Large-Scale Syntactic Processing: Parsing the Web Final Report of the 2009 JHU CLSP Workshop

Stephen Clark[*], Ann Copestake[*], James R. Curran[†],
Yue Zhang[‡], Aurelie Herbelot[*], James Haggerty[†], Byung-Gyu Ahn[§],
Curt Van Wyk[¶], Jessika Roesner[⋆], Jonathan Kummerfeld[†], Tim Dawborn[†]

[*]University of Cambridge, [†]University of Sydney, [‡]University of Oxford,
[§]Johns Hopkins University, [¶]Northwestern College, [⋆]University of Texas

October 30, 2009

Abstract

Scalable syntactic processing will underpin the sophisticated language technology needed for next generation information access. Companies are already using NLP tools to create web-scale question answering and "semantic search" engines. Massive amounts of parsed web data will also allow the automatic creation of semantic knowledge resources on an unprecedented scale. The web is a challenging arena for syntactic parsing, because of its scale and variety of styles, genres, and domains.

The goals of our workshop were to scale and adapt an existing wide-coverage parser to Wikipedia text; improve the efficiency of the parser through various methods of chart pruning; use self-training to improve the efficiency and accuracy of the parser; use the parsed wiki data for an innovative form of bootstrapping to make the parser both more efficient and more accurate; and finally use the parsed web data for improved disambiguation of coordination structures, using a variety of syntactic and semantic knowledge sources.

The focus of the research was the C&C parser (Clark and Curran, 2007c), a state-of-the-art statistical parser based on Combinatory Categorial Grammar (CCG). The parser has been evaluated on a number of standard test sets achieving state-of-the-art accuracies. It has also recently been adapted successfully to the biomedical domain (Rimell and Clark, 2009). The parser is surprisingly efficient, given its detailed output, processing tens of sentences per second. For web-scale text processing, we aimed to make the parser an *order of magnitude faster* still. The C&C parser is one of only very few parsers currently available which has the potential to produce detailed, accurate analyses at the scale we were considering.

Chapter 1

Introduction

Statistical parsing has been one of the success stories of the last 15 years of NLP research, producing robust parsers capable of accurately and efficiently analysing naturally occurring text. The creation of the Penn Treebank (PTB) (Marcus et al., 1993) was the catalyst for this research, leading to many parsers capable of producing syntactic parse trees in the style of the PTB for English newspaper text, at accuracies around 90% according to the Parseval measures; e.g. (Collins, 1997; Charniak, 2000; Bod, 2003; Petrov and Klein, 2007). A second strand of research has focused on producing dependency structures, a form of syntactic representation especially amenable to a wide range of languages, including those with freer word order than English; e.g. (Nivre et al., 2007; McDonald et al., 2005; Nivre and Scholz, 2004). Finally, there is a body of work extending parsing techniques for various linguistic formalisms, such as LFG, HPSG, TAG, and CCG, so that parsers based on these formalisms are as robust and efficient as their PTB counterparts. Typically this has been achieved by converting the PTB into a formalism-based resource, by converting the trees into the relevant analyses, or at least using the PTB as a source of training data for parse disambiguation models (Riezler et al., 2002; Sarkar and Joshi, 2003; Cahill et al., 2004; Miyao and Tsujii, 2005; Clark and Curran, 2007c).

Despite these successes, there are still barriers against the adoption of statistical parsers for large-scale NLP applications. The first is the fact that the performance of newspaper-trained parsers appears to degrade significantly when moved to another domain (Gildea, 2001). The second is that, whilst there has been some research on efficiency for wide-coverage parsing, the parsing speeds of popular off-the-shelf parsers, such as the Collins and Charniak parsers, are typically around one sentence per second.[1] One sentence per second is too slow for practical parsing of massive amounts of text, for example the approximately 1 billions words of text in Wikipedia.

For the applications/tasks we are targetting for wide-coverage parsing, high parsing speeds are essential. The first application is “Semantic search”, such as that performed by Powerset. Semantic search on the web requires all indexed web pages to be parsed,

[1] Parsing speeds are typically greater for dependency parsers, especially when using deterministic (or near-deterministic) shift-reduce style algorithms (Nivre and Scholz, 2004).

a feat which requires high parsing speeds even when massive amounts of computing power are available, because of the huge number of web pages. The second application is providing large amounts of parsed data for various types of knowledge acquisition, for example thesaurus construction (Curran, 2004) and relation extraction (Banko and Etzioni, 2008). Banko and Etzioni (2008) perform relation extraction on a massive scale, and yet, despite the availability of robust and accurate parsers, use simple finite-state methods to identify relevant linguistic relations. One of the barriers to using parsing technology for web-scale relation extraction is the slow speeds of the commonly used parsers (p.c. Oren Etzioni). Hence our main goal in the workshop was to develop a linguistically-motivated parser capable of accurately parsing sentences of web text at high speeds.

A key question was which parser to use. Various parsers were available, but we chose the Clark and Curran (2007c) parser for the following reasons:

- It is already relatively fast, parsing tens of newspaper sentences per second before the workshop;
- Its accuracy has been tested on a number of test sets, achieving scores comparable to the state-of-the-art for a number of output representations (Clark and Curran, 2007a, 2009; Rimell et al., 2009);
- It has already been successfully ported from newspaper text to the biomedical domain (Rimell and Clark, 2009);
- The parsing algorithm used by the parser is a standard chart parsing algorithm, CKY, and so the optimisations we develop will be applicable to many other chart parsers;
- The statistical models used by the parser are common to many other parsers and frameworks, and so the optimisations we develop will be widely applicable;
- The parser is based on Combinatory Categorial Grammar, a lexicalised grammar; any adaptation and optimisation techniques we develop will also be applicable to other lexicalised formalisms, such as Lexicalised Tree Adjoining Grammar;
- The two lead developers of the parser, Curran and Clark, were team leaders for the workshop.

The parser is described in Chapter 2; briefly, it works in two stages. First, there is the tagging stage, consisting of part-of-speech tagging using the standard POS tag set from the PTB, and CCG *supertagging* (Bangalore and Joshi, 1999; Clark and Curran, 2004), which is the task of assigning the correct lexical category type to each word in a sentence. In CCG, lexical categories typically express subcategorisation information and contain much more grammatical information than the POS tag set from the PTB; hence there is an order of magnitude more lexical categories than POS tags in the set used by the supertagger, making the tagging problem much harder. The solution to this difficult tagging problem is to use a multi-tagger which is allowed to assign more than one category when the context is not informative enough to determine a single category. Crucially, the number of lexical categories assigned is directly related to the speed of the parser: fewer categories leads to greater speeds.

The second stage of the parsing is the CKY chart parsing stage. Before the workshop, the parser *built the complete CKY chart* given the input lexical categories; i.e. there was no pruning at the chart parsing stage. During the workshop we focused on both improving the supertagging stage, by assigning fewer categories to each word, and pruning the chart, both of which led to significant speed increases with little loss in accuracy.

Another key question was which type of text to focus on. The web contains many different genres and text types, some of which are noisy, ungrammatical, and significantly removed from the newspaper text on which the CCG parser is based. We decided to focus on Wikipedia, since it is far enough from newspaper text to provide a challenge, and yet still contains much well-written and grammatical text. In addition, there is large interest in using Wikipedia as a semantic resource for NLP applications, as evidenced by the large number of recent papers on this topic and workshops at international conferences such as the ACL 2009 workshop on The People's Web meets NLP: Collaboratively Constructed Semantic Resources.

Given this background to the workshop, the research questions we investigated were as follows:

- How does a newspaper-trained lexicalised-grammar parser perform on Wikipedia text?
- Can we improve the accuracy of the parser on Wikipedia text?
- Can we improve the efficiency of the parser?
- Can we use large amounts of parsed data to improve coordination disambiguation?
- Can we relieve the annotation bottleneck by training on a novel form of bootstrapped Wikipedia data?

The first three questions relate directly to our goal of producing a linguistically-motivated parser capable of accurately parsing sentences of web text at high speeds. The fourth question emerged out of a desire to make use of the parser output during the workshop. And the fifth question was our "blue-sky" project looking at innovative and cheap ways of providing additional training data for the parser.

Given these research questions, our goals in the project were as follows:

- Produce a useful parser for any research community interested in parsing text, including a pipeline containing some pre-processing at the start of the parsing process;
- Produce a parsed version of Wikipedia, in multiple output formats;
- Investigate a number of research questions with respect to improving the accuracy and efficiency of a lexicalised-grammar chart parser applied to non-newspaper text.

Given the limited amount of time available during the workshop (6 weeks), we focused on the third goal, with the first two goals left as post-workshop commitments.

Given our goals and research questions, the workshop was organised into the following tasks (with the team members working on each task listed in brackets):

- Adapting the parser to Wikipedia text and accuracy evaluation (Stephen Clark, Jessi Roesner, Laura Rimell,[2] Matthew Honnibal).
- Pre-processing of raw HTML and potentially other formats, with a focus on the sentence boundary detection task (Curt van Wyk, James Curran).
- Application of self-training (McClosky et al., 2006) to the 2-stage supertagger and chart parser, with a focus on improving the *efficiency* of the parser through training the supertagger on parser output (Jonathan Kummerfeld, Jessi Roesner, James Curran).
- Improving the efficiency of the parser through various methods of pruning:
  - flexible beam search, in which low-scoring constituents are removed from a cell according to various criteria (Byung-Gyu Ahn, James Curran, Stephen Clark);
  - cell pruning, in which complete cells are removed before parsing begins based on the words and POS tags from the cell span (Yue Zhang, Stephen Clark, James Curran);
  - "1 parse per n-gram", in which pre-built derivations are entered into the chart based on frequently occurring n-grams in previously parsed text (Tim Dawborn, James Curran, Stephen Clark).
- Coordination disambiguation, in which various knowledge sources, including a distributional model of lexical semantics, are used to decide between alternative coordination structures posited by the parser (Aurelie Herbelot, Ann Copestake).
- A novel form of bootstrapping, in which short factual Wikipedia snippets which can be reliably parsed are used to provide constraints on how longer Wikipedia sentences can be parsed, giving a form of partial annotation for additional training data (James Haggerty, James Curran).

Given these tasks, the conclusions from the workshop were as follows:

- The newspaper-trained parser performs surprisingly well on Wikipedia text, obtaining accuracy scores on a grammatical relations-based evaluation comparable to scores obtained on newspaper text.

[2]Some of the people listed, e.g. Rimell and Honnibal, were not present during the workshop but collaborated with us remotely.

- Implementing a pre-processing tokenisation module using existing regular expression libraries, in particular Boost, is non-trivial. High accuracy sentence segmentation can be achieved on Wikipedia data using a rule-based segmentation module.
- Applying self-training to the supertagger and parser is effective in reducing the number of lexical categories that the supertagger assigns, thereby increasing parser speed with little or no loss in accuracy.
- Simple beam search, in which low scoring constituents are pruned from a cell, is highly effective in increasing parser speed with little or no loss in accuracy.
- Cell pruning using tagging techniques, in which a cell is pruned based on the words and POS tags from the span of the cell, is highly effective in increasing parser speed with little or no loss in accuracy.
- The parser's performance on coordination constructions can be improved using a combination of knowledge sources, in particular a distributional lexical semantics model induced from parser output.
- Pre-parsed n-grams show some promise in increasing the speed of the parser, but more investigation is needed to handle ambiguity (and the implementation is non-trivial).
- "Fact redundancy" shows some promise in providing additional training data for the parser, but more investigation is needed into how to select the factoids and how to constrain analyses based on previously analysed factoids.

The rest of the report is organised according to the tasks, with the first two tasks forming Chapter 3, and each remaining task forming a separate chapter. Chapter 2 is a description of the CCG parser.

Chapter 2

Wide-Coverage CCG Parsing

We provide a description of the CCG parser in order to make the report self-contained. This description is taken largely from Clark and Curran (2007c).

2.1 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) (Steedman, 1996, 2000) is a type-driven lexicalised theory of grammar based on categorial grammar (Wood, 1993). CCG lexical entries consist of a syntactic category, which defines valency and directionality, and a semantic interpretation. In this report we are concerned largely with the syntactic component; see Steedman (2000) for how a semantic interpretation can be composed during a syntactic derivation, and also Bos et al. (2004) for how semantic interpretations can be built for newspaper text using the Clark and Curran (2007c) parser.

Categories can be either basic or complex. Examples of basic categories are S (sentence), N (noun), NP (noun phrase) and PP (prepositional phrase). Complex categories are built recursively from basic categories, and indicate the type and directionality of arguments (using slashes), and the type of the result. For example, the following category for the transitive verb *bought* specifies its first argument as a noun phrase to its right, its second argument as a noun phrase to its left, and its result as a sentence:

$$\text{bought} := (S\backslash NP)/NP \tag{2.1}$$

Categories are combined in a derivation using *combinatory rules*. In the original Categorial Grammar (Bar-Hillel, 1953), which is context-free, there are two rules of *functional application*:

$$X/Y \;\; Y \;\Rightarrow\; X \quad (>) \tag{2.2}$$

$$Y \;\; X\backslash Y \;\Rightarrow\; X \quad (<) \tag{2.3}$$

where X and Y denote categories (either basic or complex). The first rule is *forward application* ($>$) and the second rule is *backward application* ($<$). Figure 2.1 gives an example derivation using these rules.

$$
\begin{array}{ccccccc}
\text{Investors} & \text{are} & \text{appealing} & \text{to} & \text{the} & \text{Exchange} & \text{Commission} \\
\overline{NP} & \overline{(S[dcl]\backslash NP)/(S[ng]\backslash NP)} & \overline{(S[ng]\backslash NP)/PP} & \overline{PP/NP} & \overline{NP/N} & \overline{N/N} & \overline{N} \\
 & & & & & \multicolumn{2}{c}{\overline{\qquad\qquad N \qquad\qquad}{>}} \\
 & & & & \multicolumn{3}{c}{\overline{\qquad\qquad NP \qquad\qquad}{>}} \\
 & & & \multicolumn{4}{c}{\overline{\qquad\qquad PP \qquad\qquad}{>}} \\
 & & \multicolumn{5}{c}{\overline{\qquad\qquad S[ng]\backslash NP \qquad\qquad}{>}} \\
 & \multicolumn{6}{c}{\overline{\qquad\qquad S[dcl]\backslash NP \qquad\qquad}{>}} \\
\multicolumn{7}{c}{\overline{\qquad\qquad S[dcl] \qquad\qquad}{<}}
\end{array}
$$

Figure 2.1: Example derivation using forward and backward application

CCG extends the original Categorial Grammar by introducing a number of additional combinatory rules. The first is *forward composition*, which Steedman denotes by $> \mathbf{B}$ (since $\mathbf{B}$ is the symbol used by Curry to denote function composition in combinatory logic (Curry and Feys, 1958)):

$$X/Y \quad Y/Z \quad \Rightarrow_{\mathbf{B}} \quad X/Z \quad (> \mathbf{B}) \tag{2.4}$$

Forward composition is often used in conjunction with *type-raising* ($\mathbf{T}$), as in Figure 2.2. In this case type-raising takes a subject noun phrase and turns it into a functor looking to the right for a verb phrase; *the fund* is then able to combine with *reached* using forward composition, giving *the fund reached* the category $S[dcl]/NP$ (a declarative sentence missing an object). It is exactly this type of constituent which the object relative pronoun category is looking for to its right: $(NP\backslash NP)/(S[dcl]/NP)$.

Note that *the fund reached* is a perfectly reasonable constituent in CCG, having the type $S[dcl]/NP$. This allows analyses for sentences such as *the fund reached but investors disagreed with the agreement*, even though this construction is often described as "non-constituent coordination". In this example, *the fund reached* and *investors disagreed with* have the same type, allowing them to be coordinated, resulting in *the fund reached but investors disagreed with* having the type $S[dcl]/NP$. Note also that it is this flexible notion of constituency which leads to so-called spurious ambiguity, since even the simple sentence *the fund reached an agreement* will have more than one derivation, with each derivation leading to the same set of predicate-argument dependencies.

Forward composition is generalised to allow additional arguments to the right of the Z category in (2.4). For example, the following combination allows analysis of sentences such as *I offered, and may give, a flower to a policeman* (Steedman, 2000):

$$
\begin{array}{cc}
\text{may} & \text{give} \\
\overline{(S\backslash NP)/(S\backslash NP)} & \overline{((S\backslash NP)/PP)/NP} \\
\multicolumn{2}{c}{\overline{\qquad\qquad ((S\backslash NP)/PP)/NP \qquad\qquad}{>\mathbf{B}}}
\end{array}
$$

This example shows how the categories for *may* and *give* combine, resulting in a category of the same type as *offered*, which can then be coordinated. Steedman (2000) gives a more precise definition of generalised forward composition.

$$
\begin{array}{cccccc}
\text{the} & \text{agreement} & \text{which} & \text{the} & \text{fund} & \text{reached}\\
\hline
NP/N & N & (NP\backslash NP)/(S[dcl]/NP) & NP/N & N & (S[dcl]\backslash NP)/NP\\
\cline{1-2}\cline{4-5}
\multicolumn{2}{c}{NP\;>} & & \multicolumn{2}{c}{NP\;>} & \\
\cline{4-5}
 & & & \multicolumn{2}{c}{S/(S\backslash NP)\;>\mathbf{T}} & \\
\cline{4-6}
 & & & \multicolumn{3}{c}{S[dcl]/NP\;>\mathbf{B}}\\
\cline{3-6}
 & & \multicolumn{4}{c}{NP\backslash NP\;>}\\
\hline
\multicolumn{6}{c}{NP\;<}
\end{array}
$$

Figure 2.2: Example derivation using type-raising and forward composition

Further combinatory rules in the theory of CCG include backward composition ($< \mathbf{B}$) and backward crossed composition ($< \mathbf{B}_\times$):

$$Y\backslash Z \quad X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X\backslash Z \qquad (< \mathbf{B}) \tag{2.5}$$

$$Y/Z \quad X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X/Z \qquad (< \mathbf{B}_\times) \tag{2.6}$$

Backward composition provides an analysis for sentences involving “argument cluster coordination”, such as *I gave a teacher an apple and a policeman a flower* (Steedman, 2000). Backward crossed composition is required for heavy NP shift and coordinations such as *I shall buy today and cook tomorrow the mushrooms*. In this coordination example from Steedman (2000), backward crossed composition is used to combine the categories for *buy*, $(S\backslash NP)/NP$, and *today*, $(S\backslash NP)\backslash(S\backslash NP)$, and similarly for *cook* and *tomorrow*, producing categories of the same type which can be coordinated. This rule is also generalised in an analogous way to forward composition.

Finally, there is a coordination rule which conjoins categories of the same type, producing a further category of that type. This rule can be implemented by assuming the following category schema for a coordination term: $(X\backslash X)/X$, where X can be any category. All of the combinatory rules described above are implemented in the parser.

One way of dealing with the additional ambiguity in CCG is to only consider *normal-form* derivations. Informally, a normal-form derivation is one which uses type-raising and composition only when necessary. Eisner (1996) describes a technique for eliminating spurious ambiguity entirely, by defining exactly one normal-form derivation for each semantic equivalence class of derivations. The idea is to restrict the combination of categories produced by composition; more specifically, any constituent which is the result of a forward composition cannot serve as the primary (left) functor in another forward composition or forward application. Similarly, any constituent which is the result of a backward composition cannot serve as the primary (right) functor in another backward composition or backward application. Eisner only deals with a grammar without type-raising, and so the constraints cannot guarantee a normal-form derivation when applied to the grammars used in this paper. However, the constraints can still be used to significantly reduce the parsing space, and are implemented in the parser.

2.1.1 Why use CCG *for statistical parsing?*

CCG was designed to deal with the long-range dependencies inherent in certain constructions, such as coordination and extraction, and arguably provides the most linguistically satisfactory account of these phenomena. Long-range dependencies are relatively common in text such as newspaper text, but are typically not recovered by treebank parsers such as Collins (2003) and Charniak (2000). This has led to a number of proposals for post-processing the output of the Collins and Charniak parsers, in which trace sites are located and the antecedent of the trace determined (Johnson, 2002; Dienes and Dubey, 2003; Levy and Manning, 2004). An advantage of using CCG is that the recovery of long-range dependencies can be integrated into the parsing process in a straightforward manner, rather than be relegated to such a post-processing phase (Clark et al., 2002; Hockenmaier, 2003; Clark et al., 2004). Rimell et al. (2009) evaluates how well the CCG parser is able to recover a variety of long-range dependency types in naturally occurring text and provides a comparison with a number of popular parsers.

Another advantage of CCG is that providing a compositional semantics for the grammar is relatively straightforward. It has a transparent interface between syntax and semantics and, since CCG is a lexicalised grammar formalism, providing a compositional semantics simply involves adding semantic representations to the lexical entries and interpreting the small number of combinatory rules. Bos et al. (2004) show how this can be done in practice.

Of course some of these advantages could be obtained with other grammar formalisms, such as TAG, LFG and HPSG, although CCG is especially well-suited to analysing coordination and long-range dependencies. For example, the analysis of "non-constituent coordination" described in the previous section is, as far as we know, unique to CCG.

Finally, the lexicalised nature of CCG has implications for the engineering of a wide-coverage parser. The use of a *supertagger* (Bangalore and Joshi, 1999) prior to parsing can produce an extremely efficient parser. The supertagger uses statistical sequence tagging techniques to assign a small number of lexical categories to each word in the sentence. Since there is so much syntactic information in lexical categories, the parser is required to do less work once the lexical categories have been assigned; hence Srinivas and Joshi, in the context of TAG, refer to supertagging as *almost parsing*. The supertagger forms a key component for many of the research questions investigated in this report.

2.1.2 CCGbank

The treebank on which the parser is based is CCGbank (Hockenmaier and Steedman, 2007), a CCG version of the Penn Treebank (Marcus et al., 1993). The treebank performs two roles in building the parser: it provides the lexical category set used by the supertagger, plus some unary type-changing rules and punctuation rules used by the parser, and it is used as training data for the statistical models. Penn Treebank conversions have also been carried out for other linguistic formalisms, including TAG (Chen and Vijay-Shanker, 2000; Xia et al., 2000), LFG (Burke et al., 2004) and HPSG (Miyao et al., 2004).

CCGbank was created by converting the phrase-structure trees in the Penn Treebank into CCG normal-form derivations. Some pre-processing of the phrase-structure trees was required, in order to allow the correct CCG analyses for some constructions, such as coordination. Hockenmaier (2003) gives a detailed description of the procedure used to create CCGbank.

Sentence categories (S) in CCGbank carry features, such as $[dcl]$ for declarative, $[wq]$ for *wh*-questions, $[for]$ for small clauses headed by *for*; see Hockenmaier (2003) for the complete list. S categories also carry features in verb-phrases; for example, $S[b]\backslash NP$ is a bare-infinitive; $S[to]\backslash NP$ is a to-infinitive; $S[pss]\backslash NP$ is a past participle in passive mode.

As well as instances of the standard CCG combinatory rules — forward and backward application, forward and backward composition, backward-crossed composition, type-raising, coordination of like types — CCGbank contains a number of unary *type-changing* rules and rules for dealing with punctuation. The type-changing rules typically change a verb phrase into a modifier. The following examples, taken from Hockenmaier (2003), demonstrate the most common rules. The bracketed expression has the type-changing rule applied to it:

- $S[pss]\backslash NP \Rightarrow NP\backslash NP$
  workers [exposed to it]
- $S[adj]\backslash NP \Rightarrow NP\backslash NP$
  a forum [likely to bring attention to the problem]
- $S[ng]\backslash NP \Rightarrow NP\backslash NP$
  signboards [advertising imported cigarettes]
- $S[ng]\backslash NP \Rightarrow (S\backslash NP)\backslash(S\backslash NP)$
  became chairman [succeeding Ian Butler]
- $S[dcl]/NP \Rightarrow NP\backslash NP$
  the millions of dollars [it generates]

2.1.3 CCG *Dependency Structures*

Dependency structures are one of the main output representations for the parser. One of their keys uses is parser evaluation: the accuracy of a parsing model is measured using precision and recall over CCG predicate-argument dependencies. They are also used as features in some of the models described in Clark and Curran (2007c).

CCG predicate-argument relations are defined in terms of the argument slots in CCG lexical categories. Thus the transitive verb category, $(S\backslash NP)/NP$, has two predicate-argument relations associated with it, one corresponding to the object NP argument and one corresponding to the subject NP argument. In order to distinguish different argument slots, the arguments are numbered from left to right. Thus, the subject relation for a transitive verb is represented as $\langle (S\backslash NP_1)/NP_2, 1\rangle$.

The predicate-argument dependencies are represented as 5-tuples: $\langle h_f, f, s, h_a, l\rangle$, where h_f is the lexical item of the lexical category expressing the dependency relation; f is the lexical category; s is the argument slot; h_a is the head word of the argument; and

l encodes whether the dependency is non-local. For example, the dependency encoding *company* as the object of *bought* (as in *IBM bought the company*) is represented as follows:

$$\langle bought_2,\ (S\backslash NP_1)/NP_2,\ 2,\ company_4,\ -\rangle \tag{2.7}$$

The subscripts on the lexical items indicate sentence position, and the final field ($-$) indicates that the dependency is a local dependency.

Head and dependency information is represented on the lexical categories, and dependencies are created during a derivation as argument slots are filled. Long-range dependencies are created by passing head information from one category to another using unification. For example, the expanded category for the control verb *persuade* is:

$$\text{persuade} := ((S[dcl]_{persuade}\backslash NP_1)/(S[to]_2\backslash NP_X))/NP_{X,3} \tag{2.8}$$

The head of the infinitival complement's subject is identified with the head of the object, using the variable X. Unification then passes the head of the object to the subject of the infinitival, as in standard unification-based accounts of control.

The kinds of lexical items that use the head passing mechanism are raising, auxiliary and control verbs, modifiers, and relative pronouns. Among the constructions that project unbounded dependencies are relativisation and right node raising. The following relative pronoun category (for words such as *who, which, that*) shows how heads are co-indexed for object-extraction:

$$\text{who} := (NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X) \tag{2.9}$$

In a sentence such as *The company which IBM bought*, the co-indexing will allow *company* to be returned as the object of *bought*, which is represented using the following dependency:

$$\langle bought_2,\ (S\backslash NP_1)/NP_2,\ 2,\ company_4,\ (NP\backslash NP)/(S[dcl]/NP)\rangle \tag{2.10}$$

The final field indicates the category which mediated the long-range dependency, in this case the object relative pronoun category.

The dependency annotation also permits complex categories as arguments. For example, the marked up category for *about* (as in *about 5,000 pounds*) is:

$$(N_X/N_X)_Y/(N/N)_{Y,1} \tag{2.11}$$

If *5,000* has the category $(N_X/N_X)_{5,000}$, the dependency relation marked on the $(N/N)_{Y,1}$ argument in (2.11) allows the dependency between *about* and *5,000* to be captured.

In the current implementation every argument slot in a lexical category corresponds to a dependency relation. This means, for example, that the parser produces subjects of to-infinitival clauses and auxiliary verbs. In the sentence *IBM may like to buy Lotus*, *IBM* will be returned as the subject of *may*, *like*, *to* and *buy*. The only exception is during evaluation, when some of these dependencies are ignored in order to be consistent with the predicate-argument dependencies in CCGbank, and also DepBank.

2.2 Parsing Models for CCG

Clark and Curran (2007c) describes a number of discriminative log-linear (or maximum entropy) parsing models for CCG. The model that we used in the workshop is a log-linear model defined over the normal-form derivations in CCGbank. That is, the model is designed to score the correct normal-form derivation for a sentence more highly than any of the alternative derivations. Clark and Curran (2007c) also describes a model over dependency structures, in which the probabilities of all derivations for a structure, including the non-normal-form derivations, are summed; however, the accuracy of the normal-form model is found to be close to that of the dependency model, with the advantage that the normal-form model is easier to work with.

Clark and Curran (2007b) defines a perceptron model for normal-form CCG derivations, based on work by Collins (Collins, 2002; Collins and Roark, 2004), which is shown to perform as well as the log-linear model. The advantage of the perceptron is that it is easy to train, simply requiring a decoding of the training examples and a trivial update procedure. It also has the advantage that the training can be performed on a single machine, whereas the log-linear training required a parallel implementation of the training algorithm running on a cluster.

2.3 The Supertagger

Parsing with lexicalised grammar formalisms such as CCG is a two-step process: first, elementary syntactic structures — in CCG's case lexical categories — are assigned to each word in the sentence, and then the parser combines the structures together. The first step can be performed by simply assigning to each word all lexical categories the word is seen with in the training data, together with some strategy for dealing with rare and unknown words (such as assigning the complete lexical category set) (Hockenmaier, 2003). Since the number of lexical categories assigned to a word can be high, some strategy is needed to make parsing practical; Hockenmaier (2003), for example, uses a beam search to discard chart entries with low scores.

An alternative approach is to use a supertagger (Bangalore and Joshi, 1999) to perform step one. Clark and Curran (2004) describes a CCG supertagger, which uses log-linear models to define a distribution over the lexical category set for each local 5-word context containing the target word (Ratnaparkhi, 1996). The features used in the models are the words and POS tags in the 5-word window, plus the two previously assigned lexical categories to the left. The conditional probability of a sequence of lexical categories, given a sentence, is then defined as the product of the individual probabilities for each category. The most probable lexical category sequence can be found efficiently using a variant of the Viterbi algorithm for HMM taggers.

The lexical category set used by the supertagger is described in Clark and Curran (2004) and Curran et al. (2006). It includes all lexical catgeories which appear at least 10 times in Sections 02-21 of CCGbank, resulting in a set of 425 categories. Clark and Curran (2004) shows this set to have very high coverage on unseen data.

The accuracy of the supertagger on Section 00 of CCGbank is 92.6%, with a sentence accuracy of 36.8%. Sentence accuracy is the percentage of sentences whose words are all tagged correctly. These figures include punctuation marks, for which the lexical category is simply the punctuation mark itself, and are obtained using gold standard POS tags. With automatically assigned POS tags, using the POS tagger of Curran and Clark (2003), the accuracies drop to 91.5 and 32.5. An accuracy of 91-92% may appear reasonable given the large lexical category set; however, the low sentence accuracy suggests that the supertagger may not be accurate enough to serve as a front-end to a parser. Clark (2002) reports that a significant loss in coverage results if the supertagger is used as a front-end to the parser of Hockenmaier and Steedman (2002). In order to increase the number of words assigned the correct category, a CCG multitagger is used, which is able to assign more than one category to each word. Clark and Curran (2007c) describes how the multitagger uses the forward-backward algorithm to calculate the probabilities of lexical categories given a complete lattice of alternative tag sequences, which are then used to assign sets of categories to words. The multitagger has a per-word accuracy of almost 98% with a lexical category ambiguity of only 1.4 categories per word on average.

2.4 Parser and Decoder

The parser uses the CKY algorithm (Kasami, 1965; Younger, 1967) described in Steedman (2000) to create a packed chart. The CKY algorithm applies naturally to CCG since the grammar is binary. It builds the chart bottom-up, starting with constituents spanning a single word, incrementally increasing the span until the whole sentence is covered. Since the constituents are built in order of span size, at any point in the process all the sub-constituents which could be used to create a particular new constituent must be present in the chart. Hence dynamic programming can be used to prevent the need for backtracking during the parsing process.

The chart is *packed* in the sense that any two equivalent constituents created during the parsing process are placed in the same equivalence class, with pointers to the children used in the creation. Equivalence is defined in terms of the category and head of the constituent; essentially so that the Viterbi algorithm can efficiently the find the highest scoring derivation.[1] The Viterbi algorithm can be used to find the most probable derivation from a packed chart. For each equivalence class in the chart, we record the individual entry at the root of the subderivation which has the highest score for the class. The equivalence classes are defined so that any other individual entry cannot be part of the highest scoring derivation for the sentence. The highest-scoring subderivations can be calculated recursively using the highest-scoring equivalence classes that were combined to create the individual entry in the chart.

Sometimes the parser is unable to build an analysis which spans the whole sentence. When this happens the parser and supertagger interact in the following fashion: the parser effectively asks the supertagger to provide more lexical categories for each word. This potentially continues for a number of iterations until the parser does create a spanning analysis, or else it gives up and moves to the next sentence.

[1]Use of the Viterbi algorithm in this way requires the features in the parser model to be local to a single rule application; Clark and Curran (2007c) has more discussion of this issue.

2.5 Evaluation

The accuracy of the parser has been evaluated using a variety of representations and test sets. The "native" evaluation of the parser is against the CCG dependencies in CCGbank. Here the parser obtains an F-score of 85.5% on the labelled dependencies in Section 23.

The parser has also been evaluated on the grammatical relations in Depbank (Briscoe and Carroll, 2006). Clark and Curran (2007a) shows this evaluation to be surprisingly difficult to perform, because of the difficulties of mapping from CCG dependencies to GRs, and for that reason the upper bound for the evaluation is surprisingly low. However, a mapping was implemented and the parser obtains a labelled F-score of 83.4% on this test set (Vadas and Curran, 2008), which is competitive with the state-of-the-art.

One question that is often asked of the CCG parser is how well it compares with the popular Penn Treebank parsers, such as Collins (1999) and Charniak (2000). Clark and Curran (2009) describes an experiment in which the CCG derivations output by the parser are converted to Penn Treebank trees. Again, the conversion is surprisingly difficult to perform, but a reliable conversion is possible for around 40% of the sentences in the test set, and these sentences are shown to be of a reasonable length (around 18 words). For this test set, the performance of the CCG parser is statistically no different to the Berkeley parser (Petrov and Klein, 2007).

Finally, Rimell et al. (2009) describes a test set of 700 sentences consisting of a variety of unbounded dependencies. The performance of the CCG parser at recovering these dependencies is compared with a number of other popular parsers. Along with the Enju parser (Miyao and Tsujii, 2004), the CCG parser is clearly the best performer on this test set. However, one of the main results of the paper is that all parsers perform poorly at unbounded dependency recovery, with the best performance only at around the 50% mark overall.

Chapter 3

Adapting to Wikipedia

This chapter describes our evaluation of the CCG parser on Wikipedia text, plus some experiments we performed on pre-processing, in particular sentence segmentation.

3.1 Parser Adaptation and Evaluation

It is perceived wisdom in NLP that WSJ-trained parsers perform badly on domains outside of newspaper text, and there is some experimental evidence for this, e.g. Gildea (2001). There is some previous work on adapting the CCG parser to other domains, in particular for biomedical text and questions for a QA system (Rimell and Clark, 2009, 2008).[1]

Rimell and Clark (2008) propose a parser adaptation technique based on the lexicalised nature of CCG, arguing that manually created training data can be produced relatively cheaply at the lexical category level, and that data at this level provides a large amount of syntactic information because of the detailed nature of the categories. They found that retraining the supertagger on 1,000 sentences from the new domain, annotated with lexical categories, plus the original data from CCGbank, was enough to significantly improve parser performance in both the biomedical and question domains. In fact, for the biomedical domain, most of the improvement came from simply retraining a POS tagger, rather than the supertagger, which was explained by the fact that biomedical text contains a large number of complex noun phrases, which are only recognised as such by a biomedically-trained POS tagger. For the questions, the lexical category data was more important, which was explained by the fact that the syntax of questions is different to the largely declarative sentences seen in WSJ text. We performed a similar porting experiment for Wikipedia text, by manually annotating 1,000 sentences from Wikipedia with CCG lexical categories and POS tags. (In practice, the manual annotation was performed by correcting the output of the WSJ-trained parser, simply using a text editor to make the corrections.)

In order to test the parser on Wikipedia text, we needed some manually annotated

[1] Questions do not strictly form a domain, but are nonetheless interesting as a parser adaptation problem because of the lack of questions in the Penn Treebank and the difference in syntax between questions and the declarative sentences making up the majority of the WSJ.

```
(ncmod num hundred_1 Seven_0)
(conj and_2 sixty-one_3)
(conj and_2 hundred_1)
(dobj in_6 total_7)
(ncmod _ made_5 in_6)
(aux made_5 were_4)
(ncsubj made_5 and_2 obj)
(passive made_5)
<c> Seven|CD|N/N hundred|CD|N and|CC|conj sixty-one|CD|N
were|VBD|(S[dcl]\NP)/(S[pss]\NP) made|VBN|S[pss]\NP
in|IN|((S\NP)\(S\NP))/NP total|NN|N .|.|.
```

Figure 3.1: Example Wikipedia test sentence annotated with grammatical relations. The sentence is at the bottom with POS tags and CCG lexical categories. The indices on the words indicate sentence position.

test data. The test data would be used to measure how well the WSJ-trained parser performs, and whether the adaptation techniques described above improve performance, but also used throughout the rest of the report to measure whether any of our innovations, such as the speed improvements, have a negative effect on accuracy.

Parser evaluation has generated a large literature. The standard evaluation metrics for parsers trained and tested on the Penn Treebank are the Parseval metrics (Black et al., 1991). However, the applicabilty of Parseval for evaluating CCG derivations has been questioned because of the binary-branching nature of the grammar (Hockenmaier, 2003). In addition, the Parseval metrics have been criticised as being unsuitable for general parser evaluation, with dependency-based evaluations emerging as the most promising candidate for a parser- and grammar-neutral evaluation framework (Lin, 1995; Carroll et al., 1998). Dependency-based evaluations are not without their problems, either; in particular there is usually the need for a mapping from the grammar-dependent representation used by a parser to the dependency representation used in the evaluation scheme, and this mapping can be difficult to perform (Clark and Curran, 2007a). But we still contend that, of the methods currently available, a dependency-based evaluation is the most appropriate for evaluating the CCG parser on Wikipedia text.

3.1.1 Data

We created two substantial data sets. The first is a randomly-chosen 1,000-sentence subset of Wikipedia manually annotated with CCG lexical categories. The main purpose for this dataset was to retrain the CCG supertagger for adaptation purposes. The annotation was performed by Stephen Clark and Laura Rimell by correcting the output of the WSJ-trained parser. Most of the annotation was completed during the 6 weeks of the workshop.

The second data set is a 300-sentence subset of the 1,000 sentences described above, annotated with grammatical relations (GRs) in the style of Briscoe and Carroll (2006).

P%	R%	F%
83.4	81.4	82.4

Table 3.1: Accuracy of the CCG parser on the 1,000 sentence subset of Wikipedia.

Again this was created by manually correcting the output of the WSJ-trained parser, and again the annotation was performed by Stephen Clark and Laura Rimell during the 6 weeks of the workshop. An example test sentence with GRs is given in Figure 3.1.

A subtle issue relating to the applicability of the GR test set arises from the fact that we corrected the output of the CCG parser, which is the parser that will be tested. Even though we attempted to correct all the errors made by the parser, there is still an inherent bias in the test set towards the output of the parser originally used to create it. Of course this problem arises for any evaluation resource based on the original output of a parser, but is not typically discussed in the literature. This bias is not a problem for the evaluations in this report which test different configurations of the parser, but it does mean that the overall test scores given in Section 3.1.2 must be interpreted in the light of this bias.

3.1.2 Results

Table 3.1 gives the performance of the parser, in terms of precision (P), recall (R) and balanced F-score (F), on the test set. Automatically assigned POS tags were used from the Curran and Clark (2003) tagger. To situate these results with respect to other GR evaluations, the CCG parser achieves an F-score of 83.4 on the Depbank test set (Vadas and Curran, 2008), which is a subset of Section 23 of the Penn Treebank annotated with the same style of GRs. 83.4 is competitive with the state-of-the-art on this test set.[2] Hence the 82.4 in Table 3.1 shows that the parser is performing well on Wikipedia data, although any direct comparison is difficult because of the different methods used to create the two test sets, as dicsussed above. Table 3.2 gives the parser accuracy for some of the frequent GR types. Again these numbers compare favourably with those obtained on Depbank (Clark and Curran, 2007c).

Following Rimell and Clark (2008), we retrained the POS tagger and supertagger on the 1,000-sentence subset of Wikipedia annotated with lexical categories. We also included the 40,000 sentences from CCGbank, and, in order that the newspaper data did not overwhelm the Wiki data, we included 10 copies of the 1,000 Wikipedia sentences. The somewhat surprising result was that no improvement was observed from this experiment. Given that the WSJ-trained parser is already performing at a high level on Wikipedia data, we take this as preliminary evidence that the parser is not so heavily tuned to WSJ text as has been suggested for some other Penn Treebank-based parsers; however, more careful experimentation and comparison is required to confirm this result.

[2]Scores for GR-based evaluations are typically lower than Parseval scores on the Penn Treebank, since GR-based evaluations are more difficult than the constituent matching used in Parseval.

GR	P%	R%	Freq
ncsubj	78	81	550
dobj	85	85	1,034
iobj	89	87	319
ccomp	80	71	75
xcomp	89	87	202
conj	80	75	482
ncmod	81	76	1,852
cmod	65	68	100
xmod	43	53	107

Table 3.2: Parser accuracy for some of the frequent GR types. Freq is the frequency of occurrence of the GR types in the test set.

3.2 Pre-Processing Pipeline

It was our intention at the start of the workshop to provide a full parsing pipeline, taking files in various formats as input and the existing varieties of parse representations as output. Currently the C&C parser requires the input document to be carefully processed, for example only allowing a single space between tokens (for efficiency reasons). Hence it would improve the usability of the parser to provide some pre-processing, including full processing of, for example, an HTML document into tokenised and segmented input ready for parsing.

In order to integrate the pipeline with the C&C tools, as well as provide an efficient solution to the tokenization task, tokenization was performed in C++. Furthermore, importance was placed on having dynamic tokenization and Unicode support. In order to attain these goals, we initially investigated the use of Boost Regex and Boost Xpressive. Xpressive's ability to dynamically track what part of the regular expression matched, where it matched, and associate actions to be performed with matches, was encouraging. However, as we progressed, we found the tools were not suitable for our purposes. Having the same tokenizing ability as the pre-existing Lex tokenizer required numerous complex regular expressions (a small number of which are shown in Figure 3.2). In addition, there was a problem with stack overflow: the length of the document and the number of regular expressions we required regularly led to stack overflow for our documents, which ultimately prevented us from using Boost Regex or Boost Xpressive. Consequently, we shifted our focus to the sentence boundary detection part of the pre-processing task. We decided to adapt Punkt, which is based on Kiss and Strunk (2006), from the NLTK toolkit by providing a C++ implementation.

When determining if a full stop is a sentence boundary, Punkt will consider likely collocations and abbreviations, the orthographic context of the following word, and frequent sentence starters. Punkt first looks to see if the word before a period is an abbreviation, part of an ellipse, or an initial. These are all evidence against a sentence boundary. Then Punkt checks if there is a collocation between the word that precedes the full stop

```
(?>"(?=\b)) STARTQUOTE
(?>(?<=\b)") ENDQOUTE
(?>\.{3}) ELIPSSIS
(?>(?<=\b)[A-Za-z](\.[A-Za-z])+\.|[A-Z][bcdfghj-np-tvxz]+\.
|[A-Z]{2,}\.|[A-Z][A-Z&]*[A-Z](?=\b)) ACRON
(?>(?<=\b)[A-Z]\.(?=\b)) INITIAL
(?>(?<=\b)[\w-]+@\w+\.\w+(?=\b)) E-MAIL_WORD
(?>(?<=\b)\w+-\w+(?<=\b)) HYPHEN_WORD
(?>[][(){}<>]) BRACKET
(?>(?<=\b)---+|==+|\*\*+|\.\.\.\.+|(-\s){3,}-?(?=\b)) LINES
(?>(?<=\b)[A-Z]*\$(?=\b)) DOLLARS
(?>(?<=\b)[0-9]|[0-9][0-9,]*[0-9](?=\b)) INTEGERS
(?>(?<=\b)[+-]?[0-9][0-9,]*\.[0-9]+|\.[0-9]+(?=\b)) FLOATS
(?>(?<=\b)([0-9]+-)?[0-9]+\/[0-9]+(?=\b)) FRACTION
(?>(?<=\b)[0-9]{1,2}[-/.][0-9]{1,2}[-/.][0-9]{2}([0-9]{2})?(?=\b)) DATE
(?>(?<=\b)[0-9]{1,2}([:.][0-9]{2})?(?=\b)) TIME
(?>(?<=\b)([ap]m|[AP]M)(?=\b)) TIME
```

Figure 3.2: A small number of the regular expressions used for tokenisation

and the one that follows; such evidence would imply that it is not a sentence boundary. If the stop happened to be an abbreviation or an ellipse and not an initial, then the orthographic context can be used to confirm that it indeed is a boundary. If there is not orthographic evidence for a boundary, then the following word will be checked to see if it is a frequent sentence starter to confirm the boundary. Finally, if the previous token is an initial or an ordinal number, the orthographic context is checked to try to deny the decision to classify it as a sentence boundary.

We followed this general outline to devise a solution in C++ to decide if a full stop was a sentence boundary. We did, however, change the way the full stops and tokens surrounding them were found. We went through the entire document looking for full stops and then looked to the left and to the right as opposed to the regular expression-based searching implemented by Punkt. Tokens were delimited by spaces, digits, and special characters (i.e. , ; : - ( ) [ ]   & # @ *) since these characters cannot start words. If these characters were to start the second token, then they themselves would be the second token. This implementation allowed us to catch cases Punkt would miss, such as *Mt. Fuji.*

We compared our implementation of the sentence boundary detection with that of Punkt on Sections 02-21 of the Penn Treebank. We performed the test by concatenating the sentences into one string and reinserting the sentence boundaries. This test had 39,604 sentences and 63,096 full stops. The most noticeable result was that of the time saved. Our sentence boundary detector was able to process the 63,096 full stops in 0.7 seconds, as opposed to the 24.1 seconds taken by Punkt. Accuracy, defined as the number of correct full stop classifications out of 63,096, was also improved: Punkt had 95.5%

accuracy, whereas our implementation had 97.0% accuracy. We were able to increase the accuracy further by adding months into the list of likely abbreviations, which brought the accuracy to 98.2%, and by realigning the periods that were followed by quotations, yielding an accuracy of 98.5%.

Hence, in conclusion, we were able to implement a highly efficient and accurate sentence boundary detection module using, and building on, the heuristics from the existing Punkt tool.

Chapter 4

Large-Scale Supertagging and Self-Training

The CCG parser is a 2-stage parsing process: first, the supertagger assigns lexical categories to the words in the sentence, and then the parsing algorithm combines them together. An obvious question is whether we can exploit the nature of this 2-stage process to increase the accuracy and/or the efficiency of the parser. Steedman et al. (2002) found that, when a CCG supertagger was trained on a small amount of initial data, it was possible to improve performance of the supertagger simply by training on output from a CCG parser. This type of *self-training* is used by McClosky et al. (2006) who were able to improve the accuracy of a 2-stage parser and reranker by simply training the parser on large amounts of output from the reranker.

We also investigated whether self-training could improve the *efficiency* of the parser. The speed of the parser depends crucially on the lexical categories assigned to each word by the supertagger. In particular, in order to increase the speed of the parser without affecting accuracy, the supertagger simply needs to supply the one category to each word *that the parsing model ends up choosing anyway.* And there is an obvious way to train the supertagger to supply such categories: train it on large amounts of supertagged data from the parser. A similar idea was recently suggested by van Noord (2009) for a Dutch HPSG parser.

There is a practical difficulty associated with training the supertagger on large amounts of parsed data, since the supertagger is based on a maximum entropy (ME) model which uses an iterative, batch training process (Ratnaparkhi, 1998). We investigated two approaches to this problem, one based on a parallelised version of the ME estimation algorithm, and another based on a perceptron tagger as an alternative to the maximum entropy tagger (Collins, 2002). The advantage of the perceptron is that it uses an online training process in which the weights are updated one sentence at a time.

We were able to use up to 2,000,000 parsed sentences from Wikipedia as training data. Both the self-trained maximum entropy tagger and the self-trained perceptron tagger gave significant increases in parsing speed with no loss in accuracy.

4.1 Parallel Estimation

The ME tagger can be trained with either the GIS (Ratnaparkhi, 1998; Darroch and Ratcliff, 1972) or the BFGS (Malouf, 2002; Nocedal and Wright, 1999) training algorithms. Since these are both iterative batch training processes, all the data must be in RAM at the same time. To enable the use of larger models we increased the amount of accessible RAM and processing power by parallelising the supertagger training using MPI and the MapReduce library MRMPI.

The first stage of supertagging training is feature extraction and aggregation. Extraction is trivial to parallelise by dividing the contexts amongst a set of machines. For weight estimation, the maximum entropy methods are "embarrassingly parallel", as the main calculations are sums of weights across all training instances. The parallel versions of these methods differ in three main ways to the batch versions. First, the instances are divided between a set of machines. Second, sums are calculated across all machines to determine necessary changes to weights. And third, after each update the changes are distributed to all nodes.

The online training method of the perceptron adjusts the weights based on each training instance individually and so the parallelisation above was not applicable. The training instances are still distributed across a cluster of machines, but only one machine is working at a time, adjusting the weights based on all of its instances before passing the updated weights to the next node. This saves time by removing the cost of loading the training instances from hard disk when there are too many to fit in RAM.

4.2 The Perceptron Tagger

The perceptron tagger introduced by Collins (2002) uses a global iterative training process in which whole training sentences are decoded one at a time, using the existing model, and a simple update procedure is used to modify the weights: if an instance of a feature appears in the tagger output when it should not, then its weight is reduced by one, and if an instance of a feature occurs in the gold standard tag sequence but not in the tagger output, then its weight is increased by one. In this way the tagger is effectively learning to tag the training instances correctly.

The ME supertagger uses a local training process, in the sense that conditional probabilities are estimated for each word in a sentence (given the word's context), rather than the global conditional random fields used by Lafferty et al. (2001). In order to compare with the local ME models, and to take advantage of the existing implementation, we developed a local perceptron model, in the sense that the linear score function used by the perceptron applies only to a local context and not the complete sequence of tags. The score for a tag, t, and a context, c is defined as follows:

$$\mathrm{Score}(t, c) = \sum_i \lambda_i . f_i$$

where i ranges over the features and λ_i is the weight for feature f_i. The features are defined over a 5-word window centred on the target word, and based on the words and POS in the window, plus the two previously assigned tags, as for the ME tagger. The score for a complete sequence of tags is just the sum of the local scores for each word-tag pair, and the highest scoring tag sequence can be found with the Viterbi algorithm, as for the ME tagger.

The advantage of this approach is that full sentence decoding is not required during the training process, but only local decoding in the sense of finding the highest scoring tag for a local context (which is linear in the number of tags).[1]

In order to reduce overfitting on the training data, we used the now-standard averaged version of the perceptron (Collins, 2002). We also shuffled the order of the training instances between iterations of the algorithm. This prevents the model from overfitting to the particular order of training instances. It is unclear whether shuffling is beneficial in the general case (Clark and Curran, 2007b) but it is useful in our implementation since the training instances are artificially ordered for the ME training for efficiency reasons. Shuffling led to tagging accuracy improvements of up to 0.5%.

4.2.1 Margin Infused Relaxed Algorithm (MIRA)

As an alternative to the perceptron, we experimented with a variant of MIRA (Crammer and Singer, 2003). MIRA uses a similar online training process to the perceptron, but applies a different update method. The intention is to make the smallest possible change to the weights such that the correct class would be produced by a specified margin. As defined by Crammer, the update function adjusts the weights by a set of values satisfying:

$$\begin{array}{c} \min_{\bar{\tau}} \frac{1}{2}\sum_r \mid\mid \bar{M}_r + \tau_r \bar{x}^t \mid\mid_2^2 \\ \text{subject to: } (1)\ \tau_r \leq \delta_{r,y^t} \text{ for } r = 1, ..., k \\ (2)\ \sum_{r=1}^{k} \tau_r = 0 \end{array}$$

where τ is the update to be made, $\bar{M}$ is the matrix of weights, $\bar{x}^t$ is the value of the feature, k is the number of classes, and δ is the Dirac delta function, equal to 1 only when r is the index of the correct classification.

We have applied a slight variation that can be expressed as follows:

$$\min(max, \frac{margin + \sum_f p_w - t_w}{|features|(1 + \frac{1}{nabove})})$$

where *margin* is the absolute difference that will be created between the true classification and those that previously ranked above it, the sum is over all features, p_w and t_w are the weights associated with the feature f for the predicted and true classes respectively, $|features|$ is the number of active features, and *nabove* is the number of categories that had higher sums than the correct category. The constant, *max*, was introduced to prevent a single event causing large changes to the model.

4.3 Blocking Excess Backward Composition

This section is somewhat independent of the rest of the Chapter, but we include it here since these experiments were conducted as part of the supertagging experiments.

[1]One potential disadvantage is that local models such as these may suffer from the so-called label bias problem (Lafferty et al., 2001).

In the process of parser development, we investigated the number of times particular pairs of categories were combined. We were surprised to discover that a very large number of backward compositions (see Chapter 2) were being performed by the parser, even though backward composition rarely occurred in the parser output (or in the gold standard). One motivation for backward composition is for analyses of non-constituent coordination where pairs of type-raised categories are composed (Steedman, 2000), but the parser was also using it for combining one non-type-raised and one type-raised category. Hence we added a constraint that only allows backward composition to occur if both children are type-raised.[2] Surprisingly, the addition of this simple constraint results in a significant increase in parser speed, with little or no loss in accuracy. The significance of this result is that it demonstrates that increases in parser speeds can be obtained simply by imposing constraints on which categories can combine in the wide-coverage grammar.

4.4 Evaluation and Results

We used the CCG dependencies from Section 00 of CCGbank for accuracy evaluations to ensure that any speed improvements were not accompanied by large accuracy losses. For some of the experiments we also used the GRs from the 300 sentence Wikipedia test set described in Chapter 3. For the speed experiments we used both WSJ sentences from the Treebank and a larger set of sentences from Wikipedia.

One difficulty with these experiments is the subtle interaction between the supertagger and parser, as explained in Chapter 2. Initially, the supertagger provides a relatively small number of categories for each word, on average, with the number being determined by what we call the β parameter in the supertagger. If the parser is unable to find an analysis, the supertagger reduces the value of β which results in more categories being supplied. This process continues over five different β values in the current implementation of the parser.

As well as controlling the interaction between the supertagger and parser, another difficulty is that the number of categories supplied at a particular β value depends on the volume of training data. In particular, we found that, with large amounts of parser output for training, a much smaller β value was required to produce the same level of lexical category ambiguity.

In the initial set of experiments performed at the workshop we decided to keep the levels of lexical category ambiguity relatively constant across the various experiments, and leave the investigation of how different β values affect accuracy and speed for further work. Even when keeping the level of lexical ambiguity relatively constant, we found that we were able to significantly increase the speed of the parser through training the supertagger on large amounts of parser output.

As the amount of training data scales up, so does the time it takes to train the models. To demonstrate the benefits of perceptron-based training we measured the total training time for the different models. These measurements were performed using a 3GHz Intel Core 2 Duo CPU, and 4Gb of RAM.

[2]The Eisner (1996) normal form constraints implemented in the parser also fail to prevent this combination.

Parser	Accuracy F-score (%)	Speed WSJ (sent / sec)	Speed Wiki (sent / sec)
C&C 1.00	85.49	30.7	29.4
Modified	85.47	38.9	47.8

Table 4.1: The effect of introducing extra constraints on the use of backward composition on speed and accuracy of the parser.

One final experiment we performed was to increase the range of features used by the supertagger, with the hypothesis that the large amounts of self-training data available would allow us to extend the feature set (e.g. by extending the local context window).

The following sections give the results for the various experiments we performed.

4.4.1 Modified Backward Composition

The influence of the change to backward composition handling is shown in Table 4.1. A clear speed increase of more than 25% is achieved, with no significant change to F-score measured over the CCG dependencies in Section 00 of CCGbank.

4.4.2 Training Data Type and Volume

To investigate the effectiveness of self-training we constructed a series of supertagger models using the GIS algorithm and a selection of datasets. In Table 4.2 we can see that using Wikipedia data labelled by the parser as training data for the supertagger gives a clear improvement in parsing speed on Wikipedia text, but has a variable influence on speed for parsing newspaper text.

The model trained on $40,000$ Wikipedia sentences, approximately the same amount of text as in Section 02-21 of CCGbank, has much lower supertagging accuracy, but higher parsing speed, showing the speed improvements that are possible from self-training. The best result in the table arises from a model trained on an equal amount of text from CCGbank and Wikipedia. The speed benefits from training on parser output are retained, while the parser F-score does not decrease.

4.4.3 Algorithm Comparison

Using larger datasets for training can take a prohibitive amount of time for the GIS and BFGS algorithms. Table 4.3 shows the reduced training times that can be obtained with the online perceptron-based methods. The best performing perceptron-based models also show no reduction in accuracy over the ME tagger, either for supertagging accuracy or parser accuracy.

	Accuracy (%)			Amb.	Speed	
	WSJ		Wiki	Wiki	WSJ	Wiki
Data	Cat.	F	Cat.		(sent / sec)	
WSJ	**97.34**	85.65	96.30	1.34	38.6	44.9
Wiki						
40k	95.03	82.09	95.56	1.27	37.6	59.8
400k	96.17	83.65	96.31	1.27	**43.5**	**60.7**
2000k	96.62	84.54	**96.43**	1.28	43.4	59.3
WSJ+ Wiki						
40k	97.23	**85.73**	96.27	1.30	39.5	56.8
400k	97.09	85.19	96.31	1.29	35.9	58.9
2000k	97.11	85.43	96.33	1.29	35.1	58.8

Table 4.2: The effect of self-training on supertagging accuracy and parsing F-score. The Data column gives the number of parsed Wikipedia sentences used for training. The Amb. column gives the average number of categories assigned by the supertagger at the first β level.

	Accuracy (%)			Amb.	Speed		
	WSJ		Wiki	Wiki	Train	WSJ	Wiki
Data	Cat.	F	Cat.		(sec)	(sent / sec)	
WSJ							
GIS	97.34	85.65	96.30	1.34	7,200	38.6	44.9
BFGS	**97.36**	85.71	96.31	1.32	6,300	38.9	47.7
Perc	96.83	85.58	95.55	1.40	76	37.5	54.1
MIRA	97.28	85.69	96.21	1.34	96	39.0	45.3
WSJ+ 40k Wiki							
GIS	97.23	85.73	96.27	1.30	14,000	39.5	56.8
BFGS	97.20	85.62	96.06	1.30	13,000	38.3	59.4
Perc	96.85	85.73	95.52	1.32	160	37.2	**68.1**
MIRA	97.18	**85.74**	96.18	1.31	200	**40.2**	55.4
WSJ+ 400k Wiki							
GIS	97.09	85.19	96.31	1.29	*	35.9	58.9
Perc	96.88	85.35	95.85	1.32	950	34.3	67.9
MIRA	97.04	85.03	96.21	1.29	1,200	39.1	60.0
WSJ+ 2,000k Wiki							
GIS	97.11	85.43	**96.33**	1.29	*	35.1	58.8
MIRA	97.10	85.24	96.28	1.29	*	39.4	58.5

Table 4.3: Comparison of model estimation algorithms. The models missing times were trained on a different machine with greater RAM and are provided for accuracy comparison only.

	Accuracy (%)			Speed	
	WSJ		Wiki	WSJ	Wiki
Features	Cat.	F	Cat.	(sent / sec)	
		WSJ			
All	97.28	85.59	96.03	30.5	41.9
- far tags	97.16	85.53	96.06	31.0	41.4
- bitags	97.13	85.54	96.04	30.3	40.6
- far bitags	97.23	85.60	96.15	29.9	42.2
- tritags	97.26	85.63	**96.28**	30.8	41.1
- far tritags	97.24	85.66	96.25	30.2	42.2
- far words	97.28	85.60	96.18	30.6	41.2
- biwords	97.29	85.53	96.07	32.5	43.9
- far biwords	97.28	85.62	96.07	30.6	42.4
- triwords	**97.30**	85.66	96.10	32.2	44.7
- far triwords	97.28	85.68	96.18	31.1	42.5
Baseline	97.28	**85.69**	96.21	**39.0**	**45.3**
		WSJ+ 40k Wiki			
All	97.24	85.71	96.13	31.8	53.3
- far tags	97.20	85.87	96.09	31.5	52.2
- bitags	97.13	85.54	96.04	30.2	40.7
- far bitags	97.27	85.80	96.03	32.0	52.7
- tritags	97.19	85.56	**96.22**	32.0	52.6
- far tritags	97.28	85.67	96.18	32.2	51.9
- far words	97.34	85.70	96.19	32.0	53.1
- biwords	97.30	85.84	96.12	34.2	54.8
- far biwords	97.34	85.86	96.12	32.1	53.0
- triwords	**97.37**	**85.92**	**96.22**	33.3	55.2
- far triwords	97.30	85.80	96.16	31.8	53.6
Baseline	97.18	85.74	96.18	**40.2**	**55.4**
		WSJ+ 400k Wiki			
All	97.30	85.39	96.34	32.3	56.2
- far tags	97.25	85.22	96.37	31.7	56.9
- bitags	97.13	**85.54**	96.04	30.0	40.8
- bitags far	97.27	85.48	96.27	32.3	56.7
- tritags	97.26	85.24	**96.43**	33.7	56.6
- far tritags	97.27	85.37	**96.43**	32.8	55.9
- far words	**97.33**	85.24	**96.43**	32.6	56.3
- biwords	97.23	85.17	96.31	34.1	57.0
- far biwords	97.29	85.36	96.40	32.7	56.3
- triwords	97.30	85.43	96.33	33.1	58.1
- far triwords	97.31	85.34	96.36	32.7	56.9
Baseline	97.04	85.03	96.21	**39.1**	**60.0**

Table 4.4: Subtractive analysis of various feature sets. In each section the category accuracy values that are lower than those for 'All' have been underlined as removing these features decreases accuracy. The bold values are the best in each column for each section. The baseline model uses the default feature set for the C&C parser.

4.4.4 Feature Extension

The final set of experiments involved the exploration of additional features for the supertagger. Using the MIRA training method we were able to quickly construct a large set of models, as shown in Table 4.4. The standard features used by the supertagger are unigrams of words and unigrams and bigrams of POS tags in a five word window. We investigated expansions of this set to include bigrams and trigrams of both words and POS tags, and all of the features extended to consider a seven word window, which are indicated by the word 'far' in the table.

The results in the first section of the table, training on CCGbank only, are unsurprising. With such a small amount of data these additional features are too rare to have a significant impact. We had hoped that the expansion of the training data to include Wikipedia text would make these features frequent enough to be useful, but so far they have not provided a significant improvement. However, the largest of these models used only 400,000 Wikipedia sentences from our complete set of 47,000,000. It is possible that these tests still did not have enough training data to make use of the additional features.

4.5 Conclusion

We have increased the efficiency of the C&C parser, showing that self-training for the supertagger can boost parsing speed considerably, and demonstrated that perceptron-based algorithms can estimate supertagger model parameters just as well as maximum entropy methods. To achieve this we parallelised the supertagger training process, and implemented the averaged perceptron and MIRA algorithms for feature weight estimation. Training models using perceptron-based algorithms yielded equal performance in speed and accuracy, but reduced training time by two orders of magnitude.

The simple change in backward composition handling provided a 25% speed boost, increasing parsing speed on newspaper text from 30.7 to 38.9 sentences per second. By using parsed Wikipedia as extra training data we were able to increase speed when parsing Wikipedia by a further 19% from 47.8 to 56.8 sentences per second, without decreasing F-score.

Initially the system produced an F-score of 85.48% on Section 00 of CCGbank, could parse the WSJ and Wikipedia at 30.7 and 29.4 sentences per second respectively, and took two hours to train the supertagging model, using only forty thousand sentences for training. By modifying the handling of backward composition, doubling the amount of training data by using parsed Wikipedia text, and estimating the supertagging model using MIRA, we were able to construct a model in under four minutes, achieve an F-score of 85.74, and speeds of 40.2 and 55.4 sentences per second for WSJ and Wikipedia respectively, ie. 1.3 times as fast for WSJ, and 1.9 times as fast for Wikipedia text.

Chapter 5

Chart Pruning

One of the striking aspects of the pre-workshop version of the C&C parser was that the chart-parsing algorithm constructed the complete chart, without any form of chart pruning. This is striking because the high level of ambiguity resulting from the use of an automatically-extracted wide-coverage grammar means that some form of chart pruning is usually necessary for practical parsing (Collins, 1999; Charniak, 2000). The reason that the C&C parser is able to construct the complete chart, and still achieve practical parsing speeds, is that the supertagging stage has already removed much of the ambiguity that would usually be pruned when constructing the chart. However, even with the use of a supertagger, there are still very high numbers of derivations for some sentences, suggesting that chart pruning techniques could increase the speed of the parser further (without reducing accuracy).

We investigated three different techniques for pruning the chart. The first is a standard beam search, in which a constituent is pruned from a cell if its score is below some multiple of the highest scoring constituent in that cell (where the multiple is a fixed parameter, denoted β). This is the form of pruning used in the Collins (1999) parser. We also investigated a novel form of flexible beam search, in which the β parameter is varied according to the size of the constituent, as well as a thresholding technique in which the beam was only applied to constituents below a certain length. We calculated the score using both the inside score and Viterbi score, finding that the Viterbi score, which uses only a max operation rather than any summing, performed as well as the inside score. Overall we were able to achieve a significant speed increase with no loss in accuracy.

The second method is a more aggressive form of pruning in which *complete cells* are removed from the chart before parsing begins. The idea is that certain cells are unlikely to contain constituents because the words at the boundaries of the span corresponding to the cell are unlikely to start or end a constituent. We experimented with the existing method of Roark and Hollingshead (2009), in which a tagger can be used to label words according to whether they can start or end constituents. We also extended this approach to make it potentially more aggressive, summing certain tag probabilities so that more of the cells in the chart can be ruled out. We found both methods to be highly effective at increasing the speed of the parser with little or no loss in accuracy.

The third method is also aggressive, in that it effectively rules out whole sets of cells before parsing begins (or at least fixes the analysis for sets of cells). The idea is quite simple: if a sequence of words has already been seen frequently in text, and receives the same analysis each time, there is no need to build the analysis from scratch when it is encountered in unseen text. Rather the parser only needs to retrieve the relevant analysis from a cache of previously parsed examples and insert the analysis straight into the chart. Our hypothesis, based on the idea of "one parse per n-gram", is that it is faster to retrieve such an example and insert it into the chart than build it from scratch. One difficulty with this method is deciding when an existing analysis is the correct analysis for the newly encountered instance of the n-gram. In our preliminary experiments we were unable to improve on the speed of the parser using this technique.

5.1 Beam Search

The beam search approach used in our experiments prunes all constituents in a cell having scores below a multiple (β) of the score of the highest scoring constituent for that cell.[1] The scores for a constituent are calculated using the same model as that used to find the highest scoring derivation. We consider two scores: the Viterbi score, which is the score of the highest scoring sub-derivation for that constituent; and the inside score, which is the sum over all sub-derviations for that constituent. The research questions we investigated were the trade-off between the aggressiveness of the beam search and accuracy; the comparison between the Viterbi score and the inside score; whether applying the beam to only certain cells in the chart can improve performance; and whether a flexible beam search based on the size of the constituent is more effective than using a fixed value for β. From an implementation point of view, we also needed to add parser code which allows scores to be estimated as the chart is built, rather than delaying the score calculations until the complete packed chart is constructed, as in the pre-workshop implementation.

Table 5.1 shows the results of a preliminary experiment on Section 00 of CCGbank, using the Viterbi score to prune. The parser accuracy is measured using F-score over the CCG dependencies in CCGbank. As expected, the parsing speed increases as the value of β increases, since more constituents are pruned with a higher β value. The pruning is effective, with a β value of 0.01 giving a 55% speed increase with neglible loss in accuracy. Note also that, for some β values, the accuracy increases. This can be explained by the fact that the highest scoring derivation does not necessarily have the highest F-score among those available to the parser, and by pruning some constituents early the parser is led to select a better (but lower scoring) derivation overall. There is an additional effect resulting from the fact that early pruning of constituents allows the parser to parse more sentences compared with no pruning. (In the default mode of the parser, approximately 1% of unseen CCGbank sentences fail to receive a parse, due to the chart becoming too big.) Obtaining an analysis for more sentences leads to a higher

[1] One restriction we apply in practice is that only constituents resulting from the application of a binary rule, rather than a unary rule, are pruned.

β	Speed (sents/min)	Gain	Accuracy	Gain
Baseline	43.0		85.55	
0.001	48.6	5.6 (13%)	85.82	0.27
0.002	54.2	11.2 (26%)	85.88	0.33
0.005	59.0	16.0 (37%)	85.73	0.18
0.01	66.7	23.7 (55%)	85.53	-0.02

Table 5.1: Using different beam values β.

δ	Speed (sents/min)	Gain	Accuracy	Gain
Baseline	43.0		85.55	
10	60.1	17.1 (39%)	85.55	0.00
20	70.6	27.6 (64%)	85.66	0.11
30	72.3	29.3 (68%)	85.65	0.10
40	76.4	33.4 (77%)	85.63	0.08
50	76.7	33.7 (78%)	85.62	0.07
60	74.5	31.5 (73%)	85.71	0.16
80	68.4	25.4 (59%)	85.71	0.16
100	62.0	19.0 (44%)	85.73	0.18
None	59.0	16.0 (37%)	85.73	0.18

Table 5.2: Thresholding on beam search.

recall and hence higher F-score.

Similarly, beam search may have a negative as well as positive effect on the speed. During beam search, beam scores for constituents are being calculated for all constituents created by the parser, even those which may not form part of a spanning analysis. In contrast, for the pre-workshop implementation of Viterbi search, the scores are calculated only when the complete chart has been created and hence can be restricted to those constituents forming part of a spanning analysis. A further factor is that, for beam search using the inside score, a summing operation is required compared to the max operation used by Viterbi. Given these considerations, we investigated a selective beam search in which pruning was only applied to constituents below a certain length. Our hypothesis was that pruning shorter constituents early in the chart-parsing process, preventing the creation of many larger, low-scoring constituents later, would have a larger effect on the speed.

We experimented with a selective beam search in which pruning is only applied to constituents of length less than or equal to a threshold δ. For example, if $\delta = 20$, pruning is applied only to constituents spanning 20 words or less. The results are shown in Table 5.2. The selective beam is also highly effective, showing speed gains over the baseline — which does not use a beam — with no loss in F-score.[2] For a δ value of 50 the speed increase is 78% with no loss in accuracy.

[2] Again we would expect a slight increase in recall because more sentences will receive an analysis with the beam, accounting for some of the slight increase in F-score.

	β	δ	Speed (sents/min)	Accuracy
Baseline			24.7	85.55
inside scores	0.1		40.3	81.52
	0.01		37.7	85.52
	0.001		25.3	85.79
	0.005	10	33.4	85.54
	0.005	20	39.5	85.64
	0.005	50	42.9	85.58
Viterbi scores	0.1		40.7	81.57
	0.01		38.1	85.53
	0.001		28.2	85.82
	0.005	10	33.6	85.55
	0.005	20	39.4	85.66
	0.005	50	43.1	85.62

Table 5.3: Comparison between using Viterbi scores and inside scores as beam scores.

Table 5.3 shows the comparison between the inside score and Viterbi score. The results are similar, with the Viterbi score marginally outperforming the inside score in most cases. The interesting result from these experiments is that the summing used in calculating the inside score does not improve performance over the max operator used by Viterbi.

So far all results have been on the WSJ text in CCGbank. Table 5.4 gives results on Wikipedia text, compared with a number of sections from CCGbank. (Sections 02-21 provide the training data for the parser which explains the high accuracy results on these sections.) The Wiki 300 test set is that described in Chapter 3.[3] The Wiki 2500 test set is used for speed comparison only, in order to provide a larger set of sentences than the Wiki 300 set; it contains 2,500 Wikipedia sentences.

Despite the fact that the pruning model is derived from CCGbank and based on WSJ text, the speed improvements for Wikipedia were even greater than for WSJ text, with parameters $\beta = 0.005$ and $\delta = 40$ leading to almost a doubling of speed on the Wiki 2500 set, with the parser operating at 90 Wikipedia sentences per second.

Finally, we experimented with variable beam values where β is varied with respect to the length of span. For example, we tried different equations for β such as $\beta = an + b$ or $\beta = a/n$, where n is the constituent span length. Constants such as a and b were fitted using the average ratio of the highest scores to the scores of the gold standard parse. However, the use of the variable beam did not improve performance over the fixed thresholding technique.

In future work it is worth investigating the variable beam further, as well as training the pruning model on large amounts of parser output, similar to the self-training experiments described in the next section. For the beam search experiments described here the pruning model was always derived from gold-standard CCGbank data.

[3] The results here are higher than those in Chapter 3 because gold standard POS tags were used here.

	Speed			Accuracy		
Dataset	Baseline	Beam	Gain	Baseline	Beam	Gain
WSJ 00	43.0	76.4	33.4 (77%)	85.55	85.63	0.08
WSJ 02-21	53.4	99.4	46.0 (86%)	93.61	93.27	-0.34
WSJ 23	55.0	107.0	52.0 (94%)	87.12	86.90	-0.22
Wiki 300	35.5	80.3	44.8 (126%)	84.23	85.06	0.83
Wiki 2500	47.6	90.3	42.7 (89%)			

Table 5.4: Beam search results on WSJ 00, 02-21, 23 and Wikipeida texts with $\beta = 0.005$ and $\delta = 40$.

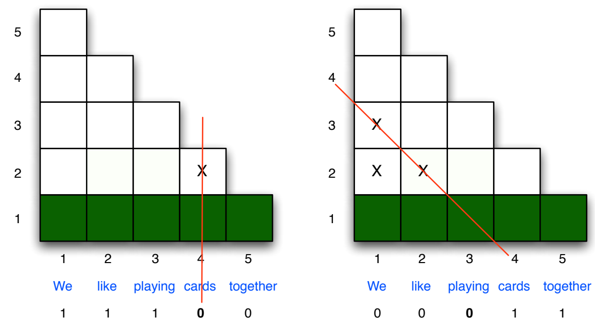

Figure 5.1: The pruning effect of binary begin (left) and end (right) tags

5.2 Cell Pruning

5.2.1 Binary tagging

Cells can be removed from the chart simply by tagging words with two types of binary label. Before parsing starts, two types of tags are assigned to each input word to indicate whether the word can be the beginning or end of a multiple-word constituent, respectively. The binary "begin" and "end" tags are assigned separately. This approach was proposed by Roark and Hollingshead (2009) for CKY parsing.

Given the input "We like playing cards together", the pruning effects of a begin tag and an end tag are shown in Figure 5.1. In a CKY chart, each column represents constituents that begin with the same word, and each row represents constituents that have the same size. Therefore, each cell in the chart represents a set of constituents that cover the same span of input words. The pruning effects of the begin and end tags are to cross out particular chart cells that correspond to multi-word constituents; hence no cell in the first row is ever pruned, since these cells correspond to only a single word.

In Figure 5.1 the begin tag for the input word "cards" is 0, meaning that it cannot begin a multi-word constituent, and that no cell in the corresponding column can contain any constituent. Therefore, the pruning effect of a binary begin tag is to cross out a column of chart cells (ignoring the first row) when the tag value is zero. In Figure 5.1 the

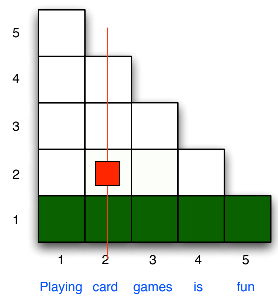

Figure 5.2: The limitation of binary begin tags

end tag of the word "playing" is 0, meaning that it cannot be the end of a multi-word constituent. Consequently the cell that begins at the first word and has size three cannot contain any constituent, nor can the cell that begins at the second word and has size 2. Therefore, the pruning effect of a binary end tag is to cross out a diagonal of chart cells (ignoring the first row) when the tag value is zero.

We use a maximum entropy trigram tagger to assign the begin and end tags. Features based on the words and POS in a 5-word window, plus the two previously assigned tags, are extracted from the tag trigram ending with the current tag and the five-word window with the current word in the middle. In our development experiments, both the begin and the end taggers gave a per-word accuracy of around 96%.

The standard method to derive training data for the taggers is to transform gold-standard parse trees into begin and end tag sequences, and we call this method gold-standard training. There is an alternative way to obtain training data for the taggers, which is to derive it from parser output. The intuition is that the tagger will learn what constituents a trained parser will eventually choose, and as long as the constituents favoured by the parsing model are not pruned, no reduction in accuracy can occur (but there is the potential for an increase in speed). The advantage of this self-training method is that the amount of parser output available is much greater than gold-standard training data.

5.2.2 Level tagging

The pruning effect of binary tags is to prune whole columns or diagonals of chart cells. A binary tag cannot take effect when there is any chart cell in the corresponding column or diagonal that contains constituents. For example, the begin tag for the word "card" in Figure 5.2 cannot be 0 because "card" begins a two-word constituent "card games". Hence none of the cells in the column can be pruned by the binary begin tag, even though all the cells from the third row above are empty. We propose what we call a level tagging approach to address this problem.

Instead of taking a binary value that indicates whether a whole column or diagonal of cells can be pruned, a level tag takes an integer value which indicates the row from

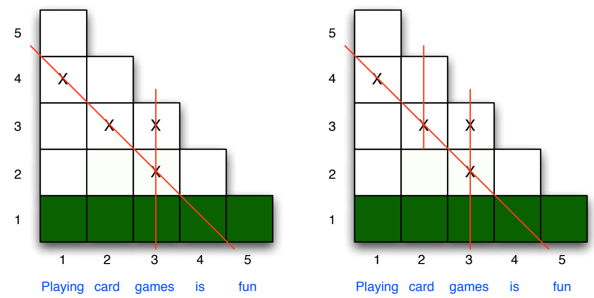

Figure 5.3: The pruning effect of binary (left) and level (right) tags

which a column or diagonal can be pruned in the upward direction. For example, a level begin tag 2 allows the column of chart cells for the word "card" in Figure 5.2 to be pruned from the third row upwards. A level tag with value 1 prunes the corresponding row or diagonal from the second row upwards; it has the same pruning effect as a binary tag with value 0. For convenience, value 0 for a level tag means that the corresponding word can be the beginning or end of any constituent, which is the same as a binary tag value 1. The maximum value N a level tag can take is decided experimentally during development. We performed development tests for tagsets with $N = 2$ and $N = 4$.

A comparison of the pruning effect of binary and level tags for the sentence "Playing card games is fun" is shown in Figure 5.3. With a level begin tag, more cells can be pruned from the column for "card". Therefore, level tags are potentially more powerful for pruning.

Level tags are derived by training a maximum entropy tagger for *maxspan* tags. A maxspan tag takes the same value as its corresponding level tag. However, the meaning of maxspan tags and level tags are different. While a level tag indicates the row from which a column or diagonal of cells is pruned, a maxspan tag represents the size of the largest constituent a word begins or ends. Parse trees can be turned directly into training data for a maxspan tagger.

We use the standard maximum entropy trigram tagger for maxspan tagging, where features are extracted from tag trigrams and surrounding five-word windows. During decoding, the maxspan tagger uses the forward-backward algorithm to compute the probability of maxspan tag values for each word in the input. Then for each word, the probability of its level tag t_l having value x is the sum of the probabilities of its maxspan t_m tag having values $1..x$:

$$P(t_l = x) = \sum_{i=1}^{x} P(t_m = i)$$

Maxspan tag values i from 1 to x represent disjoint events in which the largest constituent that the corresponding word begins or ends has size i. Summing the probabilities of these disjoint events gives the probability that the largest constituent the word begins or ends has a size between 1 and x, inclusive. That is also the probability that all the

Model	Speed	Accuracy
baseline	25.10	84.89
begin only	27.49	84.71
end only	30.33	84.56
both	33.90	84.60
gold oracle	33.60	85.67
self oracle	55.25	84.89

Table 5.5: Development tests for the binary tagger

constituents the word begins or ends are in the range of cells from rows 1 to row x in the corresponding column or diagonal. And therefore that is also the probability that the chart cells above row x in the corresponding column or diagonal do not contain any constituents, which means that the column and diagonal can be pruned from row x upward. Therefore, it is also the probability of a level tag with value x.

The probability of a level tag having value x increases as x increases from 1 to N. We set a probability threshold Q and choose the smallest level tag value x with probability $P(t_l = x) \geq Q$ as the level tag for a word. If $P(t_l = N) < Q$, we set the level tag to 0 and do not prune the column or diagonal. The threshold value determines a balance between pruning power and accuracy, with a higher value pruning more cells but increasing the risk of incorrectly pruning a cell. During development experiments we arrived at a threshold value of 0.8 as providing a suitable compromise between pruning power and accuracy.

5.2.3 Experiments

Experiments were performed with both gold-standard training data (gold-training) and parser output training data (self-training) to train the binary and level taggers. We used CCGbank data for gold-training and Wikipedia data for self-training. In all the experiments, accuracy is measured using F-score over CCG dependencies in the CCGbank development and test sets, and speed is measured in sentences per second.

5.2.3.1 Development tests. Table 5.5 shows a set of development tests for the binary tagger, using data from CCGbank. Sections 02-21 were used for training of the tagger and section 00 was used to test the pruning effect. With the begin tags or the end tags alone, the parser achieved speed increases with a small loss in accuracy. With both the begin and the end tags, the parser achieved further speed increases, with no loss in accuracy compared to the end tag alone. The "gold oracle" row shows the oracle for gold-training. For this experiment, the binary tags were extracted directly from gold-standard parses. Pruned with these tags, the parser was about as fast as when both the begin tags and end tags from a trained tagger are applied, but more accurate than the baseline. The "self oracle" row shows the oracle for self-training. For this experiment, the binary tags are extracted from the parser output directly. Then these tags were used to prune the parser, which led to more than a doubling in speed. This oracle shows the

Model	Speed	Accuracy
baseline	25.10	84.89
binary	33.90	84.60
binary gold oracle	33.60	85.67
binary self oracle	55.25	84.89
level $N = 2$	32.79	84.92
level $N = 2$ gold oracle	47.31	86.49
level $N = 2$ self oracle	76.07	84.89
level $N = 4$	34.91	84.95
level $N = 4$ gold oracle	47.45	86.49
level $N = 4$ self oracle	77.35	84.89

Table 5.6: Development tests for the level tagger

potential speed up that can be obtained from self training.

Table 5.6 shows a set of development tests for the level tagger, comparing the effect of the binary tagger and level taggers with $N = 2$ and $N = 4$. For all the tests shown in the table, sections 02-21 of CCGbank were used to train the level or binary taggers, while section 00 was used to test the pruning effect. The table shows that the parsing accuracies with the level taggers are higher than with the binary tagger; they are also higher than the baseline parsing accuracy. The parser achives the highest speed and accuracy when pruned with the $N = 4$ level tagger. Comparing the oracle scores, the level taggers lead to higher speeds than the binary tagger for both gold-standard training and self training. The self oracle speeds for the level taggers are more than three times as high as the baseline speed, showing the potential of self training.

5.2.3.2 Self training tests. Now we report the final self training tests, using Wikipedia test data. For the self-training, different amounts of sentences from the parser output were used to train the binary and level taggers. We performed two sets of tests. In the first set, we used 300 manually annotated sentences from Wikipedia to test the accuracy of the parser (described in Chapter 3). By using a small manually annotated test set, we wanted to observe the influence of the binary and level taggers on parsing accuracy (in particular wanting to ensure that no significant accuracy loss was obtained).[4] In the second set of tests, we used 2,500 unannotated sentences from Wikipedia to test the speed of the parser. For all the tests using binary or level taggers, both begin and end tags are applied to prune the chart.

Tables 5.7 and 5.8 give the self-training results. The results show that the accuracy loss using self-trained binary or level taggers was not large (in the worst case, the accuracy dropped from 84.23% to 83.39%), while the speed was significantly improved. Using binary taggers, the largest speed improvement was from 47.56 sentences per second to 83.48 sentences per second (a 75.5% relative increase). Using level taggers, the largest speed improvement was from 47.56 sentences per second to 104.03 sentence per second

[4]The accuracy figures here are higher than those in Chapter 3 because gold standard POS tags were used in these experiments.

Model	Speed	Accuracy
baseline	36.64	84.23
binary 40K	48.79	83.64
binary 200K	51.51	83.71
binary 1M	47.78	83.75
level 40K	54.76	83.83
level 200K	48.57	83.39
level 1M	52.54	83.71

Table 5.7: Self training test for the parsing accuracy

Model	Speed
baseline	47.56
binary 40K	83.48
binary 200K	79.34
binary 1M	80.14
level 40K	104.03
level 200K	101.06
level 1M	99.38

Table 5.8: Self training test for the speed improvements

(a 101.9% relative increase). However, as the number of training sentences increased from 40 thousand to a million, the speed did not improve.

5.2.3.3 Comparing self-training with gold-training. In this experiment, we compare the pruning effect of self-trained binary and level taggers with the effect of gold-trained taggers. In Table 5.9, rows "Binary Gold" and "Level Gold" represent experiments with binary and level taggers trained using sections 02-21 of CCGbank (which contains around $40,000$ sentences), respectively, while rows "Binary 40K" and "Level 40K" represent experiments with binary and level taggers trained using $40,000$ Wikipedia sentences from the parser output. The test set was the 300 manually annotated sentences from Wikipedia. The results showed that, with the same amount of training data, gold-training is more effective than self-training.

5.2.4 Conclusion

We applied the binary tagging approach from Roark and Hollingshead (2009) to prune whole chart cells, and found that this method led to a substantial improvement in parsing speed with little loss in accuracy. We generalized the binary method into a novel level tagging method, which gave a better pruning effect than the binary approach. We experimented with self-training, using the parser output to train the taggers, and observed a doubling in speed with the level taggers (from 47.56 sentences per second to 104.03 sentences per second) with a small loss in accuracy (from 84.23% to 83.71% in F-score).

Model	Speed	Accuracy
baseline	36.64	84.23
Binary 40K	48.79	83.64
Binary Gold	49.59	84.36
Level 40K	54.76	83.83
Level Gold	58.23	84.12

Table 5.9: Comparison between self-training and gold-training of taggers.

One potential advantage of the self-training method is that a large number of training examples can be easily be produced. In our experiments, however, we did not find a consistent improvement in the pruning power when the size of the self-training data increased from $40,000$ sentences to $1,000,000$. The accuracy of the parser did not improve consistently either, as the size of the training data for the tagger increased. Possible future work would be to further investigate the potential of a larger amount of self-training data.

5.3 1 Parse per n-gram

The notion of "1 parse per n-gram" is inspired by other observed redundancies in natural language, such as "1 sense per discourse" (Gale. et al., 1992), and relies on the observation that some frequently occurring n-grams typically appear with the same syntactic analysis. We intended to exploit this redundancy in order to improve the efficiency of the parser, by pre-constructing the parse structures for these frequent n-grams and inserting the structures straight into the parse chart whenever an existing n-gram is encountered in unseen text. Our hypothesis was that it is faster to retrieve and insert these pre-constructed analyses into the chart than build them from scratch.

In order to test this hypothesis, we built a database of CCG analyses for common n-grams from Sections 02-21 of CCGbank, and applied these analyses to sentences from Section 00. Once an n-gram from the databased was encountered, we inserted the corresponding analysis straight into the relevant cell of the chart. Our preliminary result was that no significant change in accuracy or parse time for the C&C parser was observed. However, we did formulate a number of new hypotheses regarding how this method can be improved as part of future work.

5.3.1 N-gram analysis

In order for our proposal to improve parsing speed, there has to be a large number of frequently occurring n-grams in the data which form constituents; frequently occurring because we need to see them in unseen text in order to have an impact on speed; and constituent-forming so that we have an analysis to insert straight into the chart. Table 5.10 shows the percentage of n-grams in Sections 02-21 of CCGbank which always form a constituent, and the average number of analyses for those n-grams. The relatively high percentage of n-grams forming constituents, especially for bigrams, and the relatively low average number of derivations, suggested that our proposal has potential

n-gram size	2	3	4
Avg number derivations	1.19	1.09	1.04
Always form constituents	23%	10%	5%
Never form a constituent	73%	89%	93%

Table 5.10: N-gram statistics for Sections 02-21 of CCGbank

bigram	# No	# Yes	# Uniq
`the company`	8	1157	1
`a share`	3	1082	7
`New York`	4	868	7
`a year`	34	572	9
`do n't`	0	474	9
`the market`	37	410	1
`did n't`	0	378	11
`is n't`	1	367	21
`The company`	0	359	1
`does n't`	0	328	10

Table 5.11: Statistics for the 10 most frequent bigrams in Sections 02-21 of CCGbank which typically form constituents

to increase the speed of the parser.

However, when we investigated the most frequently occurring bigrams, the results were not so promising. Table 5.11 shows the 10 most frequent bigrams in Sections 02-21 of CCGbank which usually form constituents. The columns show the number of times the n-gram was seen not forming a constituent, the number of times it was seen as a constituent, and the number of unique constituent-forming derivations the n-gram was seen with. As can be seen from the table, only 3 of the top 10 most frequent bigrams occur with 1 unique derivation, and the next smallest number of unique derivations is 7.

5.3.2 Utilising "non-constituents"

The flexible notion of constituency employed by CCG means that there is the potential to store analyses for frequently occurring "non-constituents"; these are constituents that do not appear often in the normal-form derivations in CCGbank, but are nonetheless constituents according to the grammar formalism, typically formed using forward composition. For example, *of the* occurs frequently in CCGbank, but rarely as as constituent; however assuming that forward composition has no restrictions applied to it, the following analysis is possible:[5]

[5]The theory of CCG (Steedman, 2000) stipulates various restrictions on the use of forward composition, which for practical purposes we ignore.

bigram	# No	# Yes	# Uniq	$\sum$	Coverage	Ambiguity
`of the`	4936	0	0	9872	1.06	1.031
`in the`	3911	5	1	17704	1.90	1.747
`, the`	3489	0	0	24682	2.66	1.005
`, and`	2219	9	3	29138	3.13	1.000
`, a`	2167	0	0	33472	3.60	1.000
`, which`	1705	0	0	36882	3.97	1.118
`for the`	1638	0	0	40158	4.32	1.958
`to the`	1588	1	1	43336	4.66	1.925
`on the`	1533	0	0	46402	4.99	1.962
`, said`	1258	0	0	48918	5.26	1.290
`, but`	1193	1	1	51306	5.52	1.045
`the company`	8	1157	1	53636	5.77	1.000
`, he`	1165	0	0	55966	6.02	1.000
`that the`	1150	0	0	58266	6.27	1.283
`a share`	3	1082	7	60436	6.50	1.107

Table 5.12: Statistics for the 15 most frequent bigrams in Sections 02-21 of CCGbank. The columns show the number of times the bigram was seen forming a non-constituent, forming a constituent, and the number of unique constituent-forming chart structures. The next two columns show accumulatively what percentage of Sections 02-21 these bigrams cover. The last column shows the ambiguity the C&C supertagger associates with each n-gram.

$$\begin{array}{cc}
\texttt{of} & \texttt{the} \\
\hline
(NP\backslash NP)/NP & NP/N \\
\hline
\multicolumn{2}{c}{(NP\backslash NP)/N} \quad {>}\mathbf{B}
\end{array}$$

The constituent for *of the* could then be used to form a prepositional phrase:

$$\begin{array}{ccc}
\texttt{of} & \texttt{the} & \texttt{potato} \\
\hline
(NP\backslash NP)/NP & NP/N & N \\
\cline{1-2}
\multicolumn{2}{c}{(NP\backslash NP)/N} \quad {>}\mathbf{B} & \\
\hline
\multicolumn{3}{c}{NP\backslash NP} \quad {>}
\end{array}$$

Since these flexible constituents typically lead to non-normal-form derivations, the use of them violates the Eisner normal-form constraints implemented in the parser, which are intended to increase efficiency by ruling out some non-normal-form derivations (Eisner, 1996; Clark and Curran, 2007c). We modified the parser so that any sub-analyses loaded from a pre-constructed database are allowed to lead to full analyses violating the Eisner constraints.

Table 5.12 shows the 15 most frequent bigrams in Sections 02-21 of CCGbank. Only two of the top 15 most frequent bigrams typically form a constituent, again leading to the conclusion that using only constituent-forming bigrams is not the correct approach. Note also that seven out of these 15 contain a comma, which are difficult to deal with in our approach because the comma is ambiguous and can take on a number of syntactic roles, making it difficult to insert a single analysis for the bigram. The Σ column shows an accumulative sum of the number of tokens covered in Sections 02-21 just by using the bigrams in the table. The coverage figures shown in the neighbouring column show this sum as a percentage of the total number of tokens in Sections 02-21. Assuming this trend were to continue for a large number of the most frequently occurring n-grams, then a reasonable proportion of uneen text could be covered by our pre-constructed database of n-grams.

5.3.3 Implementation

Tokyo Cabinet[6] is an open source, lightweight database API which provides a number of different database implementations, including a hash database, B+ tree, and a fixed-length key database. Our experiments used Tokyo Cabinet to store the pre-constructed n-grams because of its ease of use, speed, and maximum database size (8EB).

The construction of the final set of n-gram databases is a multi-stage process, with intermediate databases being generated and then refined. The first stage in the construction of the databases is to parse all of the training data, which in our case is Sections 02-21 of CCGbank. The parse tree for every sentence is then analysed for constituent-forming n-grams (using the flexible CCG notion of constituency), for a particular value of n. If a constituent-forming n-gram is found, then the n-gram and its corresponding chart structure are written out to a database. These first stage databases are implemented using a simple key-value Tokyo Cabinet hash database. A key in this database is a pair consisting of the n-gram and a hash value corresponding to the analysis (sub-derivation) of the n-gram, and a value is a pair consisting of an analysis (`chart`) and a counter recording the frequency of occurrence (`occurrence`).

A 64-bit hash function was developed for chart structures (sub-derivations) so that we can represent a whole sub-derivation by its value, while obtaining very few hash collisions. Empirical evaluation of this hash function was performed over all of the chart structures, for all constituent-forming n-grams of size 2 to 5 contained in the noun phrase corrected version of CCGBank (Vadas and Curran, 2008). No collisions were experienced for 536,165 n-grams.

The `chart` attribute in the value is a serialised version of the chart which can be unserialised at some later point for reuse. The `occurrence` counter is incremented each time an occurrence of a key is seen in the parsed training data. A record is also kept in the database for the number of times a particular n-gram was seen forming a non-constituent, for later use in the filtering stage.

One subtle property of the serialisation process for particular n-grams is that n-grams of the same size often have the same chart structure. This is beneficial when creating our pre-constructed chart databases, since, instead of having a one-to-one mapping from n-gram to chart structure in our final database, we can have two different databases. The first database is called `ngram2id` which maps an n-gram to a unique id within the second database. The second database, named `id2chart`, maps a unique id for a particular

[6] `http://tokyocabinet.sourceforge.net/`

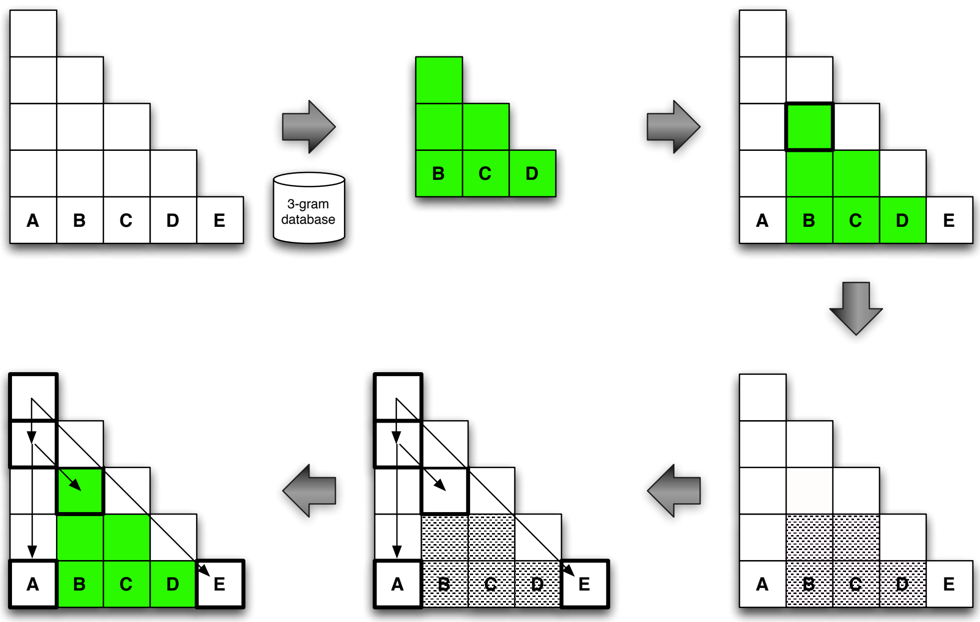

Figure 5.4: Illustration of using the n-gram databases. The trigram `B C D` is loaded from the pre-constructed database, and blocks out the corresponding cells

chart structure to the chart structure itself.

When constructing the initial set of databases over a body of text, a large number of the seen n-grams should not be kept in the final databases because they occur too infrequently, or because the number of times they are seen forming a non-constituent outweighs the number of times they are seen forming a constituent (in the flexible CCG sense). Hence we apply a frequency based filtering stage to the initial set of databases to produce the final `ngram2id` and `id2chart` databases.

5.4 Experiments and Results

Sections 02-21 of CCGbank were used for constructing the databases, and Section 00 was used as the test set. When parsing a sentence from Section 00, the n-gram database check is performed from left to right; if two n-grams overlap and both have pre-constructed n-grams in the database, then only the first n-gram has its pre-constructed form loaded from the database. Once a pre-constructed chart structure is loaded into the chart, the corresponding cells are blocked off from further use in the parsing process, as illustrated in Figure 5.4.

We used parsing models based both on CCGbank and the noun phrase corrected version (Vadas and Curran, 2008). The accuracy figures remained the same, and a small, non-significant change was observed for the speed.

5.5 Conclusion and Future Work

Our initial experiments have not provided any speed improvements to the parser. An obvious extension for future work is to look at longer n-grams, which will have two positive effects: one, larger parts of the chart will be ruled out given one n-gram; and two, longer n-grams are likely to be less ambiguous. Another obvious extension, which will be necessary for using longer n-grams, is to use a large amount of parsed data for constructing the n-gram database, rather than Sections 02-21 of CCGbank. Finally, preliminary experiments performed post-workshop suggest that loading chart structures based on frequent, unambiguous sequences of CCG *lexical categories*, rather than words, may be more effective at increasing the speed of the parser.

Chapter 6

Improving Coordination Disambiguation

Coordination is perhaps the most difficult of the common grammatical relations for statistical parsers to process. In the DepBank corpus, for example, it is about half as frequent as determiners and more frequent than auxiliaries; however, while auxiliaries and determiners can be analysed with over 90% F-score by the C&C parser, coordination is below the 80% mark. This effect can also be observed in other systems; e.g. the RASP parser has an F-score of 72% on conjunctions (Briscoe and Carroll, 2006).

The reason why coordination creates difficulties for parsers is the high level of ambiguity it engenders. Consider the following sentence:

1. Food and Drug Administration spokesman Jeff Nesbitt said the agency has turned over evidence in a criminal investigation concerning Vitarine Pharmaceutical Inc. to the US attorney's office in Baltimore.

The C&C parser chooses in this case to coordinate *Food* and *Nesbitt*; that is, an analysis equivalent to *Food said the agency had turned over evidence and Drug Administration spokesman Jeff Nesbitt said the agency had turned over evidence.* In addition to producing an incorrect `conj` relation, the analysis also introduces an incorrect subject-verb relation. Consequently, improving coordination has a potentially positive effect on other grammatical relations in the sentence.

Despite the importance of parsing coordinations correctly, the issue has been little studied. Agarwal and Boggess (1992) used syntactic symmetry and semantic labelling to detect coordination in the Merck Veterinary Manual. Their inclusion of lexical semantics as a determining feature for disambiguation was echoed in subsequent papers. Further examples can be found in Chantree et al. (2005) and Resnik (1999). Another method for dealing specifically with coordination in compounds is the use of Hearst-like patterns (Nakov and Hearst, 2005). Overall, it can be said that most papers have been dealing with specific coordination constructs rather than the phenomenon as a whole: Resnik (1999) andNakov and Hearst (2005) focus on noun compounds, Chantree et al. (2005)

on modification and Goldberg (1999) on coordination within prepositional phrases. The Agarwal and Boggess study is domain-specific.

We apply here some existing ideas to the task of improving the C&C parser on coordinations. Our aim is to boost the accuracy of the parser when considering which two words should be coordinated in any instance of an *and* or *or* conjunction in a sentence. As such, we are dealing with a wide range of coordination types and must therefore implement a classifier using a suitably generic set of features. We are specifically interested in techniques which exploit previously parsed text to produce appropriate semantic features.

The following sections introduce some theory about coordination, describe our classifier and present the results obtained in the course of the workshop.

6.1 Coordination and parallelism

The idea that natural language prefers to coordinate constructs that are similar, either syntactically or semantically, is well-referenced in the literature. Dubey et al. (2005) give a corpus-based analysis of the phenomenon of similarity of conjuncts, showing its frequent occurrence, but it was implicitly assumed in earlier computational linguistics papers: Agarwal and Boggess (1992) build their coordination resolution system around lexical similarity of conjuncts and syntactic parallels. Okumura and Muraki (1994), working on Japanese, also take advantage of parallelisms in the syntax of the sentence.

Taking parallelism into account, we can say that, in example 1, *food* and *drug* are more likely coordinated than *food* and *Nesbitt* because they are semantically more similar. Note, however, that this feature alone is not sufficient for correctly predicting coordination, as the following example shows, where *stand* and *want* are the correct conjuncts, despite the fact that *feel* and *want* are semantically closer.

2. I just don't feel that the company can really stand or would want a prolonged walkout.

Note also that, in this example, the two correct conjuncts are much closer to each other than the incorrect ones, showing that distance between conjuncts is another useful predictor. Agarwal and Boggess note the effect of distance on coordination, and argue that the second coordinate is usually close to the conjunction; that is, sentences like the following are relatively rare:

3. The cat and - I couldn't believe my eyes - the dog were sleeping next to each other.

To take into account the parallelism effect, we introduce in our system syntactic and semantic-based similarity features alongside distance features. The next section describes our choice of features and the experimental framework for the task.

6.2 Experimental Framework

In order to try and improve the performance of the parser on coordination constructions, we worked within a re-ranking framework. For each sentence containing a coordination, we parse that sentence with the n-best version of the C&C parser, returning 1000 possible analyses. Out of those 1000 choices, all possible pairs of *and/or* coordinates for the sentence are retrieved and the best one(s) selected according to a classifier. Hence each instance of a `conj` relation in a sentence forms a test case, and precision and recall is calculated over all instances, where a point is awarded if the pair of conjuncts is correctly returned by the classifier. The baseline is provided by the top-ranked parse returned by the parser.

6.2.1 Classifier

We use a Naive Bayes classifier (as part of the WEKA machine learning toolkit) to choose the pair of conjuncts for each instance of the `conj` relation. The following features were used by our system:

- original rank in the n-best parse;
- distance of the second coordinate to the conjunction;
- distance between coordinates;
- n-gram similarity of coordinates;
- lexical similarity between coordinates.

Our approach to lexical similarity is described in the next section. The distance features simply count the number of words between the seond coordinate and the conjunction, and between the two coordinates, respectively. The n-gram similarity feature records whether the potential coordinates belong to identical unigrams, bigrams or trigrams, as given by their part of speech information:

4. *Mary* and *Jane* like going to the park. (Unigram match)
5. *The cat* and *the dog* like going to the park. (Bigram match)
6. *The black cat* and *the brown dog* like going to the park. (Trigram match)

For each sentence, and for each potential pair of coordinates in that sentence, we return a string containing the values of each feature and pass it to the Naive Bayes classifier. The classifier is asked to make a decision, for each presented pair, as to whether the two conjuncts are likely or not. Figure 6.1 shows the input format for an example sentence with some coordination instances and feature values.

6.3 The lexical similarity systems

We developed two similarity systems, one based on a manually created resource (WordNet) and another based on distributional information obtained automatically from a subset of Wikipedia parsed by the C&C parser.

```
*********************
Sentence with POS tags
*********************
Dr.|NNP_0 Talcott|NNP_1 lead|VBD_2 a|DT_3 team|NN_4 of|IN_5 researcher|NNS_6 from|IN_7
the|DT_8 National|NNP_9 Cancer|NNP_10 Institute|NNP_11 and|CC_12 the|DT_13 medical|JJ_14
school|NNS_15 of|IN_16 Harvard|NNP_17 University|NNP_18 and|CC_19 Boston|NNP_20
University|NNP_21 .|._22

COORD 19
13 (conj and_19 University_21) (conj and_19 University_18) 1 0.721348 0.910239 1 2 1
13 (conj and_19 University_21) (conj and_19 school_15) 0.8 0.513898 0.910239 0.617647 0 0
13 (conj and_19 University_21) (conj and_19 Institute_11) 0.6 0.417032 0.910239 0.37037 2 0
13 (conj and_19 University_21) (conj and_19 team_4) 0.4 0.345976 0.910239 0.4 0 0
13 (conj and_19 University_21) (conj and_19 researcher_6) 0.2 0.360674 0.910239 0.15 0 0
ORIG BEST: 13 (conj and_19 University_21) (conj and_19 University_18)

COORD 12
13 (conj and_12 school_15) (conj and_12 researcher_6) 1 0.434294 0.721348 0.0909091 2 0
13 (conj and_12 school_15) (conj and_12 Institute_11) 0.833333 0.621335 0.721348 0.188679 0 1
13 (conj and_12 University_21) (conj and_12 Institute_11) 0.666667 0.417032 0.434294 0.37037 2 0
13 (conj and_12 University_21) (conj and_12 researcher_6) 0.5 0.360674 0.434294 0.15 0 0
13 (conj and_12 school_15) (conj and_12 team_4) 0.333333 0.40243 0.721348 0.285714 0 0
13 (conj and_12 University_21) (conj and_12 team_4) 0.166667 0.345976 0.434294 0.4 0 0
ORIG BEST: 13 (conj and_12 school_15) (conj and_12 researcher_6)
```

Figure 6.1: Input format for the Naive Bayes classifier for coordination instances with word indices 12 and 19 in the example sentence

6.3.1 WordNet system

We first developed a WordNet-based similarity system (Fellbaum, 1998), using the Wu and Palmer (1994) measure to determine the distance between two concepts in the hierarchy (and using both the noun and verb hierarchies from WordNet). The idea is that the closer two concepts are, the more hypernyms they share. For each pair of words $(w1, w2)$, we extract the hypernyms of both $w1$ and $w2$, collapsing all senses. We then calculate a similarity using the cosine measure between the two hypernym lists.

6.3.2 Distributional similarity system

Our second system was based on distributional similarity, with the now standard idea that words which are semantically similar appear in similar contexts (Harris, 1954). Examples of existing distributional similarity systems include Lin (1998), Lin and Pantel (2001), Curran (2004) and Geffet and Dagan (2005). The context of a word can be defined in a number of ways; one successful approach, which we follow here, is to use the syntactic contexts in which a word is found, based on grammatical relations (GRs). For example, given the sentence *Talcott leads the university team*, the following relations can be extracted:

`(ncsubj lead_2 Talcott_1 _)`

which indicates that *Talcott* is subject of the head *lead*;

`(dobj lead_2 team_5)`

which indicates that *team* is object of the head *lead*;

`(det team_5 the_3))`

which indicates that *the* is the determiner of *team*; and
`(ncmod _ team_5 university_4)`
which indicates that the argument of *university* is *team*.

A GR can be transformed into a feature for a particular lexical item by replacing the slot containing the word with a "hole": `(dobj lead_2 team_5)` becomes `(dobj lead_2 HOLE)`, a potentially characteristic context for the word *team*. For each word w in the pair of words to be compared, we extract all GR-based patterns (i.e., GRs with holes) containing w. We then calculate the weight for each contextual element (i.e. each component of the feature vector) using Pointwise Mutual Information (PMI). The intuition is that we want the weight to reflect how well the contextual element reflects the meaning of the word; for example, we might expect a high PMI between the word *shirt* and the contextual element represented by the direct object of the verb *wear*, since the fact that shirts are worn is highly indicative of the meaning of *shirt*.

The PMI between a contextual pattern p and a word w is defined as follows:

$$pmi(p,w) = \log\left(\frac{P\,(p,w)}{P\,(p)\,P\,(w)}\right) \tag{6.1}$$

where $P(p)$ and $P(w)$ are the probabilities of occurrence of the contextual pattern and the word respectively and $P(c,w)$ is the probability that they appear together.

PMI is known to have a bias towards less frequent events (Manning and Schutze, 1999). In order to counterbalance that bias, we apply a simple logarithm function to the results as a discount:

$$d = \log\left(c_{wp} + 1\right) \tag{6.2}$$

where c_{wp} is the cooccurrence count of a word and a contextual pattern. We multiply the original PMI value by this discount to find the final PMI.

Once the feature vectors are built, we calculate their similarity using the measure of Lin (1998):

$$Lin(w_1,w_2) = \frac{\sum_{f\in F_{w_1}\cap F_{w_2}}[W(f,w_1)+W(f,w_2)]}{\sum_{f\in F_{w_1}}W(f,w_1)+\sum_{f\in F_{w_2}}W(f,w_2)} \tag{6.3}$$

where F_w is the feature vector for word w and $W(f,w)$ is the weight of feature f for word w (in our system, the corresponding PMI).

6.4 Evaluation

We built the distributional vectors for each word from a subset of Wikipedia parsed by the C&C parser, consisting of around 600,000 sentences. We used 180 short Wikipedia sentences (less than 30 words each) as a development corpus, all of which contained at least one coordination. We split that corpus into 100 sentences for training and 80 sentences for testing. As our evaluation corpus, we used Sections 02-21 of CCGbank (Hockenmaier, 2003) for training, and 300 sentences of GR-annotated Wikipedia data (described in Chapter 3) for testing. The 300 Wikipedia sentences contained a total of

System	Precision	Recall	F-score	Gain
Baseline	77.4%	66.3%	71.4%	–
Similarity (WordNet)	79.7%	69.9%	74.5%	3.1
Similarity (distributional)	79.2%	69.4%	74.0%	2.6

Table 6.1: Results for coordination disambiguation

445 coordinations. Table 6.1 gives precision and recall values for all *and* and *or* `conj` relations in the test data and N-best parser output.

6.5 Discussion

Our results show that improvements of up to 3 percentage points can be obtained using some distance and syntactic features combined with WordNet-based similarity. Similar, but slightly smaller, gains were obtained for the system using distributional similarity.

Our error analysis indicated that some grammatical patterns might benefit from being treated separately, which has precedents in the literature. Resnik (1999), for example, focuses on the coordination of noun compounds and argues that lexical similarity is a particular strong feature for this construction. We tested this idea towards the end of the workshop by adding two simple binary features to our system, recording whether the conjuncts under consideration were part of a prepositional phrase, or were subjects to the main verb of the sentence. Examples of both constructs, *prep* n_1 *and* n_2 and n_1 *and* n_2 *verb*, are shown below.

7. Dr Talcott led a team **of** researchers **from** the medical schools **of** Harvard University(1) and Boston University(2).

8. Food and Drug Administration spokesman Jeff Nesbitt **said**...

For both constructs, our intuition is that lexical similarity and distance between conjuncts are particularly strong features. The fact that *food* and *Nesbit* are so semantically different suggests they are unlikely to both be subjects to the same verb. And even though *schools* and *University(2)* are semantically similar, and could both be in a relation of meronymy to *researchers*, *University(1)* and *University(2)* are even more similar and also closer in the sentence.

In order to detect prepositional phrases and subjects in our n-best parses, we used the following heuristics, based on the idea that dependencies between words close to each other in a sentence are likely to be correct:

- *prep* n_1 *and* n_2: if n_1 is the argument of *prep* in more than x% of the parses, then it belongs to a prepositional phrase.

- n_1 *and* n_2 *verb*: if n_2 is the first argument of *verb* in more than x% of the parses, then it is in the subject of *verb*.

We set x at 80%. Adding those two features to our WordNet-based system resulted in an additional improvement in both recall and precision, at 70.3% and 80.3% respectively, yielding an F-score of 75.0% and an overall gain of 3.5 percentage points.

6.6 Conclusion and future work

The workshop allowed us to make several contributions to the problem of coordination disambiguation. First, we were able to improve the F-score of the parser on coordination constructions by over 3 percentage points using a combination of syntactic and semantic features. Next, we showed that the output of the C&C parser on Wikipedia can successfully be used as data for building distributional similarity models. Finally, our results suggest that additional accuracy can be achieved by considering various coordination types — based on the types being coordinated — individually.

There is still work to be done to determine the effect of the coordination improvements on the parser as a whole. The current coordination system does not feed its output back into the parser, so we cannot comment on how the improvement affects recall and precision over the rest of the GRs. Our expectation is that other relations should benefit from the higher precision on coordination. Finally, other aspects of coordination still need to be investigated. In particular, we did not investigate argument effects, where the attachment of an argument to one or the other of the coordination conjuncts might be ambiguous:

9. At the JHU workshop, we parsed the web, ate and played cards.

While the parser is likely to coordinate eating and playing, it is not clear whether we ate the cards or not.

Chapter 7

Factual Bootstrapping

So far this report has dealt with improving data-driven parsing without recourse to additional training data. However, relying entirely on CCGbank as our data source is inherently limiting as we attempt to parse text further away from the WSJ text in CCGbank. Even though the self-training approaches discussed in Chapter 5 can help with domain adaptation, their primary effect was to increase speed rather than accuracy as they reinforce the existing biases in the model.

It has become perceived wisdom in NLP that more labelled data is the easy route to better accuracy, but the cost of manual annotation for parsing data is prohibitive. Therefore an automated approach to the acquisition of labelled parsing data is an attractive alternative to expensive manual annotation.

7.1 Fact Redundancy

One novel idea for the automatic acquisition of parsing data, first introduced by Howlett and Curran (2008), is to exploit the redundancy which pervades the web: if we can identify sentences that express the same idea, we can expect the dependency relationships between the words expressing that idea to remain constant. Consider the following two sentences:

Soderbergh directed the movie Traffic

Soderbergh directed Out Of Sight and Traffic

We can see that the dependencies between *Soderbergh*, *Traffic* and *directed* remain constant, despite the increased complexity of the second sentence. If we make the assumption that, the shorter the sentence is, the more likely it will be parsed correctly, we can extract the relevant dependencies from a short sentence to inform our processing of a longer sentence. In our approach, the dependencies extracted from the short sentence will form hard constraints on the analysis given to the longer sentence. Similar to Section 5.3, the assumption that certain small sets of relations might be consistent across sentences can be seen as an extension of the argument for one sense per collocation (Yarowsky, 1993).

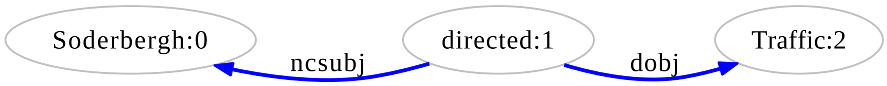

Figure 7.1: The constraints found from the sentence *Soderbergh directed the movie traffic*.

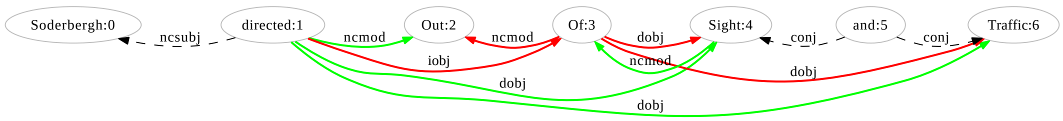

Figure 7.2: The constraints found in Figure 7.1 are applied to the more complex sentence *Soderbergh directed Out of Sight and Traffic*. Red lines indicate relations removed from the unconstrained parse; green lines indicate the new relations added.

Figures 7.1 and 7.2 give an example of how such constraints are applied, with 7.1 containing the original, short sentence, and 7.2 the longer sentence. The output format is the grammatical relations output from the C&C parser. The new analysis for the longer sentence has been obtained by selecting the highest scoring derivation which satisfies the dependency constraints from the short sentence. The red lines show the relations that were in the original analysis of the longer sentence but were removed after the constraints were applied, and the green lines show the relations that were added. Note that the new parse in 7.2 is not completely correct — *directed Out* is regarded as a verb particle construction — but is considerably better than the initial analysis where *Traffic* was a dependent of *of*. The idea is that the new analysis for the longer sentence can be used as additional training data for the parser.

7.2 Relation Selection

A key problem for this approach is deciding which sentences, and sets of dependencies from those sentences, can be used to constrain analyses of future sentences (in order to provide additional training data). For example, the simple sentence *Jack loved Jill* is unlikely to provide useful constraints. Even though the sentence can most likely be parsed reliably, the problem is that any future sentence containing *Jack*, *Jill*, and *loved* may not contain the same dependencies between these words. Future sentences might be *Jill loved Jack*, or *Jill and Jack loved John*, or *Jill loved the ball that Jack gave to her*.

The proposed solution of Howlett and Curran (2008) is to focus on simple 'facts', under the assumption that at least some of the dependencies in sentences expressing the fact remain constant across different instances. For example, if we believe that Alexander Graham Bell invented the telephone, and we see the words *Bell*, *invented*, and *telephone* in the same sentence, then it seems reasonable to constrain the parse to reflect this information, by requiring the relevant dependencies expressing the fact to appear in the parse.

The procedure we follow is to choose some set of key words representing a given fact, words which we expect to have a fixed relationship. Then, a "simple" sentence expressing the fact is parsed, giving a dependency graph. In order to create constraints on the analyses of future sentences expressing the fact, we need to find the minimal way in which the relevant words are connected (which may involve additional words). For example, if we decided that all sentences with *Mozart*, *1756*, and *born* contained the same set of dependencies, we might discover that the connection between *born* and *1756* relies on the word *in*; if our simple sentence was *Mozart was born in 1756*, the constraints extracted would be `dobj`(*in*, *1756*), `ncmod`(*born*, *in*), and `ncsubj`(*born*, *Mozart*, obj). Determining the minimal way in which words are connected in a dependency graph is equivalent to finding the minimal spanning tree of a subset of vertices (the keywords), known as the Steiner Tree (Karp, 1972). This is an NP-complete problem, but for our graphs can be solved in reasonable time because of the small size of the dependency graphs, derived as they are from simple sentences.

7.3 Experiments

The initial system that formed the basis for our work, that of Howlett and Curran (2008), was only evaluated on a small scale, as the selection of keywords (those determining sentences to be constrained) was done manually, and so less than 1,000 new sentences were generated. Given that these were added to the 40,000 sentences in CCGbank for training purposes, and testing was performed against CCGbank rather than from a related domain, it was unsurprising that accuracy was not improved. Our main goal in the workshop was to extend the approach, using many more facts and creating more training data, so that a more comprehensive evaluation could be performed.

7.3.1 Finding New Facts

The most critical step was to dramatically increase the amount of training data by automating the process of fact identification. This was done by relying on `freebase.com` (Bollacker et al., 2008), a complex database which is built upon a number of web resources. Using this, we can ask for any attributes associated with a particular entity; for instance, a simple search on the Person data type will reveal that Persons have a birthdate. Then a simple query will extract all possible Person/birthdate combinations, which gives us a large set of possible facts.

Once a fact has been extracted, we require a simple sentence expressing the fact from which to derive dependency constraints. One difficulty we encountered is that the way in which the fact is expressed in the simple sentence may not generalise to other instances. For example, suppose we discover from `freebase.com` that Khalil Sultan was born in 1384, and in our simple sentences this fact is expressed using a parenthetical expression, such as *Khalil Sultan (born 1384)*. The relevant dependencies from the simple sentence will not generalise to more complex sentences in which the fact is expressed differently, such as:

> Nasir al-Din Khalil Sultan, the son of Miranshah b. Timur (Tamerlane), was born in 1384.

However, given our use of `freebase.com`, it is relatively straightforward to generalise our constraint extraction over all possible connections. That is, we can discover, by looking at many thousands of simple sentences, that *X was born in Y* is a common way to express the Person-birthdate connection. This not only aids us with data sparsity for lower frequency connections (such as Khalil-1384 above), but is also helpful in confirming that our extracted relation set is reliably associated with the selected connection.[1]

7.3.2 Applying the constraints

Another extension to the system of Howlett and Curran (2008) was that we investigated various ways in which to decide whether to use a constrained sentence as training data. For example, if applying the constraints to a sentence radically changed the highest scoring analysis, that might suggest that the constrained parse would be an informative example for the parser to learn from. Another idea we explored was to consider the difference in probability between the constrained and unconstrained analysis. Exploring these aspects allows us to feed potentially more informative sentences to the parser, as well as restricting the application of nonsensical constraints.

7.4 Process Outline

The steps in our system for acquiring and applying additional training data are as follows:

1. Gather sentences

    1. Automatically extract binary connections from **freebase.com** (e.g. *Bell*, *telephone*), as described in 7.3.1.
    2. For each connection, such as Inventor-Invention, search for both terms (e.g. *Bell* and *telephone*) on the web, collecting all unique sentences that include both.

2. Extract constraints

    1. Parse simple sentences and extract the Steiner Tree containing at least those keywords. For our experiments, a "simple" sentence was defined as one having at most two verbs, less than 15 tokens, and exactly one instance of each of the keywords.
    2. If there is more than one unique simple sentence supporting a particular constraint, regard it as a valid means of expressing the relationship between the relevant words. The constraints may include variables over terms; for example, *Bell invented the telephone* and *Wright invented the aeroplane* suggest that the relations for *X invented the Y* are valid for all Inventor-Invention connections.

3. Apply constraints

    1. For all sentences extracted in Part 1, attempt to parse with and without the constraints found in Part 2.

[1]The way in which we generalise across patterns is reminiscent of Lin and Pantel (2001).

Model	double the data			triple the data		
	P	R	F	P	R	F
baseline	79.05%	73.60%	76.23%	79.05%	73.60%	76.23%
wiki	78.25%	74.25%	76.20%	78.34%	74.19%	76.21%
web	77.95%	74.12%	75.99%	78.12%	74.23%	76.13%
useddata	78.58%	73.18%	75.78%	78.34%	73.18%	75.67%
constrained	78.11%	72.87%	75.40%	78.44%	73.34%	75.80%
nonmod	78.44%	73.30%	75.78%	78.70%	73.52%	76.02%
mod	78.00%	72.77%	75.29%	78.12%	72.79%	75.36%
ordered	78.41%	73.01%	75.61%	78.26%	72.90%	75.48%

Table 7.1: Parser accuracy on CCGbank dependencies in Section 00.

2. For those sentences to which the constraints apply, and have constrained parses (it is possible to parse the sentence with the implicated constraint), record the parse as *good*.
3. Record whether the *good* parse was different to the unconstrained parse (we refer to such parses as *mod*), or whether it was not modified (*nonmod*). If modified, record the difference in probability of the constrained and unconstrained parses.

4. Use new *good* parses to supplement existing gold standard data when training; this step can be informed by whether the constraint induced a change and the difference in probabilities between the constrained and unconstrained parses.

7.5 Results

In order to determine if the additional training data improves parser performance, we attempted to improve a parser model trained on a subset of CCGbank (Sections 02-05, containing around 7,000 sentences). This is also the model we used for the constraint extraction and returning the highest scoring parse for a sentence to which the constraints had been applied (in order to create additional training data). Given the speculative nature of the approach, our feeling was that it was better to start with a lower baseline, rather than immediately try and improve the model trained on the entire CCGbank.

We investigated doubling and tripling the original gold standard data based on the *constrained* parses generated from Step 3. A number of different selection strategies were evaluated: selecting any valid parse (*constrained*), selecting only those parses where the constraints did not induce a change (*nonmod*), only those which did induce a change (*mod*), and selecting those which had the least probability reduction out of those changed (*ordered*). Apart from the latter method, all sentences were randomly sampled. We also generated a number of additional baselines, in addition to just using CCGbank data (*baseline*): we sampled sentences from Wikipedia (*wiki*), the *web*, and from the sentences spidered for our constraining data (*useddata*). For the baselines the parses derived from the sentences and used as additional training data were all unconstrained, so that the

Model	double the data			triple the data		
	P	R	F	P	R	F
baseline	76.52%	74.94%	75.72%	76.52%	74.94%	75.72%
wiki	74.99%	73.60%	74.28%	75.30%	73.95%	74.62%
web	75.97%	75.52%	75.74%	75.47%	75.03%	75.25%
useddata	75.69%	74.41%	75.05%	75.27%	74.25%	74.76%
constrained	74.84%	72.69%	73.75%	75.25%	74.07%	74.65%
nonmod	75.56%	72.95%	74.23%	75.44%	74.70%	75.07%
mod	74.77%	71.43%	73.06%	74.14%	72.71%	73.42%
ordered	74.76%	73.16%	73.95%	74.44%	73.72%	74.08%

Table 7.2: Parser accuracy on labelled grammatical relations in Wikipedia text.

system was simply self-trained with a comparable amount of data.

Table 7.1 shows the accuracy of the parser on Section 00 of CCGbank for the various volumes and types of additional training data, compared to the baselines. Table 7.2 gives the accuracy of the parser on the Wikipedia data set. As can be seen from the tables, none of the results were promising: the small change that was induced was almost uniformly negative, and clearly some of the *mod* parses used as training data were incorrect (note the consistent drop in performance between *nonmod* and *mod*).

We did perform some limited manual evaluation of the modified parses, and our impression, which needs to be confirmed in further experimental work, is that the majority of the modified parses are improved. However, this is clearly not being reflected in the results.

7.6 Conclusions and Future Work

There are two obvious areas in which the current system could be improved. First, we need to investigate the proportion of sentences chosen for additional training data which do express the relevant fact using the relevant dependencies, and investigate ways in which to rule out erroneous examples such as *The telephone was invented by Elisha Gray, not Alexander Graham Bell.* Second, we need to investigate methods for ensuring that the analyses used for additional training data, after the constraints have been applied, are of a high quality. One approach would be to further develop our *ordered* method, which considers the change in probability of the constrained and unconstrained parses. Another approach would be to employ a human annotator in the loop, since, although full parse annotation itself is prohibitively expensive, confirming that a larger sentence expresses the same idea as a shorter sentence appears more feasible. It may even be possible to exploit resources such as Amazon's Mechanical Turk, provided our problem could be phrased without undue complexity.

Chapter 8

Conclusion

Our main result from the workshop is that we were able to substantially improve the speed of the parser, without harming accuracy, using various optimisation techniques applied to the parse chart. Standard beam search-based pruning and a novel method for cell pruning — building on the work of Roark and Hollingshead (2009) — were particularly effective. Using these techniques we are now able to parse over 100 Wikipedia sentences per second on a single machine, compared with a pre-workshop baseline of around 30 sentences per second.

We were also able to perform some large-scale self-training experiments, using a cluster built from commodity hardware at the University of Sydney School of Information Technologies, together with a parallelised version of the maximum entropy tagger training code, and a new perceptron-based supertagger. Again these self-training experiments resulted in improvements to the *speed* of the parser. There is much room to further investigate our idea that self-training on a 2-stage parsing system can significantly improve the efficiency of the parser, without harming accuracy, by getting the first stage to propose structures which the second stage will end up choosing anyway.

Given that the various efficiency techniques we investigated are largely orthogonal, there is potential for further speed increases by incorporating all the optimisations in a single implementation. We were unable to perform this experiment during the workshop because of time constraints.

We were able to use the output of the parser to improve the performance of a re-ranker on coordination disambiguation, through the use of a distributional model of lexical semantics. There is room here to incorporate this model directly into the disambiguation model used by the parser, rather than rely on a reranking phase.

We were unable to improve the speed of the parser based on our notion of one parse per n-gram, nor improve the accuracy of the parser based on our idea that additional training data can be acquired cheaply by exploiting fact redundancy on the web. However, both these projects were in their infancy during the workshop and continue to be investigated by members of the team.

Our accuracy evaluations showed that the newspaper-trained parser performs surprisingly well on Wikipedia text. The impressive speeds we now obtain allow the potential for large-scale, accurate, sophisticated analysis of web text, for use in various knowledge acquisition tasks and semantic search engines.

Version 2.0 of the parser, incorporating many of the innovations described in this report, will be available post-workshop at http://svn.ask.it.usyd.edu.au/trac/candc/wiki.

Acknowledgements

We would like to thank the following people for helping make the 2009 workshop such a success:

- Laura Rimell for performing the manual annotation described in Chapter 3.
- Matthew Honnibal for use of his gold-standard CCG Wikipedia data.
- Fred Jelinek for overall organisation and supporting our proposal and team.
- The JHU faculty, in particular Chris, Damianos, David, Jason, and Sanjeev, for providing useful feedback and enjoyable company during the 6 weeks.
- Sanjeev for organising many of the social events, especially the weekly dinners.
- The JHU admin staff, in particular Desiree and Monique.
- The other teams for providing feedback and being excellent companions during the workshop.
- Mark Steedman and Julia Hockenmaier for early collaboration on the parser and for producing CCGbank which made the CCG parsing work possible.

The workshop was supported by National Science Foundation Grant Number IIS-0833652, with additional funding from Google Research, the Defense Advance Projects Research Agency's GALE Program and the Johns Hopkins University Human Language Technology Center of Excellence.

References

Agarwal, R. and Boggess, L. (1992). A simple but useful approach to conjunct identification. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, pages 15–21, Morristown, NJ, USA. Association for Computational Linguistics.

Bangalore, S. and Joshi, A. (1999). Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Banko, M. and Etzioni, O. (2008). The tradeoffs between open and traditional relation extraction. In *Proceedings of the 46th Meeting of the ACL*, Columbus, Ohio.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *HLT '91: Proceedings of the workshop on Speech and Natural Language*, pages 306–311.

Bod, R. (2003). An efficient implementation of a new DOP model. In *Proceedings of the 10th Meeting of the EACL*, pages 19–26, Budapest, Hungary.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM.

Bos, J., Clark, S., Steedman, M., Curran, J. R., and Hockenmaier, J. (2004). Wide-coverage semantic representations from a CCG parser. In *Proceedings of COLING-04*, pages 1240–1246, Geneva, Switzerland.

Briscoe, T. and Carroll, J. (2006). Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the Poster Session of COLING/ACL-06*, Sydney, Austrailia.

Burke, M., Cahill, A., O'Donovan, R., van Genabith, J., and Way, A. (2004). Large-scale induction and evaluation of lexical resources from the Penn-II Treebank. In *Proceedings of the 42nd Meeting of the ACL*, pages 367–374, Barcelona, Spain.

Cahill, A., Burke, M., O'Donovan, R., van Genabith, J., and Way, A. (2004). Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Meeting of the ACL*, pages 320–327, Barcelona, Spain.

Carroll, J., Briscoe, T., and Sanfilippo, A. (1998). Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st LREC Conference*, pages 447–454, Granada, Spain.

Chantree, F., Kilgarriff, A., de Roeck, A., and Wills, A. (2005). Disambiguating coordinations using word distribution information. In *RANLP05*, Borovets, Bulgaria.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *NAACL-00*, pages 132–139, Seattle, WA.

Chen, J. and Vijay-Shanker, K. (2000). Automated extraction of TAGS from the Penn Treebank. In *Proceedings of IWPT 2000*, Trento, Italy.

Clark, S. (2002). A supertagger for Combinatory Categorial Grammar. In *Proceedings of the TAG+ Workshop*, pages 19–24, Venice, Italy.

Clark, S. and Curran, J. R. (2004). The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING-04*, pages 282–288, Geneva, Switzerland.

Clark, S. and Curran, J. R. (2007a). Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Meeting of the ACL*, Prague, Czech Republic.

Clark, S. and Curran, J. R. (2007b). Perceptron training for a wide-coverage lexicalized-grammar parser. In *Proceedings of the ACL-07 Workshop on Deep Linguistic Processing*, pages 9–16, Prague, Czech Republic.

Clark, S. and Curran, J. R. (2007c). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Comp. Linguistics*, 33(4):493–552.

Clark, S. and Curran, J. R. (2009). Comparing the accuracy of CCG and Penn Treebank parsers. In *Proceedings of the Short Papers of the Joint conference of the Association for Computational Linguistics and the Asian Federation of Natural Language Processing (ACL-IJCNLP-09)*, pages 53–56, Singapore.

Clark, S., Hockenmaier, J., and Steedman, M. (2002). Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Meeting of the ACL*, pages 327–334, Philadelphia, PA.

Clark, S., Steedman, M., and Curran, J. R. (2004). Object-extraction and question-parsing using CCG. In *Proceedings of the EMNLP Conference*, pages 111–118, Barcelona, Spain.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Meeting of the ACL*, pages 16–23, Madrid, Spain.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.

Collins, M. (2002). Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP-02*, pages 1–8, Philadelphia, PA.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the ACL*, pages 111–118, Barcelona, Spain.

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Curran, J. R. (2004). *From Distributional to Semantic Similarity*. PhD thesis, University of Edinburgh.

Curran, J. R. and Clark, S. (2003). Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 10th Meeting of the EACL*, pages 91–98, Budapest, Hungary.

Curran, J. R., Clark, S., and Vadas, D. (2006). Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the Joint Conference of COLING/ACL-06*, pages 697–704, Sydney, Austrailia.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic: Vol. I.* Amsterdam, North Holland.

Darroch, J. N. and Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480.

Dienes, P. and Dubey, A. (2003). Deep syntactic processing by combining shallow methods. In *Proceedings of the 41st Meeting of the ACL*, pages 431–438, Sapporo, Japan.

Dubey, A., Sturt, P., and Keller, F. (2005). Parallelism in coordination as an instance of syntactic priming: evidence from corpus-based modeling. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 827–834, Morristown, NJ, USA. Association for Computational Linguistics.

Eisner, J. (1996). Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Meeting of the ACL*, pages 79–86, Santa Cruz, CA.

Fellbaum, C., editor (1998). *WordNet : An Electronic Lexical Database*. MIT Press, Cambridge, Mass.

Gale., W., Church, K., and Yarowsky, D. (1992). One sense per discourse. In *Proceedings of the 4th DARPA Speech and Natural Language Workshop*, pages 233–237.

Geffet, M. and Dagan, I. (2005). The distributional inclusion hypotheses and lexical entailment. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 107–114, Morristown, NJ, USA. Association for Computational Linguistics.

Gildea, D. (2001). Corpus variation and parser performance. In *2001 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Pittsburgh, PA.

Goldberg, M. (1999). An unsupervised model for statistically determining coordinate phrase attachment. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 610–614, Morristown, NJ, USA. Association for Computational Linguistics.

Harris, Z. (1954). Distributional structure. *Word*, 10:146–162.

Hockenmaier, J. (2003). *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Hockenmaier, J. and Steedman, M. (2002). Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA.

Hockenmaier, J. and Steedman, M. (2007). CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Howlett, S. and Curran, J. R. (2008). Automatic acquisition of training data for statistical parsers. In *Proceedings of the Australasian Language Technology Association Workshop 2008*, pages 37–45, Hobart, Australia.

Johnson, M. (2002). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Meeting of the ACL*, pages 136–143, Philadelphia, PA.

Karp, R. (1972). Reducibility among combinatorial problems. *Complexity of computer computations*, 43:85–103.

Kasami, J. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Kiss, T. and Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, Williams College, MA.

Levy, R. and Manning, C. (2004). Deep dependencies from context-free statistical parsers: correcting the surface dependency approximation. In *Proceedings of the 41st Meeting of the ACL*, pages 328–335, Barcelona, Spain.

Lin, D. (1995). A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of IJCAI-95*, pages 1420–1425, Montreal, Canada.

Lin, D. (1998). An information-theoretic definition of similarity. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 296–304, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Lin, D. and Pantel, P. (2001). Dirt - discovery of inference rules from text. In *In Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 323–328.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Workshop on Natural Language Learning*, pages 49–55, Taipei, Taiwan.

Manning, C. and Schutze, H. (1999). *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusetts.

Marcus, M., Santorini, B., and Marcinkiewicz, M. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

McClosky, D., Charniak, E., and Johnson, M. (2006). Effective self-training for parsing. In *Proceedings of HLT-NAACL-06*, Brooklyn, New York.

McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Meeting of the ACL*, pages 91–98, Michigan, Ann Arbor.

Miyao, Y., Ninomiya, T., and Tsujii, J. (2004). Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the Penn Treebank. In *Proceedings of IJCNLP-04*, pages 684–693, Hainan Island, China.

Miyao, Y. and Tsujii, J. (2004). Deep linguistic analysis for the accurate identification of predicate-argument relations. In *Proceedings of COLING-2004*, pages 1392–1397, Geneva, Switzerland.

Miyao, Y. and Tsujii, J. (2005). Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd meeting of the ACL*, pages 83–90, University of Michigan, Ann Arbor.

Nakov, P. and Hearst, M. (2005). Using the web as an implicit training set: application to structural ambiguity resolution. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 835–842, Morristown, NJ, USA. Association for Computational Linguistics.

Nivre, J., Hall, J., Kubler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007). The CoNLL 2007 shared task on dependency parsing. In *Conference on Empirical Methods in Natural Language Processing and Natural Language Learning*.

Nivre, J. and Scholz, M. (2004). Deterministic dependency parsing of English text. In *Proceedings of COLING-04*, pages 64–70, Geneva, Switzerland.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer, New York, USA.

Okumura, A. and Muraki, K. (1994). Symmetric pattern matching analysis for English coordinate structures. In *Proceedings of the fourth conference on Applied natural language processing*, pages 41–46, Morristown, NJ, USA. Association for Computational Linguistics.

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In *Proceedings of the HLT/NAACL conference*, Rochester, NY.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of the EMNLP Conference*, pages 133–142, Philadelphia, PA.

Ratnaparkhi, A. (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania.

Resnik, P. (1999). Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res. (JAIR)*, 11:95–130.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., III, J. T. M., and Johnson, M. (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the ACL*, pages 271–278, Philadelphia, PA.

Rimell, L. and Clark, S. (2008). Adapting a lexicalized-grammar parser to contrasting domains. In *Proceedings of the EMNLP Conference*, pages 475–484, Honolulu, Hawaii.

Rimell, L. and Clark, S. (2009). Porting a lexicalized-grammar parser to the biomedical domain. *Journal of Biomedical Informatics*, doi:10.1016/j.jbi.2008.12.004.

Rimell, L., Clark, S., and Steedman, M. (2009). Unbounded dependency recovery for parser evaluation. In *Proceedings of the EMNLP Conference*, pages 813–821, Singapore.

Roark, B. and Hollingshead, K. (2009). Linear complexity context-free parsing pipelines via chart constraints. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 647–655, Boulder, Colorado. Association for Computational Linguistics.

Sarkar, A. and Joshi, A. (2003). Tree-adjoining grammars and its application to statistical parsing. In Bod, R., Scha, R., and Sima'an, K., editors, *Data-oriented parsing*. CSLI.

Steedman, M. (1996). *Surface Structure and Interpretation*. The MIT Press, Cambridge, MA.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press, Cambridge, MA.

Steedman, M., Baker, S., Clark, S., Crim, J., Hockenmaier, J., Hwa, R., Osborne, M., Ruhlen, P., and Sarkar, A. (2002). Semi-supervised training for statistical parsing: Final report. Technical report, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD.

Vadas, D. and Curran, J. R. (2008). Parsing noun phrase structure with CCG. In *Proceedings of the 46th Meeting of the ACL*, pages 335–343, Columbus, Ohio.

van Noord, G. (2009). Learning efficient parsing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL-09)*, pages 817–825, Athens, Greece.

Wood, M. M. (1993). *Categorial Grammars*. Routledge, London.

Wu, Z. and Palmer, M. (1994). Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138, Morristown, NJ, USA. Association for Computational Linguistics.

Xia, F., Palmer, M., and Joshi, A. (2000). A uniform method of grammar extraction and its applications. In *Proceedings of the EMNLP Conference*, Hong Kong.

Yarowsky, D. (1993). One sense per collocation. In *HLT '93: Proceedings of the workshop on Human Language Technology*, pages 266–271, Morristown, NJ, USA. Association for Computational Linguistics.

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.