Statistical Parsing of Noun Phrase Structure

David Vadas

A dissertation submitted in fulfilment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science)

School of Information Technologies
The University of Sydney

June 2009

Abstract

Noun phrases (NPs) are a crucial part of natural language, exhibiting in many cases an extremely complex structure. However, NP structure is largely ignored by the statistical parsing field, as the most widely-used corpus is not annotated with it. This lack of gold-standard data has restricted all previous efforts to parse NPs, making it impossible to perform the supervised experiments that have achieved high performance in so many Natural Language Processing (NLP) tasks.

We comprehensively solve this problem by manually annotating NP structure for the entire Wall Street Journal section of the Penn Treebank. The inter-annotator agreement scores that we attain refute the belief that the task is too difficult, and demonstrate that consistent NP annotation is possible. Our gold-standard NP data is now available and will be useful for all parsers.

We present three statistical methods for parsing NP structure. Firstly, we apply the Collins (2003) model, and find that its recovery of NP structure is significantly worse than its overall performance. Through much experimentation, we determine that this is not a result of the special base-NP model used by the parser, but primarily caused by a lack of lexical information.

Secondly, we construct a wide-coverage, large-scale NP Bracketing system, applying a supervised model to achieve excellent results. Our Penn Treebank data set, which is orders of magnitude larger than those used previously, makes this possible for the first time. We then implement and experiment with a wide variety of features in order to determine an optimal model. Having achieved this, we use the NP Bracketing system to reanalyse NPs outputted by the Collins (2003) parser. Our post-processor outperforms this state-of-the-art parser.

For our third model, we convert the NP data to CCGbank (Hockenmaier and Steedman, 2007), a corpus that uses the Combinatory Categorial Grammar (CCG) formalism. We experiment with a CCG parser and again, implement features that improve performance. We also evaluate the CCG parser against the Briscoe and Carroll (2006) reannotation of DepBank (King et al., 2003), another corpus that annotates NP structure. This supplies further evidence that parser performance is increased by improving the representation of NP structure.

Finally, the error analysis we carry out on the CCG data shows that again, a lack of lexicalisation causes difficulties for the parser. We find that NPs are particularly reliant on this lexical information, due to their exceptional productivity and the reduced explicitness present in modifier sequences. Our results show that NP parsing is a significantly harder task than parsing in general.

This thesis comprehensively analyses the NP parsing task. Our contributions allow wide-coverage, large-scale NP parsers to be constructed for the first time, and motivate further NP parsing research for the future. The results of our work can provide significant benefits for many NLP tasks, as the crucial information contained in NP structure is now available for all downstream systems.

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

— David Vadas

Acknowledgments

I'll start with James Curran, who has been an amazing supervisor over the past few years. Always putting in more effort than he really should, James spends a ridiculous amount of time on his students. In 2004 I wandered into James' office looking for an honours project (some weeks after students were supposed to have one). I'd been told that he was a good supervisor, which turned out to be very true. The next year, I was doing a PhD. I didn't really know what I was getting into when I started, but James has always guided me in what I was doing, and encouraged me to do better. James taught me how do research, how to be an academic, and what it takes to succeed in this field.

Many members of the Language Technology Research Group at The University of Sydney have been instrumental during my candidature. I was fortunate enough to start my PhD at the same time as a number of other students, and being in a group together with them has made it a fun ride. From many ALTAs, to the joy of group meetings, and especially our trip to ACL in 2007, I have many fond memories to look back upon. So thank you, to Tara McIntosh, for many lunches; to Matthew Honnibal, for being the linguist that I am not; and Toby Hawker, for always having an interesting topic of conversation. Matthew in particular has helped many times with the thesis work itself, working out how to annotate noun phrases and even doing some annotation himself.

Outside of Sydney Uni, I've had helpful feedback and comments from Stephen Clark and Mark Steedman. Mark provided advice on converting my NP data to CCG derivations, while Steve provided feedback on the document that became this thesis and was an excellent "associate supervisor" while I was visiting.

I would also like to thank my markers, Mitch Marcus, Tim Baldwin and the mysterious third marker who may or may not be Dan Bikel. All provided useful comments that have improved this thesis.

Finally, I'd like to thank my parents, who provided me with an education, a house to live in, and even proofread this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parsing is a vitally important Natural Language Processing (NLP) task, where the goal is to identify the syntactic structure of a sentence. Accurate and robust parsing is a major step towards the automatic interpretation of language, as it is informative for many NLP systems such as information retrieval and machine translation.

In recent years, high performance parsing has been achieved for a number of languages, domains and formalisms. The creation of a large corpus annotated with gold-standard parse trees, the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993) has encouraged a great deal of this research in statistical parsing. The parsing field is now well-established, and its results have proven effective in many NLP problems, e.g. Pado and Lapata (2007).

The parsing of noun phrases (NPs) involves the same difficulties as parsing in general. NPs contain structural ambiguities, just as other constituent types do, and resolving these ambiguities is required for their proper interpretation. Despite this, high performance NP parsing has not been achieved until now. The primary reason for this is the absence of a large corpus of NP annotations. The Penn Treebank does not fully annotate the internal structure of NPs, making it impossible to perform the supervised experiments that have performed so well in so many NLP tasks. Parsers trained on this, the de facto standard parsing corpus for English, are unable to recover NP structure, and as a result, the NP parsing problem has largely been ignored.

Most research on parsing NPs has focused on a simplified problem where all NPs are made up of exactly three words, all of which are nouns. In this thesis, we will look at a realistic use of NPs, presenting the first publicly available, large-scale corpus of gold-standard manually annotated NPs. Furthermore, experiments with this data demonstrate that statistical methods can accurately recover NP structure. We identify the difficulties in parsing NPs, compared to parsing in general, and then present a number of techniques for improving performance.

This is the problem solved in this thesis: identifying the syntactic structure of NPs for a wide-coverage, large-scale application.

1.1 Motivation

Noun phrases are a crucial part of natural language. They convey much of the content in a sentence and are therefore vitally important when parsing. Nouns and noun phrases are also particularly productive, and interpreting the new vocabulary that is constantly introduced to the language is a difficult task. NPs are also exceedingly common, and because of this, NP ambiguities occur frequently. In the corpora considered in this thesis, over a quarter of all sentences contain an ambiguous NP. In technical domains, longer, more complex NPs are even more common, and as such, exhibit an even greater level of ambiguity.

In fact, many NLP systems are aimed at identifying the information that is carried in NPs. Question Answering (QA) systems are designed to take a natural language question and find the exact answer in a large quantity of text. These answers are often places, people, or other objects, i.e. noun phrases. Many QA systems employ a parser to generate candidate answers, which can then be ranked to determine the most likely. If the parser cannot recover NP structure then the correct candidate may never be found, even if the correct dominating noun phrase has been found. As an example, consider the following extract:

. . . as crude oil prices rose by 50%, a result of the. . .

and the question:

The price of what commodity rose by 50%?

The answer *crude oil* is internal to the NP *crude oil prices*. Most commonly-used parsers today will not identify this internal NP, and will therefore never be able to get the answer correct.

A similar problem occurs when performing Anaphora Resolution (AR). This task involves identifying an antecedent, for example, the person that a pronoun refers to. Without the ability to recover internal NP structure, it may be impossible to determine the correct antecedent.

For example, the appositional structure in *Margaret Thatcher, the Prime Minister* can cause problems in a number of ways. Firstly, the gender features that are very important in AR can be confused by appositional NPs. Mapping a feminine pronoun onto *the Prime Minister* may be

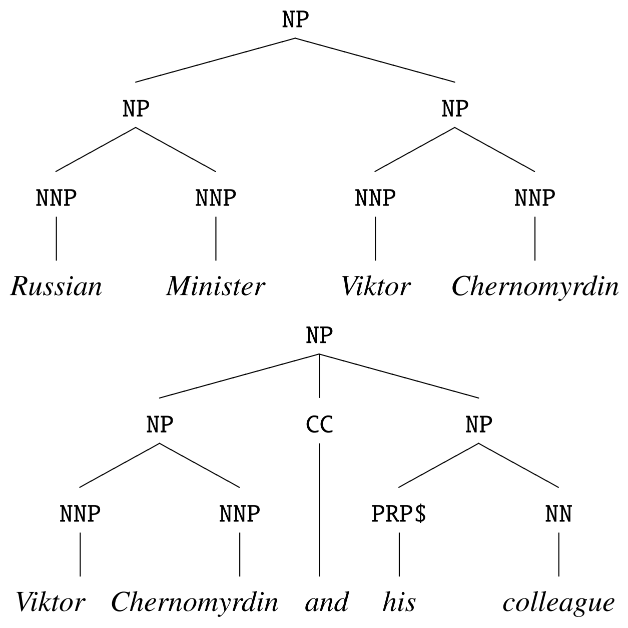


Figure 1.1: Binarisation example from Wang, Knight and Marcu (2007)

statistically unlikely, but knowing that the clearly feminine *Margaret Thatcher* is the same entity makes this decision easier. Secondly, interpreting appositions correctly means that they can be analysed in a consistent way. An AR system could return *Margaret Thatcher*, or *the Prime Minister*, or the entire NP as the antecedent, but only one will be correct according to the evaluation. A parser can provide the information needed to resolve these problems by identifying that only a single entity is being referred to in the NP. This correct appositional structure can then be supplied to the AR system.

NP structure can also be informative for Syntax-Based Statistical Machine Translation (SBSMT) systems, which operate on tree structures rather than a stream of words. Wang, Knight, and Marcu (2007) find that the flat tree structure of the Penn Treebank elongates the tail of rare tree fragments, diluting individual probabilities and reducing performance. This problem can be seen in the two tree fragments in Figure 1.1. Both of these trees are flat in the training data, unlike the structures we have shown, and thus not decomposable by a standard SBSMT system. This means that even though both NPs share a common substructure, *Viktor Chernomyrdin*, this information cannot be generalised in the SBSMT model. The flat NP structure specifically prevents the system from learning the necessary information.

Wang, Knight, and Marcu (2007) attempt to solve this problem by automatically binarising the phrase structure trees. They present three simple methods: left-binarisation, right-binarisation, and head binarisation, and also derive a more complex binarisation method using the Expectation Maximisation (EM) algorithm. All four of these methods improve performance, with the latter approach providing the biggest increase. However, this SBSMT system, as well as others (Melamed, Satta, and Wellington, 2004; Zhang et al., 2006), are relying on a non-gold-standard binarisation of the parse tree. Additional NP annotation, such as explicitly bracketing *Russian Minister* in the example, would provide these SBSMT systems with the more detailed structure that is required, and thus improve performance to an even larger degree. Our experiments in Section 5.4.4 also suggest that using supervised techniques trained on gold-standard NP data would be superior to unsupervised methods such as EM.

The importance of NPs to machine translation is further demonstrated by Koehn (2003), who constructs a dedicated subsystem to translate NPs. This approach is similar to the postprocessor we build in Chapter 5, where we take NPs produced by a parser and reanalyse them with our own NP Bracketing system. In the same way, Koehn uses a parser to identify NPs, which are then translated using an NP-specific system. This system implements additional features, improving accuracy from 53.9% to 67.1%. When this subsystem is embedded in a word-based MT system, its BLEU score (Papineni et al., 2002) increases from 0.172 to 0.198.

Particularly relevant for the NP structure dealt with in this thesis, Koehn (2003, §3.2.2.1) breaks apart NPs in order to achieve a better alignment. This is similar to the binarisation of Wang, Knight, and Marcu (2007) described above, although Koehn only uses the existing structure produced by the parser. This process increases NP alignment from 46.0% to 64.9%, which is by far the largest gain out of all the approaches taken to improve alignment. If more detailed NP structure was available then an even larger increase could have been achieved.

Recovering NP structure can also help in the parsing process itself. Consider two NPs: *lung cancer deaths* and *lung cancer*. Both of these NPs share a similar structure, and so if these NPs are not annotated consistently, then it will cause problems for a machine learner using this training data. The machine learner may have difficulty bracketing the second NP correctly, even though it possesses no internal structure.

For all of the reasons we have given here, NP structure is crucial for NLP applications, and thus its recovery is a vital task for all parsers.

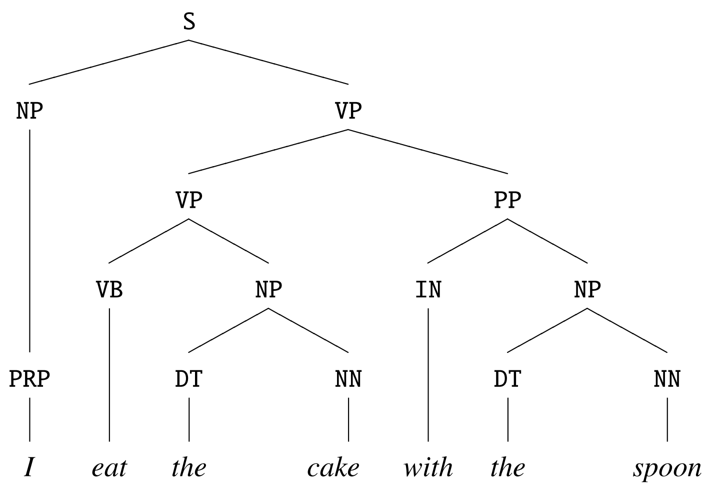


Figure 1.2: An example parse tree

1.2 Parsing

Parsing is the task of finding the syntactic structure for a given sentence. An example sentence and its parse tree is shown in Figure 1.2. This tree describes the constituents that are formed and how they are connected. For example, the NP *the spoon* attaches to the preposition *with*. The tree structure shows that even though *eat* and *spoon* are quite distant in the stream of words, they are actually quite closely connected by the syntactic structure.

Building a parser is a difficult task, for reasons we will describe below. These difficulties also apply to noun phrase parsing, as will be described in Section 1.3. Ambiguity is one of the largest problems for any parsing system, as it creates decision points where errors can be made. Humans are able to disambiguate to the correct meaning using a wide range of semantic and world knowledge. However, these decisions are a tremendously difficult problem for computers, as they do not associate words with any meaning. The programming languages that computers can interpret are specifically designed to be unambiguous to avoid this problem.

There are many aspects to the ambiguity problem, from Part Of Speech (POS) tags to prepositional phrase and other attachment decisions. For example, the word *saw* can be a noun: a tool used to cut wood; or a verb: when something has been observed. Another example is in Figure 1.3, where although the sentence remains the same as in Figure 1.2, this latter tree implies that the cake has a spoon on it, rather than that the spoon is being used to do the eating. It seems more likely that the earlier parse tree is the correct one for this sentence. However, the tree in

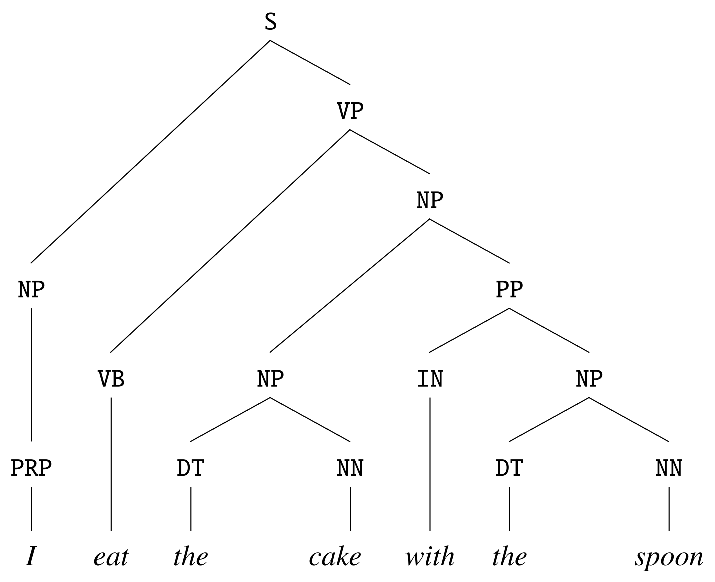


Figure 1.3: Another parse tree with prepositional phrase attachment error

Figure 1.3 will not always be wrong. If the last word of the sentence were *icing* instead of *spoon*, then it would be probable that the latter tree is correct.

It may seem that such problems in ambiguity are simple to resolve, as in the previous example where we have presented only two possibilities. However the combination of multiple ambiguities causes an exponential growth in the number of potential parse trees. Even relatively short sentences can have a huge number of analyses, while a sentence of average length[1] could have over a trillion, depending on the grammar being used. Processing this multitude of readings has a hugely detrimental effect on parser speed, and so analysing these myriad possibilities efficiently is part of the challenge of parsing.

The techniques that have proven most effective at solving these problems involve using statistical machine learning. Statistical models are used by parsers to manage the problem of ambiguity. By applying the probabilities learnt from training data, a parser can choose the most likely derivation. Statistical models are well understood and supply a strong theoretical foundation for parsing, as they can combine competing pieces of evidence in a robust way. Their use has resulted in high accuracy and coverage for a wide variety of languages and domains.

The main requirement for high quality statistical parsing is a consistent, gold-standard an-

[1]Average sentence length in the Penn Treebank is 26 words.

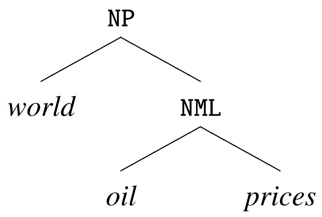


Figure 1.4: Parse tree for a right-branching noun phrase

notated corpus on which to train. This is because a large quantity of text marked up with the correct tree structure is needed to construct an accurate statistical model. Once a parser has constructed such a model, it can calculate the most probable tree for a given sentence.

A gold-standard corpus is also used for parser evaluation, by retaining a held-out test section on which performance can be measured. This is of great importance as it allows comparison between different techniques. It means we can verify improvements in the field through empirical results and significance tests.

The most widely-used corpus in the parsing field is the Wall Street Journal (WSJ) section of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993). It contains over 1.1 million words, and has been used as training and test data for a number of parsers. It has become the de facto standard in the parsing community, with specific splits for training, testing and development well-known and used by all researchers. We will describe this corpus in further detail in Section 2.2.

1.3 Noun Phrases

An example noun phrase with its correct parse tree is shown in Figure 1.4. This is a *right-branching* NP, where the right-most pair of words form a constituent. This structure describes a link or *dependency* between *oil* and *prices*, and also between *world* and *prices*. The result is an interpretation that the NP is talking about the price of oil, and those prices are from around the world.

Conversely, the NP in Figure 1.5 is a *left-branching* NP, as the left-most pair of words form a constituent. Although the two NPs in these examples are very similar, their parse trees are different. If *crude oil prices* were a right-branching NP, then it would result in the interpretation that the prices are crude, which is a clear mistake.

Most of the NP ambiguities considered in this thesis will feature premodificational structure, as in these examples. The ambiguity arises because modifiers can be modified themselves.

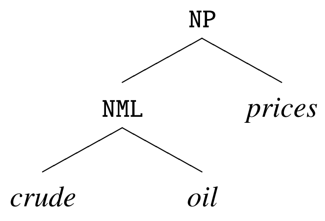


Figure 1.5: Parse tree for a left-branching noun phrase

This occurs in *crude oil prices*, whereas in *world oil prices*, each modifier attaches to the head independently.

Humans are quite good at determining which attachment is correct. In the NP *cardboard detergent box*, we understand that *cardboard detergent* is a nonsensical item, and so the NP must be right-branching. This semantic information in intrinsic to humans, but very difficult to encode for a computer. The same problem applies to humans shown an NP from an unfamiliar domain (Quirk et al., 1985, page 1343), e.g. *senior subordinated reset discount debentures*.

Parsing NPs may seem like a simpler task than parsing whole sentences, however, the same difficulties still apply. NPs can be infinitely recursive, as modifiers can be added indefinitely. For example, *French onion soup* can function as an NP, however we can append to it to make *French onion soup bowl* and continue even further to *wooden French onion soup bowl handle* and beyond. A definition of NPs and their modifiers, such as those in this example, will be presented in Section 2.1.

Another difficulty in parsing NPs is the fact that nouns are the most productive class of words. This means that there are constantly new nouns and thus NPs, to consider. As a consequence, most unknown words are nouns, making it more likely that there will be no lexical information on which to base a decision.

Ambiguity can also be present in NPs, in the same manner as in general parsing. The number of bracketing possibilities for an NP is equal to the Catalan number[2] of its length. This figure grows exponentially. The prepositional phrase attachment task we saw earlier in Figures 1.2 and 1.3 has the same problem. This problem is made more difficult by the lack of a good baseline decision. In our data, we find that there is an almost perfectly even split between left and right-branching NPs, while most other data sets, e.g. Lauer (1995b), do not have a large majority either way. As a result, a simple guessing strategy performs poorly.

[2]The nth Catalan number, C_n, is $\frac{1}{n+1}\binom{2n}{n}$

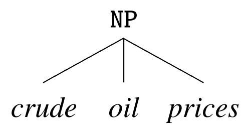


Figure 1.6: The flat Penn Treebank representation for noun phrases

Having seen that NP parsing suffers from the same difficulties that we observed in Section 1.2, it may be expected that the statistical modelling techniques that have been successful there may be applied here. Unfortunately, there is no sufficiently large corpus of NP structure for accurate supervised machine learning. The Penn Treebank commits only to flat structures which lack the required information. For example, the NP we saw earlier in Figure 1.5 is annotated with the parse tree shown in Figure 1.6. This tree is flat — it has no internal bracketing whatsoever and so provides no information on what structure should apply. This lack of annotated structure means that NP parsing has typically been performed with unsupervised techniques. This unsupervised approach has been attempted on a number of NLP tasks, however performance is almost always significantly lower than with supervised learning, e.g. in POS tagging (Brill, 1995).

The most widely-used NP data set contains only 244 NPs (Lauer, 1995b), all of which are only three words long. The set also includes no non-noun parts of speech at all, and is restricted to only include words that appear in Roget's thesaurus. This lack of data prevents NPs from being included in many full parsing models. This is the major impediment to high performance NP parsing, and one of the problems we solve in this thesis.

1.3.1 Syntax vs Semantics

This thesis is primarily interested in syntactic, rather than semantic, NP relationships. This latter distinction can be seen in the NP *chocolate money*, which may be money made out of chocolate; or money set aside for the purchasing of chocolate. This task has been has undertaken by many researchers, such as Lauer (1995b) and Girju (2007). Determining the set of relationships itself is a complex problem, for which a number of solutions have been proposed, e.g. in Levi (1978) and Barker and Szpakowicz (1998).

NP syntax is of course affected by semantic information, and vice versa. For example, we use semantic information to inform our annotations in Chapter 3, and implement features based on semantic relationships in the NP Bracketing experiments in Chapter 5. Conversely, syntactic analysis determines which noun pairs form a dependency, and thus, which noun pairs have semantic relationships to analyse. For this reason, we would want to parse *cardboard detergent box* first and find the correct dependencies, so that we do not attempt to characterise the semantic relationship between *cardboard* and *detergent*.

In this thesis, we will not perform semantic relationship experiments, instead wholly focusing on syntactic parsing.

1.4 Contributions

In this thesis, we present a comprehensive approach to performing statistical parsing of noun phrases. We identify two key problems for this task: the lack of gold-standard data; and the inability of existing parsing models to recover NP structure accurately. We address the entire breadth and depth of these problems, by creating new data, validating it for correctness, utilising it in three separate parsing architectures, and implementing new features that achieve an excellent level of performance in each case.

Chapter 2 reviews the background of statistical parsing. We cover the two parsing models used in this thesis, amongst others, and how they recover syntactic structure. We also detail why NP structure has *not* been recovered in the past.

Chapter 3 describes our augmentation of the Penn Treebank with gold-standard NP structure. Our corpus is the first publicly available, large-scale corpus of manually annotated NP structure. We also describe the multiple approaches we undertook to ensure an accurate and consistent corpus. This chapter is based on work we presented in Vadas and Curran (2007a).

Chapter 4 describes the experiments we performed with our extended Penn Treebank, using the Bikel (2004) implementation of the Collins (2003) model.[3] We analyse the effect of various aspects of Collins' model, as well as how the representation we use for NP structure affects performance. Finally, we perform a comprehensive error analysis which identifies why NPs are difficult to parse. These are the first large-scale experiments on NP parsing. The work in this chapter is based on the initial experiments in Vadas and Curran (2007a) and the analysis in Vadas and Curran (2007c).

The task of NP Bracketing is introduced in Chapter 5, as we work towards a post-processor for NPs produced by Collins' parser. We begin with the simplified problem undertaken in most of the NP Bracketing literature, before building supervised models on realistic NPs. These are the

[3]In all of our experiments we use Bikel's and not Collins' parser. We will still refer to Collins' model in the thesis.

first NP Bracketing models built for large-scale data and the first supervised models that achieve high performance. The final result of this chapter is a post-processor that significantly outperforms Collins' parser. The work presented in this chapter is based on that presented in Vadas and Curran (2007b).

Chapter 6 covers our experiments with the Combinatory Categorial Grammar (CCG) formalism (Steedman, 1996; Steedman, 2000). We demonstrate the utility of the NP data for multiple formalisms, and in particular, solve a problem with the representation of NPs in the CCG corpus. The C&C parser (Clark and Curran, 2007b) is used to recover NP structure, while its architecture allows us to relatively easily add features that were effective in NP Bracketing. This allows the CCG parser to recover NP structure for the first time. This chapter is based on and extends work published in Vadas and Curran (2008).

This thesis contributes not only a new data set and results from a number of experiments, but also makes large-scale wide-coverage NP parsing a possibility for the first time. Whereas before it was difficult to even evaluate what NP information was being recovered, this thesis sets a high benchmark for NP structure accuracy, and opens the field for even greater improvement in the future. As a result, downstream applications can now take advantage of the crucial information present in NPs.

Chapter 2

Background

This chapter will present a summary of statistical parsing models. Our focus is specifically on the models of Collins (2003) and Clark and Curran (2007b), as they will be used in Chapters 4 and 6 respectively. We will also examine the annotation scheme of the Penn Treebank, and how it has affected the parsing field.

In particular, the Penn Treebank's treatment of noun phrases will be explored, explaining why no internal NP structure is annotated, and the problems that arise as a result. Parsers trained and evaluated on this corpus have inherited this underspecified NP representation, and so are unable to recover this vitally important structure.

2.1 Noun Phrases

In this thesis, we will analyse and annotate the internal structure of NPs. This structure can be interpreted in a few different ways, e.g. the DP (determiner phrase) analysis argued by Abney (1987) (and argued against by van Eynde (2006)) treats the determiner as the head, rather than the noun. We will use a definition that is more informative for statistical modelling, where the noun — which is much more semantically indicative — acts as the head of the NP structure.

A noun phrase is a constituent that has a noun as its head[1], and can also contain determiners, premodifiers and postmodifiers. The head by itself is then an unsaturated NP, to which we can add modifiers and determiners to form a saturated NP. Or, in terms of X-bar theory, the head is an N-bar, as opposed to the fully formed NP. Modifiers do not raise the level of the N-bar, allowing them to be added indefinitely, while determiners do, making NPs such as **the the dog* ungrammatical.

[1]The Penn Treebank also labels substantive adjectives such as *the rich* as `NP`, see Bies et al. (1995, §11.1.5)

The Penn Treebank annotates at the NP level, but leaves much of the N-bar level structure unspecified. This will be described in Section 2.2.1. As a result, most of the structure we annotate will be on unsaturated NPs. There are some exceptions to this, such as appositional structure, where we bracket the saturated NPs being apposed.

Quirk et al. (1985, §17.2) describe the various components of a noun phrase as follows:

- The head is the central part of the NP, around which the other constituent parts cluster. It also determines how the NP combines with and affects the rest of the sentence. Some heads form NPs all by themselves, such as pronouns and proper nouns.

- The determinative, which includes predeterminers such as *all* and *both*; central determiners such as *the*, *a* and *some*; and postdeterminers such as *many* and *few*. For example *all the furniture* contains a predeterminer and a central determiner, while *the few survivors* has a central determiner and a postdeterminer.

- Premodifiers, which come between the determiners and the head. These are principally adjectives (or adjectival phrases) and nouns, as in *red car* and *interest rate* respectively. Each of these modifiers can attach to the head, or to any of the other modifiers. Premodifiers can also be built up recursively into long sequences, as in our example from Chapter 1: *wooden French onion soup bowl handle*.

- Postmodifiers are those items after the head, such as prepositional phrases, as well as nonfinite and relative clauses.

As we described in Section 1.3 and will again discuss in Section 2.2.1, most of the ambiguity that we deal with in this thesis arises from premodifiers. Quirk et al. (1985, page 1243) specifically note that *"premodification is to be interpreted … in terms of postmodification and its greater explicitness"*. Comparing *an oil man* to *a man who sells oil* demonstrates how a postmodifying clause and even the verb contained therein can be reduced to a much less explicit premodificational structure. Understanding the NP is much more difficult because of this reduction in specificity, however the NP can still be interpreted with the appropriate context.

2.2 The Penn Treebank

Statistical parsing for English has, in many ways, been dominated by the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993). This was the first large-scale corpus manually anno-

```
( (S
    (SINV-TPC-2
      (VP-TPC-1 (VBG Offsetting)
        (NP
          (NP (DT the) (JJR lower) (NN stake) )
          (PP (IN in)
            (NP (NNP Lyondell) ))))
      (VP (VBD were)
        (VP (-NONE- *T*-1) ))
      (NP-SBJ
        (NP (JJR higher) (JJ crude) (NN oil) (NNS prices) )
        (, ,)
        (NP (JJ increased) (JJ natural) (NN gas) (NNS volumes) )
        (CC and)
        (NP (JJR higher) (NN coke) (NNS prices) )))
    (, ,)
    (NP-SBJ (DT the) (NN company) )
    (VP (VBD said)
      (SBAR (-NONE- 0)
        (SINV (-NONE- *T*-2) )))
    (. .) ))
```

Figure 2.1: An example sentence from the Penn Treebank

tated with gold-standard parse trees. Its creation has allowed supervised experimentation that was simply not possible before. An example sentence from the Penn Treebank is shown in Figure 2.1. The labelled brackets describe the syntactic structure and POS tags of individual words.

Here is a brief description of the POS tags that frequently occur in NPs. There are a number of noun POS tags, the basic tag being `NN`. Proper nouns are `NNP` and plural nouns are `NNS`, while proper plural nouns are `NNPS`. The determiner tag is `DT`, while predeterminers such as *all* have their own `PDT` tag. The basic adjective tag is `JJ`, and there are also relative and superlative forms, `JJR` and `JJS` respectively. Finally, possessive pronouns such as *his* and *her* have the tag `PRP$`. The basic pronoun tag itself, `PRP`, is very infrequent in the NPs we analyse, as a pronoun typically forms an NP all by itself with no additional structure to annotate.

The second version of the Treebank introduced two new levels of annotation, which are noticeable in this extract. Firstly, the traces denoted by `*T*` indicate null elements which are co-indexed with the relevant constituent elsewhere in the tree. For example, the verb phrase beginning with *Offsetting* has been extracted from its canonical position following *were*. To mark this, a trace is inserted at the extraction point, and indexed `1`. This can then be mapped to the `1` on the `VP`.

Treebank II also introduced functional markers that indicate some limited semantic structure. In the above sentence, `TPC` marks the topicalised `VP`, while `SBJ` denotes the `NP` as the subject of the sentence. Marcus et al. (1994) describe encountering difficulties while annotating semantic structure, and as such, these markers are relatively limited and not always consistently used.

The Penn Treebank treatment of NPs is also visible in the sentence above. Although both *higher crude oil prices* and *increased natural gas volumes* feature clear internal structure (*crude oil* and *natural gas* respectively), this is not annotated in the corpus. We described this deficiency in Section 1.3 and will solve the problem in Chapter 3.

Evaluating on the Penn Treebank is performed by comparing the brackets produced by a parser to those in the gold-standard data. Each bracket must begin and end at the same token as in the gold standard, and have the correct label. This matched bracket evaluation can then be applied to determine precision, recall, and thus F-score figures (Black et al., 1991). Most parsers do not recover, and thus are not evaluated on, the functional markers or traces described above. There are some exceptions to this, such as Collins' Model 3, Blaheta and Charniak (2000) and Gabbard, Marcus, and Kulick (2006), however these are in the minority.

2.2.1 Noun Phrases in the Penn Treebank

As we saw in the previous section, NPs do not receive the same treatment as other constituents in the Penn Treebank. The annotation guidelines (Bies et al., 1995, page 120) sum this up by saying:

As usual, NP structure is different from the structure of other categories.

This gives an indication of the approach that has been taken for NPs in many cases, where special treatment is required or assumed.

The Penn Treebank does not annotate the internal structure of noun phrases, instead leaving them flat. Here is the Penn Treebank representation of the NPs we saw earlier in Section 1.3:

```
(NP (NN world) (NN oil) (NNS prices))
(NP (NN crude) (NN oil) (NNS prices))
```

Despite the first NP being right-branching and the second being left-branching, they are both annotated in exactly the same way. The difference in their structures is not reflected in the underspecified Penn Treebank representation. This absence of annotated NP data means that any parser trained on the Penn Treebank (or another corpus derived from it) is unable to recover NP structure.

The Penn Treebank literature provides some explanation for the absence of NP structure. Marcus, Santorini, and Marcinkiewicz (1993) describe how a preliminary experiment was performed to determine what level of structure could be annotated at a satisfactory speed. This chosen scheme was based on the Lancaster UCREL project (Garside, Leech, and Sampson, 1987). This was a fairly skeletal representation that could be annotated 100–200 words an hour faster than when applying a more detailed scheme. However, it did not include the annotation of NP structure.

Another potential explanation is that Fidditch (Hindle 1983; 1989) — the partial parser used to generate a candidate structure, which the annotators then corrected — did not generate NP structure. Marcus, Santorini, and Marcinkiewicz (1993, page 326) note that annotators were much faster at deleting structure than inserting it, and so if Fidditch did not generate NP structure, then the annotators were unlikely to add it.

The bracketing guidelines (Bies et al., 1995, §11.1.2) suggest a further reason why NP structure was not annotated, saying *"it is often impossible to determine the scope of nominal modifiers"*. That is, Bies et al. (1995) claim that deciding whether an NP is left or right-branching is difficult in many cases. Bies et al. (1995) give a handful of example NPs, a subset of which we have reproduced below:

```
(NP fake sales license)
(NP fake fur sale)
(NP white-water rafting license)
(NP State Secretary inauguration)
```

The scope of these modifiers is quite apparent. The reader can confirm this by making his or her own decisions about whether the NPs are left or right-branching. Once this is done, compare the bracketing decisions to those made by our annotators, shown in this footnote.[2] We expect that there is perfect agreement.

There are some examples from the guidelines that were more difficult for our annotators:

```
(NP week-end sales license)
(NP furniture sales license)
```

[2]Right, left, left, left

However this difficulty in large part comes from the lack of context that we are given. If the surrounding sentences were available, we expect that the correct bracketing would become more obvious. Unfortunately, this is hard to confirm, as we searched the corpus for these NPs, but it appears that they do not come from Penn Treebank text, and therefore the context is not available. And if the reader wishes to compare again, here are the decisions made by our annotators for these two NPs.[3]

Our position then, is that consistent annotation of NP structure is entirely feasible. This seems apparent, as despite the reservations expressed in the guidelines, even the examples therein support our argument. Furthermore, Quirk et al. (1985, page 1343) have this to say:

> *Indeed, it is generally the case that obscurity in premodification exists only for the hearer or reader who is unfamiliar with the subject concerned and who is not therefore equipped to tolerate the radical reduction in explicitness that premodification entails.*

Accordingly, an annotator with sufficient expertise at bracketing NPs should be capable of identifying the correct premodificational structure, except in domains they are unfamiliar with. This hypothesis will be tested in Chapter 3.

2.3 Penn Treebank Parsing

With the advent of the Penn Treebank, statistical parsing without extensive linguistic knowledge engineering became possible. The first model to exploit this large corpus of gold-standard parsed sentences was described in Magerman (1994; 1995). This model immediately demonstrated the effectiveness of training on a large-scale corpus by outperforming a hand-written grammar with 10 years of development. The hand-written grammar achieved a 0-crossing-brackets score of 69%, while Magerman (1995) attains 76%. Magerman thus showed that a machine learner assigning probabilities to parse structures was far superior to a hand-built grammar.

Magerman's model first builds a candidate parse tree with probability above a certain threshold. Magerman chose a value of 10^{-5} for this parameter. A breadth-first search is then used, where partial parses can be pruned if their probability is below the best complete candidate found so far. Leaf probabilities from a decision tree are used to calculate the probabilities of potential parsing actions, and this information is in turn used to build the parse tree bottom-up. The probability of the entire tree is then the product of the probabilities assigned by the decision tree to individual parsing actions. The model achieves 86.3% precision and 85.8% recall on matched brackets for sentences

[3]Right, left

with less than 40 words on Section 23 of the Penn Treebank. This is the standard test set which has been used by all researchers since.

One of Magerman's important innovations was the use of deterministic head-finding rules to identify the head of each constituent. The head word was then used to represent the constituent in the features higher in the tree. This original table of head-finding rules has since been adapted and used in Collins' and Charniak's parsers, as well as in the creation of CCGbank (Hockenmaier, 2003a).

Collins (1996) followed up on Magerman's work by implementing a statistical model that calculates probabilities from relative frequency counts in the Penn Treebank. The conditional probability of the tree is split into two parts: the probability of individual base-NPs; and the probability of dependencies between constituents. This separation allows NPs to be represented by a single head token when calculating probabilities external to the NP. The other tokens that are internal to an NP will not affect these outside dependencies, but would increase the sparsity of the data, and thus have an adverse effect on the model.

Collins uses the CKY chart parsing algorithm (Kasami, 1965; Younger, 1967; Cocke and Schwartz, 1970), a dynamic programming approach that builds parse trees bottom-up, (see Jurafsky and Martin (2000, page 453) for a full description). This is an improvement over Magerman's rather convoluted method for building trees, and has been used in a number of parsers since. Collins also adapts Magerman's (1995) head-finding rules. The Collins (1996) model outperforms Magerman's, achieving 86.3% precision and 85.8% recall for sentences with less than 40 words, while also being simpler and much faster.

Collins (1997) describes a cleaner, generative model. For a tree T and a sentence S, this model calculates the joint probability, $P(T, S)$, rather than the conditional, $P(T|S)$. This second of Collins' models uses a lexicalised Probabilistic Context Free Grammar (PCFG), and solves the data sparsity issues by making independence assumptions. The head is generated first, followed incrementally by the modifiers, each of which is conditioned only on the head and not on any of the other modifiers. Collins (1997) then goes on to describe extensions to the model, first including subcategorisation frames, and then incorporating the traces that result from *Wh*-movement. We will describe Collins' parsing models in more detail in Section 4.1. The best performing model, including all of these extensions, achieves 88.6% precision and 88.1% recall on sentences with less than 40 words.

Charniak (1997) presents another probabilistic model that builds candidate trees using a chart, and then calculates the probability of chart items based on two values: the probability of the head; and that of the grammar rule being applied. Both of these are conditioned on the node's category, its parent category, and the parent category's head. This model achieves 87.4% precision and 87.5% recall on sentences with less than 40 words, a better result than Collins (1996), but inferior to Collins (1997).

Charniak (2000) improves on his initial result, with the greatest performance increase coming from generating the lexical head's pre-terminal node before the head itself, as in Collins (1997). Collins (2000) also improves on his previous figure by reranking parser output with a second model that includes additional features. With these additions, the models of Collins and Charniak were both achieving slightly over 90% precision and recall.

However, by this stage both models had become exceedingly complex, and the analysis of such models was in itself a significant task. Bikel (2004) performs a detailed study of the Collins (2003) parsing model, finding that lexical information is not the greatest source of discriminative power, as was previously thought, and that 14.7% of the model's parameters could be removed without decreasing accuracy.

Further performance increases proved difficult to obtain, and so researchers moved onto reformulations of the parsing problem, such as dependency parsing (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; McDonald et al., 2005), or onto particular issues, such as domain adaptation (Roark and Bacchiani, 2003; McClosky, Charniak, and Johnson, 2006). The parsing of NPs dealt with in this thesis is one issue still left unresolved.

2.4 Alternate Approaches to Parsing

As we have seen, the dominant statistical parsing paradigm has been to use generative models with an underlying PCFG that incorporates lexical information. However, this is not the only way to build an accurate parsing model, and many alternative models have been implemented to good effect.

Klein and Manning (2003) implement an unlexicalised PCFG that performs better than the Magerman (1995) and Collins (1996) models. This is achieved by splitting constituent labels into multiple categories, depending on how they are being used. For example, subject NPs are distinguished from object NPs, as the former will be dominated by an S and the latter by a VP. Different probabilities can then be assigned to the more fine-grained categories, allowing the model to distinguish between them. Although the results are not state-of-the-art, they demonstrate that lexicalisation may not be as important as had been thought in the past. An unlexicalised model does have the advantage that it is more easily applied to different domains. This is because the lexical differences between the two domains will not affect the parsing model.

Following on from this work, Matsuzaki, Miyao, and Tsujii (2005) and Petrov et al. (2006) use Expectation Maximisation (EM) to automatically identify optimal splits for the constituent labels. This improves on Klein and Manning's (2003) approach, where splits were identified manually. Petrov et al. (2006) achieve 90.2% F-score, a similar result to the Collins (2000) and Charniak (2000) models, even though the grammar is unlexicalised.

Another area that has seen recent interest is reranking parser output, as initially suggested by Collins (2000). This involves taking the best *n* outputs from a parser, and then applying a second, discriminative model. The advantage of using this second model is that many additional features can be added, while the complexities involved in building the parse trees can be ignored. This is possible because only a small number of already generated parses need to be considered. This is a far smaller search space than what the initial model must cover.

Charniak and Johnson (2005) use a Maximum Entropy model to perform discriminative reranking on the output of the Charniak (2000) parser. As a result, the matched bracket F-score increases from 89.7% to 91.0%. McClosky, Charniak, and Johnson (2006) further demonstrate that reranking is effective across domains and when combined with self-training.

Bod (2001) achieves 90.8% precision and 90.6% recall with his Data Oriented Parsing (DOP) model, which was first introduced in Bod (1992). This model does not attempt to capture the notion of constituency as a PCFG does, instead it draws probabilities from every possible tree fragment in the training corpus. Because of this approach, DOP is able to parse idioms well, since it can simply copy the idiom tree fragment from the training data. Bod (1993) describes the DOP process of building a derivation forest using a CKY chart. However the vast number of possible fragments means that finding the most probable tree in the forest is NP-complete. Monte Carlo sampling sidesteps this problem, however this is only feasible on the small sentences of the Penn Treebank ATIS corpus, and not on the Wall Street Journal section. Bod (2001) presents a better solution that estimates the most probable parse by looking at the top 1000 derivations and pruning unlikely chart items. This results in performance in the same range as Collins (2000) and Charniak (2000).

Another recently available parser is RASP (Briscoe and Carroll, 2006), which is designed to be effective on a wide range of domains. To achieve this, the parser uses an unlexicalised model and a grammar developed on the Susanne corpus, a subset of the balanced Brown corpus (Francis and Kučera, 1982), which contains text from newspaper press, biographies, technical writing and fiction. Briscoe and Carroll (2006) describe how the parser was tuned for Wall Street Journal text, by adding and removing specific grammar rules to better fit this corpus. This demonstrates how RASP aims to be a general purpose parser, with a broadly applicable backbone grammar that can quickly be modified to suit specific domains.

There are a number of recently developed parsers that apply a deep grammar, i.e. a grammar that implements a theoretical syntax or possesses a rich linguistic representation. Head Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994), Tree Adjoining Grammar (TAG) (Joshi and Schabes, 1992), Lexical-Functional Grammar (LFG) (Kaplan and Bresnan, 1982) and Combinatory Categorial Grammar (CCG) (Steedman, 1996; Steedman, 2000) all allow the direct recovery of predicate-argument structure, which is advantageous compared to the simple phrase structure of the Penn Treebank. These deep grammars intrinsically capture the idea of syntactic or semantic role within a sentence, in a more comprehensive manner than the function tags and traces in the Penn Treebank.

Of course, these grammars require a large annotated corpus in order to build and evaluate a statistical parsing model. Accordingly, there have been projects to convert the formalism-neutral Penn Treebank to HPSG (Miyao, Ninomiya, and Tsujii, 2004), TAG (Xia, 1999; Chen, Bangalore, and Vijay-Shanker, 2006), LFG (Cahill et al., 2002) and CCG (Hockenmaier, 2003a; Hockenmaier and Steedman, 2007). And using the resulting corpora, parsers have been developed for each of these grammars: the Miyao and Tsujii (2005) parser for HPSG; Chiang's (2000) TAG parser; and the Cahill et al. (2004) system which converts a Penn Treebank parser's output to LFG.

All of these deep grammars are similar in many respects. They are all weakly equivalent in expressive power, as they are all capable of describing mildly context-sensitive languages. In this thesis, we will focus on CCG, as its corpus has specific problems with NPs (which will be discussed in Section 2.5.1). However our analysis of NP structure could be applied to any and all of these formalisms.

2.5 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) (Steedman, 1996; Steedman, 2000) is a linguistically expressive, lexicalised theory of grammar based on Categorial Grammar (Ajdukiewicz, 1935; Bar-Hillel, 1953; Wood, 1993). By *lexicalised*, we mean that the grammar is not recorded in rules, as in a CFG. Instead, this information is stored on individual tokens which are assigned lexical categories. For example, a transitive verb (such as *bought* in *IBM bought the company*) has the category

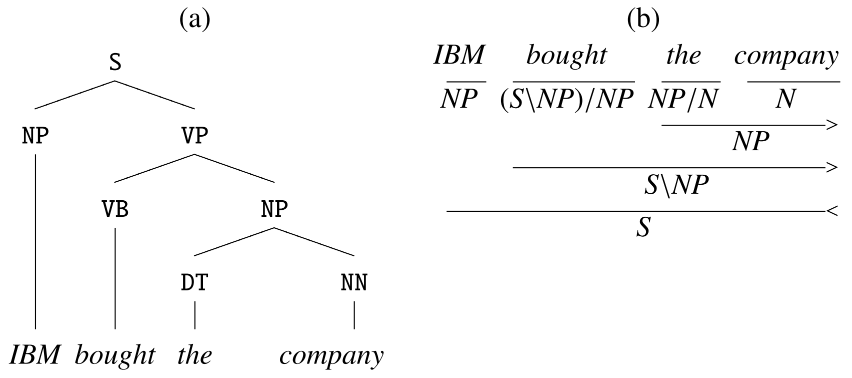


Figure 2.2: Example derivations in (a) Penn Treebank Phrase Structure and (b) Combinatory Categorial Grammar

$(S\backslash NP)/NP$. The slashes here indicate the direction of the arguments. One NP, the subject, is expected on the left and a second NP, the object, is expected on the right. Once these arguments are filled, a sentence is produced.

The CCG lexicalisation results in the same derivation as using a pair of grammar rules: VP → VB NP and S → NP VP. Figure 2.2 shows such a comparison between Penn Treebank-style Phrase Structure and CCG.

All lexical categories in CCG are built using a small number of atomic types, such as Sentence *S*, Noun Phrase *NP* and Prepositional Phrase *PP*, which can be combined recursively using the slash notation.

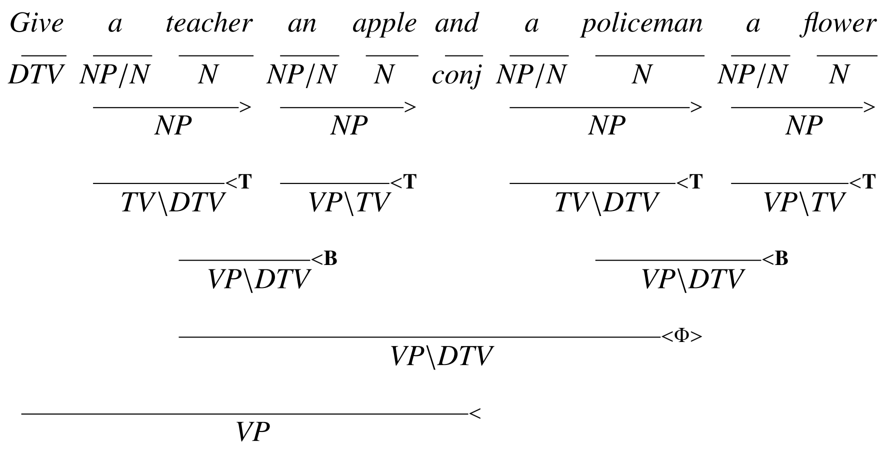
CCG uses a small number of combinatory rules, which govern the interaction between lexical categories. These rules are largely language neutral, allowing the language-specific part of the grammar to remain in the lexicon. We have already seen the two simplest combinatory rules, forward and backward application, in Figure 2.2(b). These rules were present in the original Categorial Grammar (Bar-Hillel, 1953) and are all that are required for a context-free grammar. They are described formally below:

$$X/Y \quad Y \quad \Rightarrow \quad X \quad (>) \tag{2.1}$$

$$Y \quad X\backslash Y \quad \Rightarrow \quad X \quad (<) \tag{2.2}$$

These rules can be interpreted in a rather straight-forward manner. A function category X/Y finds the argument Y it is looking for, producing the function's result X.

CCG is type-driven, meaning that each of the grammar rules is associated with a combinatory logic operator. This is the underlying mechanism that allows CCG to manage long-range



$$VP = S\backslash NP \quad TV = (S\backslash NP)/NP \quad DTV = ((S\backslash NP)/NP)/NP$$

Figure 2.3: Argument cluster coordination in CCG

dependencies as well as giving it a transparent semantic interface, as exploited in Bos et al. (2004). This is a major advantage of CCG: its ability to seamlessly incorporate long-range dependencies into its parsing model, as we will describe in Section 2.5.2.

The power of CCG becomes clear when looking at linguistic constructions such as gapping and movement. These constructions are extremely common in everyday language, and even more so in newspaper text like that of the Wall Street Journal. 70% of sentences in The Penn Treebank contain the traces that annotate these structures, and most parsers simply ignore them. CCG allows the dependencies created by these structures to be recovered intrinsically. Further, the lexical categories for tokens undergoing these processes remain in their canonical form. This is an attribute shared by LFG, but not by TAG and HPSG, wherein relativisation and coordination require special treatment.

These advantages can be seen in the derivation in Figure 2.3. This example of *argument cluster coordination* from Steedman (2000, page 46), shows two arguments of a ditransitive verb being conjoined with another pair of arguments before attaching to the verb itself. As can be seen, all of the tokens have their standard categories: *Give* is a ditransitive verb and each of the arguments forms an *NP*. The underlying combinatory head-passing rules also mean that the correct dependencies are created, between *Give* and each of the four arguments. In order to make this possible, three additional rules have been introduced: type-raising, composition and coordination. The first two of these are what extend CCG to be mildly context sensitive.

Type-raising, denoted with **T** in Figure 2.3, turns an argument category into a function category over functions expecting the original argument. That is, rather than the function taking the argument, the argument takes the function. There are two rules, forward and backward type-raising, both shown below:

$$X \quad \Rightarrow_{\mathbf{T}} \quad T/(T\backslash X) \quad (< \mathbf{T}) \tag{2.3}$$

$$X \quad \Rightarrow_{\mathbf{T}} \quad T\backslash(T/X) \quad (> \mathbf{T}) \tag{2.4}$$

Composition, denoted with **B**, allows function categories to combine before their arguments have been filled. In the backward composition in Figure 2.3, *an apple* has been type-raised to act as a verb phrase looking for a transitive verb. To the left, *a teacher* has been type-raised to a transitive verb category looking to fill a ditransitive verb argument. The two constituents can be composed, with *a teacher* filling the transitive verb arguments slot, while the ditransitive verb argument is appended to the resulting category: $VP\backslash DTV$. The rule schemas of composition: forward, backward and backward crossed, are shown below:

$$X/Y \;\; Y/Z \quad \Rightarrow_{\mathbf{B}} \quad X/Z \quad (> \mathbf{B}) \tag{2.5}$$

$$Y\backslash Z \;\; X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X\backslash Z \quad (< \mathbf{B}) \tag{2.6}$$

$$Y/Z \;\; X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X/Z \quad (< \mathbf{B}_{\mathrm{X}}) \tag{2.7}$$

Finally, the ternary CCG coordination rule, denoted with Φ, has the interpretation of conjoining like categories:

$$X \;\; conj \;\; X \quad \Rightarrow_{\Phi} \quad X \quad (<\Phi>) \tag{2.8}$$

The composition and type-raising combinatory rules supply the expressive power of CCG. However, they also introduce the problem of *spurious ambiguity*. This is where a single sentence has multiple valid derivations, all of which result in the same logical representation. For example, the sentence which we earlier analysed in Figure 2.2 can also be derived as shown in Figure 2.4, which results in the same dependencies. Additional derivations, like this one, substantially increase the level of ambiguity in the grammar. As a result, a statistical parser must perform more calculations over the much larger number of derivations.

One way of dealing with the spurious ambiguity problem is to impose *normal-form* constraints on the grammar (Eisner, 1996). These constraints mean that type-raising and composition are only used when the derivation requires them. This is achieved by not allowing categories that

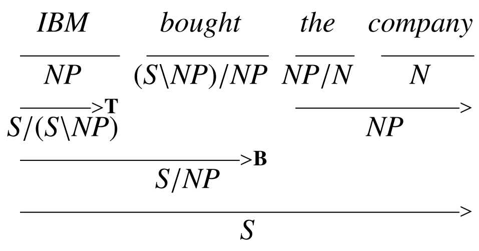


Figure 2.4: Spurious ambiguity in a CCG derivation

have resulted from composition to serve as functions themselves. In CCGbank, the primary CCG corpus, all derivations are in normal form.

The spurious ambiguity problem was long thought to make CCG parsing infeasible, however numerous CCG parsers have now been developed, which manage the problem in various ways. These will be discussed in Section 2.5.2.

2.5.1 Noun Phrases in CCGbank

CCGbank is the primary English corpus for the Combinatory Categorial Grammar formalism, and has been used to train a number of parsers. CCGbank was developed by a semi-automatic conversion process from the Penn Treebank, which as we mentioned earlier, has been done for a number of deep grammars.

Each of these formalisms has its own variations on what syntactic information is required. This information may be recoverable from the Penn Treebank, but if not, it can be problematic. In some cases, as for head-finding, heuristics can be applied to ascertain the missing information with satisfactory accuracy. Hockenmaier (2003a) does so by adapting the head-finding heuristics of Collins (2003). But unfortunately, for some information there is no sufficient heuristic. For example, complements and adjuncts are not annotated in the Penn Treebank, as functional markers such as `SBJ` and `CLR` are not always consistent or even present. The technique chosen to reproduce this information can have a large effect on the size and coverage of the resulting grammar (Chen, Bangalore, and Vijay-Shanker, 2006).

In CCGbank, the unavailability of gold-standard noun phrase structure is particularly problematic. Because CCGbank is binary branching, it cannot replicate the flat structure of the Penn Treebank. Instead, *all* NPs were made right-branching, as shown below:

```
(N (N/N lung)
  (N (N/N cancer) (N deaths) ) )
```

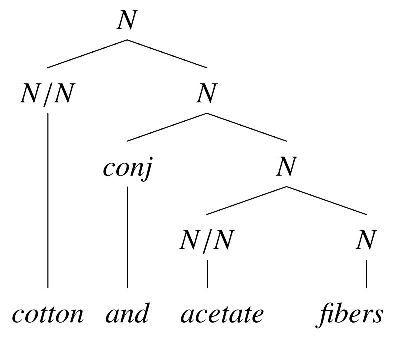


Figure 2.5: Incorrect CCG derivation from Hockenmaier (2007)

This structure is correct for a number of English NPs and is the best solution that does not require manual reannotation. However, the resulting derivations often contain errors. This can be seen in the example above, where *lung cancer* should form a constituent, but does not. This is a major deficiency in CCGbank, and one of the principal contributions of this thesis will be to correct it.

A further problem with NP derivations in CCGbank is shown in Figure 2.5. When a conjunction occurs inside an NP, a non-CCG rule is required in order to reach a parse:

$$conj\ N \quad \Rightarrow \quad N \tag{2.9}$$

This is due to the default right-branching interpretation assigned to the derivation. The rule treats the conjunction in the same manner as a modifier, and results in the incorrect derivation shown in Figure 2.5. Rather than *cotton* and *acetate* being conjoined to form a multi-headed construct, this derivation treats them as separate modifiers.

2.5.2 CCG Parsing

Once CCGbank became available (or perhaps, while it was still under development is more accurate) statistical parsers using this new corpus began to be constructed. Two early models were those of Clark, Hockenmaier, and Steedman (2002) and Hockenmaier and Steedman (2002).

The former model was based on Collins (1996), in that it divides the probability of a sentence into two parts: the probability of the lexical category sequence and the probability of the dependencies between the lexical categories. The former is calculated using the Clark (2002) supertagger (described in the following paragraph), while the latter is done with relative frequencies from CCGbank, in the same way that Collins used the Penn Treebank.

Supertagging (Bangalore and Joshi, 1999) is a sequence tagging task in the same vein as POS tagging. However, rather than assigning 48 POS tags, the task is to assign the lexical categories of a lexicalised grammar. This was originally proposed for parsing with TAG, with the set of elementary trees corresponding to the set of supertags. However, the automatically extracted TAG grammars were too large to be recovered accurately. Supertagging has been effective in increasing parser efficiency when applied to HPSG (Blunsom and Baldwin, 2006; Matsuzaki, Miyao, and Tsujii, 2007).

In CCGbank, there are 1,206 lexical categories, however a large number of these occur only once. The Clark (2002) supertagger (and subsequent CCG supertaggers) applied a cutoff, including only those categories that occurred ten or more times. The 398 supertags remaining still cover 96.5% of sentences in Section 00.

Returning to our two CCG parsing models, Clark, Hockenmaier, and Steedman (2002) deal with spurious ambiguity as part of the chart pruning performed by the parser. Constituents that share a head word and lexical category with another constituent in the same cell, but do not have equal or higher probability, are not added to the chart. The parser achieves 81.7% F-score recovering predicate-argument dependencies from all sentences in Section 23.

The Hockenmaier and Steedman (2002) model is similar to Collins (1997) or Charniak (1997). The CCG tree is generated top-down, with children conditioned on the previous node. Spurious ambiguity is dealt with by conditioning probabilities on the normal-form derivations in CCGbank. These include the use of composition and type-raising, although they are unlikely. As a result, non-normal-form derivations can be produced, but they will have low probability. This parser achieves 83.9% F-score.

The Hockenmaier and Steedman (2002) model can be thought of as Combinatory Categorial Grammar within a PCFG. The small number of CCG rules are not actually being applied freely, and instead the corpus is treated in the same way as earlier experiments with the Penn Treebank's Phrase Structure grammar. That is, the advantages of CCG, such as its ability to intrinsically recover long-range dependencies, are not being taken advantage of.

Hockenmaier (2003b) presents another parser that does model the predicate-argument dependencies in CCG derivations. However, including the long-range dependencies is problematic for the dynamic programming algorithm. Because heads can now be conditioned on multiple other words — for example, a verb in a relative clause must also be conditioned on the noun which it modifies — their inside probability cannot be calculated until the additional conditioning head's probability has also been determined. To solve this problem, the equivalence between constituents in the chart is approximated, while an aggressive beam search prunes cells with over 100 entries.

Even with these measures, the model is only efficient enough to parse sentences of 40 words or less. Its results are also worse than the previous models, achieving only 81.5% F-score on the predicate-argument dependencies.

The C&C parser

Clark and Curran (2004b; 2007b) present the C&C parser, which uses a discriminative Maximum Entropy (also known as log-linear) model. This allows multiple complex features to be included more easily than the probability distributions of earlier CCG parsers. The negative training examples required by the discriminative model are created by applying all possible CCG rules, rather than only those that reach the gold-standard derivation. In order to store this tremendous number of derivations efficiently, the *packed chart* representation of Miyao and Tsujii (2002) is applied. In this dynamic programing approach, constituents that generate the same subsequent parse structure and dependencies are treated as equivalent. The models that are created are still massive, requiring over 25GB of memory during the training process. A cluster of computers train in parallel in order to manage this immense quantity of data.

The C&C parser, like the earlier Clark, Hockenmaier, and Steedman (2002) model, uses a supertagger to supply lexical categories. The supertagger, described in Clark and Curran (2004a), actually suggests multiple categories as long as they are within some parameterised ratio of the most probable category, again in the same way as the earlier parser. This serves to limit the ambiguity that the parser itself must resolve, while still supplying a number of categories to choose between. The parser may find that a category deemed less probable by the supertagger is actually the most likely. This may be because the original supertags do not form a spanning analysis or because of the larger context and different feature space that the parser considers.

Clark and Curran (2007b) present two models for dealing with spurious ambiguity: a normal-form model, which calculates the probability for a single normal-form derivation; and a dependency model, which models predicate-argument dependencies directly, thus including non-normal-form derivations. The former model is the simpler of the two and similar to that of Hockenmaier and Steedman (2002), while the latter is closer to the Hockenmaier (2003b) model. For the dependency model to optimise the log-linear model during training, a sum over only the correct derivations is required. They must therefore be identified within the feature forest. Another complication occurs during decoding, where summing over all of the dependency structures in the packed chart is too time-consuming an operation. Instead, Clark and Curran (2004; 2007b) use an algorithm adapted from Goodman (1996), which maximises the expected recall of the dependencies. This only requires a sum over individual dependencies, and can thus take advantage of dynamic programming in the packed chart.

The dependency model is initially superior, achieving 85.73% F-score compared to the normal-form model's 85.08%. However, the normal-form model requires significantly smaller charts and thus uses less memory. The supertagger can supply additional categories to the parser, increasing normal-form performance to 86.73%. A comparable increase is not feasible for the dependency model, as the chart size is limited by the memory available in the cluster. Instead, Clark and Curran (2007b) suggest a hybrid model, using the dependency model and also applying Eisner constraints, which reduces the size of the chart in a different way. With this model, their best performance figure of 87.24% F-score is attained.

2.6 Maximum Entropy Models

The C&C parser that we have just described, as well as the machine learner that we will use in Chapter 5, MegaM (Daumé III, 2004), both use a Maximum Entropy model. Berger, Pietra, and Pietra (1996) describe such models, and we will give a brief description here. The intuition behind a Maximum Entropy model is to maximise the likelihood of the training data, but to make no further assumptions that there is no evidence for. That is, the best model will have the most uniform distribution possible, while still agreeing with the feature distribution observed in the data. Entropy can be used to measure the uniformity of the distribution, hence the best model will maximise this value, giving the model its name.

Each ⟨feature, class⟩ pair in a Maximum Entropy model is assigned a corresponding weight λ by the training process. The λ values indicate how probable each class is for the feature. For example, a POS tagger would have a high λ for the word *the* being a DT and a negative λ for all other classes. There is no closed-form solution to training a Maximum Entropy model, and so setting the λ parameters is treated as a standard numerical optimisation problem. Solving this problem can be done in a number of ways, all of which iteratively move the values estimated by the model closer to the empirical values observed in the training data. The simplest method is Generalised Iterative Scaling (GIS) (Darroch and Ratcliff, 1972), although convergence is rather slow with this algorithm. The C&C parser and MegaM both use the faster converging limited memory BFGS (L-BFGS) algorithm (Nocedal and Wright, 1999).

Once the λ parameters are set, the probability of test example x having the class y can be calculated as follows:

$$p(y|x) = \frac{1}{Z} \exp(\sum_i \lambda_i f_i(x, y)) \tag{2.10}$$

Z is the sum over all alternatives for y, ensuring a proper probability distribution. The feature functions f typically return binary values, thus determining which features are active and which of the λ weights are summed.

There are many advantages to using Maximum Entropy models. Firstly, and unlike in a Naive Bayes classifier, the model makes no conditional independence assumptions. Related features can be included in the model with no degradation in performance, as dependent features will share probability mass between themselves. Secondly, Maximum Entropy models can include a massive number of features. Feature selection is, in a sense, included by the model, as uninformative features should receive a weight near zero. Thirdly, features can be arbitrarily complex, and encapsulate any kind of information. This makes Maximum Entropy models easy to develop and experiment with. The ability of Maximum Entropy models to efficiently deal with a huge feature space makes them ideal for NLP tasks.

2.7 Summary

In this chapter, we described why NP structure is not analysed by most statistical parsers. The Penn Treebank does not annotate the necessary NP structure, which means that most parsers have no data on which to train or evaluate. This has also affected CCGbank, where the binary branching CCG grammar is unable to replicate the flat, underspecified NPs of the Penn Treebank. Incorrect derivations are subsequently created for all left-branching NPs, while NP derivations with conjunctions are also analysed incorrectly.

We gave a brief description of a number of parsing models, providing a starting point for when we go into detail in later chapters. The Collins (2003) model is the focus of Chapter 4, while Chapter 6 presents experiments with the Clark and Curran (2007b) parser. In particular, we will focus on how these models generate NPs. This information will motivate the experiments that are performed and help to explain the results that are achieved. Additional background material will be presented throughout the thesis as needed.

Chapter 3

Annotating Noun Phrases

The first step to statistical parsing of NPs is to create a gold-standard data set which can be used for machine learning. This chapter will describe the process of manually annotating such a corpus of NP structure. This data will be used throughout the thesis, in the parsing experiments of Chapters 4 and 6 and the NP Bracketing experiments in Chapter 5. Extending the Penn Treebank annotation scheme and corpus is one of the major contributions of this thesis.

There are a handful of corpora annotated with NP structure already, however these do not meet our requirements. DepBank (King et al., 2003) fully annotates NPs, as does the reannotated Briscoe and Carroll (2006) version. However this corpus consists of only 700 sentences. The Biomedical Information Extraction Project (Kulick et al., 2004) introduces the use of NML nodes to mark internal NP structure in its Addendum to the Penn Treebank Bracketing Guidelines (Warner et al., 2004). However, this corpus is specifically focused on biomedical text, rather than newspaper text. We still base our approach to bracketing NP structure on these biomedical guidelines, as the grammatical structure being annotated remains similar.

We chose to augment the Wall Street Journal section of the Penn Treebank with the necessary NP structure, as it is the corpus most widely-used in the parsing field for English. This also meant that the NP information would not need to be imported from a separate model, but could be included into existing parsers and their statistical models with a minimum of effort.

3.1 Annotation Guidelines

One principle we applied during the augmentation process was to avoid altering the original Penn Treebank annotations. This meant that results achieved with the extended corpus would be comparable to those achieved on the original, as much as possible. We created a set of guidelines in order to aid in the annotation process and to keep the result consistent and replicable. Appendix A presents these guidelines in full, however we give a general description of our annotations, together with a number of examples here.

Our approach is to leave right-branching structures unaltered, while labelled brackets are inserted around left-branching structures.

```
(NP (NN world) (NN oil) (NNS prices) )
```

```
(NP (NML (NN crude) (NN oil) )
  (NNS prices) )
```

Left and right-branching NPs are now differentiated. Although explicit brackets are not added to right-branching NPs, they should now be interpreted as having the following implicit structure:

```
(NP (NN world)
  (NODE (NN oil) (NNS prices) ) )
```

This representation was used in the biomedical guidelines, and has many advantages. By keeping right-branching structure implicit, the tree does not need to be binarised. Binarisation can have a harmful effect on parsers using PCFGs, as it reduces the context-sensitivity of the grammar (Collins, 2003, page 621). It also reduces the amount of clutter in the trees, making them easier to view and annotate. Right-branching structure can still be added automatically if required, as we experiment with in Section 4.4.1. However, not inserting it makes the annotator's task simpler.

The label of the newly created constituent is NML (*nominal modifier*), as in the example above, or JJP (*adjectival phrase*), depending on whether its head is a noun or an adjective. Examples using the JJP label are shown below:

```
(NP (JJP (JJ dark) (JJ red) )
  (NN car) )
```

```
(NP (DT the)
  (JJP (JJS fastest) (VBG developing) )
  (NNS trends) )
```

Rather than this separate JJP label, the biomedical treebank replicates the use of the ADJP label in the original Penn Treebank. We wanted to be able to distinguish the new annotation from the old in later experiments, which required the creation of this additional label. JJPs can easily be reverted back to ADJP later on, as will be done in Section 4.2.1, but the reverse would be impossible, because the separate annotations would not be differentiable.

Non-base-NPs may also need to be bracketed, as shown below:

```
(NP-SBJ
  (NML (JJ former)
    (NAC (NNP Ambassador)
      (PP (TO to)
        (NP (NNP Costa) (NNP Rica) ) ) ) )
  (NNP Francis) (NNP J.) (NNP McNeil) )
```

In this example, we join *former* and the NAC node, as he is formerly the *Ambassador*, not formerly *Mr. McNeil*.

Speech marks and brackets are annotated explicitly:

```
(NP-PRD (DT a)
  (NML (‘‘ ‘‘) (JJ long) (NN term) (’’ ’’) )
  (NN decision) )


(NP (DT an)
  (JJP (-LRB- -LCB-) (VBG offending) (-RRB- -RCB-) )
  (NN country) )
```

NMLs themselves can be nested:

```
(NP
  (NML
    (NML (NNP New) (NNP York) )
    (NNP Stock) (NNP Exchange) )
  (JJ composite) (NN trading) )
```

This correct bracketing describes *composite trading* on the *Stock Exchange* of *New York*.

Many conjunctions need to be bracketed, as in the following examples:

```
(NP (DT the)
  (NML (NNPS Securities)
    (CC and) (NNP Exchange) )
  (NNP Commission) )

(NP (PRP$ its)
  (JJP (JJ current)
    (CC and) (JJ former) )
  (NNS ratepayers) )
```

Without these brackets, the NP's implicit structure, as shown below, would be incorrect.

```
(NP (DT the)
  (NODE
    (NODE (NNPS Securities) )
    (CC and)
    (NODE (NNP Exchange) (NNP Commission) ) ) )
```

The erroneous meaning here is *the Securities* and *the Exchange Commission*, rather than the correct *the Securities Commission* and *the Exchange Commission*.

The previous NPs are also good examples showing that we do bracket clear internal structure in company names. Our argument for doing so is a pragmatic one, similar to the Question Answering (QA) example given in Section 1.1. Consider a QA question:

What city's stock exchange fell by over 3% yesterday?

Without a NML bracket around *New York*, it would not be identifiable as a candidate, and thus the question would not be answered correctly.

Compare this to a person's name, which we do not add substructure to, as described in Section A.2.6. However, this instruction is only to prevent an annotator from trying to decide whether a first name attaches to a middle name, or directly onto the last name. If a person's name was *John New York Smith*, where *New York* was a nickname, then the internal structure would be bracketed.

3.1.1 Difficult Cases

During the annotation process, we encountered a number of NPs that were difficult to bracket. The main cause of this difficulty was technical jargon, for example in the phrase *senior subordinate reset discount debentures*. The Penn Treebank guidelines devote an entire section to this *Financialspeak* (Bies et al., 1995, §23). The biomedical guidelines similarly contain some examples that are difficult for a non-biologist to annotate:

```
liver cell mutations
p53 gene alterations
ras oncogene expression
polymerase chain reaction
```

However, even these NPs were simple to bracket for an expert in the biological domain. We did find that there were relatively few NPs which the annotator clearly understood, but still had difficulty bracketing. This agrees with our hypothesis in Section 2.2.1, that modifier scope in NPs is resolvable.

For those difficult-to-bracket NPs that were encountered, we bracket what structure is clear and leave the remainder flat. This is described in Section A.1.2 of our NP annotation guidelines. The biomedical guidelines (Warner et al., 2004, §1.1.5) also take this approach, which can be compared to how ambiguous attachment decisions are bracketed in the Penn Treebank. Bies et al. (1995, §5.2.1) says *"the default is to attach the constituent at the highest of the levels where it can be interpreted"*. Section 7.1.1 talks in more detail about the Penn Treebank method for bracketing ambiguous structure, and how it is relevant to our own annotations.

We also observed a small percentage of *semantically indeterminate* NPs during our own annotation process. Hindle and Rooth (1993) noted this problem for the prepositional phrase attachment task. For example, given the following sentence, with the relevant tokens for the prepositional phrase attachment in bold:

We have not ***signed*** *a settlement* ***agreement with*** *them.*

Hindle and Rooth (1993, page 112) say

The problem... is that signing an agreement usually involves two participants who are also parties to the agreement.

That is, there is a dependency between the preposition and the verb as well as between the preposition and the noun. Or, semantically, it makes no difference because one implies the other. The same problem can occur in NPs such as *college basketball players*, where the *players* are in *college* and they play *basketball*. These NPs do not fit into the left or right-branching paradigm. However it did not become clear until after the annotation process that these semantically indeterminate NPs should be distinguished from implicitly right-branching ones. Once again, Section 7.1.1 has a fuller discussion on this subject.

3.1.2 Divergences from the Biomedical Guidelines

One notation from the biomedical guidelines that we have not used is the marker `*P*`, to denote shared modifiers or heads:

```
(NP (NP (ADJP-1 bronchial)
        washings)
    and
    (NP (ADJP-1 *P*)
        brushings))
```

```
(NP (NP K-
        (NML-1 *P*))
    and
    (NP N-
        (NML-1 ras)))
```

Note that *P* is not a trace, as the guidelines explicitly make clear, but a placeholder for the shared constituent. The *P* notation is also used by the biomedical guidelines to make a distinction between intersective and disjunctive adjectival modifiers. The former, where the coordinated modifiers overlap to some degree, form a constituent node:

```
(NP (ADJP strong and competitive)
    athletes)
```

While the latter, where the sets do not overlap at all, use the *P* placeholder:

```
(NP (NP cancerous
        (NML-1 *P*))
    and
    (NP (NML non - cancerous)
        (NML-1 cells)))
```

Quantifier scope is discussed in more detail in Reinhart (1997), while Kamp and Partee (1995) look at the semantic effect when such modifiers are composed.

We have not used the *P* marker in our annotations, as differentiating between intersective and disjunctive adjectives is a subtle semantic distinction that is not part of the syntactic analysis in this thesis. The Penn Chinese Treebank (Xue et al., 2005) and the Penn Arabic Treebank (Maamouri et al., 2003) do not use *P* either, and the biomedical guidelines themselves point out that phrases with coordinated modifiers *and* coordinated heads can't be fully annotated with this mechanism. The biomedical guidelines give the example: *the N- and K- ras cells and tumors*, where a placeholder cannot be included for each of the four combinations: *N- ras cells*, *N- ras tumors*, *K- ras cells* and *K- ras tumors*. Thus we argue that the placeholder is unnecessary, while requiring extra work of the annotators, and so it has not been applied.

Another difference between the biomedical corpus and our extended Penn Treebank stems from our principle of not altering pre-existing brackets. The biomedical project is able to improve the analysis of some treebank structures, as it is annotating a brand new corpus. Hyphenated tokens were separated, solving difficulties in phrases like *New York-based*, where we would like to bracket *New York* but cannot. Also, NMLs are used instead of NX and NP-internal NAC nodes. These changes have not been replicated in our NP annotations. Instead, phrases like *New York-based* remain flat in our extended Penn Treebank; NMLs have to be inserted around existing NX and NAC nodes; and annotator errors that prevent the correct NP analyses being applied have not been corrected. Section 7.1.2 provides an in-depth discussion of this problem as well as suggesting possible solutions.

PATTERN	EXAMPLE
`* CC *`	*Bill and Ted*
`$ * * -NONE-`	*$ 1 million *U**
`DT * *`	*the red car*
`PRP$ * *`	*his red car*
`* * POS`	(`NML` *John Smith) 's*

Table 3.1: Unambiguous POS tag patterns

3.2 Annotation Tool

A bracketing tool was developed to identify ambiguous NPs and present them to the annotator. An ambiguous NP is any (possibly non-base) NP with three or more contiguous children that are either single words, i.e. leaves, or nested NPs. Some common parse tree patterns are unambiguous, and so were filtered out, that is, they were not shown to the annotator. The entire list of patterns is shown in Table 3.1. The bottom-most pattern in the table is left-branching and so a `NML` bracket is automatically inserted around the first two tokens, as shown in the example.

In order to better inform the annotator, the tool also displayed the entire sentence surrounding the ambiguous NP. During the annotation process, most NPs could be bracketed without specifically reading this information, because the NP structure was clear and/or because the annotator already had some idea of the article's content from the NPs (and surrounding context) shown previously. In those cases where the surrounding sentence provided insufficient context for disambiguation, it was typically true that no amount of surrounding context was informative. For these NPs, the principle of leaving difficult cases flat was applied.

3.2.1 Automatic Bracketing Suggestions

We designed the bracketing tool to automatically suggest a bracketing, using rules based mostly on named entity tags. These NER tags are drawn from the BBN Pronoun Coreference and Entity Type Corpus (Weischedel and Brunstein, 2005). This corpus of gold-standard data annotates 28 different entity tags, some of which are shown in Table 3.2. Some of the NER tags have subcategories, for example, **`GPE`** is divided into **`Country`**, **`City`**, **`State/Province`** and **`Other`**, however we only use the coarse tags for the annotation tool suggestions.

This NER information is useful, for example in bracketing the NP *Air Force contract*. Because *Air Force* is marked as an organisation, the tool can correctly suggest that the NP is left-

CATEGORY	DESCRIPTION	EXAMPLES
PERSON	Proper names of people	*George Bush*, *Mickey Mouse*
PER_DESC	Descriptions of people	*vice president*, *Chief Executive*
NORP	Nationality, Other, Religion or Political	*Brazilian*, *Catholic*, *Arab*
ORGANIZATION	Names of organizations	*FBI*, *CNN*, *Taliban*
ORG_DESC	Entities describing organizations	*offices*, *school*, *party*
GPE	Countries, cities, states and other places	*Sydney*, *Australia*
GPE_DESC	Descriptions of countries and other places	*nation*, *countries*
LOCATION	Named locations	*Silicon Valley*, *Europe*, *Pacific*
DATE	A date or period of time	*yesterday*, *March 30*
CARDINAL	Numbers, including decimals and fractions	*4000*, *half*, *18.20*
SUBSTANCE	Chemicals, elements, drugs and foods	*beef*, *tea*, *uranium*
WORK_OF_ART	Books, songs and other creations	*Nobel Prize*, *The Bible*

Table 3.2: Some of the named entity categories in the BBN corpus

branching. Using NER tags is more informative than simply looking for NNP POS tags, as there are many common nouns that are entities, e.g. *vice president* is a **PER_DESC**.

The tool also suggests bracketings based on the annotator's previous decisions. Whenever the annotator inserts a bracket, the current NP and its structure, together with the label and placement of the new bracket, is stored. Then, whenever the same NP and structure is seen in the future, the same bracketing is suggested. This source of suggestions is particularly important, as it helps to keep the annotator consistent.

Other suggestions are based on gazetteers of common company and person name endings. Preliminary lists were generated automatically by searching for the most frequently occurring final tokens in the relevant named entities. Certain incorrect items were then removed from the lists by manual inspection. Some of the most common examples are *Co.* and *Inc* for companies and *Jr* and *III* for people's names. Full lists are presented in Appendix B.

The guidelines also mandate the insertion of nodes around brackets and speech marks. These are detected automatically and included in the suggestion system accordingly. Unbalanced quotes do not result in any suggestions.

The last source of suggestions is final possessives, as in *John Smith 's*. In these cases, a bracket around the possessor *John Smith* is suggested.

3.3 Annotation Process

Once the guidelines and annotation tool were complete, annotation of the corpus began. I[1] manually annotated the entire Wall Street Journal section of the Penn Treebank. Matthew Honnibal[2] annotated Section 23 in an additional, independent pass. Matthew is a graduate in liguistics, and we were both Computational Linguistics PhD students at the University of Sydney. Matthew and my supervisor, James Curran, also contributed to how certain NP constructions were bracketed through the many discussions we had.

Having a second annotator helped to ensure the reliability of the annotations, by allowing inter-annotator agreement to be measured (see Section 3.4.1) and also maximised the quality of the section used for parser testing. Over 60% of sentences in the corpus were manually examined as part of the annotation process.

3.3.1 Annotation Time

Initially, over nine hours of continuous annotation were required per section of the Treebank. However, with practice this was reduced to about three hours per section. Each section contains around 2,500 ambiguous NPs, i.e. annotating took approximately five seconds per NP. Most NPs required no bracketing, or fitted into a standard pattern which the annotator soon became accustomed to, and hence the task could be performed quite quickly.

As a comparison, during the original creation of the Treebank, annotators performed at 375–475 words per hour after a few weeks, and increased to about 1000 words per hour after gaining more experience (Marcus, Santorini, and Marcinkiewicz, 1993). For our annotations, we expected to be in the middle of this range, as the task was not large enough to gain more than a month's experience, or perhaps faster as there is less structure to annotate. The actual figure, calculated by counting each word in every NP shown, is around 800 words per hour. This matches the expectation quite well.

3.3.2 Annotation Post-Processes

In order to increase the reliability of the corpus, a number of post-processes have been carried out since the annotation was first completed. Firstly, 915 NPs were marked by the annotator as difficult during the main annotation phase. In discussion with two other computational linguists,

[1] To make things perfectly clear, the first-person pronoun is used here, rather than an editorial *We*.
[2] To who I am very grateful.

the best bracketing for these NPs was determined. Secondly, 241 phrases that occurred numerous times and were non-trivial to bracket, e.g. *London Interbank Offered Rate*, were identified. An extra pass was made through the corpus, ensuring that every instance of these phrases was bracketed consistently.

The main annotator made a second pass over the corpus in order to change the standard bracketing for conjunctions, speech marks and brackets. These changes, from version 0.9 to 1.0 of the guidelines, were aimed at increasing consistency and bringing our annotations more in line with the biomedical guidelines (Warner et al., 2004). For example, *royalty and rock stars* was previously left flat, as the implicit right-branching structure correctly indicates that *stars* is not shared. However if the conjuncts were in the reverse order, *rock stars and royalty*, then a bracket would need to be inserted around *rock stars*. This inconsistency, where a reordered phrase with the same meaning was given a different bracketing, was removed by applying a bracket to both conjuncts in all cases. This change meant that there were NML nodes spanning only a single token in version 1.0, whereas in version 0.9 there were not.

In this second pass, only those NPs that had at least one bracket inserted during the first pass were manually inspected. The exception to this was NPs with a conjunction followed by multiple tokens, such as *president and chief executive officer*, which also needed to be reannotated. By only reanalysing this subset of ambiguous NPs, the annotator's workload was reduced, while still allowing for a number of errors to be noted and corrected. This second phase of annotation was carried out in less than a week.

Lastly, we identified all NPs with the same word sequence and checked that they were always bracketed identically. Those that differed from the majority bracketing were manually reinspected and corrected as necessary. However, even after this process, there were still 48 word sequences by type (201 by token) that were inconsistent. In these remaining cases, such as the NP below:

```
(NP-TMP (NML (NNP Nov.) (CD 15))
  (, ,)
  (CD 1999))
```

we were inconsistent in inserting the NML node because the Penn Treebank sometimes already has the structure annotated under an NP node. Hogan (2007) notes the same problem with conjunctions. Although the Penn Treebank guidelines say that coordinate NPs with only nominal modifiers should be left flat, Hogan finds that 21.3% of these NPs are annotated with some internal structure. Since we do not make changes to existing brackets, we cannot correct these cases. We only change the

	COUNT	% (OF PREVIOUS)	% (OF TOTAL)
NPs	432,639	–	100.00
Ambiguous NPs	60,959	14.09	14.09
Annotated NPs	23,129	37.94	5.35
NML nodes	26,372	114.02	6.10
JJP nodes	894	3.87	0.21

Table 3.3: Counting the corpus

inconsistency from flat or structured to labelled NML or NP. Unfortunately, this will cause difficulty for a parser deciding which label to use, as we will find in Section 4.6.

3.4 Corpus Analysis

Table 3.3 presents an analysis of the NP structure that has been added to the Penn Treebank. There are over 430 thousand NPs in the corpus, over 60 thousand of which were manually inspected by the annotator. Over 60% of these NPs required no bracketing. These are not all right-branching NPs, as the count is affected by appositions, prepositional phrases, and other non-base-NP structures that were not annotated.

There are more NML nodes than annotated NPs because of cases where multiple new brackets are added to a single NP. There are 2,867 such multiply bracketed NPs, comprising 4.70% of all ambiguous NPs or 12.40% of annotated NPs. The most nested left-branching NP has a depth of 4, one example of which is:

```
(NP
  (NP
    (NML (NNP Boston) (NNP Co.) )
    (POS ’s) )
  (NML
    (NML (NNP Boston)
      (NML
        (NML (NNP Safe) (NNP Deposit) )
        (CC &)
        (NML (NNP Trust) ) ) )
    (NNP Co.) )
  (NN subsidiary) )
```

Quirk et al. (1985, page 1338) notes that NPs rarely have more than three or four premodifiers,

LEVEL	COUNT	POS TAGS	EXAMPLE
NP	3557	NNP NNP NNP	*John A. Smith*
	2453	DT NN POS	*(the dog) 's*
	1693	JJ NN NNS	*high interest rates*
	1663	NNP NNP POS	*(John Smith) 's*
NML	8605	NNP NNP	*(John Smith) Jr.*
	2475	DT NN	*(the dog) 's*
	1652	NNP NNP NNP	*(A. B. C.) Corp*
	1486	NN NN	*(interest rate) rises*
JJP	162	`` JJ ''	*(" smart ") cars*
	120	JJ CC JJ	*(big and red) apples*
	112	RB JJ	*(very high) rates*
	78	JJ JJ	*(dark red) car*

Table 3.4: Common POS tag sequences

which puts the same limit on nested structures that we have found here. There are only four NPs with this maximal depth in the entire corpus.

Another finding in Table 3.3 is that JJP nodes form a clear minority, accounting for a very small amount of the total brackets. To compare, if we count the number of existing NP and ADJP nodes in ambiguous NPs, we find 32,772 and 579 brackets respectively. These figures are quite similar to the number of NML and JJP nodes we have added, with adjectival modifiers occurring much less frequently in both cases. These result show that the annotation process has introduced almost as much structural information into NPs as there was in the original Penn Treebank.

In Table 3.4, the most common POS tag sequences for NP, NML and JJP nodes are shown. An example is given showing typical words that match the POS tags. For NML and JJP, the example shows the complete NP node, rather than just the NML or JJP bracket. It is interesting to note that RB JJ sequences are annotation errors in the original Treebank, and should have an ADJP bracket already.

3.4.1 Inter-Annotator Agreement

To determine the correctness and consistency of our corpus, we calculated inter-annotator agreement on Section 23. Note that the second annotator was following version 0.9 of the bracketing guidelines, and since then they have been updated to version 1.0. Because of this, we can only analyse the 0.9 version of the corpus, i.e. before the primary annotator made the second pass

	PREC.	RECALL	F-SCORE
Brackets	89.17	87.50	88.33
Dependencies	96.40	96.40	96.40
Brackets, revised	97.56	98.03	97.79
Dependencies, revised	99.27	99.27	99.27

Table 3.5: Agreement between annotators

mentioned in Section 3.3.2.[3] This is not problematic, as the definition of what constitutes a `NML` or `JJP` node has not changed, only their representation in the corpus. That is, the dependencies that can be drawn from the NPs remain the same. This can be seen in the example mentioned earlier, *royalty and rock stars*, where the unbracketed 0.9 version leaves implicit what the version 1.0 scheme makes explicit.

A standard inter-annotator agreement measure such as Kappa is difficult to apply to our data, as the many NPs where no bracket is added create zeroes that make the ratios uninformative. Instead, we measure the proportion of matching brackets and dependencies between annotators, by taking one as a gold-standard and then calculating precision, recall and F-score. Table 3.5 shows the results, both before and after cases of disagreement were discussed and the annotations revised. The number of dependencies is fixed by the length of the NP, so the dependency precision and recall are the same. Counting matched brackets is a harsher evaluation, as there are many NPs that both annotators agree should have no additional bracketing, which are not taken into account by the metric. For example, consider an NP that both annotators agree is right-branching.

```
(NP (NN world) (NN oil) (NNS prices))
```

The agreement score is not increased by the matched bracket evaluation here, as there is no `NML` or `JJP` bracket. A dependency score on the other hand, would find two matching dependencies (between *world* and *prices* and *oil* and *prices*), increasing the inter-annotator agreement measure accordingly.

We can also look at exact matching on NPs, where the annotators originally agreed in 2667 of 2908 cases (91.71%), and after revision, in 2864 of 2908 cases (98.49%). Again, this is a harsher evaluation as partial agreement is not taken into account.

All of these inter-annotator figures are at a high level, thus demonstrating that the task of identifying nominal modifier scope can be performed consistently by multiple annotators. We have

[3]Although the subsequent consistency checks described there *had* been carried out, and were applied again afterwards.

attained high agreement rates with all three measures, and found that even difficult cases could be resolved by a relatively short discussion.

When the annotators discussed how differing brackets should be revised, we observed that most of the disagreements that occurred were a result of the second annotator's inexperience with the task. While the first annotator had already learnt from the majority of the corpus, this was the second annotator's first attempt at the task. Furthermore, we specifically avoided helping the second annotator, aside from going through the guidelines briefly, in order to avoid creating bias. Thus, the agreement figures we have given probably underestimate what more practiced annotators would achieve.

The bracketing guidelines were revised as a result of the post-annotation discussion between the annotators, to be more specific on those cases where the disagreements had occurred. The disagreements after revision occurred for a small number of repeated instances, such as this case:

```
(NP                                   (NP (NNP Goldman)
  (NML (NNP Goldman)                    (, ,)
    (, ,)                               (NNP Sachs)
    (NNP Sachs) )                       (CC &) (NNP Co) )
  (CC &) (NNP Co) )
```

The second annotator felt that *Goldman , Sachs* should form its own NML constituent, while the first annotator did not.

We would like to be able to compare our inter-annotator agreement to that achieved in the original Penn Treebank project. Marcus, Santorini, and Marcinkiewicz (1993) describe a 3% estimated error rate for their POS tag annotations, but unfortunately, no figure is given for bracketing error rates. As such, a meaningful comparison between the NP annotations described here and the original Penn Treebank cannot be made.

3.4.2 DepBank Agreement

Another approach to measuring annotator reliability is to compare with an independently annotated corpus of the same text. We used the Briscoe and Carroll (2006) version of the PARC700 Dependency Bank (King et al., 2003). These 560 sentences from Section 23 of the Penn Treebank are annotated with labelled dependencies, and are used to evaluate the RASP parser.

This is not a trivial task, and a significant effort is required to compare our brackets to DepBank dependencies. We map the brackets to dependencies by finding the head of the NP, using the Collins (2003) head finding rules, and then creating a dependency between each other child's

	MATCHED	TOTAL	%
By dependency	1027	1114	92.19
By noun phrase	358	433	82.68
By dependency, only annotated NPs	476	541	87.99
By noun phrase, only annotated NPs	150	203	73.89

Table 3.6: Agreement with DepBank

head and this head. We do not attempt to match the labels used in the Briscoe and Carroll reannotation, and so this is an unlabelled comparison. The results are shown in Table 3.6. Once again the dependency numbers are higher than those at the NP level.

Matched brackets cannot be evaluated as before, because the DepBank representation uses dependencies. Converting these to a bracketed representation would be even more complex than the translation we have carried out from Penn Treebank brackets to dependencies. Instead, we supply a figure for only those cases where annotations have been inserted, which is similar in effect. This can be seen in the *world oil prices* example above, where there is no matched bracket to evaluate, because no brackets have been added. As expected, these are more difficult cases and the score is not as high.

The results of this analysis are quite positive, however in actuality, they are even better than they appear. Clark and Curran (2007a) also performed a cross-formalism evaluation with DepBank, finding that their conversion method only achieved 84.76% F-score on labelled dependencies, even when using gold-standard data. In the same way, our agreement figures could not possibly reach 100%, as the automatic conversion between the two corpora does not work perfectly. Accordingly, the errors were manually investigated to determine their cause, with the results shown in Table 3.7.

True disagreement between the Briscoe and Carroll (2006) annotations and ours is only the second most common cause. In the example, the complete sentence is: *These "clean-bank" transactions leave the bulk of bad assets, mostly real estate, with the government, to be sold later.* We annotated *mostly real estate* as a right-branching NP, i.e. with dependencies between *mostly* and *estate* and *real* and *estate*. Briscoe and Carroll form a dependency between *mostly* and *real*.

The largest source of disagreements arises from how company names are bracketed. While we have always separated the company name from postmodifiers such as *Corp* and *Inc*,

ERROR TYPE	COUNT	EXAMPLE NP
Company name postmodifier	26	*Twenty-First Securities Corp*
True disagreement	25	*mostly real estate*
Head finding error	21	*Skippy the Kangaroo*
Determiner + quantifier	6	*the last 45 minutes*
No DepBank dependency	5	*other OTC issues*
Penn Treebank error	4	*a shareholders '*

Table 3.7: Disagreement analysis with DepBank

DepBank does not in most cases. The other large cause of annotation discrepancies comes as a result of the head-finding rules. In these cases, the DepBank dependency will often be in the opposite direction to the Penn Treebank one, or the head found by Collins' rules will be incorrect. For example, in the NP *Skippy the Kangaroo*, the Collins' head-finding rules identify *Kangaroo* as the head, while the DepBank head is *Skippy*. In both cases, a dependency between the two words is created, however the direction is different and so no match is found.

Other repeated errors come from phrases with a determiner followed by a quantifier, e.g. *the last* in *the last year* and *a few* in *a few exact questions*. A dependency between the two tokens is present in DepBank, while the NP was left flat in our annotations.

Some tokens do not form dependencies in the Briscoe and Carroll (2006) reannotation. In the NP *other OTC issues* for example, there is no dependency that mentions *other*. Although the original DepBank annotations (King et al., 2003) include the following dependency: *adjunct(issue, other)*, Briscoe and Carroll only form this dependency: *(ncmod _ issues OTC)*. It is unclear why *other* is not included in the reannotated corpus.

Finally, there are errors in the Penn Treebank which make finding a matching dependency impossible. For example, consider the following bracketing:

```
(NP
  (NP (DT a) (NNS shareholders) (POS ') )
  (NN meeting) )
```

This incorrect annotation suggests that the meeting is for a single shareholder, rather than forming a dependency between *a* and *meeting* as it should. As no Penn Treebank brackets were altered as part of the NP augmentation process, this problem could not be resolved. As a result, although we would like to agree with the DepBank annotation here, there is no possible bracketing which makes it so.

Even without taking these problems into account, these results show that consistently and

SUGGESTIONS USED	PREC	RECALL	F-SCORE
NER only	94.16	32.57	48.40
All	94.84	**54.86**	**69.51**
−NER	**97.46**	41.31	58.02
−Company and name endings	94.55	41.42	57.60
−Brackets and speech marks	95.03	50.62	66.05
−Possessives	94.51	50.95	66.20
All, Section 00	95.64	59.36	73.25
All, Section 23	94.29	56.81	70.90

Table 3.8: Suggestion rule performance

correctly bracketing noun phrase structure is possible, and that inter-annotator agreement is at an excellent level.

3.5 Evaluating the Annotation Tool's Suggestions

This last analysis of our corpus evaluates the annotation tool's suggestion feature. This will serve as a baseline for NP bracketing performance in Chapter 4, and will be a much stronger baseline than making all NPs left or right-branching. A left-branching baseline would perform poorly, as Table 3.3 showed that only 38% of NPs have left-branching structure. A right-branching baseline would be even worse as no brackets would be inserted, resulting in an F-score of 0.0%.

The annotation tool was run over the entire Penn Treebank in its original state. Suggestions were automatically followed where possible and no manual changes were made. All the suggestion rules (described in Section 3.2.1) were used, except for those from the annotator's previous bracketings, as these would not be available unless the annotation had already been completed. The results in Table 3.8 show that in all cases, the suggestion rules have high precision and low recall. This indicates that the rules are helpful, but only in certain cases. NER-based features, for example, are only helpful in NPs that dominate named entities, although whenever they can be applied, they are almost always correct.

The subtractive analysis shows that each of the suggestion types increases performance, with NER and company and name endings providing the biggest gains. Surprisingly, precision improves with the removal of the NER suggestion type. We suspect that this is caused by some of the annotation choices in the BBN corpus that do not align well with the parse structure. For example, *Mr* in *Mr Vinken* has no NER tag, rather than **PERSON**; while all three tokens in *a few years* are annotated as **DATE**.

Note that all of the results in Table 3.8, except for the last two lines, are evaluating over the entire corpus, as there was no need for training data. With this baseline, we have set a significant challenge for finding further improvement.

3.6 Summary

The results of this chapter have demonstrated that consistent annotation of NP structure is indeed possible. We have verified the statement of Quirk et al. (1985, page 1343) *"that obscurity in premodification exists only for the hearer or reader who is unfamiliar with the subject concerned"* and after annotating the entire corpus, we are quite familiar with its subject matter. The quality of our annotations has also been demonstrated by measuring inter-annotator agreement in multiple ways. The figures we calculated were very high, both when comparing against a second annotator and against the independently annotated DepBank. This chapter has also shown how the suggestion feature of the annotation tool supplies a strong baseline on which to improve.

The construction of this corpus was made possible by the principled approach we applied while building it. Having a well-defined set of guidelines based on an existing, successful project has allowed the creation of a large-scale, consistent corpus of NP annotations. We are now ready to begin experimenting with the extended Penn Treebank. Indeed, this corpus makes possible all of the experiments in the following chapters and therefore allows NP structure to be recovered by Penn Treebank-trained parsers for the first time.

Chapter 4

Parsing with Collins' Models

In the previous chapter, we described the augmentation of the Penn Treebank with NP structure. This extended corpus will now be used as the data set for parsing experiments. We use the Bikel (2004) implementation of the Collins (2003) model, as it is a widely-used and well-known parser with state-of-the-art performance. It is important to make the distinction between Collins' and Bikel's parsers, as they are not identical. The same is true for their underlying models, which again have slight differences. We use Bikel's *parser* in all of our experiments, but will still refer to Collins' *models* for the most part.

Collins' models generate NPs in a different way to all other constituents, which is particularly relevant for our experiments. Section 4.1.2 will describe these differences and why they were implemented. This will be important later in the chapter, where we make alterations to the model and analyse its performance.

4.1 Collins' Models

All of the Collins (2003) models use a Probabilistic Context Free Grammar (PCFG), with the Penn Treebank training data used to define the grammar and to estimate the probabilities of the grammar rules being used. The CKY algorithm is used to find the optimal tree for each sentence.

More formally, the conditional probability of each individual rule $LHS \rightarrow RHS$ is determined by taking counts from the corpus:

$$P(RHS|LHS) = \frac{\text{count}(LHS \rightarrow RHS)}{\text{count}(LHS)} \tag{4.1}$$

The product of the probability of individual rules can then be used to calculate the joint probability

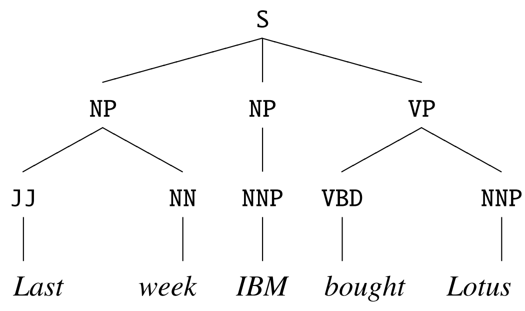


Figure 4.1: Example parse tree from Collins (2003)

of a tree T with a given sentence S:

$$P(T,S) = \prod_{i=1}^{n} P(RHS_i|LHS_i) \tag{4.2}$$

And the optimal tree T^* maximises this probability:

$$T^* = \operatorname*{argmax}_T P(T|S) = \operatorname*{argmax}_T \frac{P(T,S)}{P(S)} = \operatorname*{argmax}_T P(T,S) \tag{4.3}$$

4.1.1 Lexicalisation

The grammar is also lexicalised, i.e. each non-terminal is associated with a head token and its POS tag. The head is determined using the rules in Collins (1999, Appendix A). Lexicalisation can be very informative, allowing a better parsing decision to be made. However, in practice it also creates a sparse data problem. Collins (2003, page 594) gives the example in Figure 4.1, where the top-level rule S → NP NP VP occurs often enough that statistics from the corpus provide a reliable estimate of its frequency. This rule is put into Equation 4.1 to calculate the estimate:

$$P(\texttt{NP NP VP}|\texttt{S}) = \frac{\text{count}(\texttt{S} \rightarrow \texttt{NP NP VP})}{\text{count}(\texttt{S})} \tag{4.4}$$

However, when the rule is lexicalised with words and POS tags it becomes:

$$\texttt{S}(\textit{bought}, \texttt{VBD}) \rightarrow \texttt{NP}(\textit{week}, \texttt{NN})\ \texttt{NP}(\textit{IBM}, \texttt{NNP})\ \texttt{VP}(\textit{bought}, \texttt{VBD}) \tag{4.5}$$

and its probability is:

$$\begin{gathered} P(\texttt{NP}(\textit{week}, \texttt{NN})\ \texttt{NP}(\textit{IBM}, \texttt{NNP})\ \texttt{VP}(\textit{bought}, \texttt{VBD}) \mid \texttt{S}(\textit{bought}, \texttt{VBD})) = \\ \frac{\text{count}(\texttt{S}(\textit{bought}, \texttt{VBD}) \rightarrow \texttt{NP}(\textit{week}, \texttt{NN})\ \texttt{NP}(\textit{IBM}, \texttt{NNP})\ \texttt{VP}(\textit{bought}, \texttt{VBD})}{\text{count}(\texttt{S}(\textit{bought}, \texttt{VBD}))} \end{gathered} \tag{4.6}$$

The problem here is that even though *week*, *IBM* and *bought* are all reasonably common words and the POS tags are similarly frequent, the number of times that this lexicalised rule occurs is still very low. For most rules, the numerator in Equation 4.6 is likely to be zero, meaning that all rules will appear improbable. As a result, the statistical model that is built will not perform well.

In order to get more reasonable estimates, Collins (2003) splits the generation probabilities into smaller steps, instead of calculating the probability of the entire rule. Note that each rule can be framed as follows:

$$P(h) \rightarrow L_n(l_n) \ldots L_1(l_1) H(h) R_1(r_1) \ldots R_m(r_m) \tag{4.7}$$

where H is the head child, $L_n(l_n) \ldots L_1(l_1)$ are its left modifiers and $R_1(r_1) \ldots R_m(r_m)$ are its right modifiers. Making independence assumptions between the modifiers and then using the chain rule yields the following expressions:

$$P_h(H|Parent, h) \tag{4.8}$$

$$\textstyle\prod_{i=1 \ldots n+1} P_l(L_i(l_i)|Parent, H, h) \tag{4.9}$$

$$\textstyle\prod_{i=1 \ldots m+1} P_r(R_i(r_i)|Parent, H, h) \tag{4.10}$$

The head is generated first, then the left and right modifiers, which are conditioned on the head but not on any other modifiers. A special STOP symbol is also introduced (the $n+1^{th}$ and $m+1^{th}$ modifiers), which is generated when there are no more modifiers to the left or right.

Going back to the example in Figure 4.1, the probability of the rule would be calculated as the product of five individual terms, each of which will occur often enough in the corpus to give a reliable estimate.

$$\begin{aligned} P(\texttt{NP}(\textit{week}, \texttt{NN})\ \texttt{NP}(\textit{IBM}, \texttt{NNP})\ \texttt{VP}(\textit{bought}, \texttt{VBD}) \mid \texttt{S}(\textit{bought}, \texttt{VBD})) = \\ P_h(\texttt{VP} \mid \texttt{S}, \langle \textit{bought}, \texttt{VBD} \rangle) \times \\ P_l(\texttt{NP}(\textit{IBM}, \texttt{NNP}) \mid \texttt{S}, \texttt{VP}, \langle \textit{bought}, \texttt{VBD} \rangle) \times \\ P_l(\texttt{NP}(\textit{week}, \texttt{NN}) \mid \texttt{S}, \texttt{VP}, \langle \textit{bought}, \texttt{VBD} \rangle) \times \\ P_l(\texttt{STOP} \mid \texttt{S}, \texttt{VP}, \langle \textit{bought}, \texttt{VBD} \rangle) \times \\ P_r(\texttt{STOP} \mid \texttt{S}, \texttt{VP}, \langle \textit{bought}, \texttt{VBD} \rangle) \end{aligned} \tag{4.11}$$

The resulting probability will thus be more effective than that calculated using one very large rule. This is a key part of Collins' models, allowing lexical information to be included while still calculating useful probability estimates.

4.1.2 Generating NPs

For base-NPs, the parser generates modifiers using a slightly different model. Instead of conditioning on the head, the current modifier is dependent on the previous modifier, resulting in what is almost a bigram model. Formally, equations 4.9 and 4.10 above are changed as shown below:

$$\prod_{i=1...n+1} P_l(L_i(l_i)|Parent, L_{i-1}(l_{i-1})) \tag{4.12}$$

$$\prod_{i=1...m+1} P_r(R_i(r_i)|Parent, R_{i-1}(r_{i-1})) \tag{4.13}$$

There are a few reasons given by Collins for this. Most relevant for this work, is that because the Penn Treebank does not fully bracket NPs, the head is unreliable. When generating *crude* in the NP *crude oil prices*, we would want to condition on *oil*, the true head of the internal NP structure. However, *prices* is the head that would be found. Using the NP submodel thus results in the correct behaviour. As Bikel (2004) notes, the model is not conditioning on the previous modifier *instead* of the head, the model is treating the previous modifier *as* the head.

With the augmented Penn Treebank that we have created, the true head can now be identified. This may remove the need to condition on the previous modifier, and will be experimented with in Section 4.3.

The separate NP submodel also allows the parser to learn NP boundaries effectively, i.e. that it is rare for words to precede a determiner in an NP. Collins (2003, page 602) gives the example *Yesterday the dog barked*, where conditioning on the head of the NP, *dog*, results in incorrectly generating *Yesterday* as part of the NP. On the other hand, if the model is conditioning on the previous modifier, *the*, then the correct STOP category is much more likely to be generated, as words do not often come before *the* in an NP.

Collins also notes that a separate X-bar level is helpful for the parser's performance. For this reason, and to implement the separate base-NP submodel, a preprocessing step is taken wherein NP brackets that do not dominate any other non-possessive NP nodes are relabelled as NPB. For consistency, an extra NP bracket is inserted around NPB nodes not already dominated by an NP. These NPB nodes are removed before evaluation. An example of this transformation can be seen below:

```
(S
  (NP (DT The) (NN dog) )
  (VP (VBZ barks) ) )
```

```
(S
  (NP
    (NPB (DT The) (NN dog) ) )
  (VP (VBZ barks) ) )
```

4.1.3 Extensions to the Basic Model

Collins (2003) describes a number of additions to the model, which add back some of the information lost by the independence assumptions made in Section 4.1.1. The first is a distance measure, which aims to predispose the model to right-branching structure and favour modification by the most recent verb. Equations 4.9 and 4.10 are altered to include a distance function:

$$\prod_{i=1...n+1} P_l(L_i(l_i)|Parent, h, H, distance_l(i-1)) \tag{4.14}$$

$$\prod_{i=1...m+1} P_r(R_i(r_i)|Parent, h, H, distance_r(i-1)) \tag{4.15}$$

which looks at the surface string of the previous modifiers. The two pieces of information carried by $distance_l$ and $distance_r$ are whether or not the string is empty, and whether or not it contains a verb. These allow the model to learn the biases mentioned above. This is Collins' Model 1.

Collins' Model 2 makes use of subcategorisation frames, which help the model differentiate between adjuncts and complements. Collins identifies which is which using various heuristics based on the constituent labels and the Treebank II semantic tags: DIR, LOC, TMP, etc. This information is included in the model as an extra step between generating the head and the modifiers. Left and right subcategorisation frames, *LC* and *RC*, specify the complements required and are generated with the following probabilities:

$$P_{lc}(LC|Parent, h, H) \tag{4.16}$$

$$P_{rc}(RC|Parent, h, H) \tag{4.17}$$

These values are then included in the generation of the left and right modifiers:

$$\prod_{i=1...n+1} P_l(L_i(l_i)|Parent, h, H, distance_l(i-1), LC) \tag{4.18}$$

$$\prod_{i=1...m+1} P_r(R_i(r_i)|Parent, h, H, distance_r(i-1), RC) \tag{4.19}$$

This is the most-widely used of Collins' models and the one that we experiment with.

Collins' Model 3 adds support for traces and *Wh*-movement, using a mechanism similar to that used in Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1985). As with the subcategorisation frames, this model introduces a new parameter which is generated (conditioned on the head) and then used in the subsequent generation of the modifiers. This gap parameter serves to propagate the trace through the tree. When a gap has been probabilistically determined, it is added to the parameter. A trace will then be generated further down in the tree.

None of these extensions are particularly relevant for parsing NPs. The distance features for example are made entirely ineffectual by treating the previous modifier as the head. This means that they will always have the same values: the surface string is always empty and never contains a verb. Because of this, we have not described the additional terms in great detail. However, it is important to note the measures that must be taken in order to add new information sources to the model.

Coordination

Collins' models have difficulties with coordination, as the head-finding rules and probability estimates do not handle multiple heads. The independence assumptions that are necessary to allow lexicalisation also remove the information that is required to model coordinate structures well. As a result, constituents with unbalanced conjunctions, such as `NP CC` and `NP CC NP NP`, are given too much probability mass.

Collins (1999) introduces a solution to this problem, generating the conjunct and the following constituent together. For each constituent, a binary flag is generated. If the flag is true, then an additional step is taken which creates the conjunct node. An extra term is added to the product of rule probabilities, alongside those in Equation 4.11. This parameter P_α is conditioned on the words being coordinated, their constituent labels and the resulting constituent label. For example:

$$P_\alpha(\texttt{CC}, \mathit{and} | \mathit{Bill}, \mathit{Ted}, \texttt{NP}, \texttt{NP}, \texttt{NP}) \tag{4.20}$$

Punctuation

One deficiency of the original Collins (1997) model is that it did not generate punctuation at all, ignoring the information that it can provide. Collins (1999) takes some measures towards including punctuation, generating commas and colons, although not in the same way as other constituents are. Other punctuation marks are still ignored, as are any punctuation marks that begin or end a sentence. The remaining commas and colons are raised as high in the tree as possible, which means they always occur between constituents. Figure 4.2 shows an example of the process taken from Collins (2003, page 604). Once this transformation is performed, punctuation is generated in the same way as coordination. A flag is generated with every constituent, and when it is true, an additional term is included in the calculation of the rule probability. The punctuation flag is conditioned on the same variables as the coordination flag. An example is shown below:

$$P_p(\texttt{comma}, \mathit{comma} | \mathit{Vinken}, \mathit{old}, \texttt{NP}, \texttt{NPB}, \texttt{ADJP}) \tag{4.21}$$

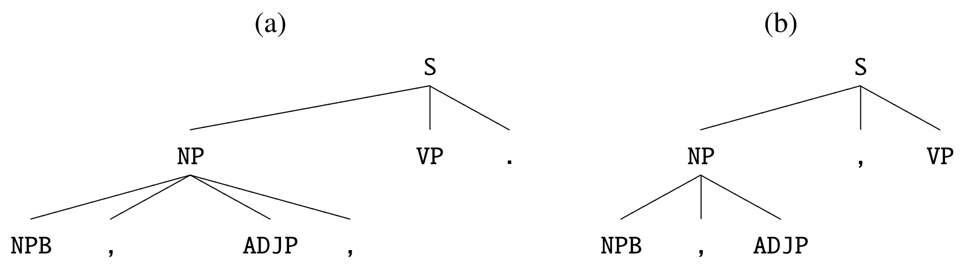


Figure 4.2: (a) Before punctuation preprocessing (b) After punctuation transformation

The standard evaluation used by Collins' and most other researchers since, actually ignores punctuation entirely. This may be because speech marks in particular were given the lowest priority in the Treebank and are thus inconsistent. The bracketing guidelines (Bies et al., 1995, page 54) say:

. . . [speech marks] just get yanked around by whatever is inside them.
They are at the very bottom of the pecking order.

Thus, the addition of punctuation to Collins' model is not so that they are recovered better, but to increase the model's ability to recover other structures using the information that punctuation provides.

Bikel's treatment of coordination and punctuation

The two previous sections have described the treatment of coordination and punctuation in Collins' models. Bikel (2004, §3.5.4) points out that this approach causes the model to be inconsistent, as the number of intervening conjunctive items is not taken into account. This means that there are an infinite number of sentence/tree pairs (with different amounts of coordination and punctuation) all of which must be assigned some probability mass. This causes the sum of the probability of all trees to diverge.

To solve this problem, Bikel implements a different solution that generates conjunctions and punctuation in the same manner as other constituents. A mapping function is used:

$$\delta(M_i) = \begin{cases} \texttt{+START+} & \text{if } i = 0 \\ \texttt{CC} & \text{if } M_i = \texttt{CC} \\ \texttt{+PUNC+} & \text{if } M_i = \texttt{,} \text{ or } M_i = \texttt{:} \\ \texttt{+OTHER+} & \text{otherwise} \end{cases} \tag{4.22}$$

where M_i is a left or right modifier. This function is then included in the conditioning context, rather than as a separate parameter class as in Collins' models. That is, Equations 4.9 and 4.10 are altered as shown shown below:

$$\prod_{i=1...n+1} P_m(M_i(m_i)|Parent, H, h, \delta(M_{i-1}), side) \tag{4.23}$$

where *side* indicates whether the modifier is to the left or right of the head. This approach properly estimates the joint probability for coordination and punctuation, without causing the model to be inconsistent.

4.2 Initial Experiments

To begin with, we compare the parser's performance on the original Penn Treebank and the new NML and JJP bracketed version. We report the standard Parseval measures (Black et al., 1991) labelled bracket precision, recall and F-scores over all sentences. F-score is the harmonic mean of precision and recall: $F = \frac{2PR}{P+R}$. Both the label and the placement of the bracket must be correct, although we follow Collins (2003) in ignoring punctuation.

As is the standard split, Sections 02–21 are used for training, Section 00 for development, and testing is carried out on Section 23. Table 4.1 shows the results on Section 00. The first row comes from training and evaluating on the original Penn Treebank, while the next three are all using the extended NP corpus. The first of these, **Original structure**, evaluates only the brackets that existed before the NP augmentation. That is, the NML and JJP brackets are removed before calculating these figures, in the same way that the NPB brackets added as part of Collins' parsing process are excised. The next figures, for **NML and JJP brackets only**, work in the opposite manner, with all brackets besides NML and JJP being ignored. The final row shows the results when all of the brackets — NMLs, JJPs and the original structure — are evaluated.

These figures supply a more detailed picture of how performance has changed, showing that although the new brackets make parsing marginally more difficult overall (by about 0.5% in

	PREC	RECALL	F-SCORE
Original PTB	88.88	88.85	88.86
Original structure	88.81	88.88	88.85
NML and JJP brackets only	76.32	60.42	67.44
All brackets	88.55	88.15	88.35

Table 4.1: Parsing performance, Section 00

F-score), accuracy on the original structure is only negligibly worse. The new NML and JJP brackets are the cause of the performance drop, with an F-score more than 20% lower than the overall figure.

The all-brackets result compares well to the original Penn Treebank model, as the latter is not recovering or being evaluated on NP structure and as such, has a much easier task. However the parser's performance on NML and JJP brackets is surprisingly poor. Indeed, the figure of 67.44% is over 5% lower than the baseline established using the annotation tool's suggestions (see Table 3.8). And even though the suggestions were in part based on NER information that the parser does not possess, this still demonstrates the difficulty it has in bracketing NPs. The rest of this chapter will describe a number of attempts to improve the parser's performance, by altering the data being used and the parser model itself.

4.2.1 Relabelling NML and JJP

The data that Collins' model was originally designed for did not include NML or JJP brackets and these new labels could cause problems. For example, head-finding for these constituents is undefined. Further, changing the structure of NPs (which are already treated differently in many aspects of Collins' model) also has deeper implications, as we shall experiment with in Section 4.4. In an attempt to remove any complications introduced by the new labels, we ran an experiment where the new NML and JJP labels were relabelled as NP and ADJP. These are the labels that would be given if NPs were originally bracketed with the rest of the Penn Treebank. This approach is also easy to implement, and negates the need for any change to the parser itself.

Relabelling the brackets means that the model does not have to discriminate between two different types of noun and adjective structures, and for this reason, we might expect to see an increase in performance. The figures in Table 4.2 show that this is not the case, as the all-brackets F-score has dropped by almost half a percent, compared to the numbers in Table 4.1. To evaluate the NML and JJP brackets only, we compare against the corpus without relabelling, and whenever a test NP matches a gold NML we count it as a correct bracketing. The same is done for ADJP and JJP

	PREC	RECALL	F-SCORE	DIFF
Original structure	87.92	88.68	88.30	−0.55
NML and JJP brackets only	–	53.54	–	−6.88
All brackets	88.09	87.77	87.93	−0.42

Table 4.2: Parsing performance with relabelled brackets

brackets. For example, if the parser's output is:

```
(NP
  (NP (NN crude) (NN oil) )
  (NNS prices) )
```

then the bolded bracket will be marked as correct against the following test data:

```
(NP
  (NML (NN crude) (NN oil) )
  (NNS prices) )
```

However, only recall can be measured in this way, and not precision, as the parser does not produce NML or JJP brackets that can be evaluated. These nodes can only be known when they have already been matched against the gold-standard, which falsely suggests a precision of 100%. The incorrect NML and JJP nodes are hidden by incorrect NP or ADJP nodes and the difference cannot be recovered. This also means that the figures given for the original structure are not entirely accurate, as the original NPs cannot be distinguished from the NMLs we annotated and have converted to NPs. This explains why precision drops by 0.89%, while recall is only 0.20% lower. Thus the NML and JJP brackets difference in Table 4.2 is for recall, not F-score.

Despite all these complications, the decreases in performance on every evaluation make it clear that the relabelling has not been successful. We carried out a visual inspection of the errors that were made in this experiment, which had not been made when the NP and NML labels were distinct. It was noticeable that many of these errors occurred when a company name or other entity needed to be bracketed, such as *W.R. Grace* in the example NP below:

```
(NP
  (ADVP (RB formerly) )
  (DT a) (NML (NNP W.R.) (NNP Grace) )
  (NN vice) (NN chairman) )
```

	PREC	RECALL	F-SCORE	DIFF
Original structure	88.78	88.86	88.82	−0.03
NML and JJP brackets only	75.27	58.33	65.73	−1.71
All brackets	88.51	88.07	88.29	−0.06

Table 4.3: Parsing performance with correct head-finding rules

We conclude that the model was not able to generalise a rule that multiple tokens with the proper noun POS tag should be bracketed. Even though NML brackets often follow this rule, NPs do not. As a result, the distinction between the labels should be retained, and we must change the parser itself to deal with the new labels properly.

4.2.2 Head-Finding Rules

The first and simplest change was to create head-finding rules for NML and JJP constituents. In the previous experiments, these nodes would be covered by the catch-all rule, which simply chooses the left-most child as the head. Label-specific rules are defined for all other constituent types, meaning that this catch-all rule would only be used for NML and JJP brackets. This rule is incorrect in most NMLs and JJPs, where the head is usually the right-most child.

To define the NML and JJP rules, we copy those for NPs and ADJPs respectively. Because of the limited types of structure in NMLs and JJPs, these effectively make the right-most child the head of the constituent. We also add to the rules for NPs, so that child NML and JJP nodes can be recursively examined, in the same way that NPs and ADJPs are. This change is not needed for other labels, as NMLs and JJPs only exist under NPs. We trained and ran the parser again with this change, and achieved the results in Table 4.3. The differences shown are against the original results from Table 4.1.

Once again, we were surprised to find that the F-score has been reduced, though by only a small amount overall, which chiefly comes from the NML and JJP brackets. This can be explained by considering an example NML: *lung cancer*. The corrected head-finding rule conditions the modifier *lung* on the head *cancer*. This NML constituent would then be quite likely, as the set of possible modifiers is restricted by the probability distribution. However, the reverse: conditioning the head *cancer* on the modifier *lung*, would also be informative, as the set of heads is likewise restricted. A NML's left-most token is rarely *the* or another uninformative token, and thus the uncorrected head-finding rules are also quite effective.

Furthermore, for NMLs such as *Judge Curry* or *Mr Vinken*, the left-most token is actually a much better generalisation to pass up the tree and base probabilistic actions upon. Finally, Bikel (2004, §6.1.1) and Chiang and Bikel (2002) note that head-finding rules do not affect Collins' models to a large degree. Using a much simpler set of rules degrades performance by only a small amount, while an optimal set of rules derived using Expectation Maximisation (EM) does not perform significantly better than the standard ones. For these reasons, choosing the left or right-most token as the head achieves similar performance.

4.3 The Base-NP Submodel

The next alteration to the parser is to turn off the base-NP submodel. Collins (1999, page 179) explains that this separate model is used because the Penn Treebank does not fully annotate internal NP structure, something that we have now done. Hopefully, with these new brackets in place, we can remove the NP submodel and perhaps even improve performance in doing so.

We experimented with three different approaches to turning off the base-NP model. All three techniques involved editing the Bikel (2004) parser code:

1. Changing the isBaseNP() method to always return false. This means that the main model, rather than the NP submodel, is always used.

2. Removing the preprocessing step that creates NPB nodes (described in Section 4.1.2). This alteration will have the same effect as the one above, and will also remove the distinction between NP and NPB nodes.

3. Changing the isNP() method to return true for NMLs. This will affect which NPs are turned into NPBs during the preprocessing step, as NPs that dominate NMLs will no longer be basal.

The third change does not turn off the base-NP model, but it does modify where it functions.

The results are in Table 4.4, and in all cases the overall F-score has decreased. With the first change, to isBaseNP(), performance on only NML and JJP brackets has actually increased by 3.78% F-score, although the original structure is almost 10% worse. The second change, to the preprocessing step, results in a much smaller loss to the original structure, but also not as big an increase on the internal NP brackets. The third change, to isNP(), is most notable for the large drop in performance on the internal NP structure.

		PREC	RECALL	F-SCORE	DIFF
1	Original structure	72.09	88.19	79.33	−9.52
	NML and JJP brackets only	72.93	69.58	71.22	+3.78
	All brackets	72.11	87.71	79.14	−9.21
2	Original structure	87.75	87.65	87.70	−1.05
	NML and JJP brackets only	72.36	69.27	70.78	+3.34
	All brackets	87.37	87.17	87.27	−1.08
3	Original structure	86.90	88.66	87.77	−1.08
	NML and JJP brackets only	48.61	3.65	6.78	−60.66
	All brackets	86.83	86.46	86.64	−1.71

Table 4.4: Performance with the base-NP model off

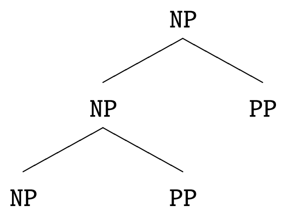


Figure 4.3: An unlikely structure

There are a few reasons for these rather terrible results, which demonstrate the necessity of the base-NP submodel. Collins (1999, §8.2.2) explains why the distinction between NP and NPB nodes is needed: otherwise, structures such as that in Figure 4.3, which *never* occur in the Treebank, are given too high a probability. The parser needs to know where NPs will not recurse anymore (when they are basal), so that it can generate the correct flat structure. Furthermore, the third change effectively treats NP and NML nodes as equivalent, and we have already seen problems caused by this approach in Section 4.2.1.

4.4 Bracket Structure

We have now seen how a Collins-style parser performs on internal NP structure, but a question remains about whether the structure itself is optimal. Treebank structure can have a large effect on parser performance, as has been studied by many researchers. Collins (2003, page 621) notes that binary trees would be a poor choice, as the parser loses some context sensitivity, and the

	PREC	RECALL	F-SCORE	DIFF
Original structure	87.96	88.06	88.01	−0.84
NML and JJP brackets only	82.33	74.28	78.10	+10.66
All brackets	87.33	86.36	86.84	−1.51

Table 4.5: Explicit right-branching structure

distance measures become ineffective. He advocates one level of bracketing structure per X-bar level.

Goodman (1997) on the other hand, explicitly converts trees to a binary branching format as a preprocessing step, in order to avoid problems from varying structures. Johnson (1998) finds that the performance of simple PCFGs can be improved through tree transformations, while Klein and Manning (2001) observe that some simple tree transformations can increase parser speed. The variation shown in these approaches, all for the same task, highlights the difficulty in identifying optimal tree structure.

The issue of treebank structure extends to other languages as well, and implies further difficulties when comparing between languages. Kübler (2005) investigates two German treebanks with different annotation schemes, and finds that certain properties, such as having unary nodes and flatter clauses, increase performance. Rehbein and van Genabith (2007) suggest that evaluation methods are also affected by treebank structure, showing that the Parseval measures are biased towards an increased number of non-terminal nodes. We will experiment with an alternate annotation scheme in this section, and see how it affects NP structure performance.

4.4.1 Explicit Right-Branching Structure

It may be argued that explicitly bracketing right-branching structure would increase performance. For example, in the NP *the New York Stock Exchange*, if there were a bracket around *New York Stock Exchange*, then it would be useful training for when the parser comes across *New York Stock Exchange composite trading* (which it does quite often). The parser should learn to add a bracket in both cases. The current bracketing guidelines do not mark such right-branching constituents, they are simply assumed implicitly to be there. We can automatically add them and then examine what difference this change makes.

We find, in Table 4.5, that overall performance drops by 1.51% F-score. This is a surprising result, as there are a number of easily recoverable brackets that are introduced by making

	PREC	RECALL	F-SCORE
Original PTB	88.58	88.45	88.52
Suggestion baseline	94.29	56.81	70.90
Original structure	88.49	88.53	88.51
NML and JJP brackets only	80.06	63.70	70.95
All brackets	88.30	87.80	88.05

Table 4.6: Parsing performance, Section 23

right-branching structure explicit. For example, a POS tag sequence of DT NN NN is always right-branching. This explains the more than 10% increase in F-score when evaluating internal NP brackets only. As Rehbein and van Genabith (2007) found, increasing the number of non-terminal nodes has caused an increase in performance, though we may question, as they do, whether performance has truly increased, or whether the figure is inflated by the evaluation method. Either way, the deleterious effect on overall performance suggests that right-branching structure should be left implicit.

4.5 Test Set Results

Having found that the best performing model is the initial one with no alterations, we now evaluate its results on the test data: Section 23. Table 4.6 shows that, as with the Section 00 results, the original Penn Treebank structure is barely affected by the additional NML and JJP brackets. The new brackets themselves are recovered slightly better than they were on the development data, achieving a figure that is almost the same as the suggestion baseline in this case.

These results confirm those we saw in our initial experiments: recovering NP structure is a difficult task for the Collins (2003) parser. As a result, there is a slight drop in overall performance.

4.6 Error Analysis

Despite the large number of experiments we have performed in this chapter, we are no closer to outperforming the suggestion baseline established in the previous chapter. The highest accuracy has come from the unaltered parser, while changes to the corpus and model have proven unsuccessful. We need to look at the errors being made by the parser, so that any problems that appear can be solved. Accordingly, we categorised each of the 560 NML and JJP errors in our initial model through manual inspection. The results of this analysis are shown in Table 4.7, together with

ERROR	#	%	FP	FN	EXAMPLE	
Modifier attachment	213	38.04	56	157		
NML	122	21.79	21	101	*lung cancer deaths*	×
Internal Entity Structure	43	7.68	24	19	*(Circulation Credit) Plan*	×
Appositive Title	29	5.18	6	23	*(Republican Rep.) Jim Courter*	√
JJP	10	1.79	4	6	*(More common) chrysotile fibers*	√
Company/name postmodifiers	9	1.61	1	8	*(Kawasaki Heavy Industries) Ltd.*	√
Mislabelling	92	16.43	30	62	*(ADJP more influential) role*	√
Conjunctions	92	16.43	38	54	*(cotton and acetate) fibers*	√
Company names	10	1.79	0	10	*(F.H. Faulding) & (Co.)*	√
Possessives	61	10.89	0	61	*(South Korea) 's*	√
Speech marks and brackets	35	6.25	0	35	*(" closed-end ")*	√
Clearly wrong bracketing	45	8.04	45	0		
Right-branching	27	4.82	27	0	*(NP (NML Kelli Green))*	×
Unary	13	2.32	13	0	*a (NML cash) transaction*	×
Conjunction	5	0.89	5	0	*(NP a (NML savings and loan))*	×
Structural	8	1.43	3	5	*(NP … construction spending) (VP (VBZ figures) …*	×
Other	14	2.50	8	6		
Total	560	100.00	180	380		

Table 4.7: Error analysis for the Collins (2003) parser

examples of the errors being made. Only relevant brackets and labels are shown in the examples, while the final column describes whether or not the bracketing shown is correct.

The largest group in the table (38.04%) is made up of modifier attachment errors that do not fit into any more specific category. In the example, because there is no bracket around *lung cancer*, there is a dependency between *lung* and *deaths*, instead of *lung* and *cancer*. We can further divide these errors into general `NML` and `JJP` cases, and instances where the error occurs inside a company name or in a person's title.

As this is the largest cause of errors, we performed some analysis to determine their cause. We searched the training data for each of the 142 unique false negatives (i.e. *lung cancer* in the example from the table) and found that 93 of them do not occur at all. A further 17 of the *n*-grams do occur, but not as constituents, which would make reaching the correct decision even more difficult for the parser. In order to fix this problem, it appears that an outside source of information must be consulted, as the lexical information is currently not available.

The next largest source of errors (16.43%) is mislabelling the bracket itself. In particular, distinguishing between using `NP` and `NML` labels, as well as `ADJP` and `JJP`, accounts for 75 of the 92 errors. This is not surprising, as we noted during the final preparation of the corpus (see Section 3.3.2) that the labels of some NPs were inconsistent. The previous relabelling experiment suggests that we should not evaluate the pairs of labels equally, meaning that the best way to fix these errors would be to change the training data itself. This would require alterations to the original Penn Treebank brackets, something we deliberately avoided during the annotation process.

Conjunctions are another significant source of errors (16.43% again), and are quite a difficult problem. This is because coordinating multi-token constituents requires brackets around each of the constituents, as well as a further bracket around the entire conjunction. Getting just a single decision wrong can mean that a number of these brackets are in error.

Another notable category of errors arises from possessive NPs, which always have a bracket placed around the possessor (*South Korea* in the example in Table 4.7) in our annotation scheme. The parser is not very good at replicating this pattern, perhaps because these constituents would usually not be bracketed if it were not for the possessive. In particular, `NML` nodes that begin with a determiner are quite rare, only occurring when a possessive follows. For example:

```
(NP
  (NP (NML (DT a) (NN dog) )
    (POS 's) )
  (NN life) )
```

The parser also has difficulty in replicating the constituents around speech marks and brackets. This may be because, as we described in Section 4.1.3, annotation of these constituents is far from consistent.

There are a number of NML and JJP brackets in the parser's output that are clearly incorrect, either because they define right-branching structure (which we leave implicit) or because they dominate only a single token. Single token NMLs only exist in conjunctions, but unfortunately the parser is too liberal with this rule. The final group of errors is caused by structural problems; that is, the entire parse for the sentence is malformed. In the Table 4.7 example, *figures* is actually a noun.

4.7 Summary

In this chapter, we have had the first indication of the tremendous difficulty involved in parsing NPs. Despite all the experiments that have been performed, the Collins (2003) parser was unable to outperform the rule-based suggestion baseline from Chapter 3. However, we have gained some important insights into how best to build data and models for NP parsing:

- NML labels should remain distinct from NPs[1];
- head-finding rules for these new brackets do not significantly affect performance;
- the base-NP submodel remains an integral part of the parsing model;
- right-branching structure is best left implicit;
- the primary difficulty in bracketing NPs is a lack of lexical information.

This last point is perhaps the most important. The error analysis in Section 4.6 demonstrated that the modifier attachment problem is the best avenue to pursue. Not only is it the largest cause of errors, but it suggests a potential approach to reducing the lexicalisation problem: find and make use of more data.

Incorporating this additional data into a parsing model is still a challenging task. One way to include this information into a Collins-style parser would be to add a new probability distribution to the model (akin to the subcategorisation frames and distance measures). However, we will take a different approach in Chapter 5: applying NP Bracketing techniques. In this way, we can make

[1]It may also be true that JJP and ADJP brackets should remain distinct, although we have not shown this explicitly. Due to their relative infrequency, it is a minor point either way.

use not only of *n*-gram counts, but also a wide range of other features drawn from many different sources.

Chapter 5

Noun Phrase Bracketing

Using a general parser is only one way to recover NP structure. There are a number of approaches that have been used for this task. Chunking (Ramshaw and Marcus, 1995) involves identifying noun phrases (and other types of phrases) and is a well-explored task. However internal NP structure is not analysed as part of the task. Recursive NP chunking — as in the CoNLL 1999 shared task and as performed by Daumé III and Marcu (2004) — is closer, but still does not recover full sub-NP structure. This is in part because gold-standard annotations have not been available in the past. The CoNLL 1999 data for example, was automatically derived from the Penn Treebank, which as we have seen, does not fully annotate NP structure.

Multi-word expressions (MWEs) can be used to identify some NP structure. For example, knowing that *stock market* is a MWE may help bracket *stock market prices* correctly, and named entities (NEs) can be used the same way. However, this only resolves some of the structure and only in those NPs that dominate MWEs or NEs. NP Bracketing is applicable to all NPs and covers the full depth of syntactic structure.

Our goal is to construct the first large-scale wide-coverage NP Bracketing system. We will begin with the data set used by most researchers in the past, in order to have a meaningful comparison between our work and theirs. This model will then be used as a basis for experiments later in the chapter. The supervised models that we will implement are the first to achieve high performance, and are capable of bracketing realistic NPs from the Penn Treebank.

One immediate application of our NP Bracketing system will be to build a post-processor for the Collins (2003) parsing experiments in the previous chapter. The NPs that the parser produces will be extracted, reparsed by the NP Bracketing system, and then inserted back into the parser output. This will allow us to include a wide range of information on which to base NP parsing decisions, without having to make significant structural changes to the parser model itself. Performance on `NML` and `JJP` brackets should improve as a result.

One note before we begin, is that throughout this thesis we will refer to the task as Noun Phrase Bracketing. Other researchers have called the task Noun Compound Bracketing or Noun-Noun Bracketing, however we are intending to build general purpose models that do not restrict the possible inputs in some arbitrary way. Thus, we take a simple and pragmatic view of how the NPs being bracketed are defined: an NP is what the Penn Treebank tells us an NP is. And seeing that we are bracketing NPs with no further delineation, it seems sensible to call the task NP Bracketing.

5.1 Task Description

The traditional NP Bracketing task, as framed by Marcus (1980, page 253) and Lauer (1995b), is: given a three word noun phrase like those below, decide whether it is left branching (5.1) or right branching (5.2).

```
((crude oil) prices)          (5.1)
(world (oil prices))          (5.2)
```

Most NP Bracketing research has restricted itself to cases like this, which we call *simple* NPs, as they do not reflect the full variety of NP structures. Non-noun parts of speech are excluded from this definition, meaning that the wider vocabulary of adjectives, pronouns and prepositions is omitted. Furthermore, the NP must consist of exactly three words, ignoring the difficulties that longer phrases create. We do allow proper nouns into our data set of simple NPs, although NPs where all three tokens are proper nouns are removed (see Section 5.3.1). Simple NPs will be experimented with in Section 5.4.

As a more realistic view of the NP Bracketing task, we also define *complex* NPs as those that may be longer than three words and may include non-noun parts of speech. This definition matches our aim of dealing with all NPs in the Penn Treebank. Bracketing NPs of arbitrary length necessitates using an algorithm that manages the additional levels of structure, rather than simply having a classifier choose from two possibilities. This makes complex NP Bracketing a significantly harder task than only analysing simple NPs. We suggest that complex NPs present a more interesting, difficult and relevant task for the NLP community. Section 5.5 details our experiments on complex NPs.

5.2 Previous Work

Marcus (1980, Appendix A) presents an algorithm to choose between left and right-branching bracketings, which has been the basis of almost every approach since. The idea of the algorithm is to compare the relationship between words 1-2 to that between words 2-3. For example, comparing *crude oil* to *oil prices*. The comparison being made is of *semantic acceptability*, which Marcus describes as *"the relative 'goodness' of [the] two possible noun-noun modifier pairs"*. That is, how likely it is that the bigram forms a coherent semantic unit. Marcus does not suggest a way to measure such a value, but instead makes his own semantic judgements, leaving a general solution as an exercise for the reader.

The solution that a number of researchers have proposed is to calculate the semantic acceptability of a bigram based on how often it occurs in a corpus. Pustejovsky, Berger, and Anick (1993, page 341) briefly mention this idea, while Liberman and Sproat (1992) use more than the raw counts, applying the pointwise mutual information measure to make the bracketing decision.

Resnik (1993, §5.5) applies the *selectional association* measure derived in his thesis, which involves mapping words onto classes from a thesaurus. This provides a more generalised measure of bigram acceptability than simply calculating on the raw bigram count. However, the use of semantic classes can also introduce problems, because distinguishing between different senses of a word is a difficult task. Resnik chooses the class that maximises the association measure, which can give an overly optimistic score for the relationship between the words.

Resnik also provides an evaluation of his system, rather than simply looking at a handful of examples as had been done previously. The test set used is 200 three noun compounds from the Wall Street Journal section of the Penn Treebank, 64.1% of which were left-branching. The evaluation exposes a problem of coverage, where 18.5% of the NPs contain a word that is not in the thesaurus. Resnik's algorithm also uses a parameter that can be used to bias the bracketing decision towards left or right-branching. However, the parameter is not optimised on a development set. The best result achieved is 72.6%, outperforming a left-branching baseline by about 8%.

5.2.1 Adjacency and Dependency

Lauer (1995a; 1995b) presents the seminal work on Noun Phrase Bracketing. All of the methods described previously are labelled as *adjacency* models, which compare the association between words 1-2 to words 2-3. Lauer suggests a new variation, the *dependency* model where words 1-2 are compared to words 1-3. Both models are illustrated in Figure 5.1. The dependency

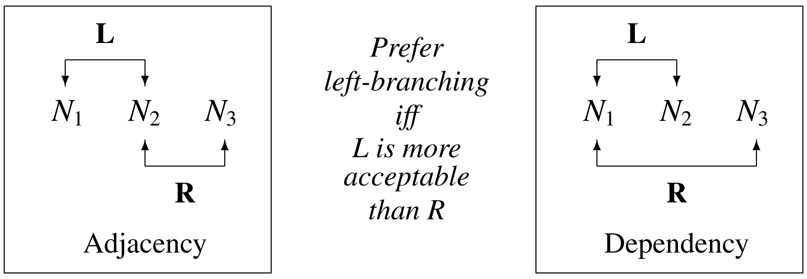


Figure 5.1: The associations in the adjacency and dependency models, from Lauer (1995)

model has the advantage that it better matches the parse tree formed by the NP. That is, the goal is to determine whether word 1 attaches to word 2 or word 3, and the semantic acceptability between words 2-3 (as measured in the adjacency model) does not factor into this.

Lauer's data is drawn from Grolier's encyclopedia, by searching for a sequence of words that are unambiguously nouns surrounded by non-noun tokens. This results in a training set of 35,343 noun pairs and a test set of 625 triples. Lauer also uses Resnik's idea of smoothing words into semantic classes, choosing Roget's thesaurus where Resnik used WordNet (Fellbaum, 1998). This means that the data set can only contain words that are present in the thesaurus, reducing the test set to 308 noun compounds. Manual inspection also showed that 29 of the compounds were errors. These came from sentences like *In monsoon regions rainfall does not...*, where the three noun sequence *monsoon regions rainfall* is not actually a single syntactic structure. There were also 35 indeterminate noun compounds, which as we described in Section 3.1, do not fit into left or right-branching categories. Some examples from Lauer's data set are shown in Table 5.1.

Lauer does not supply any context, making the error cases in particular difficult to understand. Requiring that all tokens exist in the thesaurus can also be viewed as limiting the domain for the task, or that the coverage of Lauer's algorithm is only 49% ($\frac{308}{625}$). The final set of 244 (216 unique) noun compounds is undoubtedly small, but despite all this, Lauer's data set has been used by most researchers since.

Lauer finds that the dependency model performs better than the adjacency model in all his experiments. The best performance is achieved using the Brill (1993) POS tagger to extract training data, rather than the unambiguous nouns used previously. With this addition, the dependency model achieves 80.7% accuracy compared to the adjacency model's 72%.[1]

Lauer also experiments with not using the semantic classes, finding that performance

[1]This is an estimate from the graph that Lauer provides, as no exact figure is given.

TYPE	EXAMPLE
Left	*minority business development* *disaster relief assistance* *customs enforcement vehicles*
Right	*satellite data systems* *county extension agents* *world food production*
Error	*principle organ systems* *quality movie animation*
Indeterminate	*engine lubrication system* *speech communication skills*

Table 5.1: Examples from Lauer's data set

drops by about 4% with the dependency model, and by about 0.5% with the adjacency model.[2] Also notable in this experiment is that the guess rates, i.e. how often the training data does not provide any information, increase dramatically. From under 5% when using semantic classes, they rise to over 15% in the adjacency model and over 25% in the dependency model.

Buckeridge and Sutcliffe (2002) present four different data sets from four different domains: a software manual, library science abstracts, aeronautics abstracts and magazine articles. Each of these corpora contains 200-300 NPs. Buckeridge and Sutcliffe use Latent Semantic Indexing (LSI) to measure semantic acceptability, the idea being that LSI will capture the associations across documents better than the word-word co-occurrences that previous researchers have used.

Their results vary greatly, ranging from 62% to 84% over the four corpora. We suspect that this large variation in performance is partially a result of the size of the data. We have found that unsupervised techniques are particularly sensitive to the counts being used, and this is magnified when using such small data sets.

Another point of interest is that the adjacency model outperforms the dependency model on two of the corpora and on the other two they are equal. This is contrary to Lauer's (1995b) results, and suggests that the superiority of either model is corpus dependent.

More recently, the web has been used as a source of bigram counts and to thus measure semantic acceptability. Lapata and Keller (2004) derive estimates from the Altavista search engine, comparing the raw counts returned at a lexical level. They achieve 78.7% accuracy on Lauer's data set. Although this is less than Lauer's best result, considering the small size of the corpora the

[2]Again, these are estimates from the graph.

difference may not be significant. Another explanation for Lapata and Keller's lower performance is that they did not use the semantic classes that Lauer found so effective.

Nakov and Hearst (2005a) also use web counts, but incorporate additional counts from several variations on simple bigram queries. These include concatenating the bigram or joining it with a hyphen, as well as a number of more complicated permutations that include morphological analyses and appending some amount of context. For example, the NP *brain stem cells* generates queries for *stemcells*, *stem-cells*, *stem cell*, *the brain associated with stem cells* and many more. There can be hundreds or even thousands of queries for a single three word noun phrase.

In order to combine the multitude of counts that the different queries return, Nakov and Hearst (2005a) use a rather ad-hoc voting method. Each query will have a left and right-branching alternative, for example *brainstem* vs *stemcells*, and whichever count is larger will determine the vote. However additional scaling constants are required for some queries. These hand-tuned weights are necessary when one count will naturally be much larger than another, for example, an NP with its first two words reversed compared to the original NP.

The vote of each query variation is also weighted, again using parameters that are hand-tuned on the test set itself. This allows the most effective variations to contribute an appropriately large amount to the final decision. Nakov and Hearst's (2005a) system is successful, resulting in an impressive 89.3% accuracy. However, reaching this figure requires a multitude of queries, each of which is parameterised manually, and the vast majority of which result in a zero count. Expanding the range of bigram counts being used is an excellent idea, although it creates a new problem of how best to combine these competing factors. This motivates our use of supervised methods, beginning in Section 5.4.4.

Nakov and Hearst (2005a) also present their own corpus, made up of 430 simple NPs from the biomedical domain. They achieve 95.4% accuracy on this data set, although the left-branching baseline on this set is also much high than on Lauer's, at 84.0%.

5.2.2 Supervised Approaches

All of the work described in the previous section employs unsupervised techniques, as bigram counts can easily be drawn from unannotated text. Lauer (1995b, page 15) specifically says *"While some medium-sized parsed corpora are available, the effort involved in constructing them is enormous and relies on semi-automatic methods to be feasible (see for example, Marcus et al, 1993). If [Statistical Language Learning] is to tap into the enormous volumes of on-line text,* *unsupervised techniques must be developed.”* Despite this, we have found that there are a number of informative features — the simplest example being the lexical items in the NP, rather than their counts — that a supervised model could take advantage of. The multitude of queries performed by Nakov and Hearst (2005a) could be combined in a robust manner with such a model, rather than the idiosyncratic voting and weighting scheme they use.

Girju et al. (2005) build a supervised model, despite a shortage of gold-standard data. They train a decision tree classifier using 362 manually annotated NPs from the Wall Street Journal (WSJ) as training data, and test on Lauer’s data. For each of the three words in the NP, they extract five features from WordNet. These are:

1. The related verb, which is the derivationally related form of the noun;
2. the top semantic class of the noun;
3. the second top semantic class of the noun;
4. the third top semantic class of the noun;
5. whether or not the noun is a nominalisation according to NomLex (Macleod et al., 1998).

This approach achieves 73.1% accuracy, although when they shuffled their WSJ data with Lauer’s to create a new test and training split, performance increased to 83.1%. This may be a result of the ~10% duplication in Lauer’s data set.

5.2.3 Complex NPs

The bracketing of complex NPs has been largely neglected in the literature, as all of the approaches that we have described so far are only effective for three word NPs. Marcus (1980, Appendix A) posits that an NP of arbitrary length can be bracketed by *“iteratively examining only the three left-most nouns in the modifier string”*. However we find that this does not work for right-branching NPs. Consider an NP: *(A (B (C D)))*. The three left-most nouns cannot be bracketed in any way until *D* is included. In the same way, iteratively moving from right to left does not work for left-branching NPs. Marcus does give some examples of NPs with which his arbitrary length algorithm has problems, and comes to the conclusion that they *“all seem to have an initial noun that describes a time, a place, or the stuff out of which something is made”*. We find that the common factor is that they are all right-branching NPs.

Barker (1998) describes an algorithm for bracketing complex NPs, which we will describe in full in Section 5.5.1. Like Marcus’ idea, the algorithm uses a sliding three word window, reducing the complex NP problem to making multiple simple NP decisions. However, unlike Marcus' algorithm, the window can move back and forth and only inserts brackets when the correct structure has been fully determined. In the example above, the algorithm would observe that if $A\ B\ C$ is right-branching, then a bracket cannot be added yet, but that the window should be shifted to the right.

Barker's system is actually an interactive bracketing tool, as although it makes a decision automatically, the result is then shown to a user for confirmation. The system bases its bracketing decisions on NPs that it has seen earlier in the process. For example, it may first come across the NP: *laser printer* and learn that this is an acceptable constituent. When it is subsequently presented with *laser printer stand*, it will suggest that this is a left-branching NP, which the user will then accept. Not surprisingly, the system performs noticeably better as it progresses through the data, as it has learnt from a greater number of NPs in these later stages.

Barker (1998) experiments with two data sets drawn from a computer installation manual and a book on the mechanics of small engines. On the former, there are 188 individual bracketing decisions, on which the system achieves 65% accuracy. On the latter data set there are 164 decisions to be made, of which the system gets 62% correct.

5.3 Data

In order to build an NP Bracketing system, there must be data to evaluate it, and for supervised models, to train on as well. We extract both a simple and a complex NP data set from the extended Penn Treebank that we annotated in Chapter 3.

5.3.1 Simple NPs

Simple NPs are extracted from our extended Penn Treebank data using the following method. If the last three children of an `NP` or `NML` are nouns, then they became an example in our data set. We mark the example as left branching if the first and second words are bracketed, and as right branching otherwise. Note that because only the right-most part of the NP is being looked at, we know that incomplete NPs are not being extracted. This is assuming that the head of the NP is right-most, following the Collins (1999) head-finding rules. For example, in the NP *the average seven-day compound yield*, which is fully right-branching, we can extract *seven-day compound yield*. However, we do not extract *average seven-day compound* because it is not right-most. This

CORPUS	# ITEMS	LEFT (%)	RIGHT (%)
Penn Treebank	5569	59	41
Lauer (1995b)	244	67	33
Buckeridge and Sutcliffe (2002) AmiPro	307	58	42
Buckeridge and Sutcliffe (2002) CISI	235	63	37
Buckeridge and Sutcliffe (2002) CRAN	223	74	26
Buckeridge and Sutcliffe (2002) Time	214	48	52
Nakov and Hearst (2005a) Biomedical	430	84	16
Barker (1998) SPARC	188	45	55
Barker (1998) small engines	164	91	9

Table 5.2: A comparison of NP bracketing corpora

sequence is made up of modifiers without a head and as such is nonsensical on its own. Note that we are making the assumption here that the head of the phrase will be final. This is true almost all the time (it is one of Collins' head-finding rules) and so does not cause any problems. We also remove examples where each word has the same NER tag, ignoring many flat base-NP cases such as *John A. Smith*, as these will not fit into a left or right-branching category.

This process extracts 5569 three word NPs, which is an order of magnitude larger than all previous data sets. Previous researchers have typically used Lauer's set (244 NPs) or created their own small data set (~500 NPs at most). This new, much larger corpus means that we can carry out large-scale machine learning effectively, rather than using unsupervised methods.

Statistics comparing our new data set to those used by other researchers are shown in Table 5.2. As can be seen, the Penn Treebank-based corpus is significantly larger than all other data sets. The distribution of left and right-branching NPs also appears to vary greatly, which may be affected by the content of the corpus. The Nakov and Hearst (2005a) biomedical and Barker (1998) small engines data sets are both very technical texts, while the Buckeridge and Sutcliffe (2002) AmiPro software manual and Buckeridge and Sutcliffe (2002) Time magazine articles are probably aimed at a more general audience.

We also measured the amount of lexical overlap between our Penn Treebank corpus and Lauer's data set, shown in Table 5.3. This displays the percentage of *n*-grams in Lauer's corpus that are also in our corpus. We can clearly see that the two corpora are quite dissimilar, as even on unigrams barely half are shared.

n-GRAM	MATCH
Unigrams	51.20%
Adjacency bigrams	6.09%
Dependency bigrams	3.61%
All bigrams	5.50%
Trigrams	1.40%

Table 5.3: Lexical overlap between Lauer's data set and our Penn Treebank corpus

#	H	SEQUENCE	EXAMPLE
2228	0.00	`( NNP , NN )`	[ABC] Co , [the market] leader
1796	1.00	`( ( NNP NNP ) NNP )`	John Smith Co.
1762	1.00	`( NNP NNP NNP )`	John A. Smith
1481	0.54	`( JJ NN NNS )`	high interest rates
1359	0.59	`( DT JJ NN NN )`	the high interest rate
1054	0.13	`( JJ JJ NNS )`	big red cars

Table 5.4: Complex NP POS tag sequences

5.3.2 Complex NPs

To create a set of complex NPs, we retrieve each NP of length three or more in the Penn Treebank. This includes non-base-NPs, although only the heads of the non-leaf constituents are represented so that we can treat them in the same manner as base-NPs. Some common POS tag sequences (e.g. initial determiner and final possessive) are unambiguous in three word NPs, and so we remove these cases. This has the side effect of increasing the ambiguity in the data and making the task harder. 53,568 instances are left in our data set, which is two orders of magnitude larger than any that has been created previously.

Table 5.4 shows the most common POS tag sequences in our complex NP data set. The entropy, H, describes the distribution of bracketings for each POS tag sequence. These values thus give an indication of the difficulty of the task. Larger entropy means that the sequence is more ambiguous, because there are many bracketing alternatives to choose from and/or because the alternatives are close to equally likely. The entropy figure for the `NNP NNP NNP` bracketings reinforce the result we saw in Section 4.6: a sequence of three nouns is very hard to bracket, as there is no good baseline decision.

Figure 5.2 shows a histogram of the entropy distribution across POS tag sequences. While

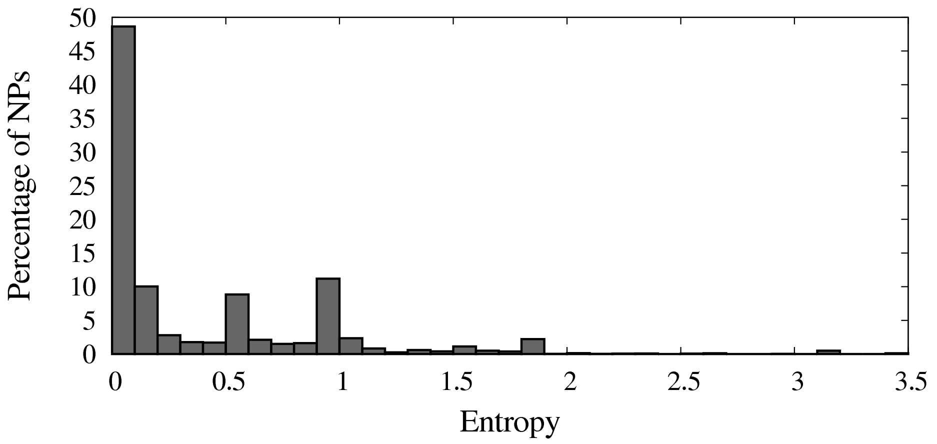


Figure 5.2: Entropy of NP POS tag sequences

43.71% of all sequences have a single bracketing, the majority of sequences are ambiguous. There is a spike just below 1, mostly made up of sequences with two almost equally likely bracketings. This demonstrates that complex NP bracketing is far from a trivial task.

5.4 Experiments with Simple NPs

With our new data set of simple NPs, we can now run experiments similar to those in the literature. We will implement both an adjacency and a dependency model and compare the results on Lauer's data set and on our Penn Treebank corpus.

5.4.1 Association Measures and *n*-gram Counts

Before we begin experimenting, the association measures and source of *n*-gram counts must be determined. As we described in Section 5.2, there have been a number of different approaches taken by researchers in the past. We will follow Nakov and Hearst (2005a) and use three different association measures: the raw bigram count, the bigram probability, and χ^2. All of these are defined below:

$$\text{Raw bigram count} = count(w_i, w_j) \tag{5.3}$$

$$P(w_i|w_j) = \frac{count(w_i, w_j)}{count(w_j)} \tag{5.4}$$

$$\begin{aligned}
\chi^2(w_i, w_j) &= \frac{N(AD - BC)^2}{(A + C)(B + D)(A + B)(C + D)} && (5.5)\\
\text{where } A &= count(w_i, w_j) && (5.6)\\
B &= count(w_i, \bar{w}_j) && (5.7)\\
C &= count(\bar{w}_i, w_j) && (5.8)\\
D &= count(\bar{w}_i, \bar{w}_j) && (5.9)\\
\text{and } N &= A + B + C + D && (5.10)
\end{aligned}$$

$\bar{w}$ indicates any word *except* w

Nakov and Hearst drew bigram counts from the number of hits returned by the Google and MSN search engines and we will do the same. We performed the queries during March 2007, with the search engine language filter set to English.

In addition to these search engines, we also use the Google Web 1T corpus (Brants and Franz, 2006) as a third source of counts. This corpus supplies counts for *n*-grams of length 1–5, drawn from approximately 1 trillion words of web text. The web text was tokenised in a similar manner to the Penn Treebank, with the main exception being hyphens, which are split into separate tokens. We split all queries in the same way before searching the corpus in order to account for this difference. Automatic sentence boundary detection was also applied to the web text, with special tokens marking the beginning and end of sentences. A frequency cutoff of 200 was applied to all unigrams, with words that fell below the threshold being replaced with a special unknown word token. A cutoff of 40 was applied to all other *n*-grams. The immense amount of data in the Web 1T corpus allows for accurate counts with excellent coverage.

It is difficult to calculate D in Equation 5.9, as it requires a count of every bigram in the corpus that does not include two specific words. Instead, we determine a figure for N and then calculate $D = N - A - B - C$. When using search engine counts, we take the same estimate for N as Nakov and Hearst (2005a): 8 trillion. For the Web 1T corpus, we can count the total number of bigrams, giving a figure of 910,884,463,583 for N.

One problem with the bigram probability and χ^2 measures is that a single zero count will cause the entire measure to be zero, ignoring the effect of other non-zero counts. To solve this problem, Nakov and Hearst (2005a) apply a basic form of smoothing: Lidstone's Law (Manning and Schutze, 1999, page 204), by adding 0.5 to the denominator of both measures. Although this is not a particularly effective form of smoothing, we use the same technique so that our results will be comparable with theirs.

COUNTS	ADJACENCY			DEPENDENCY		
	RAW	PROB.	χ^2	RAW	PROB.	χ^2
Google	72.5	68.4	**73.0**	**77.5**	75.0	76.2
MSN	71.3	65.6	**72.1**	**75.0**	74.6	74.6
Web 1T	74.2	70.5	**75.4**	81.2	**82.8**	77.5

Table 5.5: Unsupervised results for Lauer's data set

COUNTS	ADJACENCY			DEPENDENCY		
	RAW	PROB.	χ^2	RAW	PROB.	χ^2
Google	75.53	69.85	**79.98**	69.58	68.61	**69.94**
MSN	76.53	74.38	**80.07**	69.22	69.29	**69.82**
Web 1T	**80.05**	79.62	79.33	74.18	**75.18**	70.71

Table 5.6: Unsupervised results for Penn Treebank data set

5.4.2 Initial Results

The results from the experiments, on both Lauer's and our data set, are shown in Tables 5.5 and 5.6 respectively. Our results on Lauer's corpus are similar to those reported in the literature: the 78.7% achieved by Lapata and Keller (2004) and the 80.7% attained by Lauer (1995b). The dependency model outperforms the adjacency model on all measures, while the Web 1T counts are the most effective. The raw counts – the simplest association measure – also work surprisingly well. The results on the Penn Treebank corpus are also surprising, as the adjacency model outperforms the dependency model by a wide margin. Once again, the Web 1T counts perform well in all cases, although the best result uses the MSN search engine. The χ^2 measure gives the highest accuracy for both search engines, but is least effective with the Web 1T counts. The two search engines give reasonably similar results on both data sets, a result that is not unexpected.

Our analysis shows that the good performance of the adjacency model comes from the large number of named entities in the corpus. When we remove all items that have any word as a named entity, the results are reversed, and the dependency model is superior. On the 1,556 NPs that remain, using Web 1T counts and the χ^2 measure, the adjacency model achieves 71.85% accuracy, while the dependency model attains 73.84%. The results using the other count sources and association measures show the same trend.

Table 5.7 shows the correlation between the count sources on the two data sets. It is

COUNT SOURCES	LAUER	PTB
Google / MSN	0.97	0.72
Google / Web 1T	0.92	0.60
MSN / Web 1T	0.93	0.81

Table 5.7: Correlation between counts

clear that the Penn Treebank corpus has a much larger degree of variability than Lauer's data set. However, Tables 5.5 and 5.6 showed that in most cases, the results for both corpora were more affected by the association measure and whether the adjacency or dependency model was used, rather than the source of counts. These results agree with those of Nakov and Hearst (2005b), who found that although the counts returned by search engines varied, it did not have a significant impact on the performance of their NP bracketing system.

5.4.3 *n*-gram Variations

Both the adjacency and dependency models are relatively knowledge-poor, only utilising a pair of bigram counts in order to make a decision. In order to increase the amount of information available, we retrieved hit counts for a number of other variations on the simple bigrams, as proposed by Nakov and Hearst (2005a). We did not perform the morphological or paraphrase queries of Nakov and Hearst, as they would increase the number of searches dramatically. The full list of query variations that we used is shown in Table 5.8, together with whether or not each count source used that particular pattern. Some patterns cannot be used by some count sources, for example, *n*-grams that are longer than five words are not listed in the Web 1T corpus. In some of these cases, we instead included a different query variation that was intended to capture the same information, e.g. the multiple abbreviation and possessive queries. Also, we no longer use MSN because of its similarity to Google. Snippets is another source of counts suggested by Nakov and Hearst (2005a), utilising the short piece of text that comes with each search result. The variations involving hyphens are not ideal when using Web 1T, as it cannot be determined whether the author intended them as a separate token, or whether they were tokenised when the corpus was created.

As we described in Section 5.2.1, Nakov and Hearst (2005a) used *n*-gram variations in a complicated voting scheme, where different counts from different sources were given hand-tuned weights and then combined. Rather than implementing such a complex algorithm, we performed some simpler voting experiments. Each *n*-gram variation, as well as the standard adjacency and

NAME	LEFT BRANCHING	RIGHT BRANCHING	GOOGLE	WEB 1T	SNIPPETS
Wildcard 1	brain stem * cells	brain * stem cells	√	×	×
Wildcard 2	brain stem * * cells	brain * * stem cells	√	×	×
Wildcard 3	brain stem * * * cells	brain * * * stem cells	√	×	×
Reverse wildcard 1	cells * brain stem	stem cells * brain	√	×	×
Reverse wildcard 2	cells * * brain stem	stem cells * * brain	√	×	×
Reverse wildcard 3	cells * * * brain stem	stem cells * * * brain	√	×	×
Adjacency concatenation	brainstem	stemcells	√	√	×
Dependency concatenation	brainstem	braincells	√	√	×
Concatenation triple	brainstem cells	brain stemcells	√	√	×
Swap first two words	brain stem cells	stem brain cells	√	√	×
Reorder	cells brain stem	stem cells brain	√	√	×
Abbreviation	brain stem bs cells	brain stem cells sc	√	√	×
Abbreviation with brackets	brain stem (BS)	stem cells (SC)	×	√	×
Possessive	stem's	brain's	√	×	×
Possessive triple	brain stem's cells	brain's stem cells	√	√	√
Capitalisation	brain stem Cells	brain Stem cells	×	√	√
Internal hyphenation	brain-stem cells	brain stem-cells	×	√	√
External hyphenation	brain stem cells-*	*-brain stem cells	×	√	√
Internal slash	brain/stem cells	brain/stem cells	×	√	√
External slash	brain stem cells/*	*/brain stem cells	×	√	√
Left brackets	(brain stem) cells	(brain) stem-cells	×	√	√
Right brackets	brain stem (cells)	brain (stem-cells)	×	√	√
Comma	brain stem, cells	brain, stem cells	×	√	√
Colon	brain stem: cells	brain: stem cells	×	√	√
Period	brain stem. cells	brain. stem cells	×	√	×
N&H period	brain. stem cells	brain stem. cells	×	√	√

Table 5.8: Variations on the basic query

GOOGLE	WEB 1T	SNIPPETS
Wildcard 2	Dependency probability	Possessive
Abbreviation	Concatenation triple	Capitalisation
Possessive	Abbreviation with brackets	Internal hyphenation
	Capitalisation	Right brackets
	Internal hyphenation	
	Internal slash	
	External slash	
	Left brackets	
	Right brackets	

Table 5.9: Optimal set of voters on Lauer's data set

dependency models, was given a single unweighted vote. If the left and right counts were equal, then the variation supplied no vote, and if the final votes were equally split, then we defaulted to a left-branching decision.

We performed a greedy search through the possible sets of voters, attempting to optimise performance on Lauer's data. Our best result uses the list of voters in Table 5.9. This set achieves 90.2% accuracy, a similar figure to Nakov and Hearst's 89.3%, without using the morphological or paraphrase queries, and without manually weighting any features.

However, both of these voting systems are effectively supervised models, where the training process determines the optimal set of features (and weights for Nakov and Hearst's model). As such, a separate training set should be used to avoid over-estimating performance. Due to the small size of Lauer's data set, we followed Nakov and Hearst (2005a) in developing on the test data itself. They note that Lapata and Keller (2004) divided Lauer's in half to develop, and that the difference in performance on the two halves was negligible. Despite this, we argue that neither of the results give an accurate representation of NP Bracketing performance. The optimal set of voters we identified is unlikely to be as effective for any other data set.

We can actually test this by applying the Lauer optimal voter set (from Table 5.9) to the Penn Treebank data. This results in 76.49% accuracy, which is lower than using the adjacency model alone. Considering the seemingly random variations in the set of voters, this is not particularly surprising, although it may be because of the different performance levels of the dependency and adjacency models on the two corpora. It could also be a result of the lexical differences between the two corpora, as we showed in Table 5.3. In the following section, we will perform the reverse operation, training on the Penn Treebank data and testing on Lauer's. This will provide a better idea of the true performance levels.

The main problem with a voting technique is that it is does not effectively combine competing factors into a single model. The new Penn Treebank data set enables a much better solution: apply a robust supervised model. This Penn Treebank data set is an order of magnitude larger than Lauer's, making available a sufficient amount of training, development and test data for the first time.

5.4.4 Supervised Models

Supervised models typically outperform unsupervised models for most NLP tasks. For NP bracketing, the small quantity of gold-standard data has meant that few supervised models have been implemented, and those that have been, performed poorly. With our new, significantly larger data set covering the Penn Treebank, we have built the first large-scale supervised NP bracketer.

We use the MegaM Maximum Entropy classifier (Daumé III, 2004), which, as we described in Section 2.6, allows diverse and overlapping features to be incorporated in a principled manner. We also discretise non-binary features using Hawker's (2007) implementation of Fayyad and Irani's (1993) supervised entropy-based discretisation algorithm.

The data set is split into training, development and test sets, with 4451, 559 and 559 NPs respectively. Our initial features use counts from Google, Web 1T and the snippets. We use the adjacency and dependency models with counts from Google and Web 1T, and all three association measures. The *n*-gram variations in Table 5.8 for the three count sources are also used, but only the raw count. This is because the counts are often too small for the other measures to be effective. For each of these, there is a feature for the left and right association measure score, as well as a binary feature representing the left or right vote. If the left and right measures are equal, then neither vote feature is active. This first supervised model has 947 features in total.

The results on our Penn Treebank development set are shown in Table 5.10, compared to an unsupervised adjacency model, and the unsupervised voting system from Section 5.4.3. As we described there, calling the latter model unsupervised is a misnomer, as the set of voters needs to be optimised on training data. With the larger Penn Treebank corpus available, we can now "train" this unsupervised voting model on the training set, rather than on the test set itself. This avoids over-estimating its performance figures.

The supervised model outperforms the unsupervised voting model by 0.6%, even though both models are using the same information to base their decisions on. This improvement comes

MODEL	F-SCORE
Unsupervised, Web 1T adjacency	82.5
Unsupervised, voting	89.6
Supervised model	90.2

Table 5.10: Comparing unsupervised approaches to a supervised model

from the supervised model's ability to weight the individual contributions of all the unsupervised counts from Google and the Web 1T corpus.

We can also test on Lauer's data set using the supervised model trained on Penn Treebank data. The result is an 82.4% accuracy figure, which is higher than our unsupervised dependency model and Lauer's. However, it is much lower than Nakov and Hearst's (2005a) best result and our own voting scheme. This suggests that the voting schemes, by training on their own test data, have over-estimated their performance by about 9%.

Additional Features

One of the main advantages of using a Maximum Entropy classifier is that we can easily incorporate a wide range of features in the model. We now add lexical features for all unigrams, bigrams and the trigram within the NP. All of these features are labelled with the position of the *n*-gram within the NP.

Since we are bracketing NPs *in situ*, rather than stand-alone NPs as performed by Lauer, the context around the NP can be exploited as well. To do this we added bag-of-word features for all words in the surrounding sentence, as well as specific features for a two-word window around the NP. For the context sentence, there are features for words before the NP, after the NP, and either before or after the NP. As an example, when bracketing *lung cancer deaths*, in the sentence:

The number of lung cancer deaths has grown recently.

the context sentence bag-of-words features would be *The, number, of, has, grown, recently*. And the context window features would be:

$$\begin{array}{lllllll} \text{word}_{-2} & = & \textit{number} & \quad & \text{word}_{-1} & = & \textit{of} \\ \text{word}_{+1} & = & \textit{has} & \quad & \text{word}_{+2} & = & \textit{grown} \end{array}$$

We have access to gold-standard POS and NER tags, from the Penn Treebank and the BBN Pronoun Coreference and Entity Type Corpus (Weischedel and Brunstein, 2005) respectively. These are used by adding generalised features for every *n*-gram and context window feature, replacing the words with their POS and NER tags. POS tags are included even though all the words in the NP are nouns for these simple NP experiments, as they may be proper and/or plural. We use the coarse-grained NER tags, of which there are 28 (plus `O`), including the B- and I-. Using the *lung cancer deaths* example again, its POS tag trigram would be `NN NN NNS` and its NER tag trigram is `O B-DISEASE O`.

Finally, we incorporate semantic information from WordNet (Fellbaum, 1998). For each sense of each word in the NP, we extract a semantic feature for its synset, and also the synset of each of its hypernyms up to the WordNet root. These features are marked with how far up the tree from the original synset the hypernym is, but there is also an unordered bag-of-hypernyms for all senses.

All of these semantic features are applied to each word in the NP, including a label describing whether it is the first, second or third word. For the word *cancer* there are five synsets to which it belongs: *malignant neoplastic disease, Crab, Cancer, Cancer the Crab, genus Cancer* all of which are included. The first level of hypernyms for these synsets is *malignant tumor, person, arthropod genus* (two of the senses have no hypernyms), and this continues up the tree. The bag-of-hypernyms would include all of the synsets we have listed and many more.

These additional feature types increase the number of features in the maximum entropy classifier to 86,116, compared to the 947 we had previously. This number is still small compared to some other tasks, because our data set is comparatively small, being made up of only 4,263 unique tokens. Almost all of the models converge after 50 training iterations, the one exception being that using only unsupervised features, which takes about 200.

Results

Table 5.11 shows the results for a model using only the additional features, and also once the "unsupervised" features used in Table 5.10 are included. The additional features do not perform as well as the unsupervised ones, but once they are combined a further performance increase of 3.5% is attained.

Table 5.11 also presents a subtractive analysis of all feature groups. The Google and snippets features do not appear to contribute at all, probably because they overlap significantly with each other and the Web 1T searches. Of the supervised features, the context window and NER are most important but all make a positive contribution, except for the semantic features. Our best performance of 93.8% F-score is obtained by removing this group.

MODEL	F-SCORE
Unsupervised, voting	89.6
Additional features	89.5
Additional + unsupervised features	93.0
−Google	93.0
−Snippets	93.0
−Web 1T corpus	92.1
−Lexical	92.3
−POS	92.5
−NER	92.1
−Context sentence	92.7
−Context window	92.0
−Semantic	**93.8**

Table 5.11: Subtractive analysis of simple NP features on development set

MODEL	ACCURACY
Unsupervised, Web 1T adjacency	77.6
Unsupervised, voting	86.8
Best supervised model	93.4

Table 5.12: Test set results for the supervised model

Finally, results on the test set are shown in Table 5.12. The supervised model has improved over the unsupervised baseline by 6.6%. This larger increase, compared to the development set, shows that the voting method's performance is quite variable, while the Maximum Entropy model remains consistent.

5.5 Experiments with Complex NPs

All of our experiments so far have been on simple NPs. Complex NPs can be longer, resulting in higher levels of ambiguity and making the correct bracketing significantly more difficult to find. Adjectives, determiners and other non-nominal parts of speech also complicate the task. But the greatest difficulty introduced by complex NPs is constructing an architecture that can choose between potential bracketings.

The number of possible bracketings depends on, and grows exponentially with, the length of the phrase. Incorporating this into a machine learner may be difficult, and a better solution *w* is the current position of the window

1. *w* initially covers the last 3 words
2. Bracket the words in *w*
3. If *w* is left branching:
   (a) If *w* is in the left-most position, bracket the left two words in *w*
   (b) Otherwise, move *w* one word to the left. We cannot left bracket yet, because it might be X)Y)Z, not (XY)Z
4. If *w* is right branching:
   (a) Bracket the right two words in *w*
5. If there are only two words left, then finish. Otherwise, go to step 2

Figure 5.3: Barker's (1998) NP bracketing algorithm

may be to break the complex NP bracketing decision into multiple smaller decisions. This would be analogous to the way that the Collins (2003) model makes independence assumptions when calculating grammar rule probabilities. The algorithm we use functions in this way, as will be described in the following section.

Unfortunately, this does cause one complication with our terminology. When our complex NP Bracketing system splits up a complex NP into multiple three word NPs, these three word NPs may not be simple. That is, they may contain adjectives and other non-noun parts of speech. Despite this, we will apply simple NP Bracketing techniques to them. It is important to remember the distinction between simple and complex NPs before entering into the experiments of this section.

5.5.1 Barker's Algorithm

Barker (1998) describes a method for bracketing complex NPs, by reducing the problem to a series of three word bracketing decisions using a sliding window. In this way, the algorithm allows us to take advantage of the supervised models that we developed and that proved effective on simple NPs. Barker's algorithm is shown in Figure 5.3.

When a pair of words is bracketed, the head is chosen to represent the phrase and remains in the window. We use the standard head-finding rules of Collins (1999). The window then expands one word to the right, unless it is already right-most in which case it grows to the left.

For these experiments, we use the complex NP data set previously described in Section 5.3.2. The 53,568 complex NPs are split in a 8:1:1 ratio, giving 42,854 examples for training and 5357 each for development and testing.

5.5.2 Evaluation Measures

The complex NP results are evaluated using several measures. Firstly, matching brackets is one of the standard Parseval evaluation methods (Black et al., 1991). Secondly, because our annotation only marks left-branching structure explicitly (see Section 3.1), we can also report implicit matching brackets, where we automatically insert the implicit right-branching brackets for evaluation purposes. This takes into account fully right-branching NPs, which contribute no score using the harsher, explicit matching brackets evaluation. For example, a baseline of always choosing right branching will achieve 0.0% F-score, as no explicit brackets will be inserted.

We also measure exact NP match, which measures the percentage of complex NPs that are entirely correct, and the model's performance on the three word NPs that are processed during Barker's algorithm. We only report accuracy for implicit brackets, as there is a set number of brackets dependent on the length of the word, and so precision and recall are always equal. Finally, note that the three word NPs are different for each model, as the next three word NP to bracket depends on the decisions made previously for this complex NP. Consequently, the numbers for this measure are not directly comparable.

5.5.3 Results

Our first experiment implements Barker's algorithm, using only the χ^2 dependency and adjacency methods to make each decision. We only use counts from the Web 1T corpus, since performing web searches has become impractical with the increased data set size and NP length. The difficulty of complex NP bracketing can be seen in Table 5.13, by the drop in performance using these simple approaches, e.g. the adjacency model drops from 79.33% in Table 5.6 to 56.29% here.

We next apply our supervised approach to complex NPs. This is more complicated now as we need to extract a training set of three word windows from the complex NPs. To do this, we run Barker's algorithm on the 42,854 complex NPs. At each decision point, we bracket left or

MODEL	MATCHED BRACKETS			IMPLICIT	EXACT	SIMPLE
	P	*R*	*F*			
Right-branching	0.00	0.00	0.00	74.67	69.31	74.82
χ^2 Dependency	13.79	42.84	20.87	38.40	24.32	48.11
χ^2 Adjacency	16.13	41.00	23.15	49.27	34.50	56.29
All features	**89.14**	**84.26**	**86.63**	**94.96**	**92.18**	**95.67**
−Web 1T corpus	*89.58*	82.79	86.05	94.75	91.69	95.55
−Lexical	87.95	83.00	85.40	94.57	91.58	95.30
−POS	89.09	83.37	86.13	94.73	91.92	95.44
−NER	*89.27*	84.11	86.61	94.88	*92.25*	95.64
−Context sentence	*91.45*	*86.11*	*88.70*	*95.69*	*93.19*	*96.33*
−Context window	*90.41*	*85.37*	*87.82*	*95.32*	*92.79*	*96.00*
−Semantic	*89.61*	84.00	*86.72*	*94.97*	92.14	95.67
−Non-head words	84.84	81.58	83.18	94.03	90.76	94.80
−Border words	*89.69*	*84.74*	*87.14*	*95.18*	*92.48*	*95.85*
−POS tag sequence	*89.93*	*85.05*	*87.42*	*95.26*	*92.70*	*95.96*
−Parser	*89.35*	*84.32*	*86.76*	*95.04*	*92.25*	*95.78*
Best	92.09	86.37	89.14	95.88	93.49	96.48

Table 5.13: Complex NP results on development data

right according to the gold standard, and store the three word window as a training example. This process is similar to a shift-reduce parser, like that used in the RASP parser (Briscoe and Carroll, 2006) or Ratnaparkhi's (1997) maximum entropy parsing model. The complex NP data produces 95,964 training examples.

We experiment with the same features used for simple NPs, as well as some novel features. Firstly, we add features encoding the non-head words when the window already contains a bracket. This means that for each bracket that has already been inserted for the complex NP, all words dominated by the bracket are labelled with their position in the window and added as features. For example, consider the NP *French onion soup bowl* after *onion soup* has been bracketed. Although only *soup* remains in the window, *onion* is added as a feature and labelled as the first word of the second node in the window. The POS tag, NER tag and Web 1T count of these words are also included as separate features. This feature group proved to be very informative for the model.

Secondly, we add the bigram of the words on the NP border, that is, where it overlaps with the context. This combines the *n*-gram features with the context window. For example, *lung cancer deaths* would have two features: *of lung* and *deaths has*.

Thirdly, we measured the entropy of every POS tag sequence in the training data. Some

MODEL	MATCHED BRACKETS			IMPLICIT	EXACT	SIMPLE
	P	*R*	*F*			
Right-branching	0.00	0.00	0.00	72.79	68.08	72.98
χ^2 Adjacency	17.76	41.89	24.95	50.08	36.64	57.07
Best	91.32	88.19	89.73	95.95	93.69	96.68

Table 5.14: Complex NP results on test data

of these figures were shown earlier in Table 5.4. Those sequences with entropy below 0.05 bits, i.e. the ones that are quite unambiguous, were then extracted. For example, `DT JJ NN` is almost always right-branching. We then implemented a feature explicitly encoding their most common branching. There are only two features for left and right-branching, rather than features for each POS tag.

Finally, we introduce features based on the Bikel (2004) parser's output, which have been informative in PCFG parsing. For the parent and grandparent of the NP, we add a feature for the constituent label, as well as the head-word and its POS tag, NER tag, and Web 1T count.

The results are shown in Table 5.13. The supervised methods significantly outperform the unsupervised methods, with a matched brackets F-score comparable to the Bikel (2004) parser's overall performance. We hope that a similar performance figure is reached later on, when we use a post-processor to improve the parser's performance on NP structure. We carry out a subtractive analysis of the feature types and find that both context feature groups, as well as the semantic, border, POS tag rule and parser features all have a negative impact on performance. Our optimal result comes from removing these feature groups. The 89.14% F-score achieved with this model is shown as **Best** in Table 5.13. All experiments were run using 500 iterations in MegaM, to allow the estimation to converge.

Finally, we applied our best model to the test data. The results in Table 5.14 are similar to those we achieved on the development set. This demonstrates that our complex NP Bracketing system achieves high performance on a wide range of inputs.

5.6 Parser Post-Processor

This final set of experiments uses the complex NP models as a post-processing step for the Bikel (2004) parser. As we saw in Chapter 4, the parser failed to outperform the suggestion baseline of 73.12% on `NML` and `JJP` brackets. We intend to surpass this figure with our NP bracketing technique. This will be made more difficult by the fact that the post-processor is dependent on NPs

MODEL	MATCHED BRACKETS			IMPLICIT	EXACT	SIMPLE
	P	*R*	*F*			
Right-branching	0.00	0.00	0.00	81.83	80.31	81.86
χ^2 Dependency	9.93	39.90	15.90	36.46	31.20	43.23
χ^2 Adjacency	12.50	42.55	19.32	47.24	41.41	51.37
All features	**76.37**	**83.53**	**79.79**	**93.04**	**92.42**	**93.70**
−Web 1T corpus	*77.10*	80.53	78.78	92.90	*92.45*	93.55
−Lexical	73.67	81.73	77.49	92.23	91.66	92.97
−POS	*76.61*	83.05	79.70	*93.36*	*92.65*	*93.96*
−NER	*76.78*	85.46	*80.89*	*93.43*	*92.70*	*94.08*
−Context sentence	*78.53*	*84.86*	*81.57*	*93.78*	*93.26*	*94.38*
−Context window	*76.41*	*84.50*	*80.25*	*93.33*	*92.57*	*93.98*
−Semantic	75.73	*83.65*	79.50	*93.11*	92.37	*93.83*
−Non-head words	74.21	81.97	77.90	*93.15*	91.97	*93.78*
−Border words	76.37	83.53	79.79	*93.27*	*92.50*	*94.01*
−POS tag sequence	*76.77*	*84.62*	*80.50*	*93.55*	*92.68*	*94.27*
−Parser	76.33	*84.50*	*80.21*	*93.33*	*92.60*	*93.95*
Best	78.78	85.70	82.10	94.08	93.41	94.67

Table 5.15: Complex NPs for parsing results

identified by the parser, which are incorrect in approximately 10% of cases.

Atterer and Schütze (2007) use a similar approach, applying prepositional phrase attachment techniques to parser output, rather than to manually prepared, gold-standard examples. Doing so provides a more realistic view of a PP attachment system's performance, as it must contend with the additional difficulties created by parser error. The same applies to our NP Bracketing system.

We train the complex NP bracketer on gold-standard NPs from Sections 02–21, extracting 78,757 complex NPs that produce 132,195 three word training examples. The development set is created by first parsing Section 00 using the Bikel (2004) parser. We then extract the base-NPs that the parser identifies and insert the gold-standard NP bracketing for evaluation. We reject brackets that cross an NP boundary (i.e. a parsing error). This results in a development set of 3,946 complex NPs. A test set of 4,834 NPs is also produced in the same way from Section 23.

The results of these experiments, including subtractive analysis on the feature types, are shown in Table 5.15. Unfortunately, many of the features are not helpful, and our best model utilises only the Web 1T, lexical, POS and non-head word features. It achieves 82.10% matched bracket F-score.

This is 7.04% lower than the figure previously achieved for complex NPs, despite the fact

EVALUATING	MODEL	*P*	*R*	*F*
NML JJP	Suggestions	95.64	59.36	73.25
	Parser	76.32	60.42	67.44
	Post-processor	76.40	76.56	76.48
All brackets	Parser	88.55	88.15	88.35
	Post-processor	88.49	88.56	88.53

Table 5.16: Development data performance

that unambiguous NPs are now included in the data. There are a number of reasons for this. Firstly, the test NPs produced by the parser may be incorrect, while the model is trained on gold-standard NPs. Also, the brackets that we rejected for crossing NP boundaries would introduce a noticeable amount of noise, and mean that the evaluation might not be entirely accurate. Finally, the POS tags used in these experiments are no longer gold standard, as they come from the parser's output.

5.6.1 Parsing Evaluation

Finally, we can now put the rebracketed NPs back into the parser output and re-evaluate. This requires the additional task of labelling the brackets. There are only two labels to distinguish between (NML and JJP), and they can be inferred directly from the POS tag of the head. If it is a verb or an adjective, we label the node as JJP, and otherwise it is a NML. A small number of errors (0.42% drop in matched bracket F-score) are introduced by this method, because of errors in the Penn Treebank POS tags and in our annotation, as well as errors in head finding.

Tables 5.16 and 5.17 show the final results. A suggestion result is not shown for all brackets since they only apply to NMLs and JJPs and it is difficult to post-process the parser's output with them. The post-processor outperforms the parser by 9.04% and 8.10% on the development and test data respectively. The post-processor has also improved on the suggestion baseline established earlier. These results demonstrate the effectiveness of large-scale NP bracketing techniques, and show that internal NP structure can be recovered with better performance than has ever been possible in the past.

We also measure statistical significance using a computer-intensive, randomised, stratified shuffling technique, as described in Noreen (1989) and Cohen (1995, §5.3). The null hypothesis — that the results are produced by the same model — is tested by swapping the scores on individual sentences between the two models. These swaps are performed repeatedly, with precision, recall

EVALUATING	MODEL	*P*	*R*	*F*
NML JJP	Suggestions	94.29	56.81	70.90
	Parser	80.06	63.70	70.95
	Post-processor	79.44	78.67	79.05
All brackets	Parser	88.30	87.80	88.05
	Post-processor	88.23	88.24	88.23

Table 5.17: Test data performance

and F-score recalculated for each model at each iteration. A count is kept of how many times the difference between these recalculated metrics is greater than or equal to the difference between the original figures. The null hypothesis is rejected if this number is sufficiently low.

Ideally, all possible permutations would be performed, however this is infeasible, as it would require testing 2^n permutations, where n is the number of sentences (2,416). Instead, an approximate randomised test can be used with a sufficiently large number of iterations, in this case 10,000. The p-value is then calculated as:

$$p = \frac{c + 1}{n + 1} \tag{5.11}$$

where c is the number of random swaps that resulted in a difference greater than or equal to the original difference, and n is the number of iterations performed.

The p-value on the test data all-brackets F-score is 0.0001. This is the smallest p-value attainable for the number of iterations we performed. That is, there were no iterations whatsoever where randomly swapped figures resulted in an F-score difference that was as large as the original, unswapped permutation. The p-value for the recall measure gave the same result. For precision, where the parser is actually superior to the post-processor, we calculated a p-value of 0.0163. Thus, this difference is also statistically significant, although less so than the recall and F-score metrics.

5.7 Summary

We have created the first large-scale supervised models that achieve excellent results. These experiments are also the first to scale effectively to complex NPs, attaining similarly high levels of performance. We expect that the data and models described in this chapter will provide the impetus for much more work on NP Bracketing in the future.

One particularly important contribution of this chapter is the data sets that we have created. These data sets are orders of magnitude larger than those used previously, and have made possible the wide range of experiments we carried out. The final result of this chapter, where our post-processor outperforms the Collins (2003) parser, is another of the major contributions of this thesis. We previously observed the tremendous difficulty in bracketing NPs, demonstrated by the below-baseline performance of the parser in Chapter 4. We have now overcome this difficulty and outperformed the suggestion baseline.

Chapter 6

Parsing with CCG

Although the NP Bracketing system was successful, using a post-processor is hardly an elegant solution. A better solution would be to include the NP model into a standard parsing model, as this would allow NP structure to be optimised together with the entire sentence. In this chapter, we will make such an addition to the C&C CCG parser (Clark and Curran, 2007b). There are a number of advantages to this approach:

- The C&C parser uses a maximum entropy model, which will make it relatively easy to add NP-based features, compared to the Collins (2003) models.
- It will allow us to correct the errors in the CCG corpus, CCGbank, which we described in Section 2.5.1.
- Utilising a second parser will demonstrate that NP structure is recoverable across multiple parsing architectures.

6.1 The C&C Parser

In Section 2.5.2, we described the C&C parser (Clark and Curran, 2007b). Here, we will describe the features used by the parser in its Maximum Entropy model. This will be relevant for when the novel features we have added are described in Section 6.5.

Firstly, the model uses a lexical feature that combines the word and its lexical category. Another feature generalises the word to its POS tag. There are also features that are only active for the root constituent of the sentence. These are the root category; the root category and its head word;

and the root category and its head word's POS tag. Another feature that applies to all non-terminal constituents is the rule that was applied to generate it. Once again, this feature is also expanded to the head word of the constituent, and generalised to the head word's POS tag.

All of the above features are identical in both the normal-form and dependency models. However, for the following features, the former uses local rule applications, such as $S[dcl] \rightarrow NP\ S[dcl]\backslash NP$; while the latter uses the predicate-argument dependencies, like the one that will be described in Section 6.2.1. The dependency model can thus include information from long-range dependencies.

For any non-terminal constituent, the head words of the child nodes form a feature. The rule or dependency involved is also included, for the normal-form and dependency models respectively. This feature is also generalised, to the first word's POS tag; the second word's POS tag; and the first and second word's POS tag.

The last feature group adds distance information to the models. The number of words, verbs and punctuation marks between the head words of the child constituents are counted. All counts two or greater are subsumed into a single class. The feature also includes the parent constituent's head word, and the rule or dependency that applies.

Most of these features will be described again in Section 6.5.1, where we give examples of the original features compared to their generalisations that use named entity tags.

6.2 Hockenmaier's Conversion Process

Converting sentences from the Penn Treebank Phrase Structure to a CCG representation is a very complex task. Hockenmaier and Steedman's (2007) algorithm and the resulting corpus, CCGbank, must not only carry out the translation itself, but also robustly deal with significant differences between the two corpora. A large part of these differences arise from the need to translate Penn Treebank traces and their co-indexation to CCG long-range dependencies. The two also differ on how to analyse various syntactic constructions, such as the small clause example below:

```
(S (NP-SBJ that volume)
  (VP (VBZ makes)
    (S (NP-SBJ it)
      (NP-PRD the largest supplier...in Europe) ) ) )
```

Following Pollard and Sag (1992) and Steedman (1996), CCGbank treats these constructions as having two NP complements, rather than a nested sentence. Here, we will not describe every such complication, but give an outline of how the conversion algorithm works.

The basic algorithm applied to every sentence in the Treebank is:

1. Determine the type of each constituent (head, complement or adjunct);
2. binarise the tree;
3. assign lexical categories;
4. assign dependencies.

The first step labels each constituent in the tree with its syntactic role. Heads are identified using heuristics adapted from Magerman (1994) and Collins (1999), while additional heuristics are applied to distinguish between adjuncts and complements. For example, NPs that are children of VPs are labelled as complements, except where the NP has a function marker such as LOC, DIR or TMP.

Binarisation is a simple step, requiring only the constituent head as information. All children to the left of the head are placed into additional right-branching nodes, while all children on the right of the head are placed into new left-branching nodes. Note that this is the step that transforms the Penn Treebank's flat, underspecified NPs into CCGbank's right-branching, possibly incorrect representation.

CCG lexical categories are assigned to the tree in a recursive, top-down process. Firstly, the root node's label is determined according to the root Penn Treebank label. For example, S, SINV and SQ all become *S* in the CCGbank tree, while additional features provide a further level of information. *S*[*dcl*] indicates a declarative sentence, which is applied when the head word is an auxiliary with the POS tag VBZ. Other features include *S*[*wq*] for *wh*-questions, and *S*[*frg*] for fragments. Complements are assigned a category in a similar way, using a mapping from the Penn Treebank constituent label. The simplest of these rules being NP becomes *NP* and PP becomes *PP*.

Each head node is assigned a label based on its parent, as well as any complement that is present. These two lexical categories are combined, with the direction of the slash determined by the relative positions of the head and its complement. That is, with a parent node X and a complement Y, the head will be assigned the category X/Y if the complement is on the right, or $X\backslash Y$ if the complement is on the left.

Adjuncts function as modifiers that return the original category, and are thus given the label X/X or $X\backslash X$, where X is the lexical category of the head. Applying this method in all cases leads to an unnecessary proliferation of categories, as it requires a modifier label for each head subcategorisation. To avoid this, the translation algorithm takes advantage of the CCG composition rules, stripping off the head's outermost forward arguments before generating the adjunct label.

6.2.1 Dependencies

The final step in the basic algorithm is to generate the dependency structure for each sentence. Dependencies are represented as 4-tuples: $\langle h_f,\ f,\ s,\ h_a\rangle$, where h_f is the head of the predicate; f is the lexical category of h_f; s describes which argument of f is being filled; and h_a is the head of the argument. For example, the dependency encoding *company* as the object of *bought* (as in *IBM bought the company*) is represented by:

$$\langle bought, (S\backslash NP_1)/NP_2, 2, company\rangle \tag{6.1}$$

In this dependency, *company* is filling the second argument slot, the object.

These dependencies are generated in a bottom-up process, using the lexical head information on all CCGbank categories. This information is shown as the indices on the arguments in the example above. For modifier categories, co-indexation is necessary to specify that the head of the category is the head of its argument. For example, in the category $(S\backslash NP)_i/(S\backslash NP)_i$, the matching indices represent this relationship. Using this lexical head information, a dependency can be generated between the two heads whenever a rule is applied.

6.2.2 Traces and Long-Range Dependencies

One of the most important parts of the CCGbank conversion algorithm is to translate Penn Treebank traces into the correct CCG long-range dependencies. The * null trace is used to identify passive, control, raising and extraposition constructions, which can then be interpreted accordingly.

Extraction traces, marked with `*T*` in the Penn Treebank are more complicated to deal with. These traces are passed up the tree, in a manner similar to Collins' Model 3 and the GPSG slash-feature passing technique (Gazdar et al., 1985) it was based upon. When lexical categories are subsequently assigned, the trace that has percolated up the tree indicates that non-normal-form rules should be applied. Thus, the correct derivation is reached, including the co-indexation that will in turn create the appropriate long-range dependency.

6.3 Our Conversion Process

In order to train and test the C&C parser on our NP data, we will have to convert it from the Penn Treebank matched bracket format into the CCG formalism. This process will correct the NP structure errors that currently exist in CCGbank, which we described in Section 2.5.1. Over a quarter of the sentences in CCGbank need to be altered, demonstrating the magnitude of the NP problem and how important it is that these errors are fixed.

Honnibal and Curran (2007) have altered CCGbank in a similar way, in an attempt to improve the distinction between complements and adjuncts. PropBank (Palmer, Gildea, and Kingsbury, 2005) was used as a gold-standard to inform these decisions, in the same way that we use the NP data from Chapter 3.

6.3.1 The Algorithm

For each NML and JJP bracket in the augmented Penn Treebank, the conversion process will form a CCG constituent that dominates the same tokens in the corresponding CCGbank sentence. We generate the two forms of output that CCGbank contains: AUTO files, which represent the tree structure of each sentence; and PARG files, which list the predicate-argument dependencies (Hockenmaier and Steedman, 2005, Appendix D).

One preprocessing step is applied to the Penn Treebank data, where if multiple tokens are enclosed by brackets, then a NML node is placed around those tokens. For example, we would insert the NML bracket shown in bold below:

```
(NP (DT a)
  (NML (-LRB- -LRB-)
  (NML (RB very) (JJ negative) )
  (-RRB- -RRB-) )
  (NN reaction) )
```

This simple heuristic captures NP structure not explicitly bracketed in the annotation scheme.

The conversion algorithm applies the following steps for each NML or JJP bracket:

1. Identify the CCGbank *lowest spanning node*, i.e. the lowest constituent that dominates all of the words in the NML or JJP bracket;
2. flatten the lowest spanning node, to remove the right-branching structure;
3. insert new left-branching structure;

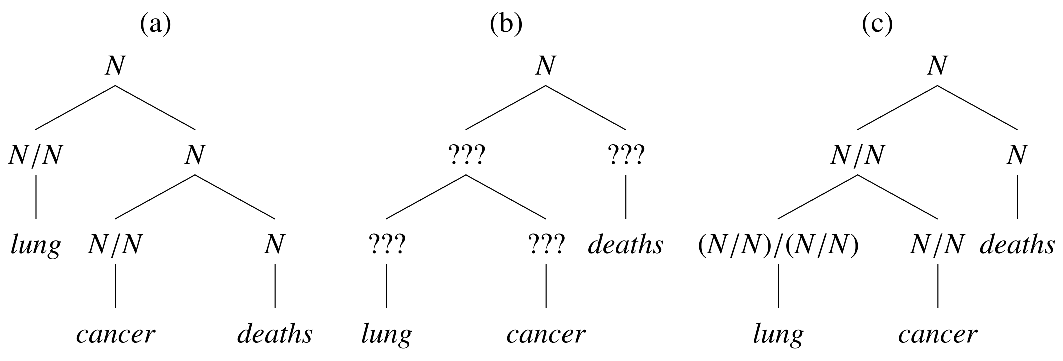


Figure 6.1: (a) Original right-branching CCGbank (b) Left-branching (c) Left-branching with new lexical categories

4. identify heads;
5. assign lexical categories;
6. generate new dependencies.

It is noticeable, and unsurprising, that some of these steps are common to both our algorithm and that of Hockenmaier and Steedman (2007). As an example, we will follow the conversion process for the Penn Treebank NML bracket below:

```
(NP (NML (NN lung) (NN cancer) )
  (NNS deaths) )
```

The corresponding CCGbank lowest spanning node, which incorrectly has *cancer deaths* as a constituent, is shown in Figure 6.1(a). To flatten the node, we recursively remove brackets that partially overlap the NML bracket. Nodes that don't overlap at all are left intact. This process results in a list of nodes (which may or may not be leaves), which in our example is [*lung*, *cancer*, *deaths*]. We then insert the correct left-branching structure, shown in Figure 6.1(b). At this stage, the lexical categories are still incomplete.

Heads are then assigned using heuristics adapted from Hockenmaier (2003a, Appendix A.1). Since we are applying these to CCGbank NP structures rather than the Penn Treebank, the POS tag based heuristics are sufficient to determine heads accurately.

Next, lexical categories are assigned to the new structure. We want to minimise the number of changes that are made to the entire sentence's derivation, and so the lexical category of the dominating node is fixed. Categories are then propagated recursively down the tree. For a node with category X, its head child is also given the category X. The non-head child is always treated as an adjunct, and given the category X/X or $X\backslash X$ as appropriate. Figure 6.1(c) shows the final result of this step for our example.

6.3.2 Dependency Generation

The changes described so far have generated the new tree structure, but the last step is to create the corresponding dependencies. We recursively traverse the tree, at each level creating a dependency between the heads of the left and right children. These dependencies are never long-range, and are therefore easy to generate. We may also need to change dependencies reaching from inside to outside the NP, if the head(s) of the NP have changed. In these cases we simply replace the old head(s) with the new one(s) in the relevant dependencies. The number of heads may change because we now analyse conjunctions correctly, as will be discussed in Section 6.3.3.

In our example, the original dependencies were:

$$\langle lung, N/N_1, 1, deaths, -\rangle \tag{6.2}$$

$$\langle cancer, N/N_1, 1, deaths, -\rangle \tag{6.3}$$

while after the conversion process, the dependency in (6.2) becomes:

$$\langle lung, (N/N_1)/(N/N)_2, 2, cancer, -\rangle \tag{6.4}$$

To determine that the conversion process worked correctly, we manually inspected its output for unique tree structures in Sections 00 – 07. This identified problem cases to correct, such as those described in the following section.

6.3.3 Exceptional Cases

Firstly, when the lowest spanning node covers the NML or JJP bracket exactly, no changes need to be made to CCGbank. These cases occur when CCGbank had already received the correct structure during the original conversion process. For example, brackets separating a possessive from its possessor were detected automatically.

A more complex case is conjunctions, which do not follow the simple head/adjunct method of assigning lexical categories. Instead, conjuncts are identified during the head-finding stage, and then assigned the lexical category dominating the entire coordination. Intervening non-conjunct

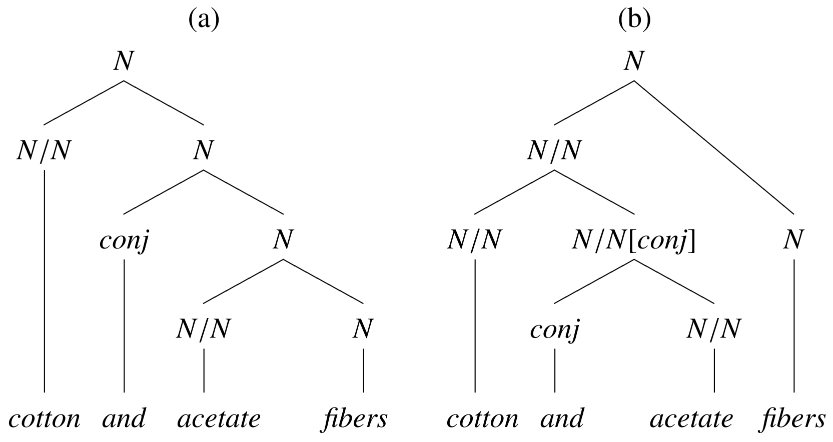


Figure 6.2: (a) Incorrect CCG derivation from Hockenmaier (2007) (b) The correct derivation

nodes are given the same category with the *conj* feature, resulting in a derivation that can be parsed with the binary coordination rules used in CCGbank:

$$conj\ X \Rightarrow X[conj] \tag{6.5}$$

$$X\ X[conj] \Rightarrow X \tag{6.6}$$

An example CCGbank derivation is shown in Figure 6.2(a), while the correct coordination derivation that we produce is in Figure 6.2(b). The process solves the problem we described in Section 2.5.1, where conjunctions are treated as modifiers instead of forming coordinate structures.

As a result of the conversion process, applications of the non-CCG rule in (2.9) are reduced from 1,378 to 145 cases. Most of the remaining cases included a conjunction that was already structured correctly. For example, *president and chief executive officer* has the correct implicit structure, even when unbracketed in the Penn Treebank. In NPs like these, the lowest spanning nodes were already correct and so no change was made during the conversion process. Removing this non-CCG rule from the grammar is an important step, as it allows it to be removed from the parser as well. This would reduce the ambiguity that the parser must resolve and the size of the charts that it has to build.

Some POS tags require special behaviour. Punctuation marks receive the same lexical category as their POS tag, and this should not be changed by the conversion process. Determiners and possessive pronouns are both usually given the lexical category *NP*[*nb*]/*N*, and this should not be altered either. In order to make this possible, the categories of other tokens often needed to be

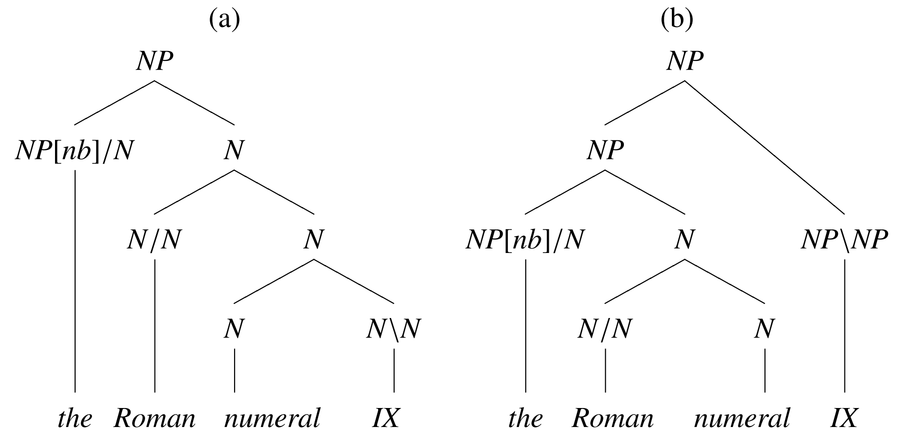


Figure 6.3: Special behaviour for a determiner (a) Original CCGbank tree (b) The corrected derivation

changed. The NP in Figure 6.3 is an example of this, before and after the conversion process has been applied. The sibling of the DT node, *Roman numeral*, is given the category *N*, rather than the dominating node's label. The *NP* formed by the determiner is made the head of the constituent, and its sibling, *IX*, is assigned the adjunct category *NP\NP*.

Finally, there are cases where the lowest spanning node covers a constituent that should not be changed. For example, in the following NP:

```
(NP
  (NML (NN lower) (NN court) )
  (JJ final) (NN ruling) )
```

with the original CCGbank lowest spanning node:

```
(N (N/N lower)
  (N (N/N court)
    (N (N/N final) (N ruling) ) ) )
```

the *(N final ruling)* constituent should not be altered.

It may seem trivial to process in this case, but consider a similarly structured NP: *lower court ruling that the U.S. can bar the use of*... Although this NP begins in the same way, it also contains a subordinate clause that could dominate a wide range of linguistic structure. Our minimalist approach avoids reanalysing complex constructions such as these, as it would unnecessarily replicate the much larger task of creating CCGbank in its entirety. As a result, we only flatten those constituents that partially overlap the NML or JJP bracket. The existing structure and dependencies

	#	%
Possessive	224	43.75
Apposition with DT/PRP$	87	16.99
Non-leaf assignment	66	12.89
Conjunction	35	6.84
False positive	26	5.08
Entity with internal brackets	23	4.49
Determiner	22	4.30
`NML`/`JJP` bracket is an error	12	2.34
Other	17	3.32
Total	512	100.00

Table 6.1: Manual analysis

of other constituents are retained. Note that we are still converting every `NML` and `JJP` bracket, as even in the subordinate clause example, only the structure around *lower court* needs to be altered.

6.3.4 Manual Annotation

A handful of problems that occurred during the conversion process were corrected manually. The first indicator of a problem was the presence of a possessive. This is unexpected, because possessives were already bracketed properly when CCGbank was originally created (Hockenmaier, 2003a, §3.6.4). Secondly, a non-flattened node should not be assigned a lexical category that it did not already have. This is because, as described in the previous section, a non-leaf node could dominate any kind of structure. Finally, we expect the lowest spanning node to cover only the `NML` or `JJP` bracket and one more constituent to the right. If it does not, because of unusual punctuation or an incorrect bracket, then it may be an error. In all these cases, which occur throughout the corpus, we manually analysed the derivation and corrected any errors that were observed. 512 cases were flagged by this approach, or 1.90% of the 26,993 brackets converted to CCG. Table 6.1 shows the causes of these problems.

The most common cause was possessives, as the conversion process highlighted many instances where the original CCGbank analysis was incorrect. An example of this error is in Figure 6.4(a), where the possessive doesn't take any arguments. Instead, *largest aid donor* incorrectly modifies the NP one word at a time. The correct derivation after manual analysis is in (b).

Possessives nested inside a left-branching structure also cause problems, as in Figure 6.5. Most possessives are assigned the lexical category $(NP[nb]/N)\backslash NP$, and combine with the preceding

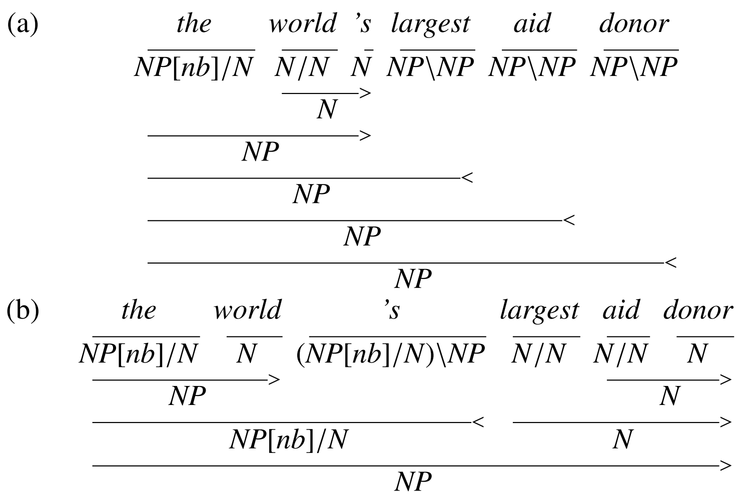

(a)
the world 's largest aid donor
NP[nb]/N N/N N NP\NP NP\NP NP\NP
N
NP
NP
NP
NP
(b)
the world 's largest aid donor
NP[nb]/N N (NP[nb]/N)\NP N/N N/N N
NP
N
NP[nb]/N
N
NP

Figure 6.4: (a) Original CCGbank derivation with a possessive (b) The corrected derivation

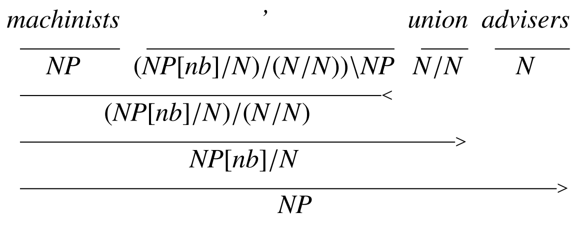


Figure 6.5: A possessive nested into left-branching structure

NP to form the standard determiner category. When the bare NP expected by this "determiner" is a nominal modifier instead, the possessive category must be altered accordingly. For example, the lexical category of the possessive in Figure 6.5 reflects the fact that *union* is acting as a modifier to *advisers*. This manually assigned lexical category which corrects the derivation did not previously exist in CCGbank.

Figure 6.6 shows the second-most common reason for manual analysis: an apposition inside an NP that also includes a determiner or possessive pronoun. As there is no punctuation on which to coordinate (which is how CCGbank treats most appositions) the best derivation we can obtain is to have *Victor Borge* modify the preceding NP.

The non-leaf assignment errors in Table 6.1 were cases like the *lower court ruling* example we gave in Section 6.3.3. There were also problems caused by conjunctions in some NPs, while others were simply false positives in the error detection algorithm. Entities with internal brackets,

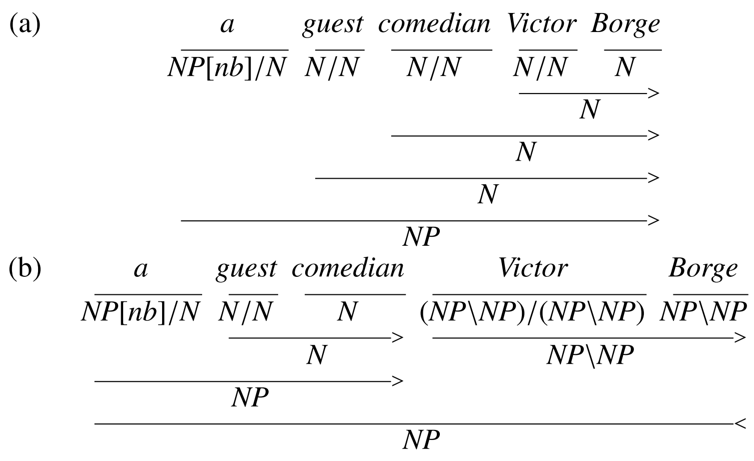

(a)
a guest comedian Victor Borge
NP[nb]/N N/N N/N N/N N
N
N
N
NP
(b)
a guest comedian Victor Borge
NP[nb]/N N/N N (NP\NP)/(NP\NP) NP\NP
N
NP\NP
NP
NP

Figure 6.6: (a) Original CCGbank derivation with apposition and DT (b) The corrected derivation

such as *Citibank (Madrid)*, were problematic because CCGbank often gives the opening bracket a functional category such as $(N\backslash N)/NP$, rather than the same lexical category as its POS tag. Similarly, some determiners in CCGbank receive a non-standard lexical category when they are part of an entity. For example, in *Budget Rent a Car*, the determiner is given a modifier category N/N.

6.3.5 Grammar Validation

The final step in the conversion process was to validate the corpus against the CCG grammar, first by those productions used in the existing CCGbank, and then against those actually licensed by CCG (with pre-existing ungrammaticalities removed). Sixteen errors were identified by this process and subsequently corrected by manual analysis.

In total, we have altered 12,475 of the 48,934 CCGbank sentences (25.5%) and 20,409 out of 1,046,136 dependencies (1.95%).

6.4 Initial Experiments

Having converted our NP annotations to CCGbank, we can now experiment with the data using the C&C CCG parser (Clark and Curran, 2007b). The parser is evaluated on predicate-argument dependencies derived from CCGbank, which we earlier described in Section 6.2.1. We train on Sections 02–21, develop on Section 00 and test on Section 23, as is the standard split in parsing experiments. All of the experiments below are evaluated on Section 00, except those in Section 6.6.2.

	PREC	RECALL	F-SCORE
Original	91.85	92.67	92.26
NP corrected	91.22	92.08	91.65

Table 6.2: Supertagging results

	PREC	RECALL	F-SCORE
Original	85.34	84.55	84.94
NP corrected	85.08	84.17	84.63
NP dependencies only	–	77.03	–

Table 6.3: Parsing results with gold-standard POS tags

6.4.1 Supertagging

Before we begin full parsing experiments, we evaluate on the supertagger alone. The supertagger, described in Section 2.5.2, is an important stage of the CCG parsing process and its results will affect performance in later experiments.

Table 6.2 shows that F-score has dropped by 0.61%. This is not surprising, as the conversion process has increased the ambiguity of supertags in NPs, and thus the difficulty of determining them. Previously, a bare NP could only have a sequence of N/N tags followed by a final N. There are now more complex possibilities, equal to the Catalan number of the length of the NP. This is an exponential increase in ambiguity.

In fact, there are even more potential supertags for words in an NP, even for tokens that are normally unambiguous. Possessives now have a wider ranger of categories, as we saw in Figure 6.5, and even regular nouns can have complex categories like $(NP\backslash NP)/(NP\backslash NP)$, as we saw in Figure 6.6. Unsurprisingly, the supertagger has difficulty dealing with this increased level of ambiguity.

6.4.2 Parsing Results

We use the normal-form parser model and report labelled precision, recall and F-score for all dependencies. Table 6.3 shows the results from comparing parser performance on our NP corrected version of the corpus to that on original CCGbank. The F-score drops by 0.31% in our new version of the corpus. However, this comparison is not entirely fair, as the original CCGbank test

	PREC	RECALL	F-SCORE	DIFF
Original	83.65	82.81	83.23	−1.71
NP corrected	83.31	82.33	82.82	−1.81
NP dependencies only	–	75.21	–	−1.82

Table 6.4: Parsing results with automatic POS tags

data does not include the NP structure that the NP corrected model is being evaluated on. We saw a similar drop in the Collins (2003) parsing experiments in Section 4.2, where the F-score for NML and JJP brackets was about 20% lower than the overall figure. We suspect that a similar effect is causing the drop in performance here.

Unfortunately, there are no explicit NML and JJP brackets in the CCG corpus. In order to evaluate the new NP structure, we take the same approach as the relabelling experiments of Section 4.2.1. The dependencies that were altered during the NP conversion process, for example the one in (6.2), can be marked in the test data, and we can then evaluate only the marked brackets. However, as before, this approach can only supply a recall figure, as the dependencies outputted by the parser are not marked. Table 6.3 shows that NP structure performance is about 7% lower than the overall recall, which compares favourably to the almost 30% recall drop in the Collins (2003) parsing experiments (see Table 4.1). Although of course, because of the different data, formalism and evaluation, as well as the gold-standard POS tags being used here, the results are not directly comparable.

Automatic POS and NER tags

We have also experimented with using automatically assigned POS tags, using the C&C POS tagger. The tagger achieves an accuracy figure of 96.49%, trained on Sections 02–21 and tested on Section 00. Table 6.4 shows that, unsurprisingly, performance is lower without the gold-standard data. The NP corrected model drops an additional 0.1% F-score over the original model, suggesting that POS tags are even more important for recovering internal NP structure.

6.5 Novel Features

Now that we have a baseline model from the NP corrected CCGbank, we can improve upon it. To this end, we will implement new features targeted at increasing performance on NP structure.

These novel features will be based on Named Entity Recognition (NER) and counts from the Web 1T corpus. We chose these features because of their contributions to the experiments in Chapter 5, where they were the best performing features that are not already included in the C&C parser.

6.5.1 Named Entity Recognition

Named entity recognition provides information that is particularly relevant for NP parsing, as the entities suggest a likely syntactic structure. For example, knowing that *Air Force* is an entity tells us that *Air Force contract* is a left-branching NP. The evaluation of the annotation tool's suggestion feature in Section 3.5 also demonstrated the usefulness of NER-based features. The BBN SIFT parser (Miller et al., 1998) is an existing system that makes use of NER information. There is also more recent work combining NER and parsing in the biomedical field. Lewin (2007) experiments with detecting base-NPs using NER information, while Buyko, Tomanek, and Hahn (2007) use a Conditional Random Field (CRF) to identify coordinate structure in biological named entities.

We draw NE tags from the BBN Pronoun Coreference and Entity Type Corpus (Weischedel and Brunstein, 2005), which annotates 28 different entity types. These include the standard person, location and organisation classes, as well as person descriptions (generally occupations), NORP (National, Other, Religious or Political groups), and works of art. Some classes also have finer-grained subtypes, although we use only the coarse tags in our experiments.

We have implemented a number of novel NER features in the C&C parser, most of which are generalisations of existing features that use head words and/or their POS tags. The original model used 489,196 features and this figure rises to 540,898 when our NER features are included. This is an increase of 51,702 or 11%.

Lexical

Our first addition is a lexical feature describing the NE tag of each token in the sentence together with its lexical category. The same feature already exists in the model for the word and its POS tag. This new feature, and all others that we describe here, are not active when the NE tag(s) are O, as there is no NER information from tokens that are not entities.

Local Tree

The next group of features is based on the local tree (a parent and two child nodes) formed by every grammar rule application. We add a feature where the rule being applied is combined with the head of the parent node's NE tag. For example, when joining two constituents[1]: $\langle$*five*, CD, **CARD**, $N/N\rangle$ and $\langle$*builders*, NNS, **PER_DESC**, $N\rangle$, the feature would be:

$$N \rightarrow N/N\ N + \texttt{PER\_DESC}$$

as the head of the constituent is `builders`. This feature is based on the following pre-existing features that use the head word and its POS tag instead of the NE tag:

$$N \rightarrow N/N\ N + \textit{builders}$$
$$N \rightarrow N/N\ N + \texttt{NNS}$$

The local tree feature type also combines the grammar rule with the child nodes. There are already features in the model describing each combination of the children's head words and POS tags, which we extend to include combinations with the NE tags. Using the same example as above, the pre-existing features would be:

$$N \rightarrow N/N\ N + \textit{five} + \textit{builders}$$
$$N \rightarrow N/N\ N + \textit{five} + \texttt{NNS}$$
$$N \rightarrow N/N\ N + \texttt{CD} + \textit{builders}$$
$$N \rightarrow N/N\ N + \texttt{CD} + \texttt{NNS}$$

to which we add five new features:

$$N \rightarrow N/N\ N + \textit{five} + \texttt{PER\_DESC}$$
$$N \rightarrow N/N\ N + \texttt{CARD} + \textit{builders}$$
$$N \rightarrow N/N\ N + \texttt{CD} + \texttt{PER\_DESC}$$
$$N \rightarrow N/N\ N + \texttt{CARD} + \texttt{NNS}$$
$$N \rightarrow N/N\ N + \texttt{CARD} + \texttt{PER\_DESC}$$

Entity Spanning

Another feature group is based on the NE spanning categories. We identify constituents that dominate tokens that all have the same NE tag, as these nodes will not cause a "crossing bracket" with the named entity. For example, the constituent *Force contract*, in the NP *Air Force contract*, spans two different NE tags, and should be penalised by the model. *Air Force*, on the other hand, only spans **ORG** tags, and should be preferred accordingly. We also take into account whether the

[1]These 4-tuples are the node's head and the head's POS tag, NE tag and lexical category.

constituent spans the *entire* named entity. Combining these nodes with others of different NE tags should *not* be penalised by the model, as the NE must combine with the rest of the sentence at some point. This feature group has no equivalent in the pre-existing C&C model.

There are two NE spanning features for each grammar rule instantiation: one for the parent node and a second for the child nodes. These features also describe whether or not the nodes span entire entities. There are two possibilities for the parent node, it does or it doesn't, and four possibilities for the child nodes, depending on whether neither, the left, the right or both nodes span the entire NE.

Consider the example NP: *Royal Air Force contract*. When *Air* and *Force* combine, the parent feature would be:

$$N \rightarrow N/N\ N + \text{PARTIAL} + \textbf{ORG}$$

and when *Royal* and *Air Force* subsequently combine, the parent feature would be:

$$N \rightarrow N/N\ N + \text{ENTIRE} + \textbf{ORG}$$

We expect that both of these features would be assigned positive weights by the model, with the latter given the greater magnitude of the two.

On the other hand, if *Air Force* and *contract* were incorrectly joined, then the parent feature would be:

$$N \rightarrow N/N\ N + \text{PARTIAL} + \textbf{X}$$

where **X** indicates that disparate NE tags have been combined. Thus, no **X** constituents can span an entire entity. The presence of **X** means that this latter feature should have a negative weight, resulting in an accordingly lower probability for the constituent. Note that an **X** would also be in the parent feature when *Royal Air Force* and *contract* are combined, even though this is the correct structure. This will be moderated by the child features as described below.

For the child features, joining *Royal* and *Air Force* would result in:

$$N \rightarrow N/N\ N + \text{NEITHER} + \textbf{ORG} + \textbf{ORG}$$

as neither child spans the entire entity. This feature should still receive a positive weight. When *Royal Air Force* and *contract* form the entire NP node, then (assuming that there are more **O** tags to the right) the child feature will be:

$$N \rightarrow N/N\ N + \text{LEFT} + \textbf{ORG} + \textbf{O}$$

The left child does span the entire entity in this case, and so this feature should be given a sufficiently high weight to overcome the negative weight that may be assigned by the parent feature with its `X`. Incorrect constituents will still have low probability, such as *Air* combined with *Force contract* which produces the following feature:

$$N \rightarrow N/N\ N + \text{NEITHER} + \texttt{ORG} + \texttt{X}$$

because neither node spans an entire entity and `X` is present.

Entity Spanning without Rules

It may be argued that the spanning feature group does not need to describe the grammar rule being applied, as only the NE tags are relevant. We experimented with this by adding back-off features, namely the spanning categories without rules. These features work in the same way as the previous feature group, only without conditioning on the grammar rule. For example, the parent feature for *Air Force* would be:

$$\text{ENTIRE} + \texttt{ORG}$$

and the child feature combining *Air Force* and *contract* would be:

$$\text{LEFT} + \texttt{ORG} + \texttt{O}$$

Distance

The final feature group is based on the distance features that the parser currently uses. They give a measure of the distance between the heads of the constituents being combined. Individual features exist that count the number of words, verbs and punctuation marks in the intervening tokens. These features already include the head word of the combined constituent and generalise to its POS tag. We add another generalisation to the NER tag.

For example, if *Air* were combining with *Force contract*, then there would be one word between the heads (*Force*), and the following word distance features would be created:

$$N \rightarrow N/N\ N + \textit{contract} + 1$$
$$N \rightarrow N/N\ N + \texttt{NN} + 1$$

Our additional NER-based feature would be:

$$N \rightarrow N/N\ N + \texttt{O} + 1$$

The same is done analogously for the verb and punctuation distance features.

6.5.2 Web 1T

We saw in Section 4.6 that the primary cause of NP structure errors was a lack of lexical information. There is no reason why we should not suffer from the same problem when parsing with CCG, as the text in the training data remains the same. In these cases where the parser has not seen the lexical items, the bigram count from an unannotated corpus will serve as an alternative source of information on which to base a parsing decision. We draw these bigram counts from Google's Web 1T corpus (Brants and Franz, 2006), as was done previously in the NP Bracketing experiments.

The new feature is implemented as the bigram count of the head words of the two constituents being combined. This is made slightly more complicated as the C&C parser uses binary features only and thus we must discretise the bigram counts, as was done previously in Chapter 5. However in this case, adding the multitude of features produced by the Fayyad and Irani (1993) algorithm would be problematic, because of complications in the current C&C parser architecture and also because of the increased difficulty in optimising the resulting Maximum Entropy model. Instead, we discretise based on the order of magnitude of the count. This simpler method still describes the frequency of the bigram, while only requiring a small, predeterminable number of features.

We also experimented with using the χ^2 measure described and used in Chapter 5. Again this results in a real-valued figure, which we discretise to a number of binary features using the order of magnitude technique. Using the Web 1T corpus increases the total feature count to 545,777. This is an additional 4,879 features, or 1% of the original model's count.

6.6 Results

Table 6.5 shows the results achieved with our novel features on the NP corrected CCGbank. We report the overall dependency figures, as well as the NP recall measure described in Section 6.4.2.

The second and third rows show that the χ^2 version of the Web 1T features outperforms the raw counts. An 0.94% F-score increase is achieved over the initial model. The rest of Table 6.5 presents a subtractive analysis of individual feature groups, using the χ^2 version of the Web 1T features. The figures show that none of the features provide a tremendous improvement on their own, but together, the increase is quite substantial. The Web 1T features provide the biggest gain, although it appears that this is because they are informative in a different way to all of the NER features, whose improvements overlap. This can be seen by comparing the last two rows of the

	PREC	RECALL	F-SCORE	NP RECALL
No new features	85.08	84.17	84.63	77.03
All features, Web 1T raw counts	85.95	85.03	85.49	76.75
All features, Web 1T χ^2	**86.05**	**85.09**	**85.57**	**77.45**
−Lexical	86.04	85.09	85.56	*77.73*
−Local tree	85.97	85.02	85.49	*78.29*
−Entity spanning	85.87	84.97	85.42	77.03
−Entity spanning without rules	86.02	85.07	85.54	*78.01*
−Distance	85.91	85.00	85.45	76.75
−Web 1T χ^2	85.79	84.59	85.34	*77.73*
Web 1T χ^2, no new NER features	85.60	84.67	85.13	76.89

Table 6.5: Feature analysis on NP corrected CCGbank

table, where using only the Web 1T χ^2 features is inferior to using only the NER features (i.e. subtracting Web 1T).

The figures for NP recall are calculated over only 714 NP dependencies, and as such, the changes are relatively minor in all cases. The 0.42% improvement from using all features and Web 1T χ^2 only recovers three more dependencies than the no new features model. This is very unexpected, as the novel features are specifically intended to improve NP performance. The subtractive analysis actually suggests that some features are harmful for NP performance, although the small changes mean that the features are merely indifferent. Also, because we can only measure recall and not precision, the NP figures do not provide the most accurate view of the performance changes. Because of this, we cannot draw too many conclusions from them. It does appear that our novel features have actually been successful at improving parser performance as a whole, rather than the NP dependencies we were aiming for. This is a positive result for the parsing field in general, rather than NPs in particular.

Considering the negligible changes in NP recall, the best performing model according to the subtractive analysis uses all NER feature groups and the Web 1T features with χ^2 scores.

Having determined the optimal set of features, we now compare performance on the NP corrected version of CCGbank to that on the original. Table 6.6 shows the results, along with the improvements from using all of the feature groups and the χ^2 Web 1T counts. We do not show NP recall, since there are no NP dependencies marked in the original corpus.

Performance has increased by 0.71% F-score on the original CCGbank, and by 0.94% F-score on the NP corrected version. This difference is not surprising, as the incorrect right-branching

	PREC	RECALL	F-SCORE
Original	85.34	84.55	84.94
Original, with novel features	86.10	85.21	85.65
NP corrected	85.08	84.17	84.63
NP corrected, with novel features	86.05	85.09	85.57

Table 6.6: Comparing the NP corrected CCGbank to the original

	PREC	RECALL	F-SCORE
Original	83.65	82.81	83.23
Original, with NER features	83.99	83.13	83.56
Original, with all features	84.07	83.21	83.64
NP corrected	83.31	82.33	82.82
NP corrected, with NER features	83.71	82.76	83.23
NP corrected, with all features	83.84	82.89	83.36

Table 6.7: Results using automatically assigned POS and NER tags

NPs would introduce noise to our novel features, causing them to be less effective. For example, *Air Force contract* is annotated as right-branching in the original CCGbank. The NER features will thus appear inconsistent in NPs like this, as the gold-standard constituent *Force contract* crosses the NE boundary. Likewise, the Web 1T features will have a high count for *Air Force*, although this is not a gold-standard constituent. In the NP corrected version of the corpus, both of these inconsistencies will disappear, allowing the model to add more weight to the features. These results are further evidence of the effectiveness of our novel features.

6.6.1 Automatic POS and NER Tags

As in Section 6.4.2, we experimented with automatically assigning POS tags. In light of the features we have added, automatic NER tags were also experimented with. We use the C&C NER tagger (Curran and Clark, 2003), which attains 84.45% F-score on all non-O classes, with precision being 78.35% and recall 91.57%. These figures are achieved on Section 00, after training on Sections 02–21, and use automatically generated POS tags.

The results are shown in Table 6.7. The automatic POS and NER tags have caused the all-features models to drop by 2.21% and 2.01% F-score on the NP corrected and original versions of CCGbank respectively. This is a larger increase than the 1.81% and 1.71% drops when using none of

	PREC	RECALL	F-SCORE	NP RECALL
Original	85.94	85.25	85.59	–
Original, with novel features	86.57	85.90	86.23	–
NP corrected	85.59	84.95	85.27	76.10
NP corrected, with novel features	86.26	85.64	85.95	78.12

Table 6.8: Final test set results

the novel features. The gap between the NP corrected and original corpora has also increased from 0.1% to 0.2% F-score. These figures show that the novel features in particular become less effective when parsing raw text. This is to be expected, as the NER features will be less reliable when the NEs are not gold-standard.

6.6.2 Test Set Results

Our final set of figures presents results on Section 23 of the corpus. These are shown in Table 6.8. As with our development results, the original CCGbank F-score is higher than the NP corrected version. And also as before, the performance increase from using the novel features is slightly greater on the NP corrected corpus, although the difference here is very small. These test set figures are more encouraging for NP recall, showing an increase of 2.02% or 21 additional dependencies being recovered.

We perform statistical significance tests on these results, using the same randomised shuffling technique earlier described in Section 5.6.1. We can only compare models trained on the same version of CCGbank, as the dependencies being evaluated differ between the two versions of the corpora. Comparing the original CCGbank model to that using novel features (the first two rows of Table 6.8), we calculate a p-value of 0.0001 for precision, recall and F-score. This is the smallest p-value attainable for the 10,000 iterations performed. The same p-values are achieved when comparing the two models trained on the NP corrected version of CCGbank. For NP recall, we calculate a p-value of 0.0092, demonstrating that this 2.02% increase is a statistically significant increase in performance.

	PREC	RECALL	F-SCORE
Original, gold parses	86.86	81.61	84.15
NP corrected, gold parses	87.97	82.54	85.17

Table 6.9: Upper bound on performance for the DepBank evaluation

6.7 DepBank Evaluation

One problem with the evaluation in the previous section, is that the original CCGbank is not expected to recover internal NP structure, making its task easier and inflating its performance. To remove this variable, we carry out a second, formalism and annotator-independent evaluation against the Briscoe and Carroll (2006) reannotation of DepBank (King et al., 2003), as described in Clark and Curran (2007a). Parser output is made similar to the Grammatical Relations (GRs) of the Briscoe and Carroll (2006) data, however, the conversion remains complex. Clark and Curran (2007a) report an upper bound on performance, using gold-standard CCGbank dependencies, of 84.76% F-score.

This evaluation is particularly relevant for our full parsing of NPs, as the Briscoe and Carroll (2006) corpus *has* been annotated for internal NP structure. With our new version of CCGbank, the parser will be able to recover these GRs correctly, whereas before this was unlikely. The evaluation is particularly important, as it allows a comparison between the same GRs before *and* after the NP structure has been corrected. In the earlier experiments with the C&C and Collins (2003) parsers, we could only evaluate NP structure after making alterations to the corpus. Thus, this DepBank evaluation will best show the effect that NP structure has on parsing performance.

Table 6.9 shows the figures achieved using the gold-standard CCGbank derivations. In the NP corrected version of the corpus, performance has increased by 1.02% F-score. This decisive result is a reversal of those we saw in Section 6.6 and in the Collins (2003) experiments. It demonstrates that correct NP structure improves parsing performance, rather than decreasing it, and that the alterations we have made to CCGbank represent NP structure in a measurably better way compared to the original corpus. Because of this increase to the upper bound of performance, we are now even closer to a true formalism-independent evaluation. Note that our original CCGbank recall figure is slightly different to that reported by Clark and Curran (2007a). The reason for this is still unresolved.

We now move to evaluating the C&C parser itself and the improvement gained by the

	PREC	RECALL	F-SCORE
Original	82.57	81.29	81.92
NP corrected	83.53	82.15	82.84
Original, novel features	83.22	81.95	82.58
NP corrected, novel features	84.27	83.05	83.65
Original, novel features, auto. POS and NER	83.01	81.63	82.31
NP corrected, novel features, auto. POS and NER	84.13	82.81	83.46

Table 6.10: DepBank evaluation results

novel features in Section 6.5. Table 6.10 shows the results with no new features; the best model from Section 6.6, which uses all of the novel features; and the effect of automatic POS and NER tags on the all-features model. As we saw in the gold-standard results, the NP corrected corpus performs better than the original CCGbank in all cases. This further validates the superiority of the NP corrected corpus.

In the models without any new features, training on the corrected NP structure outperforms the original CCGbank by 0.92% F-score. Adding in our novel features results in a further improvement of 0.81% F-score. Thus, our work has increased the parser's performance by 1.73% F-score in total. These results demonstrate that our NP data and novel features have both been very successful.

Furthermore, the performance increase of 0.81% on the NP corrected corpus is more than the 0.66% increase on the original. This is the same result we saw in Section 6.6 and again validates our expectation that NER and Web 1T features are particularly helpful for NP structure. The figures for automatic POS and NER tags show the same trend, providing a greater increase on the NP corrected corpus. These increases are, as we would expect, not as large as when using gold-standard tags. However, the novel features still provide a noticeable gain. We also note that the drop in performance when using automatic POS and NER tags is smaller in this evaluation, compared to that experienced in Section 6.6.1. The DepBank evaluation is less sensitive to the errors introduced by using automatic taggers.

6.8 Error Analysis

The results achieved in this chapter have been excellent. The CCG parser has performed extremely well at recovering NP structure and there was a noticeable improvement from using the NP corrected corpus. However, it is still informative to perform an error analysis in order to determine

ERROR	#	%
Modifier attachment	64	39.02
Mislabelling	32	19.51
Named Entities	31	18.90
Apposition with DT	12	7.32
Possessives	10	6.10
Conjunctions	5	3.05
Other	10	6.10
Total	164	100.00

Table 6.11: Error analysis

where future improvements may lie. Accordingly, we manually examined and categorised the NP errors being made by the parser on the development data in the original model, i.e. before including the novel features in Section 6.5. The results are in Table 6.11.

As with the Collins (2003) parser analysis in Table 4.7, about 40% of the errors are caused by a general modifier attachment problem. Section 6.8.1 will analyse these errors in more detail.

The next-most common cause of errors (19.51%) is the supertags assigned to the lexical items. In these cases, the dependency created is correct, but the label is not. For example, a pair of modifiers might be assigned the categories N/N and N instead of $(N/N)/(N/N)$ and N/N. As we mentioned in Section 6.4.1, the NP corrected corpus causes an increase in NP supertag ambiguity and so it is not surprising to see a number of these errors.

Although named entities are present in a significant proportion of the errors (18.90%), we noted that only 18 of the cases (10.98% of the total errors) could conceivably be corrected by the NER information. This is because when the error is internal to the entity, the NE tags remain uninformative. For example, the fact that *Consolidated Gold Fields PLC* is marked as an organisation does not disambiguate between *Consolidated* forming a dependency with *Gold*, *Fields* or *PLC*. This may explain why the NER features did not increase NP performance overmuch. The overall increase from these features is still understandable, as the NE tags on *Consolidated Gold Fields PLC* do effectively demarcate the NP's boundaries.

The errors caused by appositions and possessives are the same problems that were manually analysed during the conversion process in Section 6.3.4. The unusual supertags that were assigned in those analyses, such as $(NP\backslash NP)/(NP\backslash NP)$ for *Victor* in *the guest comedian Victor Borge* are not surprisingly, difficult for the parser to recover. Similarly, possessives with a non-standard

	CORRECT	INCORRECT	TOTAL
IN TRAINING	346	18	364
NOT IN TRAINING	204	146	350
TOTAL	550	164	714

Table 6.12: The effect of lexical information on NP accuracy

	CORRECT	INCORRECT	TOTAL
IN TRAINING	20,183	2,174	22,357
NOT IN TRAINING	13,119	4,088	17,207
TOTAL	33,302	6,262	39,564

Table 6.13: The effect of lexical information on overall accuracy

supertag, such as $(NP[nb]/N)/(N/N))\backslash NP$ in Figure 6.5, were problematic.

It is interesting to note that the mislabelling, apposition and possessive errors could all be improved with a better supertagger analysis. We have implemented novel features for the parser, but it would also be feasible to apply these kinds of features to the supertagger. Section 7.2 will discuss this possibility further.

6.8.1 The Effect of NP Lexicalisation

Table 6.12 displays a deeper analysis of the NP lexicalisation problem. It details the number of NP dependencies that were recovered correctly or incorrectly by the parser and whether those dependencies are present in the training data. Note that the entire NP, e.g. *lung cancer deaths*, is not necessarily seen in the training data, only the specific labelled dependency, e.g. $\langle cancer, (N/N)/(N/N), 1, lung, - \rangle$. The figures show that a dependency that occurs in the training data has a 95% ($\frac{346}{364}$) chance of being recovered correctly. On the other hand, a dependency that does not occur in the training data has only a 58% ($\frac{204}{350}$) chance of being correct. Assuming that the surrounding decisions have been made correctly, the baseline for a three word NP is 50%, (the two choices being left or right-branching), and so this latter figure seems quite poor.

Of course, lexicalisation can be informative for parsing and so these proportions may seem reasonable. To demonstrate that they are relatively anomalous, Table 6.13 displays the same figures for *all* dependencies. These are recall figures only and so remain comparable to the previous NP only table. In this case, only 90% ($\frac{20183}{22357}$) of dependencies that occur in the training data are recovered correctly, while 76% ($\frac{13119}{17207}$) of dependencies that the parser has not been trained on are also correct. That is, NP structure performance is higher on dependencies that exist in the training data, while still being lower on unseen dependencies.

This analysis gives a excellent indication of the difficulty involved in parsing NPs. Consider an ambiguous NP, *brain stem cells*, which may describe stem cells from the brain, or cells from the brain stem. The 95% result from Table 6.12 suggests that NPs like this are quite rare, and that most NPs will have only a single attested bracketing. But despite this, the NP parsing problem remains more difficult than parsing in general. The productivity of NPs means that many will occur only once, and even though just a single example is enough to inform all further decisions, it will not help if the NP has never been seen. These results support those found in Chapter 4, demonstrating that poor lexicalisation of NP structure is a parser-independent problem.

Considering all this, the obvious question is: why are the Web 1T features not a panacea for the problem? Bikel (2004, §7.4) suggests a possible reason: the bilexical statistics that we are including as features here are not actually very useful to the Collins (2003) parser. It is the lexico-structural dependencies, i.e. those that describe how lexical items form constituents, that are important.

Another possibility is that the bigram counts do not discriminate effectively. For a node high in the tree, the two head words may be quite distant in the stream of words, and actually hardly ever be seen next to one another. Thus, they will have a low bigram count even in correct structures. Returning to an example from Chapter 1, there is a dependency between *eat* and *spoon* in the sentence *I eat the cake with the spoon*, and the semantic relationship between the pair of words is clear. However, their Web 1T bigram count is only 226. One possibility, that may reduce the negative effect of dependencies between non-contiguous words, is to include the grammar rule in the feature. The rule will be indicative of height in the tree, allowing the model to learn that low counts are not necessarily a bad thing in these cases. However, it is hard to tell if this is true. It may be that removing the rule, as was successful for the NER spanning features, would also improve performance here, as it would allow the model to generalise more. Modelling the height of the constituent directly may be a better alternative, although it would be harder to implement in the parser.

Finally, it may be that the Maximum Entropy model is depending too heavily on the lexical features. As we have seen, they are extremely effective when available, however when they are not, it may be better to have more weight placed upon the Web 1T features. It remains to be seen how best to incorporate this unannotated information into a supervised parsing model.

6.9 Summary

In this chapter, we experimented with NP parsing using Combinatory Categorial Grammar. The data from Chapter 3 was converted to this different formalism and used to correct important errors which were systematic in CCGbank. We have also implemented a number of novel features in the parser: using Named Entity Recognition, which is a source of semantic information unused by most parsers; and also counts from the Web 1T corpus, as a replacement for lexical information on unknown NPs.

Our experimental results have shown that this more accurate representation of CCGbank's NP structure increases parser performance. The DepBank evaluation in particular showed that correcting CCGbank's NP structure results in a large performance gain. The use of our novel features then allowed these figures to be raised even higher. NP structure is now recoverable by the C&C parser and its overall performance has been increased by 1.73% F-score.

Chapter 7

Future Work

This thesis is the first to create and make use of a large-scale corpus of NP annotations. Our experiments with this new data have set a high benchmark for NP parsing. In many cases, there has been no previous work or state-of-the-art result to compare to, only experiments on a data set that is limited in scale and coverage. Our NP Bracketing experiments in Chapter 5 for example, set a new bar for what can be achieved, and demonstrate the applicability of supervised methods. There is now a tremendous amount of future work that can be performed on the subject of NP parsing.

7.1 NP Annotation

In Chapter 3, we extended the Penn Treebank with NP annotations. For the first time, this widely-used corpus could train parsers to recover NP structure. We are aware of only one other corpus that has been annotated with a large volume of NP structure: the Biomedical Information Extraction Project (Kulick et al., 2004). Because these are the first NP annotation schemes, it seems probable that they can be improved. We suggest some potential refinements here.

7.1.1 More Detailed NP Structure Categorisation

In our work, and in the Biomedical Project, there are only two possible structures: left-branching and right-branching. Other possibilities have been noted by different researchers, which we also came across during the annotation process.

Flat NPs

Firstly, there are NPs that are neither left nor right-branching, but exhibit only a flat, monolithic structure. Entities such as *John A. Smith* and *International Business Machines* are examples of this. In these cases, there is no real head-modifier relationship, *John* is not modifying *Smith*, but the tokens taken together still convey a meaning. McInnes, Pedersen, and Pakhomov (2007) recognise monolithic NPs in their annotation of medical terms, giving the example *serous otitus media*.

Perhaps the easiest way to annotate these flat NPs is to change their tokenisation, joining them together as a single token. This would simplify any structural problems and let a parser (or any NLP system) treat the entity as the single object that it is. This would not be a very practical approach to the problem, as much important lexical information would be lost. A better annotation scheme would be to add a marker to the relevant bracket in the corpus, in the same way that semantic markers (CLR, PRD, etc) are used:

```
(NP-FLAT (NNP John) (NNP A.) (NNP Smith) )

(NP
  (NML-FLAT (NNP John) (NNP A.) (NNP Smith) )
  (NNS apples) )
```

Another possibility is to include flat NPs into the parsing algorithm itself. The monolithic structure could be inserted at the appropriate level in the chart, rather than being formed as a constituent via a combination of lexical items. These two structures could then probabilistically compete, with the parser choosing the most likely option. However, this could introduce a problem similar to the bias in PCFGs, where smaller derivations are more likely because they involve multiplying fewer probabilities. The monolithic structure would likewise be made up of fewer probabilities. Other problems would be deciding how to apply the feature set to multiple words together, and how to determine which NPs are flat in the first place. If these issues could be resolved, then the resulting model would be able to statistically decide between left, right, and flat structures.

Indeterminate NPs

The second additional category is semantically indeterminate NPs, which we noted the presence of in Section 3.1. These NPs can be thought of as both left and right-branching, i.e. a dependency should exist between all word pairs. Lauer (1995b) found that 35 out of the 279 non-error NPs in his data set fitted this category, for example *city sewerage systems* and *government policy decisions*. It is the *government policy* in question in the latter example, but also *policy* *decisions* and *government decisions*, resulting in all three possible dependencies. In the same way as flat NPs, a marker could be added to the bracket to denote indeterminate NPs:

```
(NP-IND (NN government) (NN policy) (NNS decisions) )

(NP
  (NML-IND (NN government) (NN policy) (NNS decisions) )
  (NN report) )
```

Note that some NPs may appear to be indeterminate, but can actually be resolved. For example, in *American President George Bush*, *George Bush* is *American*, and the *President*, and the *American President*. However, the first meaning in this list is not intended by the utterance. *Bush*'s nationality is not relevant in the document, and so we argue that that the right-branching dependency should not be created. This NP should be annotated as left-branching.

Marcus, Santorini, and Marcinkiewicz (1993) make some mention of indeterminate NPs, calling them *permanent predictable ambiguities*, a term they ascribe to Martin Kay. The example *a boatload of warriors blown ashore* is given, which is similar to those in Hindle and Rooth (1993). In Section 3.1.1 we described how both meanings of the prepositional phrase attachment are true in cases like this: the *boatload* was *blown ashore*, and so were the *warriors*. Marcus et al. (1994) describe the `*PPA*` trace used in the Penn Treebank, which is applied to these permanent predictable ambiguities, or as we have called them, indeterminates. However `*PPA*` is also applied to cases of general ambiguity (those described in the following paragraphs), whereas we would separate the two.

Ambiguous NPs

The final category that we suggest is for ambiguous NPs. These NPs do have a left or right-branching structure, however the annotator has no hope of determining which is correct. This may be because of technical jargon, e.g. *senior subordinated debentures*, or simply an ambiguity that cannot be resolved by the given context, as in the often cited PP-attachment example: *I saw the man with the telescope*. In these cases, there is a definite correct answer. The man either has a telescope, or a telescope is being used to do the seeing, but not both.[1] This differentiates these ambiguous cases from indeterminate NPs, where *both* readings are true.

[1]In theory, the telescope could be with the man *and* used to do the seeing, but we will ignore this rather pathological possibility.

The Penn Treebank's X constituent exists for when the correct category is unknown or uncertain, demonstrating that this problem occurs in the Treebank. However, we expect that the consistent use of this label is difficult at best. In any annotation task there will be hard-to-bracket cases, but drawing a line between those that are unresolvable and those that are merely complex would be up to individual annotators, whose opinions could vary greatly. In our experience, it is better to simply make a decision between left and right-branching. Accordingly, Section A.1.2 of our guidelines instructs annotators to leave an NP flat when they are unsure. Having this default strategy is one way to manage this problem, similar to high PP attachment in the Penn Treebank (Bies et al., 1995, §5.2.1) and in the Redwoods Treebank (Oepen et al., 2002).

The Frequency of these Additional Categories

Annotating for each of these flat, indeterminate and ambiguous NPs would require a further pass through the corpus, which would be a significant amount of work. From a pragmatic point of view, it may be better to leave them as is, as they comprise such a small proportion of all NPs. We can present no gold-standard figures for the Penn Treebank, as the annotation of these additional NP structure categories has not been performed as yet. However, considering that the annotator only marked 915 of the 60,959 inspected NPs as difficult (1.50%), we suggest that almost all NPs can be assigned to left or right-branching classes. From our experience annotating, we estimate that approximately 5% of NPs do not fit into one of these major categories. Indeterminate NPs would be the least part of these, and ambiguous NPs (relating to financial jargon) the most.

McInnes, Pedersen, and Pakhomov (2007) found that flat NPs comprised 10.3% of their corpus, however another category of NPs that they define, *non-branching*, appears to be equivalent to right-branching NPs. Also, one of the flat examples given, *difficulty finding words*, does not seem to be an NP. For these reasons, a comparison between our corpus and theirs may not be reliable.

In Lauer's data set, 12.54% of NPs are indeterminate, however we suspect that many of these cases could fit into a left or right-branching category. The same logic we applied to *American President George Bush* could be used for some of Lauer's indeterminate NPs, such as *college basketball players*. The NP is unlikely to be stressing that the players are college students, rather that they are playing in an official college basketball league. Although the lack of context adds some confusion, we suspect this is actually a left-branching NP.

Flat, indeterminate and ambiguous NPs are interesting problems, but they are only a small part of the larger NP parsing task.

7.1.2 Improving Consistency

In Section 4.6, we saw that a number of errors occurred where the parser mislabelled an NP as NML, or vice versa (and the same with JJP and ADJP). Section 3.3.2 earlier pointed out that the extended Penn Treebank retained some inconsistencies, which could only be corrected by altering the original Penn Treebank brackets.

It seems clear then, that our principle of retaining all pre-existing brackets has reduced performance somewhat. This does not mean that our approach was flawed, as it was necessary to allow a direct comparison between the old corpus and the new. However, now that this has been performed, we have little doubt that changing some of the original Penn Treebank brackets would create a better corpus. Machine learning techniques would be more effective, and performance of statistical models would be higher, if the original Penn Treebank annotations and ours were made more consistent.

Performing manual reanalysis with another annotation phase would provide guaranteed coverage of all those cases that need to be made consistent. However, sufficient consistency is perhaps achievable using an automated approach. Hogan (2007) uses such a technique, automatically detecting and correcting NPs in the Penn Treebank that were bracketed differently to the guidelines' (Bies et al., 1995) specifications. For example, coordinations with nouns only should not receive internal structure, while coordinations with shared non-nominal modifiers should. This pattern matching approach increases Treebank consistency and parser performance as a result.

7.1.3 Head Marking

Another issue raised during the annotation process was that an NP's head may not necessarily be the right-most token. For example, the head of *IBM Australia* (the Australian division of the company) is *IBM*, and not *Australia*. This problem may create inaccurate probability estimates. For example, *bought IBM Australia* is a likely utterance. However, the bigram of the verb and the head of the NP, *bought Australia*, is unlikely (one cannot buy a country). Identifying the correct head would result in a much more likely bigram *bought IBM*, and thus a better probabilistic model. The hand-written head-finding rules applied by a parser do not correctly deal with cases like this.

Magerman (1994, §9.2), whose original head-finding table has been a basis for Collins (2003) and in turn Hockenmaier (2003a), also objects to using head-finding rules. He calls them a *"blatant violation"* of his goal to minimise manual knowledge engineering. The principle of a statistical parser should be to extract its knowledge from a treebank, rather than defining it explicitly.

Thus, a better solution for NPs and treebanks in general, would be to annotate the head of each constituent.

Many different approaches could be undertaken for this task, e.g. marking syntactic vs semantic heads, or annotating the DP analysis (Abney, 1987). However, doing this for every NP in the Penn Treebank would be a tremendous task in itself, and annotating every constituent would be exponentially greater. Considering the general effectiveness of the head-finding rules and their minor impact on performance (as we found in Section 4.2.2), a solution for this problem may not be available for some time to come.

7.1.4 Apposition

Appositions are a very common linguistic construction in English NPs. They have been used in areas such as Information Extraction (Sudo, Sekine, and Grishman, 2003) and Question Answering systems (Moldovan et al., 2003). However, there is little work on automatically identifying them. Researchers have typically used simple patterns for this task, although the accuracy of this method has not been determined.

Apposition in the Penn Treebank is extremely common, as shown in the (slightly edited) example below:

```
(NP-SBJ
  (NP (NNP Darrell) (NNP Phillips) )
  (, ,)
  (NP (NN vice) (NN president) ))
```

This annotation does not describe in any way that *Darrell Phillips* is the *vice president*. As a result, a parser will not be able to identify the correct semantic structure. A better interpretation would be to treat the NP as a multi-headed construct, rather than a single entity (or even worse, as two different entities).

Appositions are particularly problematic in CCGbank, where they are treated in exactly the same manner as conjunctions. The derivation in Figure 7.1 shows this. By annotating apposition explicitly, using an additional bracket, these problems can be solved.

```
(NP-SBJ
  (APP
    (NP (NNP Darrell) (NNP Phillips) )
    (, ,)
    (NP (NN vice) (NN president) )))
```

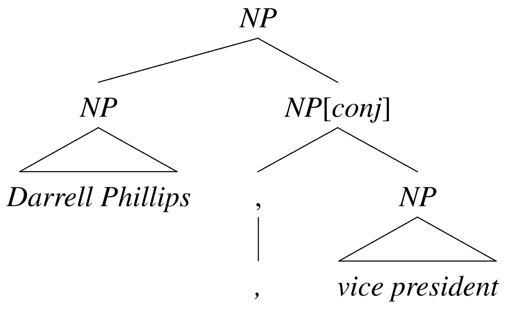


Figure 7.1: Apposition treated as conjunction in CCGbank

7.2 NP Parsing

We experimented with three different parsing architectures in this thesis: the Collins (2003) model; NP Bracketing; and with the CCG formalism. In each of these cases, our experiments can now be built on in an attempt to find further improvement.

Our experiments in Chapter 4 highlighted the difficulty of parsing NPs. However, the information sources that were successfully applied in later chapters could be used to good effect in a Collins-style parser. In particular, the possibility exists to include NER and Web 1T features as additional probability distributions in the model. It would be interesting to observe the change in performance compared to what was seen using the C&C parser.

Our NP Bracketing experiments were the first on large-scale data sets. We hope that other researchers will now make use of our data to carry out further NP Bracketing research. In particular, being able to construct supervised models opens the field for new features, models and representations. Other machine learning algorithms, such as Support Vector Machines (SVM), could be experimented with to good effect. There is also always the possibility that new features will be devised and implemented, and our data means that this kind of exploration is now possible.

One specific point of future improvement is our use of the Barker (1998) algorithm, described in Section 5.5.1. This is only one way to bracket complex NPs, which is a standard structured search problem. Another potential framing would be to treat the task as a sequence tagging problem, where the goal is to generate some number of brackets between individual tokens. For an NP of length n, there would be $n + 1$ decision points where brackets could be inserted, as shown below:

$$d_1\ w_1\ d_2\ w_2 \cdots d_n\ w_n\ d_{n+1} \tag{7.1}$$

where the the words in the NP are $w_{1...n}$ and the decision points are $d_{1...n+1}$. So for the NP *crude* *oil prices*, an opening bracket should be inserted at d_1 (before *crude*) and a closing bracket at d_3 (after *oil*). The systems in Daumé III and Marcu (2004) and Bergsma and Wang (2007) function in a similar manner.

As part of our experiments with the C&C CCG parser, we added novel features based on NER and Web 1T information. The representation of these features is a possible area for improvement. For example, the Web 1T features could be conditioned on how distant the head words are in the sentence or how high in the tree the constituent is. We would also like to add Nakov and Hearst (2005a)-style query variations with the Web 1T features, however implementing this multitude of features would be difficult given the current parser architecture. It would also be interesting to use the actual *n*-gram counts, rather than discretising. Unfortunately, the C&C parser only accepts binary features at present.

The supertagger used by the CCG parser is another likely source of performance gains. The features that we implemented in the parsing model would also be applicable there. In fact, considering that a number of the NP errors made by the parser were related to the lexical categories, it may be the most likely and easiest way to improve performance. An increase in supertagger accuracy would then in turn, result in a more effective parsing model.

Chapter 6 demonstrated the advantages of converting NP structure data to CCG. However, as we described in Section 2.4, CCGbank is just one of a number of corpora that were created by conversion from the Penn Treebank. Other formalisms such as TAG, HPSG and LFG could benefit in the same way by converting our NP data to their respective corpora. Thus, our work allows an even wider range of parsers to recover NP structure for the first time.

7.3 Applications

Now that gold-standard NP structure data is available and parsers can be trained using it, the next step is to actually show that this is useful for NLP applications. Although we increased performance over the Collins' parser with our NP Bracketing system and over the C&C parsing baseline, it is still interesting to see how this improvement translates to downstream systems like Anaphora Resolution (AR). Due to the importance of NP structure in tasks such as this, as described in Section 1.1, we would expect a performance improvement even greater than what the parsing results have shown.

Furthermore, it may also prove useful to add features from downstream systems back into the parsing model. For example, jointly maximising the parsing structure and anaphora resolution would allow the overall best interpretation to be found. In this way, both systems could take advantage of the information supplied by the other. Such a model is only possible now that that NP structure is annotated properly and parsers are able to analyse it effectively.

Applicability to real-word tasks is a tremendously important part of our work. This can be seen in the effort made to bracket complex NPs in Chapter 5, rather than only considering the simplified problem. Because of this approach, our models are able to accurately recover NP structure in a wide-coverage large-scale context. Downstream systems can now take advantage of this crucial NP information, which is one of the most important outcomes of this thesis.

Chapter 8

Conclusion

This thesis has presented a thorough examination of statistical NP parsing. There are many tasks that we have accomplished in order to make this possible. From creating a NP data set that is much larger than all others, to experimenting with multiple parsers and creating a large-scale, wide-coverage bracketing system, we have addressed the NP parsing problem from beginning to end.

Chapter 3 began by annotating the entire Wall Street Journal section of the Penn Treebank. NP annotation was not present in the corpus previously, which in turn has caused most statistical parsers in the literature to ignore the problem. Correcting this inadequacy finally allows this long-existing problem to be solved.

Our annotation process has also refuted a long-held belief that NP structure is too difficult to bracket consistently. The inter-annotator agreement scores we achieved dispelled this belief comprehensively and were further supported by a comparison to DepBank. Our experience conclusively shows that humans *are* able to bracket NP structure correctly and consistently.

In Chapter 4, we put the NP augmented Penn Treebank to use in training and evaluating the Collins (2003) parsing model. The results of these experiments demonstrated the difficulty that statistical methods have in bracketing NPs. The parsing model could not effectively adapt to the productivity of NP structure, and as a result, its performance was lower than the baseline we set using deterministic rules. This baseline from the annotation tool's suggestion feature outperformed the parser by 5.81%.

Despite this, our analysis of Collins' model highlighted a number of interesting points. In particular, the continued importance of the base-NP submodel was a surprising result, as performance dropped spectacularly when it was removed. We also observed that NML and NP brackets should remain distinct. However, it is likely that performance would be increased if the pre-existing Penn Treebank annotations and our NP augmentations were made consistent. We also found that the head-finding rules were relatively unimportant, while leaving right-branching structure implicit performed better than bracketing it explicitly.

Lastly, our comprehensive error analysis showed that the largest cause of errors was a lack of lexical information in the training data. This theme was repeated throughout the thesis, and many of our approaches were designed to mitigate this problem. Although increased performance figures were not achieved in this chapter, the detailed analysis that we carried out was, in many ways, even more informative.

Chapter 5 saw the development of our NP Bracketing system. This is the first NP Bracketer that uses a supervised model to good effect and that can analyse NPs of arbitrary length and complexity. The initial simple NP bracketing experiments demonstrated that we could achieve performance on Lauer's small data set akin to that of previous researchers, e.g. Lauer (1995b) and Nakov and Hearst (2005a). However, our much larger data set from the Penn Treebank allowed us to build supervised models with even higher performance.

We moved onto the more realistic task of bracketing complex NPs. Utilising the supervised model we built for simple NPs, and including a wide range of features, both novel and based on those used by other researchers, we achieved an excellent performance figure of 89.14% matched bracket F-score. These results demonstrated that complex NP Bracketing is an interesting task with much room for innovation.

Using this complex NP Bracketer, we constructed a post-processor for the parsing experiments from the previous chapter. In doing so, we finally outperformed the suggestion baseline and improved on the parser's result by 9.04% F-score. That our NP Bracketing system performed better than a state-of-the-art parsing model is an excellent result.

In Chapter 6, many of the results from throughout the thesis were reinforced. We created a corpus with the correct NP structure and trained and evaluated a parser on this extended data. We then improved on the initial results using a number of novel information sources. Finally, we performed a thorough error analysis and determined that the main difficulty in parsing NPs is a lack of lexical information.

It was particularly important to solve the problems with the existing CCGbank. Previously, the annotation of NP structure and conjunctions were explicitly wrong, rather than merely underspecified as in the Penn Treebank. Correcting these errors is a substantial improvement to the corpus.

Having solved this problem, we began experimenting with the C&C parser. By implementing novel features using NER information and counts from the Web 1T corpus, performance was improved by 0.94% F-score. Most of these features have not been used before in any statistical parser.

The cross-formalism evaluation against DepBank was perhaps even more important. These results demonstrated that even though annotating NP structure lowered performance in previous experiments, this was only because the original model was not being evaluated on the NP structure. On this DepBank evaluation we found that after correcting the NP errors in CCGbank, performance rose by 0.92% F-score, even without any additional features. When our novel features were added, a total performance increase of 1.73% F-score was achieved.

Finally, we performed another error analysis, and again found that the model's greatest difficulty was parsing NPs that weren't in the training data. The disparity between NP dependencies and all dependencies was clear: previously seen NPs are very easy to parse, while unseen NPs are extremely difficult. Unfortunately, the nature of NPs means that each individual NP is likely to occur only once, and never be seen again. This behaviour is extremely difficult for a statistical model to deal with, as attested data is unavailable for so many NPs.

This thesis has been an important step on the road to better NP interpretation. Now that large-scale statistical NP parsing is possible, we expect to see further research on this problem in the future. Our annotation of the Penn Treebank has shown that consistent NP annotation is feasible, but also suggested many improvements, which will provide an even greater level of detail. The information supplied by explicitly bracketed NP structure can be used to good effect in parsing, as well as any system analysing NPs, which is one of the primary contributions of this thesis.

We have experimented with three different systems for parsing NPs, demonstrating that the results we achieved are widely applicable. In each case, we performed many experiments, not only identifying the best performing model, but also providing an in depth analysis of the models themselves. This approach has supplied a much better view of the various aspects involved in parsing NPs, the difficulties that are involved, and potential solutions to overcoming these problems.

Our work establishes NP parsing as a complex, interesting task. From our annotation of a large-scale data set, to the parsing experiments and detailed analysis we performed, we have made numerous contributions to this very difficult problem. The benefit of these results is now available for further research in many areas of Natural Language Processing. NP structure is particularly relevant for a number of downstream systems, all of which can now take advantage of this crucial information source.

Appendix A

Annotation Guidelines

This document describes guidelines for bracketing noun phrase (NP) structure in the Penn Treebank. These guidelines are in addition to the Treebank II Guidelines (Bies et al., 1995). They are also based on, and overlap with the Addendum for BioMedical Annotation (Warner et al., 2004). An earlier version (0.9) of these guidelines was used in the annotation described in Vadas and Curran (2007a), while this version was used in a subsequent pass over the data.

A.1 Bracketing NPs

The goal of our annotation is to identify and bracket multi-token premodifiers in NPs. Quirk et al. (1985, page 1321) describes such premodifiers, which include adjectives, participles, nouns, genitives and adverbs. All of these items are modifiable themselves, and this is precisely the behaviour that we have annotated. Indeed, NPs with multiple premodifiers can be recursive to an arbitrary depth (though more than 3 or 4 levels is unusual), and the underlying structure is by no means always right-branching. However, we can still resolve this ambiguity, as (with our emphasis)

> *. . . obscurity in premodification exists* ***only*** *for the hearer or reader who is unfamiliar with the subject concerned. . .* (Quirk et al., 1985, page 1343)

Thus, our most difficult cases come from the financial jargon of the Wall Street Journal, but the correct bracketing of most NPs is simple to ascertain.

The main change described in these guidelines is a different way of representing NP structure. Treebank II Style is to leave NPs flat, not specifying additional structure. In our extension, we assume a right-branching structure in all NPs, and mark explicitly any left-branching constituents.

As most NPs are right-branching, this reduces the amount of bracketing required and thus increases legibility. This means that NPs like this one do not need further bracketing:

```
(NP (DT The) (JJ average)
  (JJ seven-day) (NN compound) (NN yield) )
```

And the implicit structure represented is:

```
(NP (DT The)
  (NODE (JJ average)
    (NODE (JJ seven-day)
      (NODE (NN compound)
        (NODE (NN yield) ) ) ) ) )
```

When a left-branching modifier is present, as in the NP below,

```
(NP (NN lung) (NN cancer) (NNS deaths) )
```

it is bracketed explicitly. To specify that *lung cancer* is a constituent, we insert a bracket around those words:

```
(NP
  (NML (NN lung) (NN cancer) )
  (NNS deaths) )
```

Though less frequent, brackets can also be necessary in non-base-NPs, as in these examples:

```
(NP-SBJ
  (NML (JJ former)
    (NAC (NNP Ambassador)
      (PP (TO to)
        (NP (NNP Costa) (NNP Rica) ) ) ) )
  (NNP Francis) (NNP J.) (NNP McNeil) )
```

```
(NP
  (NML
    (NP (NN Wendy) (POS 's) )
    (NNP International) )
  (NNP Inc.) )
```

In the first example, we join *former* and the NAC node, as he is formerly the Ambassador, not formerly Mr. McNeil.

Multiple words can be included in a bracket, and internal to the bracket, are still implicitly right-branching.

```
(NP
  (NML (JJ chief) (JJ financial) (NN officer) )
  (NNP John) (NNP Pope) )


(NP
  (NML (JJ hot-dipped) (JJ galvanized) (NN sheet) )
  (NNS products) )
```

So the sheet is hot-dipped and galvanized, and the products are made of this sheet. Alternate, incorrect bracketings could suggest the galvanization is hot-dipped (a NML node around those two words) or that the products themselves are hot-dipped and galvanized (if no NML node was used).

New brackets can be nested, and this is needed quite often.

```
(NP
  (NML
    (NML (NNP New) (NNP York) )
    (NNP Stock) (NNP Exchange) )
  (JJ composite) (NN trading) )
```

This correct bracketing describes composite trading on the Stock Exchange of New York.

Note that we never alter existing Treebank brackets or POS tags, we only add new brackets to specify our extended representation. Similarly, we have not corrected errors that have been noticed during the annotation process. This is so that the corpus remains as comparable as possible to the original version. However, pre-existing errors can mean that the correct extended annotation cannot possibly be implemented. In these cases, we try to markup any constituents that we still can, while not adding any brackets that are incorrect. This often results in the opposite to what is done in the normal case.

```
(NP
  (NP (DT the) (NNP Carper) (POS 's) )
  (NNP Creek)
  (NN wine) )
```

In this example, the determiner should be outside the inner `NP`, so that it has scope over *wine*. Normally, we would bracket *the Carper* to separate it from the possessive (see Section A.2.7), but that is incorrect here. Similarly, we do not bracket *the Carper's Creek* because it would include *the*, This would be incorrect, as *the* is the determiner for the overall `NP`, not just *Carper's Creek*.

A.1.1 Node Labels

We use two new node labels: `NML` and `JJP`. We have distinguished these from the existing `NP` and `ADJP` labels, so that we can analyse them separately. This approach has the advantage that they can be mapped back to the existing labels if needed. `NML` is used when the modifier's head is a noun, as in previous examples, while `JJP` is used when the head is adjectival, as in the example below.

```
(NP (JJP (JJ dark) (JJ red) )
  (NN car) )
```

The label should also be `JJP` in cases where the head is a gerund.

```
(NP (DT the)
  (JJP (JJS fastest) (VBG developing) )
  (NNS trends) )
```

A `JJP` node is needed when an adverb modifies an adjective:

```
(NP
  (JJP (RB relatively) (JJR higher) )
  (NNS rates) )
```

Finally, we also apply the JJP label to coordinated adjectives premodifying a noun. In cases like these with multiple heads, only one head needs to be adjectival for the label to be JJP. We do not have a label similar to UCP.

```
(NP (PRP$ its)
  (JJP (JJ current)
    (CC and) (JJ former) )
  (NNS ratepayers) )


(NP (DT the)
  (JJP (JJ British)
    (CC and) (NNP U.S.) )
  (NNS troops) )
```

In all other cases (the vast majority), a NML label should be used. This means that cases with unusual heads, like the one below where DTs are being coordinated, are labelled NML.

```
(NP
  (NML (DT any)
    (CC or) (DT all) )
  (NNS warrants) )
```

If any POS tag has been incorrectly annotated, then the label used should reflect the correct POS tag, rather than propagate the error.

A.1.2 Ambiguous Cases

In general, if an annotator is unsure as to whether bracketing is needed, or if both alternatives seem equally likely, then they should leave the NP flat. The NPs below are examples of such semantically ambiguous cases. In the first, both dependencies are true, i.e. the players are in college, and they play basketball. While the third example has a genuinely flat structure.

```
(NP (NN college) (NN basketball) (NNS players) )


(NP (NN army) (NN ordnance) (NN depot) )


(NP (NNP John) (NNP A.) (NNP Smith) )
```

A.1.3 Head Derivation

Head-finding rules for `NML` and `JJP` constituents are the same as for `NP` and `ADJP` nodes respectively. For a detailed description of these rules, see Collins (1999, page 238). In most cases, the head is either the right-most noun, or inside the right-most `NML` node.

This is more complicated with coordinated and apposited structures, which will have multiple heads. The individual heads can still be determined with the standard rules.

A.1.4 Identifying the Correct Bracketing

The bracketing task involves deciding which words belong together as constituents. It is often useful to reword the sentence to see whether a constituent makes sense. In doing so, the aim is to determine the dependencies that will be formed, i.e. to create a syntactic structure which yields the correct semantic structure. Here are a few ways this can be done:

1. Inversion – In the following NP, we are deciding whether or not to bracket *other two*.

   ```
   (NP-LGS (DT the) (JJ other)
     (CD two) (JJ outside) (NNS bidders) )
   ```

   If we invert these words to *two other*, then the NP retains the same meaning. Therefore *other* does not modify *two* and they should not be bracketed.

2. Removal – This test involves trying to force one word to modify another by placing them side by side, removing the intervening text.

   In the example below, does *Japanese* modify *auto maker* or *Mazda Motor Corp*? If we remove *auto maker*, then the NP would not make sense, and so it must be the former. We have inserted the appropriate `NML` node.

   ```
   (NP (NML (JJ Japanese) (NN auto) (NN maker) )
     (NML (NNP Mazda) (NNP Motor) )
     (NNP Corp) )
   ```

3. Postmodifier – If we move a premodifier to the end of the NP, making it postmodify the head, then the correct bracketing should become clearer. In the following description of a car that is a certain shade of red,

```
(NP (JJ tomato) (JJ red) (NN car) )
```

if we change the NP to *red car that is tomato* then we get a meaning that doesn't make sense. As this is not the case, we know that *tomato* and *red* should be joined in a constituent.

```
(NP
  (JJP (JJ tomato) (JJ red) )
  (NN car) )
```

A.2 Specific Cases

A.2.1 Conjunctions

Conjunctions are one of the most difficult structures to bracket in NPs. This is because of the multi-headed nature of such constructs. We should not read the first example below as implicitly right-branching, but with dependencies between *Bill* and *and*, and *Ted* and *and*. It does not need further bracketing.

```
(NP (NNP Bill) (CC and) (NNP Ted) )
```

On the other hand, the following example does need the `NML` bracket shown:

```
(NP (DT the)
  (NML (NNPS Securities)
    (CC and) (NNP Exchange) )
  (NNP Commission) )
```

Otherwise, its implicit structure would be as follows:

```
(NP (DT the)
  (NODE
    (NODE (NNPS Securities) )
    (CC and)
    (NODE (NNP Exchange)
      (NODE (NNP Commission) ) ) ) )
```

The erroneous meaning here is *the Securities* and *the Exchange Commission*, rather than the correct *the Securities Commission* and *the Exchange Commission*.

Bracketing is also needed in the first example below, or else the interpretation will be *rock stars* and *rock royalty*, which is clearly incorrect. However, this *is* the case in the second example (both the words and actions are rude) and so no new brackets are needed there.

```
(NP (NML (NN rock) (NNS stars) )
  (CC and)
  (NML (NN royalty) ) )


(NP (JJ rude) (NNS words)
  (CC and) (NNS actions) )
```

Also note that *royalty* is bracketed as a single word. This is because whenever one coordinated constituent is bracketed, all other constituents of the coordinate must be bracketed as well, even single tokens as above. This has changed since version 0.9 of these guidelines.

The implicit structure of the NP is correct below, as *rock stars* is already right-most.

```
(NP (NN royalty)
  (CC and) (NN rock) (NNS stars) )
```

However, this NP should be treated in the same way as the one above. We therefore insert brackets around *rock stars* and *royalty* as before.

```
(NP (NML (NN royalty) )
  (CC and)
  (NML (NN rock) (NNS stars) ) )
```

If *any* constituent to be coordinated is multi-token (even right-most and implicitly correct ones), then *all* constituents of the coordinator must be explicitly bracketed. This is another change since the version 0.9 guidelines, which would not add any new brackets to the example above.

Lists do *not* need any bracketing.

```
(NP (NNS cars)
  (, ,)
  (NNS trucks)
  (CC and) (NNS buses) )
```

This is true even when the conjunction is missing:

```
(NP
  (NP (DT no) (NN crack) (NNS dealers) )
  (, ,)
  (NP
    (NP (DT no) (JJ dead-eyed) (NNS men) )
    (VP (VBG selling)
      (NP
        (NP (JJ four-year-old) (NNS copies) )
        (PP (IN of)
          (NP (NNP Cosmopolitan) )))))
  (, ,)
  (NP
    (NP (DT no) (PRP one) )
    (VP (VBD curled)
      (PRT (RP up) )
      (PP-LOC (IN in)
        (NP (DT a) (NN cardboard) (NN box) )))))
```

However, the entire list may still need to be bracketed before being joined to words outside the list, as below:

```
(NP
  (NP (NNP Mazda) (POS 's) )
  (NNP U.S.)
  (NML (NNS sales)
    (, ,)
    (NN service)
    (, ,)
    (NNS parts)
    (CC and) (NN marketing) )
  (NNS operations) )
```

A list of attributes separated by commas does *not* need any bracketing:

```
(NP
  (JJ tricky)
  (, ,)
  (JJ unproven) (NN chip) (NN technology) )
```

This is because *tricky* and *unproven* are *not* being coordinated here. They are simply both acting as modifiers on *technology*, like in the NP: *big red car*.

Conjunctions over a *neither/nor* pair do *not* need any bracketing.

```
(NP-SBJ (DT Neither)
  (NP (NNP Lorillard) )
  (CC nor)
  (NP
    (NP (DT the) (NNS researchers) )
    (SBAR
      (WHNP-3 (WP who) )
      (S
        (NP-SBJ (-NONE- *T*-3) )
        (VP (VBD studied)
          (NP (DT the) (NNS workers) ))))))
```

A.2.2 Speech Marks

Tokens surrounded by speech marks should be bracketed:

```
(NP-PRD (DT a)
  (NML (‘‘ ‘‘) (JJ long) (NN term) (’’ ’’) )
  (NN decision) )
```

This includes when there is only a single token inside the speech marks, and when the speech marks are right-most:

```
(NP-PRD (DT a)
  (JJP (‘‘ ‘‘) (JJ long) (’’ ’’) )
  (NN decision) )
```

```
(NP-PRD (DT a)
  (NML (‘‘ ‘‘) (JJ long) (NN term) (’’ ’’) ) )
```

Note that the label of the bracket should reflect the internal head, as in the first example in the previous block, where JJP is used.

If the speech marks and the tokens they surround are the only items under the NP, then a new bracket should *not* be added.

```
(NP-PRD (‘‘ ‘‘) (JJ long) (NN term) (’’ ’’) )
```

The bracketing of speech marks has changed since the 0.9 version guidelines. The internal tokens were bracketed previously, while right-most speech marks were not.

Conventional editorial style for speech marks does not lend itself to bracketing easily. Because of this, there are a number of exceptions and corner cases when annotating NPs with speech marks. Firstly, in the example below:

```
(NP (‘‘ ‘‘)
  (NP-TTL (DT A) (NNP Place) (IN in) (NNP Time) )
  (, ,)
  (’’ ’’)
  (NP
    (NP (DT a) (JJ 36-minute) (JJ black-and-white) (NN film) )
    (PP (IN about)
      (NP
        (NP (DT a) (NN sketch) (NN artist) )
        (, ,)
        (NP
          (NP (DT a) (NN man) )
          (PP (IN of)
            (NP (DT the) (NNS streets) ) ) ) ) ) ) )
```

the comma serves to separate the film’s title from its description, while the speech marks surround just the title. This causes a “crossing” constituent, as we cannot bracket the speech marks and the title together without including the comma. In these cases, we still add a NML bracket around the speech marks:

```
(NP
  (NML (`` ``)
    (NP-TTL (DT A) (NNP Place) (IN in) (NNP Time) )
    (, ,)
    ('' '') )
  (NP
    (NP (DT a) (JJ 36-minute) (JJ black-and-white) (NN film) )
    (PP (IN about)
      (NP
        (NP (DT a) (NN sketch) (NN artist) )
        (, ,)
        (NP
          (NP (DT a) (NN man) )
          (PP (IN of)
            (NP (DT the) (NNS streets) ) ) ) ) ) ) )
```

Many NPs contain a single opening or closing speech mark, whose partner is stranded in another constituent. For example, the NP below, with more context containing the closing speech mark shown afterwards.

```
(NP (DT the) (`` ``)
  (NML (NN type) (NN F) )
  (NN safety) (NN shape) )
```

```
(NP
  (NP (DT the) (`` ``)
    (NML (NN type) (NN F) )
    (NN safety) (NN shape) )
  (, ,)
  ('' '')
  (NP
    (NP (DT a) (JJ four-foot-high) (JJ concrete) (NN slab) )
    (PP (IN with)
      (NP (DT no) (NNS openings) ) ) ) )
```

In these cases, we could not bracket the speech marks properly without altering the existing structure. So once again, we do not add any new brackets in NPs such as this. In the next example, the speech marks have not been put in the right place:

```
(NP-PRD (`` ``) (DT a) (JJ worst-case) ('' '') (NN scenario) )
```

The determiner should be outside the speech marks. In cases such as these, the annotator should not follow the incorrect placement. Because no accurate bracketing can be inserted, no brackets should be added at all.

A.2.3 Brackets

These should be treated the same as speech marks, and bracketed as described above.

```
(NP (DT an)
  (JJP (-LRB- -LCB-) (VBG offending) (-RRB- -RCB-) )
  (NN country) )
```

An example of another corner case is shown below:

```
(NP (-LRB- -LCB-)
  (NML (NNP Fed) (NNP Chairman) )
  (NNP Alan)
  (-RRB- -RCB-)
  (NNP Greenspan) )
```

Once again, the tokens cannot be bracketed without a crossing constituent. We can still bracket *Fed Chairman*, but beyond that, no other brackets should be added.

A.2.4 Companies

Company names may need to be bracketed a number of ways. When there are postmodifiers such as Corp. or Ltd., the rest of the company needs to be separated if it is longer than one word.

```
(NP-SBJ
  (NML (NNP Pacific) (NNP First) (NNP Financial) )
  (NNP Corp.) )
```

```
(NP
  (NML (NNP W.R.) (NNP Grace) )
  (CC &) (NNP Co.) )

(NP
  (NML (NNP Goldman)
    (, ,)
    (NNP Sachs) )
  (CC &) (NNP Co.) )
```

Other identifiable nominal groups within the company name, such as locations, also need to be bracketed separately.

```
(NP
  (NP (NN today) (POS 's) )
  (NML (NNP New) (NNP England) )
  (NNP Journal) )

(NP (DT the)
  (NML (NNP Trade)
    (CC and) (NNP Industry) )
  (NNP Ministry) )
```

A.2.5 Final Adverbs

The tokens preceding a final adverb should be separated:

```
(NP (NML (NN college) (NNS radicals) )
  (RB everywhere) )
```

A.2.6 Names

Names are to be left unbracketed:

```
(NP (NNP Brooke) (NNP T.) (NNP Mossman) )
```

However, numbers, as well as *Jr.*, *Sr.*, etc should be separated:

```
(NP
  (NML (NNP William) (NNP H.) (NNP Hudnut) )
  (NNP III) )
```

Titles that are longer than one word also need to be bracketed separately.

```
(NP
  (NML (NNP Vice) (NNP President) )
  (NNP John) (NNP Smith) )
```

A.2.7 Possessives

NPs preceding possessives need to be bracketed.

```
(NP (NML (NNP Grace) (NNP Energy) )
  (POS 's) )
```

A.2.8 Postmodifying Constituents

The words preceding a postmodificational constituent, such as a preposition or SBAR, do *not* need to be bracketed.

```
(NP
  (DT the) (JJ common) (NN kind)
  (PP (IN of)
    (NP (NN asbestos) )))
```

A.2.9 Unit Traces

This trace is necessary to make the unit (dollars in the example below) the head of the NP.

```
(NP (RB over) ($ $) (CD 27) (-NONE- *U*) )
```

If the NP is longer, and there are words to the right of the amount, then the trace should be inside the bracket.

```
(NP (DT a)
  (NML ($ $) (CD 27) (-NONE- *U*) )
  (NN charge) )
```

A.2.10 Unusual Punctuation

Sometimes a period indicating an acronym will be separated from the initial letter(s). In these cases, a bracket should be added to join them back together, as below:

```
(NP (NNP Finmeccanica)
  (NML (NNP S.p) (. .) )
  (NNP A.) )
```

Some NPs also include final punctuation. These are mostly short fragmental sentences. In these cases, the rest of the NP should have a bracket placed around it:

```
(NP
  (NML
    (NML (NNP New) (NNP York) )
    (NNP City) )
  (: :) )
```

A.3 Future Improvements

Here we describe improvements to these guidelines and the bracketing scheme that we intend to carry out in the future. We noticed these issues during the first pass through the corpus, and all of them require another full pass.

A.3.1 Flat Structures

There are a number of NPs in the Penn Treebank that display genuinely flat structure. For some examples, refer back to Section A.1.2. We would like to distinguish these from the implicitly right-branching structures that make up the majority of the corpus. To do this, we intend to use a marker on the NP, NML or JJP label itself, as shown below:

```
(NP-FLAT (NNP John) (NNP A.) (NNP Smith) )


(NP
  (NML-FLAT (NNP John) (NNP A.) (NNP Smith) )
  (NNS apples) )
```

A.3.2 Appositions

Appositions are a multi-headed structure, similar but still different to coordination. They are extremely common throughout the Penn Treebank, and usually fit the pattern shown below, with a person's name and their position separated by a comma:

```
(NP-SBJ
  (NP (NNP Rudolph) (NNP Agnew) )
  (, ,)
  (NP
    (NP (JJ former) (NN chairman) )
    (PP (IN of)
      (NP (NNP Gold) (NNP Fields) (NNP PLC) ) ) ) )
```

We would like to mark these structures explicitly, so that they can be treated appropriately. This raises issues of what is and isn't an apposition (whether they are truly co-referential), and whether to discriminate between different types.

A.3.3 Head Marking

For some NPs, Collins' standard head-finding rules do not work correctly. In the example below, *IBM* is the head, but *Australia* would be found.

```
(NP (NNP IBM) (NNP Australia) )
```

Marking heads explicitly would require a much larger degree of work, as NPs of length two would be ambiguous. All other annotation described here only needs to look at NPs of length three or more.

Appendix B

Company and Name Endings

As described in Section 3.2.1, the annotation tool makes bracketing suggestions based on frequently occurring endings of company and person names. These endings are listed here.

UNIGRAMS		BIGRAMS	
Ltd.	CO.	Ltd .	S.A .
Corp.	CORP.	Corp .	L.P .
Co.	Cos.	& Co.	Cos .
Inc.	N.V.	Inc .	Co. Inc.
Co	L.P.	Co .	Co. PLC
Inc	B.V.	& Co	CORP .
PLC	Pty.	PLC .	& CO.
Corp	NV	Co. Ltd.	
INC.	AB		
Ltd	S.A		
AG	G.m.b.H.		
S.A.	AS		

Table B.1: Company name endings

UNIGRAMS		BIGRAMS
Sr.	III	Jr .
II	Jr.	Sr .
.	Jr	

Table B.2: Person name endings

References

Abney, Steven. 1987. *The English noun phrase in its sentential aspects*. Ph.D. thesis, MIT, Cambridge, MA, USA.

Ajdukiewicz, Kazimierz. 1935. Die syntaktische Konnexität. In *Polish Logic 1920–1939*. Oxford University Press, Oxford, UK, pages 207–231. Translated from Studia Philosophica, 1, 1–27.

Atterer, Michaela and Hinrich Schütze. 2007. Prepositional phrase attachment without oracles. *Computational Linguistics*, 33(4):469–476.

Bangalore, Srinivas and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Bar-Hillel, Yehoshua. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Barker, Ken. 1998. A trainable bracketer for noun modifiers. In *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence (LNAI 1418)*, pages 196–210, Vancouver, Canada.

Barker, Ken and Stan Szpakowicz. 1998. Semi-automatic recognition of noun modifier relationships. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 96–102, Montreal, Canada, August.

Berger, Adam, Stephen Della Pietra, and Vincent Della Pietra. 1996. A Maximum Entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Bergsma, Shane and Qin Iris Wang. 2007. Learning noun phrase query segmentation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 819–826, Prague, Czech Republic, June 28–30.

Bies, Ann, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for Treebank II style Penn Treebank project. Technical report.

Bikel, Daniel M. 2004. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

Black, Ezra, Steven Abney, Dan Flickinger, Claudia Gdaniec, Ralph Grishman, Philip Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith Klavans, Mark Liberman, Mitch Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the February 1991 DARPA Speech and Natural Language Workshop*.

Blaheta, Don and Eugene Charniak. 2000. Assigning function tags to parsed text. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 234–240, Seattle, WA, USA, April 29–May 4.

Blunsom, Phil and Timothy Baldwin. 2006. Multilingual deep lexical acquisition for HPSGs via supertagging. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP-06)*, pages 164–171, Sydney, Australia, July 22–23.

Bod, Rens. 1992. A computational model of language performance: Data Oriented Parsing. In *Proceedings of 14th International Conference on Computational Linguistics (COLING-92)*, pages 855–859, Nantes, France, August 23–28.

Bod, Rens. 1993. Using an annotated language corpus as a virtual stochastic grammar. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 778–783, Washington, DC, USA, July 11–15.

Bod, Rens. 2001. What is the minimal set of fragments that achieves maximal parse accuracy? In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistic and the 10th Conference of the European Chapter (ACL-01)*, pages 66–73, Toulouse, France, July 9–11.

Bos, Johan, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 1240–1246, Geneva, Switzerland, August 23–27.

Brants, Thorsten and Alex Franz. 2006. Web 1T 5-gram version 1. Technical report. LDC Catalog No.: LDC2006T13.

Brill, Eric. 1993. *A Corpus-based Approach to Language Learning*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

Brill, Eric. 1995. Unsupervised learning of disambiguation rules for part of speech tagging. In *Proceedings of the 3rd Workshop on Very Large Corpora*, pages 1–13, Cambridge, MA, USA.

Briscoe, Ted and John Carroll. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 41–48, Sydney, Australia, July 17–21.

Buckeridge, Alan M. and Richard F. E. Sutcliffe. 2002. Using latent semantic indexing as a measure of conceptual association for noun compound disambiguation. In *Proceedings of the 13th Irish International Conference on Artificial Intelligence and Cognitive Science (AICS-02)*, pages 12–19, Limerick, Ireland, September 12–13.

Buyko, Ekaterina, Katrin Tomanek, and Udo Hahn. 2007. Resolution of coordination ellipses in biological named entities with conditional random fields. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics (PACLING-07)*, pages 163–171, Melbourne, Australia, September 19–21.

Cahill, Aoife, Michael Burke, Ruth O'Donovan, Josef van Genabith, and Andy Way. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 320–327, Barcelona, Spain, July 21–26.

Cahill, Aoife, Mairéad McCarthy, Josef van Genabith, and Andy Way. 2002. Automatic annotation of the Penn-Treebank with LFG F-structure information. In *Proceedings of the Third International Conference on Language Resources and Evaluation Workshop on Linguistic*

Knowledge Acquisition and Representation - Bootstrapping Annotated Language Data, pages 8–15, Paris, France, June 1.

Charniak, Eugene. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 598–603, Providence, RI, USA, July 27–31.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-00)*, pages 132–139, Seattle, WA, USA, April 29–May 4.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 173–180, Ann Arbor, MI, USA, June 25–30.

Chen, John, Srinivas Bangalore, and K. Vijay-Shanker. 2006. Automated extraction of Tree-Adjoining Grammars from treebanks. *Natural Language Engineering*, 12(3):251–299.

Chiang, David. 2000. Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics (ACL-00)*, pages 456–463, Hong Kong, October 1–8.

Chiang, David and Daniel M. Bikel. 2002. Recovering latent information in treebanks. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING-02)*, pages 1–7, Taipei, Taiwan, August 24–September 1.

Clark, Stephen. 2002. Supertagging for Combinatory Categorial Grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 19–24, Venice, Italy, May 20–23.

Clark, Stephen and James R. Curran. 2004a. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 282–288, Geneva, Switzerland, August 23–27.

Clark, Stephen and James R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 104–111, Barcelona, Spain, July 21–26.

Clark, Stephen and James R. Curran. 2007a. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07)*, pages 248–255, Prague, Czech Republic, June 25–27.

Clark, Stephen and James R. Curran. 2007b. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Clark, Stephen, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 327–334, Philadephia, PA, USA, July 6–12.

Cocke, John and Jacob T. Schwartz. 1970. *Programming languages and their compilers: preliminary notes*. Courant Institute of Mathematical Sciences, New York University, New York, NY, USA.

Cohen, Paul R. 1995. *Empirical Methods for Artifical Intelligence*. MIT Press, Cambridge, MA, USA.

Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 184–191, Santa Cruz, CA, USA, June 24–27.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics (ACL-97)*, pages 16–23, Madrid, Spain, July 7–12.

Collins, Michael. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, pages 175–182, San Francisco, CA, USA, June 29–July 2.

Collins, Michael. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

Curran, James R. and Stephen Clark. 2003. Language independent NER using a maximum entropy tagger. In *Proceedings of the Seventh Conference on Natural Language Learning (CoNLL-03)*, pages 164–167, Edmonton, Canada, May 31–June 1.

Darroch, J.N. and D Ratcliff. 1972. Generalised iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43(5):1470–1480.

Daumé III, Hal. 2004. Notes on CG and LM-BFGS optimization of logistic regression. Paper available at `http://pub.hal3.name`, implementation available at `http://hal3.name/megam/`, August.

Daumé III, Hal and Daniel Marcu. 2004. NP bracketing by maximum entropy tagging and SVM reranking. In Dekang Lin and Dekai Wu, editors, *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP-04)*, pages 254–261, Barcelona, Spain, July 25–26.

Eisner, Jason. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 79–86, Santa Cruz, CA, USA, June 24–27.

Fayyad, Usama M. and Keki B. Irani. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence (IJCAI–93)*, pages 1022–1029, Chambery, France, August 28–September 3.

Fellbaum, Christiane, editor. 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, USA.

Francis, W. Nelson and Henry Kučera. 1982. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin, Boston, MA, USA.

Gabbard, Ryan, Mitchell Marcus, and Seth Kulick. 2006. Fully parsing the Penn Treebank. In *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL-06)*, pages 184–191, New York, NY, USA, June 5–7.

Garside, Roger, Geoffrey Leech, and Geoffrey Sampson, editors. 1987. *The Computational analysis of English: A corpus-based approach*. Longman, London, UK.

Gazdar, Gerald, Ewan H. Klein, Geoffrey K. Pullum, and Ivan Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, MA, USA.

Girju, Roxana. 2007. Improving the interpretation of noun phrases with cross-linguistic information. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL-07)*, pages 568–575, Prague, Czech Republic, June 25–27.

Girju, Roxana, Dan Moldovan, Marta Tatu, and Daniel Antohe. 2005. On the semantics of noun compounds. *Journal of Computer Speech and Language - Special Issue on Multiword Expressions*, 19(4):313–330.

Goodman, Joshua. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 177–183, Santa Cruz, CA, USA, June 24–27.

Goodman, Joshua. 1997. Probabilistic feature grammars. In *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT-97)*, September.

Hawker, Toby. 2007. USYD: WSD and lexical substitution using the Web1T corpus. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 446–453, Prague, Czech Republic, June 23–24.

Hindle, Donald. 1983. User manual for Fidditch. Technical Report 7590-142, Naval Research Laboratory.

Hindle, Donald. 1989. Acquiring disambiguation rules from text. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL-89)*, pages 118–125, Vancouver, Canada, June 26–29.

Hindle, Donald and Mats Rooth. 1993. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120.

Hockenmaier, Julia. 2003a. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh, Edinburgh, UK.

Hockenmaier, Julia. 2003b. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, pages 359–366, Sapporo, Japan, July 7–12.

Hockenmaier, Julia and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 335–342, Philadephia, PA, USA, July 6–12.

Hockenmaier, Julia and Mark Steedman. 2005. CCGbank manual. Technical Report MS-CIS-05-09, Department of Computer and Information Science, University of Pennsylvania.

Hockenmaier, Julia and Mark Steedman. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Hogan, Deirdre. 2007. Coordinate noun phrase disambiguation in a generative parsing model. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL-07)*, pages 680–687, Prague, Czech Republic, June 25–27.

Honnibal, Matthew and James R. Curran. 2007. Improving the complement/adjunct distinction in CCGbank. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics (PACLING-07)*, pages 210–217, Melbourne, Australia, September 19–21.

Johnson, Mark. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Joshi, Aravind and Yves Schabes. 1992. Tree adjoining grammars and lexicalized grammars. In *Tree Automata and Languages*. North-Holland, pages 409–432.

Jurafsky, Daniel and James H. Martin. 2000. *Speech and Language Processing*. Prentice-Hall, Upper Saddle River, NJ, USA.

Kamp, Hans and Barbara Partee. 1995. Prototype theory and compositionality. *Cognition*, 57(2):129–191.

Kaplan, Ronald and Joan Bresnan. 1982. *Lexical-Functional Grammar: A formal system for grammatical representation*. MIT Press, Cambridge, MA, USA.

Kasami, Tadao. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Bedford, MA, USA.

King, Tracy Holloway, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC700 dependency bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, Budapest, Hungary, April 13–14.

Klein, Dan and Christopher Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, pages 423–430, Sapporo, Japan, July 7–12.

Klein, Dan and Christopher D. Manning. 2001. Parsing with treebank grammars: empirical bounds, theoretical models, and the structure of the Penn Treebank. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (ACL-01)*, pages 338–345, Toulouse, France, July 9–11.

Koehn, Philipp. 2003. *Noun Phrase Translation*. Ph.D. thesis, University of Southern California.

Kübler, Sandra. 2005. How do treebank annotation schemes influence parsing results? Or how not to compare apples and oranges. In *Proceedings of the Recent Advances in Natural Language Processing Conference (RANLP-05)*, Borovets, Bulgaria, September 21–23.

Kulick, Seth, Ann Bies, Mark Liberman, Mark Mandel, Ryan McDonald, Martha Palmer, Andrew Schein, and Lyle Ungar. 2004. Integrated annotation for biomedical information extraction. In *Proceedings of BioLink Workshop at the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (BioLink-04)*, Boston, MA, USA, May 6.

Lapata, Mirella and Frank Keller. 2004. The web as a baseline: Evaluating the performance of unsupervised web-based models for a range of NLP tasks. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL-04)*, pages 121–128, Boston, MA, USA, May 2–7.

Lauer, Mark. 1995a. Corpus statistics meet the compound noun: Some empirical results. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, pages 47–54, Cambridge, MA, USA, June 26–30.

Lauer, Mark. 1995b. *Designing Statistical Language Learners: Experiments on Noun Compounds*. Ph.D. thesis, Macquarie University, Sydney, Australia.

Levi, Judith. 1978. *The Syntax and Semantics of Complex Nominals*. Academic Press, New York, NY, USA.

Lewin, Ian. 2007. BaseNPs that contain gene names: domain specificity and genericity. In *Workshop on Biological, Translational, and Clinical Language Processing*, pages 163–170, Prague, Czech Republic, June 29.

Liberman, Mark and Richard Sproat. 1992. The stress and structure of modified noun phrases in english. In *Lexical Matters – CSLI Lecture Notes No. 24*. CSLI Publications, Stanford, CA, USA, chapter 6, pages 131–182.

Maamouri, Mohamed, Ann Bies, Hubert Jin, and Tim Buckwalter. 2003. Arabic Treebank: Part 1 v2.0. Technical report. LDC Catalog No.: LDC2003T06.

Macleod, Catherine, Ralph Grishman, Adam Meyers, Leslie Barrett, and Ruth Reeves. 1998. NOMLEX: a lexicon of nominalizations. In *Proceedings of the 8th International Congress of the European Association for Lexicography (EURALEX-98)*, pages 187–193, Liege, Belgium.

Magerman, David. 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

Magerman, David. 1995. Statistical decision tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, pages 276–283, Cambridge, MA, USA, June 26–30.

Manning, Christopher and Hinrich Schutze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Marcus, Mitchell. 1980. *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, MA, USA.

Marcus, Mitchell, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology (HLT-94)*, pages 114–119, Plainsboro, NJ, USA, March 8–11.

Marcus, Mitchell, Beatrice Santorini, and Mary Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 75–82, Ann Arbor, MI, USA, June 25–30.

Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1671–1676, Hyderabad, India, January 6–12.

McClosky, David, Eugene Charniak, and Mark Johnson. 2006. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL-06)*, pages 337–344, Sydney, Australia, July 17–21.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technologies and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pages 523–530, Vancouver, Canada, October 6–8.

McInnes, Bridget, Ted Pedersen, and Serguei Pakhomov. 2007. Determining the syntactic structure of medical terms in clinical notes. In *Workshop on Biological, Translational, and Clinical Language Processing*, pages 9–16, Prague, Czech Republic, June 29.

Melamed, I. Dan, Giorgio Satta, and Benjamin Wellington. 2004. Generalized multitext grammars. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 661–668, Barcelona, Spain, July 21–26.

Miller, Scott, Michael Crystal, Heidi Fox, Lance Ramshaw, Richard Schwartz, Rebecca Stone, and Ralph Weischedel. 1998. Algorithms that learn to extract information — BBN: Description of the SIFT system as used for MUC-7. In *Proceedings of the 7th Message Understanding Conference (MUC-7)*, Washington, DC, USA, April 29.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP-04)*, pages 684–693, Hainan Island, China, March 22–24.

Miyao, Yusuke and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference (HLT-02)*, San Diego, CA, USA.

Miyao, Yusuke and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 83–90, Ann Arbor, MI, USA, June 25–30.

Moldovan, Dan, Christine Clark, Sanda Harabagiu, and Steve Maiorano. 2003. COGEX: A logic prover for question answering. In *Proceedings of the Human Language Technology and North American Chapter of the Association for Computational Linguistics Conference (HLT-NAACL-03)*, pages 166–172, Edmonton, Canada, May 27–June 1.

Nakov, Preslav and Marti Hearst. 2005a. Search engine statistics beyond the n-gram: Application to noun compound bracketing. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL-05)*, pages 17–24, Ann Arbor, MI, USA, June 29–30.

Nakov, Preslav and Marti Hearst. 2005b. A study of using search engine page hits as a proxy for n-gram frequencies. In *Proceedings of the Recent Advances in Natural Language Processing Conference (RANLP-05)*, Borovets, Bulgaria, September 21–23.

Nivre, Joakim and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 64–70, Geneva, Switzerland, August 23–27.

Nocedal, Jorge and Stephen J. Wright. 1999. *Numerical Optimization*. Springer, New York, NY, USA.

Noreen, Eric W. 1989. *Computer Intensive Methods for Testing Hypotheses: An Introduction*. John Wiley & Sons, New York, NY, USA.

Oepen, Stephan, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods Treebank: Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING-02)*, pages 1253–1257, Taipei, Taiwan, August 24–September 1.

Pado, Sebastian and Mirella Lapata. 2007. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2):161–199.

Palmer, Martha, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics (ACL-02)*, pages 311–318, Philadelphia, PA, USA, July 6–12.

Petrov, Slav, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL-06)*, pages 433–440, Sydney, Australia, July 17–21.

Pollard, Carl and Ivan Sag. 1992. Anaphors in english and the scope of binding theory. *Linguistic Inquiry*, 23:261–303.

Pollard, Carl and Ivan Sag. 1994. *Head Driven Phrase Structure Grammar*. CSLI/Chicago University Press, Chicago, IL, USA.

Pustejovsky, James, Sabine Berger, and Peter Anick. 1993. Lexical semantic techniques for corpus analysis. *Computational Linguistics*, 19(2):331–358.

Quirk, Randolph, Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik. 1985. *A Comprehensive Grammar of the English Language*. Longman, London, UK.

Ramshaw, Lance A. and Mitchell Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*, pages 82–94, Cambridge, MA, USA, June 30.

Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-2)*, pages 1–10, Providence, RI, USA, August 1–2.

Rehbein, Ines and Josef van Genabith. 2007. Treebank annotation schemes and parser evaluation for German. In *Proceedings of the 2007 Joint Conference on Empirical Methods in*

Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 630–639, Prague, Czech Republic, June 28–30.

Reinhart, Tanya. 1997. Quantifier scope: How labor is divided between QR and choice functions. *Linguistics and Philosophy*, 20(4):335–397.

Resnik, Philip. 1993. *Selection and Information: A Class-Based Approach to Lexical Relationships*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

Roark, Brian and Michiel Bacchiani. 2003. Supervised and unsupervised PCFG adaptation to novel domains. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL-03)*, pages 126–133, Edmonton, Canada, May 27–June 1.

Steedman, Mark. 1996. *Surface Structure and Interpretation*. MIT Press, Cambridge, MA, USA.

Steedman, Mark. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA, USA.

Sudo, Kiyoshi, Satoshi Sekine, and Ralph Grishman. 2003. An improved extraction pattern representation model for automatic IE pattern acquisition. In *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL-03)*, pages 224–231, Sapporo, Japan, July 7–12.

Vadas, David and James R. Curran. 2007a. Adding noun phrase structure to the Penn Treebank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL-07)*, pages 240–247, Prague, Czech Republic, June 25–27.

Vadas, David and James R. Curran. 2007b. Large-scale supervised models for noun phrase bracketing. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics (PACLING-07)*, pages 104–112, Melbourne, Australia, September 19–21.

Vadas, David and James R. Curran. 2007c. Parsing internal noun phrase structure with Collins' models. In *Proceedings of the Australasian Language Technology Workshop (ALTW-07)*, pages 109–116, Melbourne, Australia, December 10–11.

Vadas, David and James R. Curran. 2008. Parsing noun phrase structure with CCG. In *Proceedings of the 46th Annual Meeting of the Association of Computational Linguistics: Human Language Technologies (ACL-08: HLT)*, Columbus, OH, USA, June 15–20.

van Eynde, Frank. 2006. NP-internal agreement and the structure of the noun phrase. *Journal of Linguistics*, 42:139–186.

Wang, Wei, Kevin Knight, and Daniel Marcu. 2007. Binarizing syntax trees to improve syntax-based machine translation accuracy. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 746–754, Prague, Czech Republic, June 28–30.

Warner, Colin, Ann Bies, Christine Brisson, and Justin Mott. 2004. Addendum to the Penn Treebank II style bracketing guidelines: BioMedical Treebank annotation. Technical report, Linguistic Data Consortium.

Weischedel, Ralph and Ada Brunstein. 2005. BBN pronoun coreference and entity type corpus. Technical report. LDC Catalog No.: LDC2005T33.

Wood, Mary McGee. 1993. *Categorial Grammars*. Routledge, London, UK.

Xia, Fei. 1999. Extracting Tree Adjoining Grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, pages 398–403, Beijing, China, November.

Xue, Nianwen, Fei Xia, Fu-Dong Chiou, and Martha Palmer. 2005. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Yamada, Hiroyasu and Yuji Matsumoto. 2003. Statistical dependency analysis with Support Vector Machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206, Nancy, France.

Younger, Daniel. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Zhang, Hao, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the Human Language Technology Conference - North American Chapter of the Association for Computational Linguistics Annual Meeting (HLT-NAACL)*, pages 256–263, New York City, NY, USA, June 5–7.