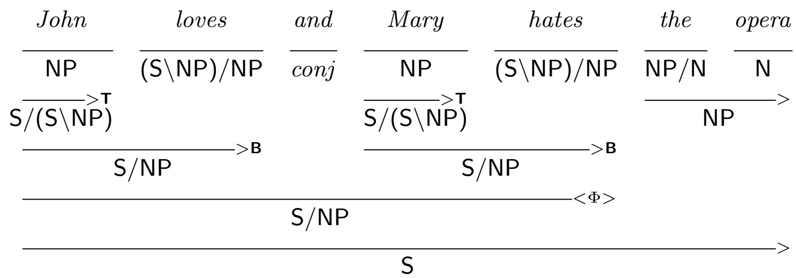

Figure 2.5: The derivation for the sentence *John loves and Mary hates the opera.* Type-raising and coordination facilitate the long-range dependency between *loves* and *opera*.

Dependent	**Head**	**Label**	**Slot**
John	loves	$(S\backslash NP_1)/NP_2$	1
opera	loves	$(S\backslash NP_1)/NP_2$	2
opera	the	NP/N_1	1
Mary	hates	$(S\backslash NP_1)/NP_2$	1
opera	hates	$(S\backslash NP_1)/NP_2$	2

Figure 2.6: Labeled predicate-argument dependencies for the sentence *John loves and Mary hates the opera.*

It is impossible to produce a perfect labeled dependency analysis without correctly predicting each lexical category unless the grammar allows type-changing, so quality of the supertagger will have a large effect on the accuracy of the parser. Wide-coverage grammars are ambiguous by necessity, and a derivational model is required to select the best parse from even a perfect sequence of supertags [2]. As we will see later in this chapter, the expressive power of CCG is such that even a relatively simple heuristic can be paired with a sufficiently accurate supertagger to produce extremely accurate parses. However, while supertag accuracy and derivational metrics defined over the rule-based parse tree are highly correlated with parsing accuracy, the ultimate measure of a syntactic CCG parser is its ability to correctly predict the labeled predicate-argument dependencies for an input sentence, as these will provide the richest representation of the sentence to components further down the NLP pipeline.

[2]For instance, the noun phrase *man next to the child with the blue hat* contains ambiguous prepositional phrase attachment; it is unclear whether the man or the child is the one with the hat, as both analyses are syntactically plausible

2.6 CCG parsing pipeline

Finally, we briefly introduce the three components of the CCG parsing pipeline: POS tagging, supertagging, and parsing.

2.6.1 POS tagging

Traditionally, the CCG pipeline relied on a POS tagger to help recognize unknown or rare words. More recent supertaggers avoid this dependency by relying on *word embeddings* [21, 22] instead of discrete word indicators; word embeddings represent each word as a multi-dimensional vector, and these vectors can be learned by processing large amount of unlabeled text. However, the best log-linear CCG chart parsers still use features defined over POS tags. As supertags themselves encode a significant amount of syntactic information, we should be able to adapt the models by treating the Viterbi supertag assignment [3] as if it were a sequence of automatically assigned POS tags, and generate features based on those lexical categories. We evaluate the effects of this modification in Chapter 4.

2.6.2 Supertagging

The supertagger prunes the space of possible analyses by rejecting improbable (and often incorrect) supertags, improving both the speed and the accuracy of the final parser [23]. Because CCG is a strongly lexicalized grammar formalism where the space of combinatory rules is small relative to the space of possible categories, an assignment of (on average) one or two possible categories to each token with a high level of accuracy significantly reduces the burden on the statistical parsing model. The sequence modeling inherent in the supertagger is reasonably robust to rare or unseen words, which can often be correctly labeled based on their surrounding context.

2.6.3 Parsing

The parser (consisting of a statistical parsing model and/or derivational heuristic) is responsible for finding the best derivation given the output of the supertagger; the best derivation is the highest-scoring analysis that spans the complete sentence. If no such analysis is found, there is a parse failure and no dependencies are predicted. When using adaptive supertagging, the parser may back off to a more permissive beam and try again. However, the productivity of the CCG lexicon is such that the size of the chart may grow too large for the computer's hardware to maintain, especially for long sentences or large beam widths. For this

[3] The most likely sequence of tags according to the supertagger, i.e. the sequence formed by taking the highest-probability tag for each word.

reason, most parsers will abort the attempt to find the spanning analysis if the chart grows larger than a preset maximum number of items. These conflicting constraints illustrate a subtle, not-very-well-understood point: the role of the grammar in pruning the search space, for both good and ill. The best treatment of the subject that we know can be found in [23]. In the WSJ domain especially, treatment of commas (which can indicate appositions, relative clauses, etc.) and coordination over compound noun phrases can have a very large effect on the size of the chart.

2.7 Conclusion

We have outlined the supervised CCG parsing task and motivated our research into the effects of the CCG parsing pipeline. In the next chapter, we will show how approaches to supertagging have evolved and the effect that recent developments in deep neural networks have had on supertagging and multitagging accuracy.

Chapter 3

CCG Supertagging Models

3.1 Introduction

The supertagger plays an important role in the CCG parsing pipeline: because of the compositional nature of CCG, the lexical category of a word fully specifies its syntactic role. The CCG lexicon can allow many different categories for a word, and even with dynamic programming it is computationally intractable for the parser to generate and score all of the possible analyses that are licensed by the grammar using the full lexicon. By predicting a probability distribution over candidate lexical categories, the supertagger provides the parser with a importance means of prioritizing derivations within the full search space.

3.2 Adaptive supertagging

While the efficiency benefits of supertagging are tangible, there is an intrinsic cost to limiting the space of lexical categories considered by the parser: specifically, it has no way to recover the correct analysis if a needed supertag is pruned away. In fact, there is no guarantee that the supertags returned by the supertagger can even form a complete parse. While the accuracy of current state-of-the-art supertaggers is high (approximately 94.5% token accuracy on in-domain text), the one-best tag sequence for a sentence is too brittle to allow for wide-coverage parsing. To avoid parse failures, the C&C parser [1] described in Section 4.3 uses a lexical category for a particular token when searching for spanning analyses if and only if the marginal probability assigned to that supertag is within a fraction β of the probability assigned to the most-likely supertag for that token. In *adaptive supertagging*, β is initially set to a relatively large value ($\beta = 0.075$) that is relaxed until a successful parse is found, the size of the chart exceeds a preset maximum number of items, or β reaches a minimum value (usually 0.001 for chart parsing). In *reverse adaptive supertagging* [23], the parser initially uses the most permissive value for β and tightens the beam until the full chart is sufficiently small. Both of these techniques are variants of *multitagging*, in which the supertagger is allowed to assign multiple lexical categories to a word.

3.3 Statistical approaches to CCG supertagging

The earliest CCG supertaggers were based on maximum-entropy Markov models [24] and conditional random fields (CRFs) [25], but recent work in CCG supertagging has promoted the use of neural nets, which have greatly improved state-of-the-art performance. The first supertagger to use these techniques is the unigram feed-forward tagger developed at the University of Edinburgh [26], which we discuss in Section 3.3.2. Later work demonstrated that a simple A* decoder provided credible (if not quite state-of-the-art) performance on the CCG parsing task using a supertag-factored model with no additional parameters beyond those by the supertagger [27]. Later, recurrent neural net architectures like the Elman neural net [28] used by Xu et al. [29] and the Long Short-Term Memory model [30] used by Vaswani et al. [31] and Lewis et al. [32] improved these results even further by capturing long-range dependencies between supertags and approach state-of-the-art parsing performance with the A* decoder; we provide a brief overview of these approaches in Sections3.3.3 and 3.3.4.

3.3.1 Conditional random fields (CRF)

The C&C supertagger of [1] is a trigram conditional random field (`CRF`) that assigns supertags probabilities conditioned on a five-word context window and the two preceding supertags. It defines features over the words and POS tags in the context window, including bigram and skip-bigram features. For multitagging, the forward-backward algorithm is used to compute the marginal probabilities on which adaptive supertagging can be performed. Due to the $\mathcal{O}(C^3)$ cost of inference in a trigram sequence model where $C \geq 425$ is the number of categories in the wide-coverage lexicon, a tag dictionary is used to restrict the set of considered supertags at each word. If the word was observed at least $k = 20$ times in the training data, only those categories observed with the word are licensed by the tag dictionary; for rare and unknown words, the tag dictionary backs off to the categories observed with word's POS tag (which is typically a large set of categories).

This model relies on POS tags to handle unknown words and is very sensitive to the quality of those tags; this is one reason that its performance degrades aggressively outside of its target domain. Later in this chapter we report supertagging and oracle multitagging accuracies for this model with both gold and automatically-assigned POS tags.

3.3.2 Feed-forward neural nets (NN)

The feed-forward neural net (`NN`) of [27] was the first CCG supertagger to leverage word embeddings; the classifer:

> "uses features of the +/- 3 word context window surrounding a word. The key feature is word embeddings, initialized with the 50-dimensional embeddings trained in [21], and fine-tuned during supervised training. The model also uses 2-character suffixes and capitalization features."

In addition to providing state-of-the-art multi-tagging performance across three domains (annotated CCG data drawn from the Penn Treebank [2,14], Wikipedia [33], and GENIA [34,35]), the supertagger is extremely efficient as all tagging decision are local and no sentence-level inference is needed. It also provides the foundation for later work in efficient A* parsing [27].

The neural net of three layers: the **lookup** layer, which maps the words in the context vector to a feature vector of fixed dimension; the (first) **linear** layer, which multiplies the feature vector by its matrix of weights to produce a real-valued input for each category; and the **soft-max** transfer function, which takes those inputs and outputs a probability distribution over lexical categories for the word in the center of the context window.

While not used in EasyCCG due to overfitting, the supertagger can learn a non-log-linear model by including a second hidden layerwhere the output of the first linear layer is passed through into a hard-tanh transfer function before entering the second linear layer, which then outputs the real-valued inputs to the soft-max output layer. In this case (but not the log-linear case), the number of hidden units in the *first* linear layer can be varied; in both cases, the number of hidden units in the linear layer preceding the soft-max output layer is fixed to the number of lexical categories permitted by the tagger (e.g. 425).

Lookup layer

The lookup layer stores a map between context words and their feature vectors; the feature vectors are real-valued, and the weights are updated during training. When tagging a word token w_t in a sentence using a context window of 7, the current feature vectors for words w_{t-3} through w_{t+3} are concatenated to produce the input for the first linear layer.

In both EasyCCG and our implementation, every word's feature vector contains exactly 60 elements: 50 of these are initialized with the Turian-50 word embeddings [21] available online [1], while the other ten are initialized with mean zero/unit variance Gaussian noise. Of these ten features, five depend on the

[1] `https://github.com/turian/crfchunking-with-wordrepresentations`

capitalization of the word and five depend on the word's 2-character suffix. These features share the same initialization across common words, but are learned separately for each word during training. If different embeddings of dimension D are used, then each word's feature vector will contain $D + 10$ elements.

In addition to the known words in the embeddings vocabulary, the lookup layer stores a randomly-initialized feature vector that it uses when presented with a word that is not in its vocabulary, as well as `START` and `END` markers used when the context window extends past the sentence boundary. With 60 stored parameters for each of the ~270K words in the vocabulary, our implementation maintains ~16.16M parameters for the lookup layer; as noted in [29], most of the performance of the model comes from the lookup tables. We will refer to the parameter matrix for the lookup tables as $\mathbf{U}$.

First linear layer

In the log-linear model using a context window of size C and embeddings of dimension D to predict a set of K categories, the first linear layer consists of K hidden nodes and is parameterized by a $CD \times K$ weight matrix and a $K \times 1$ bias vector. It outputs a $K \times 1$ vector. Thus, in our current Java supertagger using Turian-50 embeddings with a context window of seven words to predict 425 categories, the first linear layer has 178,925 parameters. The parameters in the weight matrix are initialized with zero-mean Gaussian noise; the variance of the distribution is equal to the inverse of the square root of the number of input nodes (e.g. 350 for a seven-word context window where each word requires 50 features for its embedding). We will refer to the parameter matrix for the linear hidden layer as $\mathbf{V}$.

Soft-max output layer

The output layer consists only of a soft-max transfer function and has no parameters. Its main function is to produce a normalized distribution over categories (which can be used for multitagging with a beam). By choosing to maximize the log-likelihood of the correct category for each word, the gradient back-propagated to the preceding linear layer is simple and numerically stable (see Equation 3.1).

Training the NN model

Learning proceeds stochastically over individual tokens in the training data. First, the tagger predicts a probability distribution over tags given its current parameters. Then, weights are updated by propagating gradients of the objective function back through the net. For an example, back-progagation is performed as follows:

Back-propagation: soft-max gradient

Let p_j be the probability assigned to category c_j by the net. The gradient of the objective function LL with respect to n_j, the j^{th} hidden node in the preceding linear layer is:

$$\frac{\partial LL}{\partial n_j} = \mathbf{1}_{c_j=c^*} - p_j \tag{3.1}$$

Back-propagation: linear layer

The update to the bias vector for the linear layer is simply a step towards the gradient calculated by the soft-max layer, scaled by the learning rate. For the other parameters, the update to the j^{th} row of the parameter matrix is a step towards the gradient calculated by the soft-max layer, scaled by both the learning rate and the input n_j (received from the lookup table during classification).

The gradient propagated back to the lookup layer is equal to the transpose of the parameter matrix times the gradient from the soft-max layer.

Back-propagation: lookup layer

The update to the lookup table parameters only affects the feature vectors for the words in the context window. For those words, the update is simply a step towards the gradient calculated by the linear layer, scaled by the learning rate.

Preprocessing and additional features

In addition to the word embeddings, Lewis and Steedman follow [36] and include discrete features for capitalization and the 2-letter suffix of each word; these simple features are expected to generalize to other domains better than more detailed ones would. As pre-processing, all words are lowercased before embedding, and hyphenated words back off to their suffix.

Implementation

The EasyCCG-0.2 distribution [2] depends on Torch [37] [3] for training, which itself runs on LuaJIT [4]. In particular, the process of training a model is handled completely by Torch through a separate command-line routine; once the network weights have been learned, the updated embeddings and other parameters are loaded into memory before parsing.

[2] `https://github.com/mikelewis0/easyccg`
[3] `http://torch.ch/`
[4] `http://luajit.org/`

The Torch library provides many different kinds of layers (nets are constructed by stacking these layers, called "modules", together); however, to emulate the best-performing version of the EasyCCG supertagger, we only needed to re-implement three.

3.3.3 Recurrent neural net (RNN)

Recently, an improvement on the feed-forward neural net was proposed that resulted in a new state-of-the-art [29]. While the feed-forward net uses word embeddings to remove the reliance on unreliable POS tags and sparse, discrete features over words, it does not take anything more than local observed context into account when tagging. Thus, it is not really a surprise that including contextual information beyond the local context window would improve results.

By implementing a Elman recurrent neural net [38] (`RNN`), this supertagger is able to leverage the previously-tagged context by maintaining an unbounded history in an additional hidden state layer. If we let x_t represent the 420-dimensional input vector used for tagging word w_t (exactly as in the original tagger) and h_t represent the hidden layer (again, as in the feed-forward tagger), then the new hidden state layer can be referred to as h_{t-1}.

Using this notation (taken from [29]), the output of the hidden layer for the feed-forward network is:

$$h_t = f(x_t \mathbf{U}) \tag{3.2}$$

where f is the hard-tanh activation function, and the output of the softmax layer is:

$$y_t = g(h_t \mathbf{V}) \tag{3.3}$$

where g is the softmax activation function.

In contrast, the value of the hidden layer h_t in the recurrent net additionally depends on the hidden state h_t:

$$h_t = f(x_t \mathbf{U} + h_{t-1} \mathbf{W}) \tag{3.4}$$

while the output of the softmax layer remains the same (using the new definition of h_t). By recurrently updating the value of the hidden state, contextual history is accumulated as the tagger processes the sentence.

Training

Training is similar to that of the feed-forward net, using a fixed learning rate of 0.0025 and 200 hidden nodes for h_t. They use mini-batched backpropagation through time [39] with a mini-batch size of 9 [5]. Additionally, to prevent overfitting (which they observe at around 20 training epochs) they follow [40] and randomly "drop-out" (mask) input features at a rate of 0.25. The dropout increased their 1-best tag accuracy from 92.63 to 93.07 (the feed-forward net has a 1-best accuracy of 91.10) and improved multi-tagging accuracy as well.

3.3.4 Long short-term memory (LSTM)

While the RNN is more accurate than previous supertaggers, it has difficulty capturing long-range dependencies. To help resolve this issue, Vaswani et al. [31] and Lewis et al. [32] implement bi-directional long short-term memory (LSTM) models that perform well on difficult-to-predict supertags such as those encoding prepositional arguments in verbs, as well as on prepositions themselves. We will differentiate these two models by referring to them as `LSTM-V` and `LSTM-L`, respectively. LSTMs are well-suited to a task like CCG supertagging because they are designed to model both local and long-range context. Additionally, propagating signal in both directions mirrors the way that constituents in Categorial Grammar take arguments either earlier or later in the sentence; see [31] for an extended investigation into the LSTM's ability to predict arguments in different slots based on a functor's internal direction.

Tri-training

In addition to training their supervised LSTM on sections 02-21 of CCGbank, Lewis et al. also train a semi-supervised model by adding additional data generated via *tri-training* [41]. To generate the data, they created another instance of their parser that did not have access to the supertag probabilities (which we refer to as `Dep` in the results in Chapter 4). Using these two parsers and an incremental shift-reduce CCG parser [42], they parsed an additional 43 million words (more than 40 times the size of CCGbank) where all three parsers produced parses with the same sequence of supertags; those supertags were then treated as erzatz gold annotations, and the LSTM model was retrained on this data (we refer to this model as `LSTM-L-Tri`).

[5] I don't know if this is over 9 words or 9 sentences

Supertagging model	Dev.	Test
CRF (Gold POS)	92.7	93.3
CRF (Auto POS)	91.5	92.0
NN	91.1	91.6
RNN	93.1	93.0
LSTM-V	**94.2**	**94.5**
LSTM-L	94.1	94.3
LSTM-L-Tri	**94.9**	**94.7**

Table 3.1: 1-best supertag accuracy on the development (00) and test (23) sections of CCGbank for a selection of supertagging models. The trigram `CRF` supertagger Clark and Curran (2007) uses features based on POS tags (either gold human-annotated tags, or assigned automatically by a POS tagger). The other supertaggers use neural net architectures that do not require POS tagging: `NN` is the feed-forward unigram model of Lewis and Steedman (2014) and `RNN` is the uni-directional recurrent model introduced in Xu et al. (2015). Finally, the `LSTM-V` model of Vaswani et al., (2016) and the `LSTM-L` model Lewis et al. (2016) are both based on stacked bi-directional LSTMs, and use different training procedures. `LSTM-L-Tri` is the latter LSTM model trained on additional unannotated data obtained via `tri-training`.

3.4 Performance

Viterbi supertagging accuracy

The difficulty of predicting a correct supertag out of lexicon of hundreds of categories is well documented; the task has been colorfully described as *almost parsing* [43]. Table 3.1 presents the 1-best supertagging results for each of the supertagging models discussed in this chapter on sections 00 and 23 of CCGbank.

Multitagging accuracy

State-of-the-art supertaggers can now achieve over 94% accuracy on a 1-best per-token basis, but even at that high level of performance the Viterbi supertag sequence is too brittle to support wide-coverage parsing over a domain like the Wall Street Journal. Given the highly lexicalized nature of CCG, even a single supertagging error can seriously harm the parser's ability to recover the correct parse (or even any parse at all). In a true uni-directional pipeline approach to CCG parsing, the parser would not be able to recover from a supertagging error; however, techniques like adaptive supertagging and A* parsing relax that constraint and allow the parser to recover supertags that were rejected by the supertagger (methods for doing so will be discussed in the next chapter).

In this respect, 1-best supertagging accuracy is an informative but slightly flawed measure of the utility of a supertagger to the entire CCG parsing pipeline. To more faithfully simulate the needs of the parser, we can use an oracle score computed over an assignment of multitags. Once the multitagger has assigned a set of tags to each token, the *accuracy* of the assignment can be computed as the percentage of tokens for which the correct supertag is included in the set. One way to measure the *ambiguity* of the assignment is to look at the average number of supertags assigned to each word.

	CRF (G)		**CRF (A)**		**NN**		**RNN**		**LSTM-V**	
β	Amb.	Acc.	Amb.	Acc.	Amb.	Acc.	Amb.	Acc.	Amb.	Acc.
1.0	-	92.7	-	91.5	-	91.1	-	93.1	-	94.2
0.075	1.3	97.3	1.3	96.3	1.3	96.8	1.3	97.3	1.1	97.1
0.030	**1.4**	**97.9**	**1.4**	**97.1**	**1.6**	**97.8**	**1.5**	**98.1**	1.2	97.8
0.010	1.7	98.4	1.7	97.6	2.1	98.5	1.8	98.7	**1.4**	**98.4**
0.005	2.0	98.5	2.0	97.9	2.6	98.8	2.2	99.0	1.6	98.7
0.001	3.0	99.2	3.6	98.7	4.7	99.2	3.9	99.4	2.3	99.2

Table 3.2: Multitagger performance as a function of supertagging beam width β on section 00 of CCGbank. **Amb.** refers to the level of ambiguity (i.e., the number of proposed categories per word) and **Acc.** indicates the per-word multitagging oracle accuracy. Results are presented for the trigram `CRF` supertagger of Clark and Curran using tag dictionaries and features based on gold and automatic POS tags; the baseline feed-forward neural net (`NN`) of Lewis and Steedman that only uses local context; the `RNN`) of Xu et al. that does feed-forward inference over the full sentence; and the bi-directional stacked LSTM model (`LSTM-V`) of Vaswani et al. Results were not available for either the vanilla or tri-trained `LSTM-L` supertaggers, as Lewis et al. do not use adaptive supertagging in their parsing algorithm.)

The multitagging performance metrics presented in Table 3.2 are tailored to the adaptive supertagging regime: once the supertagger has produced a probability distribution over supertags for each token, the multitag assignment is computed by applying a beam of width β; for each token, the beam defines a threshold βp_{vit} (where p_{vit} is the probability of the Viterbi supertag) and supertags assigned probability lower than that threshold are pruned.

The ambiguity levels for each supertagger are organized according to the beam widths β used by the C&C parser when adaptive supertagging was introduced. Subsequent advances in statistical techniques have improve supertagging accuracy to the point that current ambiguity levels of current supertaggers are significantly lower than that of their predecessors. This is a good thing for supertagging – the models are more certain in their predictions – but could cause unintended consequences for chart parsers tuned to the older beam widths.

3.5 Conclusion

In this chapter, we presented five supertagging models and showed that recently introduced LSTM supertaggers represent a quantum leap in performance on this task.

In the next chapter, we will present results showing the performance of these supertaggers when used during adaptive supertagging for both our reimplementation of the C&C normal-form model and the Java C&C parser.

Chapter 4

CCG Parsing Models

4.1 Introduction

In this chapter, we focus on the task of recovering labeled predicate-argument dependencies from a supertagged sentence. We introduce three parsing algorithms – A* search, CYK parsing, and beam decoding – and discuss the properties of each. We then describe the statistical parsing models of [1], and describe a simple technique to remove the models' dependence on POS tags. We show that the accuracy and certainty of the LSTM supertaggers described in the previous chapter permit an extremely efficient non-parametric A* parsing algorithm to approach state-of-the-art performance. Previous work [32] using this algorithm with a tri-trained LSTM supertagger obtained state-of-the-art accuracy, and reported that incorporating a dependency model actually hurt performance. Even using probabilities assigned a weaker supertagger, the best chart parsing model we tested outperforms both the A* baseline and their state-of-the-art system, showing that there are attachment preferences that cannot be recovered by LSTM supertagger alone.

4.2 CCG parsing algorithms

Each of the algorithms described in this section searches for a CCG parse by generating analyses *bottom-up*, beginning with supertagger assigned by the lexicon (or supertagger, in practice). The grammar specifies the rules that combine subderivations to generate new constituents. Becase this process is completely determined by the supertags assigned to words within the constituent's span – a local decision – a wide-coverage grammar will generate intermediate constituents that are inconsistent with the complete parse assigned the highest probability by the parser's statistical model.

This is part of the fundamental tension between accuracy and efficiency in a bottom-up parser: using only local context, it is often difficult to predict how useful a constituent will be in constructing analyses for the entire sentence. While incorporating signal from the supertag probabilities definitely helps, the issue cannot be resolved by the supertagger alone – even a perfect 1-best supertag assignment can produce multiple legitimately-ambiguous parses.

Where possible, we try to highlight how the tradeoffs between efficiency and accuracy motivate each of the following algorithms.

4.2.1 CYK

The Cocke-Younger-Kasami (CYK) algorithm [44, 45] is a polynomial-time [1] dynamic-programming algorithm developed for parsing sentences with context-free grammars. It was first applied to CCG parsing by [46]. The productivity of a wide-coverage CCG lexicon does not allow unconstrained CYK parsing in practice, so the search space must be pruned by techniques like adaptive supertagging, tag dictionaries, or a parsing beam that removes partial analyses based on an approximate figure of merit.

The CYK algorithm operates over a *chart* containing *cells* for every subspan of a sentence. The chart is initialized by adding an item for each supertag to the appropriate cell in the chart. Then, constituents spanning two adjacent words are generated by applying the rules of the grammar to the cross-product of the lexical categories for each word; when two categories are allowed to combine, an item representing the new constituent is created and added to the cell associated with its span. This item maintains *backpointers* to the items that were used to generate it. When a statistical model is used to score the chart, the highest-scoring subderivation for an item can be reconstructed by selecting the item's highest-scoring backpointer and recursively traversing the highest-scoring backpointers of its children. By filling cells of strictly increasing span, dynamic programming ensures that best parse can be recoverd by back-tracking from the root of the chart. However, if lexical categories or internal items have been pruned by any kind of beam, then the best parse is not guaranteed to be optimal.

Dynamic programming is only valid if the scores assigned by the model decompose over items and backpointers, so any non-local features that the model uses to score an item must be maintained as part of the item's equivalence class; thus, more expressive statistical models require more time and memory to process the larger number of items that their features dictate.

4.2.2 Beam decoding

The beam-search decoding algorithm employed in Java C&C [47] also performs inference over a chart; instead of storing the parse forest as a packed chart of constituents and backpointers, though, each cell maintains a set of complete subderivations for its span. The number of subderivations in each cell is limited by the beam size parameter, B.

[1] The algorithm requires $\Theta(n^3|\mathcal{G}|)$ time and memory in its worst case, where n is the length of a sentence and $|\mathcal{G}|$ is a constant based on the size of the grammar.

Cells are populated in the same order as in the CYK algorithm. Because each item is a complete subderivation, during beam decoding the statistical model has access to a much wider range of non-local features than would be feasible with CYK. However, this expressiveness comes at a cost: packed CYK charts to store an exponentially large number of possible parses reasonably efficiently, and dynamic programming allows the model to find the best one in polynomial time. With beam decoding, only the B highest-scoring parses for a span are maintained, and there is no notion of equivalence between items (as it would be expensive to compute). Thus, the beam decoding algorithm makes no guarantees about the optimality of the returned parse, and is heavily dependent on the supertagger's ability to correctly predict the necessary lexical categories; once a constituent is pruned from the chart (i.e., none of the items is rooted in that category) there is no mechanism to recover.

To mitigate the damage caused by pruning away a necessary constituent, beam decoding permits *skimming* [48] dependencies from partial analyses in the event of a parse failure. If no spanning analysis is found, a greedy heuristic is used to find the shortest sequence of non-overlapping subderivations that collectively span the sentence. The dependency structure produced by the parser is the union of the dependency structure for each partial analysis.

While the algorithm's dependence on the supertagger makes it fragile in a pipeline, its ability to do reasonably efficient inference [2] while using non-local context to predict the best parse is something we will continue to discuss.

4.2.3 A* search

The supertag-factored A* parsing algorithm [27] developed by Lewis was introduced using the `NN` neural net supertagger than only uses local context; at the time, its main contribution was efficiency: in constrast to the heavily-optimized C&C parser, the Java implementation of the A* baseline achieved comparable parsing speeds for considerably less engineering effort. However, the accuracy of the resulting parser was well below state-of-the-art at the time. That changed when LSTMs were applied to CCG supertagging and the A* baseline began to surpass many of the systems reported in the literature.

The A* parsing algorithm differs from CYK and beam decoding in that the order in which items are processed is not based on their span; rather, the model maintains an agenda (priority queue) of items that it intends to add to the chart. Items are prioritized by an upper bound on the score of the maximal parse with which their best subderivation is compatible. In the supertag-factored model, this upper bound is obtained

[2] The maximum size of the chart is $\mathcal{O}(Bn^2\mathcal{D}_n)$, where D_n is a constant upper bound on the size of a subderivation of span n, and the worst-case running time of the algorithm is $\Theta(B^2n^2\mathcal{C})$, where $\mathcal{C}$ bounds the time required to combine a pair of derivations and depends on the model's features and the implementation of the chart.

by multiplying the probability of the best subderivation for an item with a simple heuristic: the product of the supertagger probability of the Viterbi supertag for every word outside of the subderivation's span. When an item is popped off of the priority queue and added to the appropriate cell in the chart, the algorithm queries the grammar to see which items in adjacent spans have categories that are allowed to combine with the item's category. The subsequenct combination operation is identical to that performed in CYK, but produced items are pushed onto the agenda rather than added directly to the chart.

This "best-first" approach to parsing is well-suited to CCG parsing for at least two reasons. First, the incremental process by which items are added means that no adaptive supertagging beam is required to limit the size of a chart. When the first full parse is found, it is guaranteed that that parse is the optimal analysis according to the supertagger. Second, the supertag-factored model reflects the intuition that CCG encodes the grammar of a language is in the lexicon.

4.3 The C&C parser

For over a decade, the C&C parser [1, 25, 49, 50] developed by Clark and Curran was the standard against which all other work in supervised statistical CCG parsing for the WSJ was measured. They introduced the modern CCG parsing pipline by using a supertagger [24] to restrict the set of lexical categories considered by their log-linear discriminative parsing models. In addition to parsing, their system provides input to the semantic analysis tool Boxer [51–53].

The C&C parser is based on two statistical models: the *normal-form* model (Section 4.3.2) that defines features over rule-based head relations, and the *dependency* model (Section 4.3.3) intended to explicitly capture predicate-argument structure. The *hybrid dependency* model incorporates both of these, and performs the best. In this work, we reimplement their normal-form parsing model and test further modifications to improve in-domain performance and eliminate the pipeline's dependence on a part-of-speech tagger.

4.3.1 Java C&C

Originally written in optimized C, the C&C parser runs extremely quickly considering the algorithmic complexity of its inference (CYK parsing with adapative supertagging). A more recent version of the system, known as Java C&C, was released in part to facilitate future research. [3]. Algorithmically, Java C&C allows both CYK inference with (reverse-) adaptive supertagging and beam decoding. In addition to reimplementing the normal-form model described in the next section, the system also provides a model

[3] From [47]: "The C&C CCG parser was designed with efficiency in mind, with some cost to the readability and usability of the code base."

Structural tree features:

LexCat + Word	$(S/S)/NP$ + Before
LexCat + POS	$(S/S)/NP$ + IN
RootCat	$S[dcl]$
RootCat + Word	$S[dcl]$ + bought
RootCat + POS	$S[dcl]$ + VBD
Rule	$S[dcl] \rightarrow NP\ S[dcl]\backslash NP$
Rule + Word	$S[dcl] \rightarrow NP\ S[dcl]\backslash NP$ + bought
Rule + POS	$S[dcl] \rightarrow NP\ S[dcl]\backslash NP$ + VBD

Rule dependency features:

Word–Word	$\langle \text{company}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP, \text{bought}\rangle$
Word–POS	$\langle \text{company}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP, \text{VBD}\rangle$
POS–Word	$\langle \text{NN}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP, \text{bought}\rangle$
POS–POS	$\langle \text{NN}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP, \text{VBD}\rangle$
Word + Distance(words)	$\langle \text{bought}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 2$
Word + Distance(punct)	$\langle \text{bought}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 1$
Word + Distance(verbs)	$\langle \text{bought}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 0$
POS + Distance(words)	$\langle \text{VBD}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 2$
POS + Distance(punct)	$\langle \text{VBD}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 1$
POS + Distance(verbs)	$\langle \text{VBD}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 0$

Table 4.1: Feature types and examples for the C&C normal-form model, taken from [1]. These entries show features for a rule in the derivation for a sentence like *the company bought stake*.

specifically for beam-decoding that augments the normal-form model with non-local grandparent features. We show that this augmented model is expressive enough to outperform the best published LSTM A* baselines.

4.3.2 C&C normal-form model

The C&C normal-form model has two sets of features, both described in Table 4.1: structural tree features, which lexicalize elements of the parse tree with the head word or POS tag, and rule dependency features, which encode relationships between the head and sister words/POS tags of a binary rule (including three distance metrics based on the number of intervening word tokens, punctuation marks, and verbs).

Shared features

For leaves in a parse tree, only three features (the lexical category, along with the word and with the POS) are stored; for the root of a parse (the children of TOP), there are also three: the final category itself, as well as the category paired with the word or POS of the category's lexical head. For every other backpointer, there are three features for the CCG rule (A $\rightarrow$ B (C)), either on its own or augmented with the word or POS of the rule's lexical head.

Normal-form features

In the normal-form model, backpointers corresponding to binary rules have ten additional features defined over the two children, all of which are extensions to the CCG rule. Four are obtained by annotating the rule with the lexical heads of the children (Word-Word, Word-POS, POS-Word, and POS-POS); the remaining six consist of the rule, the lexical head of the parent (as word/POS), and one of three measures of distance between the lexical heads of the children. The three kinds of distance are in intervening words (0, 1, 2, 3+), punctuation (0, 1, 2, 3+), and verbs (0, 1, 2+), where a token is classified as punctuation or verb based on its POS tag.

A feature is included in the model if it occurs at least twice in the training data's gold derivations. Under this condition, the model includes roughly 425 thousand features.

4.3.3 C&C predicate-argument dependency model

The second C&C parsing model (which we have not reimplemented for this work) is the predicate-argument dependency model. Building on the work in [16], the dependency model uses discriminative features to predict the dependency structure directly. Like the normal-form decoder, the dependency model also uses dynamic programing, though the equivalence class for items in the chart will additionally depend on the set of unfilled dependencies. The definition of the equivalence class is motivated by model's choice of features; equivalent items must share the same set of attributes (e.g. category, lexical heads, etc.) from which the features are obtained, to avoid inconsistent scoring.

Dependency features

In the dependency model, backpointers corresponding to binary rules also have features based on the (one or more) predicate-argument dependencies that were filled by applying the rule. Each dependency feature consists of (1) the head word/POS of the parent, (2) the lexical category of the parent's head, (3) the "slot" of the argument filled by this rule, (4) the head word/POS of the sister, and (5) a field indicating locality.

Structural tree features:

LexCat + Word	$(S/S)/NP$ + Before
LexCat + POS	$(S/S)/NP$ + IN
RootCat	$S[dcl]$
RootCat + Word	$S[dcl]$ + bought
RootCat + POS	$S[dcl]$ + VBD
Rule	$S[dcl] \rightarrow NP\ S[dcl] \backslash NP$
Rule + Word	$S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ + bought
Rule + POS	$S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ + VBD

Predicate-argument dependency features:

Word–Word	$\langle \text{bought}, (S\backslash NP_1)/NP_2, 2, \text{stake}, (NP\backslash NP)/(S[dcl]/NP)\rangle$
Word–POS	$\langle \text{bought}, (S\backslash NP_1)/NP_2, 2, \text{NN}, (NP\backslash NP)/(S[dcl]/NP)\rangle$
POS–Word	$\langle \text{VBD}, (S\backslash NP_1)/NP_2, 2, \text{stake}, (NP\backslash NP)/(S[dcl]/NP)\rangle$
POS–POS	$\langle \text{VBD}, (S\backslash NP_1)/NP_2, 2, \text{NN}, (NP\backslash NP)/(S[dcl]/NP)\rangle$
Word + Distance(words)	$\langle \text{bought}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 2$
Word + Distance(punct)	$\langle \text{bought}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 1$
Word + Distance(verbs)	$\langle \text{bought}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 0$
POS + Distance(words)	$\langle \text{VBD}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 2$
POS + Distance(punct)	$\langle \text{VBD}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 1$
POS + Distance(verbs)	$\langle \text{VBD}, S[dcl] \rightarrow NP\ S[dcl]\backslash NP\rangle + 0$

Table 4.2: Feature types and examples for the C&C predicate-argument dependency model, taken from [1]. These examples describe a long range dependency as in *the stake which I bought.*

If the dependency is local, the fifth field is empty (as in *IBM bought the company*); otherwise, it contains the category which mediated the long-range dependency (as in *the company which IBM bought*).

Permutations over the word/POS of the lexical heads produces four features for each dependency; combining both words and POS tags with the three distance measures produce six more.

4.4 Removing POS tags

While POS tags served an important role in defining the tag dictionaries used by early CCG parsers [10, 11] and supertaggers [24], recent supertaggers do not benefit from the additional supervision provided by automatically-assigned POS tags. The C&C parsing models use POS tags to smooth and assist in parsing sentences with unknown words. However, they are sensitive to the accuracy of the tags: performance suffers when moving from gold to automatically-assigned POS tags within CCGbank, and degrades even further in other domains where the POS tagger is less reliable. Because CCG categories provide so much syntactic information relative to POS tags – and because the parser already has access to the output of the supertagger – it's reasonable to wonder whether the Viterbi lexical category is an acceptable replacement.

If the parsing models can achieve comparable performance using Viterbi supertags instead of POS tags, then there would be no reason to retain the POS tagger in the CCG pipeline. Removing the supertag will make it easier to develop and release self-contained NLP tools based on CCG.

4.5 Results

We now present the results of our parsing experiments on CCGbank.

4.5.1 A* baseline

In the previous chapter, we discussed how 1-best supertag accuracy was a brittle measure of supertagger performance, and that oracle multitagging accuracy was a more meaningful metric of the supertagger's ability to provide signal to the parser. We now present an even more informative measure: the ability of a supertagger to recover CCGbank predicate-argument dependencies using A* search. We present these results in Table 4.3.

Where it was difficult to determine the significance of the variation in oracle multitagging accuracy between the `RNN` and `LSTM-L` models, there is no such ambiguity in the parsing results: the LSTM supertagger

	Dev.			Test		
	LP	LR	LF1	LP	LR	LF1
NN	-	-	-	-	-	83.4
RNN	83.4	84.9	84.2	84.1	85.1	84.6
LSTM-V	86.4	85.9	86.1	87.4	**86.8**	87.1
LSTM-L	-	-	-	**87.7**	86.7	**87.2**
LSTM-L-Tri	-	-	-	**88.6**	**87.5**	**88.1**

Table 4.3: Parsing results using the A* model of Lewis and Steedman (2014) for a selection of supertagging models (none of which require POS tags). Parsers were evaluated on their ability to recover labeled, directed predicate-argument dependencies on all sentences (100% coverage). Derivational ambiguity was resolved using the attach-low heuristic of Lewis et al. (2016).

is clearly better at predicting supertags that reflect the predicate-argument dependency structure of a parse, and in fact it outperforms most of the previous work in supervised CCG parsing. This confirms the result of Lewis et al., and presents a remarkable opportunity to push forward in future work.

4.5.2 Pipeline results

One of the most frustrating aspects of doing research in NLP is the difficulty in replicating the end-to-end results of a pipeline when the intermediate performance of the individual components are not available. The A* baseline provides a way to measure the quality of a parsing model in terms of the improvement it provides, and not just against the final numbers reported in the scientific literature (where errors and improvement propagate in ways that are impossible to identify).

Table 4.4 presents the accuracy of several parsing pipelines on both development and test data, clustered based on the supertager used during evaluation. We indicate the incremental improvement of each pipeline over its A* baseline.

4.5.3 Removing POS tags

We retrained both our implementation of the C&C normal-form model and the Java C&C beam-decoding model using Viterbi supertags in place of POS tags.

Figure 4.1 shows the accuracy of the C&C normal form model on section 00, both with and without POS tags. It is clear that the drop in tagging accuracy due to the larger category space is offset by the extra utility afforded by the richness of the supertags relative to the POS tag set, as the normal-form model performs significantly better using Viterbi supertags as features.

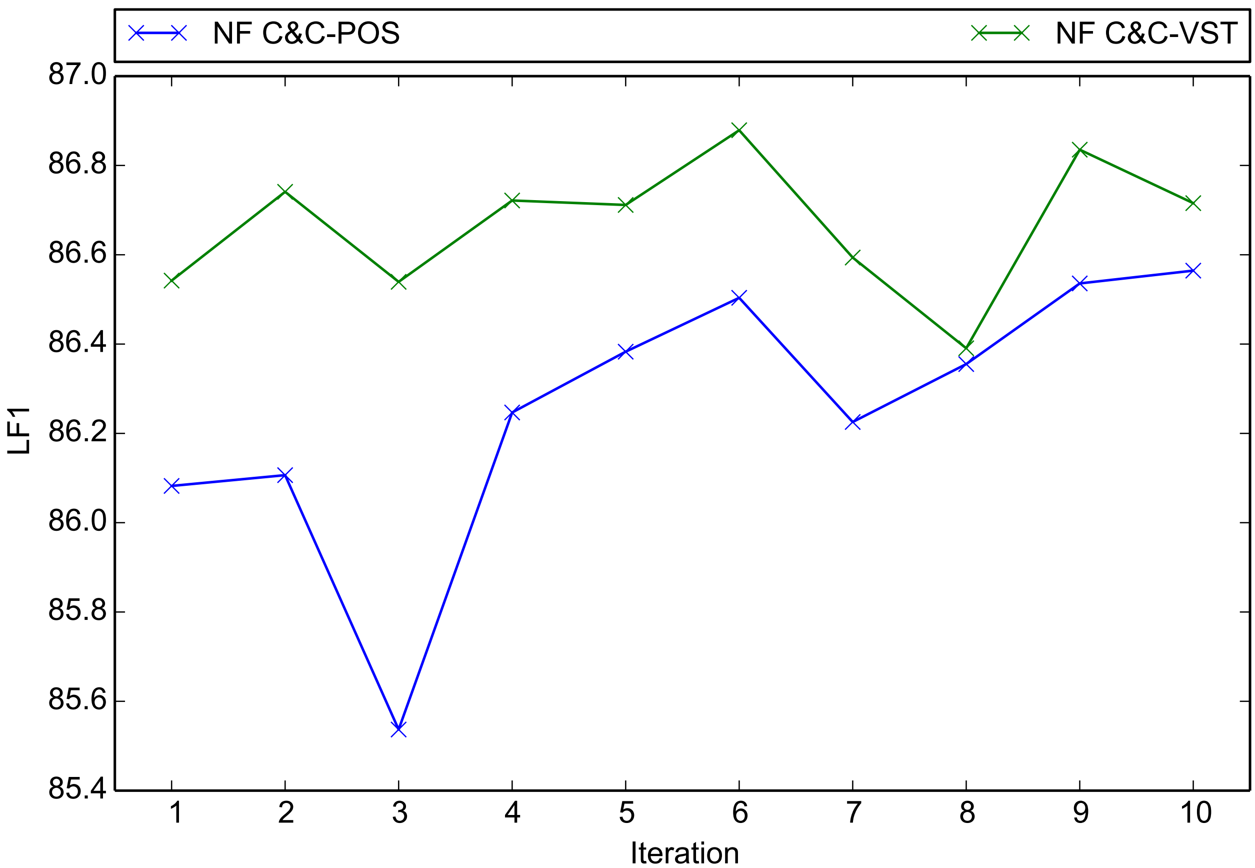

Figure 4.1: Labeled dependency accuracy on section 00 for our implementation of the C&C normal-form model when trained and evaluated using feature templates based on POS tags (`NF C&C-POS`) and Viterbi supertags (`NF C&C-VST`). Each of the models was evaluated on the full development set, including parse failures (100% coverage).

		Dev.		Test	
Supertagger	Parser	LF1	Δ	LF1	Δ
NN	A*	-		83.4	
RNN	A*	84.2		84.6	
	Beam C&C-POS	87.0	**2.8**	87.5	**2.9**
LSTM-V	A*	86.1		87.1	
	NF C&C-POS	86.6	0.5	87.5	0.4
	NF C&C-VST	86.9	0.8	87.8	0.7
	Beam C&C-POS	**87.8**	**1.7**	**88.3**	**1.2**
LSTM-L	A*	-		87.2	
	A* + Dep	-	-	87.8	**0.6**
LSTM-L-Tri	A*	-		88.1	
	A* + Dep	-	-	87.8	**-0.3**

Table 4.4: Labeled dependency accuracy on the development and training sets for several supertagger-parser combinations, along with the increase (or decrease) in accuracy for each parser relative to the A* baseline for each supertagger. The `A*` baseline is the non-parametric supertag-factored parsing model of Lewis and Steedman (2014) that performs inference using a A* search with a fixed heuristic to resolve derivational ambiguity. `NF C&C` is our implementation of the normal-form C&C parsing model of Clark and Curran, which we tested using feature tag templates defined over either POS tags (`NF C&C-POS`) or the Viterbi categories assigned by the supertagger (`NF C&C-VST`). `Beam C&C-POS` is the model provided by Java C&C that uses non-local features to score complete subderivations during beam decoding, with tag feature templates defined over POS tags. Finally, `A* + Dep` is the joint supertag-dependency model used in Lewis et al. (2016), which was adapted from their previous work. Parsers were evaluated on their ability to recover labeled, directed predicate-argument dependencies on all sentences (100% coverage). For A* inference, derivational ambiguity was resolved using the attach-low heuristic of Lewis et al. (2016).

Figure 4.2 shows the results of the same transformation for the Java C&C beam-decoding model; performance degrades slightly when using Viterbi supertags. We suspect that this is a consequence of the beam decoding algorithm itself; because the LSTM supertagger is very accurate, the model will learn to prefer lexical categories that match the Viterbi supertag, even when that tag is incorrect. The beam decoder uses an extremely aggressive pruning procedure and has difficulty resolving ambiguity.

4.5.4 Long-range dependencies

We examined the ability of different parsing models to recover long-range dependencies (i.e., attachments between words more than five tokens apart) on the development set (section 00). First, we fixed the evaluation to use supertags assigned by the LSTM supertagger. For each of the four parsing models – the A* baseline, normal-form C&C with and without POS tags, and the Java C&C beam decoding model – we binned their dependency predictions by length and report LF1 on each bin, following [32].

Figure 4.4 shows the same comparison on long-range dependencies, but between the beam decoder and the A* baseline using tags assigned by both the RNN and the LSTM supertaggers. The beam-decoding model's ability to accurately predict long-range dependencies appears to be robust to the choice of supertagger; even

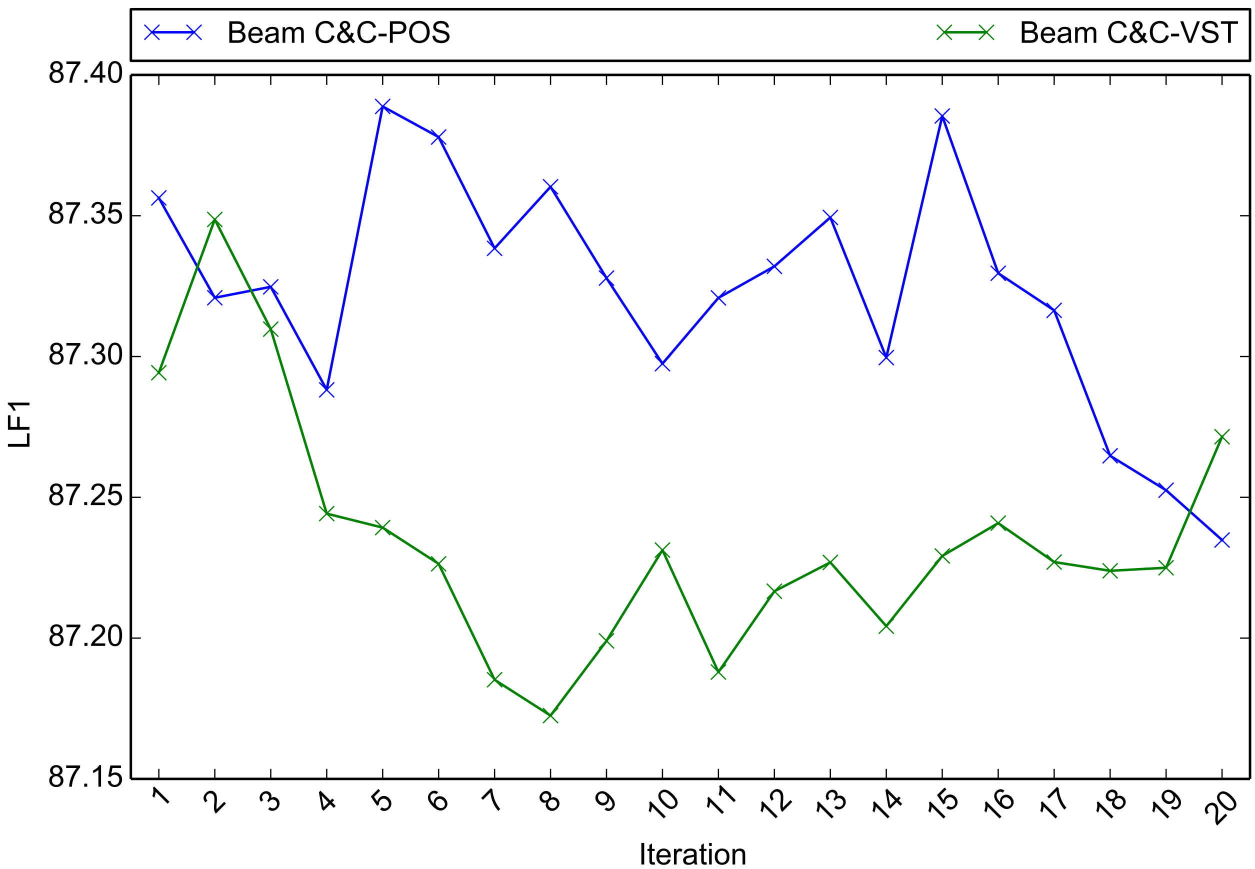

Figure 4.2: Labeled dependency accuracy on section 00 for the Java C&C's beam-decoded model when trained using feature templates based on POS tags (`Beam C&C-POS`) and Viterbi supertags (`Beam C&C-VST`). The beam decoder allows the model to define non-local features over complete sub-derivations. The dependency skimmer was used to obtain partial analyses for parse failures, so the accuracies reported here include "partial credit" for sentences that other algorithms aren't able to parse.

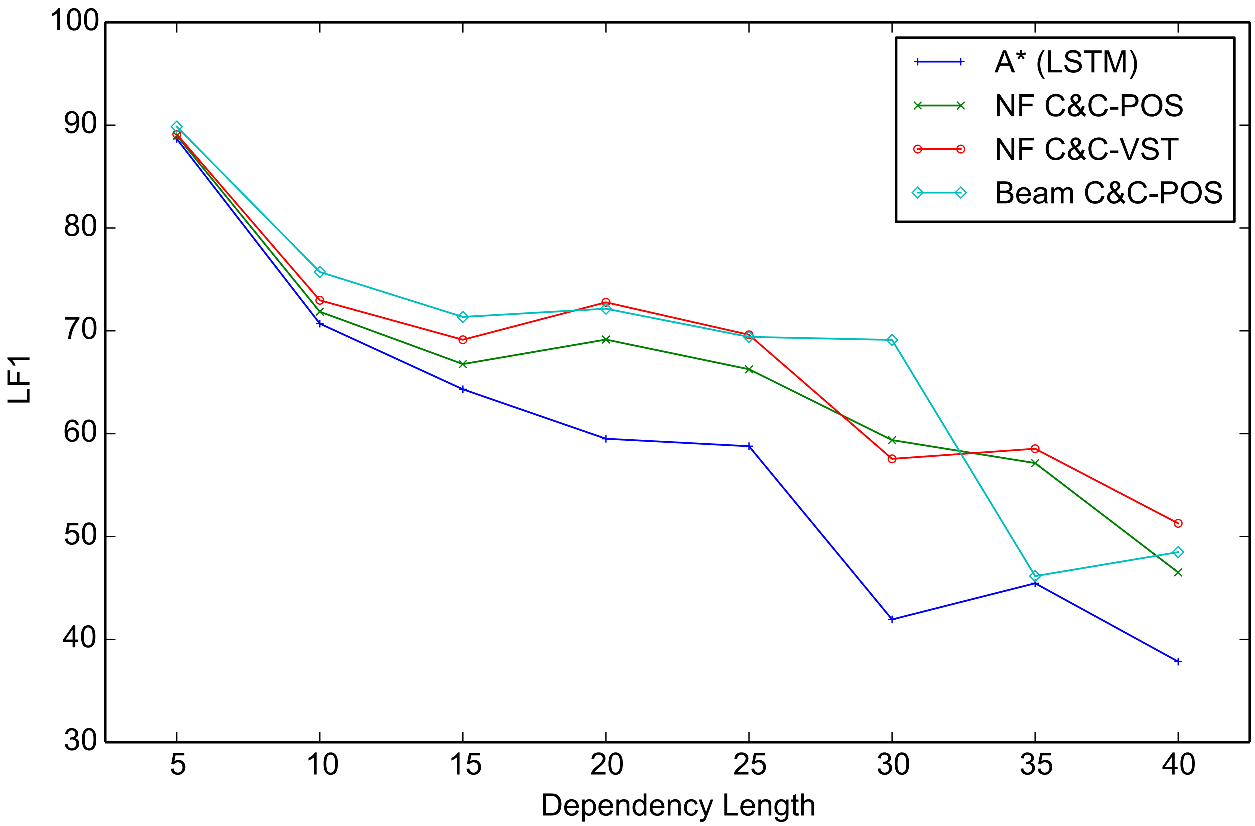

Figure 4.3: LF1 on labeled dependencies of various lengths, evaluated over section 00. All parsers were evaluated using the supertags produced by the LSTM supertagger of Vaswani et al. `A* (LSTM)` is the A* parsing model of Lewis et al.; `C&C-POS` is our implementation of the C&C normal-form model using POS tag features; `C&C-VST` is our implementation of the C&C normal-form model when the POS features are replaced by Viterbi supertag features; and `Beam C&C-POS` is the beam-decoded parsing model in Java C&C and defines additional features over full subderivations (for this instance of the model, the tag feature templates are based on POS tags).

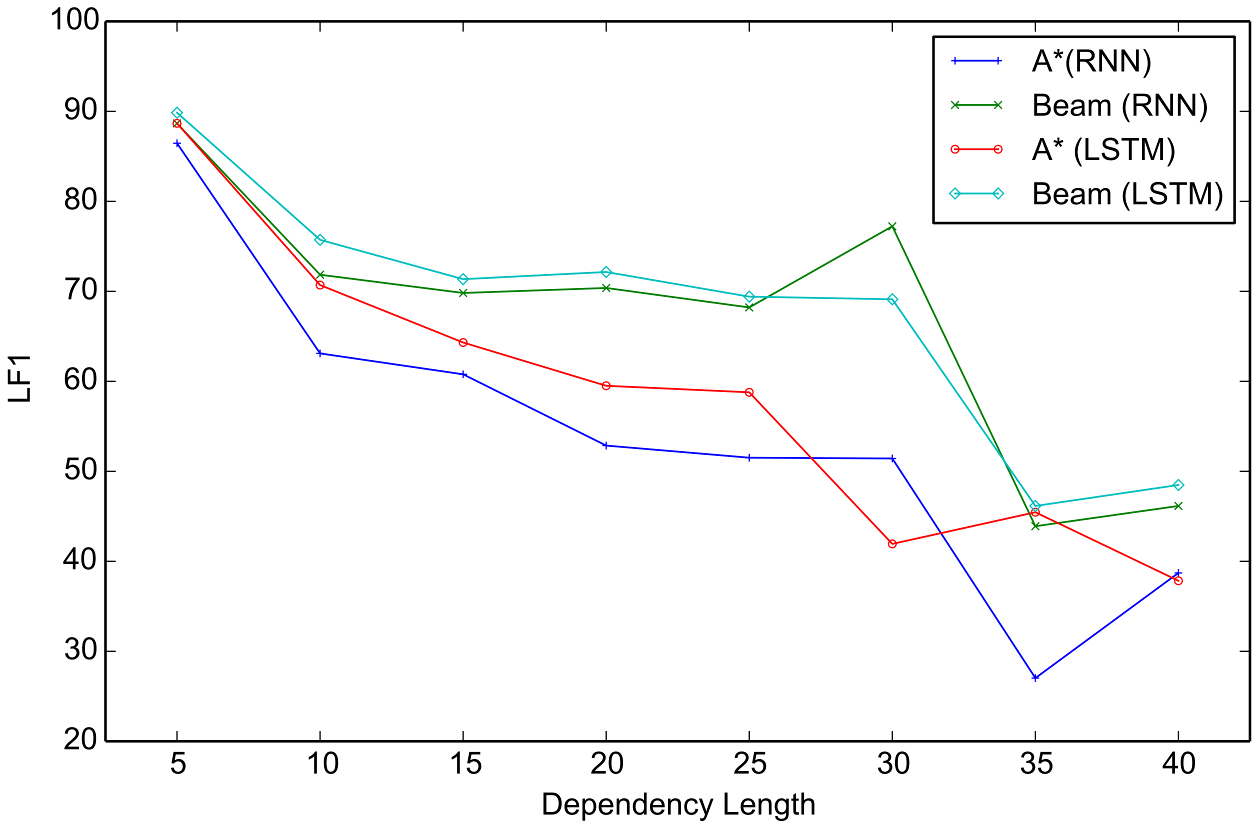

Figure 4.4: Development accuracy on long-range dependencies; we compare the A* baseline with the beam decoding model of Java C&C. Even though the RNN supertagger is significantly less accurate than the LSTM supertagger, the beam decoding model is able to recover long-range dependencies at a very similar rate regardless of which supertags it uses.

though the tags produced by the RNN supertagger are less effective at predicting dependencies on their own using A* search, the model is able to correct the errors implicit in the supertagger's predictions.

4.6 Conclusion

The results presented in this chapter confirm that A* parsing using supertagger output is feasible and effective. We also showed that a beam-decoding model with non-local features can improve on this impressive baseline, particularly when predicting long-range dependencies. In the final chapter, we will propose techniques for combining the efficiency of the A* algorithm with the accuracy of the best non-local parsing models

Chapter 5

Conclusion

As most of the contributions of this thesis are based on a critique of others' work, we conclude with a summary of observations about the state of supervised CCG parsing and a discussion of future work that could produce a parsing algorithm that can improve on some of the deficiencies we find with existing approaches.

5.1 Observations

We believe that there is a fundamental tension in the CCG parsing pipeline today: there are long-range predicate-argument dependencies that are too difficult for the A* parsing algorithm to recover using any existing supertagger, but the only currently-available parsing algorithms that can correct those kinds of errors all require some form of beam search to parse efficiently. Adaptive supertagging is particulary fragile because the parser must either (a) rely on the supertagger to provide the correct lexical categories in the initial beam, or (b) hope that the grammar does not license a successful parse based on the provided supertags, and repeat the process with a wider beam.

CCG supertagging is a difficult task, as the results in Table 3.1 and Table 3.2 illustrate. A* search using the tri-trained LSTM is an excellent baseline for future research, and current methods do have difficulty improving on its performance. However, there's a lot of meat on the proverbial bone: prepositions and known words that appear with a supertag for the first time will always be difficult to disambiguate through sequence-labeling alone, and our results from Figure 4.3 show that parametric parsing models can still help recover long-range dependencies. However, the amount of engineering required to push the baseline leads to overfitting to the training data and target domain. Long-range dependencies represent only 10-12 percent of all dependencies (even in WSJ, which is complex), but account for a quarter of the errors.

In the next section, we discuss the need to develop better parsing algorithms that leverage the power of CCG to efficiently focus on these difficult classifications.

5.2 Future work

An ideal algorithm would combine the efficiency of A* search with the accuracy of a statistical model that makes predictions using non-local context. We believe that such an algorithm exists, but requires a novel combination of the A* and beam-decoding approaches.

5.2.1 Joint CYK inference

Previous approaches to joint supertagger-parser inference using dual decomposition and loopy belief propagation [23] show that it can improve parsing accuracy, but in order to achieve their best result the speed of their parser decreased by an order of magnitude relative to the baseline (the C&C hybrid dependency model). They also trained their supertagging and parsing models separately, and left the presumed benefits of joint training as unfinished future work. We suspect that following up would produce significant gains in accuracy, but the cost to efficiency would remain high.

5.2.2 Joint A* search

The `A* + Dependency` model of [54] that uses A* search and non-local features to jointly predict CCG parses and semantic role labels has many of the properties that we are looking for in a parsing algorithm. It permits exact inference with (theoretical) optimality guarantees by defining the global upper bound as a sum over the highest-scoring analysis of each token, and it explicitly models labeled predicate-argument dependencies. It is designed to recover difficult dependencies by conditioning the dependency features on the distance between a head and its argument, and includes additional features to specifically resolve ambiguous prepositional phrase attachments and predict semantic roles.

However, in practice the full model runs much slower than the simple A* model that uses only supertagger-defined log-probabilities; we posit that it sacrifices considerable efficiency due to the overly-relaxed upper bounds of the A* heuristic when defined over the full set of features. By optimizing for accuracy during training (their objective is to maximize the marginal probability of the gold semantic dependencies, which obviates some of the effects of spurious ambiguity), their learned model has no incentive to limit the number of operations required to find the optimal parse.

This is particularly relevant given the recent gains in supertagging performance. We saw in Chapter 3 that an LSTM supertagger is much more confident in its predictions than previous supertaggers were; this is particularly relevant to A* parsing with the simple supertag-factored model. Note that the upper bound in the simple A* model is fixed during inference – it is always the log-probability of the Viterbi supertag assignment, which itself is the sum of the log-probabilities of the Viterbi supertag for each token. If the grammar can produce a successful parse using the Viterbi supertag sequence where every assignment is "certain" (i.e., the probability of each token's Viterbi tag is close to 1.0), then very few extraneous items will be added to the chart because the margin between a high-probability Viterbi supertag and the corresponding second-best supertag is very large.

The upper bound is initialized in the same way for the full model, and the upper bound for a cell is still the sum of the highest-scoring lexical entries for each of the tokens in that cell's span. However, the upper bound for a token can *increase* during inference based on the dependency structure of the intermediate parses because the model allows dependency features to have positive weight. This is a critical but eminently sympathetic mistake. Log-linear chart parsers are trained to upweight "good" features that occur in correct derivations and to penalize "bad" features that distract from them; dynamic programming algorithms like CYK and beam decoding benefit from this freedom without any caveat. However, the A* algorithm prioritizes the items in the agenda that have the highest figure of merit, and will naturally tend to combine items if they will produce net-positive dependencies. If the positive weight of the attachment causes the score of the modified lexical entry to exceed the previous highest-scoring lexical entry for that token, then the upper bound for the token rises to match it and the global upper bound will relax. As a result, many extraneous items can be added to the chart before a spanning analysis is found, slowing down inference dramatically.

5.2.3 A potential solution

The joint A* parsing model from the previous section is at the same disadvantage as any chart parsing model that tries to explicitly capture predicate-argument dependencies: most dependency attachments are local, and training to maximize the likelihood of the dependencies for an entire corpus will inevitably lead the model towards a preference for local attachments. Ironically, these are exactly the attachments that a supertagger is already effective at recovering. The net result is that instead of building on the impressive baseline performance of the A* parsing algorithm with a state-of-the-art tri-trained LSTM supertagger, the joint model of [32] is harmed by the well-intentioned but ineffective dependency model that – without access to the supertag probabilities itself [1] – naïvely tries to push the LSTM away from the non-local attachments that it correctly predicts.

An ideal parsing algorithm would be nearly as fast [2] and at least as accurate the basic A* parsing algorithm, and none of the systems we tested in the Chapter 4 meet those requirements. Even modifying the

[1] In order to provide a sufficiently different view of the unannotated data considered as part of tri-training.

[2] 214 sentences per second when tagging using a single 3.5GHz core, and 2670 sentences per second using TensorFlow on a single NVIDIA TITAN X GPU [32].

full A* model to restrict dependency features to be non-positive might not be sufficient on its own; while the modified algorithm would not be susceptible to the same insidious upper bound relaxation, inference would be slower due to two factors: first, the computational cost of scoring dependencies before a spanning analysis has been found means that inference will be slower by some constant factor; second, such a model could still penalize an item in the derivation that the simple model would have selected, which could lengthen parsing time if the penalty is large enough that more items need to be processed.

Instead, our parsing algorithm should trust the supertagger to be a bona-fide baseline – after all, it can predict labeled dependencies with 88.1% accuracy – by deferring to the simple A* model to find an initial parse; additionally, the algorithm should learn an expressive parsing model that predicts and penalizes any errors in that full derivation (but never assigns a positive score). If the parsing model does assign a penalty, then the A* baseline can continue to search for additional derivations.

By defining the joint model to be the sum of the supertagger and parsing models, this hypothetical algorithm satisfies our requirements. If the parsing model doesn't penalize the initial derivation at all, then that analysis is certifiably optimal according to the joint model. By encouraging the joint model to be cautious and only assign small penalties (if any) then the joint model should be at least as accurate as the baseline supertagger.

The algorithm is relatively flexible in terms of its stopping conditions: optimality is guaranteed once the A* agenda is exhausted of items for which the figure of merit (an upper bound on the maximal derivation containing the item) is lower than the highest-scoring full derivation, but the algorithm can also "give up" and return the highest-scoring derivation if the size of the chart grows too large, or if the time spent parsing exceeds a predefined limit. The ability to abandon the search and go with the best solution found so far could be an attractive property for an A* algorithm that lacks any other theoretical guarantees about its running time, as we saw with the positive-weight dependency model – especially since the algorithm will have an initial parse to fall back on as soon as the A* baseline can produce it.

5.2.4 Closing remarks

This proposed algorithm is underspecified due to time constraints, so we conclude by mentioning its similarities to dual decomposition [55–57] and forest rescoring/reranking [58, 59]. As in dual decomposition, the joint model recovers the exact optimal parse (modulo tiebreaking) on convergence. The richness of CCG's predicate-argument structure allows for many interesting factorizations of the the parsing model, including the ability to do lazy coarse-to-fine parsing or factor non-local features into lexical category chart items.

References

[1] S. Clark and J. R. Curran, "Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models," *Computational Linguistics*, vol. 33, no. 4, pp. 493–552, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1162/coli.2007.33.4.493

[2] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a Large Annotated Corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[3] M. Steedman, *The Syntactic Process.* The MIT Press, Sep. 2000.

[4] M. Steedman and J. Baldridge, "Combinatory Categorial Grammar," p. 61, Apr. 2007.

[5] A. Szabolcsi, *Quantification.* Cambridge University Press, 2010.

[6] A. Szabolcsi, "Bound variables in syntax (Are there any?)," pp. 295–318, 1987.

[7] A. E. Ades and M. J. Steedman, "On the order of words," *Linguistics and Philosophy*, vol. 4, no. 4, pp. 517–558, 1982.

[8] K. Ajdukiewicz, "Die syntaktische Konnexität," in *Polish Logic 1920-1939*, S. McCall, Ed. Oxford University Press, 1935, pp. 207–231, translated from *Studia Philosophica*, 1, 1-27.

[9] Y. Bar-Hillel, "A Quasi-arithmetical Notation for Syntactic Description," *Language*, vol. 29, pp. 47–58, 1953.

[10] J. Hockenmaier, "Parsing with generative models of predicate-argument structure," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1.* Association for Computational Linguistics, 2003, pp. 359–366.

[11] J. Hockenmaier and M. Steedman, "Generative Models for Statistical Parsing with Combinatory Categorial Grammar," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics.* Association for Computational Linguistics, 2002, pp. 335–342.

[12] J. Eisner, "Efficient Normal-Form Parsing for Combinatory Categorial Grammar," in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, California, USA, June 1996, pp. 79–86.

[13] J. Hockenmaier and Y. Bisk, "Normal-form parsing for Combinatory Categorial Grammars with generalized composition and type-raising," in *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, Beijing, China, Aug. 2010, pp. 465–473.

[14] J. Hockenmaier and M. Steedman, "CCGbank: a Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank," *Computational Linguistics*, vol. 33, no. 3, pp. 355–396, 2007.

[15] J. Hockenmaier and M. Steedman, "CCGbank: User's Manual," Tech. Rep., 2005.

[16] S. Clark, J. Hockenmaier, and M. Steedman, "Building Deep Dependency Structures with a Wide-Coverage CCG Parser," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics.* Association for Computational Linguistics, 2002, pp. 327–334.

[17] M. Collins, "Three Generative, Lexicalised Models for Statistical Parsing," in *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 1997, pp. 16–23.

[18] M. Collins, "Head-Driven Statistical Models for Natural Language Parsing," *Computational Linguistics*, vol. 29, no. 4, pp. 589–637, 2003.

[19] D. Gildea, "Corpus Variation and Parser Performance," in *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 2001, pp. 167–202.

[20] D. Tse and J. R. Curran, "Chinese ccgbank: Extracting ccg derivations from the penn chinese treebank," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 1083–1091.

[21] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: A simple and general method for semi-supervised learning," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 384–394.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[23] M. Auli and A. Lopez, "A* Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, ser. HLT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002472.2002532 pp. 470–480.

[24] S. Clark, "Supertagging for Combinatory categorial grammar," in *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+ 6)*, 2002, pp. 19–24.

[25] S. Clark and J. R. Curran, "The Importance of Supertagging for Wide-Coverage CCG Parsing," in *Proceedings of the 20th International Conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 282.

[26] M. Lewis and M. Steedman, "Improved CCG Parsing with Semi-supervised Supertagging," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 327–338, 2014.

[27] M. Lewis and M. Steedman, "A* CCG Parsing with a Supertag-factored Model," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, October 2014. [Online]. Available: http://www.aclweb.org/anthology/D14-1107 pp. 990–1000.

[28] J. L. Elman, "Learning and development in neural networks: the importance of starting small," *Cognition*, vol. 48, no. 1, pp. 71–99, 1993.

[29] W. Xu, M. Auli, and S. Clark, "CCG Supertagging with a Recurrent Neural Network," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, July 2015. [Online]. Available: http://www.aclweb.org/anthology/P15-2041 pp. 250–255.

[30] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] A. Vaswani, Y. Bisk, K. Sagae, and R. Musa, "Supertagging with LSTMs," in *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics (Short Papers)*, San Diego, CA, June 2016.

[32] M. Lewis, K. Lee, and L. Zettlemoyer, "LSTM CCG Parsing," in *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

[33] M. Honnibal, J. Nothman, and J. R. Curran, "Evaluating a Statistical CCG parser on Wikipedia," in *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*. Association for Computational Linguistics, 2009, pp. 38–41.

[34] L. Rimell and S. Clark, "Porting a lexicalized-grammar parser to the biomedical domain," *Journal of Biomedical Informatics*, vol. 42, no. 5, pp. 852–865, 2009.

[35] S. Pyysalo, F. Ginter, J. Heimonen, J. Björne, J. Boberg, J. Järvinen, and T. Salakoski, "BioInfer: a corpus for information extraction in the biomedical domain," *BMC Bioinformatics*, vol. 8, no. 1, p. 50, 2007.

[36] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[37] M. Yazdani, R. Collobert, and A. Popescu-Belis, "Learning to Rank on Network Data," in *Eleventh Workshop on Mining and Learning with Graphs*. ACM, 2013.

[38] J. L. Elman, "Finding Structure in Time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive Modeling*, vol. 5, p. 3, 1988.

[40] J. Legrand and R. Collobert, "Joint RNN-Based Greedy Parsing and Word Composition," in *Proceedings of ICLR 2015*, no. EPFL-CONF-210025, 2015.

[41] D. Weiss, C. Alberti, M. Collins, and S. Petrov, "Structured Training for Neural Network Transition-Based Parsing," *CoRR*, vol. abs/1506.06158, 2015. [Online]. Available: http://arxiv.org/abs/1506.06158

[42] B. R. Ambati, T. Deoskar, M. Johnson, and M. Steedman, "An incremental algorithm for transition-based CCG parsing," in *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, 2015. [Online]. Available: http://aclweb.org/anthology/N/N15/N15-1006.pdf pp. 53–63.

[43] S. Bangalore and A. K. Joshi, "Supertagging: An Approach to Almost Parsing," *Comput. Linguist.*, vol. 25, no. 2, pp. 237–265, June 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=973306.973310

[44] T. Kasami, "An Efficient Recognition and Syntax Analysis Algorithm for Context-Free Languages." DTIC Document, Tech. Rep., 1965.

[45] D. H. Younger, "Recognition and parsing of context-free languages in time n^3," *Information and Control*, vol. 10, no. 2, pp. 189 – 208, 1967. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S001999586780007X

[46] K. Vijay-Shanker and D. J. Weir, "Parsing Some Constrained Grammar Formalisms," *Comput. Linguist.*, vol. 19, no. 4, pp. 591–636, Dec. 1993. [Online]. Available: http://dl.acm.org/citation.cfm?id=972500.972502

[47] S. Clark, "The Java Version of the C&C Parser Version 0.95," 2015.

[48] S. Riezler, T. H. King, R. M. Kaplan, R. Crouch, J. T. Maxwell, and I. M. Johnson, "Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques ," in *IN PROCEEDINGS OF THE 40TH MEETING OF THE ACL*, 2002, pp. 271–278.

[49] S. Clark and J. R. Curran, "Log-Linear Models for Wide-Coverage CCG Parsing," in *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2003, pp. 97–104.

[50] S. Clark and J. R. Curran, "Parsing the WSJ using CCG and Log-Linear Models," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 103.

[51] J. Bos, S. Clark, M. Steedman, J. R. Curran, and J. Hockenmaier, "Wide-Coverage Semantic Representations from a CCG Parser," in *Proceedings of the 20th International Conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 1240.

[52] J. R. Curran, S. Clark, and J. Bos, "Linguistically motivated large-scale NLP with C&C and Boxer," in *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*. Association for Computational Linguistics, 2007, pp. 33–36.

[53] J. Bos, "Wide-Coverage Semantic Analysis with Boxer," in *Proceedings of the 2008 Conference on Semantics in Text Processing*. Association for Computational Linguistics, 2008, pp. 277–286.

[54] M. Lewis, L. He, and L. Zettlemoyer, "Joint a* ccg parsing and semantic role labelling," in *Empirical Methods in Natural Language Processing*, 2015.

[55] T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag, "Dual Decomposition for Parsing with Non-projective Head Automata," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870658.1870783 pp. 1288–1298.

[56] A. M. Rush and S. Petrov, "Vine Pruning for Efficient Multi-pass Dependency Parsing," in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2012, pp. 498–507.

[57] A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola, "On Dual Decomposition and Linear Programming Relaxations for Natural Language Processing," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2010, pp. 1–11.

[58] L. Huang and D. Chiang, "Forest Rescoring: Faster Decoding with Integrated Language Models," *ACL 2007*, p. 144, 2007.

[59] L. Huang, "Forest Reranking: Discriminative Parsing with Non-Local Features," *ACL-08: HLT*, p. 586, 2008.