Combinatory Categorial Grammar
Parsing With One Structure Per $n$-gram

TIM DAWBORN
sid: 306158957

Supervisor: Dr. James R. Curran
This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Information Technology (Honours)

School of Information Technologies
The University of Sydney
Australia

29 April 2010
Abstract

There is an inherent redundancy in natural languages whereby certain common phrases (or n-grams) appear frequently in general sentences, and each time they appear, they have the same syntactic analysis. We investigate whether the speed of statistical parsers can be improved through the use of a “one structure per n-gram” hypothesis, used in a “one sense per discourse manner”.

We explore how this hypothesis can be used to improve parser efficiency through two different techniques. We first analyse the process of memoising the analyses for frequently occurring n-grams, with the intention that these frequently occurring n-grams can be used directly in a one structure per n-gram manner. We showed through extensive corpus analysis as well as empirical results obtained from experiments that the memoisation of frequently occurring n-grams does not fulfill our one structure per n-gram hypothesis.

The second method we explore in this thesis is to memoise analyses based on frequently occurring sequences of CCG categories. As with the frequently occurring n-gram idea, extensive corpus analysis was performed, which indicated that this memoisation technique has potential to fulfill our one structure per n-gram assertion. Experiments were then performed to test this theory, and a 15% improvement in parser speed was achieved at a loss of 0.5% F-score using our one structure per n-gram assertion.

This speed improvement will have substantial benefits for NLP system such as Question Answering, that rely on very large-scale parsed text.
Acknowledgements

I would like to thank my supervisor, James Curran, for his patience and wisdom. He was always there to provide a guiding hand through the complex field of NLP.

I would also like to thank the whole “Parsing the Web” JHU summer workshop team for the experience, and the JHU staff and academics for hosting the workshop.
## CONTENTS

Abstract ................................................................. ii

Acknowledgements ...................................................... iii

List of Figures ......................................................... vii

List of Tables ........................................................ viii

Chapter 1 Introduction ................................................ 1
  1.1 Natural Language Processing ........................................ 1
  1.2 Parsing ............................................................. 2
  1.3 The Hypothesis .................................................... 3
  1.4 Contributions .................................................... 3
  1.5 Thesis Outline .................................................... 4
  1.6 Miscellaneous .................................................. 4

Chapter 2 Literature Review .......................................... 5
  2.1 Parsing ............................................................. 5
  2.2 Chart Parsing ..................................................... 7
    2.2.1 The Cocke-Kasami-Younger Algorithm ...................... 8
  2.3 Parser Evaluation ................................................ 8
  2.4 One Sense per Discourse ......................................... 10
  2.5 Summary ......................................................... 11

Chapter 3 Experimental Setup ........................................ 12
  3.1 Combinatory Categorial Grammar ................................. 12
  3.2 Statistical Parsing Using CCG ................................. 17
  3.3 Corpora .......................................................... 18
  3.4 The C&C Parser .................................................. 21
  3.5 Evaluation ....................................................... 23
Chapter 4  Token-based Analysis of \( n \)-grams  

4.1 One Structure per \( n \)-gram ................................................. 26  
4.2 Necessary Conditions for One Structure per \( n \)-gram ................. 27  
4.3 Naïve Analysis .................................................................. 27  
4.4 Constituents .................................................................... 28  
4.5 Non-constituents ............................................................. 32  
   4.5.1 Prepositional Phrase Attachment .................................. 34  
   4.5.2 Coordination ............................................................ 36  
4.6 Statistics ......................................................................... 37  
4.7 Pattern-based Constituents ................................................ 39  
4.8 Summary ........................................................................ 39  

Chapter 5  Chart Structure Persistence  

5.1 Motivation ....................................................................... 41  
5.2 Tokyo Cabinet .................................................................. 42  
5.3 Constructing the \( n \)-gram Databases ................................. 43  
5.4 Chart Serialization Process .............................................. 45  
5.5 Frequency Reduction ...................................................... 45  
5.6 Using the \( n \)-gram Databases .......................................... 47  
   5.6.1 Eisner Constraints ..................................................... 50  
5.7 Summary ........................................................................ 52  

Chapter 6  Token-based Results  

6.1 Constituent-forming \( n \)-grams ............................................ 53  
6.2 Non-constituent-forming \( n \)-grams .................................... 55  
6.3 Incremental Parsing ......................................................... 58  
6.4 Summary ........................................................................ 59  

Chapter 7  Category-based Analysis of \( n \)-grams  

7.1 Category-based Analysis .................................................. 60  
7.2 Constituent-forming Category Spans ................................. 61
CONTENTS

7.3  Ambiguity in Category Spans ................................................................. 64
7.4  Summary ............................................................................................... 69

Chapter 8  Category-based Results .......................................................... 70

8.1  Implementation Changes .................................................................... 70
8.2  Experimental Procedure .................................................................... 71
8.3  Results ................................................................................................. 71
8.4  Final Results ....................................................................................... 75
8.5  Summary ............................................................................................... 76

Chapter 9  Conclusion ................................................................................ 78

9.1  Future Work ........................................................................................ 78
9.2  Summary and Contribution ................................................................ 79

Bibliography .............................................................................................. 82
List of Figures

2.1 A cell contains a forest of parse tree root nodes 6
2.2 A cell contains all valid combinations of all the cells which it covers 7
3.1 Sentence \texttt{wsj\_0003.22} from the Penn Treebank (top) and from CCGbank (bottom) 20
5.1 Performance benchmark speeds for Tokyo Cabinet as reported on the authors website 42
5.2 When creating the trigram database, if a trigram of a sentence forms a constituent in the derivation, it is added to the database 44
5.3 Illustrating the \texttt{ngram2id} database on disk which is in a many-to-many relationship with the \texttt{id2chart} database in RAM 44
5.4 Illustration of how the $n$-gram databases are used. The trigram B C D is loaded from the database, inserted into the chart, and blocks out the cells which it spans 47
5.5 The parser could choose a parse tree which does not maintain our one parse per $n$-gram assertion if spanned cells are not blocked out 48
5.6 If overlapping pre-constructed chart structures are allowed, an overall parse is not able to be constructed due to the cell blocking 49
7.1 Longer category spans which indicate ambiguity place a “no paste” restriction on the cells which they cover, allowing the correct derivation to be constructed 68
List of Tables

3.1 Performance of the C&C parser on *WSJ* section 23 (Clark and Curran, 2007b) compared to Hockenmaier’s original **CCG** parser (Hockenmaier, 2003) 22

3.2 Comparing different parser speeds on section 23 of the *WSJ* (Sagae and Lavie, 2005; Clark and Curran, 2007b) 23

4.1 Statistics about varying sized *n*-gram in CCGbank sections 02 to 21 28

4.2 Constituent statistics about the 10 most frequent bigrams in CCGbank sections 02 to 21 which primarily form constituents 29

4.3 Constituent statistics about the 10 most frequent bigrams in the NP corrected version of CCGbank, sections 02 to 21 which primarily form constituents 29

4.4 Constituent statistics about the bigram *a share*; the most frequent bigram with more than one unique derivation in NP corrected CCGbank sections 02 to 21 30

4.5 Constituent statistics about the 15 most frequent bigrams in the NP corrected CCGbank sections 02 to 21 37

4.6 Statistics about the bigram *of the*; the most frequent bigram in the NP corrected CCGbank sections 02 to 21 38

5.1 Hash function collision statistics for all analyses for all *n*-grams of size 2 to 5 inclusive in *WSJ* 02 to 21 43

6.1 Relative frequency of varying sized *n*-grams 53

6.2 The speed versus performance trade-off for varying sized *n*-grams evaluated on CCGbank 00 using different parsing models. The evaluation attributes are parse time (s), labelled and unlabelled F-score (%), and percentage of sentences covered 54

6.3 The derivations used in the manually constructed non-constituent databases 56
6.4 Memoised structures were constructed for the most frequent derivations for varying non-constituent-forming bigrams, which were then used and evaluated against section 00 of the NP-corrected CCGbank

6.5 Performance figures when gradually combining derivations for frequent bigrams which are less likely to suffer from ambiguity issues

7.1 The 25 most frequent categories assigned to the non-leaf nodes across all derivations in CCGbank sections 02 to 21

7.2 The 10 most frequent categories, considered without their features, which are assigned to the non-leaf nodes across all derivations in CCGbank sections 02 to 21

7.3 The 10 most frequent constituent-forming category spans which result in being headed by a N

7.4 The total number of constituent-forming spans with varying heads with different frequency cutoff values

7.5 The 10 most frequent category sequences in which a frequent N headed constituent-forming span (in red) does not in fact form a constituent

7.6 The 10 most frequent category sequences in which a frequent NP headed constituent-forming span (in red) does not in fact form a constituent

8.1 Parsing time versus performance trade off illustrated for varying nopaste cutoff values. Both N and NP headed category spans were used with a minimum frequency cutoff of 50.

8.2 Parsing time versus performance trade off illustrated for varying nopaste cutoff values. N, NP, and PP category spans were used with a minimum frequency cutoff of 50

8.3 Parsing time versus performance trade off illustrated for varying nopaste cutoff values. N, NP, PP, and (S\NP)\(S\NP) category spans were used with a minimum frequency cutoff of 50.

8.4 Parsing time versus performance trade off illustrated for varying nopaste cutoff values. N and NP category spans were used with a minimum frequency cutoff of 20

8.5 Parsing time versus performance trade off illustrated for varying nopaste cutoff values. N, NP, and PP category spans were used with a minimum frequency cutoff of 20
8.6 Parsing time versus performance trade off illustrated for varying nopaste cutoff values. \( N, NP, PP, \) and \((S\backslash NP)\backslash(S\backslash NP)\) category spans were used with a minimum frequency cutoff of 20.

8.7 Section 23 evaluation for the database used in the first row of Table 8.4

8.8 Section 23 evaluation for the database used in the last row of Table 8.4
CHAPTER 1

Introduction

1.1 Natural Language Processing

Natural Language Processing (NLP) is concerned with the interpretation and manipulation of natural languages, such as English, by computer systems. When computers are able to correctly understand and process natural languages, a whole new realm of human-computer interaction will become available. One emerging field which strongly depends on the understanding of sentences is the area of Question Answering (QA). QA is the task of taking a question as a whole sentence, and if possible, returning the answer to the question, instead of a collection of documents which the user then has to crawl through. The future of search based technologies such as web search engines is most likely in the direction of QA. The most prominent interface to search technologies currently is keyword based search, where the user has to map their question into a set of keywords. In order for QA systems to return one answer to the question asked, all of the documents need to be parsed.

Any successful NLP systems will need to be able to understand sentences; that is to be able to extract the meaning of a sentence. The overall meaning of a sentence in a natural language is determined by the interaction between the words of the sentence. Consider the following sentence

\[ \text{John eats apples} \]

A simplistic approach to understanding the structure of this sentence is to use a bag of words model, whereby each word is treated in an unordered manner with respect to one another. This assumption however implies that \text{John eats apples, apples eats John, and apples John eats} would have the same semantics. This simplistic model is not enough to understand the structure of a sentence. Clearly, some form of deeper analysis of sentence structure is needed in order to extrapolate meaning.
1.2 Parsing

Parsing is the process of determining the syntactic structure of a sentence for a given language. The result of parsing a sentence is some structured representation of the sentence, in which the syntactic relationship between words of the sentence is expressed. The nature of this structured representation depends on the method and linguistic formalism used to parse the sentence.

Hand-crafted grammars for natural languages were the first type of grammars used in parsing systems. They existed well before the prevalence of large digital archives of data suitable for training statistical-based parsers. An advantage of hand-crafted grammars is that it is easy to adapt traditional parsing algorithms and technologies to use grammars of this nature. However, hand-crafted grammars are expensive and difficult to produce. Linguistic expertise in the target language as well as vast amounts of development time are required to construct hand-crafted grammars. Domain or medium-specific text often exhibits grammatical constructions which do not exist within the general text of the language. For example, a number of lexical characteristics will be different between the text in newspapers and text in biological research papers. Additionally, a number of grammatical characteristics will be different between newspaper text and Instant Messaging chat logs for example. Manually constructed grammars are not easily adaptable to suit a certain new domain due to the resource limits described, and thus are not very flexible for real world situations.

To combat this problem, parsing techniques which use an automatically extracted grammar from a corpus of text have been developed. Grammars of this nature are computationally extracted from an annotated corpus, using statistical techniques. Parsing frameworks which can use automatically extracted grammars are desirable due to their flexibility. Statistical grammar models for the new target-domain are all that is required in order to adapt a parser of this nature to parse text of a different domain. Running the statistical extraction process over a corpus of annotated text of a new target-domain produces these new statistical models. The only manual process involved here is the creation of the annotated corpus; the grammar itself does not need to be manually constructed. While the process of annotating a corpus is painful, it is less painful than maintaining or developing a hand-crafted grammar.

The speed at which natural language text is parsed is currently too slow for practical use in most real world applications. Parsers need to process large bodies of text, desirably in real time. The current state of the art parsers for English achieve only around 25 sentences per second (Clark and Curran, 2007b). While 25 sentences per second may seem fast, in the context of what needs to be parsed, this is orders
of magnitude too slow. There are currently trillions of words of English text on the web currently. In a few years time, it is predicted that the amount of Chinese content on the web will vastly overtake the amount of English content. Wikipedia is a large knowledge resource on the web which is constantly being updated and changed. If our QA system was to “learn” from Wikipedia, then we constantly have to parse this vast quantity of data. At 25 sentences per second, parsing Wikipedia is far from real time.

1.3 The Hypothesis

The hypothesis which we explore in this thesis is whether or not we can exploit the naturally occurring redundancy in text in order to speed up parsing. We propose a “one structure per n-gram” hypothesis, proposing that every time certain n-grams of text are seen, they always have the same syntactic structure. The motivation behind this idea is that certain n-grams are repeated frequently across bodies of text, and each time they occur, they usually have the same syntactic structure. This implies that the parser is repeatedly deriving the same structure time these frequent n-grams appear in sentences.

1.4 Contributions

State of the art natural language parsers are currently too slow for most real-world applications. In this thesis we present an argument, analysis, and set of experiments to explore ways to improve the speed of a state of the art natural language parser. Our experiments aim to achieve this speedup through the exploitation of the inherent redundancy which exists in natural languages via the memoisation of frequent n-grams.

We propose to explore our hypothesis through the memoisation of the analyses of these frequent n-grams. At a later point in time, if an n-gram which has a memoised form exists within the current sentence being parsed, the parser can load up the derivation from its memoised form and use that directly for the n-gram, instead of having to spend time re-deriving the same analysis again.

Since these common n-grams do not always appear with the same syntactic structure, such a technique will not always result in the correct derivation forming for the rest of the sentence. However, the more of these memoised structures that can be used for a sentence, the less time the parser has to spend constructing an analysis. Thus, there exists a direct trade-off between the efficiency of the parser, and the accuracy of the derivations it produces.
We present a thorough analysis of two different approaches to attacking this hypothesis. For each approach, we confirm the theorised result returned from the analysis via conducting a set of experiments. These experiments provide empirical evidence to support the results of the analysis.

1.5 Thesis Outline

In Chapter 2 we describe the current state-of-the-art, explaining existing parsing techniques, attempts to improve parser performance, parser evaluation, and reference related ideas. Chapter 3 outlines our experimental setup and related background, describing the grammar formalism used in our experiments and analysis, the parser we modify, and the data used in our experiments. Chapter 4 presents an analysis of token-based $n$-grams in relation to the techniques we wish to explore in this thesis, and why there is evidence to suggest that using only $n$-grams of tokens will not suffice. Chapter 5 describes the details of the implementation process used to acquire empirical evidence to backup the claims presented in Chapter 4. Chapter 6 presents the results of these empirical experiments. After showing theoretically and empirically that token-based $n$-grams are not sufficient for the task at hand, we present an analysis of CCG-category-based $n$-grams in Chapter 7. Alterations to the original implementation, as well as empirical results for this category based approach are presented in Chapter 8.

1.6 Miscellaneous

In addition to the contributions made in this thesis, some of the work done in this thesis was accepted into a conference. Our paper titled “CCG parsing with one structure per $n$-gram” was accepted into the ALTA 2009 conference, and as such, is yet to be published. Confirmation of this accepted paper can be found on the ALTA website\textsuperscript{1}. Between semester 1 and 2, I was accepted into the Johns Hopkins University 2009 Summer Workshop within the Center for Language and Speech Processing\textsuperscript{2}. The time spent at this workshop directly contributed towards the outcome of this thesis.

\textsuperscript{1}http://www.alta.asn.au/events/alta2009/alta-2009-accepted-papers.html
\textsuperscript{2}http://www.clsp.jhu.edu/workshops/ws09/
CHAPTER 2

Literature Review

In this literature review we go through the background material behind this thesis which is not dependant on our experimental and implementation decisions. While the concepts motivating our idea are formalism independent, once a formalism is chosen, features of that formalism can be used to one’s advantage. Literature relevant to our specific choice of grammar formalism is presented in Chapter 3. In this chapter, we first outline existing parsing techniques and algorithms, and how they relate to the task at hand. We outline previous attempts at improving parser performance, as well as how parser evaluation is performed. Lastly we discuss other related ideas which inspired the approach taken here.

2.1 Parsing

Parsing is the process of determining the syntactic structure of a sentence for a given language (Manning and Schütze, 2003). More formally, given the grammar definition $G$ for a language $L$, parsing is the process of determining whether or not a sentence $s$ is a member of the language; $s \in L$. A simple alteration to any parsing algorithm can be used to extract the grammatical structure for the sentence if the sentence is a member of the language.

The parsing process generally occurs in one of two ways; either top down or bottom up. Top down parsing attempts to construct a parse starting from the start symbol in the grammar, and attempting to construct a parse tree which covers all of the symbols in the sentence $s$. Bottom up parsing works from the tokens of the input sentence upwards, attempting to construct a parse tree working “backwards” through the grammar, until it generates a single start symbol.

Modern day parsing exists in two main forms. Traditional parsing exists where a parse tree (or similar hierarchical structure) is constructed for the input sentence $s$ given a formal definition of the grammar $G$. Parsers for programming languages for example fall into this category, as manually constructed
grammars for programming languages have a formal definition. Attempts at parsing natural languages using manually constructed grammars exist (Copestake, 1992; Flickinger, 2000; XTAG Research Group, 2001), however, amongst other issues, these grammars are expensive to produce and maintain. Another attribute which makes hand-crafted grammars difficult to manage is that it is difficult to foresee the interaction between different grammar rules. Whenever a new rule is added to allow for some additional construction, how this rule interacts with every other applicable rule has to be thought about to ensure that the grammar is only able to generate valid sentences. Related to this is that it is difficult to imagine all types of constructions which the grammar needs to allow for.

An alternative approach which is more commonly used in the parsing of natural languages is probabilistic parsing. Here a statistical model of a grammar is extracted ahead of time. Each of the grammar production rules has a probability assigned to it relative to the current state of the parse. The most probable parse tree from the forest of possible analyses is chosen as the parse for the sentence. The statistical model of the grammar is normally automatically created via a machine learning process, “learning” the rules of the grammar from a gold-standard annotated corpus.

The amount of literature and research into statistical parsing using linguistically motivated grammars is large and growing. Statistical parsers for most common grammar formalisms have been developed; TAG (Chiang, 2000; Joshi et al., 2003), LFG (Riezler et al., 2002; Kaplan et al., 2004; Cahill et al., 2004), HPSG (Toutanova et al., 2002; Miyao and Tsujii, 2002), and CCG (Hockenmaier, 2003; Clark and Curran, 2007b).

A number of algorithms and techniques exist to parse context-free grammars in $O(n^3)$ time relative to the length of the input sentence (Kasami, 1965; Younger, 1967; Earley, 1970; Schabes, 1991). A large portion of natural languages are context-free in nature, allowing them to be parsed reasonably efficiently.

### 2.2 Chart Parsing

Chart parsing is a parsing technique which uses a data structure called a chart (Kay, 1986). A chart is a triangular hierarchical data structure used for storing the nodes in a parse tree constructed using a bottom up parsing algorithm. The chart data structure is shown pictorially in Figure 2.1. A chart for a sentence consisting of $n$ tokens contains $\frac{n(n+1)}{2}$ cells, represented as squares in the figure. Each cell in the chart contains the parse of a contiguous span or sequence of tokens of the sentence. As such, a cell stores the root nodes of all possible parse trees for the tokens which the cell covers. This coverage is
2.2 Chart Parsing

The row of the chart (also called the span) is the size of the yield, starting from spans of only one token. The column of the chart (also called the pos) is the starting position of the span. The cell \((p, s)\) in the chart contains all possible parses for all of the tokens in the range \([p, p + s]\) for a given sentence. The chart is filled-in from the bottom up, starting with constituents spanning a single token, and then increasing the span to cover more tokens, until all of the tokens in the sentence are covered. This is illustrated in Figure 2.2. Here the cell \((1, 3)\) contains all possible valid ways of making a span of 3 tokens starting at position 1. This is achieved through combining all analyses in cells \((1, 2)\) with \((3, 1)\), as well cells \((1, 1)\) with \((2, 2)\).

A packed chart is a slight variation on the chart data structure. Here, derivations which are semantically equivalent for a particular span of tokens are not stored twice within the one cell; each semantically
equivalent derivation is stored only once. A packed chart is used to help improve the run-time space and
time requirements of any algorithm which processes a chart data structure. Packed charts perform well
in practice if the process of determining whether two derivations are equivalent is fast (Miyao and Tsujii,
2002). The definition of equivalent is not simple in this context. Two derivations are equivalent if they
perform equivalently in terms of the rest of the derivation. Alternatively, two derivations are equivalent
if they are exchangeable in terms of the overall derivation.

2.2.1 The Cocke-Kasami-Younger Algorithm

The Cocke-Kasami-Younger algorithm (Kasami, 1965; Younger, 1967), more commonly known as CKY,
is a dynamic programming chart parsing algorithm which runs in $O(n^3)$ time relative to the length of
the sentence. CKY parses based on a provided grammar which is strictly binary branching.

Any context-free grammar $G$ which accepts a language $L$ can be efficiently transformed into a grammar
$G'$ in Chomsky normal form which accepts $L - \{\varepsilon\}$ (Sipser, 2006). A grammar in Chomsky normal
form is of the form:

\[ A \rightarrow BC \]
\[ A \rightarrow a \]

where $A$, $B$, and $C$ are non-terminals, and $a$ is a terminal. The rule $S \rightarrow \varepsilon$ is also permitted, where
$S$ is the start symbol of the grammar. Since any context-free grammar can be translated into Chomsky
normal form, and any grammar in Chomsky normal form is strictly binary branching, any context-free
grammar can be parsed using CKY.

The dynamic programming nature of CKY makes it very attractive for parsing, as this means back track-
ing does not need to be performed at any stage during the parsing process. The recogniser version of
CKY is shown in Algorithm 1. This algorithm can be converted to return parse trees instead of acting as
a recogniser by simply storing parse nodes in the array $C$ instead of boolean values (Earley, 1970).

2.3 Parser Evaluation

Parser evaluation is a very difficult task (Goodman, 1996; Carroll et al., 1998; Crouch et al., 2002; Clark
and Curran, 2007a) due to fundamental differences in the ways parsers work. Factors which make this
Algorithm 1 The recogniser version of the CKY algorithm

\[
\begin{align*}
G &\leftarrow \text{grammar of the language consisting of } R \text{ non-terminal rules } R_1, \cdots, R_R \\
S &\leftarrow \text{input sentence of length } N \\
C[N, N, R] &\leftarrow \text{new array of booleans initialised to } \text{false} \\
\text{for all } i = 1 \text{ to } N &\text{ do} \\
&\quad \text{for all grammar production rules } R_j \rightarrow S_i \text{ do} \\
&\quad \quad C[i, 1, j] \leftarrow \text{true} \\
&\quad \text{end for} \\
&\quad \text{end for} \\
\text{for all } i = 2 \text{ to } N &\text{ do} \\
&\quad \text{for all } j = 1 \text{ to } N - i + 1 \text{ do} \\
&\quad \quad \text{for all } k = i \text{ to } i - 1 \text{ do} \\
&\quad \quad \quad \text{for all grammar production rules } R_x \rightarrow R_y R_z \text{ do} \\
&\quad \quad \quad \quad \text{if } C[j, k, y] \land C[j + k, i - k, z] \text{ then} \\
&\quad \quad \quad \quad \quad C[j, i, x] \leftarrow \text{true} \\
&\quad \quad \quad \text{end if} \\
&\quad \quad \text{end for} \\
&\quad \text{end for} \\
&\quad \text{end for} \\
\text{for all } i = 1 \text{ to } N &\text{ do} \\
&\quad \text{if } C[1, N, i] \text{ then} \\
&\quad \quad \text{return } S \text{ is in the language of } G \\
&\quad \text{end if} \\
&\quad \text{end for}
\end{align*}
\]

evaluation difficult include the use of various grammar formalisms, differences in parsing techniques, varying hardware and software platforms used to report performance figures, and the parser output format and information.

Parsing can be thought of as a classification task. Ahead of time a gold standard dataset is created which is a set of sentences with their correct derivations in the format which the parser outputs. Evaluation of the parser then can be performed by running these sentences though the parser and comparing the output of each sentence to its gold standard derivation. In classification tasks, the terms true positive (tp), false positive (fp), true negative (tn) and false negative (fn) are used to compare the classification of an item to its correct classification. This is illustrated in the following matrix.

\[
\begin{array}{c|cc}
\text{Correct} & \text{Classification} \\
\hline
\text{Obtained} & \text{E}_1 & \text{E}_2 \\
\hline
\text{Classification} & \text{tp} & \text{fp} \\
& \text{tn} & \text{fn} \\
\end{array}
\]
This matrix is used when the classification is a binary decision, with the two outcomes being $E_1$ and $E_2$. For parser evaluation, this classification might be whether or not two words have a dependency relation for example.

\[ P = \frac{tp}{tp + fp} \]  
\[ R = \frac{tp}{tp + fn} \]  
\[ F_\beta = (1 + \beta^2) \frac{PR}{\beta^2 P + R} \]

The standard way to express the performance of classification task then is through the use of three performance metrics; precision, recall, and F-score (Rijsbergen, 1979). Precision is the number of true positives over the number of true positives plus the false positives (Equation 2.1), recall is the true positives over the true positives plus the false negatives (Equation 2.2). $F_\beta$-score is the weighted harmonic mean of both the precision and the recall (Equation 2.3), and provides a better representation of the overall performance of the system. In practice, $F_1$-score (often denoted simply as F-score) is normally quoted, as this provides equal weighting to both the precision and recall.

One common performance metric used to evaluate parsers on the Penn Treebank is the PARSEVAL metric (Abney et al., 1991). This metric calculates precision and recall relative to linguistic groupings shared between the gold standard derivation and the derivation produced by the parser. The PARSEVAL score factors in the number of bracketings which were correct and incorrect between the gold standard and derived parse trees.

### 2.4 One Sense per Discourse

The concept of “one $X$ per $Y$” has been applied to a number of areas of NLP. The concept was originally introduced in the area of word sense disambiguation by Gale et al. (1992), under the name “one sense per discourse”. The authors found that if a polysemous word such as sentence appears two or more times in a well-written discourse, it was extremely likely that they will all share a common sense. Yarowsky (1993) used this “one $X$ per $Y$” idea again for word sense disambiguation, but this time proposing “one sense per collocation”. Their experiments resulted in a 92% precision rate when applied to “very local contexts”. Guo (1998) applies the same concept to a different task, exploring the hypothesis of “one
tokenisation per source”. Tokenisation accuracy was reported at 90 – 97% in this paper, again showing that the idea of “one X per Y” works well in a number of NLP applications.

Parsing is one NLP task to which, as far as we are aware, this concept has not yet been applied. In this thesis we explore the hypothesis of “one structure per $n$-gram”, where we propose that certain $n$-grams always exist with the same syntactic structure.

### 2.5 Summary

The hypothesis which we aimed to explore in this thesis was whether or not we can exploit the naturally occurring redundancy in text in order to speed up parsing. In a “one sense per discourse” (Gale et al., 1992) manner, we put forward a “one structure per $n$-gram” hypothesis, stating that every time certain $n$-grams of text are seen, they always appear with the same syntactic structure. This thesis explores different methods of attempting to use this hypothesis to improve the performance of the parser. Ideally the speed of the parser should increase and the accuracy should not drop by much.

Everything discussed in this chapter is grammar formalism independent. In the next chapter we will outline the our choice of grammar formalism and the motivation behind choosing it. The choice of grammar formalisms dictates the parsers available for use, as well as what corpora of data are available.
In this chapter we outline how our experiments were performed. Firstly we describe the grammar formalism we chose to use. While the general idea of exploiting the natural redundancy in text via a one structure per \( n \)-gram manner is not specific to any grammar formalism, we chose to use CCG for reasons outlined later. Next we describe the data we will be using for our experiments. A description of the C&C parser, plus the motivation behind using it for out experiments is then presented. Finally we outline how evaluation will be performed.

### 3.1 Combinatory Categorial Grammar

*Combinatory Categorial Grammar* (Steedman, 1996, 2000), hereafter referred to as CCG, is a *lexicalised grammar formalism* which provides a transparent interface between the syntactic surface structure of text, and its underlying semantics. CCG was derived from a combination of the original Categorial Grammar (Steedman, 1987) and the framework of *combinatory logic* developed by Haskell Curry (Curry and Feys, 1958; Moortgat, 1997), in which higher order functions called *combinators* are able to transform and compose other functions. Lambek (1958) proposed the use of combinatory logic in understanding sentence structure, which was formalised into the *Lambek calculus*.

The grammar of CCG is defined recursively in term of *categories*. These categories can be either *atomic* or *complex*. Four atomic categories exist in English CCG: \( S \) (sentence), \( N \) (noun), \( NP \) (noun phrase), and \( PP \) (prepositional phrase), each one representing a logical linguistic entity.

Complex categories are recursively represented in the form \( X / Y \) or \( X \setminus Y \), where \( X \) and \( Y \) are either atomic or complex categories. In the complex categories, \( Y \) is an argument to the functor \( X \). The direction of the slash indicates in which direction the functor \( X \) is expecting to pick up its argument.
3.1 Combinatory Categorial Grammar

$Y$; forward slash indicating to the right and backslash indicating to the left. For example, the complex category $(S\backslash NP)/NP$ is assigned to the word *likes* in the context of the sentence below.

\[
\begin{array}{c}
\text{George} \quad \text{likes} \quad \text{pears} \\
N \quad (S\backslash NP)/NP \quad N \\
NP \quad NP \\
S\backslash NP \\
S \\
\end{array}
\]

These complex CCG categories encode within them the type, cardinality, and directionality of the arguments, as well as the type of the result for the token that the category applies to. In the example above, the category $(S\backslash NP)/NP$ describes the transitive verb (a verb with one object argument) *like*. The category $(S\backslash NP)/NP$ is expecting an NP to the left, and an NP to the right, with the construct resulting in a $S$ being returned once both of these arguments have been bound. Ditransitive verbs (verbs which have two object arguments) are assigned the CCG category $((S\backslash NP)/NP)/NP$ since they require two object arguments to the right $((X/\NP)/\NP)$, and their subject to the left $(X\backslash \NP)$. For example, consider the sentence

I gave the policeman a flower

The verb *gave* is ditransitive as the policeman and a flower are both arguments to the verb gave.

CCG is a binary branching grammar by definition as each category can only pick up one argument. The CCG combinators are high-order functions in the *untyped lambda calculus*, where the arguments to lambda functions dictate the underlying semantics. The original context-free Categorial Grammar (Bar-Hillel, 1957) from which CCG was derived, defines a set of combinators. These combinators are described in terms of CCG categories, as well as with their definitions in the lambda calculus, below.

**Forward application:** $X/Y \quad Y \quad \Rightarrow \quad X$ 
$f \quad a \quad \Rightarrow \quad fa$

**Backward application:** $Y \quad X\backslash Y \quad \Rightarrow \quad X$ 
$a \quad f \quad \Rightarrow \quad fa$

**Forward composition:** $X/Y \quad Y/Z \quad \Rightarrow_B \quad X/Z$ 
$B: \quad f \quad g \quad \Rightarrow \quad \lambda x. f(gx)$
3.1 Combinatory Categorial Grammar

Backward composition: \[ X \setminus Y \Rightarrow_B Z \setminus Y \]

\[ B: \quad g \quad f \Rightarrow \lambda x. f(gx) \]

Forward cross composition: \[ X / Y \quad Y / Z \Rightarrow_{B_x} X / Z \]

\[ B_x: \quad f \quad g \Rightarrow \lambda x. f(gx) \]

Backward cross composition: \[ Y / Z \quad X \setminus Y \Rightarrow_{B_{\times}} X / Z \]

\[ B_{\times}: \quad g \quad f \Rightarrow \lambda x. f(gx) \]

Type Raising:

\[ X \Rightarrow_T Y \setminus (Y / X) \]

\[ X \Rightarrow_T Y / (Y \setminus X) \]

\[ T: \quad a \Rightarrow \lambda f. fa \]

Coordination:

\[ X \text{ conj } X \Rightarrow_{\varPhi} X \]

\[ \Phi: \quad f \quad b \quad g \Rightarrow \lambda \ldots b(f \ldots)(g \ldots) \]

Forward substitution:

\[ (X / Y) / Z \quad Y / Z \Rightarrow_{S} X / Z \]

\[ S: \quad f \quad g \Rightarrow \lambda x. f x(gx) \]

Backward substitution:

\[ Y \setminus Z \quad (X \setminus Y) \setminus Z \Rightarrow_{S} X \setminus Z \]

\[ S: \quad g \quad f \Rightarrow \lambda x. f x(gx) \]

Forward crossed substitution:

\[ (X / Y) \setminus Z \quad Y \setminus Z \Rightarrow_{S} X \setminus Z \]

\[ S_{\times}: \quad f \quad g \Rightarrow \lambda x. f x(gx) \]

Backward crossed substitution:

\[ Y / Z \quad (X \setminus Y) / Z \Rightarrow_{S} X / Z \]

\[ S_{\times}: \quad g \quad f \Rightarrow \lambda x. f x(gx) \]

The composition combinator allows CCG to express mildly context-sensitive situations (Vijay-Shanker and Weir, 1994), whereas the original Categorial Grammar could only express context free expressions. Vijay-Shanker and Weir (1994) proves the weak equivalence in generative power between CCG and other lexicalised grammar formalisms, including LIG (linear indexed grammar) and TAG (tree adjoining grammar), while at the same time outlining CCGs implicit practical benefits over these other formalisms.

When parsing a sentence using CCG, the first step is to assign the appropriate CCG categories to the leaf nodes in the parse tree; the tokens of the sentence. Since CCG categories are lambda calculus functors, a CCG-annotated set of tokens can form a parse tree through lambda reduction. This reduction can be achieved through successive applications of the reduction rules defined for the untyped lambda calculus, or through successive application of the combinators provided by CCG.
A parse of the sentence *I gave the policeman a flower* using CCG can be seen below. The diagram shows how the CCG categories are combined to form the overall parse tree (or derivation) for the sentence.

This sentence however only demonstrates CCG's function application rules, and does not utilise any of its more powerful combinators. Consider instead the sentence *I dislike and Mary likes musicals* (Steedman, 1996). Here, we require the use of CCG type raising in order to provide the correct coordination of arguments within the sentence. Both the transitive verbs *dislike* and *likes* require the same noun (*musicals*) as their object argument in the correct interpretation of this sentence. The derivation below for this sentence shows the underlying lambda calculus operations underneath each of the categories.

Both the *I* and the *Mary* noun phrases are type raised from the category *NP* to the category $S/(S\setminus NP)$ in order to achieve the correct coordination within the sentence. Type raising is required as the *conj* combinator requires both of its arguments to be of the same type. This type raising allows the binding of the subject argument of both verbs to occur before having to bind their object argument. Following this, coordination happens though the use of the conjunction *and*, which in this context allows both of
the object-unbound verb functors to be treated as one from that point onwards in the parse. The object argument is then bound to the resultant functor returned from the \textit{conj} in the last stage of the parse, though forwards function application.

Consider the ambiguous sentence

\begin{center}
\textit{Time flies like an arrow.}
\end{center}

This sentence can be interpreted in a number of different ways. One interpretation is that the sentence is stating that time metaphorically flies the same way in which an arrow flies. Under this interpretation \textit{flies} acts as an intransitive verb (a verb with no object arguments), and \textit{like} acts as a verbal modifier allowing the metaphorical interpretation. The CCG derivation of this interpretation is

\[
\begin{array}{c}
\text{Time flies} \quad \	ext{like} \quad \text{an} \quad \text{arrow} \\
NP \quad S\NP \quad (S\NP)(S\NP)/NP \quad NP/N \quad N \quad NP \quad > \\
(S\NP)(S\NP) \quad > \\
S\NP \quad < \\
S \\
\end{array}
\]

An alternative way to interpret the sentence is that named entities exist which are called \textit{Time flies}, and they are attracted to arrows. Here, \textit{like} acts as a transitive verb, as it is describing the relationship between a subject, the time flies, and an object, the arrow.

\[
\begin{array}{c}
\text{Time flies} \quad \	ext{like} \quad \text{an} \quad \text{arrow} \\
NP/N \quad N \quad (S\NP)/NP \quad NP/N \quad N \quad NP \quad > \\
NP \quad > \\
S\NP \quad > \\
S \quad < \\
\end{array}
\]

This ambiguity example is token level ambiguity, where one token (\textit{like}) can have more than one meaning. Another form of ambiguity exists where two sentences have the same categories, but have two different derivations. Consider the ambiguous sentence

\begin{center}
\textit{I saw the girl on the hill with the telescope}
\end{center}
There are two different ways in which the phrase the girl on the hill with the telescope in this context can be interpreted. The first interpretation is that I used the telescope to see the girl on the hill.

An alternative interpretation of this phrase is that I saw the girl on the hill, and she was geographically located close to a telescope (with the telescope). The categories assigned to each of the tokens in these two interpretations are the same. However, their derivations are different.

3.2 Statistical Parsing Using CCG

CCG is an attractive grammar formalism for use within natural language parsing systems for a number of reasons. Firstly, it is simple to define as the combination of words which are associated via a directional relation with one another. Secondly, it has a strict well-defined formal definition in the realms of untyped lambda calculus and combinatory logic. Another attractive feature of CCG is that it encapsulates within it both the syntactic and semantic representation of the text, allowing even long-range dependencies to be defined using the same syntax. A number of other grammar formalisms do not allow long-range dependencies to be encapsulated. Parsers based on other formalisms consequentially either perform a post-processing stage to identify such dependencies, or ignore long range dependencies all together.
3.3 Corpora

Charniak (1997) outlines the idea of statistical parsing for a context-free grammar. In this paper, the grammar and the statistics used to compute the empirically observed probability distribution are automatically extracted from the annotated Wall Street Journal (WSJ) Penn Treebank (Marcus et al., 1993). Sentences are parsed using a standard context-free parsing method such as a *shift reduce* or *chart parsing* algorithm. These parsing algorithms return a set of possible parses from which the most likely parse needs to be selected. Charniak (2000) proposed an improvement to the original outline for a parser which employs *log-linear* modelling in order to select the most probable parse from the forest of parse trees. Log-linear modelling is also known as *maximum entropy* modelling.

The techniques outlined by Charniak (Charniak, 1997) combined with many other contributions to the field of efficient statistical natural language parsing (Geman and Johnson, 2002) have allowed for the development of efficient CCG based parsing techniques (Eisner, 1996; Hockenmaier, 2003). In particular, Clark and Curran (2007b) provide a detailed description on how to use CCG and log-linear statistical models to perform highly efficient and accurate parsing.

Within the NLP community, there has been some criticism over the use of CCG as a grammar formalism for parsing, due to CCGs “spurious ambiguity” (Wittenburg, 1986). By this it is meant that CCG will produce a number of “spurious” derivations for sentences. This complicates the parsing process because of the *combinatorial explosion* of alternative paths that the parser has to consider for a given sentence. Eisner (1996) describes constraints which can be used in CCG parsers to eliminate this “spurious ambiguity”, by culling certain parse structures. The derivations which uphold these Eisner constraints are known as *normal form derivations*. Clark and Curran (2007b) tried two techniques to combat the combinatorial explosion in their implementation. Firstly to define parsing models in terms of normal-form derivations (Eisner, 1996; Hockenmaier, 2003), and secondly, to define a parsing model over the predicate-argument dependencies themselves. As shown later in Section 3.4, these techniques worked well in practice.

### 3.3 Corpora

The largest CCG annotated corpus available at the current point in time is CCGbank (Hockenmaier and Steedman, 2002; Hockenmaier, 2003). CCGbank was created via a semi-automated translation of the
Penn Treebank\(^1\) (Marcus et al., 1993) to use CCG. CCGbank was created by converting the phrase-structure trees in the Penn Treebank into CCG normal form derivations. An additional post-processing step was required in order to correct the CCG derivation for some constructions such as coordination.

Figure 3.1 shows an example sentence from the Penn Treebank, and its corresponding entry from CCGbank. This sentence is *In July, the Environmental Protection Agency imposed a gradual ban on virtually all uses of asbestos.* In the Penn Treebank phrase-structure tree it can be observed that noun phrases are not represented with any internal hierarchical structure. This is the case for the noun phrase *the Environmental Protection Agency* in this example sentence. Since CCG is a binary branching grammar, these noun phrase cases posed an issue during the CCGbank conversion process as a binary branching hierarchical structure has to be assigned to these flat structures. The result was that strictly right-branching trees were created for all noun phrases in CCGbank. For example, the conversion process results in the bracketing

\[
\text{(consumer ((electronics) and (appliances (retailing chain))))}
\]

instead of the correct bracketing

\[
\text{(((consumer electronics) and appliances) retailing) chain)}
\]

This noun phrase bracketing issue has since been corrected for CCGbank (Vadas and Curran, 2007; Vadas, 2009).

Atomic categories in CCGbank can carry features, notated in square brackets after the name of the CCG category. Sentences (S) can carry features such as \([\text{decl}]\) for declarative, \([\text{wq}]\) for wh-questions, and \([\text{for}]\) for small clauses headed by *for*. Verb phrases also carry features; for example \(S[^b][NP]\) is a bare-infinitive; \(S[^to][NP]\) is a to-infinitive; \(S[^pss][NP]\) is a past participle in passive mode. Determiners specify that the resulting noun phrase is non-bare; \(NP[^nb]/N\). Hockenmaier (2003) contains the complete list of features contained in CCGbank.

An example of these features being used is shown in the derivation below for the sentence *I watched it happen*. A verb which indicates an act of perception (*watched*) often takes a direct object (*it*) and a bare-infinitive (*happen*), where the bare-infinitive indicates that the action is being undertaken by

\(^1\)Available from the Linguistic Data Consortium: http://ldc.upenn.edu/
(S
  (NP (NNP July) ))
  
  (, ,)
  
  (NP-SBJ (DT the) (NNP Environmental) (NNP Protection) (NNP Agency) )
  
  (VP (VBD imposed)
    (NP (DT a) (JJ gradual) (NN ban) )
    (PP-CLR (IN on)
      (NP
        (ADJP (RB virtually) (DT all) )
        (NNS uses) )
        (PP (IN of)
          (NP (NN asbestos) )))))

(Figure 3.1: Sentence wsj_0003.22 from the Penn Treebank (top) and from CCG-bank (bottom))
3.4 The C&C Parser

The C&C parser (Clark and Curran, 2007b) is a statistical natural language parser which uses CCG as its grammar formalism. The parser is implemented using two closely-connected subsystems which work in tandem during the parsing process. First a maximum-entropy supertagger (Bangalore and Joshi, 1999; Clark and Curran, 2004) is used to suggest a set of lexical CCG categories for each token in the input sentence. Following this, the CKY chart parsing algorithm is employed to produce a forest of parse trees. A log-linear model is then used to identify and return the most statistically probable parse for the input sentence.

As mentioned in Section 3.1, using CCG as a grammar for parsing requires you first assign CCG categories to each of the tokens in the sentence, so that reduction (in the form of lambda reduction rules or CCG combinators) can be successively applied in order to produce a parse of the sentence. The job of the supertagger is to assign a small set of CCG categories to each token in a sentence based on a statistical model of the grammar (Curran et al., 2006). One reason that Clark and Curran claim that their parser has such a high throughput while still producing accurate parses, is this close interaction between the parser and the supertagger within the C&C system. When the supertagger is assigning this small set of categories to each token, it does so with respect to a input parameter $\beta$. This $\beta$ parameter describes the maximum level of ambiguity the set of assigned tokens contain. A lower $\beta$ level corresponds to more
ambiguity. When the C&C parser is first asked to parse a sentence, it first asks the supertagger to assign categories to the input tokens relative to a small value $\beta_1$. If the parser is unable to construct a parse using the category assignments provided by the supertagger, another request is made to the supertagger for a larger set of category assignments based on a more lenient parameter $\beta_2$. This feedback loop scenario continues until either the parser can successfully construct a parse, or no more $\beta_i$ values exist. If the parser runs out of $\beta_i$ values then it returns that no parse could be constructed. The value of $\beta_1$ in the default parser settings corresponds to an average of 1.27 categories being assigned to each token. This interaction has proved to be very effective in reducing parsing speeds, as discussed later on (and seen in Table 3.2).

The algorithm used to build the packed charts within the parser is the CKY algorithm. CKY applies naturally to CCG because the grammar is binary branching. Steedman (2000) describes the method for using CKY in CCG parsing.

In order to perform evaluation against parsers which use alternative grammar formalisms, Clark and Curran (2007a) translated the output of the parser into the publicly available DepBank (King et al., 2003) format. Evaluation was also performed on the predicate-argument dependencies within CCGbank. There are a number of general issues associated with evaluating parser performance, as discussed in Section 2.3. In addition, there are a number of issues associated with evaluating a CCG-based parser (Section 3.5.1). Even though the CCG dependencies have to be mapped into another representation to undertake this comparative evaluation, the parser still receives an F-score of over 81% on labelled dependencies, against an upper bound of 84.4% (see Section 2.3 for a definition of F-score). Table 3.1 shows the precision, recall, and F-scores for the C&C parser using the hybrid dependency statistical model, when evaluated on section 23 of the WSJ. The F-score produced by the parser was approximately 3.24% higher than that produced by Hockenmaier (Hockenmaier, 2003). The C&C parser does this while also achieving the fastest known parsing times for the WSJ to date (at the time of publication of
### 3.5 Evaluation

#### 3.5.1 Evaluating a CCG Parser

As mentioned previously in Section 2.3, PARSEVAL is a common metric used for parser evaluation. While PARSEVAL might be a useful metric for comparing phrase-structure grammar based parsers, it has some shortcomings when used to evaluate CCG parsers, as outlined in Clark and Hockenmaier (2002). PARSEVAL works by counting the number of crossed-brackets between the derivation returned by the parser and its gold-standard counterpart. A crossed-bracket occurs when two leaf nodes in the gold standard derivation are joined by some constituent node, but they are not joined in the parser-produced derivation. Penn Treebank trees are very flat structured compared to their corresponding CCG derivations. This means that the cross-bracketing scores for Penn Treebank based trees will be much lower than those achieved by a grammar which produces at most binary-branching trees. Additionally, two CCG derivations can be structurally different yet have the same semantics.

Clark and Hockenmaier (2002) propose two new evaluation metrics for CCG parsers which capture within them the coordination of arguments in the final parse. The first metric is based upon the definition provided in Collins (1999) that a dependency relation exists between two words $w$ and $w'$ if the parse...

---

<table>
<thead>
<tr>
<th>Parser</th>
<th>Time (min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collins</td>
<td>45.0</td>
</tr>
<tr>
<td>Charniak</td>
<td>28.0</td>
</tr>
<tr>
<td>Sagae</td>
<td>11.0</td>
</tr>
<tr>
<td>C&amp;C CCG</td>
<td>1.9</td>
</tr>
</tbody>
</table>

Table 3.2: Comparing different parser speeds on section 23 of the WSJ (Sagae and Lavie, 2005; Clark and Curran, 2007b)

Clark and Curran (2007b)). Table 3.2 demonstrates that the C&C parser is an order of magnitude faster than other state-of-the-art parsers section 23 of the WSJ (Sagae and Lavie, 2005). Additional work has been done to further increase the speed of the parser. Djordjevic et al. (2007) integrated chart repair as well as beam search to the parser, resulting in a 35 – 40% reduction in parsing time without reducing the accuracy of the parser. These changes also resulted in the failure rate of the parser decreasing by 40 – 65%.
contains a local tree such that \( w' \) is the head of this tree, and \( w \) is the head of a non-head daughter. The following tree defines a dependency between \textit{Vinken} and \textit{will}

\[
S(\text{will}) \\
\quad NP(\text{Vinken}) \quad VP(\text{will}) \\
\quad \text{Pierre Vinken will join the board}
\]

The dependency relation is defined by the label of the parent node (\( S \)), the label of the head daughter (\( VP \)), the label of the non-head daughter (\( NP \)), and the direction of the non-head daughter (left): \( \langle S, VP, NP, \text{left} \rangle \). The definition of a dependency here implies that only one relation has to be determined for every word of the sentence. Unlabelled dependencies only take into account whether there is a relation between \( w \) and \( w' \) such that \( w' \) is the head of a derivation which modifies \( w \).

This first metric, however, does not take into account the “deep” dependencies which exist in cases such as raising, control, and coordination. This first metric also makes the assumption that there exists only one dependency relation within the sentence per word. Consider the phrase

\begin{quote}
investors and portfolio managers who want to secure this year’s profits
\end{quote}

Here, both \textit{investors} and \textit{managers} are both subjects of \textit{want}, as well as subjects of \textit{profits}. Consider the CCG derivation for the simple sentence \textit{IBM bought Lotus}.

\[
\begin{align*}
NP \quad (S\backslash NP)/NP \quad NP \\
\quad S\backslash NP \\
\quad S
\end{align*}
\]

There are two different dependency relations encoded within the category for \textit{bought}: \( (S\backslash NP_1)/NP_2 \); the subject of this transitive verb indicated with the subscript 1, and the object, indicated with the subscript 2. The second metric proposed takes into account the correct attachment of these arguments within such a complex relation. A dependency in this second evaluation metric is defined to be the 4-tuple \( \langle h_f, f, s, h_a \rangle \), where \( h_f \) is the head word functor, \( f \) is the functor category extended with the subscript dependency information, \( s \) is the argument slot, and \( h_a \) is the head word of the argument. For our example sentence above, the subject-verb dependency relation for \textit{bought} is thus defined to
be \langle \textit{bought}, (S\backslash NP_1)/NP_2, 1, \textit{IBM} \rangle$. For a dependency to be marked as correct under a labelled dependency evaluation using this metric, all four of these attributes have to be correctly assigned.

### 3.5.2 Evaluating the C&C Parser

The C&C parser is evaluated by comparing the output of the parser against the predicate-argument dependencies in CCGbank. Precision, recall, and $F_1$-score values are reported for both labelled and unlabelled dependencies using the second evaluation metric described in Clark and Hockenmaier (2002).

The aim of the work presented in this thesis is to improve the speed of the parser with minimal loss of performance. As such, all of our experiments will be evaluated with respect to their parsing time versus F-score trade off. It may be acceptable to sacrifice a small percentage of parser F-score in order to achieve a significant decrease in parsing time. Since we are only altering the C&C parser, the changes made in our experiments can be assessed solely through comparing performance figures from the parser itself. Recording a baseline figure before modification, and then comparing this figure with the performance achieved after our modifications is enough to assess what effect the changes had on the performance. As such, cross-parser evaluation is not needed for these experiments.

Empirical program execution times are difficult to compare directly for a number of reasons, including the differences in machine load at the time of different executions. All of the reported speed figures for our experiments are the average of three runs of the same experiments.

### 3.6 Summary

In this chapter we outlined the grammar formalism we have chosen to explore our one structure per $n$-gram assertion. The use of CCG as a grammar formalism has many benefits, but these will become apparent later in Chapters 4 and 7. The use of the C&C parser for our experimentation allows us to evaluate this assertion on a state of the art parser. While the use of such a parser allows the reporting of state of the art performance figures, it also means that any changes we make to the parser have a high baseline to beat.
In this chapter we present our first contribution of this thesis. This chapter contains the analysis we performed on CCGbank to evaluate how our one structure per \( n \)-gram assertion would perform if we memoised our “one structure”s based on frequently occurring \( n \)-grams. The results reported in this chapter are based on corpus analysis. Chapter 6 contains experiments we performed using the parser based on the results obtained in this chapter.

4.1 One Structure per \( n \)-gram

It is important to remember that the concepts motivating this paper could be applied to any grammar formalism. Our experiments, however, were conducted using CCG and the C&C parser (Clark and Curran, 2007b) (Chapter 3). CCG was chosen because it provides attractive features for tackling this task, such as composition and type raising. The reason why this is the case is explained in greater detail later in this chapter.

This thesis explores the idea of exploiting the natural redundancy which occurs in natural languages by using a “one structure per \( n \)-gram” assertion in a “one sense per discourse” manner. The first method we explored was to construct a database which contains the parse structures of certain frequently occurring \( n \)-grams. The idea behind this is that if we can construct and memoise ahead of time the analyses for frequent \( n \)-grams, then instead of letting the parser derive the internal structure of the \( n \)-grams as they are encountered during the parsing process, their analyses can be loaded up from the memoised database instead.

This database of memoised analyses is constructed in order to enforce our one structure per \( n \)-gram assertion. When parsing a sentence, the parser checks to see if it contains an \( n \)-gram which has a memoised form in the database. If this is the case, the memoised analysis will be inserted into the
overall derivation for the current sentence in place of the analysis for the \( n \)-gram the parser would have constructed. If the amount of ambiguity introduced into the overall analysis via loading up a memoised structure is less than the ambiguity associated with the analysis the parser would derive, then a decrease in the overall parse time can be achieved. This decrease in parse time is a result of the parser having to consider less potential parse trees.

**4.2 Necessary Conditions for One Structure per \( n \)-gram**

In order for this “one structure per \( n \)-gram” assertion to work well in practice, the parsed data must possess two properties. Firstly, there must be a large percentage of \( n \)-grams which appear with very few derivations in the corpus. If this property was not present, then this would imply that most of the \( n \)-grams within the text exist with numerous different derivations. As a result, every \( n \)-gram which was loaded up from the database would carry a number of different derivations, which would increase the combinatorial explosion of parse trees which the parser has to consider. More potential parse trees leads to a longer parse time as the parser has to consider these additional parse trees.

The second property is that the most frequent \( n \)-grams in the corpus must have on average very few derivations. If the most frequent \( n \)-grams in the corpus all occurred with a large number of derivations, then the combinatorial explosion problem mentioned for the first property is exacerbated.

Letting the parser construct the derivations from scratch would be faster than using this memoisation technique on data which does not exhibit both of these properties. This is because of the relatively large number of categories assigned to the \( n \)-gram from the databases relative to the small number of categories which would get assigned to the \( n \)-gram in the normal parsing process. These additional categories result in more parse trees that the parser has to consider, which leads to a larger parse time overall.

**4.3 Naïve Analysis**

By analysing all of the \( n \)-grams within sections 02 to 21 of CCGbank purely at the token level, we were able to show that under a very basic analysis, CCGbank satisfies both of the properties discussed in Section 4.2. The results of this analysis can be seen in Table 4.1.
4.4 Constituents

Table 4.1: Statistics about varying sized n-gram in CCGbank sections 02 to 21

<table>
<thead>
<tr>
<th>n-gram size</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg number derivations</td>
<td>1.19</td>
<td>1.09</td>
<td>1.04</td>
</tr>
<tr>
<td>Always form constituents</td>
<td>23%</td>
<td>10%</td>
<td>5%</td>
</tr>
<tr>
<td>Never form a constituent</td>
<td>73%</td>
<td>89%</td>
<td>93%</td>
</tr>
</tbody>
</table>

One interesting result here is the average number of derivations varying n-grams occur with. As described in Section 3.4, when the C&C parser parses a sentence, the supertagger and parser work in tandem. The parser passes a β level to the supertagger, which describes how many different supertags should be assigned to each token in the sentence. From the resultant category set assigned by the supertagger, the parser then attempts to form a spanning analysis for the sentence. If no such spanning analysis could be constructed, the parser increases the β value and tries again.

What is interesting about the average number of derivations each of these different sized n-grams have in Table 4.1, is that they are all less than the average number of categories assigned to each word at the first β level the parser uses. The first β level corresponds to an average of 1.27 categories being assigned to each token in the sentence, which is less than the numbers reported in the first row of Table 4.1. The consequence of this observation is that the insertion of pre-constructed n-gram structures will introduce less ambiguity into the parse compared to what the supertagger would introduce, assuming the insertion process inserts a number of different derivations for n-grams which is less than or equal to the average number of derivations. In theory, less ambiguity implies a faster parse time, as fewer possible parse trees have to be considered by the parser.

4.4 Constituents

The first idea we explored is how well we can do memoising only n-grams which primarily form constituents when parsed. Table 4.2 shows the 10 most frequent bigrams in CCGbank sections 02 to 21 which primarily form constituents in the gold standard data. The columns show the number of times the n-gram was seen not forming a constituent, the number of times it was seen forming a constituent, and the number of distinct constituent-forming derivations for the n-gram.

Knowing that the CCGbank corpus is newspaper text from the Wall Street Journal, one of the first interesting results observed in Table 4.2 is that the bigram New York does not appear. As mentioned in Section 3.3, in the original CCGbank corpus, the bracketing of noun phrases is incorrect due to the
### Table 4.2: Constituent statistics about the 10 most frequent bigrams in CCGbank sections 02 to 21 which primarily form constituents

<table>
<thead>
<tr>
<th>bigram</th>
<th># No</th>
<th># Yes</th>
<th># Uniq</th>
</tr>
</thead>
<tbody>
<tr>
<td>the company</td>
<td>12</td>
<td>1153</td>
<td>1</td>
</tr>
<tr>
<td>a share</td>
<td>3</td>
<td>1082</td>
<td>7</td>
</tr>
<tr>
<td>a year</td>
<td>37</td>
<td>569</td>
<td>9</td>
</tr>
<tr>
<td>do n’t</td>
<td>0</td>
<td>474</td>
<td>9</td>
</tr>
<tr>
<td>the market</td>
<td>37</td>
<td>410</td>
<td>1</td>
</tr>
<tr>
<td>did n’t</td>
<td>0</td>
<td>378</td>
<td>11</td>
</tr>
<tr>
<td>is n’t</td>
<td>1</td>
<td>367</td>
<td>21</td>
</tr>
<tr>
<td>The company</td>
<td>0</td>
<td>359</td>
<td>1</td>
</tr>
<tr>
<td>does n’t</td>
<td>0</td>
<td>328</td>
<td>10</td>
</tr>
<tr>
<td>vice president</td>
<td>8</td>
<td>313</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 4.3: Constituent statistics about the 10 most frequent bigrams in the NP corrected version of CCGbank, sections 02 to 21 which primarily form constituents

<table>
<thead>
<tr>
<th>bigram</th>
<th># No</th>
<th># Yes</th>
<th># Uniq</th>
</tr>
</thead>
<tbody>
<tr>
<td>the company</td>
<td>8</td>
<td>1157</td>
<td>1</td>
</tr>
<tr>
<td>a share</td>
<td>3</td>
<td>1082</td>
<td>7</td>
</tr>
<tr>
<td>New York</td>
<td>4</td>
<td>868</td>
<td>7</td>
</tr>
<tr>
<td>a year</td>
<td>34</td>
<td>572</td>
<td>9</td>
</tr>
<tr>
<td>do n’t</td>
<td>0</td>
<td>474</td>
<td>9</td>
</tr>
<tr>
<td>the market</td>
<td>37</td>
<td>410</td>
<td>1</td>
</tr>
<tr>
<td>did n’t</td>
<td>0</td>
<td>378</td>
<td>11</td>
</tr>
<tr>
<td>is n’t</td>
<td>1</td>
<td>367</td>
<td>21</td>
</tr>
<tr>
<td>The company</td>
<td>0</td>
<td>359</td>
<td>1</td>
</tr>
<tr>
<td>does n’t</td>
<td>0</td>
<td>328</td>
<td>10</td>
</tr>
</tbody>
</table>

Way they are represented in the Penn Treebank. During the creation of CCGbank, Hockenmaier had to somehow convert all flat structured noun phrases into a binary branching tree. The decision made was to make all of these noun phrase trees strictly right-branching (Hockenmaier, 2003). Vadas and Curran (2008) presented a new noun phrase corrected version of CCGbank where the noun phrase bracketing has been corrected (Vadas, 2009). Since the ideas we are exploring here rely on the correct formation of n-grams, the incorrect bracketing of noun phrases in the original corpus will contribute noise to the results. As such, our experiments use the NP corrected version of CCGbank. The most frequent constituent-forming bigrams in the NP corrected corpus are shown in Table 4.3. The bigrams present here are much closer to what was expected.

A number of interesting observations can be made from Table 4.3. Firstly, the number of times these bigrams occur drops off very quickly, with the fourth most frequent bigram appearing just under half the
number of times the most frequent bigram occurs. This frequency drop off goes against our first desirable property for parsed data, that there should be a large number of frequent \( n \)-grams. The last column describes the number of unique derivations each \( n \)-gram was seen forming in the gold standard data. Only three of the top 10 bigrams occur with less than five unique derivations, which goes against our second desirable property for the parsed data, that the most frequent \( n \)-grams occur with very few unique derivations. These two factors indicate that an approach which memoises only constituent-forming \( n \)-grams in a database for use later on will not perform well, as neither of the two properties discussed in Section 4.2 are fulfilled.

Another factor which indicates that using only constituent-forming \( n \)-grams will not perform well is the occurrence frequencies shown in Table 4.3. This table shows the 10 most frequent constituent-forming bigrams across all of CCGbank sections 02 to 21. The most frequent bigram here, *the company*, occurs 1157 times. However, there are around 900000 tokens across sections 02 to 21. This means that the most frequent constituent-forming bigram occurs only around a very 0.13% of the time.

Table 4.4 shows each of the derivations and their occurrence frequencies for the bigram *a share*; the most frequent bigrams in Table 4.3 with more than one unique derivation. An example usage of the most frequent derivation here is when the price of a share is being quoted directly. Since CCGbank is Wall Street Journal newswire text, it is understandable that this the most frequent form of the bigram *a share*.

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Frequency</th>
<th>Coverage</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>((NP\backslash NP)/N)</td>
<td>N</td>
<td>1039</td>
<td>1039</td>
</tr>
<tr>
<td>(((S\backslash NP)\backslash (S\backslash NP))\backslash((S\backslash NP)\backslash (S\backslash NP))/N)</td>
<td>N</td>
<td>19</td>
<td>1058</td>
</tr>
<tr>
<td>(NP/N)</td>
<td>N</td>
<td>12</td>
<td>1070</td>
</tr>
<tr>
<td>((N\backslash N)/N)</td>
<td>N</td>
<td>6</td>
<td>1076</td>
</tr>
<tr>
<td>(((N/N)\backslash (N/N))/N)</td>
<td>N</td>
<td>4</td>
<td>1080</td>
</tr>
<tr>
<td>(((NP\backslash NP)\backslash (NP\backslash NP))/N)</td>
<td>N</td>
<td>1</td>
<td>1081</td>
</tr>
<tr>
<td>(((S\backslash NP)/(S\backslash NP))\backslash((S\backslash NP)/(S\backslash NP))/N)</td>
<td>N</td>
<td>1</td>
<td>1082</td>
</tr>
</tbody>
</table>

Table 4.4: Constituent statistics about the bigram *a share*; the most frequent bigram with more than one unique derivation in \( NP \) corrected CCGbank sections 02 to 21.
The next most frequent derivation occurs when a share is being used to describe the target of an action, such as the falling in price of a share:

<table>
<thead>
<tr>
<th>Telstra</th>
<th>fell</th>
<th>$</th>
<th>2.50</th>
<th>a</th>
<th>share</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>S[decl]\NP</td>
<td>N[num]</td>
<td>(S\NP)((S\NP))!/N[num]</td>
<td>(S\NP)((S\NP))!/((S\NP))!/(S\NP) !)/N</td>
<td>N</td>
</tr>
<tr>
<td>NP</td>
<td>S[decl] \NP</td>
<td>&lt;</td>
<td>S[decl] \NP</td>
<td>&lt;</td>
<td></td>
</tr>
</tbody>
</table>

The third most frequent usage of a share occurs when share is not being used to describe a stock-market share. Share is instead being used to describe a portion of a whole. For example, when somebody owns a share of a house:

<table>
<thead>
<tr>
<th>They</th>
<th>own</th>
<th>a</th>
<th>share</th>
<th>of</th>
<th>Telstra</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>(S[decl]\NP)/NP</td>
<td>NP[\nb]/N</td>
<td>(NP\NP)/NP</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>NP[\nb] &gt; NP\NP</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>NP\NP &lt; &lt;</td>
<td>S[decl] \NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S[decl] \NP</td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These statistics shown in Table 4.4 were generating using the NP corrected version of CCGbank. This table shows that the most frequent derivation covers a massive 96% of all occurrences of the bigram, which from a “one structure per n-gram” point of view is excellent because this result directly implies the second favourable property of the parsed data. If the tail-end of these multiple derivations are not memoised in our databases then the first desirable property may also be achievable.
4.5 Non-constituents

Section 4.4 showed that an approach to this problem which utilises only constituent-forming \( n \)-grams most likely will not perform well as neither of the properties discussed in Section 4.2 are fulfilled. Ergo, another approach is needed for choosing what to memoise in the database.

The next natural direction to turn in is the storing of non-constituent-forming \( n \)-grams. CCG type raising and composition provides us with a method to store non-constituent-forming derivations in these databases. This property is unique to CCG, and provides us with a very powerful mechanism to perform this non-constituent-forming memoisation. For example, consider the bigram \textit{of the}. This bigram has a very low chance of ever forming a constituent in a parse, as \textit{the} will bind with its object argument before \textit{of} comes into play. The CCG derivation for the phrase \textit{of the pear} is

\[
\begin{array}{c}
\text{of} \quad \text{the} \quad \text{pear} \\
(NP\setminus NP)/NP \\
\frac{NP/N \quad N}{NP} \\
\Rightarrow \\
NP\setminus NP
\end{array}
\]

In this derivation, \textit{the} is forward applied to \textit{pear} before \textit{of} binds any of its arguments. Even though \textit{of the} does not form a constituent here, using CCG composition, a derivation for the bigram can be constructed. Consider this CCG derivation for the bigram

\[
\begin{array}{c}
\text{of} \quad \text{the} \\
(NP\setminus NP)/NP \\
\frac{NP/N \quad N}{(NP\setminus NP)/N} \Rightarrow \text{B}
\end{array}
\]

Here we use CCG forward composition to allow \textit{of} and \textit{the} to form a constituent. This parse structure can then be used in the context of \textit{of the pear}, resulting in a derivation which is semantically equivalent to original.

\[
\begin{array}{c}
\text{of} \quad \text{the} \quad \text{pear} \\
(NP\setminus NP)/NP \\
\frac{NP/N \quad N}{(NP\setminus NP)/N} \Rightarrow \text{B} \\
\frac{NP\setminus NP}{NP\setminus NP} \Rightarrow
\end{array}
\]
Here the forward composed version of the bigram of the constructed above is inserted, and forward application is used to complete the spanning analysis. Using the lambda calculus definitions of the forward application and composition combinators, we can show that these two derivations are semantically equivalent.

A number of steps are hidden during the application of the composition combinator (B) in the right derivation above. The steps used to get from the lambda definitions of of and the to their forward composed form is shown below.

\[
\begin{align*}
Bfg & \equiv \lambda a.f(ga) \\
B(\lambda x.\lambda y.((of')x)y)(\lambda x.(the'x)) & = \lambda a.((\lambda x.\lambda y.((of')x)y)((\lambda x.(the'x))a)) \\
& = \lambda a.((\lambda x.\lambda y.((of')x)y)(the'a)) \\
& = \lambda a.\lambda y.((of')(the'a))y
\end{align*}
\]

An example of this technique which uses CCGs type raising combinator (T) on top of the composition combinator is shown below, where we allow the bigram George likes to form an alternative semantically equivalent derivation in the context of George likes pears.
As before, the steps used in the application of the composition combinator are shown below.

\[
\text{B}fg \equiv \lambda a. f(ga)
\]

\[
\text{B}(\lambda f. f\text{George}')(\lambda x. \lambda y. \text{likes'}xy) = \lambda a. (\lambda f. f\text{George}')(\lambda x. \lambda y. \text{likes'}xy)a
\]

\[
= \lambda a. (\lambda f. f\text{George}')(\lambda y. \text{likes'}ay)
\]

\[
= \lambda a. (\lambda y. \text{likes'}ay)\text{George'}
\]

\[
= \lambda a. \text{likes'}a\text{George'}
\]

### 4.5.1 Prepositional Phrase Attachment

CCG forward composition and type raising provides us with the power needed to store non-constituent-forming parse structures in our pre-constructed databases. This technique works for most non-constituent-forming cases, but there exists some situations where it does not. One situation where this technique does not work is prepositional phrase attachment.

Prepositional phrase attachment often constitutes the single largest source of errors in current parsing systems (Merlo and Ferrer, 2006). It is the task of working out where in the sentence to attach a prepositional phrase (PP). Consider the sentence

\[
\text{I saw the girl with the telescope}
\]
The task here is to determine whether the PP \textit{with the telescope} should be attached to the noun \textit{girl}, describing the girl (\textit{the girl with the telescope}), or whether it should be attached to the verb \textit{saw}, describing the method by which the girl was seen (\textit{saw with the telescope}).

The sentence \textit{X of the king of England} is another example of a PP. Here the PP bracketing is either \textit{(X of the king) of England} where \textit{the king} is attached to the placeholder \textit{X}, or \textit{X of (the king of England)}, where \textit{the king of England} is attached to the \textit{X}. The correct CCG derivation for this phrase is

\[
\begin{array}{c}
\text{X of the king of England} \\
\frac{X}{NP} \quad \frac{\text{of}}{(NP\backslash NP)/NP} \quad \frac{\text{the king}}{NP[nb]/N} \quad \frac{of England}{(NP\backslash NP)/NP} \quad \frac{N}{NP[\backslash N]} \quad \frac{NP}{NP} \quad \frac{\backslash NP}{NP} \\
\end{array}
\]

resulting in the noun phrase (NP) \textit{the king of England} forming a constituent in the parse, and then attaching to the placeholder \textit{X} via of. This sentence contains the bigram \textit{of the}, which was used as an example of how to persist non-constituent-forming \textit{n}-grams back in Section 4.5. In this derivation above, \textit{of the} again does not form a constituent. If we were to use the forward composed form of the bigram as described previously, an incorrect analysis for this sentence is constructed.

\[
\begin{array}{c}
\text{X of the king of England} \\
\frac{X}{NP} \quad \frac{\text{of}}{(NP\backslash NP)/N} \quad \frac{the king}{N} \quad \frac{of England}{(NP\backslash NP)/NP} \quad \frac{N}{NP[\backslash N]} \quad \frac{NP}{NP} \quad \frac{\backslash NP}{NP} \\
\end{array}
\]

While an \textit{NP} was still the resultant overall category assigned to the sentence, the internal structure of the noun phrase is incorrect; \textit{king} gets attached to \textit{X} instead of \textit{king of England}. 

\[
\begin{array}{c}
\text{X of the king of England} \\
\frac{X}{NP} \quad \frac{of the}{NP\backslash NP} \quad \frac{king}{NP[\backslash N]} \quad \frac{of England}{NP[\backslash N]} \\
\end{array}
\]
4.5.2 Coordination

Commas can be parsed in one of two ways when using CCG, depending on the role of the comma within the sentence. Commas are either used for coordination purposes, or they are simply absorbed without structural modification to the derivation. Consider the CCG derivation for the sentence shown below:

Here the second comma between England and ate is absorbed. This absorption occurs on the second-last line of the derivation where NP[nb] goes to NP[nb]. The first comma between George and the king of England performs a different syntactic role within the sentence, as it is used to express apposition. Apposition in CCG is represented using the same coordination structure which the conjunction and uses for example; the coordination combinator conj. The type signature for this combinator is

$$X \text{ conj } X' \Rightarrow \Phi X''$$

This definition states that the CCG category to the left and right of the conj have to be the same, and when the combinator is invoked, the resultant category is of the same type. The invocation of the conj combinator can be seen on the third last line of the derivation.

The n-gram pre-construction attempts to load in n-gram chart structures from the databases based purely on n-grams of tokens. At the token level it is not possible to determine if the comma will be absorbed or will be used in apposition, as shown in the sentence above. As such, any n-gram which contains a comma should not be used in the pre-constructed databases, as they are inherently ambiguous when considered purely on the lexical level.
4.6 Statistics

Table 4.5 shows the 15 most frequent bigrams in the NP corrected CCGbank sections 02 to 21. The first group of columns describe the number of times each bigram was seen forming a non-constituent, forming a constituent, and then the number of unique constituent-forming derivations it was seen with. The next column group describes the accumulative percentage of the overall corpus covered if just the bigrams seen already are correctly accounted for. The last column shows the ambiguity the C&C supertagger associates with each bigram when using the default parameters.

The first thing to note about this table is that only two of the top 15 most frequent bigrams primarily form a constituent; the company and a share. Additionally, these two bigrams are not ranked high within the list; 12th and 15th out of 15. This empirical evidence again points towards the theorised conclusion that using only constituent-forming bigrams is not the correct approach to the problem, as was discussed in Section 4.4.

The next observation to be made from this table is seven out of these 15 bigrams contain a comma. As outlined in Section 4.5.2, any n-grams which contain a comma cannot be used in the pre-constructed databases as it is impossible to identify at the lexical level what the semantics of the comma are in its
local context. Unfortunately this means that just by excluding the comma cases, almost half of the 15 most frequent bigrams cannot be used successfully in the pre-constructed databases.

The coverage figures show that just by considering the 15 most frequent bigrams, a coverage of 6.5% of the total number of tokens in CCGbank 02 to 21 is achieved. If a trend like this continues linearly down this list of frequency sorted bigrams, then if a pre-constructed derivation for the first 1000 bigrams exists for example, there is a great potential for the parse time to be improved as a large chunk of the total tokens will not have to be considered by the parser. However, this trend is not linear, as the bigram distribution is very long-tailed.

The ambiguity column provides another perspective on what information this manual analysis provides. This column states the ambiguity level which the C&C supertagger assigns to the bigram when using the default settings. The first $\beta$ level which the parser parses at corresponds to each token of the sentence getting an average of 1.27 categories assigned to it. Nine of the 15 bigrams listed in Table 4.5 have an ambiguity level less than that which the supertagger would provide. This means that parsing using memoised derivations for these nine instances has the potential to decrease parsing time, as the overall sentence ambiguity would decrease compared to if the supertagger was left to assign categories to these bigrams.

Table 4.6 shows all of the categories assigned to the most frequent bigram in the NP corrected CCGbank sections 02 to 21; of the. In a similar manner to Table 4.4, it is observed that even if only the most frequent derivation is memoised for this bigram, even though it is a non-constituent, a coverage of 91.4% of all instances can be achieved.

As a whole, Table 4.5 does not paint a promising picture for the use of token-based $n$-grams for our one structure per $n$-gram assertion. Half of the 15 most frequent bigrams are not able to be used in
4.8 Summary

Under a naïve analysis, it was believed at first that CCGbank possessed both of the desirable properties described in Section 4.2 in order to allow our one structure per \( n \)-gram idea to work well in practice. However, under deeper analysis this was proven incorrect for a number of linguistic reasons. It was hypothesised that using only constituent-forming \( n \)-grams would not perform well due to small number of primarily constituent-forming \( n \)-grams which exist in the dataset. Analysis of using non-constituents instead showed that while CCG allows this technique to be possible, there are a number of reasons why using non-constituents is problematic. Frequency analysis of the gold standard data indicates provided

our assertion due to the coordination issues described in Section 4.5.2. The remaining non-constituent-forming bigrams have the potential to suffer from prepositional phrase attachment issues as described in Section 4.5.1 due to the nature of the tokens in question. Table 4.6 indicates that for some PP-attachment bigrams however, the most frequent derivation occurs the vast majority of the time. As such, empirical results should be obtained to test this idea, as while the data as a whole does not shine an overly positive light, there still exists some positive attributes.

4.7 Pattern-based Constituents

Through manual analysis of the list of all bigrams in CCGbank sections 02 to 21, it was observed that a number of frequent structures were repeated frequently. For example, there are 2080 constituent-forming and 4926 non-constituent-forming occurrences of the bigram pattern “\$\ <\text{NUM}\>”, where \(<\text{NUM}\>\) matches the regular expression “\^[0-9.]*\^[0-9.]*\^”. Comparing these occurrence counts to those listed in Table 4.5, you can see that this simple “\$\ <\text{NUM}\>” pattern occurs more frequently than the most frequent bigram considered purely at the lexical level. No actual instances of this pattern will rank highly in a lexicalised occurrence list sorted by frequency because each instance of a \(<\text{NUM}\>\) only occurs once or twice in the training data.

Pattern-based analysis has the problem that patterns which are useful for the task at hand probably have to be discovered manually through the analysis of the data. It is also not clear whether there are a number of other patterns which are as useful as this “\$\ <\text{NUM}\>” example. This idea will not be explored further in this thesis, but poses an interesting question to investigate for future work.

4.8 Summary

Under a naïve analysis, it was believed at first that CCGbank possessed both of the desirable properties described in Section 4.2 in order to allow our one structure per \( n \)-gram idea to work well in practice. However, under deeper analysis this was proven incorrect for a number of linguistic reasons. It was hypothesised that using only constituent-forming \( n \)-grams would not perform well due to small number of primarily constituent-forming \( n \)-grams which exist in the dataset. Analysis of using non-constituents instead showed that while CCG allows this technique to be possible, there are a number of reasons why using non-constituents is problematic. Frequency analysis of the gold standard data indicates provided
both positive and negative indicators towards the successful performance of this idea. As such, it is concluded to perform an empirical experiment in order to determine how well the idea works.
In this chapter we describe the modifications made to the C&C parser to allow for the memoisation and usage of memoised chart structures. The motivation behind the memoisation of chart structures is outlined in Chapter 4. We explain how our implementation was performed and optimised in order to get maximum throughput.

5.1 Motivation

In Chapter 4, we conducted an analysis of the derivations in CCGbank to determine whether testing our one structure per \( n \)-gram assertion on \( n \)-grams would work well in practice. This analysis was performed in terms of \( n \)-grams of tokens. As such, there are a very large number of \( n \)-grams to consider due to the vast number of different tokens in CCGbank. While there are a very large number of different \( n \)-grams to consider, there are only a relatively small number of different syntactic structures which are assigned to these \( n \)-grams. Both of these properties were shown in Chapter 4.

We test our one structure per \( n \)-gram hypothesis through the construction of a database containing the memoised analyses for frequent \( n \)-grams. This database will be referred to as an \( n \)-gram database. Since there are so many unique \( n \)-grams which exist in CCGbank, it is not feasible to store this database in RAM, as the database will potentially be too large. However, since there exists only a small number of unique syntactic structures, these are able to stored in RAM. As the size of the \( n \)-gram database is potentially too large to store in RAM due to the vast number of unique \( n \)-grams, a fast disk-based lookup alternative is needed so that we can test our one structure per \( n \)-gram assertion.

As mentioned in Section 3.4, the C&C parser is a state-of-the-art parser. Any modifications we make to the parser have to perform very efficiently, as the performance of our changes have to be compared to the baseline performance in terms of both parsing accuracy and speed. Our original motivation for
testing our one structure per \( n \)-gram hypothesis was to improve the efficiency of parsers as they are currently too slow for most real-world applications. Being state-of-the-art, the parser is a tough baseline to beat. As such, our implementation is highly tuned in order to minimise any additional overhead our memoisation process introduces.

5.2 Tokyo Cabinet

Tokyo Cabinet\(^1\) is an open source, lightweight database API which provides a number of different database implementations, including a hash database, B+ tree, and a fixed-length key database. Our experiments used Tokyo Cabinet to store the pre-constructed \( n \)-grams because of its ease of use, speed, and maximum database size (8EB\(^2\)). A large maximum database size is important because more data is better for the database construction phase.

Figure 5.1 shows benchmarking performance figures for Tokyo Cabinet as reported by the authors\(^3\). The benchmarks were conducted to compare the performance of Tokyo Cabinet against other leading lightweight database APIs. All rows in the figure which are prefixed with TC are Tokyo Cabinet database implementations. As the figure illustrated, Tokyo Cabinet is very efficient, making it a desirable choice for use in our implementation.

---

\(^1\)http://tokyocabinet.sourceforge.net/

\(^2\)1 EB = 1 exabyte = 1 billion gigabytes = \(10^{18}\) bytes

\(^3\)http://1978th.net/tokycabinet/benchmark.pdf
5.3 Constructing the \(n\)-gram Databases

The construction of the final set of \(n\)-gram databases is a multi-stage process, with intermediate databases being generated and then refined.

The first stage in the construction of the databases is to parse all of the training data, which in our case is \texttt{WSJ} sections 02 to 21. The parse tree for every sentence is then analysed for constituent-forming \(n\)-grams. If a constituent-forming \(n\)-gram is found and the size of its span (number of tokens) is a value which we would like to construct a database for, then the \(n\)-gram and its corresponding chart structure are written out to a database. These first stage databases are implemented using a simple key-value Tokyo Cabinet hash database. The structure of the keys and values in this database are

\[
\text{Key} = (n\text{-gram}, \text{hash of chart})
\]
\[
\text{Value} = (\text{chart}, \text{occurrence counter})
\]

A 64-bit hash function was developed for analyses so that we can represent a whole analysis by its value, while obtaining very few hash collisions. Empirical evaluation of this hash function was performed using all of the chart structures for all constituent forming \(n\)-grams of size 2 to 5 contained in the NP corrected version of \texttt{CCGBank}. No hash collisions were discovered during this process as shown in Table 5.1.

\begin{table}[h]
\begin{tabular}{|c|c|}
\hline
Unique charts & 536165 \\
\hline
Collisions & 0 \\
\hline
\end{tabular}
\caption{Hash function collision statistics for all analyses for all \(n\)-grams of size 2 to 5 inclusive in \texttt{WSJ} 02 to 21}
\end{table}

The \texttt{chart} attribute in the value is a memoised (serialised) version of the chart structure which an analysis exists within. This memoised chart structure can be unserialised at some later point for reuse. The \texttt{occurrence} counter is incremented each time an occurrence of a key is seen in the parsed training data. A record is also kept in the database for the number of times a particular \(n\)-gram was seen forming a non-constituent, for use in the filtering stage to be discussed in Section 5.5.

This process of \(n\)-gram chart serialisation is illustrated in Figure 5.2. When parsing the sentence \texttt{A B C D E}, the trigram \texttt{B C D} formed a constituent in the spanning analysis for the sentence. Because it formed a constituent, the trigram is added to the first stage trigram database.
One subtle property of the serialisation process is that \( n \)-grams of the same size often have the same syntactic analysis. This is beneficial when creating our \( n \)-gram databases, as instead of having a one-to-one mapping from \( n \)-gram to chart structure, we can have two different databases. The first database is called \texttt{ngram2id} which maps an \( n \)-gram hash to a unique id within the second database. The second database, named \texttt{id2chart}, maps a unique id to a unique analysis. This is illustrated in Figure 5.3.

As mentioned earlier, the size of the \texttt{ngram2id} database can become very large due to the number of unique \( n \)-grams which exist within the corpora these databases are constructed over. The size of this database is too large to fit into \texttt{RAM}. The \texttt{id2chart} database however is much smaller, and is able to be stored in \texttt{RAM}. This is beneficial as if we can efficiently perform an existence check on the \texttt{ngram2id} database, then the loading of the corresponding memoised analysis can be performed very efficiently since the database is kept in \texttt{RAM}. The use of a Tokyo Cabinet hash database for the \texttt{ngram2id} database provides us with the desired efficient existence lookup.
5.4 Chart Serialization Process

We previously stated that the implementation has to be as fast as possible since the parser is already a highly tuned system which is difficult to beat. One area where a lot of speed could be lost is in the chart serialisation and unserialisation process. A naïve way to implement this serialisation would be to map the data structures and their dependencies to some string representation. This encoding could then be stored in the database as a simple string. When the derivation needs to be unserialised, the encoding process is reversed, and the data structures are re-constructed appropriately. The encoding and decoding process is an expensive operation however, so this is not ideal.

In order to achieve maximum throughput of our implementation, we implemented our own object serialisation using a technique known as pointer swizzling (Kemper and Kossmann, 1995). In order to swizzle some data structure, we first identify how much memory, \( m \), the variable \( v \) and all of its dependencies (deep searching) use. A new block of memory \( B \) of size \( m \) is then allocated. Starting from \( v \), we copy this variable to the start of \( B \); this copy of \( v \) is called \( v' \). For every pointer \( p \) contained within \( v \), we then copy the target of the pointer to the next available slot in \( B \), leaving no empty spaces. If the target of pointer \( p \) from variable \( v' \) is copied into memory located \( x \) (which is somewhere between \( B \) and \( B + m \)), the value of \( p \) is then updated to be \( x \). Once \( v \) has been deep copied into \( B \), all of the pointers within the memory range \( B \) to \( B + m \) will have values between 0 and \( m \), indicating the offset of the target of the pointers relative to \( B \). \( B \) can then be dumped into our database as the serialised form of the analysis.

Pointer swizzling is a highly efficient form of serialisation as the only computation required to perform serialisation and unserialisation is copying memory and updating pointer values; no encoding or decoding is required.

5.5 Frequency Reduction

When constructing the initial set of databases over a body of text, a large number of the \( n \)-grams which were memoised should not be kept in the final databases. This is either because they occur too infrequently to be reliable, or more importantly, because the number of times they are seen forming a non-constituent outweighs the number of times they are seen forming a constituent. As such, a frequency based filtering stage is performed on the initial set of databases to produce the final `ngram2id` and `id2chart` databases.
The inequalities in equations 5.1 and 5.3 describe the conditions which need to be fulfilled in order for a particular \(n\)-gram not to be filtered out. One of these inequalities is chosen depending on whether or not the \(n\)-gram was seen forming a non-constituent during the database development phase. In these equations, \(C\) is a mapping from chart structure to frequency count for the current \(n\)-gram, the 0th index into \(C\) is the non-constituent forming frequency count, and \(X\) and \(Y\) are parameters to the filtering process. \(X\) is a cutoff value which is used to define what is a ratio between the frequencies of each of the different derivations for a \(n\)-gram which is to be kept in the final set of databases. \(Y\) is another cutoff value which sets the minimum number of times an \(n\)-gram should be seen for it to be considered for use in the final set of databases.

\[
\frac{C_0}{\sum_{k\neq 0} C_k} < X \quad (5.1)
\]

\[
m = \arg \max_k C_k \quad (5.2)
\]

\[
\frac{(\sum_k C_k) - m}{m} < X \land m > Y \quad (5.3)
\]

If an \(n\)-gram was seen forming a non-constituent during the initial database development phase, then Equation 5.1 is used. This inequality states that the \(n\)-gram must be seen as a constituent more frequently than as a non-constituent. The size of this difference is governed by the parameter \(X\). If an \(n\)-gram was never seen forming a non-constituent during the development phase, then Equation 5.3 is used. This inequality states that the most frequent derivation the \(n\)-gram occurs with must occur at least \(Y\) times, and that the most frequent derivation occurs much more frequently than its other derivations, governed by \(X\).

The values given to the \(X\) and \(Y\) parameters in the filtering process were optimised experimentally, training on sections 02 to 21 and testing on section 00 of the NP corrected CCGBank. For all of our results, \(X\) was set to 0.05 and \(Y\) was set to 15.

The output of this frequency reduction process is a mapping from an \(n\)-gram to a list of chart hashes. A third parameter \(Z\) is provided to the filtering process, which describes the number of unique chart structures the frequency filtering should output for use in the final set of databases. Algorithm 2 describes how the frequency reduction process outputs its final list of \((n\text{-gram}, \text{chart hash})\) pairs. For our results,
Z was set to 255 so that the unique id’s in the \texttt{id2chart} database can be represented using only one byte.

**Algorithm 2** The algorithm for producing output in the frequency reduction step

\begin{algorithmic}
\State $P \leftarrow$ stack of all \((n\text{-gram, chart hashes})\) pairs
\State order $P$ by their occurrence frequency in descending order
\State $S \leftarrow \emptyset$
\While {$|S| < Z$}
\State $p \leftarrow$ pop \((n\text{-gram, } h)\) pair off the top of $P$
\State \textbf{print} $p$
\State $S \leftarrow S \cup h$
\EndWhile
\end{algorithmic}

### 5.6 Using the \(n\)-gram Databases

Once the \(n\)-gram databases have been constructed, they are used when parsing sentences in the future. For every sentence that is parsed, the parser checks to see if any \(n\)-gram contained within the current sentence exists within one of the \(n\)-gram databases it is running with, and if so, uses the pre-constructed chart structure.

A rolling 64-bit hash function was used to create hash values of each possible \(n\)-gram, to efficiently check to see if the \(n\)-gram exists in the pre-constructed database. The construction of these cell hash...
values requires an $O(n^2)$ pass over the chart before parsing; $n$ being the length of the sentence currently being parsed. Once the hash value for a cell has been created, the ngram2id database is queried to establish whether the $n$-gram covered by the current cell in the chart has a memoised form in the databases. If a memoised form does exist, then the corresponding analysis is loaded from the id2chart database. The analysis is then loaded into the cell of the current chart which spans the current $n$-gram. All cells in the current chart which this $n$-gram spans are then blocked out. This whole process is illustrated in Figure 5.4, where the pre-constructed derivation is loaded from the database (in green) and inserted into the current chart. The cells which are spanned by this $n$-gram are then blocked out (grey hashed).

Cells which have been blocked out do not participate in the CKY parsing process. The result of this is that blocked out cells never have the potential to be part of the overall parse tree. The cells which are spanned by a $n$-gram which has been loaded from the databases are blocked out because of our one structure per $n$-gram hypothesis. If these cells were not blocked out, a parse tree as illustrated in Figure 5.5 could be constructed. In this figure, even though we wanted to make an assertion that the trigram $B$ $C$ $D$ would always have the derivation in green, the parse tree for the overall sentence does not use cell $(1, 3)$ as all of the cells which it spans (cells $(1, 1)$, $(1, 2)$, $(2, 1)$ and $(2, 2)$) were allowed to participate in CKY.

The check to test whether any $n$-gram in the current sentence exists in the database is performed through a left to right iteration across the tokens of the sentence. A consequence of this is that if two $n$-grams overlap and both have pre-constructed derivations in the database, then only the first $n$-gram will have its pre-constructed derivations loaded into the current from the database. Overlapping cells are not allowed because of the cell blocking which is required to enforce our one structure per $n$-gram assertion. To see why, consider the situation illustrated in Figure 5.6. Here both the trigrams $A$ $B$ $C$ and $B$ $C$ $D$ exist in
the trigram database, and are loaded into the current chart (into cells (0,3) and (1,3) respectively). All cells which are covered by these two n-grams are then blocked out from CKY, as shown in the bottom-right of the image. However now no derivation for this sentence is able to be constructed as the token A is not being used in a pre-constructed form, and was never considered during CKY. Cells (0,1) through (0,2) are blocked out from when the red pre-constructed structure was loaded in. Since cell (0,3) was not chosen in the final derivation, no derivation can reach token A.

One important thing about this implementation is that the parser only performs these database lookup checks when it is parsing a sentence using the first provided β level from the supertagger (see Section 3.4). This means that if the use of the pre-constructed n-gram structures results in an overall parse not being formed, then as normal, the next β level will be used and the parser will try the sentence again, this time without the additional assistance of the database checks. Each successive β level requires more parse time compared to the previous β level as more categories are assigned to each token, resulting in a greater level of combinatorial explosion of potential parse trees which the parser has to consider. If the parser is left with the default set of β levels, moving on to β2 is a hash decision to make if an overall parse structure could not be constructed because of our databases. This is because the parser never got to consider all of the tokens with their β1 category assignments, as pre-constructed structures may have
been inserted for some of the tokens instead. As such, the set of beta levels used in our experiments are $\{\beta_1, \beta_1, \beta_2, \beta_3, \beta_4\}$, where $\beta_i$ are the default beta levels used by the parser normally. The first $\beta$ level is duplicated so that the parser has a chance to consider the sentence as a whole with the categories assigned using $\beta_1$ if the first time around the parser could not construct a parse for the sentence as a whole.

**Algorithm 3** A rolling hash function is used to ensure efficient database lookups

```plaintext
for all $i = 1$ to $N$ do
  for all $j = 0$ to $N - 1$ do
    if $i == 1$ then
      cell$(j, i).hash \leftarrow$ Hash$(tokens_i)$
    else
      cell$(j, i).hash \leftarrow$ roll cell$(j, i - 1).hash$ with $tokens_{i+j}$
    end if
  end for
end for
```

Algorithms 3, 4, and 5 describe the various implementation changes which need to occur. In order to perform the database lookups efficiently, each token sequence has a hash value associated with it. These are the same hash values used in the keys in the database. Algorithm 3 describes the process of creating these cell hashes, whereby a rolling hash function is used in an $O(n^2)$ loop in order to develop a unique hash value for all substrings of the sentence. This algorithm is performed before the sentence is checked for memoised structures as this checking process is dependant on the hash values.

Algorithm 4 describes the process of checking to see if any substring in the sentence has a memoised structure associated with it. This algorithm is run before CKY is executed as CKY depends on cells being blocked. The cell blocking process happens in this algorithm.

Algorithm 5 describes the updated version of CKY which takes into account the blocked cells in its decision making process. The original version of CKY in the parser is updated with this algorithm.

### 5.6.1 Eisner Constraints

From an implementation point of view, being able to construct and use the forward composed and/or type raised structures discussed in Section 4.5 involves violating one of the constraints proposed in Eisner (1996). The constraints provided by Eisner facilitate the efficient parsing of CCG through the elimination of the “spurious ambiguity” (Wittenburg, 1986) which exists in CCG. The constraint which is violated states that the left child of forward application should not be the result of forward composition.
Algorithm 4  The algorithm for identifying and using memoised analyses

load in the database containing $n$-grams of size $N$

for all $i = N$ to 1 do
  for all $j = 0$ to $N - 1$ do
    if cell($j, i$).hash is not the database then
      continue
    end if

    skip ← false
    for all $x = 1$ to $i$ do
      for all $y = j$ to $i + j - x$ do
        if cell($y, x$).blocked then
          skip ← true
        end if
      end for
    end for
    if skip then
      continue
    end if

    load all pre-constructed derivations for $n$-gram into cell($j, i$)

    for all $x = 1$ to $i$ do
      for all $y = j$ to $i + j - x$ do
        cell($y, x$).blocked ← true
      end for
    end for
  end for
end for

Algorithm 5  Updated version of CKY to take into account the blocked cells

for all $j = 2$ to $N$ do
  for all $i = j - 2$ to 0 do
    for all $k = i + 1$ to $j - 1$ do
      $c_1$ ← cell($i, k - i$)
      $c_2$ ← cell($k, j - k$)
      if $¬(c_1$.blocked $\lor c_2$.blocked) then
        COMBINE($c_1, c_2$)
      end if
    end for
  end for
end for

The C&C parser implements the constraints Eisner proposes, and as such a special rule needs to be added to the parser to allow any chart structures which were loaded from a pre-constructed database to violate the Eisner constraints.
5.7 Summary

We believe that the implementation changes we have made to the parser in order to facilitate us to test our one structure per $n$-gram assertion are highly efficient, with every step being optimised to help us get the maximum throughput. The changes made to the parser only come into effect when running with the first assigned $\beta$ level (combined with when a flag is set), to ensure that our baseline figures are not effected by any changes we have made to the source.
In this chapter we outline the experiments we performed to determine empirically how token-based \( n \)-gram memoisation performs. In Chapter 4 it was proposed through a thorough analysis of the gold-standard data that this idea does not have a good chance of performing well. In this chapter we perform experiments using the highly tuned implementation described in Chapter 5, using both constituent-forming \( n \)-grams and non-constituent-forming \( n \)-grams. We evaluate the performance of both approaches and report the results.

### 6.1 Constituent-forming \( n \)-grams

Through the analysis of the gold standard data presented in Chapter 4, we established that a constituent-only approach to our one structure per \( n \)-gram idea probably would not perform well in practice. Using the implementation described in Chapter 5, we obtained empirical results in order to prove the hypothesis. An initial set of databases were constructed for \( n \)-grams of size 2 and 3 by running the parser over sections 02 to 21 of CCGbank. These databases were then refined using the filtering process described in Section 5.5, with \( X = 0.05 \), \( Y = 15 \), and \( Z = 255 \). The refined databases were used to parse section 00 of CCGbank, the result of which was used for evaluation.

We decided to only construct bigram and trigram databases as larger \( n \)-grams do not appear frequency enough as constituents. Table 6.1 shows constituent statistics about varying sized \( n \)-grams in sections

<table>
<thead>
<tr>
<th>( n )-gram size</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances</td>
<td>335850</td>
<td>632850</td>
<td>731606</td>
<td>736254</td>
</tr>
<tr>
<td>Constituents</td>
<td>79271</td>
<td>65477</td>
<td>43362</td>
<td>29926</td>
</tr>
<tr>
<td>Percentage</td>
<td>23</td>
<td>10</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 6.1:** Relative frequency of varying sized \( n \)-grams
02 to 21 of NP corrected CCGbank (Vadas and Curran, 2008). The rows of this table show the number of $n$-grams in total of that size, irrespective of whether or not they form a constituent, the number of these instances which form a constituent the majority of the time, and then lastly this constituent figure as a percentage of the total number of instances. As expected, the total number of instances increases relative to the size of the $n$-gram, and the number of these instances which primarily form constituents decreases. Using $n$-grams of size larger than 3 in a constituent-based approach has a very low chance of success due to the small number, less than 5%, of primarily-constituent-forming $n$-grams of that size.

The results for our constituent-forming experiment can be seen in Table 6.2. The same experiment was conducted four times, each time using a different parser model. Each group in the table contains the results of four different evaluation criteria. First, the total parse time is reported in seconds, following by the labelled and unlabelled F-score evaluation values, and lastly the percentage of coverage the parser achieved. The first two groups of results in the table were using a grammar model developed using the original CCGbank, and the second two rows use a grammar model developed using the noun phrase corrected version of CCGbank. The baseline column reports parsing performance of section 00 when no databases are used.

<table>
<thead>
<tr>
<th>Model</th>
<th>Baseline</th>
<th>2-gram</th>
<th>3-gram</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSJ LF</td>
<td>85.26</td>
<td>85.26</td>
<td>85.26</td>
</tr>
<tr>
<td>derivs UF</td>
<td>92.01</td>
<td>92.01</td>
<td>92.01</td>
</tr>
<tr>
<td>Cov</td>
<td>98.64</td>
<td>98.64</td>
<td>98.64</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSJ LF</td>
<td>87.40</td>
<td>87.40</td>
<td>87.40</td>
</tr>
<tr>
<td>hybrid UF</td>
<td>93.10</td>
<td>93.10</td>
<td>93.10</td>
</tr>
<tr>
<td>Cov</td>
<td>98.64</td>
<td>98.64</td>
<td>98.64</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP LF</td>
<td>83.60</td>
<td>83.60</td>
<td>83.60</td>
</tr>
<tr>
<td>derivs UF</td>
<td>90.48</td>
<td>90.48</td>
<td>90.48</td>
</tr>
<tr>
<td>Cov</td>
<td>98.54</td>
<td>98.54</td>
<td>98.54</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP LF</td>
<td>85.88</td>
<td>85.88</td>
<td>85.88</td>
</tr>
<tr>
<td>hybrid UF</td>
<td>91.63</td>
<td>91.63</td>
<td>91.63</td>
</tr>
<tr>
<td>Cov</td>
<td>98.54</td>
<td>98.54</td>
<td>98.54</td>
</tr>
</tbody>
</table>

**Table 6.2:** The speed versus performance trade-off for varying sized $n$-grams evaluated on CCGbank 00 using different parsing models. The evaluation attributes are parse time (s), labelled and unlabelled F-score (%), and percentage of sentences covered.
As hypothesised, using only constituent-forming \( n \)-grams in our one structure per \( n \)-gram assertion does not perform well, with no statistically significant change being observed when using either the bigram or trigram databases when compared to the baseline figure. One subtle but important thing to note here is that the performance did not decrease. When using the databases, the parser has to an \( O(n) \) pass over the chart before parsing so that the database checks can be performed. If a pre-constructed \( n \)-gram could be loaded in then the reported parsing time also includes the time taken to load in these structures from disk and insert them into the chart. Even though all of this extra processing is being performed, the overall parsing time is almost the same as the baseline. This means that the parsing time being saved through number of times the memoised analyses result in a derivation being formed, is almost entirely counteracted by two components. Firstly, the overhead of performing this step requires additional computation, which consumes time. Secondly, each time the use of a memoised analysis results in a parse not being able to be constructed, the entire sentence needs to be re-parsed again, consuming a lot more time. Clearly we need a process that gets more hits for the loading cost required.

6.2 Non-constituent-forming \( n \)-grams

Using only constituent-forming \( n \)-grams did not perform well in practice as earlier hypothesised. As suggested in Section 4.5, the next natural direction to turn is to the use non-constituent-forming \( n \)-grams to exploit the combinatory power of CCG. In order to first assess their potential, we manually constructed small databases for each of the most frequently occurring non-constituent-forming \( n \)-grams and their most frequent derivations. These small databases were then used to parse section 00 of CCGbank for evaluation purposes. By placing more than one derivation for each \( n \)-gram in these databases, we are not following our one structure per \( n \)-gram hypothesis. However, we are initially performing these multiple-derivation experiments in order to establish how well this idea works given a small number of derivations. If the performance of databases containing multiple derivations is poor, then there is no point attempting databases containing only one analysis per \( n \)-gram.

Looking back at Table 4.5, the two most frequent bigrams in sections 02 to 21 of CCGbank were \textit{of the} and \textit{in the}, both of which primarily did not form constituents. The occurrence counts for both of these bigrams are more than four times that of occurrence counts for \textit{the company}; the most frequently occurring constituent-forming bigram.
To assess how well non-constituents may perform in our one structure per $n$-gram assertion, we manually constructed a database for each of these two bigrams which contains only their most frequently occurring derivations. By doing this, both of these two databases possess the desirable properties described in Section 4.2. The most frequent derivations for these bigrams were determined through analysing sections 02 to 21 of CCGbank. The derivations we chose to use in these small database instances can be seen in Table 6.3, along with their coverage statistics. Since the performance of the parser will not differ after the first $\beta$ level, the set of $\beta$ values used in these experiments was $\{\beta_1, \beta_1\}$. This means that a particular sentence failed to form a spanning analysis at the first $\beta$ level, then the derivations which exist in the database were not enough to account for the instance of the bigram in the current sentence.

The results for these experiments can be seen in Table 6.4. The columns report the total parse time in seconds, the number of sentences which formed a derivation at the first and second $\beta_1$ levels respectively, the number of sentences which failed to form a derivation after both $\beta$ levels had been attempted, the labelled and unlabelled F-score, and lastly the coverage percentage. The first row reports the baseline performance figures, running the parser without any database attached. The next two rows report performance figures for each of the two manually generated databases. The performance of the of the database was not great, with the parser dropping 0.28% F-score. The performance of the in the database at first glance appears to be slightly better than the baseline, with the parser picking up two
additional sentences while achieving a slight reduction in parsing time. However this change in parsing time is still within the noise limit for the machine. The conclusion here is that again, no significant change was observed compared to the baseline when using these two databases. The last row in the table shows the performance figures when combining both databases into one, in the hope that together they might achieve a statistically significant gain. The result of this was a speed difference within the noise threshold, as well as a further drop in F-score.

As these two bigrams were the most frequently occurring in CCGbank sections 02 to 21, these results shown in Table 6.4 would indicate that storing non-constituents will not provide the required performance. There are a couple of reasons why these non-constituent experiments did not perform well. Firstly, both of the and in the potentially suffer from prepositional phrase attachment issues (see Section 4.5.1), which would lead to a decrease in both parser speed and F-score when using preconstructed databases. Secondly, the C&C parser is already a highly tuned state-of-the-art parser, and as such, provides a very tough baseline to beat. One way to test this theory would be to implement this idea on another existing parser. However this would take a lot of time and resources, and is outside the scope of this thesis. Another potential explanation for this poor performance is that pre-constructing these chart structures for token-based n-grams is just not the way to go. The number of times token-based n-grams reappear in sentences of general text may not be enough to achieve any gain with memoisation of chart structures relative to tokens.

In order to rule out that prepositional phrase attachment is the cause of the poor performance reported in Table 6.4, another experiment was performed. Here we picked the nine most frequent bigrams from sections 02 to 21 of CCGbank which were not likely to suffer from ambiguity issues such as PP attachment. A database was constructed using the top 3 derivations for the most frequent of these bigrams (for the). Then the top 3 derivations were also added for the next most frequent bigram (to the) and the experiment was run again. This process was repeated, each time adding the next most frequent bigram to the database. Like before, these databases were evaluated by parsing section 00 of CCGbank. The results for this experiment can be seen in Table 6.5.

Looking down the rows of the table, it can be observed that the parsing time figures all vary within the error bounds for the machine compared to the baseline. Some additional sentences are gained which previously failed to form a derivation, however the critical figure to note change in across the table is that of the labelled F-score. With each additional row, both the labelled and unlabelled F-score value decreases, with the final row exhibiting a loss of 0.74% F-score compared to the baseline. These results
6.3 Incremental Parsing

Incremental parsing is an alternative parsing technique to chart parsing, in which a parse is constructed in linear time relative to the length of the sentence. Tokens are consumed incrementally, potentially from both ends, and are attached to the current parse tree for the sentence. Research has been performed into the use of CCG as a grammar formalism for incremental parsing (Niv, 1994; Park et al., 2001; Hassan et al., 2008). Incremental CCG parsers assign composed or type raised categories to each token in the sentence which then allows the sentence to be parsed solely from left to right. CCG-based incremental parsings suffer from this same prepositional phrase attachment problem as well.

Hassan et al. (2008) report unlabelled dependency figures of 87.5% F-score when evaluated in section 23 of CCGbank using gold standard POS tags. Without gold standard POS tags, an F-score for the unlabelled dependencies of 86.7% was achieved. This is clearly lower than both the figures reported in Clark and Curran (2007b) and in our tables above. However, considering that incremental parsing is highly subject to the aforementioned attachment issues, their parser performs quite well.

---

**Table 6.5: Performance figures when gradually combining derivations for frequent bigrams which are less likely to suffer from ambiguity issues**

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>$\beta_1$ #1</th>
<th>$\beta_2$ #2</th>
<th># Fail</th>
<th>LF</th>
<th>UF</th>
<th>Cov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>70.4</td>
<td>1804</td>
<td>–</td>
<td>109</td>
<td>87.58</td>
<td>93.14</td>
<td>94.30</td>
</tr>
<tr>
<td>for the</td>
<td>71.8</td>
<td>1803</td>
<td>1</td>
<td>109</td>
<td>87.47</td>
<td>93.03</td>
<td>94.30</td>
</tr>
<tr>
<td>to the</td>
<td>71.6</td>
<td>1802</td>
<td>2</td>
<td>109</td>
<td>87.24</td>
<td>92.83</td>
<td>94.30</td>
</tr>
<tr>
<td>the company</td>
<td>71.1</td>
<td>1803</td>
<td>2</td>
<td>108</td>
<td>87.24</td>
<td>92.83</td>
<td>94.35</td>
</tr>
<tr>
<td>a share</td>
<td>70.1</td>
<td>1803</td>
<td>2</td>
<td>108</td>
<td>87.22</td>
<td>92.82</td>
<td>94.35</td>
</tr>
<tr>
<td>New York</td>
<td>71.2</td>
<td>1803</td>
<td>2</td>
<td>108</td>
<td>87.08</td>
<td>92.70</td>
<td>94.35</td>
</tr>
<tr>
<td>will be</td>
<td>69.5</td>
<td>1804</td>
<td>2</td>
<td>107</td>
<td>87.01</td>
<td>92.66</td>
<td>94.41</td>
</tr>
<tr>
<td>the U.S.</td>
<td>71.0</td>
<td>1806</td>
<td>2</td>
<td>105</td>
<td>86.94</td>
<td>92.60</td>
<td>94.51</td>
</tr>
<tr>
<td>has been</td>
<td>70.9</td>
<td>1804</td>
<td>4</td>
<td>105</td>
<td>86.88</td>
<td>92.58</td>
<td>94.51</td>
</tr>
<tr>
<td>the first</td>
<td>70.9</td>
<td>1804</td>
<td>4</td>
<td>105</td>
<td>86.84</td>
<td>92.53</td>
<td>94.51</td>
</tr>
</tbody>
</table>

are similar to what was achieved in our first non-constituent experiment; parsing times do not change and F-score decreases. As such, it can be concluded that the ambiguity introduced by PP attachment (for example), while it will introduce some level of performance decrease, was not the leading cause of the poor performance.
6.4 Summary

These experiments show that our manually constructed databases did not increase performance, even though they contained within them the most frequent derivations for the most frequent bigrams. If these databases could not increase performance then this indicates that the use of token-based non-constituent-forming \( n \)-grams can not provide the performance results we are after. There

This chapter has comprehensively demonstrated that the one structure per \( n \)-gram strategy cannot work for improving parser speed or accuracy using sections 02 to 21 of CCGbank as training data. The experiments conducted here showed that both constituent and non-constituent based approaches fail to report any statistically significant change in performance. As such, a different approach to satisfy our one structure per \( n \)-gram assertion needs to be established.
Chapter 7

Category-based Analysis of $n$-grams

Now that we have shown that token-based $n$-gram memoisation does not perform well under our one structure per $n$-gram assertion, in this chapter we conduct an analysis of the data in CCGbank to explore the idea of category-based memoisation instead. We define category-based memoisation to be the memoisation of analyses based on their leaf-level CCG categories.

7.1 Category-based Analysis

Through analysis of the gold standard derivations in Chapter 4, it was hypothesised that memoising analysis for frequent $n$-grams based on their tokens would not perform well. Empirical evidence to support this theory was presented in Chapter 6. Aside from the linguistic reasons presented in Chapter 4 outlining why token-based $n$-grams would not work, another contributing factor in their poor performance comes from the sheer number of tokens. Tokens do not reappear frequently enough in text; however the linguistic role which an $n$-gram of tokens plays does repeat frequently. For example, consider the prepositional bigram of the. This bigram plays the same structural role within a sentence as the bigram in the. However, since these are two different bigrams, our original method would treat them differently and have two different derivations memoised.

Instead of memoising $n$-grams based on their token-level frequency, we now explore the idea of memoising $n$-grams based on their CCG categories, irrespective of their tokens. This approach has a number of advantages, one primary one being that there are a lot less CCG categories than there are tokens. While theoretically there are an infinite number of CCG categories, there is only a small finite set of categories which are used in CCGbank; 768 different categories to be precise. Since the supertagger can only assign CCG categories which it knows about to tokens, this means that any token the supertagger is asked to tag will be assigned some subset of the categories which appear in CCGbank.
7.2 Constituent-forming Category Spans

In this chapter, we present an analysis of this idea in a similar manner to Chapter 4. For all analysis and experiments from this point onwards, the original version of CCGbank is being used instead of the NP corrected version. The motivation to use the noun phrase corrected version of CCGbank during our token-level analysis (Chapter 4) does not exist here. The correct internal structure of noun phrases in the corpus is not an issue for a purely category-based analysis. As such, the analysis performed in this chapter is performed using the canonical version of CCGbank (Hockenmaier, 2003).

As with our analysis of n-grams in Chapter 4, the first set of analysis we will perform is on what can be achieved looking at constituent-forming sequences of neighbouring categories. Table 7.1 lists the 25 most frequent categories assigned to the heads of all non-leaf nodes across all derivations in CCGbank sections 02 to 21.

<table>
<thead>
<tr>
<th>Head Category</th>
<th>Frequency</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NP$</td>
<td>195028</td>
<td>20.87</td>
</tr>
<tr>
<td>$N$</td>
<td>173421</td>
<td>18.55</td>
</tr>
<tr>
<td>$S[\text{decl}]$</td>
<td>130886</td>
<td>14.00</td>
</tr>
<tr>
<td>$S[\text{decl}]\backslash NP$</td>
<td>92630</td>
<td>9.91</td>
</tr>
<tr>
<td>$NP\backslash NP$</td>
<td>60889</td>
<td>6.51</td>
</tr>
<tr>
<td>$(S\backslash NP)\backslash (S\backslash NP)$</td>
<td>38177</td>
<td>4.08</td>
</tr>
<tr>
<td>$S[b]\backslash NP$</td>
<td>34012</td>
<td>3.64</td>
</tr>
<tr>
<td>$PP$</td>
<td>20085</td>
<td>2.15</td>
</tr>
<tr>
<td>$NP[\text{conj}]$</td>
<td>17627</td>
<td>1.89</td>
</tr>
<tr>
<td>$S[\text{ng}]\backslash NP$</td>
<td>15529</td>
<td>1.66</td>
</tr>
<tr>
<td>$S[\text{pss}]\backslash NP$</td>
<td>13824</td>
<td>1.48</td>
</tr>
<tr>
<td>$S[\text{to}]\backslash NP$</td>
<td>13604</td>
<td>1.46</td>
</tr>
<tr>
<td>$N/N$</td>
<td>10136</td>
<td>1.08</td>
</tr>
<tr>
<td>$S/S$</td>
<td>9980</td>
<td>1.07</td>
</tr>
<tr>
<td>$NP[\text{nb}]/N$</td>
<td>9158</td>
<td>0.98</td>
</tr>
<tr>
<td>$S[\text{decl}]\backslash S[\text{decl}]$</td>
<td>8476</td>
<td>0.91</td>
</tr>
<tr>
<td>$S[\text{pt}]\backslash NP$</td>
<td>8036</td>
<td>0.86</td>
</tr>
<tr>
<td>$S[\text{adj}]\backslash NP$</td>
<td>8026</td>
<td>0.86</td>
</tr>
<tr>
<td>$N[\text{num}]$</td>
<td>5396</td>
<td>0.58</td>
</tr>
<tr>
<td>$S[\text{decl}][\text{conj}]$</td>
<td>5131</td>
<td>0.55</td>
</tr>
<tr>
<td>$S[\text{em}]$</td>
<td>4717</td>
<td>0.50</td>
</tr>
<tr>
<td>$N[\text{conj}]$</td>
<td>3830</td>
<td>0.41</td>
</tr>
<tr>
<td>$(S[\text{decl}]\backslash NP)/NP$</td>
<td>3323</td>
<td>0.36</td>
</tr>
<tr>
<td>$(S[\text{decl}]\backslash NP)/(S[b]\backslash NP)$</td>
<td>3244</td>
<td>0.35</td>
</tr>
<tr>
<td>$S[\text{decl}]\backslash NP[\text{conj}]$</td>
<td>2805</td>
<td>0.30</td>
</tr>
</tbody>
</table>

Table 7.1: The 25 most frequent categories assigned to the non-leaf nodes across all derivations in CCGbank sections 02 to 21
7.2 Constituent-forming Category Spans

Table 7.2: The 10 most frequent categories, considered without their features, which are assigned to the non-leaf nodes across all derivations in CCGbank sections 02 to 21.

<table>
<thead>
<tr>
<th>Head Category</th>
<th>Frequency #</th>
<th>Coverage %</th>
<th>Coverage Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>212655</td>
<td>22.75</td>
<td>212655</td>
</tr>
<tr>
<td>S\NP</td>
<td>192605</td>
<td>20.61</td>
<td>405260</td>
</tr>
<tr>
<td>N</td>
<td>182647</td>
<td>19.54</td>
<td>587907</td>
</tr>
<tr>
<td>S</td>
<td>144620</td>
<td>15.47</td>
<td>732527</td>
</tr>
<tr>
<td>NP\NP</td>
<td>61357</td>
<td>6.56</td>
<td>793884</td>
</tr>
<tr>
<td>(S\NP)(S\NP)</td>
<td>40000</td>
<td>4.28</td>
<td>833884</td>
</tr>
<tr>
<td>PP</td>
<td>20316</td>
<td>2.17</td>
<td>854200</td>
</tr>
<tr>
<td>(S\NP)(S\NP)</td>
<td>12055</td>
<td>1.29</td>
<td>866255</td>
</tr>
<tr>
<td>N/N</td>
<td>11111</td>
<td>1.19</td>
<td>877366</td>
</tr>
<tr>
<td>S/S</td>
<td>10948</td>
<td>1.17</td>
<td>888314</td>
</tr>
</tbody>
</table>

Table 7.2 was generated in the same way as Table 7.1, except that this time categories were considered without their features. As this table shows, irrespective of features, the top 10 most frequent categories account for just over 95% of all cases. The implication of this is that if the majority of the category spans which exist below all 10 of these head categories are mostly unambiguous, then the memoisation of these spans has a great potential to decrease the overall parsing time. This is because around 95% of all category spans may be able to be loaded up from a memoised form.

Table 7.3 lists the 10 most frequent constituent-forming spans of categories which end up being headed by a N. As discussed earlier, the use of the original version of CCGbank (not the NP corrected version) is being used to our advantage, as a large number of additional “rolling” noun and NP modifier structures will exist for all flat-structured noun phrases. This can be seen in Table 7.3, where the majority of the rows are stacked nominal modifying structures of various forms.

The number of constituent-forming category spans which are headed by a particular category is potentially very large due to the combinatorial explosion property of CCG. The second column of Table 7.4 shows the number of unique constituent-forming category spans headed by varying categories.
7.2 Constituent-forming Category Spans

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Constituent sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>67250</td>
<td>N/N N</td>
</tr>
<tr>
<td>18210</td>
<td>N/N N/N N</td>
</tr>
<tr>
<td>4495</td>
<td>N/N N/N N/N N</td>
</tr>
<tr>
<td>3408</td>
<td>N/N[num] N/N N[num]</td>
</tr>
<tr>
<td>2943</td>
<td>(N/N)/(N/N) N/N N</td>
</tr>
<tr>
<td>1996</td>
<td>N/N[num] N[num]</td>
</tr>
<tr>
<td>1417</td>
<td>N conj N</td>
</tr>
<tr>
<td>1106</td>
<td>N/N N/N N/N N/N N</td>
</tr>
<tr>
<td>850</td>
<td>N/N conj N/N N</td>
</tr>
<tr>
<td>696</td>
<td>(N/N)/(N/N) N/N N/N N</td>
</tr>
</tbody>
</table>

Table 7.3: The 10 most frequent constituent-forming category spans which result in being headed by a N

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>25122</td>
<td>735</td>
<td>404</td>
<td>266</td>
<td>206</td>
</tr>
<tr>
<td>N</td>
<td>2462</td>
<td>204</td>
<td>135</td>
<td>103</td>
<td>88</td>
</tr>
<tr>
<td>S[</td>
<td>dec</td>
<td>]</td>
<td>36613</td>
<td>51</td>
<td>15</td>
</tr>
<tr>
<td>S[</td>
<td>dec</td>
<td>]\NP</td>
<td>34167</td>
<td>333</td>
<td>165</td>
</tr>
<tr>
<td>NP\NP</td>
<td>14772</td>
<td>428</td>
<td>224</td>
<td>167</td>
<td>145</td>
</tr>
<tr>
<td>(S\NP)(S\NP)</td>
<td>11340</td>
<td>274</td>
<td>155</td>
<td>120</td>
<td>97</td>
</tr>
<tr>
<td>S[b]\NP</td>
<td>12019</td>
<td>218</td>
<td>120</td>
<td>77</td>
<td>58</td>
</tr>
<tr>
<td>PP</td>
<td>5538</td>
<td>148</td>
<td>90</td>
<td>69</td>
<td>52</td>
</tr>
<tr>
<td>Total</td>
<td>142033</td>
<td>2391</td>
<td>1308</td>
<td>923</td>
<td>730</td>
</tr>
</tbody>
</table>

Table 7.4: The total number of constituent-forming spans with varying heads with different frequency cutoff values

As before, these figures were generated over all the derivations in CCGbank sections 02 to 21. This second column shows that just the eight listed category heads listed in the first column, 142033 unique constituent-forming category spans exist. The remainder of the columns in Table 7.4 show the number of unique constituent-forming category spans when a simple frequency cutoff value is applied. By setting the cutoff predicate to a minimum of 5 occurrences, the total number of unique spans is dramatically reduced to 2391; 1.9% of the original number of unique spans.

Analysis of these cutoff values provides useful statistics for our memoisation process. There is a very large tail in the full lists of constituent-forming category spans as indicated by the drop-off encountered by setting a frequency cutoff value to 5. Memoising structures which only appear very infrequently, once or twice for example, would introduce error when they are used as memoised analyses.
7.3 Ambiguity in Category Spans

As was the case with our token-based \( n \)-gram analysis, the memoisation of constituent-forming structures is not without ambiguity. If we memoised all constituent-forming category spans for certain category heads with some frequency cutoff value, then a lot derivations will be incorrectly formed when using the memoised analyses. When a memoised span is being reused in the parsing of some sentence, each category span is subject to different ambiguity depending on the categories immediately to the left and right of where the span is inserted. For example, Table 7.3 shows that the category span \( N \text{ conj } N \) is the seventh most frequent span in CCGbank sections 02 to 21 which results in an \( N \) at the top of the derivation. As such, a valid memoisation of this category span would be in the context of the following sentence, and is highlighted in red.

\[
\begin{array}{ccc}
\text{George} & \text{ate} & \text{fish and chips} \\
N & (S[decl]\NP)/NP & N \text{ conj } N \\
NP & \quad & N \phi N \\
NP & \quad & (S[decl]\NP)> \\
S[decl] & \quad & <
\end{array}
\]

At a later point in time, the parser is then asked to parse the sentence George ate spicy fish and chips, where the fish was more spicy than in the original sentence. The correct derivation for this sentence is the following, where spicy is acting as a modifier of fish.

\[
\begin{array}{ccc}
\text{George} & \text{ate} & \text{spicy fish and chips} \\
N & (S[decl]\NP)/NP & N/N \text{ conj } N \\
NP & \quad & N \phi N \\
NP & \quad & (S[decl]\NP)> \\
S[decl] & \quad & <
\end{array}
\]

When this new sentence is given to the parser however, the parser will realise that the category span \( N \text{ conj } N \) is memoised, and will insert the memoised structure from before in place in this new sentence (again highlighted in red). As the derivation below shows, this results in an incorrect analysis.
being constructed for the sentence. The use of the memoised form of \( N \text{ conj } N \) results in both the fish and the chips becoming spicy, instead of just the fish.

Any memoised span of categories which ends in an \( N \) or \( NP \) is subject to this form of ambiguity where modifiers are being stacked up to the left of the span. This includes modifiers such as \( N/N \), modifiers of modifiers such as \( (N/N)/(N/N) \), modifiers of modifiers of modifiers such as \( ((N/N)/(N/N))/((N/N)/(N/N)) \) and so on; there is no upper bound on the number of stacked modifiers which might exist to the left of the span.

Scope ambiguity issues such as this are not limited to attachment decisions on the left-hand side of the spans. Consider the following correct derivation.

The category sequence \( NP[nt]/N \) \( N \) is the most frequent constituent-forming sequence which results in a \( NP \) at the head of the derivation. If we were to use a memoised version of this category span in this previous derivation, the incorrect analysis shown below will be constructed. In this incorrect
derivation, to remain silent modifies have the right, instead of the whole sentence being a right-branching tree structure.

These are just two example situations where the use of a memoised constituent-forming category span resulted in the incorrect derivation being constructed. Table 7.5 shows the 10 most frequent contexts which result in a incorrect derivation would be constructed if the frequent span in the context is used. The frequent spans are highlighted in red. The most frequent item in this table is the noun version of the situation observed in our previous derivation. This first row states that whenever the sequence $N/N N$ is seen directly before a $(S[to]\NP)/(S[b]\NP)$, a memoised version of $N/N N$ should not be used as the $(S[to]\NP)/(S[b]\NP)$ needs to end up combining with the $N$ before the $NP/N$ does. The use of a memoised form for $N/N N$ would bind the $N/N$ and $N$ first, thus resulting in an incorrect derivation being constructed.

Table 7.6 also lists the top 10 non-constituent-forming category spans, but for frequent category spans headed by an $N$ instead of an $N$. By looking at these two tables some common trends can be identified. $N$ and $NP$ ending spans are ambiguous when they appear before prepositional categories; $(NP\NP)/NP$ and $(S[to]\NP)/(S[b]\NP)$ for example. This was the case in the to remain silent example discussed earlier. $N$ and $NP$ starting spans are also ambiguous when they appear before nounal modifiers such as $N/N$. This was the case in our first example about eating fish and chips.

Clearly these ambiguous cases will cause problems when trying to reuse memoised spans, as illustrated in our previous example derivations. One way we can attempt to alleviate this issue is through a slight change to how the memoised structures are used. As well as constructing a database which maps category span to memoised analysis, a list is also kept containing all of these frequently occurring contexts where the use of a memoised category span would result in an incorrect derivation being constructed.
As was the case previously when performing our token-based memoisation, the chart is examined top to bottom, left to right, looking for memoised structures to insert. In the list of bad sequences did not...
7.3 Ambiguity in Category Spans

Figure 7.1: Longer category spans which indicate ambiguity place a “no paste” restriction on the cells which they cover, allowing the correct derivation to be constructed exist, both of the cells highlighted in green in the first image of Figure 7.1 would have memoised analyses inserted, as both of their category spans exist in the database. With the addition of our list of bad sequences, the top path of images in this figure are taken. The cell highlighted in blue is identified as existing in the bad list. Consequently, all of the cells which are covered by this blue cell are marked with a marker indicating that no memoised structures can be inserted into these cells. These cells however still participate in CKY as normal. This marker is indicated by the blue hashing in the second image. Cells (4, 1) and (5, 1) are blocked out from CKY in this second image as the memoised structure for NP/N N was allowed to be inserted; no category sequence in the bad sequences list stopped this from happening. Once this pass over the chart has been completed, CKY is performed, and the derivation in the top right image is constructed.

The images in the bottom row of this figure illustrate what would happen if the list of bad sequences did not exist. In the center bottom image, both of the memoised forms would be inserted and have their corresponding cells blocked out from CKY. The bottom right image shows the resultant parse tree for the sentence, which is clearly different (and incorrect) to the top right image. Note that in the top right image, cell (0, 3) is not an internal node in the parse tree, whereas it is in the bottom right image.
7.4 Summary

In this chapter we have shown that a category-based approach to the memoisation of analyses has a lot of potential to perform well in practice. Firstly, the number of different categories which exist in the training data (CCGbank sections 02 to 21) is far less than the number of different tokens. We hypothesise that the number of distinct tokens was one contributing factor as to why our previous approach did not perform well in practice, and so the use of category-based sequences will alleviate this issue. Another reason motivating the choice of category-based spans is that while category-based spans suffer from ambiguity just as the token-based spans did, it is easier to identify all ambiguous cases due to the smaller number of contexts (due to the smaller number of categories relative to number of tokens).
In Chapter 7 we outline an alternative approach to testing our one structure per \( n \)-gram hypothesis, whereby instead of memoising analyses based on tokens, we memoise analyses based on their CCG categories. This approach has a number of factors motivating it as outlined earlier. In this chapter we present experiments which we performed in order to test how well this new category-based approach works in practice.

8.1 Implementation Changes

In order to use category sequences in our one structure per \( n \)-gram assertion, some modifications need to be performed to our original implementation described in Chapter 5. Firstly, the memoised databases will be manually constructed instead of being “learnt” automatically through frequency analysis performed over parsed CCGbank sections 02 to 21. As such, a method of being able to manually specify analyses which should be used in the database needs to be developed.

Secondly, the list of ambiguous category sequences (Section 7.3) needs to be accounted for. If a cell yields a category sequence which exists in this list, all of its covered cells need to be marked to indicate they cannot have a memoised form inserted into them. Algorithm 4 needs to be updated to allow for this. Algorithm 6 is an updated version of this previous algorithm, showing in pseudocode how this “no paste” marker works relative to the cell blocking.

There is also one important issue with implementing this idea that is specific to the way the C&C parser works. When the supertagger is asked to assign supertags all of the tokens in a sentence, it assigned a set of supertags to each token. Each supertag in these sets has a probability associated with it, which indicates how confident the supertagger is about the decision to assign the supertag to the token. Our analysis of this category-based approach presented in Chapter 7, assumed that each token in the
sentence has only one supertag assigned to it. In order to minimise any additional ambiguity which could be introduced by tokens which are assigned more than one supertag at the first $\beta$ level, any cell with more than one supertag is marked with the aforementioned “no paste” marker before the entire chart is analysed, to indicate that no memoised structure can be inserted into it. In addition to this, if a cell has only one supertag but the probability of this supertag is less than a given parameter, the cell is also marked in this manner. If the supertagger is not confident about its decision then we do not want this supertag to interfere with our one structure per $n$-gram assertions. For our experiments, this probability parameter was set to 95%, indicating that the supertagger has to be at least 95% confident in its decision of assigning the supertag if we are to allow the cell to participate in our one structure per $n$-gram hypothesis.

8.2 Experimental Procedure

As outlined in Chapter 7, the use of category spans to test our one structure per $n$-gram hypothesis requires both analyses to memoise as well as a list of “no paste” sequences which forbid a memoised structure being inserted into the chart (see Section 7.3).

Our experimental approach was as follows. We started off by going down the sorted list of derivation heads shown in Table 7.1, selecting incremental groups of category heads. For example, we first considered just \{NP\}, then \{NP, N\}, then \{NP, N, S[dcl]\}, etc. For each of these category sets, all possible combinations of two different parameters were then tested. The first parameter was the cutoff frequency for the constituent-forming category spans. It is these spans whose analyses are memoised. The second parameter is the cutoff frequency for the non-constituent-forming category spans; the spans which block a memoised category span from being inserted. This process is described in Algorithm 7.

8.3 Results

As the experimental procedure illustrates, a large number of figures were returned as the result of this experimentation process. Here we will present only a select few of the result sets. Each of the tables of figures have their category span datasets fixed with a minimum cutoff frequency for the spans also fixed. The cutoff column in each table shows the varying nopaste minimum frequency cutoff value.
Algorithm 6 The updated algorithm for the use of the memoised analyses contained within the database

for all \( i = N \) to 1 do
    for all \( j = 0 \) to \( N - 1 \) do
        if cell\((j, i)\).hash is not the database or in the bad sequences list then
            continue
        end if

        skip ← false
        for all \( x = 1 \) to \( i \) do
            for all \( y = j \) to \( i + j - x \) do
                if cell\((y, x)\).nopaste then
                    skip ← true
                end if
            end for
        end for
        if skip then
            continue
        end if

        blocked ← false
        if cell\((j, i)\).hash is not in the bad sequences list then
            load all pre-constructed derivations for n-gram into cell\((j, i)\)
            blocked ← true
        end if
        for all \( x = 1 \) to \( i \) do
            for all \( y = j \) to \( i + j - x \) do
                cell\((y, x)\).blocked ← blocked
                cell\((y, x)\).nopaste ← true
            end for
        end for
    end for
end for

Algorithm 7 Experimental procedure

for all sequential category subset \( C \) from the sorted list of heads do
    for span cutoff value \( s \in \langle 10, 30, 50, 100 \rangle \) do
        \( G \leftarrow \) all constituent-forming spans headed by any category in \( C \) with minimum frequency \( s \)
        for nopaste cutoff value \( n \in \langle 5, 10, 15, 20, 30, 40, 50, 100, 200, 400, 800 \rangle \) do
            \( N \leftarrow \) all span-blocking sequences \( \forall g \in G \) with minimum frequency \( n \)
            parse CCGbank section 00 using \( G \) and \( N \), and perform evaluation
        end for
    end for
end for
### Table 8.1: Parsing time versus performance trade off illustrated for varying nopaste cutoff values. Both $N$ and $NP$ headed category spans were used with a minimum frequency cutoff of 50.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>Time (s)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Ours</td>
</tr>
<tr>
<td>5</td>
<td>71.4</td>
<td>65.0</td>
</tr>
<tr>
<td>10</td>
<td>71.4</td>
<td>64.3</td>
</tr>
<tr>
<td>15</td>
<td>71.4</td>
<td>63.8</td>
</tr>
<tr>
<td>20</td>
<td>71.4</td>
<td>62.3</td>
</tr>
<tr>
<td>30</td>
<td>71.4</td>
<td>62.6</td>
</tr>
<tr>
<td>40</td>
<td>71.4</td>
<td>61.7</td>
</tr>
<tr>
<td>50</td>
<td>71.4</td>
<td>61.8</td>
</tr>
<tr>
<td>100</td>
<td>71.4</td>
<td>61.3</td>
</tr>
<tr>
<td>200</td>
<td>71.4</td>
<td>61.0</td>
</tr>
<tr>
<td>400</td>
<td>71.4</td>
<td>60.7</td>
</tr>
<tr>
<td>800</td>
<td>71.4</td>
<td>60.4</td>
</tr>
</tbody>
</table>

### Table 8.2: Parsing time versus performance trade off illustrated for varying nopaste cutoff values. $N$, $NP$, and $PP$ category spans were used with a minimum frequency cutoff of 50.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>Time (s)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Ours</td>
</tr>
<tr>
<td>5</td>
<td>71.4</td>
<td>64.6</td>
</tr>
<tr>
<td>10</td>
<td>71.4</td>
<td>64.5</td>
</tr>
<tr>
<td>15</td>
<td>71.4</td>
<td>63.8</td>
</tr>
<tr>
<td>20</td>
<td>71.4</td>
<td>62.6</td>
</tr>
<tr>
<td>30</td>
<td>71.4</td>
<td>62.4</td>
</tr>
<tr>
<td>40</td>
<td>71.4</td>
<td>61.7</td>
</tr>
<tr>
<td>50</td>
<td>71.4</td>
<td>61.7</td>
</tr>
<tr>
<td>100</td>
<td>71.4</td>
<td>61.8</td>
</tr>
<tr>
<td>200</td>
<td>71.4</td>
<td>61.2</td>
</tr>
<tr>
<td>400</td>
<td>71.4</td>
<td>60.7</td>
</tr>
<tr>
<td>800</td>
<td>71.4</td>
<td>60.4</td>
</tr>
</tbody>
</table>
### Table 8.3: Parsing time versus performance trade off illustrated for varying nopaste
cutoff values. *N, NP, PP, and *(S\(\backslash\)NP)\(\backslash\)(S\(\backslash\)NP) category spans were used with a
minimum frequency cutoff of 50.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>Time (s)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Ours</td>
</tr>
<tr>
<td>5</td>
<td>71.4</td>
<td>68.9</td>
</tr>
<tr>
<td>10</td>
<td>71.4</td>
<td>64.3</td>
</tr>
<tr>
<td>15</td>
<td>71.4</td>
<td>63.6</td>
</tr>
<tr>
<td>20</td>
<td>71.4</td>
<td>62.3</td>
</tr>
<tr>
<td>30</td>
<td>71.4</td>
<td>62.4</td>
</tr>
<tr>
<td>40</td>
<td>71.4</td>
<td>62.3</td>
</tr>
<tr>
<td>50</td>
<td>71.4</td>
<td>61.3</td>
</tr>
<tr>
<td>100</td>
<td>71.4</td>
<td>61.0</td>
</tr>
<tr>
<td>200</td>
<td>71.4</td>
<td>60.8</td>
</tr>
<tr>
<td>400</td>
<td>71.4</td>
<td>60.4</td>
</tr>
<tr>
<td>800</td>
<td>71.4</td>
<td>59.9</td>
</tr>
</tbody>
</table>

### Table 8.4: Parsing time versus performance trade off illustrated for varying nopaste
cutoff values. *N and NP category spans were used with a minimum frequency cutoff of 20.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>Time (s)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Ours</td>
</tr>
<tr>
<td>5</td>
<td>71.4</td>
<td>64.3</td>
</tr>
<tr>
<td>10</td>
<td>71.4</td>
<td>63.7</td>
</tr>
<tr>
<td>15</td>
<td>71.4</td>
<td>63.2</td>
</tr>
<tr>
<td>20</td>
<td>71.4</td>
<td>61.3</td>
</tr>
<tr>
<td>30</td>
<td>71.4</td>
<td>61.1</td>
</tr>
<tr>
<td>40</td>
<td>71.4</td>
<td>60.4</td>
</tr>
<tr>
<td>50</td>
<td>71.4</td>
<td>60.4</td>
</tr>
<tr>
<td>100</td>
<td>71.4</td>
<td>60.3</td>
</tr>
<tr>
<td>200</td>
<td>71.4</td>
<td>59.9</td>
</tr>
<tr>
<td>400</td>
<td>71.4</td>
<td>60.1</td>
</tr>
<tr>
<td>800</td>
<td>71.4</td>
<td>59.2</td>
</tr>
</tbody>
</table>
8.4 Final Results

The results presented in the previous section are somewhat positive. A direct relationship between the speed increase of the parser and the amount of F-score that is sacrificed can easily be seen. In order to be able to quote some final performance figures for this technique, our databases need to be evaluated against section 23 of CCGbank. We decided to evaluate two different databases to provide contrasting figures; one set of figures for the best performing database with the smallest nopaste cutoff value, and

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>Time (s)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Ours</td>
</tr>
<tr>
<td>5</td>
<td>71.4</td>
<td>64.1</td>
</tr>
<tr>
<td>10</td>
<td>71.4</td>
<td>63.8</td>
</tr>
<tr>
<td>15</td>
<td>71.4</td>
<td>63.2</td>
</tr>
<tr>
<td>20</td>
<td>71.4</td>
<td>61.4</td>
</tr>
<tr>
<td>30</td>
<td>71.4</td>
<td>61.1</td>
</tr>
<tr>
<td>40</td>
<td>71.4</td>
<td>60.3</td>
</tr>
<tr>
<td>50</td>
<td>71.4</td>
<td>60.8</td>
</tr>
<tr>
<td>100</td>
<td>71.4</td>
<td>60.3</td>
</tr>
<tr>
<td>200</td>
<td>71.4</td>
<td>60.0</td>
</tr>
<tr>
<td>400</td>
<td>71.4</td>
<td>59.5</td>
</tr>
<tr>
<td>800</td>
<td>71.4</td>
<td>59.5</td>
</tr>
</tbody>
</table>

Table 8.5: Parsing time versus performance trade off illustrated for varying nopaste cutoff values. N, NP, and PP category spans were used with a minimum frequency cutoff of 20.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>Time (s)</th>
<th>F-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baseline</td>
<td>Ours</td>
</tr>
<tr>
<td>5</td>
<td>71.4</td>
<td>64.2</td>
</tr>
<tr>
<td>10</td>
<td>71.4</td>
<td>63.7</td>
</tr>
<tr>
<td>15</td>
<td>71.4</td>
<td>62.9</td>
</tr>
<tr>
<td>20</td>
<td>71.4</td>
<td>61.2</td>
</tr>
<tr>
<td>30</td>
<td>71.4</td>
<td>61.4</td>
</tr>
<tr>
<td>40</td>
<td>71.4</td>
<td>60.0</td>
</tr>
<tr>
<td>50</td>
<td>71.4</td>
<td>59.9</td>
</tr>
<tr>
<td>100</td>
<td>71.4</td>
<td>60.0</td>
</tr>
<tr>
<td>200</td>
<td>71.4</td>
<td>59.7</td>
</tr>
<tr>
<td>400</td>
<td>71.4</td>
<td>58.9</td>
</tr>
<tr>
<td>800</td>
<td>71.4</td>
<td>58.6</td>
</tr>
</tbody>
</table>

Table 8.6: Parsing time versus performance trade off illustrated for varying nopaste cutoff values. N, NP, PP, and (S\NP)\(S\NP) category spans were used with a minimum frequency cutoff of 20.
one set of figures for the best performing database with the largest nopaste cutoff value. These two figures should provide us with the two extremes for how well our one structure per \( n \)-gram hypothesis works using this category-span implementation.

The best performing database with a nopaste cutoff value of 5 was observed in Table 8.4; the experiment were only \( N \) and \( NP \) headed category spans were considered, with the minimum occurrence frequency set to 20. During our initial experiments, this database yielded an overall parsing time decrease of 9.9% with a loss of 0.55% labelled F-score. The results of evaluating this database on section 23 of CCGbank are shown in Table 8.7. The database performed better on the test set than it did on the development set, resulting in a 15% decrease in the overall parsing time with a loss of 0.40% F-score.

The best performing database with a nopaste cutoff value of 800 was also observed in Table 8.4. This database yielded a 17.1% decrease in the overall parsing time at a loss of 1.06% labelled F-score in our initial experiments. The results for evaluating this database on section 23 of CCGbank are shown in Table 8.8. As was the case in Table 8.7, the database here performed better on section 23 than on the development set. An overall speed decrease of 18.76% was recorded at a cost of 0.90% labelled F-score.

### 8.5 Summary

In this chapter we performed experiments to test whether the memoisation of analyses based on frequently occurring CCG category sequences allows our one structure per \( n \)-gram hypothesis to improve parser performance. The results of our initial experiments evaluating on CCGbank section 00 indicated

<table>
<thead>
<tr>
<th></th>
<th>Without</th>
<th>With</th>
<th>( \Delta )</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>75.37</td>
<td>64.07</td>
<td>11.30</td>
<td>14.99%</td>
</tr>
<tr>
<td>LF</td>
<td>88.13</td>
<td>87.73</td>
<td>-0.40</td>
<td>-</td>
</tr>
<tr>
<td>UF</td>
<td>93.27</td>
<td>92.92</td>
<td>-0.35</td>
<td>-</td>
</tr>
<tr>
<td>Cov</td>
<td>95.26</td>
<td>95.35</td>
<td>0.09</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 8.7:** Section 23 evaluation for the database used in the first row of Table 8.4

<table>
<thead>
<tr>
<th></th>
<th>Without</th>
<th>With</th>
<th>( \Delta )</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>74.41</td>
<td>60.45</td>
<td>13.96</td>
<td>18.76%</td>
</tr>
<tr>
<td>LF</td>
<td>88.13</td>
<td>87.23</td>
<td>-0.90</td>
<td>-</td>
</tr>
<tr>
<td>UF</td>
<td>93.27</td>
<td>92.42</td>
<td>-0.85</td>
<td>-</td>
</tr>
<tr>
<td>Cov</td>
<td>95.26</td>
<td>95.35</td>
<td>0.09</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 8.8:** Section 23 evaluation for the database used in the last row of Table 8.4
that some parser throughput can be gained using such an assertion, but it comes at a loss of F-score. After selecting the best performing set of parameters from our initial experiments, we performed the final evaluation of our system, evaluating on section 23 (the test set) of CCGbank. This evaluation returned a 15% improvement in parser speed at a loss of 0.4% F-score. While 0.4% is a small percentage, in terms of parser performance, this unfortunately is a large number.
The hypothesis which we aimed to explore in this thesis was whether or not we can exploit the naturally occurring redundancy in text in order to speed up parsing. In a “one sense per discourse” (Gale et al., 1992) manner, we put forward a “one structure per n-gram” hypothesis, stating that every time certain n-grams of text are seen, they always appear with the same syntactic structure. This thesis explores different methods of attempting to use this hypothesis to improve the performance of the parser. Ideally the speed of the parser should increase and the F-score should remain the same or drop only slightly.

9.1 Future Work

There are a number of directions in which the work presented in this thesis can be explored in future work. Firstly, it would be interesting to work out what are the best $\beta$ levels for the supertagger to work with under our one structure per n-gram assertion. For all of our experiments and analysis, the first default $\beta$ level was used without any experimentation to determine whether or not a better $\beta$ value exists for our task.

In Section 5.6, and specifically around Figure 5.6, we stated and showed by example why two overlapping analyses for n-grams could not both be inserted into the chart. This decision was made as there is no obviously simple way to allow for this to happen problem free. No additional time was spent looking into whether or not overlapping analyses can be inserted. This problem provides an interesting area for future work as if overlapping structures are possible, there is the potential for a greater speed gain, as potentially more of the chart can be blocked out before CKY runs (see Chapter 5 for details on cell blocking).

All of our experiments used the memoised databases in a very simplistic manner. The chart was examined top to bottom, left to right, and if the yield of the current cell has a memoised form with in the
database, this form was loaded into the derivation. The decision to load in a memoised analyses for an
\( n \)-gram does not take into account the context of the sentence, or any features of the \( n \)-grams surround-
ings. One idea for future work here is to research into knowing when to use \( n \)-grams and when not to use them. During the JHU summer workshop this idea was initially explored through the use of a clas-
sifier to recognise constituents. While the results are not published anywhere yet, the results achieved
during the workshop were positive, indicating that this approach has the potential to further increase the
performance of the technique outlined in this thesis.

Section 4.7 proposed a future research area which was not going to be explored in this thesis. During
the analysis of frequent \( n \)-grams, an observation was made that pattern-based constituents occurred a lot
more frequently than token-based constituents. See Section 4.7 for further details.

Another possible extension to the work presented in this thesis is to investigate whether or not an inte-
gration between the parser and the supertagger is possible, in order to achieve faster parsing times. If
the parser-supertagger knows that some \( n \)-gram in the sentence will be headed by an \( NP \) for example
due to the existence of a memoised form in the database, the supertagger could use this knowledge to its
advantage. By considering all of the tokens contained within this \( n \)-gram as one token with the supertag
\( NP \), then the supertagger is able to see further ahead in the sentence when it is making tagging decisions.

Being able to see further ahead potentially leads to more accurate supertags, which in turn, potentially
leads to faster parsing.

One last future work idea motivated by the work presented in this thesis is that new features could be
added to the parsers statistical model which exploit the rolling \( n \)-gram hashes we introduced in this
thesis. We introduced a method for efficiently creating unique hash values for all subsequences of
category spans within a sentence (subsequences of any cell attribute in the general case). These specific
hash values may be able to be used as features due to the rolling nature of the hash function. This is a
very interesting (and technical) area to look into in the future.

\section{Summary and Contribution}

In Chapter 2 we started off by presented background material on parsing and the current state of natural
language parsing. We then explain what “one sense per discourse” means, and how it has been success-
fully applied to many other NLP tasks. This section is concluded with an outline of how we plan to use
our “one structure per $n$-gram” hypothesis in a “one sense per discourse” manner, through the assertion that for certain $n$-grams, there exists only one syntactic structure.

Before Chapter 3, everything we had discussed was grammar formalism and parser independent. In this chapter we outline our grammar formalism of choice for making our one structure per $n$-gram hypothesis; CCG. This chapter then continues to outline in detail how CCG is used for statistical parsing, the data we will be using for our experiments and analysis, as well as outlining the CCG parser we will be using; the state-of-the-art C&C parser.

Chapter 4 contained a thorough analysis of one approach we undertook in order to test our one structure per $n$-gram hypothesis. In this chapter, we first investigated how effective the memoisation of constituent-forming analyses of $n$-grams would be. The analysis indicated that the idea would not perform well due to the small number of frequently occurring constituent-forming $n$-grams. Analysis was then performed on non-constituent-forming derivations. We observed that these occurred a lot more frequently than their constituent-forming counterparts, but suffered from a lot more ambiguity. It was concluded that an implementation and experiment to empirically evaluate this memoisation approach would be needed to determine whether non-constituent-forming derivations would perform well for our one structure per $n$-gram hypothesis.

A description of the implementation used to test the previously mentioned non-constituent-forming cases is provided in Chapter 5. This chapter describes in detail how we constructed a highly tuned system which allowed us to perform our one structure per $n$-gram experiments with minimal overhead. As the C&C parser is already a state-of-the-art parser, it proves to be a tough baseline to beat. All of the tuning described in this chapter was performed so that our changes have a chance of beating the parser in terms of overall parse time.

Chapter 6 describes the set of experiments performed in order to evaluate empirically the idea of $n$-gram-based memoisation for both constituents and non-constituents. The implementation used for these experiments is described in Chapter 5. The results obtained in Chapter 6 indicated clearly that an $n$-gram based approach to our hypothesis cannot work.

An alternative approach is then presented in Chapter 7, where instead of performing memoisation of analyses based on $n$-grams, we investigate performing memoisation based on the CCG categories assigned to each token. As with Chapter 4, we perform a thorough analysis of the idea in this chapter also.
The results of this analysis indicate that this category-based approach to our hypothesis has potential to perform well in practice.

Chapter 8 describes the changes which need to be made to the implementation from Chapter 5 so that the experiments from Chapter 7 can be performed. The initial set of results from these experiments indicated that these category-based spans did seem to work under our one structure per $n$-gram assertion. Final evaluation was performed on the best set of parameters extracted from these experiments, and the final result was that the category-based spans resulted in a 15% improvement in parsing speed with a loss of 0.4% F-score when used under our one structure per $n$-gram hypothesis.

Current parsers are too slow to be able to deal with the vast amount of data which exists on the web. This speed improvement we have achieved here will have substantial benefits for NLP systems such as Question Answering that rely on very large-scale parsed text.


M Kay. 1986. Algorithm schemata and data structures in syntactic processing. pages 35–70.


