Statistical parsing with non-local dependencies

Péter Dienes

Dissertation zur Erlangung des Grades eines
Doktors der Philosophie der Philosophischen
Fakultäten der Universität des Saarlandes

Saarbrücken, im April 2003

Verfasser:	Péter Dienes dienes@coli.uni-sb.de	
Betreuer:	Prof Dr. Hans Uszkoreit Miles Osborne, PhD	(Universität des Saarlandes) (University of Edinburgh)
Berichterstatter:	Prof Dr. Hans Uszkoreit Prof Dr. Dietrich Klakow Miles Osborne, PhD	(Universität des Saarlandes) (Universität des Saarlandes) (University of Edinburgh)

Erstellungszeitraum: 01.04.2001 – 20.04.2004

Abstract

Non-local dependencies occur when a head (e.g. a verb) and its dependent (e.g. an argument) is separated by some intervening material which does not depend on the same head. Because non-local dependencies play an important role in determining predicate–argument structure of sentences, identifying them correctly is essential for Natural Language Processing. However, this task appears to be rather difficult, and for this reason, non-local dependencies have received only limited amount of attention in the statistical parsing literature.

This thesis explores the problems identifying non-local dependencies poses for statistical parsing technology. We argue that the difficulties are due to the enlarged search space, an effect of the large amount of locally unresolvable ambiguities introduced by non-local dependencies. We show that the search space can be efficiently reduced by taking lexical and local information into account. We present several simple parsing models incorporating this knowledge.

We claim that non-local dependencies in English can be efficiently and accurately recovered by an appropriate combination of shallow approaches. In particular, we show that a finite-state machine without explicit knowledge of phrase structure information is able to detect heads participating in non-local constructions with state-of-the-art accuracy. This machine is employed to constrain the search space of a phrase-structure parser. The parser, when coupled with the finite-state preprocessor, is fast and achieves the best reported results on recovering non-local dependencies.

The accuracy of the system crucially depends on the way the parser and the preprocessor are combined. We develop a novel probabilistic framework where a preprocessor guides a parser through imposing soft constraints on the search space. The parser takes not only the hypotheses of the finite-state machine into account, but also incorporates the preprocessor's probability estimate for these hypotheses, and thus improves its own estimate for the given structure. This combination method attains our goals: the system is simple and, at the same time, efficient and accurate.

Zusammenfassung

Als nichtlokale Abhängigkeiten bezeichnet man Fälle, in denen ein syntaktischer Kopf (z.B. ein Verb) von seinem Dependenten (z.B. einem Argument) durch dazwischenliegendes Material, das nicht vom selben Kopf abhängt, getrennt ist. Da nichtlokale Abhängigkeiten eine wichtige Rolle für die Bestimmung von Prädikat–Argument Strukturen spielen, ist deren korrekte Erkennung eine Grundvoraussetzung für die natürliche Sprachverarbeitung. Diese Aufgabe scheint jedoch ziemlich schwierig zu sein, und deswegen wurde nichtlokalen Abhängigkeiten in der Literatur über statistisches Parsing nur wenig Beachtung geschenkt.

Diese Arbeit erforscht die Probleme, die das Erkennen nichtlokaler Abhängigkeiten mit statistischen Parsing-Technologien mit sich bringt. Wir argumentieren, dass diese Probleme von einem vergrößerten Suchraum herrühren, einer Folge der großen Anzahl von lokal nicht aufzulösender Ambiguitäten, die von nichtlokalen Abhängigkeiten hervorgerufen werden. Wir zeigen, dass der Suchraum durch den Einbezug lexikalischer und lokaler Information effizient eingeschränkt werden kann. Wir beschreiben mehrere einfache Parsing-Modelle, die solches Wissen einbeziehen.

Wir behaupten, dass nichtlokale Abhängigkeiten im Englischen durch eine angemessene Kombination von flachen Ansätzen effizient und mit hoher Genauigkeit entdeckt werden können. Insbesondere zeigen wir, dass ein endlicher Automat ohne explizites Wissen über Phrasenstrukturinformation in der Lage ist, syntaktische Köpfe, die an nichtlokalen Konstruktionen beteiligt sind, mit einer Genauigkeit, die dem Stand der Technik entspricht, zu entdecken. Dieser Automat wird benutzt, um den Suchraum eines Phrasenstrukturparsers einzuschränken. Der Parser, verbunden mit dem endlichen Automat als Präprozessor, ist schnell und erreicht die besten berichteten Ergebnisse für das Entdecken nichtlokaler Abhängigkeiten.

Die Genauigkeit des Systems hängt entscheidend davon ab, wie Parser und Präprozessor kombiniert werden. Wir entwickeln einen neuen statistischen Ansatz, in dem ein Präprozessor den Parser durch das Auferlegen weicher Einschränkungen durch den Suchraum führt. Der Parser berücksichtigt dabei nicht nur die Hypothesen des endlichen Automaten, sondern bezieht auch dessen Wahrschein-

lichkeitsschätzungen mit ein und verbessert damit seine eigenen Schätzungen für eine gegebene Struktur. Diese Kombinationsmethode führt zum Erreichen unseres Ziels: das System ist einfach und gleichzeitig effizient und genau.

Acknowledgements

Writing a PhD thesis is hardly a solitary enterprise, even if only one name appears on the title page. I would like to thank everyone who helped me to get to the point where LaTeX is formatting 240-odd pages.

I am immensely grateful to Amit Dubey, who has been a great fellow to work with. This thesis contains many ideas developed during innumerable hours of excited discussion with him. Additionally, I thank Amit for giving me prompt support with programming issues and with using his unlexicalised parser.

I am very much indebted to my supervisors, Miles Osborne and Hans Uszkoreit, who provided me with the necessary guidance, push, feedback, criticism and support, and who asked the right questions at the right time. Their comments have improved both my results and the readability of the dissertation.

Many thanks go to Jason Baldridge, Andreas Eisele, Frederik Fouvry, Geert-Jan Kruijff, Mirella Lapata, and Shravan Vasishth, who read the manuscripts of this dissertation and of earlier papers and managed to make them readable. They were also the people who helped me patiently with everyday problems such as finding a reference, an Emacs package or a LaTeX style. And I really appreciate Shravan's determination to improve my English.

I have been very lucky to be a member of the European/International Postgraduate College (EGK); I thank Matt Crocker for running the EGK and the German Science Foundation (DFG) for funding it. The feedback I got from my fellow EGK-students during the annual meetings and other talks, or even during the discussions we had on the corridor, has a large impact on the research presented here. Cheers folks on both sides of the Channel!

Csaba Oravecz initiated me into Computational Linguistics. We also worked together for a couple of years in Budapest, and I cannot tell how much I learned from him. I am also very much indebted to the members and students of the Theoretical Linguistics Department in Budapest for the inspiration and education I got there. I am grateful to Jean Senellart for devoting his time to teach me the basics of programming during my stay at Systran.

Finally, I would like to thank my family and my friends: Ádám, Balogh, Bidru, Bözse, Dubi, Eli, Gergice, J, Jochen, Malte and Ute, Nina, Philipp, Olli, Stefan,

Tobi, and Torgrim, who made it possible to have two homes, both in Germany and in Hungary.

Saarbrücken, April 2004

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Non-local dependencies occur when a head (e.g. a verb) and its dependent (e.g. an argument) is separated by some intervening material which does not depend on the same head. Such constructions have always been a major concern for linguistic theories. As Sag (1982) puts it:

> Few linguists would take seriously a theory of grammar which did not address the fundamental problems of English that were dealt with in the framework of 'standard' transformational grammar by such rules as There-insertion, It-extraposition, Passive, Subject–Subject raising, and Subject–Object raising. (Sag 1982, p. 427)

Such non-local phenomena including unbounded dependencies (Gazdar 1981) are the topic of the present dissertation. Our goal, however, is not linguistic in nature: we do not intend to argue for or against linguistic theories explaining the phenomena. Instead, we investigate the difficulties non-local dependencies (NLDs) present for parsing technology by designing more and more refined systems capturing NLDs. Our design is guided by three criteria: accuracy, efficiency and simplicity. Our final system combines simple and thus fast models to achieve state-of-the-art accuracy in handling NLDs.

Although English is generally viewed as a strictly configurational language, where arguments of a given predicate occur in well-defined neighbouring positions in the syntactic structure, it often exhibits dependencies which do not conform to this pattern. Specifically, some of the dependencies are non-local, as in the following example:

It is difficult to understand what I want to do. (1.1)

Here, the one-word phrase *I*, which is the immediate dependent (subject) of the headword *do*, is separated from its head by the sequence *"want to"*, where neither words in the sequence depend on *do*.

Identifying such dependencies is very important for a number of reasons. First, non-local dependents are often semantic arguments and, as a consequence, they should show up in the predicate–argument structure. Moreover, knowing the dependency structure of the sentence, including both local and non-local dependencies, might prove to be useful in a number of tasks, such as question answering, information retrieval, machine translation, language modelling, lexical acquisition, etc.

For instance, translating the sentences

I find it difficult to remember you. (1.2)
I promised John to remember you.

with four online machine translation engines illustrates the importance of handling NLDs (Table 1.1). Each sentence contains a non-local construction: the non-local subject of *remember* is *I*. Crucially, the word *remember* is translated with reflexive verbs in both German and French, where the reflexive pronouns agree in number and person with the (possibly non-local) subject of the clause. The verbs in the correct translations, *erinnern* and *rappeller* or *souvenir*, should occur with first person singular reflexive pronouns (*mich* and *me*). As it turns out, none of the translation engines get both examples right: they use third person reflexive pronouns (*sich* and *se*) instead of the first person ones. Incorporating a correct analysis of NLDs would substantially improve the systems.[1]

Given the common belief that English generally conforms to phrase structure rules enforcing locality, the frequency of non-local dependencies is surprisingly high. In our development data, Section 0 of the Wall Street Journal corpus in the Penn Treebank (Marcus *et al.* 1993, 1994), around three quarters of the sentences contain NLDs. When looking at verbs, 22% of the verbal arguments and 31% of the subjects are non-local. If our ultimate goal is to recover predicate–argument structure, we cannot ignore NLDs: they do play an important role in the meaning of the sentence.

The amount of attention devoted to NLDs in the statistical parsing community lags far behind their importance, although recently the interest in them has considerably increased (Chapter 3). With one notable exception, namely Model 3 of Collins (1997, 1999), which does handle a small subset of NLDs, the most

[1]The websites of the engines are:

SYSTRAN	`http://www.systransoft.com/`
PROMT ONLINE	`http://translation2.paralink.com/`
REVERSO	`http://www.reverso.net/`
LOGO MEDIA	`http://www.logomedia.net`

CORRECT	Ger.	Ich finde es schwierig, **mich** an Sie zu erinnern. Ich versprach John, um **mich** an Sie zu erinnern.
	Fr.	Je le trouve difficile à **me** vous rappeler. J'ai promis à John de **me** vous rappeler.
SYSTRAN	Ger.	Ich finde es schwierig, **sich** an Sie zu erinnern. Ich versprach John, um mich an Sie zu erinnern.
	Fr.	Je le trouve difficile de **se** rappeler vous. J'ai promis John pour me rappeler vous.
PROMT ONLINE	Ger.	Ich finde es schwierig, mich an Sie zu erinnern. Ich versprach John, **sich** an Sie zu erinnern.
REVERSO	Fr.	Je trouve difficile de me vous rappeler. J'ai promis à John de **se** vous rappeler.
LOGO MEDIA	Ger.	Ich finde es schwierig, **sich** an Sie zu erinnern. Ich versprach John, mich an Sie zu erinnern.
	Fr.	Je le trouve difficile de **se** souvenir de vous. J'ai promis à John de **se** souvenir de vous.

Table 1.1: Machine translation to German (Ger.) and French (Fr.) of the sentences *"I find it difficult to remember you."* and *"I promised John to remember you."*

popular probabilistic phrase-structure parsers simply ignore non-local dependencies (Charniak 1996). They typically return a syntactic structure for the sentence in (1.1) as illustrated in Figure 1.1. There are three important pieces of information which are not straightforward to recover from such a representation:

(i) *What is the object of do?* The analysis suggests that *do* is intransitive: there is no apparent indication of an object. How does this structure express that the object is, in fact, *what*?

(ii) *What is the subject of do?* Again, it is not straightforward to find out that it is actually *I*.

(iii) *What is the subject of understand?* How does the representation encode that the subject is not in the sentence? How is the difference between the verbs *do* and *understand* captured by the structure?

There are two possible explanations why researchers avoid addressing the problem of non-local dependencies: either these constructions are very easy to recover once we have a correct phrase-structure tree, and thus their recovery requires no scientific effort, or it is very difficult to handle them, and therefore, lacking an appropriate model, they are ignored for the sake of simplicity or efficiency. Johnson's (2002) experiments appear to support the latter claim: even if one knows the perfect phrase-structure, it is not straightforward to recover the NLDs in the sentence. The fact that all recent attempts in the statistical parsing literature employ more powerful grammar formalisms than a simple context-free grammar also points in the same direction: handling non-local dependencies requires extra generative power and, as a consequence, imposes an additional burden on sentence processing.

The present dissertation challenges this view. We will show that a combination of very simple tools, such as a context-free parser and a finite-state machine, can achieve the goal of recovering non-local (as well as local) dependencies efficiently with state-of-the-art accuracy.

1.2 The problem

Employing phrase-structure grammars to recover non-local dependencies requires a *configurational* representation of NLDs. In this scheme, verb phrases with an extracted object, for instance, must have a different structure from intransitive VPs: either the non-terminals should be different or the tree itself has to have a different shape (or both). In the corpus we use in our experiments, the Penn Treebank

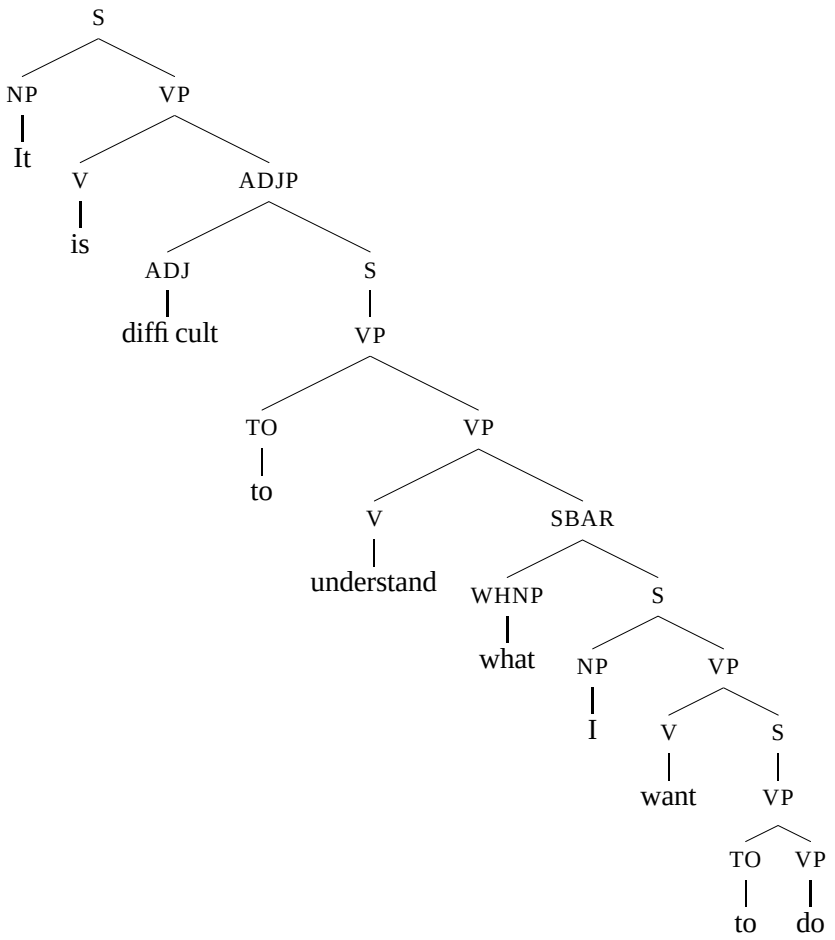


Figure 1.1: A typical output of a statistical parser for the sentence *“It is diffi cult to understand what I want to do.”*

(Marcus *et al.* 1993, 1994), NLDs are represented with the aid of *empty elements* (EEs) and *co-indexation*.

As an illustration, consider the tree in Figure 1.2.[2] The idea underlying the configurational representation is that the labelled tree in itself determines what a predicate is and what its arguments are. For instance, an NP with a sister VP immediately dominated by an S nonterminal is the *subject* of the main verb in the VP. In the present example, the subject of *want* is *I*, by virtue of its position with respect to the VP. Now, how does this scheme represent the fact that the subject of *do* is also *I*? Looking at the tree we see that there is something sitting in the subject position of the clause *"to do"*: an *empty element* acting as a placeholder which indicates that *do* has a subject. This empty element is *co-indexed* with the actual subject of *do*, that is, with the one-word phrase *I*.

In a similar vein, it is assumed that roughly every NP which is a sister of a verb immediately preceding it is the *(direct) object* of the verb. In the figure, *do* appears to be followed by such an NP. This NP, however, contains only an EE as a placeholder which is co-indexed with the actual direct object, *what*. Note, however, that not all empty elements are co-indexed: in the subject position associated with *understand*, we observe such a placeholder. This configuration indicates that *understand* does have a subject, but it is not in the sentence and gets arbitrary interpretation.

Such a representation scheme suggests a natural division of the problem of recovering NLDs into two subtasks: we can regard the task of finding where EEs occur and finding what they are co-indexed with (if with anything at all) as two separate problems. Such a division of labour, as we will see throughout this thesis, turns out to be extremely useful. In what follows, we refer to the first task as EE *detection*, whereas to the second one as *antecedent recovery*.[3]

Both EE detection and antecedent recovery appear to be challenging, albeit for different reasons. In the case of EE detection, the problem is obvious: empty elements are hidden, they do not appear in the input. Therefore, the parser has to guess where EEs might or might not occur; in the worst case, it might end up positing an unbounded number of empty nodes (see e.g. Johnson and Kay 1994). This is a problem since the parser might not even terminate, or at least it might be very slow. Co-indexation also seems to be a fundamental problem, because it might

[2]For the sake of clarity, we abstract away from some minor details and complications in the presentation below.

[3]We use the term *antecedent* somewhat liberally in this dissertation referring to the co-indexed material, even if it follows the EE, such as in the case of right dislocation.

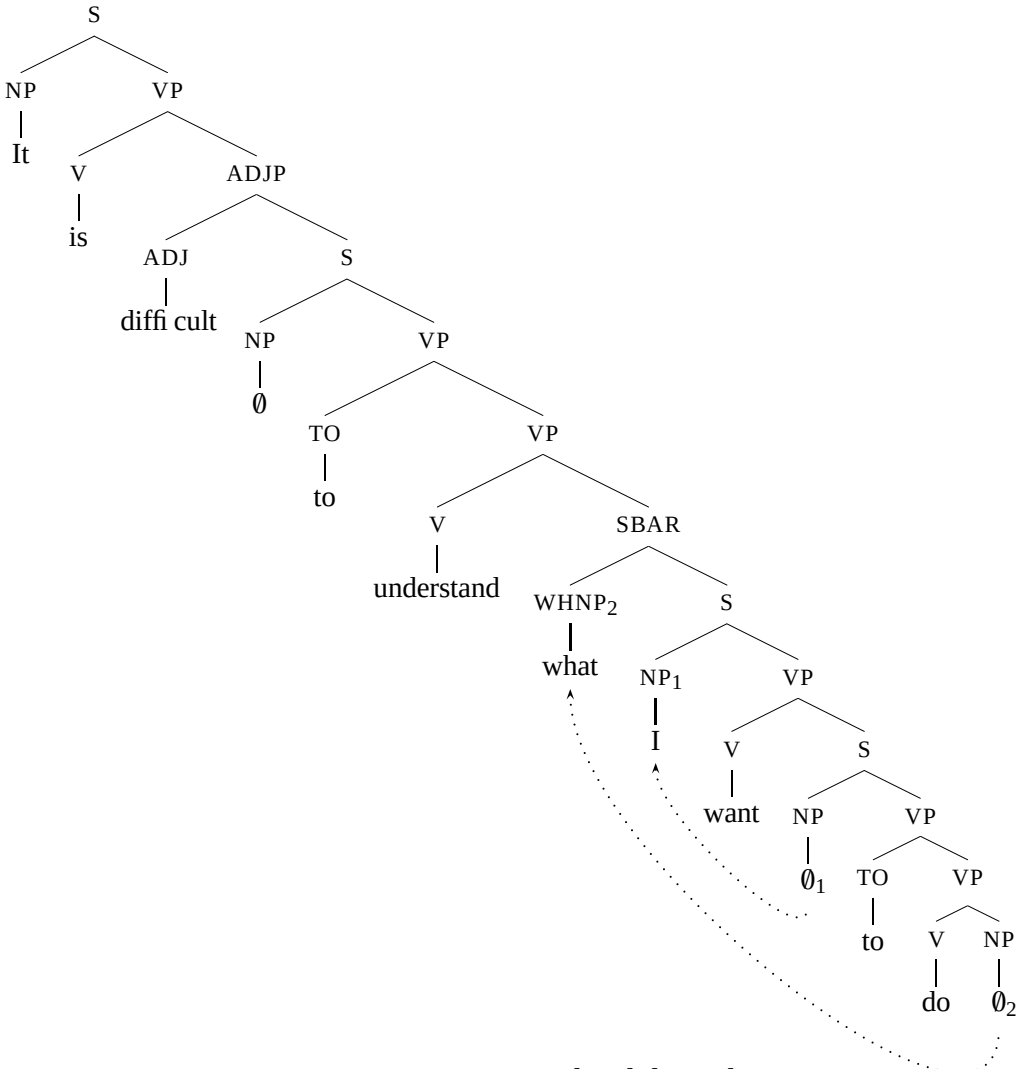


Figure 1.2: Representing non-local dependencies.

introduce extra generative power to the grammar.[4] This suggests that handling NLDs, i.e., parsing with empty elements, is a difficult task.

Are empty elements necessary for representing NLDs? Many linguistic theories, e.g. Head-Driven Phrase Structure Grammar (Pollard and Sag 1994) or Combinatorial Categorial Grammar (Steedman 2000), are able to analyse NLDs *without* empty elements (although they still heavily rely on indices). Would parsers designed for these frameworks also find it difficult to parse with NLDs? In the present thesis, we argue that they would, because the problem roots much deeper than the issue of how NLDs are represented. The difficulty that *all* parsers handling NLDs have to face is the considerable increase in the search space: NLDs introduce a large amount of local ambiguity which might only be resolved late in the derivation. For instance, a verb might be ambiguous between intransitive or transitive with an extracted object. Now, it is possible that the parser has to process the whole sentence in order to decide for either alternative (cf. Section 2.2). In practice, when given such a sentence as input, a parser handling NLDs has to explore a search space twice as large as for a parser without the ability to deal with NLDs. Should there be more sites with non-local ambiguity, the search space would increase rapidly with the number of such sites.

The large search space, in general, has at least two unwelcome effects. The first problem concerns *efficiency*: the larger the search space is, the slower the parser runs. In fact, even parsers not designed to handle NLDs cannot afford to explore their whole search space and require heuristic search to filter out unlikely search states. The second problem concerns *accuracy*: the larger the search space is, the harder it is for the parser to find the right solution. Therefore, if we manage to reduce the search space reliably, we have good chances to kill two birds with one stone: we might have a more accurate and more efficient parser which is able to handle non-local dependencies. In the present dissertation, we set out to achieve exactly this goal.

1.3 Contributions

Our ultimate goal is to design a fast and accurate parsing architecture which recovers non-local as well as local dependencies. Our thesis is that this **goal is best achieved by a careful combination of simple systems**. Specifically, we show that a probabilistic combination of a finite-state preprocessor and a lexicalised context-free parser is very successful at this task: the system handles non-local

[4] It is easy to write a grammar with indices that generates languages that are beyond context free, provided we allow for an unbounded number of such indices.

dependencies with state-of-the-art accuracy with only a small amount of overhead in time and space.

Further contributions of the present dissertation are the following results:

- Recovering antecedents proves to be relatively easy; even an unlexicalised PCFG parser is able to perform this task reliably and efficiently if an oracle informs it about the sites where empty elements occur in the input.

- Finding sites for non-local dependencies can be efficiently and reliably achieved without explicit knowledge of phrase structure information; we present a finite-state machine, the trace tagger, which is able to find EEs prior to parsing.

- Lexicalisation and local cues are the most important factors contributing to the success of the trace tagger.

- Even the simplest combination of the preprocessor and the unlexicalised parser outperforms the best reported results on the task of NLD recovery; the resulting model is fast as well.

- We present a lexicalised parser which is able to handle the majority of NLDs. This parser is a generalisation of Model 3 of Collins (1997, 1999) and it outperforms the combination of the trace tagger and the unlexicalised model, albeit only by a small margin.

- The trace tagger is more accurate at EE detection than the lexicalised parser, because it explicitly takes into account information not available for the parser. Specifically, non-local information as well as information cutting across phrase boundaries play an important role in EE detection.

- We propose a general probabilistic architecture for combining a preprocessor and a parser; in particular, the architecture allows the preprocessor to impose soft constraints on the parser and guide it during its search. When tested on recovering NLDs, this system outperforms all previous systems by a considerable margin.

1.4 Overview

In Chapter 2, we define in a data-oriented way what exactly we mean by non-local dependencies in the present dissertation. Then, we discuss why processing NLDs is a problem for any parsing algorithm based on any underlying linguistic

framework. Section 2.3 presents a technique which enables us to reliably approximate NLDs with a context-free parser. We use this technique to convert the original trees into trees without co-indexation; the new corpus, the *Tracebank*, serves as our training and test data. Finally, we give a brief inventory of non-local dependencies occurring in our corpus. A more detailed description of non-local phenomena is presented in the Appendix.

Chapter 3 is devoted to a discussion of previous work addressing the problem: we review how machine learning approaches have been used for recovering non-local dependencies.

Chapter 4, first, introduces the evaluation metrics we use throughout the thesis. Then, we present an algorithm recovering antecedents: this algorithm operates on fully specified tree structures of the Tracebank. The main focus of the chapter, however, is the design of an unlexicalised PCFG parser which is able to detect NLDs. The performance of this parser turns out to be poor: it is inaccurate, slow and cannot parse 40% of the test data. Interestingly, the same parser proves to be accurate, fast and robust once it is informed where EEs occur. Therefore, we conclude that antecedent recovery is relatively easy and that the main bottleneck is EE detection.

As a consequence, in Chapter 5, we present a system which provides the parser with information about the EE sites. Since this module is a preprocessing unit before parsing, it is not allowed to use explicit phrase structure information. Therefore, we cast the problem of detecting EEs as a tagging problem and develop a finite-state tagger, the trace tagger to perform this task. The trace tagger achieves state-of-the-art accuracy on the EE-detection task. We give a rigorous analysis of its performance in order to isolate a handful of cues which contribute to its success. These cues might be straightforwardly incorporated into other systems to improve their accuracy.

Chapter 6 combines the trace tagger and the parser to perform the task of recovering NLDs accurately and efficiently. We use a simple pipeline architecture: the best hypothesis of the tagger serves as the input to the parser. We also test the combined architecture using a state-of-the-art probabilistic lexicalised parser, that of Collins (1999). Interestingly, using this parser improves the scores only by a few percent, although it is much more accurate at building phrase structure. The reason for this, we argue, is lexicalisation: by using a lexicalised trace tagger we implicitly lexicalise the unlexicalised parser. Further lexicalisation, i.e., employing a lexicalised parser, has only limited contribution: in a way, the trace tagger and the lexicalised parser incorporate the same cues.

This observation, however, suggests that a lexicalised parser might be more successful at detecting NLDs than its unlexicalised counterpart. Therefore, in Chapter 7, we develop a lexicalised parser which is able to handle NLDs. Specifically, we generalise Model 3 of Collins (1997). This parser proves to be success-

ful: it does not suffer from the problems exhibited by the unlexicalised parser. Interestingly, the lexicalised parser is worse at detecting EEs than the trace tagger, which shows that it still could benefit from consulting the tagger. However, the parser turns out to be more accurate at recovering antecedents than the combined model of Chapter 6. We argue that the problem is the rigid pipeline architecture.

Therefore, in Chapter 8, we present a general probabilistic framework to combine shallow modules with a parser. In this architecture, the shallow module provides its *k*-best hypotheses, which guide the parser throughout its search. We use this architecture to combine the trace tagger and the parser. Such a combination method is successful: the system is more accurate at detecting NLDs than either of its components. A further benefit of the architecture is the reduction of the search space the parser has to explore. In another experiment, where a simple yet fairly accurate NP chunker is combined with the parser, the search space is almost half of the original size, yet the combined system is more accurate.

Finally, in Chapter 9, we draw our conclusions and sketch possible directions for future research.

Chapters 4–7 are extensions of joint work with Amit Dubey (Dienes and Dubey 2003a,b). These chapters largely follow the structure of the two papers, and they contain a wider set of experiments and updated results.

Chapter 2

Non-local dependencies

Although the linguistic community has long been interested in non-local dependencies (Chomsky 1957, 1965, 1977, Harman 1963, Ross 1967, Bresnan 1976, Kaplan and Bresnan 1982, Gazdar 1981, Gazdar *et al.* 1985, Pollard and Sag 1994, Hudson 1990, Steedman 1996, 2000, etc.), there is little consensus on what should be regarded as a non-local dependency. Most theories agree that unbounded dependency constructions, like the ones exemplified by WH-questions and relative clause in English, qualify as such; but this is where the agreement ends. Chomskyan transformational grammar, for instance, treats passive and control as involving non-local dependencies, i.e., as movement (Chomsky 1965), whereas most other formalisms do not. Others, for instance Hockenmaier (2003a,b), consider auxiliary constructions (locally mediated) NLDs. Although this point of view makes perfect sense under the assumption that the main verb depends on the auxiliary, it is a problematic claim if our theory views the auxiliary as the modifier (i.e., dependent) of the main verb. In fact, Falk (2003) argues that some of the auxiliaries in English are heads, while others are modifiers. Although we do not think that such choices would alter our methodology or make the task considerably easier or harder, they would definitely affect the actual scores we obtain.

Clearly, it is rather problematic to define the notion of non-local dependencies on purely linguistic grounds independent of any theory: probably all linguistic frameworks would classify considerably different sets of phenomena as NLDs. Even if we accepted one underlying theory, the problem would not be solved, since there are many debated theory-internal issues which might alter the notion of non-locality. This indeterminacy would make it very difficult if not impossible to compare the results presented here to other approaches based on even a slightly different underlying linguistic framework.

Therefore, we are forced to take a different, more theory-natural position: in the present work, we let our data define non-local dependencies. Over the years, the Wall Street Journal part of the Penn Treebank corpus Marcus *et al.* (1993,

1994) has emerged as a widely accepted benchmark for training and testing statistical parsers and other systems. Although the annotation scheme is clearly influenced by transformational-grammar, which treats non-local dependencies in terms of movement and represents them with empty elements and co-indexation, the data is now written in stone, which allows us to propose a fixed, theory-independent, albeit corpus-specific, definition of non-local dependencies. In particular, we regard every dependency construction involving an empty element as a NLD. As we discuss it in Section 2.4, this definition is rather broad and includes phenomena such as unbounded dependencies, passive, raising and control, and even some idiosyncratic constructions in the Treebank, such as empty units or ellipsis. Nevertheless, we adopt it as a working definition here, which facilitates straightforward comparison with previous (Johnson 2002, Jijkoun 2003, Dienes and Dubey 2003a,b) and possible future work.

It is important to emphasise, however, that we do not intend to argue for (or against) the linguistic theory underlying the annotation. We also do not aim at entering the discussion about the psychological reality and linguistic adequacy of movement and empty elements. We regard these questions as representational issues beyond the scope of the present enquiry, and focus our attention on the difficulties NLDs present for parsing technology instead. Nevertheless, we find the *metaphors* of movements, empty elements, gaps, and traces as convenient ones to describe the phenomena and use them frequently throughout this dissertation.

In Section 2.1, we introduce the general scheme the Penn Treebank employs to represent non-local dependencies. Then, in Section 2.2, we discuss why non-local dependencies are hard to parse regardless of how they are represented and of the kind of parsing algorithm employed. In Section 2.3, we show how the original annotation can be converted to a traceless scheme which can be parsed with a context-free parser with only minimal loss of information. We refer to the modified corpus as the *Tracebank*. Finally, we give a brief overview of various kinds of non-local phenomena as defined by our data. A more detailed inventory is presented in the Appendix.

2.1 Non-local dependencies in the Penn Treebank

In this dissertation, we follow the general practice of using the Penn Treebank (Marcus *et al.* 1993, 1994) to train and test our models. Specifically, our training set of approximately 1 million tokens (around 40000 sentences) consists of Sections 2–21 of the Wall Street Journal part of the corpus (WSJ02–21), whereas the results are presented testing the systems on Section 23 (WSJ23, 56684 tokens, 2416 sentences) of the same part. Finally, we reserve Section 0 (WSJ00, 46451 to-

```
(S (NP-SBJ (NP It)
           (S *EXP*-1))
   (VP is
       (ADJP-PRD difficult
                 (S-1 (NP-SBJ *)
                      (VP to
                          (VP understand
                              (SBAR (WHNP-2 what)
                                    (S (NP-SBJ-3 I)
                                       (VP want
                                           (S (NP-SBJ *-3)
                                              (VP to
                                                  (VP do
                                                      (NP *T*-2)
                                                  ))))))))))))
```

Figure 2.1: The Penn Treebank representation of the sentence in (2.1).

kens, 1921 sentences) as a development set, to optimise parameters of the model and to carry out error analysis.

One advantage of using the Penn Treebank (PTB) is that it not only describes local syntactic structure but also handles non-local dependencies and thus represents predicate–argument structure (Marcus *et al.* 1994). To illustrate the annotation scheme, let us return to our example from Section 1.1, repeated here for convenience:

It is difficult to understand what I want to do. (2.1)

This sentence would be represented in the PTB as in Figure 2.1. There are several things to observe. First, the PTB annotation scheme follows a structural representation of predicate–argument structure: every node without a special functional label that is immediately dominated by a VP is the complement of the verb. Similarly, every NP under an S node bearing the functional label -SBJ is the subject of the sentence. Therefore, the subject of the verb *want* in the clause is *I*, because it sits in the subject position of the corresponding S node. Similarly, by virtue of its position in the phrase structure, the complement of the same verb is *"to do"*. Hence, the subclause is associated with the following (partial) predicate–argument structure (cf. Marcus *et al.* 1994):

`want(I,"to do")` (2.2)

Now, this approach faces a problem when it comes to non-local dependencies: how to represent entities that are arguments of more than one predicate? The an-

notation scheme takes a relatively theory neutral stance and marks those places where a constituent is interpreted but does not occur with an empty element `*` or `*T*`.[1] Thus, for instance, the clause *"to do"* has, in fact, a subject which happens to be empty. In order to make the recovery of the argument possible, the annotation scheme uses co-indexation: the phrase to be interpreted as the subject of the clause bears the same index as the empty element. Consequently, the subject of *do* in the present example is the constituent bearing the index 3, i.e., *I*. Similarly, the direct object of *do* is *what*. The partial predicate–argument structure of the sentence is:

```
want(I,do(I,what))
```
(2.3)

Some of the empty elements (henceforth EEs, but sometimes referred to as *traces* or *gaps*) are not co-indexed though, that is, they do not have an antecedent. In the present example, for instance, the subject of the clause *"understand what I want to do"* is not co-indexed – it gets an arbitrary interpretation: someone or something. Thus, the predicate–argument structure of the clause is something like:

```
understand(*someone*,want(I,do(I,what)))
```
(2.4)

Finally, observe another EE-type, labelled as `*EXP*`: the annotation scheme uses four further EEs to represent discontinuous structures. Discontinuity poses a problem for the essentially context-free representation of the Treebank. To handle the problem, Marcus *et al.* (1994) propose marking the relation between the parts of the discontinuous material with an empty element co-indexed with the corresponding part. In the example, since predicative uses of adjectives are viewed to apply on the subjects,[2] the clause *"understand what I want to do"* is viewed as part of the subject, and therefore, it is co-indexed with an empty element which is part of the subject. Now, the structure is parallel to structures without raising, and the predicate–argument representation is assumed to be:

```
difficult(understand(*someone*,want(I,do(I,what))))
```
(2.5)

In the present dissertation, we regard the task of recovering non-local dependencies as consisting of three subtasks: (i) detecting where empty elements occur, (ii) deciding whether they are co-indexed, i.e., whether they have an antecedent,

[1] In the annotation scheme, `*` roughly stands for **PRO**s and `*T*` for WH-movement. In Section 2.4, we give an overview of the various types of empty elements used in the Treebank.

[2] That is, in the sentence *"The reader is tired"*, the predicate is *tired* and its argument is the subject, *the reader*.

Type	Percentage empty
Arguments	22%
Subjects	31%
Verbal complements	16%
Sentences	75%
Words	7%

Table 2.1: Various statistics of EEs in WSJ00.

and (iii) finding the antecedent if one is required. In fact, as we discuss in Sections 2.3 and 2.4, the type of the EE determines whether it has an antecedent or not. Therefore, we treat the subtasks (i) and (ii) as one, and refer to it as EE *detection*. Finding the antecedents will often be called *antecedent recovery*. We often use the term NLD *site* for the immediate environment where an EE occurs.

In Section 1.1, we argued that finding non-local dependencies is very important for a number of tasks. Now, we are in the position to quantify this intuition. Table 2.1 shows a handful of statistics describing the distribution of EEs. The most important observation is the surprisingly high occurrence of EEs in semantically "interesting" positions: more than one fifth of the total verbal arguments is an EE, and almost one third of the subjects is empty. This shows that recovering non-local dependencies is indeed very important if we define our aim as reconstructing predicate–argument structure.

2.2 The problem with NLDs

Intuitively, the problem of parsing with NLDs is that the empty elements representing these dependencies are not in the input. Therefore, the parser has to hypothesise where these EEs might occur – in the worst case, it might end up suggesting exponentially many traces, rendering parsing infeasible (Johnson and Kay 1994). One apparent solution would be to adopt a linguistic theory which does not treat non-local dependencies in terms of empty elements and co-indexation; in fact, most modern (non-transformational) theories explicitly deny the existence of empty elements.

We claim here, however, that the problem roots much deeper than this representational question. As a matter of fact, in the present dissertation, we do not intend to make any claim about the (in)appropriateness of analysing NLDs in terms of movement, or about the psychological/linguistic (ir)reality of phonologically empty elements. Instead, we are interested in the computational burden and diffi-

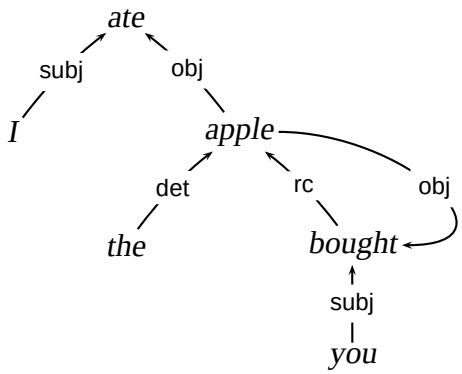


Figure 2.2: A dependency graph with a re-entrancy for the sentence "*I ate the apple you bought*."

culties NLDs impose on broad-coverage parsing systems. We follow e.g. Briscoe (1987) in using terms like movement, empty elements, traces, and gaps only as convenient metaphors for non-local dependency constructions.

Although various linguistic frameworks disagree on how to represent and analyse non-local dependencies, they all agree that these phenomena are challenging. In fact, the presence of such constructions in the world's languages is one of the major driving force for proposing new linguistic theories which can handle non-locality in a formal yet linguistically and maybe psychologically plausible way. In the present section, we discuss why non-local dependencies are hard to process efficiently with *any* parser using *any* underlying framework.

From the point of view of the dependency structure, NLDs are difficult because they violate the assumption that dependency structures are represented as directed trees. Specifically, NLDs give rise to re-entrancies in the dependency graph, i.e., it is no longer a directed tree but a directed graph, with nodes possibly having multiple parents (e.g. *apple* in Figure 2.2). Now, the parser has to explore a much larger search space: in a tree the number of edges is always $n-1$ (where n is the number of words in the sentence). On the other hand, in the case of the directed graph, the parser does not know the number of edges in advance: it might lie anywhere between $n-1$ and n^2-n. Moreover, observe the cyclic structure in Figure 2.2: the presence of such structures makes it difficult to use a dynamic programming approach. Probability estimation for general directed graph structures also turns out to be more complicated than for directed trees (Abney 1997).

Arguably, the search space is much more restricted by an actual grammar that exploits, for instance, the knowledge that *buy* is a transitive verb and thus requires a direct object. Nevertheless, the problem does not disappear. Consider the following example:

When demand is stronger than suppliers can handle **WH–ADVP** *and delivery times lengthen* **WH–ADVP**, *prices tend to rise.* (wsj_0036.mrg) (2.6)

This sentence illustrates two complications: first, the subcategorisation frame of a verb does not necessarily restrict where (some) NLDs may or may not occur, since the extracted material might be an adjunct. Second, one non-local dependent (*when* in the above example) might be associated with more than one head down in the tree (*lengthen* and *handle*) – the parser has no means to infer this before it actually processes both embedded verbs and the complementiser. The situation might be exacerbated by the fact that the WH-word is optional in some cases (e.g. in the sentence *"The way he walks* **WH–ADVP** *is funny"*).

In fact, the problem is that any parser either has to access an unbounded amount of left and right context when making its decision, or keep a large amount of ambiguity in memory until the whole sentence is processed (Briscoe 1987, Maxwell and Kaplan 1995a). As an illustration, consider the following examples taken from (Briscoe 1987); **WH–NP?** marks ambiguous sites for extracted WH-objects:

(a) Who do you want **WH–NP?** to succeed **WH–NP?** (2.7)

(b) Who do you want **WH–NP?** to replace **WH–NP**

(c) Who does Kim want **WH–NP?** to think that the boss expects the directors to replace Sandy (with **WH–NP**)

First, note that a top-down parser would ultimately fail analysing sentences with NLDs in general, because they might contain left-recursion (e.g. in the case of relative clauses). Therefore, we have to resort to a bottom-up or to a mixed parsing strategy (Johnson and Kay 1994). However, all known polynomial parsers (for CFGs) use a common underlying data structure, the chart, in essentially the same way (Sheil 1976, cited by Maxwell and Kaplan 1995a). Therefore, without loss of generality, we show that non-local dependencies are difficult for chart parsers.

An important property of a chart parser is that prior to the creation of an edge spanning a sequence of words, all edges spanning substrings of these words are created and stored (Kay 1986 (1980)). Consequently, by the end of the recognition process, all possible edges for all substrings of the input are built and stored. The efficiency of the chart parser is due to the fact that equivalent edges are created only once. "In the context-free case, two edges are equivalent if they span the same substring and impose exactly the same requirements for further matching the same rule" (Maxwell and Kaplan 1995a, p. 407).

Now, consider the sentence in (2.7a). It is ambiguous: the extracted object might be the object of either *want* or *succeed*, both verbs being ambiguous between transitive and intransitive. Even if the sentence is globally unambiguous

(as in 2.7b and c), the parser has to process the whole sentence before the local ambiguity can be resolved. Initially, the parser creates two edges for *want*: an intransitive one and a transitive one. (A similar requirement holds for *succeed* as well.) These two edges, however, behave identically with respect to most of the context-free rules: they combine (locally) with exactly the same edges. Their different behaviour becomes apparent only later in the derivation, when we try to combine them[3] with *who*: one of them can be combined with this word, the other one cannot.

In order to prevent the parser from treating the two edges as equivalent (even though they appear to be equivalent as far as most context-free rules are concerned), we have to keep them separate: we either associate a different nonterminal label with these edges or use a special marker on the one with extraction to indicate that the verb is not yet saturated. The latter solution captures the generalisation that both edges behave in the same way with respect to most of the context-free rules. Regardless of how we keep the two edges , however, all the edges dominating the verb *want* have to be doubled, although they are equivalent with respect to the majority of the context-free rules (except for the one combining them with *who*).

The situation further deteriorates when we join edges that cover both *want* and *succeed*: since we still do not know which of the two verbs are transitive or intransitive, we have to keep all four possibilities.[4] Again, these four edges behave identically with respect to the other edges that can be combined with them, except for the one dominating *who.* This means that we have to have four copies of all the edges spanning both *want* and *succeed.* Should there be another verb ambiguous between transitive or intransitive, we would be forced to have eight copies of all the edges containing the sites. Clearly, with the possibility of having even more sites and different extracted constituents (e.g. WH-adverbs as well as WH-complements), the size of the chart grows enormously. It is, therefore, not surprising that all parsing algorithms devised for the family of linguistically motivated formalisms generating mildly context-sensitive languages have at least $\mathcal{O}(n^6)$ worst-case time-complexity in the number of words n (Vijay-Shanker and Weir 1994). If large feature structures are associated with the nodes, the complexity might easily increase to exponential (Maxwell and Kaplan 1995a).

There is a very important difference between local (e.g. PP-attachment) and non-local ambiguities: in the former case, it never happens that two edges behave in the same way with respect to all but one edge, i.e., they span the same words and have the same non-terminal label. This means that even if the sentence is exponentially ambiguous, the parser has to store the ambiguous edge only once; the

[3] To be more precise, apparently equivalent edges dominating them.

[4] The example in (2.6) shows that we cannot discard any of them.

exponential number of readings poses a problem only if we have to unpack all possible analyses. Non-local dependencies, on the other hand, force us to keep several edges apart, even though they span the same words and have the same non-terminal label.

Let us conclude this discussion with a quote from (Kaplan and Maxwell 1995) justifying the introduction of functional uncertainty in LFG, a device which is designed to handle non-local dependencies.

> Uncertainty specifications are a compact way of expressing a large number of disjunctive possibilities that are uncovered one by one as our procedure operates. It might seem that this is an extremely expensive descriptive device, one which should be avoided in favor of apparently simpler mechanisms. But the disjunctions that emerge from processing uncertainties are real: they represent independent grammatical possibilities that would require additional computational resources no matter how they were expressed. In theories in which long-distance dependencies are based on empty phrase structure nodes and implemented, for example, by gap-threading machinery, ATN HOLD lists, and the like, the exact location of these empty nodes is not signalled by any information directly visible in the sentence. This increases the number of phrase structure rules that can be applied. What we see as the computational cost of functional uncertainty shows up in these systems as additional resources needed for phrase structure analysis and for functional evaluation of the larger number of trees that the phrase structure component produces. (Kaplan and Maxwell 1995, p. 194)

2.3 Context-free parsing and NLDs

In the previous section, we argued that processing NLDs is or might be difficult for any parser. But is it really difficult in practice? Which types of NLDs are difficult to process? What information could help a parser? In order to answer these questions, we present experiments with various probabilistic context-free parsers. The main motivation for choosing a PCFG parser is the relative simplicity of the model: although parsers based on other frameworks might be considered to be more adequate for handling NLDs, they usually allow a larger space of free parameters. It would be thus harder to determine the exact cause of the phenomena we observe.

Moreover, we believe that it is methodologically more adequate to start with a very simple system and understand why it does or does not work before moving to a more sophisticated one. This approach helps us understand the problem

more thoroughly and keep the search space for possible solutions relatively small. Given that many parsing algorithms for more sophisticated grammar formalisms actually use a context-free backbone, our results with a PCFG might generalise to these systems as well and help improve them.

Practical considerations also play an important role in our choice: PCFG models are straightforward to train from either labelled or unlabelled data; various smoothing techniques are investigated to fight data sparseness; efficient algorithms exist for finding the most probable parse; the resulting models are reportedly robust and coverage does not constitute a problem. Furthermore, large resources are available for training PCFGs, therefore no additional manual work is necessary to create them or to develop a wide-coverage grammar. Finally, PCFGs are very successful at handling surface dependencies in languages with relatively fixed word order, such as English (Collins 1999, Charniak 2000).

In order to handle NLDs in a context-free parser, two issues are to be addressed: empty elements and co-indexation. Treating empty elements in a parser does not present any major problem as long as the they are dominated by a branching node. Therefore, it is necessary to prune non-terminals dominating only empty material. Afterwards, it is fairly straightforward to modify any parsing algorithm to posit an empty node whenever a rule allows it. Note that by pruning non-branching empty non-terminals we considerably restrict the positions where EEs might occur and thus the search space the parser has to explore: the parser only hypothesises an empty element when a rule forces it to do so.[5]

Co-indexation, on the other hand, is much more problematic. Indices in the grammar might increase its generative power (Vijay-Shanker and Weir 1994). Unless we restrict where antecedents might occur, we might not be able to model NLDs with a context-free grammar. Fortunately, there tend to be strong restrictions on possible sites for antecedents in English, with the exception of pseudo-attachments (see below). In particular, the antecedent of an EE is required to *strictly c-command* the EE in the vast majority of the cases, where strict c-command is defined as:

> **Strict c-command** (Haegeman 1991, p. 122) (2.8)
> Node *A* strictly c-commands *B* if and only if
> (a) *A* does not dominate *B* and *B* does not dominate *A*; and
> (b) the first branching node dominating *A* also dominates *B*.

The above definition entails that on the path in the parse tree from the EE to its antecedent, we always go up, except for the very last step which is downward.

As a consequence of this restriction, co-indexation can be compiled into the rewrite rules: all the nodes that lie on the path from the EE to its antecedent get

[5] Pruning is done internally by our parsers.

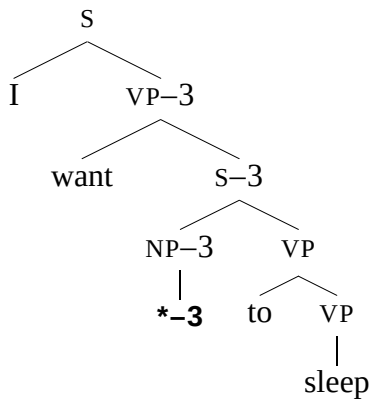


Figure 2.3: Connecting the antecedent and the EE by enriching the representation of nonterminals with the index.

the index (cf. Figure 2.3). Finding the antecedent of the EE is basically following the index up to the point where it disappears – the nonterminal dominated by this node is the antecedent for the EE.[6]

This approach, however, faces a problem: in principle, we could use any number (or any symbol) as an index, which means that the number of nonterminals is potentially infinite. Grammars of this type with infinitely many nonterminals generate languages that are not context-free (van Wijngaarden *et al.* 1975, Vijay-Shanker and Weir 1994). This problem does not show up in the present setting, however, since we extract our grammar from a finite training set, which ensures that we always have a finite set of nonterminals (and rules). Still, the potentially large number of nonterminals makes this approach virtually impossible to couple with a probabilistic setting: the resulting model would tremendously overfit the training data and would be incapable of generalisation – at least as far as NLDs are concerned. Yet, observe that we do not rely on the index itself to find the antecedent: we make use of the structure instead. Consequently, we can use the same marker (`gap`) to thread the EE to its antecedent (Figure 2.4). This approach is essentially what Collins (1997), following e.g. Harman (1963), Gazdar (1981), Gazdar *et al.* (1985), proposes to handle WH-movement; some minor differences are discussed in Section 7.2.2. This decision keeps the induced grammar context-free and reduces the problem of overfitting.

This technique is perfect when we only have one EE-type, which is not the case in the present situation. Consider the sentence in (2.1) and the corresponding tree in Figure 2.1: the clause *"what I want to do"* would be, under the present scheme, represented as in Figure 2.5. Such a representation would lose the in-

[6] As we discuss it shortly, this process is a bit more involved in the case of grammars with non-binary rules.

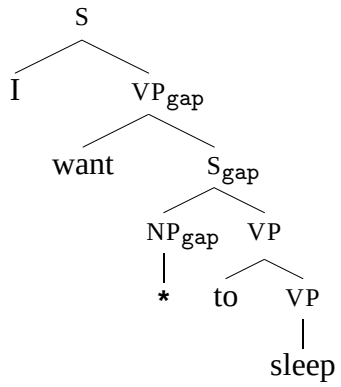


Figure 2.4: Connecting the antecedent and the EE by enriching the representation of nonterminals with a uniform symbol (`gap`).

formation which EE is co-indexed with which antecedent: is the WH-word to be interpreted as the subject or the object of *do*? To avoid this problem, we propose to use "typed" gap features, which indicate whether the EE is of type **WH** (corresponding the label `*T*` in the PTB annotation, see below for an inventory of EEs) or **PRO** (`*` in the PTB). This move also captures the insight that the distribution of the antecedents for WH-movement is different from that of the antecedents for **PRO**s, namely, they occur in different syntactic environments: WH-antecedents are in non-argument positions (under an SBAR node), whereas **PRO**-antecedents always occur in argument positions.

A further complication is due to the relatively shallow representation adopted in the PTB: in many cases, nodes have more than two daughters. Such a situation is shown in Figure 2.6 (the sentence is a simplified example from `wsj_0044.mrg`). In order to find the correct antecedent, we have to know that the appropriate landing site of the gap-variable on the VP node is an NP and not a PP. This information is encoded in the original tree by virtue of the NP node dominating the EE **PRO**. In order to be able to access this information locally, we include it on the gap variables as well as on the empty element. We refer to this scheme as using `gap+` variables. Figure 2.7 shows the representation the example sentence in (2.1) gets under the present scheme.

Although we adopt this approach throughout the present dissertation, we are aware of three important limitations. First, we heavily rely on the assumption that the antecedent strictly c-commands the corresponding EE. This is true of most of the antecedents, except for the cases of pseudo-attachments (cf. Section 2.4). Their antecedents more often than not do not strictly c-command them, that is, in our walk from the EE to the antecedent, first we have to move up in the parse tree, then more than one step down. Although there might be conceivable solutions to this problem, for the sake of simplicity, we opt for not connecting EEs of

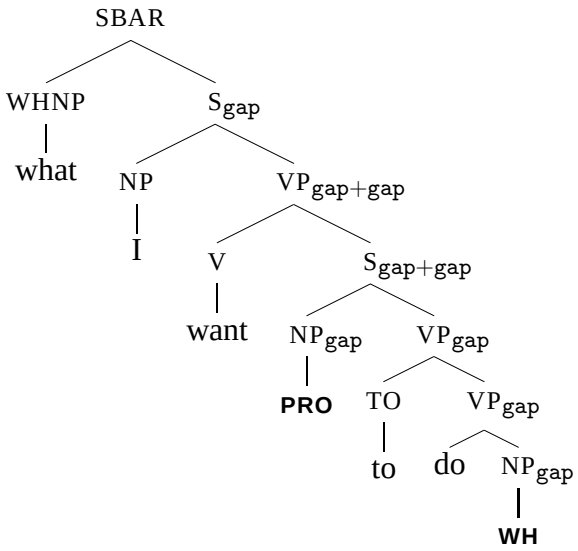


Figure 2.5: Connecting the antecedent and the EE: one reason for introducing typed gap variables.

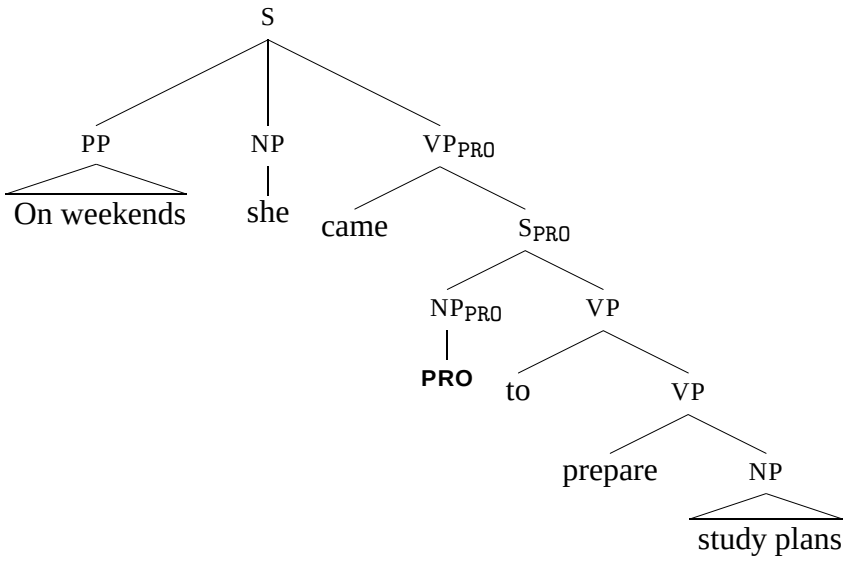


Figure 2.6: Connecting the antecedent and the EE: the reason for adding the label dominating the EE.

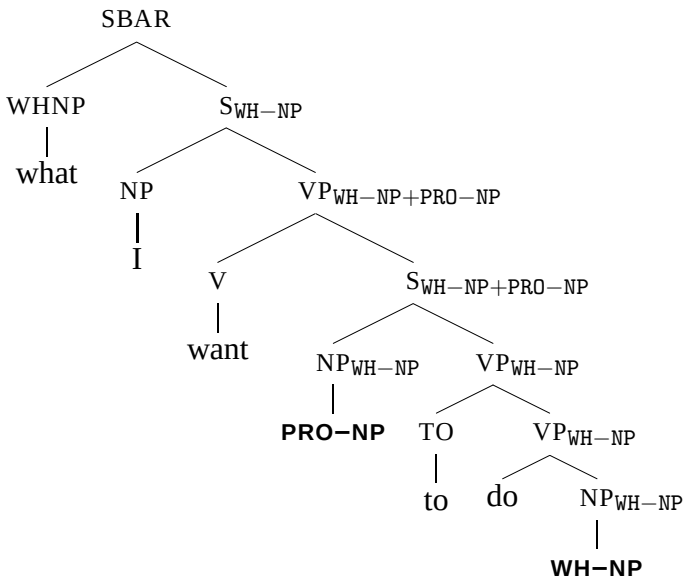


Figure 2.7: Connecting the antecedent and the EE: the use of `gap+` features.

this type with their antecedents. This essentially means that the systems presented here are incapable of handling this type of non-local dependency. Fortunately, such NLDs only constitute less than 4% of all EEs, and 6.5% of the EEs requiring an antecedent.

Apart from sentences with pseudo-attachment, there are 82 sentences in the whole Treebank where the technique presented here fails. Figure 2.8 shows such a sentence; the antecedent *"the Fed"*, co-indexed with the empty element `*-1` in the subject position of the clause *"to bring down rates"*, does not strictly c-command the EE.

Another problem occurs when, at the node immediately dominating the antecedent, there is more than one constituent with the same nonterminal label (e.g. the two NPs in Figure 2.9). Here, we lose the information which of the NPs is the antecedent: the one dominating *she* or the one headed by *yesterday*? In order to cope with this problem, we can use the heuristics of selecting, for instance, the rightmost NP as the antecedent (the algorithm employed for recovering the antecedents is discussed in Section 4.2).

Finally, the scheme presented here cannot handle the cases properly where multiple non-local dependencies of the same type occur, such as in the example:

> *This is a problem which*$_1$ *John*$_2$ *is difficult to talk to* __$_2$ *about* __$_1$. (2.9)
> (Pollard and Sag 1994, p. 159)

Here, the gap variables corresponding to the two non-local dependencies might get mixed up, resulting in the incorrect interpretation where the dependent of *to* is

```
(S (S (NP-SBJ (NNP Bush)
                  (NN administration)
                  (NNS officials))
          (VP (VBP are)
              (VP (VBG looking)
                  (PP-CLR (TO to)
                          (NP-1 (DT the)
                                (NNP Fed)))
                  (S-CLR (NP-SBJ (-NONE- *-1))
                         (VP (TO to)
                             (VP (VB bring)
                                 (PRT (RP down))
                                 (NP (NNS rates))))))))
       (, ,)
       ... )
```

Bush administration officials are looking to the Fed to bring down rates, and financial markets seem to be expecting easier credit as well. (wsj_0072.mrg)

Figure 2.8: A problematic case for our gap-threading approach: the antecedent does not strictly c-command the EE.

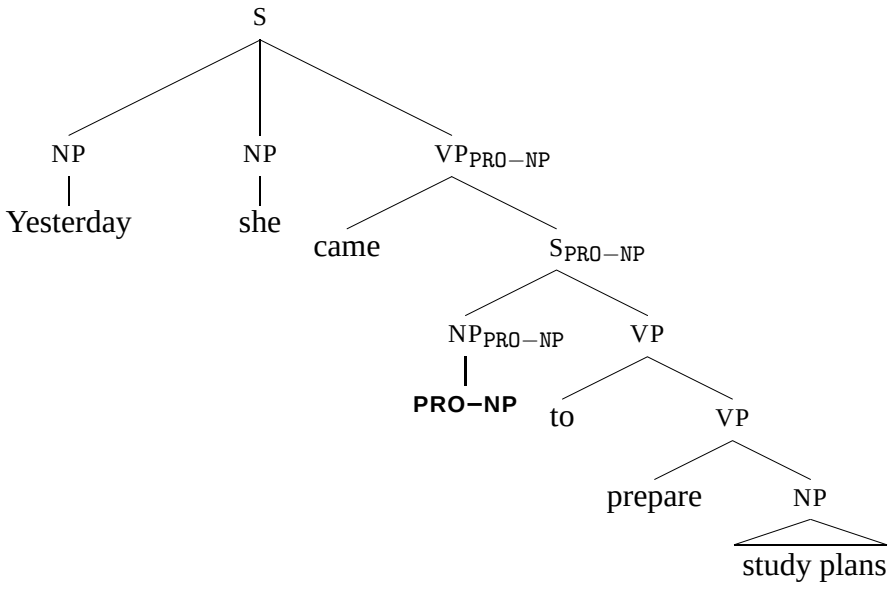


Figure 2.9: A problematic case for our gap-threading approach: there is more than one possible antecedent.

which and the dependent of *about* is *John.* We discuss one attempt to solve this problem in Section 4.2. It is important to note, however, that the problem is more theoretical than practical in nature, since such constructions do not occur in the development section at all; we conjecture that they are indeed infrequent in the PTB.

In summary, in order to be able to handle non-local dependencies with a probabilistic context-free grammar, we approximate co-indexation by compiling the information about the non-local dependency into the nonterminals lying on the path from the EE to its antecedent. This technique is a generalisation of Collins's (1997) approach: we are able to handle multiple types and instances of EEs. We refer to this gap-threading scheme as the one with `gap+` variables. In order to train and test our context-free models, we modified the Treebank as described above. This version of the Treebank is called the *Tracebank*. The labels for the EEs used in this version are summarised in the next section.

This dissertation presents experiments employing the head-lexicalised PCFG models of Collins (1997, 1999) and a generalisation thereof (Chapters 6–8). These models require further annotations. First, for each phrase, the headword of the phrase is detected (cf. Figure 2.10), using the head-finding rules from (Collins 1999) originally proposed by Magerman (1995). Second, the models require the distinction between complements and adjuncts. They are determined using the complement rules presented in (Collins 1999, p. 174). Finally, Collins (1999) extends the probability model to better capture non-recursive noun phrases (baseNPs). This extension requires additional modifications to the Tracebank. Specifically, the nonterminal labels for non-recursive NPs is changed to NPB, and, for consistency reasons, "whenever an NP is seen with no pre or post modifiers, an NPB level is added" (Collins 1999, p. 179). Thus, the final representation of the clause "what I wanted to do" is given in Figure 2.10. Note, however, that these modifications are applied only in tandem with the lexicalised models of Chapters 6–8; the unlexicalised models of Chapters 4 and 6 use trees as presented in Figure 2.7.

2.4 A brief inventory of EEs

In this section, we briefly describe the different types of EEs we use in the Tracebank and the kind of linguistic phenomena they intend to capture. For an exhaustive description with examples, see Appendix A.

The PTB annotation scheme distinguishes six different types of empty elements (Marcus *et al.* 1994, Bies *et al.* 1995, p. 59):

- traces of A′-movement, including parasitic gaps (`*T*`)

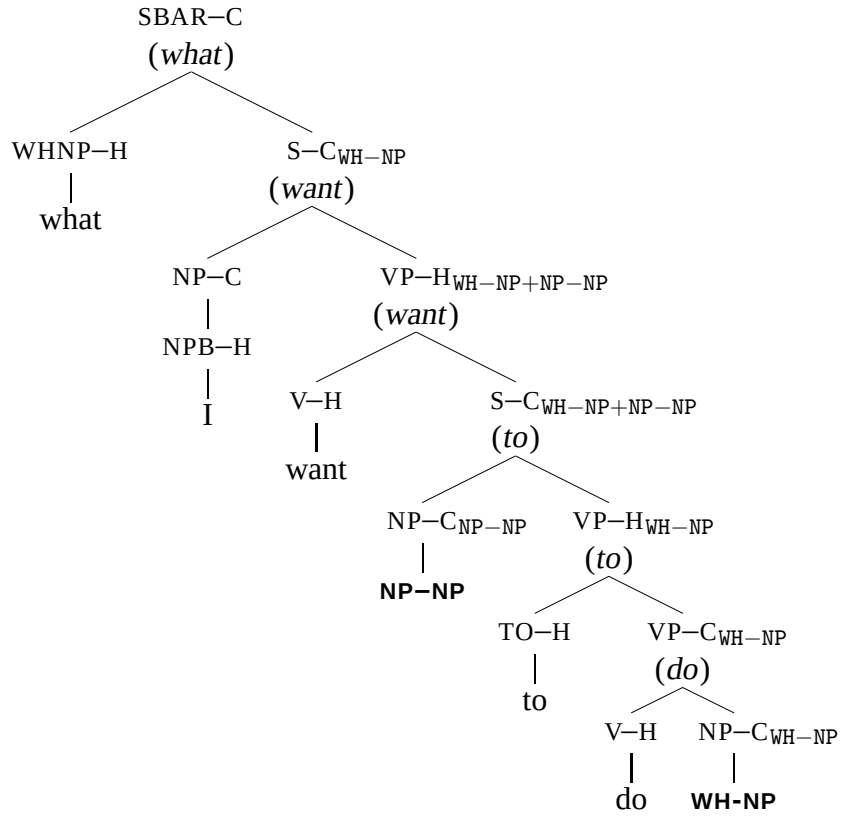


Figure 2.10: A head-lexicalised tree in the Tracebank using the head and complement rules of Collins (1999).

- traces of A-movement, controlled and arbitrary **PRO** (*)
- pseudo-attachments (*RNR*, *ICH*, *PPA* and *EXP*)
- placeholders for ellipsis (*?*)
- empty complementisers (0)
- empty units (*U*)

Arguably, some of these empty elements, namely empty complementisers and empty units, do not represent non-local dependencies, yet we do include them here. The main reason for this decision is to remain compatible with Johnson (2002). Moreover, empty WH-complementisers might serve as antecedents for WH-traces.

As we argued in the previous section, antecedent recovery and CFG-parsing require the encoding of not only the basic type of an EE but also of the nonterminal label dominating it. Therefore, we substituted all occurrences of EEs in the Treebank with the concatenation of their type and the nonterminal label.[7] We also replaced the different symbols for the different EEs with a more mnemonic name as described in Table 2.2. For instance, since *T* mostly stands for WH-construction, a *T* node dominated by an NP label is changed to **WH–NP**. In the remainder of this section, we briefly summarise the syntactic constructions where the different EE-types are used. For a more complete description and a handful of examples, we refer the interested reader to Appendix A. The references in the headers here correspond to the appropriate sections there.

WH–... (A.1.1 and A.1.3)

These empty elements, originally represented in the PTB with *T*, stand for traces of A′ movement; they always have an antecedent. The most frequent EEs of this type are **WH–NP** and **WH–ADVP**. They are used for handling:

- WH-questions (A.1.1, p. 179)
- relative clauses (A.1.1, p. 179)
- fronting, topicalisation (A.1.1, p. 180)
- *tough*-constructions (A.1.1, p. 180),

[7] In order to keep our results with Johnson's (2002), the nonterminal label of empty units (*U*) is not encoded. This is no problem for antecedent recovery, since empty units never have antecedents.

Notation		Antecedent	WSJ00	
Here	PTB		Freq.	Rel. freq.
WH−...	*T*	+	582	17.6%
TOP−S	*T*[8]	+	233	7.0%
NP−NP	*	+	987	29.8%
PRO−NP	*	−	426	12.9%
PSEUDO−...	*PPA* *RNR* *ICH* *EXP*	+	127	3.8%
ELLIPSIS−...	*?*	−	37	1.0%
COMP−...	0	−	584	17.6%
UNIT	*TU*	−	334	10.1%
All			3311	100%

Table 2.2: Frequencies and relative frequencies of EEs in WSJ00.

- parasitic gaps (A.1.1, p. 181), and
- *so*-constructions (A.1.3, p. 183).

A typical example of usage is:

```
(NP (NP answers)                                                    (2.10)
    (SBAR (WHNP that/which)
          (S_WH−NP (NP we)
                   (VP_WH−NP encountered
                             (NP_WH−NP WH−NP)))))
```

TOP−S (A.1.2)

Although the PTB annotation scheme uses the same symbol (*T*) when topicalisation of sentences occur, for the sake of clarity, we represent with **TOP−S** every *T* which is immediately dominated by an S nonterminal. These EEs also have an antecedent and are mainly used in quotations (A.1.2, p. 182):

[8] We use **TOP−S** to represent *T* traces if and only if the EE is immediately dominated by an S node; otherwise the *T* is changed to **WH−...**.

```
(S ``                                                    (2.11)
   (S (NP Willie)
          (VP caught
              (NP the ball)))
   ''
   (NP-SBJ Casey)
   (VP_TOP−S said
          (S_TOP−S TOP−S))
   .)
```

NP–NP and PRO–NP (A.2)

These EE-types are the most frequent ones in the Treebank. Originally, both of them are represented with `*`. Some of them have an antecedent, however, while others do not. Therefore, for the ease of reference, we use the label **NP–NP** for the occurrences with antecedents, and **PRO–NP** for the ones without co-indexation. They vaguely correspond to controlled and arbitrary **PRO**s, respectively. Therefore, we will often use **PRO** to refer to both of them. Their usage covers:

- passives (A.2.1, p. 184),
- reduced relatives (A.2.2, p. 185),
- empty subjects of infinitival clauses (A.2.3, p. 186),
- empty subjects of participial clauses and gerunds (A.2.4, p. 189), and
- empty subjects of *as*- and *than*-clauses (A.2.5, p. 192).

In the example (2.12b), observe the lack of threading gaps: **PRO–NP**s cannot have antecedents, i.e., they are never co-indexed.

(a)
```
(S (NP Everyone)                                         (2.12)
   (VP_NP−NP seems
          (S_NP−NP (NP_NP−NP NP−NP)
                 (VP to
                     (VP dislike
                         (NP Drew Barrymore))))))
```

(b)
```
(NP (NP John 's)
    decision
    (S (NP PRO−NP)
       (VP to
           (VP leave))))
```

PSEUDO–... (A.3)

This label stands for pseudo-attach constructions. In the Tracebank, we merge the four different types; in the list below, the original markers are given in brackets. These EEs always have an antecedent, though it often does not strictly c-command the EE.

- permanent predictable ambiguity, such as illustrated by the sentence *"I saw a man with a telescope."* (PTB:*PPA*, A.3.1, p. 194),
- shared constituents, right node raising (PTB:*RNR*, A.3.2, p. 194),
- discontinuous structures (PTB:*ICH*, A.3.3, p. 195), and
- *it*-extrapositon (PTB:*EXP*, A.3.4, p. 196).

The example in (2.13) shows the most frequent usage (discontinuous structures). Observe the lack of gap+ variables connecting the **PSEUDO–SBAR** element and its antecedent *"whom she at once recognized as Jemina Broadwood"*.

```
(S (NP (NP a young woman)                                    (2.13)
       (SBAR PSEUDO−SBAR))
   (VP entered
       (SBAR (WHNP whom)
             (S_WH−NP (NP she)
                      (PP at
                          (ADVP once))
                      (VP_WH−NP recognized
                               (NP_WH−NP WH−NP)
                               (PP as
                                   (NP Jemima Broadwood)
                                ))))))
```

ELLIPSIS–... (A.4)

This empty element acts as a placeholder for an ellipsed predicate or a part thereof. Even if the missing material is identical with another constituent in the sentence, these EEs never have antecedents (cf. the example below). The annotators mainly use this label as a last resort, although there are two constructions where they regularly occur:

- comparative deletion (A.4, p. 196) and
- representing a missing VP after auxiliaries (A.4, p. 197).

```
(S (NP John)                                                    (2.14)
   (VP is
       (ADJP (ADJP sillier)
          (SBAR than
             (S (NP I)
                (VP am
                    (ADJP ELLIPSIS−ADJP)))))))
```

COMP−... (A.5)

These labels are inserted in **SBAR** constituents with a missing overt complementiser. There are three such labels: **COMP−SBAR**, **COMP−WHNP**, **COMP−WHADVP**. The latter ones represent null WH-operators in relative clauses (A.5.2, p. 200). None of the three labels have an antecedent, though **COMP−WH...** traces can serve as antecedents for **WH−...** gaps:

```
(NP (NP the bird)                                               (2.15)
    (SBAR (WHNP COMP−WHNP)
          (S_WH−NP (NP I)
                   (VP_WH−NP saw
                             (NP_WH−NP WH−NP)))))
```

UNIT (A.6)

This label is used in conjunction with currency symbols, such as $ or FFr (French francs), and marks the position in the string where they would appear if the phrase were pronounced. The relevance of these empty elements is rather questionable, and they in no way represent non-local dependencies. The only reason for including them here is compatibility – we want to keep our results comparable with those of Johnson (2002). For the same reason, we do not differentiate between **UNITS** under NP and ADVP nodes. A typical example of usage is:

```
(NP $ 5 UNIT)                                                   (2.16)
```

Note

In this dissertation, we define every construction involving empty elements in the PTB as representing a non-local dependency. Nevertheless, we are aware of the deficiencies of this decision: some of the empty elements represent some idiosyncrasies in the annotation scheme and do not contribute to the meaning of the sentence (e.g. empty **UNITS**). Therefore, apart from scores involving all EEs, we

will also report results separately on a class of empty elements that we regard semantically important, namely on the class containing **WH–...** and **TOP–S** gaps as well as controlled and uncontrolled **PRO**s (**NP–NP** and **PRO–NP**, respectively). In the original annotation scheme, these empty elements are easy to recognise, since they are represented as *T* and * traces.

Chapter 3

Previous work on applying machine-learning techniques to NLD-recovery

In this chapter, we review previous approaches to recovering non-local dependencies using machine-learning techniques. Previous work falls into three different groups based on the methodology they employ to handle NLDs. Most studies propose carrying out this task while parsing. Others attempt to recover NLDs after parsing with a context-free parser not trained to deal with non-locality. The third approach proposed does not rely on parsing at all, it operates with chunks instead.

The present work explores a fourth possibility: we detect NLDs prior to parsing with a finite-state preprocessor, and use the parser to incorporate this information into the parse tree. We also experiment with parsers that are designed to do both EE detection and antecedent recovery during parsing (Chapters 4 and 7).

3.1 Statistical parsing with non-local dependencies

The problem of parsing with NLDs (discontinuities) is well studied (e.g. Proudian and Pollard 1985, Johnson 1985, Vijay-Shanker 1987, Bunt *et al.* 1987, Covington 1990a,b, 1994, Reape 1991, van Noord 1991, Bunt 1991, 1996, Müller 1996). In fact, most modern formal grammar formalisms come with implemented parsers. The problem, however, is far from being solved. Although we can parse and recover non-local dependencies, there are a couple of serious issues to address.

The first issue concerns efficiency. Exhaustive parsing does not appear to be feasible even with PCFG parsers having only cubic worst-case time complexity in the length of the sentence. The recognition problem with the unlexicalised version of Tree-Adjoining Grammar or Combinatorial Categorial Grammar re-

quires $\mathcal{O}(n^6)$ time in the worst case, where n is the length of the sentence (Vijay-Shanker and Weir 1994). In the case of Lexical Functional Grammar or Head-Driven Phrase Structure Grammar, ensuring well-formedness restrictions on feature structures might further increase this complexity to exponential (Maxwell and Kaplan 1995a). These results emphasise the need for good search strategies to make parsing feasible with these grammar formalisms.

Second, the task of natural language parsing is not just deciding whether the sentence is grammatical or not according to the grammar, but also producing the *most appropriate analysis* for the input. Since the number of analyses might be exponential in the number of words, as is the case, for instance, for PP-attachment ambiguity in English (Church and Patil 1982), the approach of enumerating all possible parses and letting another module select the right one is infeasible (Osborne 2000a, Johnson 2003): the parser has to perform this task while parsing.

Third, manually developed grammars tend to suffer from decreased coverage; Baldwin *et al.* (2004) report 18% coverage for a large-scale HPSG grammar (English Resource Grammar Copestake and Flickinger 2000) on the British National Corpus; for LFG, using the large-scale LFG grammar (Butt *et al.* 1999), Riezler *et al.* (2002) report the coverage to be 75% on Section 23 of the Wall Street Journal. Since natural language shows a Zipfian behaviour, there is no hope for a lexicalised grammar to achieve 100% coverage: in any large test corpus, there will be unknown words. Therefore, it is necessary to introduce mechanisms that handle this problem, which, in turn, further increase the ambiguity inherent in the grammar.

Furthermore, a parser might be employed under circumstances where its input is not grammatical or even ambiguous at the string level. For instance, parsing sentences that come from an OCR system or a speech recogniser might involve dealing with multiple, often corrupted input, occasionally associated with the preprocessor's probability estimation of the strings given the utterance or picture (confidence score). A parser has to be able to handle these inputs robustly and should attempt to incorporate the confidence score.

Finally, manual development of large-scale grammars is time-consuming and error-prone. Therefore, it is important to investigate automatic grammar induction, using either supervised or unsupervised learning. Automatically extracted grammars, on the other hand, tend to extremely overgenerate, which increases ambiguity even further. In fact, they usually produce many unacceptable analyses which must be discarded as early as possible in the parsing process.

There might be several ways and heuristics to tackle all or some of these problems. We believe, however, that there is a universal, well-founded approach: using probability theory to rank possible (partial) parses. In terms of probabilities, the notion of the most appropriate parse can be formally defined: the most appropriate parse is the one with the highest probability given the input (and possibly other

information).[1] The probability distribution over possible (partial) syntactic structures for the input can efficiently and accurately guide the parser in exploring the search space of possible analyses. Probability theory allows a sound combination of different modules, such as a preprocessor and the parser.

In what follows in this section, we review statistical parsing approaches to handling NLDs. Most of the approaches assume an underlying grammar formalism, ranging from context-free grammars to LFG, HPSG, TAG, CCG or to dependency grammar. We organise the related work according to the formalisms they are based on.

3.1.1 Probabilistic Context-Free Grammars

Although there is a large amount of literature on stochastic parsing of English with CFGs using various parametrisation schemes (Sampson 1986, Haigh *et al.* 1988, Chitaro and Grishman 1990, Magerman and Marcus 1991, Magerman and Weir 1992, Black *et al.* 1992a,b, Pereira and Schabes 1992, Schabes *et al.* 1993, Jelinek *et al.* 1994, Magerman 1995, Charniak 1993, 1996, 1997, 2000, Collins 1997, 1999, 2000, Collins and Duffy 2002, Ratnaparkhi 1997, Johnson 1998, Henderson 2003, Klein and Manning 2003, just to mention a few examples), none of these studies try to model NLDs, with the exception of Model 3 of Collins (1997, 1999). In fact, most of these models ignore empty elements representing NLDs by deleting them from both the training and the test data (Charniak 1996).

Model 3 of Collins (1997, 1999) captures a subset of NLDs, namely (a subset of) the cases of extracted NPs. The probability model explicitly handles these gaps: Collins employs the technique of enriching nonterminals with `gap` variables (cf. Section 2.3), and trains his grammar with the extended set of nonterminals. PARSEVAL scores for Model 3 are marginally higher than for Model 2, showing that handling EEs also improves the parser with respect to phrase structure. Collins (1997, 1999) reports 93.8%/90.1% precision/recall for detecting the correct sites of these EEs, but he excludes infinitival relatives from this evaluation (where he can only find the traces with 41%/18% precision/recall). According to the metrics proposed by Johnson (2002) (cf. Section 4.1), Model 3 achieves 85.7% overall F-score for detecting **WH–NP**s and 84.1% for recovering their antecedents.

The most important deficiency of Model 3 is that it only handles **WH–NP**s.[2] In particular, it cannot model multiple types and multiple instances of EEs (the lat-

[1] We interpret probabilistic parsing in a very broad sense here, where the probability of a parse might depend on many types of both syntactic and extra-syntactic information ranging from intonation to dialogue structure to pragmatics.

[2] In fact, it only handles a subset of them: it ignores reduced relative constructions, in which around 12% of **WH–NP**s occur. As it turns out, these cases are difficult for other approaches as well.

ter occurring in the presence of coordination or parasitic gaps). Furthermore, it assumes all EEs to be complements, hence it cannot treat extraction involving adverbs (e.g. **WH–ADVP**). In Chapter 7 we discuss a possible extension of the model to cover a wider range of NLDs; we postpone the detailed discussion of Model 3 to Section 7.1.

3.1.2 Probabilistic Discontinuous Phrase Structure Grammar

The most important limitation of using CFGs in handling non-local dependencies is the continuity assumption present in the grammar formalism: the yield of each non-terminal is taken to be a contiguous substring. There are several extensions to CFG that relax this assumption while attempting to retain the simplicity of context-free rules over a relatively small set of non-terminals (Blevins 1990, Bunt *et al.* 1987, Bunt 1991, 1996). Plaehn (1999, 2000) presents such a model, Probabilistic Discontinuous Phrase Structure Grammar, which is a parametrised (and axiomatised) version of Discontinuous Phrase Structure Grammar (DPSG, Bunt 1996).

Informally, rules in DPSG are very similar to context-free rules. Consider, for instance, a rule for relative clause adjunction: NP $\rightarrow$ NP RC expresses that the NP constituent is rewritten as the non-terminal sequence NP RC. In CFG, the yield of the outer NP is the concatenation of the yields of the inner NP and the RC, which are presumed to be contiguous, and every terminal dominated by the inner NP precedes every terminal dominated by RC. In DPSG, the same rule imposes weaker constraints: (i) only the leftmost child of the inner NP (and therefore the leftmost terminal dominated by it) has to precede the leftmost child of the RC; (ii) it is possible that the yield of the outer NP is not contiguous, that is, some terminals not dominated by the outer NP can intervene. In natural language, however, the set of intervening material is rather restricted: therefore, DPSG requires the explicit specification of this set. For instance, if we want to model right dissociation, such as in the sentence

I saw the man yesterday $[_{\text{RC}}$ *who gave us an apple a week ago.*$]$ (3.1)

we have to write NP $\rightarrow$ NP [ADVP] RC, which allows the relative clause being separated from the NP by an intervening adverbial.

Plaehn (1999, 2000) presents a possible parametrisation of the grammar and also develops an agenda-based bottom-up chart parser returning the most probable parse. He also describes supervised learning experiments with an automatically extracted unlexicalised grammar on the German Negra corpus (Krenn *et al.* 1998). The results he reports are slightly discouraging: the parser is rather slow and parsing time appears to be exponential in sentence length. When compared to a PCFG parser (trained on the same data minus empty elements), the PDPSG parser is 50

times slower on sentences no longer than 15 tokens. Moreover, the PCFG parser is 1.5% more accurate according to the PARSEVAL metrics, although it is not able to handle discontinuity. No evaluation is given with respect to discontinuities.

3.1.3 Tree-Adjoining and Tree-Substitution Grammars

Another way of extending context-free grammars to cover non-locality is to allow the grammar to manipulate not only local trees (represented by context-free rules), but also larger fragments. This is the key idea behind Tree-Adjoining Grammars (TAG, Joshi *et al.* 1975, Joshi and Schabes 1997) and Data-Oriented Parsing (DOP, Scha 1990, Bod 1992, Scha *et al.* 1999). We discuss the two models in the same section, because, apart from sharing the main idea, the grammar formalism underlying the DOP model, Tree-Substitution Grammar (TSG), is a restricted form of TAG. Therefore, many of the theoretical results in one framework translate to the other. For example, Carroll and Weir (1997, 2003) show how closely related the probability models for TAG and TSG are. Also, Sima'an (1996, 2003) shows that finding the maximum probability (minimum weight) parse tree for any weighted versions of TSG and hence of TAG is $\mathcal{NP}$-hard.

TAG

The initial probability model for TAG was developed by Resnik (1992) and Schabes (1992). The model parametrises TAG derivations, which define context-free tree structures (Schabes and Shieber 1994). There are many ways to assign parameters to such a derivation tree; essentially, both Resnik (1992) and Schabes (1992) estimate the probability of a local derivation tree by the product of the probabilities of the daughter nodes (i.e., elementary and auxiliary trees) given the parent node.[3] Informally, the generation of the subtrees adjoined and substituted in an elementary tree is a unigram Markov process conditioned on the elementary tree. As a consequence, this model assumes independence between the generation of subjects and objects of a given verb.

Carroll and Weir (1997, 2003) discuss further extensions to the probability model, such as generating the derivation trees with a PCFG, which would capture dependencies between subjects and objects, or between adjuncts. Carroll and Weir do not test their models empirically, but Chiang (2000, 2003), for instance, incorporates their ideas into his TAG parser by assuming a bigram Markov model generating auxiliary trees being adjoined at the same site.

[3] The models are a bit more complicated, since they define two separate probability distributions for adjunction and substitution given the parent node. Furthermore, because adjunction is optional, the model also has to assign probability to cases where no adjunction takes place.

Hwa (1998) reports on large-scale grammar induction and parsing experiments with stochastic Lexicalised Tree-Insertion Grammars (TIG, Schabes and Waters 1995, 1996), a restricted version of TAG, which allows limited amount of adjunction. The TIG is automatically induced from a treebank making use of bracketing and is parametrised using the EM algorithm. Hwa (1998) employs a version of TIG where nonterminal nodes are not labelled with their categories, that is, she models unlabelled bilexical dependencies. Moreover, the grammar is only lexicalised to POS-tags. The main results of the paper are that the automatically induced TIG is a better language model than the automatically induced Chomsky-normal-form PCFG, and that grammar induction converges much faster for TIGs.

Chiang (2000, 2003) presents experiments with a properly lexicalised stochastic TIG extracted from the PTB and parametrised using supervised learning. First, he induces a lexicalised TIG grammar employing the same heuristics for finding heads and complements as Collins (1997, 1999). As a second step, he reconstructs the derivations in the training data using the extracted grammar. Model parameters are obtained by maximum likelihood estimation and are subsequently smoothed. The goal of the parser is set to finding the most probable derivation (as opposed to the most probable parse, i.e., the sum of all derivations leading to the given parse). The resulting model is compared to lexicalised context-free parsers: its accuracy measured on the PARSEVAL metrics lies between the models of Collins (1996) and Collins (1997). Although TIG is able to capture some non-local dependencies (e.g. control constructions or short-distance WH-extraction), no results are given concerning how good the parser is at finding them.

Another approach to parsing with TAG is supertagging (Joshi and Srinivas 1994, Srinivas and Joshi 1999), which is a two-step process: first a stochastic tagger, the supertagger, associates a sequence of trees (either elementary or auxiliary ones) with each word in the input, then a linear-time, heuristic-based, non-probabilistic parser assembles these trees into a possible parse (or partial parses, if it cannot analyse the full sentence). In order to reduce the effect of tagging errors, the supertagger is run in *n*-best mode. This approach requires a large-scale grammar; Srinivas and Joshi (1999) experiment with a wide-coverage hand-written TAG grammar (the XTAG grammar, Doran *et al.* 1994, XTAG Research Group 2001) as well as with a grammar automatically extracted from the PTB. Using the XTAG grammar, Srinivas (1997) reports 93.8%/82.3% unlabelled precision/recall for retrieving dependencies in WSJ20. It is not clear whether these dependencies include non-local ones or not. He also evaluates the supertagger on a number of tasks; the ones including NLDs are parentheticals and relative clauses, where the supertagger finds the sites with 80.0% and 55.1% F-score, respectively.

DOP

Although DOP is a restricted form of TAG, they differ in two important respects: (i) they have different views on the grammar, and (ii) they employ different probability models. As for the grammar, the DOP model does not assume a grammar *a priori*, since "human language production and perception works with past language experiences, rather than with abstract grammar rules" (Bod and Kaplan 1998). This means that, ultimately, the best way to approach the task of parsing a new utterance is to remember all previously heard utterances with their corresponding analyses and retrieve the analysis or analyses which match the new utterance. This extreme approach, however, fails because of the inherent sparseness of utterances: since natural languages show a Zipfian behaviour, most of the new utterances are genuinely new to the hearer, i.e., they do not occur in the previously observed data.

In order to deal with data sparseness, the DOP approach *decomposes* the well-formed analyses of previous utterances into smaller fragments. The novelty of the DOP model with respect to other approaches is that this decomposition is not restricted by a grammar framework: the fragments can be, for instance, subtrees of arbitrary depth. Formally, DOP models define several *decomposition operators* to cut well-formed trees into smaller elementary trees. These fragments, then, are stored with their probabilities (there are different variations on how these probabilities are estimated; see Bonnema 2003). One problem is the number of possible elementary trees, which is exponential in the number of words in the sentence. Goodman (1996, 2003) shows how to decrease the cardinality of the set of fragments to linear in the number of words using PCFG-reductions, so that the resulting grammar induces the same probability distribution over parse trees.

When it comes to parsing, the parser selects several fragments and reassembles them into a well-formed analysis of the new utterance. In order to achieve this goal, a set of *composition* operators are defined; in the most popular version, the only composition operator is substitution. Then, the best analysis is chosen according to the probability model.

One way to parametrise this approach is to define a probability model over possible derivations of analyses. Formally, the probability of an analysis R is the sum of all possible derivations ($d \in \mathcal{D}(R)$) leading to R:

$$P(R) = \sum_{d \in \mathcal{D}(R)} P(d) \tag{3.2}$$

The probability of a derivation is the product of the probabilities of the tree fragments involved in the derivation. Note that several different derivations might give rise to one and the same analysis. Therefore, it does not suffice to find the most probable derivation: the parsing algorithm has to determine *all* possible deriva-

tions in order to calculate the probability of an analysis. This set might be exponentially large in practice and thus this calculation might be very costly. In fact, as Sima'an (1996, 2003) shows, finding the best analysis is an $\mathcal{NP}$-hard problem.

There are several ways to go about this problem. The classic one, suggested by Bod (1992), *samples* from the distribution of all possible derivations to estimate $P(R)$, instead of summing over all derivations. The samples can be derived by Monte Carlo sampling, which turns out to be prohibitive for parsing the PTB, or using the *n*-best Viterbi-derivations as a sample, with *n* being sufficiently large (Bod 2000).

Another approach is to modify the criterion for the best parse: instead of finding the most probable parse, one might search for the most probable derivation, or even the derivation containing the smallest number of trees. Both approaches alleviate the complexity of the parsing problem. As Bod (1995a,b) shows, however, the "maximum probability derivation" criterion performs significantly worse than the "maximum probability parse" one. On the other hand, when using the "simplicity" criterion, Bod (2000) reports results comparable with the accuracy under the "maximum probability parse" criterion. Bod (2003) combines the two criteria and reports around 1% improvement over the model presented in (Bod 2000).

Although the DOP model is perfectly capable of capturing NLDs (by simply keeping the EEs present in the training trees), no evaluation is reported concerning its performance with respect to them.

3.1.4 Unification-based grammars: LFG and HPSG

Currently, there are three widely accepted models for statistical parsing with unification-based grammar (UBG) formalisms such as LFG and HPSG. The first approach approximates a hand-written UBG with a CFG, and parametrises this grammar (Briscoe and Carroll 1993, Eisele 1994, Brew 1995, Kiefer *et al.* 2002). Although this technique is successful in practical applications, it is not without its problems. As Abney (1997) notes, this model makes unwarranted independence assumptions, that is, it fails to model context-sensitivity, hence it is not a sound probability model for UBGs. To include the necessary amount of context-sensitivity in the model, Abney (1997) proposes a parametrisation technique for UBGs based on random fields (or log-linear) models. This second approach is further elaborated by Mark Johnson, Stefan Riezler and their colleagues (Johnson *et al.* 1999, Riezler *et al.* 2000, 2002, Johnson 2003). Both PCFG approximation and log-linear models require a wide-coverage hand-written grammar and rank possible analyses according to the probability model. The third approach, first introduced by Bod and Kaplan (1998), develops a DOP model for LFG. This model learns both the grammar and its parametrisation from an annotated corpus. Below, we briefly discuss the three models.

Context-free approximation

The main idea behind the context-free approximation of UBGs is to find a finite context-free grammar which generates a superset of possible analyses. This CFG is called the context-free backbone of the UBG. There are several ways of constructing this context-free grammar (cf. Briscoe and Carroll 1993, Eisele 1994, Brew 1995), the most general approach may be the one presented by Kiefer and Krieger (2000, 2002).

Once we have the context-free backbone for a given hand-written UBG, the only thing to be done is parametrising the resulting CFG, i.e., inducing a probability model. One straightforward solution, employed by Eisele (1994), Brew (1995), Kiefer *et al.* (2002), is to use the standard PCFG model (Booth and Thompson 1973) and obtain the parameters via the EM algorithm (Dempster *et al.* 1977). Briscoe and Carroll (1993) apply a supervised training technique and define their probability model on the actions of their LR parser (this model bears strong resemblance to the model proposed by Ratnaparkhi 1997).

Since UBG formalisms are more powerful than CFGs,[4] this approach can only be an approximation. It is important to note, however, that although it might allow context-free trees that are ruled out by the UBG, all possible UBG-analyses are generated by the CFG. The induced model is *not* a probability distribution over the possible analyses licensed by the UBG, since malformed analyses might get some probability mass, but all well-formed analyses get a probability higher than 0. Consequently, this approach can be used to *rank* hypotheses according to the probability model associated with the CFG and hence is applicable for parse disambiguation.

This technique is quite successful in practical situations. Note that the construction of the corresponding CFG can be carried out offline, thus performing a large amount of costly (and usually failing) unifications in a preprocessing step. As a consequence, a considerable speedup can be achieved at parsing time, since a relatively cheap context-free parsing algorithm can be used to search the space of possible analyses. Even though they have to deterministically replay unifications after possible context-free backbones are constructed to filter out impossible readings and get the well-formed analyses according to the UBG, Kiefer and Krieger (2002) still report a 71.1% parse-time speedup for a realistic mid-sized grammar.

The second improvement context-free approximation brings about is a better disambiguation capacity. Large-scale grammars tend to be very ambiguous: Kiefer *et al.* (2002) report an ambiguity rate of 1.44 readings per sentence on average for their mid-sized grammar, whereas for a larger-scale grammar used by Baldwin *et al.* (2004), it rises to 64 analyses per sentence of length $10-20$

[4] It is easy to write a grammar which generates the language $\mathcal{L} = \{ww | w \in \{a,b\}^*\}$, known to be context-sensitive

words. Using the PCFG probabilities to rank possible analyses proves to be valuable: Kiefer *et al.* (2002) achieve a 16% better score for disambiguation (measured by the precision of ranking the correct analysis first) when they use the probabilities returned by the PCFG parser than when they treat all analyses exactly likely and choose from them randomly.

There are two important problems with the approach. First, the resulting CFG is enormous. When converting the large-scale LinGO English grammar (Oepen *et al.* 2002a,b,c) to a CFG, Kiefer *et al.* (2002) arrive at a grammar with 5537 nonterminals and 1.5M rule productions, although they only cover less than one eighth of the lexical items present in the grammar. Reliable estimation of probabilities for a PCFG of such a size calls for a large amount of training data, which turns out to be a major bottleneck for the approach. Furthermore, the EM algorithm used in training is very slow: each iteration requires re-parsing the training corpus, which takes $\mathcal{O}(sN^3)$ time in the worst case, where s is the number of sentences and N is the number of non-terminals in the grammar.

The second problem is that context-free approximation is only an approximation. That is, the true probability for a given analysis might lie far from the probability assigned to its context-free backbone (Eisele 1994). As Abney (1997) points out, the problem with the PCFG-approximation technique is that it introduces unwarranted independence assumptions; he, therefore, proposes a random fields (log-linear) model which does not hinge on independence assumptions and can model context-sensitivity present in UBG formalisms (see next Section). Goodman (1997) and Schmid (2002) propose generative models for grammars with feature structures, though the applicability of these approaches for modelling LFG- or HPSG-style grammars is problematic. Goodman (1997) associates probabilities with fully specified feature structures (usually becoming available only at the end of a UBG-derivation process). Schmid (2002), on the other hand, augments rules with probabilities, which "poses problems if the grammar is not rule-based (like HPSG) or if the result of rule application is not uniquely defined (as with functional uncertainty in LFG)."

The log-linear model

Abney (1997) proposes a parametrisation of UBGs based on random fields models which does not posit unwarranted independence assumptions. In this model, (well-formed) analyses are represented with the aid of indicator features (cf. Section 5.1.1): the model assigns weights to each feature; the probability of an analysis is proportional to (the log of) the sum of the weights of the active features for the input (with multiplicity). Relevant features are induced automatically by adding (the combination of) atomic features to the feature space using a greedy algorithm (Della Pietra *et al.* 1997). Atomic features can be freely de-

fined – they are the means for capturing linguistic knowledge in the model. The weights are determined by an iterative scaling algorithm (Darroch and Ratcliff 1972, Della Pietra *et al.* 1997) so that the induced probability distribution maximises the log-likelihood of the training data with respect to the empirical distribution observed in the training set.

One drawback of Abney's (1997) proposal is that it, in principle, requires the enumeration of all possible analyses the UBG licenses – a set that might well be infinite. Abney (1997) proposes Metropolis-Hastings sampling to counter this problem, which, according to Johnson *et al.* (1999), turns out to be intractable for realistic-sized grammars. Therefore, they modify the optimisation task by maximising the pseudo-likelihood of the syntactic analyses over the training corpus, i.e., maximising the conditional likelihood of the analyses given the observed sentences. This optimisation criterion only requires the enumeration of all possible analyses for the sentences in the training data, which is computationally less demanding, but it may still be infeasible (Osborne 2000a). Note that this approach also requires a (hand-written) grammar prior to training, which provides all possible analyses for the training and test data.

Johnson *et al.* (1999), using a handwritten LFG grammar, test their approach on a small test set and report promising results. In experiments with a large-scale HPSG grammar, Toutanova *et al.* (2002) compare a generative PCFG approximation and the random fields (conditional) model, and find that the conditional model outperforms the generative one by 13% on the disambiguation task, even though both of them employ the same features. This difference is less dramatic ($3-5\%$) when they use a richer PCFG representation encoding ancestor information in the non-terminals (cf. Johnson 1998). Interestingly, Baldridge and Osborne (2003) find that a log-linear model based on very simple *n*-gram features derived from flattened tree representations is virtually as accurate as the one using linguistically motivated information.

Although the above approaches are promising, they remain relatively small scale. The HPSG treebank they use contains only 5302 sentences of length 9.3 words on average (Oepen *et al.* 2002a,b,c). Since the underlying grammar is developed in parallel with the treebank, coverage is not a problem either.

The first attempts at large-scale parsing with UBGs using a random fields model are presented in (Riezler *et al.* 2000) and (Riezler *et al.* 2002). They make use of a hand-written LFG grammar (ParGram, Butt *et al.* 1999) to determine all possible well-formed LFG analyses for the input. They employ a log-linear model to disambiguate the analyses that takes both lexical and structural information into account (at the level of c- and f-structures).

The novelty of the two models is that they do not require hand-annotated UBG corpora to derive the parameters of the models. Riezler *et al.* (2000) employ an EM-like algorithm to infer model parameters from *unlabelled* (incomplete) data.

The singly iterative IM algorithm, described in detail in (Riezler 1999), combines the iterative steps of the EM algorithm (Dempster *et al.* 1977) and the iterative scaling algorithms employed for training log-linear models (Darroch and Ratcliff 1972, Della Pietra *et al.* 1997). This approach avoids the problem of chaotic convergence behaviour of the EM estimation.

Riezler *et al.* (2000) claim that their model greatly benefits from training on incomplete data: on a small evaluation set, they report 10% higher disambiguation scores when the training is performed on a larger set of unlabelled (and ambiguous) data than when they apply supervised training of the log-linear model on a smaller set of unambiguous data. Note, however, that the lower accuracy in the latter case might be attributed to the lack of negative examples in the training material. Bouma *et al.* (2001) apply the same technique to their wide-coverage HPSG grammar and find $7-14\%$ improvement with respect to the uniform baseline (depending on their test corpus). They also experiment with some hand-written heuristic penalty rules and also with a bilexical dependency model, both of which outperform the log-linear model by a relatively large margin, although the log-linear model uses the same features.

In a different approach using *partially unsupervised* learning, Riezler *et al.* (2002) propose a discriminative log-linear model to parametrise LFG analyses produced by the hand-written grammar. They present the first successful large-scale parsing project using a unification-based grammar formalism augmented with probabilities to guide disambiguation.

Riezler *et al.* (2002) make extensive use of the existing annotation for the Penn Treebank in a very elegant way. The labelled phrase-structure trees of the Treebank can be viewed as partial (underspecified) LFG analyses and used to guide the discriminative estimation. That is, they define the discriminative criteria with respect to the "*set of parses consistent with the* WSJ *annotations*" (*ibid.*). Intuitively, this means that the training algorithm is to adjust the model parameters so that most of the weight is put on the parses which are consistent with the bracketing present in the training data. This idea is very similar to that of Pereira and Schabes (1992), who use the unlabelled bracketing of the training data to guide the Inside-Outside algorithm.

Manually created grammars tend to suffer from the problem of relatively low coverage. For example, the ParGram grammar (Butt *et al.* 1999) used by Riezler *et al.* (2002) covers only 74.7% of the sentences. In order to raise the coverage of the grammar, the authors rely on two robustness techniques. First, they augment the standard grammar with a FRAGMENT grammar which allows parsing sentences into well-formed chunks, if the original grammar fails to produce an analysis. When the FRAGMENT grammar is activated, the fewest-chunks heuristics is used, i.e., partial analyses consisting of fewer chunks are preferred. The second technique Riezler *et al.* (2002) apply is SKIMMING: whenever the parser

spends too much time and/or memory on the analysis, it goes into skimming mode, which restricts the amount of work on nodes not yet processed. This technique ensures that the parser finishes its job in polynomial amount of time.

Riezler *et al.* (2002) evaluate their parsing model on approximately one third of the standard test set from the Wall Street Journal (WSJ23). They manually annotate 700 sentences from WSJ23 with LFG f-structures, which they also automatically convert to grammatical relations of Carroll *et al.* (1999). They report $35-36\%$ error reduction on this corpus when compared to a disambiguation technique where they select a parse randomly from the available ones according to the uniform distribution. Unfortunately, it is hard to compare this performance to other parsers' accuracy. Furthermore, no separate measures are given for NLDs. In a different evaluation, Riezler *et al.* (2002) report performance on the test set proposed by Carroll *et al.* (1999). Their system outperforms Carroll *et al.*'s (1999) one by 1% on recovering unlabelled dependency relations. This evaluation scheme includes NLDs as well as local dependencies.

Although the results attained by systems are very encouraging, they suffer from the relatively large amount of manual work they presuppose: both approaches rely on a large-scale hand-written grammar, a resource which is as time consuming to create as a treebank. Furthermore, Riezler *et al.* (2002) also presumes a manually bracketed treebank. Efficiency also seems to be a problem: Riezler *et al.* (2000) report faster training than parsing and they resort to parsing off-line.

Indeed, it is a serious bottleneck that both the estimation and the ranking procedures require the enumeration of all possible analyses licensed by the grammar for the sentences in the training and test data. That is, in the worst case, the (symbolic) parser might have to enumerate exponentially many analyses before the stochastic procedures come into play. To alleviate the problem, Geman and Johnson (2002) propose a dynamic programming algorithm which operates on the Maxwell-Kaplan packed parse representation (Maxwell and Kaplan 1995a,b). The speed of the algorithm crucially depends on the ordering of some internal variables. Unfortunately, finding the best ordering is an $\mathcal{NP}$-complete problem; Johnson (2003), however, claims that there are "several efficient heuristics" which find close-to-optimal ordering fast. In a different approach, Osborne (2000a) suggests using an informative sample of the training corpus to estimate the parameters.

LFG-DOP

One problem with the approaches discussed above is that they require a manually constructed grammar. LFG-DOP offers a method of inducing a probabilistic LFG grammar from a corpus annotated with LFG structures (Bod and Kaplan 1998, 2003). As is the case for TREE-DOP (Section 3.1.3), LFG-DOP first defines a set

of decomposition operators which divide the analyses into smaller fragments. In this case, however, these operators are somewhat more complex, since they have to take care of both the f- and the c-structures. Similarly, the composition operator has to construct both the c-structure (tree substitution) and the f-structure (recursive unification of attribute-value matrices).

There is no known efficient parsing algorithm for LFG-DOP which always returns the highest probability analysis. Instead, parsing is a two-step process. First, a chart parsing algorithm creates a compact representation of a superset of all possible analyses. This process ensures the well-formedness of the c-structure only. Then, the most probable well-formed LFG analysis is derived during a disambiguation (chart-decoding) phase, where a large number of well-formed derivations are sampled from the chart using Monte Carlo sampling (Neal 1993). The probability of these derivations are calculated on the fly. A further complication is that the completeness of an LFG analysis can only be checked at the end of the derivation. This involves further sampling at the end of the parsing process from partially well-formed derivations. In order to ensure that this last sampling phase finds the best well-formed LFG derivation, one has to keep the number of samples reaching this phase high (around 10000). In this approach, there is a trade-off between accuracy and speed: the larger the sample size is, the more accurate and the slower the parser is.

Bod and Kaplan (2003) report encouraging results on small corpora: the F-score for recovering semantic structures is $87-89\%$. Moreover, the LFG-DOP model is reported to outperform the TREE-DOP model of Bod (2001), which achieves state-of-the-art performance on the Wall Street Journal corpus.

Neumann (2003) presents a small-scale experiment with an HPSG-DOP parser, and he finds that the DOP parser arrives at the same best derivation as a rule-based HPSG parser in 68.6% of the cases (with 87% coverage). Since the rule-based parser is not guaranteed to find the correct parse for the input and it is, in fact, used to generate the training data for the DOP approach, it is rather difficult to interpret these results; it seems that the DOP approach approximates the parser with 68.6% accuracy.

There are two major bottlenecks which make it difficult to use LFG-DOP in practical situations. First, the parsing regime it employs is rather computation-intensive and thus time-consuming. Second, it requires a large LFG-annotated corpus, which up to this date is not available, although there are experiments trying to infer f-structures from the Penn Treebank annotation making use of functional labels present in the Treebank (Cahill *et al.* 2002).

3.1.5 Combinatorial Categorial Grammar

Osborne and Briscoe (1997) define an initial probability model for stochastic categorial grammars. They compare two parametrisation schemes. The first one is a maximum likelihood model, trained using the EM algorithm. The second model is based on the minimum description length principle and Osborne and Briscoe (1997) present a bottom-up greedy training algorithm which is much more efficient than the EM algorithm. The second model is not only more efficient to train, but outperforms the MLE model on a small manually annotated test set.

The existence of a large treebank annotated with CCG structures (Hockenmaier and Steedman 2002a, Hockenmaier 2003a) allows both grammar induction and supervised estimation of probability models for CCG. The CCGBANK is derived from the Penn Treebank, and therefore the results reported on experiments with this corpus are more or less comparable with other parsers' scores on the same corpus.[5]

Hockenmaier and Steedman (2002b) and Hockenmaier (2003a, Chapter 5) describe a generative model for normal-form derivations in CCG. The model is defined on CCG derivation trees; it is essentially a context-free approximation of CCG. The probability model is very similar to Model 2 of Collins (1997). As a consequence, it only implicitly handles non-local dependencies. Hockenmaier (2003a) reports slightly (0.4%) higher accuracy for unlabelled surface dependencies (cf. Section 4.1) than Collins (1999), although it is not clear whether the improvement is due to a more appropriate model or to cleaner training data.

Hockenmaier and Steedman (2002b) and Hockenmaier (2003a) also give an informal analysis of the accuracy of their parser with respect to a handful of constructions with NLDs. Specifically, they inspect whether overt complementisers occur with appropriate lexical categories, or whether special type-changing rules apply. The results reported there are encouraging although it is difficult to assess how good they are without reference to other approaches.

Hockenmaier (2003a) attempts to compare her results with those of Johnson (2002) and Collins (1999) (Model 3) as well, although they are not strictly comparable. She reports the same accuracy on subject extraction as Collins (1999). Since Johnson (2002) does not distinguish the cases of subject and object relatives, the comparison of the two approaches is even more difficult. Hockenmaier (2003a), however, conjectures that her model is more accurate at handling extraction than Johnson's, based on the fact that her model is highly accurate at recovering subject extraction (96.2% F-score) and that the majority (78%) of the extraction constructions in WSJ00 is of this type. For detecting control verbs (i.e.,

[5] During the semi-automatic conversion from the PTB to CCG structures, annotation errors in the original Treebank are corrected (Hockenmaier and Steedman 2002a, Hockenmaier 2003a, Chapter 3 and Appendix), therefore the models using the CCGBANK rely on cleaner training data.

the empty subjects of the embedded clause without finding its antecedents), Hockenmaier (2003a) reports 76.2% F-score. Again, it is difficult to assess how this figure relates to the scores reported by Johnson (2002).

One important shortcoming of the model presented above is that it fails to model NLDs explicitly, and thus is subject to the criticism presented in Abney (1997). To alleviate the problem, Hockenmaier (2003b) (also Hockenmaier 2003a, Chapter 6) presents a generative model for predicate–argument structure based on CCG analyses. This model explicitly handles NLDs (i.e., re-entrancies) by modelling all word-word dependencies (as well as CCG derivations). In particular, it allows a dependent to be conditioned on multiple heads.

There are two important difficulties when defining such a probability model. The first one concerns the estimation of the probability of a word given multiple heads, that is, of $P(w|h_1,\ldots,h_n)$. In order to have a reliable estimate, Hockenmaier (2003b) proposes a simple linear combination of the probabilities $P(w|h_i)$ with equal weights:

$$P(w|h_1,\ldots,h_n) \approx \frac{1}{n}\sum_i P(w|h_i) \tag{3.3}$$

Clearly, several other estimation techniques could be used here.

The second problem, which as we have seen in the previous section constitutes a problem for other approaches as well, is the design of an efficient parsing algorithm. The problem is that a dynamic programming approach is hard to conceive for the parsing problem: it might happen (indeed it does happen) that an otherwise complete edge contains a headword whose argument slot is not filled. Such a situation arises, for instance, in the case of object relative clauses, where we cannot calculate the inside probability of the edge representing the relative clause, since the inside probability of its verb is not known until the head of its object is identified. This means we have to postpone the evaluation of this edge with respect to other edges until all the heads are saturated, which might only happen late in the derivation; in fact, we might only evaluate the probabilities at the end of the derivation process. In the worst case, the dependency relations might turn out to be circular, leaving the problem undecidable.

A similar problem arises due to the fact that constituents might depend on several heads (the number of which we do not know in advance): even if we combine a constituent with one head, we cannot evaluate its inside probability, since we do not know whether the same constituent depends on other heads or not. Again, in the worst case, we can only evaluate the analyses according to the probability model at the end of the derivation. This might lead to the enumeration of exponentially many analyses.

Indeed, as Hockenmaier (2003b) notes, parsing with this model turns out to be infeasible without suitable heuristics for the beam search. Specifically, she

proposes a very aggressive beam search strategy to cut the search space of the parser; as a consequence, she can no longer ensure finding the highest probability parse. Even with this strategy, Hockenmaier (2003b) could only parse sentences of length less than or equal to 40 tokens.

Given the difficulties with the model, the results Hockenmaier (2003b) presents are impressive: she reports 88.7% unlabelled F-score for retrieving predicate argument structure (which is slightly (1.3%) worse than the score achieved by Clark *et al.* (2002)). Unfortunately, this score is difficult to compare with our results, since it includes dependency relations we do not include in our evaluation (e.g. in auxiliary constructions, the dependency between the subject and the main verb).

Hockenmaier (2003b) also presents a separate evaluation for non-local dependencies. Unfortunately, since these dependencies do not always involve empty nodes according to the original Treebank annotation (e.g. auxiliaries and coordination are treated as involving NLDs), these scores are not comparable with results reported by other approaches for recovering NLDs (Johnson 2002, Jijkoun 2003, Dienes and Dubey 2003a,b).

A third approach for handling NLDs with CCG is presented by Clark *et al.* (2002). This approach crucially differs from the ones discussed above in that it defines a probability model for the predicate–argument structure instead of modelling CCG derivations. Another interesting feature of the approach is that it employs a supertagger (Clark 2002) as a preprocessing step. This supertagger assigns a set of possible lexical categories to words and thus reduces the search space of the parser. The parser, then, builds full CCG structures, exploiting the probability model to further reduce the search space (using dynamic programming and beam search).

Although the results are encouraging, albeit difficult to compare with non-CCG approaches, the approach faces two problems. First, even with the beam search strategy, the parser seems to run into efficiency and coverage problems: Clark *et al.* (2002) report 89.2% initial coverage (with 8.7% of the sentences missing due to time-out), which they manage to raise to 98% adjusting the parameters of the parser for the unparsed cases.

The other problem concerns the probability model: Clark *et al.* (2002) adopt the model of Collins (1996), which turns out to be unsound (i.e., the probabilities are not proper maximum-likelihood estimations and the model is deficient, Collins 1999). Moreover, they assume independence between the probability of words filling the argument slots of the same head. They also normalise the probabilities according to the number of dependencies occurring in the analysis, which is a questionable approach (cf. Manning and Schütze 2001, p. 442f).

3.1.6 Dependency grammars

Dependency structures are generally regarded as an alternative syntactic representation to phrase structure; they might, in fact, be better suited as an intermediate representation facilitating the construction of semantic analysis by explicitly encoding *all* predicate–argument relations, including non-local ones. Moreover, modelling lexical affinities between heads and dependents proves to carry important cues for parsing.[6] These characteristics raised considerable interest in the parsing community and several probability models have been proposed for dependency grammars (DGs).

As it turns out, devising a parser and a probability model for dependency grammars is rather complicated. The first bottleneck is the wide variety of different formalisations of dependency grammar (Hays 1964, Gross 1964, Gaifman 1965, Robinson 1970, Sgall *et al.* 1986, Mel'čuk 1988, Hudson 1990, Bröker 1997, Duchier and Debusmann 2001). Although all of these formalisms agree on positing a connected dependency structure with one root, they greatly differ in further details which have a great impact on the generative power of the formalism. Specifically, there is a considerable amount of disagreement whether re-entrancies and non-projective structures are permitted by the formalism. Allowing both is essential for handling NLDs, but they considerably increase the generative power of the grammar formalism.

Projectivity requires the dependency structure, when drawn above the words, to be planar, i.e., the dependency links do not cross, and the root of the dependency tree is accessible, i.e., it is not buried under a dependency link. Note that projectivity is not a property of the dependency structure *per se*, rather it comes into play when the dependency structure is *linearised*, that is, when word order is determined. Non-projectivity, required for describing NLDs, is a powerful extension: even if it is restricted considerably on linguistic grounds, the recognition problem for dependency grammars allowing it turns out to be $\mathcal{NP}$-hard (Neuhaus and Bröker 1997, Koller and Striegnitz 2002).

Re-entrancies, i.e., cases where a word depends on several other heads, is a problem for statistical modelling: as we have seen in Section 3.1.4, this feature requires a more complicated probability model than simple relative frequency counts (Abney 1997). Even with the appropriate probability model at hand, both training and testing turns out to be time consuming. Furthermore, re-entrancies make it difficult to use efficient dynamic-programming approaches.

These problems have led to important restrictions on possible dependency structures. Specifically, almost all work on stochastic dependency parsing assumes projective dependency structures, which prevents them from modelling

[6] This insight also led to the development of head-lexicalised PCFG models.

the majority of NLDs. This move, in fact, restricts the power of DGs to context-free. More precisely, as Nivre (2002) shows, some of these projective dependency grammars, namely the formalisations of Carroll and Charniak (1992) and of Eisner (1996a,b), define languages that lie between context-free and regular.

The DG Carroll and Charniak (1992) present is very much like a CFG, with the restrictions that

(i) the set of nonterminals is $\mathcal{N} = \{S\} \cup \{\overline{t} | t \in \mathcal{T}\}$, where $\mathcal{T}$ contains the terminal symbols (words) and S is the start symbol; and

(ii) the rules are from the set $\{S \rightarrow \overline{t} | t \in \mathcal{T}\} \cup \{\overline{t} \rightarrow \alpha t \beta | t \in \mathcal{T}, \alpha, \beta \in \mathcal{N}^*\}$.

Carroll and Charniak discuss a scheme for automatic induction of DG rules from unlabelled corpora and use the EM algorithm to parametrise the resulting set of rules. They only present an informal evaluation and find that the unrestricted rule induction algorithm performs rather poorly. Therefore, they impose six restrictions on the right-hand side of the rules given the left-hand side (such as $\overline{pron}$ does not allow a verb on its RHS), and find that the grammar induced with restrictions enforced is the correct one. Carroll and Charniak (1992) note the inability of their grammar formalism to handle non-local constructions, such as WH-movement.

Another framework motivated by DG is Link Grammar (Sleator and Temperley 1991). In Link Grammar, the lexical entry of each word specifies two ordered sets of (labelled) connectors (the left and right connectors), called the disjunct. "A word W with disjunct $d = ((l_1, \ldots, l_m), (r_1, \ldots, r_n))$ can be linked to other words by connecting each l_i to the right connectors of words to the left and also connecting each r_j to left connectors of words on W's right. Links are only permitted between *matching* connectors" (Grinberg *et al.* 1995). The connectors in the disjunct are ordered: the word connected to W by l_i is closer to W than the word connected by l_{i+1}. A linkage for a sentence is specified by linking every connector of every word with a connector of a different word's disjunct. A valid linkage has to meet the criteria of

(i) connectivity (the graph of links and words is connected),

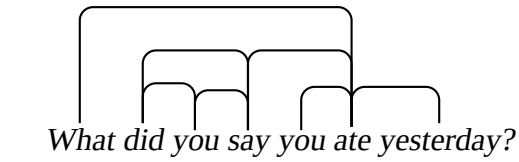
(ii) planarity (when drawn above the sentence, links do not cross),

(iii) ordering, and

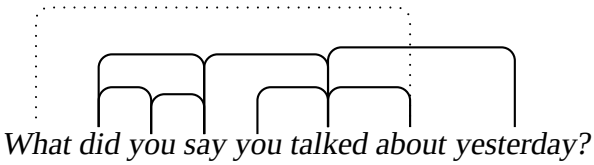
(iv) exclusion (no two links connect the same pair of words).

Note that planarity is a slightly weaker restriction than projectivity: unlike projectivity, it allows the root to be buried under a link. For example, in Figure 3.1a the root *"say"* is buried under the link between *"what"* and *"ate"*; the graph is planar,

that is, the linkage is valid, though non-projective. This freedom allows capturing some NLDs, but not all of them. For instance, the non-local dependency in Figure 3.1b cannot be expressed with a planar graph, the dotted link must cross other links. The generative power of Link Grammar is context-free.



(a) A planar but non-projective linkage



(b) A non-planar and non-projective linkage

Figure 3.1: The difference between planarity and projectivity.

Lafferty *et al.* (1992) define a generative probability model for Link Grammar. The set of model parameters is $P(W,d,O|L,R,l,r)$, where W is the word currently being generated with its disjunct d. This generation process depends on the possible left and right connectors (l and r) connecting the word W with the words L and R. Whether the word is actually linked with the word on the right or on the left, or with both is defined by the orientation O. Lafferty *et al.* (1992) decompose and approximate the probability in the following manner:

$$P(W,d,O|L,R,l,r) \approx P(W|L,R,l,r) \cdot P(d|W,l,r) \cdot P(O|d,l,r) \qquad (3.4)$$

They propose using the EM algorithm to estimate model parameters, given a handwritten grammar. No experimental results are provided.

As Fong and Wu (1995) as well as Collins (1999) note, the decomposition of the probabilities in this manner is problematic: implicitly, W is generated before it is known whether it attaches to L, R, or both. This assumption makes the model very sensitive to sparse data without the possibility to back off to bigram or unigram probabilities. To alleviate the problem, Fong and Wu (1995) propose a different estimation of the model parameters:

$$P(W,d,O|L,R,l,r) \approx P(O|l,r) \cdot P(W|L,R,l,r,O) \cdot P(d|W,l,r,L,R,O) \qquad (3.5)$$

where the last two terms can be further simplified once the orientation O is determined. They compare the two models in a small-scale experiment, inducing an untyped version of Link Grammar. They find that their model is able to find the correct parse in 68.75% of the cases, whereas Lafferty *et al.*'s (1992) model achieves 46.0% accuracy. No large-scale experiments are reported with Link Grammar.

Large-scale parsing experiments using corpus-induced dependency-based grammars are presented by Collins (1996) and Eisner (1996a,b); here we focus on Collins's model and Eisner's Models C and D, which are refinements of Models A and B. Model C is a generative model (very similar to Model 1 of Collins 1997, 1999), Model D is its conditional counterpart. Both authors restrict themselves to projective dependency grammars (although Collins (1996) does not state this explicitly), where every word except the root depends on exactly one head (that is, no re-entrancies are allowed). Projectivity forbids modelling NLDs. An important difference between these approaches and Link Grammar is that here the links are directed, pointing from the dependent to the head.

The main differences between Collins's and Eisner's models, apart from different parametrisation, are:

- Collins uses typed links, where the label for a dependency is derived from the phrase structure: the label is a triple, the elements corresponding to the nonterminal labels of the head, the dependent and the parent nodes; Eisner's parser produces bare-bones dependency structures.
- Collins employs separate probability models for tagging and NP chunking; chunks are substituted by their heads in the dependency model.
- Collins's parser takes intervening punctuation explicitly into account via the distance measure it employs.

These differences turn out to be important as far as parsing accuracy is concerned. In a large-scale experiment, using a dependency-based evaluation metrics, which measures the percentage of words finding their correct tag and correct parent, Eisner (1996a) reports 87.7% accuracy for Model C, 90.0% for Model D, and 92.6% for the model of Collins (1996). This measure does not take NLDs into account.

Schneider (2003a,b,c) develops a probabilistic dependency parser which allows non-projectivity in "specific, well-defined situations, such as in WH-questions" (Schneider 2003c). In order to avoid combinatorial explosion, however, he restricts discontinuity to the minimum by handling most of the NLDs as a post-processing step (cf. Section 3.2). The parser relies on a hand-written grammar, and also employs a number of finite-state preprocessors, such as a tagger, a rule-based chunker, a head-finder, and a lemmatiser. Schneider (2003a,b,c) defines different probability models for each dependency relation and uses these scores to guide the parser.

The parser, trained on the PTB (WSJ02–24) is evaluated using the dependency annotation scheme of Carroll *et al.* (1999) on their corpus. The results Schneider (2003a) reports are very promising. When compared to Charniak's (2000) parser, it achieves $7-10\%$ better F-scores on the object and subject relations (although the scores for Charniak's (2000) parser might be compromised by the translation process from treebank output to dependency structures (Preiss 2003)). On the same grammatical relations, Schneider (2003a) reports a 3% improvement over Briscoe and Carroll's (1993) parser.

As for NLDs, Schneider's parser covers NLDs involving **WH–...**, **TOP–S** as well as **NP–NP** and **PRO–NP** traces. **WH–NP** and **TOP–S** EEs require special rules in the grammar to deal with inversion, which depend on "linguistic non-standard assumptions" (Schneider 2003a). **NP–NP**s and **PRO–NP**s are handled by post-processing the output of the parser, an approach similar to Johnson's (2002). Schneider (2003b) presents an informal evaluation of his system with respect to some constructions involving NLDs. The results, again, are promising, but hard to compare with other approaches (due to different test data and different evaluation scheme).

Further work on statistical parsing with dependency grammars include (Nivre 2002), which generalises Eisner's (1996b) generative model. Samuelsson (2000) develops a generative statistical model for dependency syntax; in this model crossing dependencies are allowed, but re-entrancies are not. Neither authors test their models experimentally. Zeman (1998, 2002) presents a dependency parser for Czech, which allows neither non-projectivity nor re-entrancies. Dienes *et al.* (2003) propose a probability model similar to STAG (Resnik 1992, Schabes 1992) for a constraint-based dependency grammar formalism (Topological Dependency Grammar, Duchier and Debusmann 2001), which allows non-projectivity and re-entrancies. The model is not tested because it requires a grammar and extraction of this type of grammar from corpora proves to be very difficult (Korthals 2002).

Summary

It is rather difficult to compare the results presented in this section: almost each approach uses its own training and test set, as well as its own evaluation metric. The outputs of various parsers are very different, which again prevents straightforward comparison. Nevertheless, there are certain tendencies which seem to apply to most of the models. The increased ambiguity due to the presence of non-local dependencies enlarges the search space the parser has to explore. Therefore, these parsers tend to be slow thwarting their use in large-scale on-line applications such as question answering or machine translation. Furthermore, most parsing models employ a more expressive grammar formalism to handle NLDs than a CFG. These richer grammar formalisms, however, require labour-intensive manual (or

semi-automatic) creation of treebanks or large-scale hand-written grammars, and also have problems with coverage. Approaches remaining within the context-free framework fail to model all types of NLDs.

3.2 Post-processing approaches

To overcome the problems with incorporating NLDs in a parser, Johnson (2002) proposes a simple post-processing approach: first, he parses sentences with a state-of-the-art parser not designed to handle NLDs (which, on the other hand, is fast), then he attempts to re-introduce the EEs representing non-local dependencies, using patterns automatically extracted from the training corpus. Jijkoun (2003) augments Johnson's (2002) algorithm with a machine learning approach to handle ambiguous patterns. Similarly, Higgins (2003) presents another machine learning approach which detects WH-gaps given the otherwise correct parse tree.

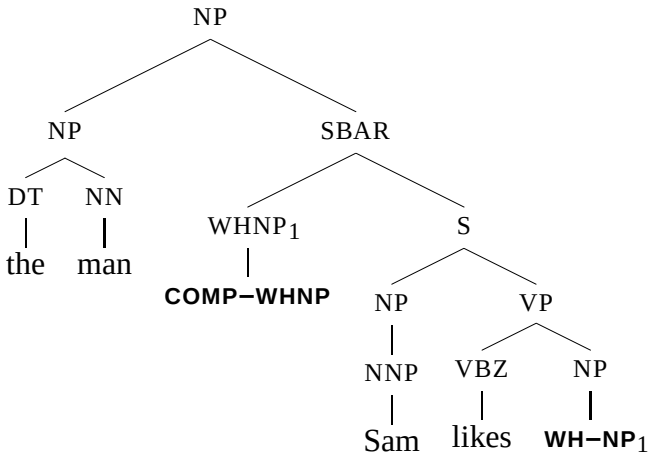
All these approaches suffer from the pre-processing setup: they are very sensitive to errors the parser commits. Indeed, parsers not designed to capture NLDs tend to make mistakes exactly in the environments that contain crucial cues for the post-processing algorithms. Therefore, post-processing approaches are considerably less accurate when tested on the output of a parser as opposed to testing them on perfect trees.

3.2.1 Johnson (2002)

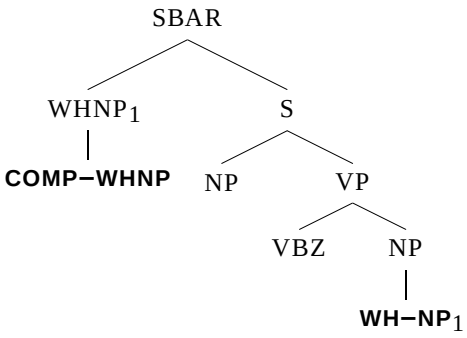
The underlying idea behind Johnson's (2002) approach is simple but elegant: EEs tend to occur in relatively restricted environments characteristic of the type of the empty node. Even though parsers, such as the ones presented in Charniak (2000) or Collins (1999), do not recover empty nodes, the characteristic environments where these gaps occur are still present in their output. Therefore, Johnson automatically extracts the patterns (tree fragments) where EEs occur and reinserts empty nodes whenever these patterns match.

Consider, for instance, the tree depicted in Figure 3.2a, containing EEs representing non-local dependencies. The pattern Johnson (2002) extracts from this subtree (Figure 3.2b) is the minimal set of connected local trees containing the EE and all the nodes it is co-indexed with.[7] If an empty node is not co-indexed, its pattern is the minimal branching local tree containing it (that is, should the **WH–NP** gap have no antecedent, the corresponding local tree would be a VP node dominating it and not the NP node). A pattern p matches a subtree t if and only if t is an extension of p ignoring empty nodes in p. For instance, the pattern in Figure 3.2b

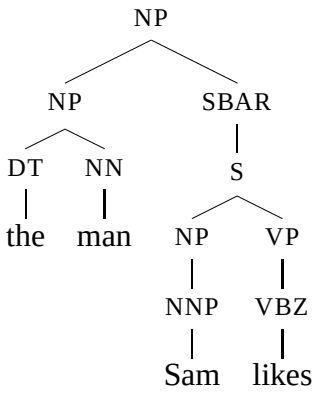
[7] Note the strong resemblance of these patterns to elementary trees in TAG (modulo lexicalisation).



(a) A tree containing empty nodes (Johnson 2002).



(b) The pattern extracted from (a).



(c) Typical parser output matching the pattern.

Figure 3.2: The pattern matching algorithm of Johnson (2002).

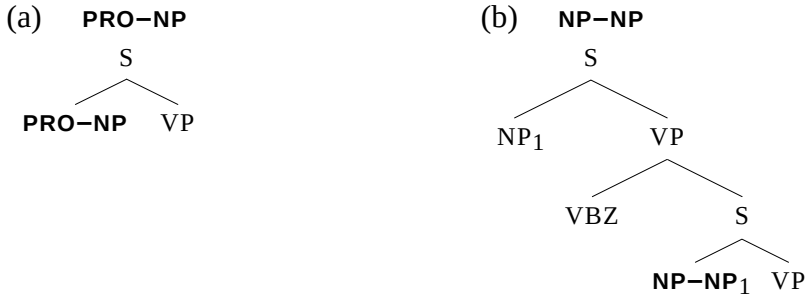


Figure 3.3: The most frequent patterns for **NP–NP** and **PRO–NP** according to Johnson (2002).

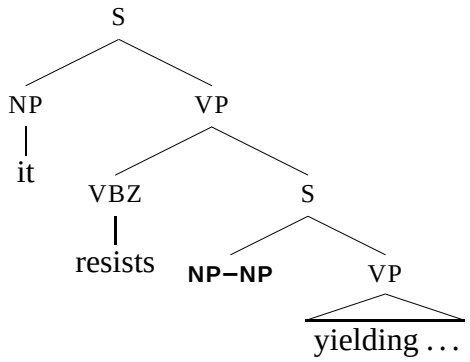
matches the tree in Figure 3.2c . If a pattern p matches a subtree t, it is possible to substitute p for t, that is, to introduce the EEs and co-indexation into the subtree. Thus, substituting the pattern of Figure 3.2b in the subtree of Figure 3.2c results in the tree depicted in Figure 3.2a.

A pattern might match subtrees which actually do not contain empty elements. Patterns which are more probable to raise a false alarm, i.e., which are more probable to match subtrees without NLDs than with them, are discarded. It is also possible that a tree might be matched with several patterns; in this case the algorithm has to select one of the patterns, that is, one has to define a global ranking of the patterns. Johnson (2002) reports on experimenting with several possible ranking schemes, without considerable difference. Therefore, he uses a simple criterion: the algorithm tries deeper patterns first.

Although this approach is relatively simple (and fast once we have the parse tree!), it still performs surprisingly well. It has, however, two important limitations. First, it is not robust enough with respect to parsing errors: Johnson (2002) reports 7% better results when he applies the algorithm to gold-standard trees stripped of EEs. Indeed, cases involving EEs seem to be difficult for the parser Johnson employs (Charniak 2000), returning parse trees which do not match the correct pattern. This behaviour affects adversely the accuracy of recovering NLDs involving long-distance dependencies such as WH-movement, where the parser might make several mistakes on the path between the EE and its antecedent. The algorithm has considerably lower score on detecting corresponding WH-complementisers as well.

The second deficiency of Johnson's (2002) algorithm is the lack of lexicalisation. This limitation is manifest in the low scores for controlled and uncontrolled **PRO**s: even in the case of the gold-standard trees, the algorithm attains only 63% and 57% F-scores, respectively. These scores further decrease when the input to the pattern matching algorithm is the parser's output. As an illustration, consider the two most frequent patterns Johnson (2002) employs to detect **PRO–NP**s and

(a) *But it resists* **NP–NP** *yielding political ground.*



(b) *But it means* **PRO–NP** *paying the horse's maintenance.*

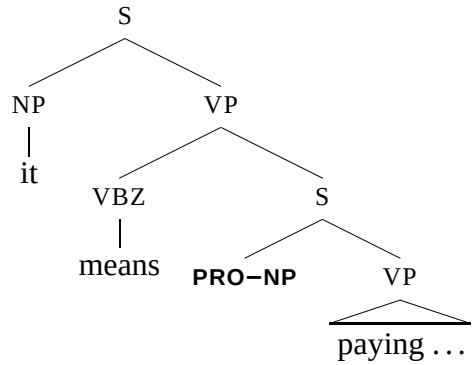


Figure 3.4: A problem for the pattern matching approach.

NP–NPs (Figure 3.3). The main difficulty is that pattern (b) contains pattern (a). If a subtree matches both patterns, the algorithm *always* chooses the deeper one, pattern (b), regardless of the lexical identity of the main verb. Now, consider the examples in Figure 3.4: if we disregard lexical information, both trees have the same structure as depicted in Figure 3.3b, but one of them contains an uncontrolled **PRO–NP**, the other one a controlled **NP–NP**. The cause of the different behaviour is the different verbs: *resists*, along with other verbs such as *begins*, *starts*, *enjoys*, *likes*, etc., requires its gerundial complement to have a referential subject. On the other hand, verbs such as *means*, *involves*, *prohibits*, etc., tend to have a gerundial complement with an uncontrolled **PRO–NP** subject. Johnson's (2002) unlexicalised pattern matching algorithm is unable to capture these generalisations and wrongly assumes that both subtrees involve **NP–NP** empty nodes.

3.2.2 Jijkoun (2003)

Jijkoun (2003) attempts to improve upon Johnson's pattern matching algorithm in two respects. First, he pre-processes the Treebank before extracting patterns by converting the phrase-structure trees of the Treebank into dependency trees using the conversion algorithm described in Buchholz (2002). This step basically localises non-local dependencies; while WH-extraction involves many intermediate nodes in Johnson's approach, it only involves one such node in the dependency structure. Non-local dependencies are dependency relations between two nodes in the dependency tree not in immediate dominance relationship. The dependency relations might have different possible labels NP-OBJ, NP-SUBJ, ADVP, etc., expressing the type of the predicate–argument relationship between the nodes.

The move from phrase structure to dependency structure results in fewer, more compact patterns: while Johnson employs approximately 9000 patterns often involving a large number of nodes, Jijkoun ends up with 16 compact patterns covering 93.7% of the EEs. All of these patterns are ambiguous: they contain two nodes that are either in non-local dependency relation of various types or not. This ambiguity corresponds to situations where a pattern matches a subtree without an EE in Johnson's approach, or where a pattern subsumes another one.

The small number of ambiguous patterns, however, allows Jijkoun (2003) to use a machine learning approach to disambiguate the patterns and determine the type of the non-local dependency (if any). He employs a memory-based learner (TiMBL, Daelemans *et al.* 2002) to perform this task. The learner takes both structural features (such as the pattern, or whether the word already has a subject/object) as well as lexical (POS) features into account.

When evaluating the system on the perfect trees (after deleting NLDs), Jijkoun (2003) reports 86% labelled F-score for recovering all non-local dependencies. These results are around 10% better than the scores Johnson (2002) achieves on gold-standard trees, though they are not strictly comparable. In particular, Johnson's (2002) scores include, on the one hand, EEs that are very easy to detect (e.g. empty **UNIT**s). On the other hand, Jijkoun (2003) does not handle **PRO–NP**s and empty WH-complemetisers, which appear to be problematic for Johnson's approach (and, as a matter of fact, for our models as well).

Although Jijkoun's (2003) results are very encouraging when testing his system on perfect trees, his model faces immense problems when he applies it on trees provided by a parser (that of Collins 1997, 1999). This is, in fact, not surprising, since the algorithm he employs to convert phrase structure trees to dependency structures heavily relies on functional tags and empty elements in the gold-standard PTB annotations (Buchholz 2002). This information, however, is not present in the output of the parser. There are several possibilities to remedy the situation: one might enrich the nonterminals with functional labels and train a parser with the richer set of nonterminals,[8] or reinsert functional tags as a post-processing step (Blaheta and Charniak 2000). Nevertheless, we suspect that Jijkoun's (2003) approach would still be subject to the same sensitivity to parse errors around NLD-sites as Johnson's (2002).

3.2.3 Higgins (2003)

A third post-processing model similar to Johnson's and Jijkoun's approach is presented by Higgins (2003). His algorithm detecting WH-gaps works in the fol-

[8] Interestingly, using functional labels in the parser seems to have a negative effect on parse accuracy (Klein and Manning 2003).

lowing fashion: it starts at the node dominating a WH-complementiser; in each step, a classifier emits symbols instructing the system to either recurse into the *n*th daughter (RECURSE-*n*) or posit a WH-gap as the *m*th daughter (GAP-*m*). The classification is done based on information about the WH-complementiser triggering the search for a gap, the depth of the recursion and structural relations such as the nonterminal label of the parent, the current and the daughter nodes. Higgins trains a feed-forward multi-layer perceptron to perform the classification task.

Higgins (2003) reports 92% precision for recovering the complete path leading to the WH-gap. These results are, however, not strictly comparable with those of Johnson (2002) and Jijkoun (2003), since he uses a different split of the data into training and test sets.

This approach has several deficiencies. First, it only covers WH-gaps, the detection of which proves to be relatively easy for other models as well. Second, the search for gaps is triggered by the presence of a WH-complementiser, which is problematic in the case of empty WH-complementisers. It is not clear how the algorithm could be modified to deal with such cases, i.e., to be applicable on the output of a statistical parser which does not predict empty complementisers. Furthermore, it is not clear how this approach could be extended to other EEs which do not require a triggering WH-operator or other syntactic cue. Finally, this approach is also very sensitive to parsing errors.

3.3 Buchholz (2002)

Buchholz (2002) approaches the problem of reconstructing the grammatical relations between a head verb and its dependents from a completely different perspective: she does not employ a parsing component. Instead, she uses a chunker to preprocess the input, and a memory-based classifier to decide for each chunk whether it is related to the verb(s) in the sentence. If it is, the classifier also attempts to find the type of the relation (e.g. temporal NP, subject NP, etc.).

The setup is the following: first, a tagger assigns POS-tags to the words and a chunker divides the input sentence into non-overlapping phrases (in most of the experiments Buchholz (2002) reports, she uses tags and chunks from the gold-standard), then a memory-based classifier classifies each chunk with respect to each verb. The classifier uses a wide variety of information: the distance of the focus chunk from the verb (counted in terms of chunks), the number of VP chunks between the focus chunk and the verb, the verb itself, and selected information about the focus chunk as well as about other chunks around the focus chunk (the type of the chunk, the headword and headPOS of the chunk and the type of the preceding preposition, if any). To evaluate the system, Buchholz uses tenfold cross-validation on WSJ10–19.

Apart from local dependencies, this model is able to find some non-local ones as well. In particular, it handles NLDs represented with **WH–...** and **NP–NP** EEs in the PTB. The system achieves 67.3% labelled F-score for finding the correct label for the dependency relations between a verb and its non-local dependent(s) (with 80.9% precision and 57.7% recall). The results are very promising, though not strictly comparable with the ones presented here. On the one hand, the task is more difficult, since Buchholz (2002) uses different (and smaller) training and test sets and a richer set of labels (encoding the type of grammatical relation as annotated in the Treebank). On the other hand, the input to her model already contains the correct chunks and headwords, whereas we start from POS-tagged text only. One very important advantage of Buchholz's (2002) approach is its speed: since it does not apply a parser, its complexity is linear in the size of the sentence.

We see three important deficiencies of this model. First, Buchholz does not handle **PRO–NP** traces, which do play a considerable role in the predicate–argument structure of the sentence. Second, the choices the system makes are essentially local. Therefore, nothing prevents the model from assigning multiple dependents with the same label to the same verb. Similarly, the system can associate a chunk with several heads. Although such constructions are not impossible, they arise only by virtue of coordination, which the approach does not handle. Finally, as we will see in Chapter 5, local cues around the extraction site (i.e., mostly around the verb) are very strong and useful; Buchholz (2002) effectively ignores these cues. Our conjecture is that the approach can be considerably improved using global search, and incorporating information about the environment the verb occurs and about coordination.

Summary

As we have seen in this chapter, there is a wide range of different approaches to applying machine learning techniques to recovering NLDs, with varying success, although it is generally hard to compare them, since most of the approaches use different evaluation metrics. We take the results presented in (Johnson 2002) as our baseline. The models we develop in this dissertation achieve state-of-the-art performance on the task of recovering non-local dependency relations according to the evaluation metrics proposed by Johnson (2002).

Our goal, however, is not only to design the best possible system: we are also interested in understanding the nature of the problem of recovering NLDs. It is widely assumed that this task is rather difficult, requiring full parsing with a sophisticated grammar framework. But how difficult is it? Are all cases of NLDs difficult to recover or can we isolate easy subproblems? What makes the easy ones easy and the difficult ones difficult? Do we really need a sophisticated parser

or would a simple PCFG do? Ultimately, do we need a parser at all? This dissertation aims at answering these questions showing that the combination of relatively simple shallow approaches manages to achieve state-of-the-art accuracy, provided they are combined in an appropriate way.

In our present overview, we have seen two major bottlenecks: efficiency and the availability of training material. Most of the approaches described above which manage to achieve good results on finding NLDs seem to suffer from both problems: they tend to be slow due to the increased parsing complexity the elaborate grammar framework imposes on them, and they require considerable amount of manually-annotated material, which is usually scarcely available. From this point of view, our method couples with the approaches described in the previous two sections: we would like to explore how far one can get using existing (labelled) training material and simple tools.

Chapter 4

Finding NLDs with an unlexicalised parser

This chapter[1] explores whether an unlexicalised PCFG parser is able to detect NLDs. First, we introduce the evaluation metrics we use throughout this thesis. We also show an algorithm recovering antecedents for EEs based on a parse tree with `gap+` variables (cf. Section 2.3). Then, we turn our attention to an unlexicalised PCFG parser designed to recover non-local dependencies. As we will see, the parser is not successful at the task: it is inaccurate, slow, and cannot parse 40% of the sentences. This is in accordance with our hypothesis that handling NLDs is difficult. Interestingly, when the same parser is informed about the sites where EEs occur, it can recover the antecedents with high accuracy. This shows that the main complication is EE detection and not antecedent recovery.

4.1 Evaluation metrics

As we formulated the task in Section 1.2, recovering NLDs can be viewed as a two-step problem: we first determine the sites where such NLDs occur, then, as a next step, we find the dependents involved in the relation. In the representation scheme we adopted, finding the sites amounts to detecting where EEs occur, whereas finding the dependents means recovering the antecedents (if any) for the EEs in question. As we will see, it proves to be useful to keep these tasks separate, even in the case of a system which performs both at the same time. Therefore, we introduce two subtask-specific evaluation metrics: one for EE detection and

[1] This chapter is an extension of (Dienes and Dubey 2003b). It contains a more detailed discussion of the antecedent recovery algorithm. We also introduce new dependency-based evaluation metrics. The unlexicalised parser, however, is the same as in (Dienes and Dubey 2003b); I am grateful to Amit for implementing it.

one for antecedent recovery. Both metrics are based on (Johnson 2002). These metrics only take string positions into account and do not evaluate dependency structures. To assess the performance of various models on recovering dependencies, we also introduce a dependency-based metric, following e.g. Lin (1995), Carroll *et al.* (1998, 1999), Collins (1999).

Detecting empty elements

Following Johnson (2002), we say that an EE is correctly detected if (i) the type of the EE is the same as in the gold standard and (ii) it occurs at the same position in the string, i.e., between the same surface words. This metric is, in fact, an extension of the standard PARSEVAL measure (Harrison *et al.* 1991, Magerman 1995). We report both precision and recall as well as the F-score, where:

$$\begin{aligned} \text{Precision} &= \frac{\text{true positives}}{\text{true positives} + \text{false positives}} = \frac{\text{true positives}}{\text{hypothesised EEs}} \\ \text{Recall} &= \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} = \frac{\text{true positives}}{\text{EEs in gold standard}} \\ F\text{-score} &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned} \tag{4.1}$$

Johnson (2002) remains unclear how the positions in the surface strings are defined as far as punctuation is concerned. When reporting PARSEVAL scores, the punctuation marks are discarded; in the present dissertation, we follow this practice. Thus, for example, in the sentence:

$$\begin{array}{l} \overset{1}{\textit{It}}\ \overset{2}{\textit{is}}\ \textit{,}\ \overset{3}{\textit{therefore}}\ \textit{,}\ \overset{4}{\textit{difficult}}\ \boxed{\textbf{PRO–NP}}\ \overset{5}{\textit{to}}\ \overset{6}{\textit{explain}}\ \overset{7}{\textit{what}}\ \overset{8}{\textit{I}}\ \overset{9}{\textit{want}}\ \overset{10}{\textit{to}} \\ \boxed{\textbf{NP–NP}}\ \overset{11}{\textit{do}}\ \boxed{\textbf{WH–NP}}\ \textit{.} \end{array} \tag{4.2}$$

the three EEs occur before words 5, 11 and 12.

An important difference between Johnson's (2002) metric and the one we use here is that we make a distinction between controlled and uncontrolled **PRO**s (**NP–NP** and **PRO–NP**, respectively). In fact, telling these two EE-types apart proves to be difficult. As a consequence, our metric is considerably harder than Johnson's and the scores are not fully comparable. However, when relevant, we also report our results according to the more permissive metric.

Finally, we also report the precision/recall/F-score separately for the semantically interesting group of EEs which consists of (controlled and uncontrolled) **PRO**s (**NP–NP** and **PRO–NP**), WH-traces (**WH–...**) as well as topicalised sentences (**TOP–S**). The scores for this group determine how good the system is at detecting

EEs involved in NLDs important for constructing predicate–argument structure. Unfortunately, Johnson (2002) does not report results for these cases separately. Note, however, that we know both the precision and the recall of his algorithm for a given EE-type, and we also know how many such EEs occur in the gold standard. It is easy to see that this enables us to determine the exact number of true positives according to his algorithm as well as the number of EEs it hypothesises. If P and R are the precision and the recall for a given EE-type, it follows from Equations (4.1) that:

$$\text{true positives} = R \cdot \text{EEs in gold standard} \tag{4.3}$$
$$\text{hypothesised EEs} = \frac{\text{true positives}}{P}$$

These formulae allow us to determine how Johnson's (2002) algorithm performs on groups of EEs. Whenever we use this trick to recover scores, we mark them with a †.

Apart from getting an overall score for the group of **WH/PRO/TOP** EEs, we rely on this trick in one further case: Johnson (2002) distinguishes **COMP–SBAR**s when they are immediately followed by an empty **TOP–S** label and other occurrences of the label **COMP–SBAR**, and reports detection scores separately for them. Since we do not see a motivation for this decision, we keep the two cases together and report only one score for them using the trick above to convert Johnson's (2002) scores.

Recovering antecedents

The metric for antecedent recovery is very similar to the one we employ for evaluating EE detection. Specifically, we say that the antecedent of an EE is correctly recovered, if and only if the following three conditions hold:

(i) the EE has the same type as in the gold standard;

(ii) the EE is at the same string position as in the gold standard (disregarding punctuation);

(iii) provided the EE-type in question takes an antecedent, the antecedent spans the same words as in the gold standard.

This metric is very stringent and requires a system to both recover antecedents and build a largely correct parse tree. The first two conditions are the same as for EE detection; therefore, if an EE is not (correctly) detected, the system is penalised. The last condition requires finding the antecedent phrase correctly, including all its modifiers. For instance, in the case of the sentence:

$$\begin{array}{l} \overset{1}{\textit{It}}\ \overset{2}{\textit{is}}\ \overset{3}{\textit{,}}\ \overset{4}{\textit{therefore}}\ \textit{,}\ \textit{difficult}\ \boxed{\textbf{PRO–NP}}\ \overset{5}{\textit{to}}\ \overset{6}{\textit{explain}}\ \overset{7}{\textit{what}}\ \overset{8}{\textit{the}}\ \overset{9}{\textit{man}}\ \overset{10}{\textit{in}}\ \overset{11}{\textit{a}} \\ \overset{12}{\textit{nice}}\ \overset{13}{\textit{blue}}\ \overset{14}{\textit{shirt}}\ \overset{15}{\textit{with}}\ \overset{16}{\textit{flowers}}\ \overset{17}{\textit{on}}\ \overset{18}{\textit{it}}\ \overset{19}{\textit{wants}}\ \overset{20}{\textit{to}}\ \boxed{\textbf{NP–NP}}\ \overset{21}{\textit{do}}\ \boxed{\textbf{WH–NP}}\ \textit{.} \end{array} \tag{4.4}$$

the system has to return

(i) an EE of type **PRO–NP** preceding word 5, with no antecedent;

(ii) an EE of type **NP–NP** preceding word 21, with antecedent spanning $[8, 18]$;

(iii) an EE of type **WH–NP** preceding word 22, with antecedent spanning $[7, 7]$.

Observe that Johnson (2002) does make a distinction between controlled and uncontrolled **PRO**s when evaluating antecedent recovery, since the latter does not have an antecedent while the former does. Therefore our results are straightforwardly comparable with (Johnson 2002). As before, we also report scores for the semantically important group of **WH–...**, **PRO–NP**, **NP–NP**, and **TOP–S** EEs; we mark results inferred using Eq. (4.3) with †.

Finally, note that although in the present thesis we do not attempt to recover pseudo-attachments, they are included in the two metrics discussed above; the recall (and the F-score) of the system on these items is simply 0%. Since pseudo-attachments amount to around 4% of all EEs, the maximal score we can achieve on both metrics is 96% recall ($\approx$ 98% F-score).

Dependency relations

The evaluation metric for antecedent recovery is both too crude and too stringent at the same time. On the one hand, it requires the parser to recover the whole dependent phrase without any mistakes, properly attaching all the modifiers. On the other hand, it takes only the string position of an EE into account. Although the position of an EE is a good indicator of the head, it is possible that the EE gets attached to the wrong head eventually. Since the main motivation for detecting EEs and recovering antecedents is to establish head–dependent relations involving non-local phenomena, we also evaluate our systems according to this criterion. Therefore, we adopt the evaluation metric proposed by Buchholz (2002): a non-local head–dependent relation is correctly recognised if and only if

(i) the type of the EE is correctly identified;

(ii) the EE is attached to the correct headword; and

(iii) the *head* of the dependent is the correct word.

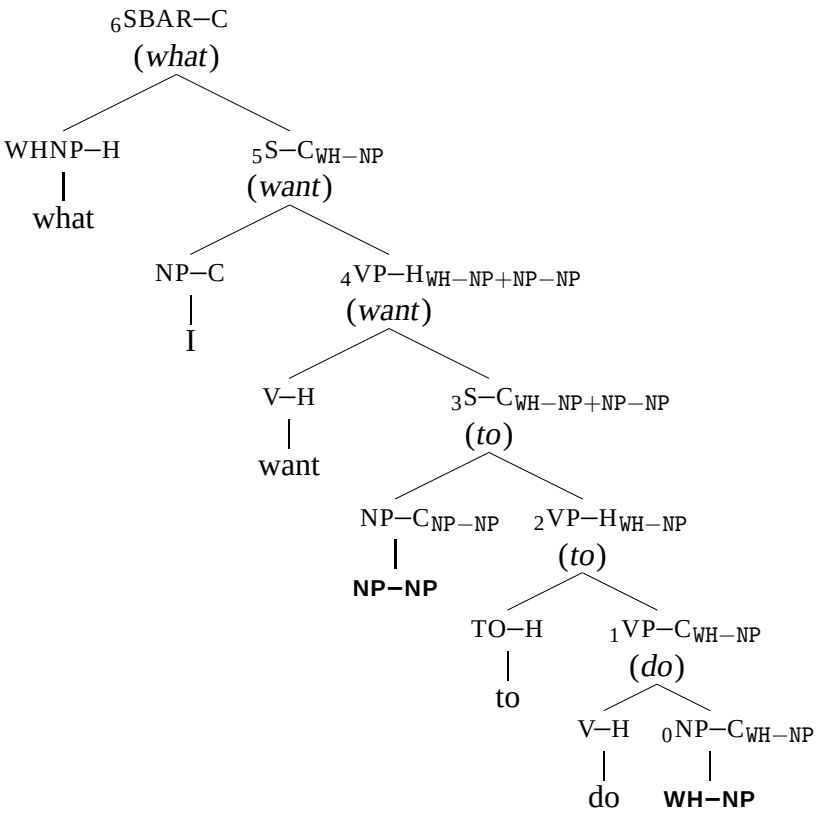


Figure 4.1: A head-lexicalised tree in the Tracebank [Figure 2.10].

This metric does not focus on string positions but evaluates dependency structures; we refer to it as the head–dependent (HD) metric.

In order to successfully employ this metric, however, one has to determine heads in the parser output. To identify heads, we apply the head-finding algorithm discussed in Section 2.3. One drawback of this algorithm in the present situation is that it is designed to retrieve surface heads only. Consider, for example, Figure 2.10 (repeated here as Figure 4.1): here, the infinitival *to* is marked as the head of the embedded sentence. Such a definition of headness is useful when defining a lexicalised probability model: clearly, *want* shows stronger preference for *to* than for *do*. On the other hand, we are recovering NLDs in order to be able to build the predicate–argument structure of the sentence. This representation requires the knowledge of deep heads (the predicates). For instance, in the case of the above example, the head of the empty element **NP–NP** should be *do* instead of *to*.

Finding deep heads, in general, might be complicated. Finding the deep head associated with WH-traces, topicalisation (**TOP–S**) and **PRO**s (**NP–NP** and

```
1  proc find_deep_head(node) ≡
2
3    Return the head if this is not an S or VP
4    if node.nonterm ≠ S ∧ node.nonterm ≠ VP
5      return node.headword
6
7    Find leftmost VP
8     var vp = find_leftmost_vp(node.children)
9
10   if vp then return find_deep_head(vp)    Recursive call
11         else return node.headword    Innermost VP
```

Figure 4.2: The algorithm for finding deep heads.

PRO–NP), however, is simpler, since it is always the main verb. Therefore, we restrict the use of the HD metric to these cases.

The recursive algorithm in Figure 4.2 returns the deep verbal heads starting from a node. The main idea is simple: the deep head of a phrase which is not a S or VP is the same as the surface head (line 4). In the case of S or VP nonterminals, we have to recurse into the deepest VP via a chain of VPs. The innermost VP dominates the deep headword (cf. also Eisner 1996a).

So far, we have only discussed evaluation metrics which assessed the accuracy of finding and the attachment of EEs and their antecedents. In order to determine how the parser performs on building the global structure for the sentence, we also report on a measure evaluating the accuracy of the parser with respect to surface dependencies (Lin 1995, Carroll *et al.* 1998, Collins 1999, Carroll *et al.* 2002). An unlabelled surface dependency relation is defined as an ordered pair of the surface head and the dependent: $\langle w_h, w_d \rangle$. This metric operates on phrase structure trees and does not take NLDs into account. For instance, the surface dependencies extracted from the phrase structure tree in Figure 4.1 are the following:

$$\langle what,want \rangle, \langle want,I \rangle, \langle want,to \rangle, \langle to,do \rangle. \tag{4.5}$$

To detect the heads, we employ the same head-finding algorithm as we use in creating the Tracebank (see Section 2.3).

We calculate both precision and recall for surface dependencies, and report the F-score in the experiments presented throughout the dissertation. We also combine the HD and surface dependency scores to arrive at a global measure capturing both local and non-local dependencies; we provide the F-score according to this metric as well.

Finally, we also report labelled bracketing accuracy according to the standard PARSEVAL measure (Harrison *et al.* 1991).

4.2 The antecedent recovery algorithm

The main motivation for the introduction of gap+ variables is that they indicate a path from the EE to the antecedent. In the case of a non-binary-branching grammar, however, this path only determines the node immediately dominating the antecedent, but does not indicate which child the EE should be co-indexed with. Moreover, a node might contain several gap+ variables, which further complicates antecedent recovery, even in the case of perfect trees. This calls for a sophisticated algorithm to recover antecedents.

The algorithm, presented in Figure 4.3, is applied after the best parse has been selected. It works in a bottom-up fashion, and for each empty node the main recursive function find_antecedent is called separately (lines 1 and 2). At every call, the number of gap+ variables of type "gap" (e.g. WH-NP) are calculated for the parent par of the current node node (*p*; line 6) and for all the children (*ch*; line 7). If the parent has at least as many unresolved gap+ variables as its children, we conclude that the current EE is resolved further up in the tree and call the same algorithm for the parent (line 21). If, however, the parent has fewer unresolved gaps ($p < ch$), the antecedent of the EE is among the children. Thus the algorithm attempts to find this antecedent (lines 10–19). For an antecedent to be selected, the syntactic category must match, i.e., an **NP–NP** must resolve to a NP. The algorithm searches from left to right for a possible candidate, preferring non-adjuncts over adjuncts. The node found (if any) is returned as the antecedent for the EE. Finally, note that in line 8, we have to remove the threaded gap+ feature in order to avoid confusion if the same parent is visited again while resolving another EE (Dienes and Dubey 2003a).

Topicalised sentences present an additional problem requiring a slight complication of the algorithm (line 17). Generally, they behave in the same way as any other empty element. However, as illustrated in Figure 4.4, when a parenthetical contains an extracted S, the PTB-annotation assigns the topmost S node as the antecedent. In such cases, i.e., when we have to drop a TOP-S gap variable and the parent is a S and there is no child bearing the label S, the antecedent is taken to be the parent node.

As an example, consider again the tree in Figure 4.1: how is the antecedent for **WH–NP** found? The algorithm starts with node 0 (gap in line 2 of the algorithm). The parent of this node (node 1) contains one gap+ variable on the parent and one on the child as well. That is, $p = ch$, and thus we conclude that the antecedent is not among the children of this parent. Therefore, the WH-NP gap variable is re-

```
1  foreach gap
2    do find_antecedent("gap", gap);
3
4  proc find_antecedent("gap", node) ≡
5    var par = node.parent;
6      p = number of gap+ features of type "gap" on par;
7      ch = sum of the gap+ features of type "gap" on par.children;
8      node.remove_one_gap("gap");
9      if p < ch
10       then Drop the gap here
11            ante = leftmost non-adjunct of par.children
12                   allowed by "gap"(≠ node);
13            return ante if ante;
14            ante = leftmost child of par.children
15                   allowed by "gap"(≠ node);
16            return ante if ante;
17            if "gap" == TOP−S ∧ par.nonterm == S
18              then return par;
19            return nil;
20       else Pass up the tree recursively
21            find_antecedent("gap", par).
```

Figure 4.3: The antecedent recovery algorithm.

"Now ," says Joseph Napolitan, a pioneer in political television, "the idea is to attack first, last and always."

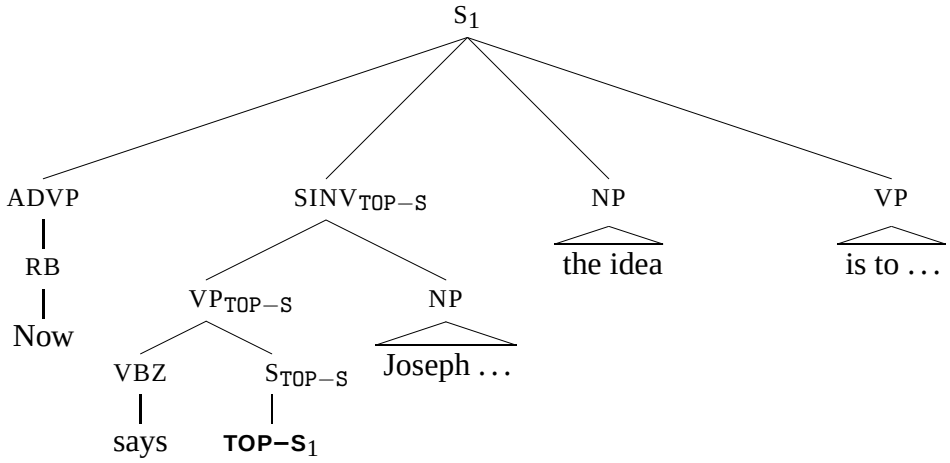


Figure 4.4: Additional problems with **TOP−S**.

moved from node 0 and the function is called recursively on the parent (line 21). Now, the WH-NP variables on node 2 and node 1 are compared – they are still the same, therefore the WH-NP is removed from node 1 and the algorithm is called on node 2, and so on. However, at node 5, the parent (node 6) does not contain any WH-NP variables, therefore the gap has to be dropped and we enter the search in lines 10–19. In this case, there is no additional complication, and the antecedent is found: *what.*

Since the algorithm works in a greedy fashion accepting the first possible antecedent for an EE, the order in which we loop through the gaps in lines 1 and 2 is very important. Consider, for instance, the parse tree depicted in Figure 4.5, the partial structure of the sentence:

> *This is a problem which*$_1$ *John*$_2$ *is difficult to talk to* __$_2$ *about* __$_1$. (4.6)
> (Pollard and Sag 1994, p. 159)

If the traces are ordered left to right, the algorithm first takes **WH–NP**$_A$, the complement of *to.* The first node where an EE should be dropped is the SBAR node marked with the subscript 1: here the number of WH-NP variables on the parent (1) is smaller than on its children (2). Since the algorithm now carries **WH–NP**$_A$, it finds its antecedent there, i.e., **WH–NP**$_A$ correctly gets co-indexed with the empty WH-operator **COMP–WHNP**. In the following loop, the algorithm takes **WH–NP**$_B$ and carries it up the tree. Now, since the previous run removed the WH-NP variables corresponding to the **WH–NP**$_A$ trace, at the SBAR position marked as 1, the number of WH-NP variables is the same on the parent as on the children (1). Consequently, the algorithm keeps on carrying the EE higher up in the tree, until it reaches the SBAR node marked as 2. Here, it has to drop the EE, and thus correctly finds the antecedent: *which.*

Thus, the algorithm, processing EEs in right-to-left order, is capable of recovering antecedents in a serial fashion. On the other hand, if the direction is changed and the algorithm starts with the other gap **WH–NP**$_B$, it associates the antecedents with the EEs in the opposite order. This mode illustrates the cross-serial dependencies familiar from Dutch (cf. e.g. Rambow and Joshi 1994, Steedman 2000). Since English does not allow cross-serial dependencies, we set the order left-to-right. It might be interesting to note, however, that in the development section (WSJ00) we have not found any instances where the order plays a role.

Finally, in Figure 4.6, we illustrate how the antecedent recovery algorithm handles parasitic gaps (Appendix A.1.1, page 181), and, as a matter of fact, coordination. The key insight is that more than one EE can be dropped at a given node. Specifically, at the SBAR node marked with the subscript 2, both **WH–NP** gaps are to be dropped and hence they share the same antecedent: **COMP–WHNP** dominated by the NP *papers.* Note that the algorithm also correctly identifies *you* as the sub-

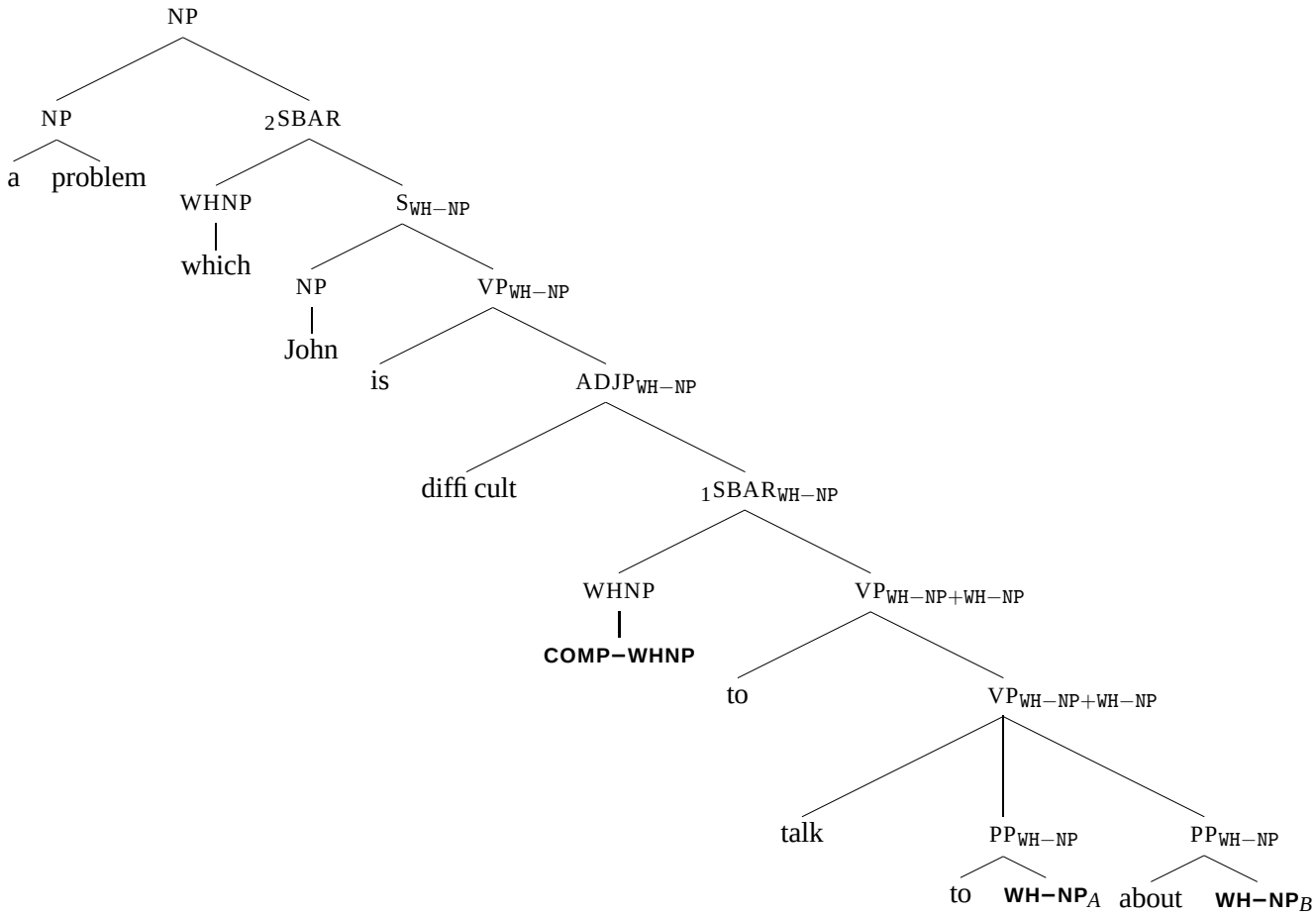


Figure 4.5: The importance of the order in which the gaps are looped through in the antecedent recovery algorithm.

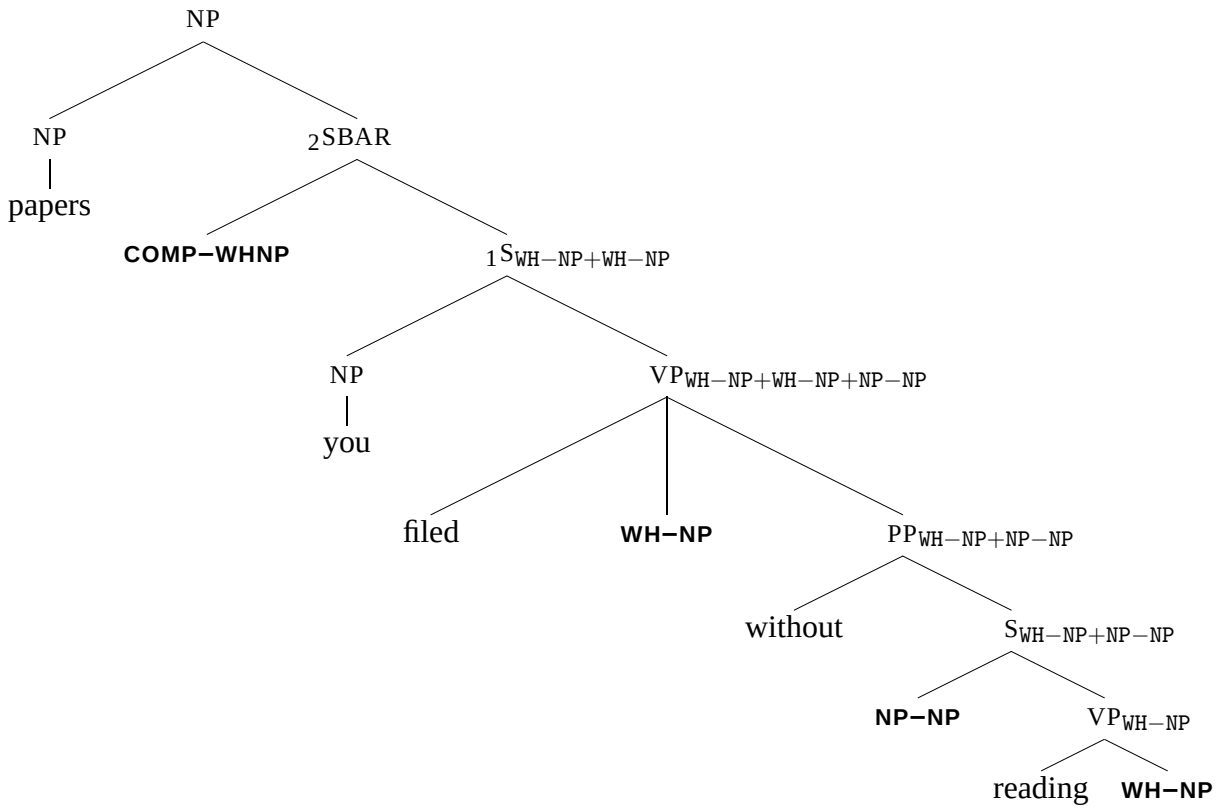


Figure 4.6: Handling parasitic gaps/coordination.

ject of the embedded gerundial clause by dropping the **NP–NP** gap at the S node marked with 1.

Although the algorithm is simple and works in a greedy manner, it is very accurate. Tested on the gold standard Tracebank trees containing the empty nodes without co-indexation, it is able to recover the antecedents with an F-score of 96.2%, whereas the HD score for **WH**, **PRO** and **TOP** gaps is 99.5%. Note that the algorithm crucially hinges on the assumption that the antecedent strictly c-commands its EE, that is, the algorithm never has to descend into a daughter to find the antecedent (Section 2.3). Although this assumption generally holds, it is not appropriate in the case of pseudo-attachments, where the algorithm gives up and (wrongly) assumes they have no antecedent. Fortunately, they only account for 3.8% of all EEs.

4.3 Experiments with an unlexicalised parser

Having discussed how to recover the antecedents once we have a full parse tree which contains EEs and follows the gap-threading schema introduced in Section 2.3, in the remaining of the chapter, we present experiments with two unlexicalised PCFG parsers following (Dienes and Dubey 2003b).

4.3.1 The unlexicalised parsers

The INSERT model

There are two important modifications a simple unlexicalised PCFG parser requires so that it can insert EEs and recover antecedents: (i) it has to be able to thread gaps, and (ii) it should insert EEs. Both modifications are trivial. As for the first one, we have to change the original nonterminal symbols to contain the gap-variables as well. The second point requires the extension of the CYK-algorithm so that it could handle unary rules as well.

We use a standard unlexicalised PCFG-parser (Charniak 1993), where the PCFG probabilities are estimated in the usual way:

$$p(\text{LHS} \rightarrow \text{RHS}|\text{LHS}) = \frac{f(\text{LHS} \rightarrow \text{RHS})}{f(\text{LHS})} \tag{4.7}$$

where $f(x)$ is the frequency count of x in the corpus. The parser uses dynamic programming (Viterbi-search) to arrive at the most probable parse tree given the input. Since even with Viterbi-search the search space is too large to allow efficient parsing, we augment the parser with beam thresholding (Goodman 1998):

subtrees for a span with low relative probability with respect to the highest probability subtree for the given span are discarded from the search space. In order to keep the number of independently tunable parameters low, no smoothing is applied to the model. We refer to this model as the INSERT model.

Note that introducing unary rules that expand a nonterminal into exactly one nonterminal might lead to an infinite loop. For instance, the rule

$$\text{VP} \rightarrow \text{VP} \ (\textbf{PRO–NP}) \tag{4.8}$$

could be applied on its own output, thus forcing the parser into an infinite loop. This would indeed be a problem for a non-probabilistic parser. Viterbi-search, on the other hand, ensures that a probabilistic parser does not suffer from the same problem, provided the probabilities are obtained from a corpus. We prove this statement informally here; for a formal proof we refer the reader to (Chi and Geman 1998).

Infinite loops occur when there is a cyclic nonterminal sequence in the derivation which does not consume any nonterminals. Without a loss of generality, we can assume that this sequence is in the following form (each A_i is a nonterminal):

$$\text{A}_1 \rightarrow \text{A}_2 \rightarrow \ldots \rightarrow \text{A}_k \rightarrow \text{A}_1 \tag{4.9}$$

A probabilistic CYK-parser falls into an infinite loop, if and only if $p(\text{A}_1 \stackrel{*}{\rightarrow} \text{A}_1|\text{A}_1) = 1$, where $\stackrel{*}{\rightarrow}$ is the transitive closure of the rewrite operator $\rightarrow$. However,

$$\begin{aligned} 1 &= p(\text{A}_1 \stackrel{*}{\rightarrow} \text{A}_1|\text{A}_1) \\ &= p(\text{A}_1 \rightarrow \text{A}_2|\text{A}_1) \cdot p(\text{A}_2 \rightarrow \text{A}_3|\text{A}_2) \cdot \ldots \cdot p(\text{A}_k \rightarrow \text{A}_1|\text{A}_k) \end{aligned} \tag{4.10}$$

holds if and only if each term

$$p(\text{A}_i \rightarrow \text{A}_{i+1}|\text{A}_i) = 1 = \frac{f(\text{A}_i \rightarrow \text{A}_{i+1})}{f(\text{A}_i)} \tag{4.11}$$

for all $i = 1 \ldots k$ (with $\text{A}_{k+1} = \text{A}_1$). This means that a potential infinite loop has probability 1 if and only if all nonterminal symbols that appear in it does not occur in the LHS of any other rule; that is, once we enter a loop there is no possibility to escape it. Our corpus, however, consists of finite strings; consequently, either (i) there are no infinite loops at all or (ii) there is an "escape" from the loop, i.e., there exists a rule $\text{A}_j \rightarrow \beta$ with nonzero frequency count, where β is a non-empty sequence of terminals and/or non-terminals, $j \in \{1, \ldots, k\}$ and $\text{A}_{j+1} \neq \beta$. This

entails that

$$\begin{aligned} p(\mathrm{A}_j \rightarrow \mathrm{A}_{j+1}|\mathrm{A}) &= \frac{f(\mathrm{A}_j \rightarrow \mathrm{A}_{j+1})}{f(\mathrm{A}_j)} \\ &\leq \frac{f(\mathrm{A}_j \rightarrow \mathrm{A}_{j+1})}{f(\mathrm{A}_j \rightarrow \mathrm{A}_{j+1}) + f(\mathrm{A}_j \rightarrow \beta)} \\ &< \frac{f(\mathrm{A}_j \rightarrow \mathrm{A}_{j+1})}{f(\mathrm{A}_j \rightarrow \mathrm{A}_{j+1})} = 1 \end{aligned} \tag{4.12}$$

showing that $p(\mathrm{A}_j \rightarrow \mathrm{A}_{j+1}|\mathrm{A}) < 1$, an consequently, $p(\mathrm{A}_1 \stackrel{*}{\rightarrow} \mathrm{A}_1|\mathrm{A}_1) < 1$. As a consequence, Viterbi-search prevents the parser from falling into an infinite loop; it will only create one edge with nonterminal label A_1 for the span, since all other such edges have lower probability.

The PERFECT model

As we will see it shortly, the INSERT model cannot cope with the task of recovering NLDs. It is interesting to ask the question: which of the two subtasks are difficult, EE detection or antecedent recovery? In order to answer this question, we test the unlexicalised PCFG parser under another condition, where the parser is not required to detect EEs: the correct EEs are supplied in the input (as separate words). The task of the parser, in this case, is to build the correct structure, incorporating the EEs, and thread the `gap+` variables to their antecedents. This model is called the PERFECT model, since it uses the perfect information about the sites where EEs occur.

4.3.2 Results

The main results for the two conditions are summarised in Table 4.1. The most striking observation is the infeasibility of the INSERT model: with the beam set to 10000, the parser cannot parse 40% of the sentences, whereas successful parses take, on average, 54 seconds[2] and enumerate some 2.3 million edges per sentence. Widening the beam to 40000 decreases the number of missed sentences marginally, whereas parsing time rises to nearly 5 minutes per sentence. Even with the large amount of search, the parser is not able to attain acceptable scores when it comes to detecting NLDs. When we restrict our attention to the sentences the parser eventually manages to parse, the F-score for EE detection is 44.3% (precision 57.6%, recall 36.0%). Antecedent recovery scores are low accordingly: precision 48.9%, recall 30.6%, and F-score 37.7%.

[2]This particular parser is around 3 times slower than the one used by Dienes and Dubey (2003b). We are, however, interested in *relative* parsing times and not absolute ones.

		NOTRACE	INSERT	PERFECT
EE det.	all	–	44.3*	100
(F-score)	WH/PRO/TOP	–	47.1*	100
ANTE rec.	all	–	37.7*	89.7
(F-score)	WH/PRO/TOP	–	38.0*	89.5
HD accuracy	WH/PRO/TOP	–	42.7*	86.3
Time (s/sent)		9.3	54.8	8.5
Chart (edges/sent)		368k	2376k	327k
PARSEVAL	≤ 100	72.2	75.4*	78.0
(F-score)	≤ 40	73.7	75.8*	79.3
Surface deps.		79.3	81.8*	82.5
All deps.		77.1	80.5*	82.7
Missed sents.		0.3%	39.9%	0.5%

Table 4.1: Overall performance of the unlexicalised INSERT and PERFECT models.

In the PERFECT case, when the sites of the empty elements are known before parsing, only about 0.5% of the sentences are missed and average parsing time goes down to 8.5 seconds per sentence. More importantly, the overall precision and recall for antecedent recovery is 89.7%[3], whereas the HD score for the semantically interesting cases is 86.3%.

4.3.3 Discussion

The result of the experiment where the parser is to detect non-local dependencies is negative. Detecting EEs with an unlexicalised PCFG parser is infeasible: the parser cannot parse a large fraction of the sentences, it is incredibly slow and inaccurate at detecting EEs and at antecedent recovery. On the other hand, the PERFECT model does not show the same problems at all: it is fast and accurate at antecedent recovery: it achieves 89.7% F-score (86.3% HD score). What is the reason for such a difference between the two parsing models?

[3]Dienes and Dubey (2003b) achieve a 91.4% F-score, but they fail to penalise the parser for not handling pseudo attachments.

*These results are reported on sentences the parser could parse.

The reason cannot be due to the lack of smoothing: the model with perfect information about the EE-sites does not run into the same problem. Consequently, the edges necessary to construct the required parse are available. Therefore, in the INSERT case, it is the beam search that loses necessary edges due to unwanted local edges having a higher probability. Doing an exhaustive search might help in principle, but it is infeasible in practice. Clearly, the problem is with the probability model: an unlexicalised PCFG parser is not able to detect where EEs can occur, hence necessary edges get low probability and are, thus, filtered out (Dienes and Dubey 2003b).

Another factor contributing to the inefficiency of the INSERT model is the larger size of the non-terminal set. Recall that a CYK-parser has a worst-case asymptotic runtime of $\mathcal{O}(n^3|V_T|^3)$, where $|V_T|$ is the number of nonterminals (which are not preterminals). The $\mathcal{O}(|V_T|^3)$ bound becomes important when the parser inserts traces because there are 7.1 times more non-terminals than in the case of a non-threading model. Three factors contribute to this larger nonterminal set:

(i) nonterminals are augmented with gap+ information encoding the type of the EE (i.e., S may become $\mathrm{S}_{\mathtt{WH-NP}}$, $\mathrm{S}_{\mathtt{NP-NP}}$, etc.);

(ii) we must include combinations of EEs as nonterminals may dominate more than one pending EE (i.e. $\mathrm{S}_{\mathtt{NP-NP+WH-NP}}$) and

(iii) a single gap+ variable may be repeated in the presence of co-ordination (i.e. $\mathrm{S}_{\mathtt{NP-NP+NP-NP}}$).

These three factors greatly increase the number of nonterminals, potentially reducing the efficiency of a parser that detects EEs. On the other hand, when EE-sites are pre-determined, the effect of the number of nonterminals on parsing speed is moot: the parser can ignore large parts of the grammar.

Interestingly, although the number of nonterminals is much higher when EEs are threaded to their antecedents, the number of rules increases by only 22%. This shows that the common practice of removing EEs before training also gives rise to complicated rules. It turns out, that the difference is in fact due to the larger number of different RHS in the case of the EE-threading approach.

The most important conclusion of the experiments with the unlexicalised parser is that antecedent recovery is relatively easy (in English) once we know where (and whether) EEs occur: the PERFECT model achieves almost 90% F-score on recovering antecedents and 86.3% F-score on the HD metric for the semantically interesting EEs. Table 4.2 gives scores broken down according to EE-types. It reveals that most of the cases are indeed easy for the parser, although there are some difficult ones with low scores.

EE	Frequency	Ante. rec.[4]	HD score[4]
ALL	3864	89.7	86.0
WH/PRO/TOP	2634	89.5	86.3
NP–NP	1148	81.7	79.4
COMP–SBAR	545	100.0	96.0
WH–NP	508	96.7	94.5
PRO–NP	477	100.0	92.9
UNIT	388	100.0	89.9
TOP–S	277	91.3	87.0
WH–ADVP	171	93.6	88.9
COMP–WHNP	107	100.0	100.0

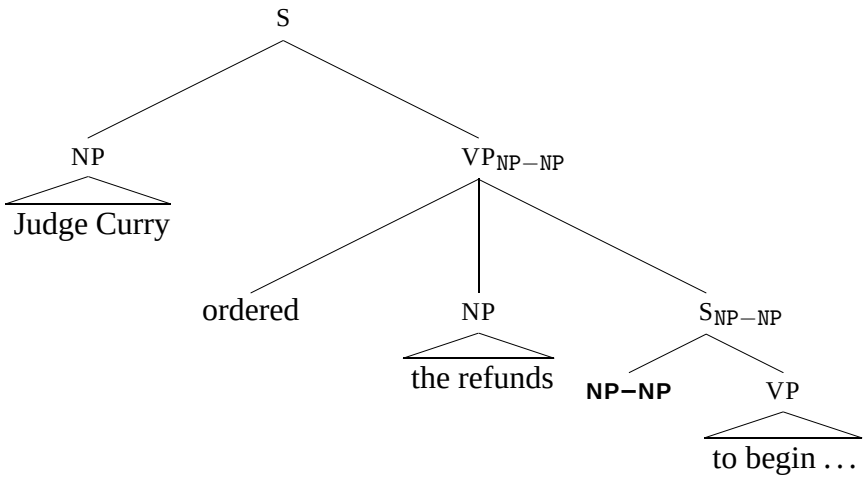
Table 4.2: Antecedent recovery and HD scores for the unlexicalised PERFECT model.

Most importantly, identifying the antecedents of **NP–NP** traces appears to be difficult: the parser achieves only 81.7% F-score on this task. There are several reasons why the parser is not so accurate finding the antecedent for this EE-type. First, its antecedent as often as not is a complex noun phrase with pre- and post-modification. Since the parser is not a very accurate one, it is highly likely to commit an error in identifying the whole NP. A similar behaviour can be observed for **TOP–S** traces as well, where the parser has to identify the whole subclause.

Another frequent type of errors concerning **NP–NP** involves control constructions: the parser is unable to correctly decide whether the antecedent is the object or the subject of the matrix verb (cf. Figure 4.7). As this decision depends solely on the matrix verb and the parser is not lexicalised, it has no means to distinguish the two constructions. Clearly, we would greatly benefit from lexicalisation, an approach we will explore in Section 6.2 and Chapter 7.

A similar situation arises in the case of **PRO–NP** traces as well: although the parser knows in advance where these EEs occur in the input, it often cannot attach them properly, as is indicated by the low HD score for this EE-type. Frequently, the problem is due to verbs in reduced relative constructions taking a small-clause argument (i.e., a clause without a verb, Figure 4.8): the parser, having no access to lexical information, wrongly assumes that the empty **PRO–NP** is an object of the verb (*named*, in this case), instead attaching it to the head of the small clause.

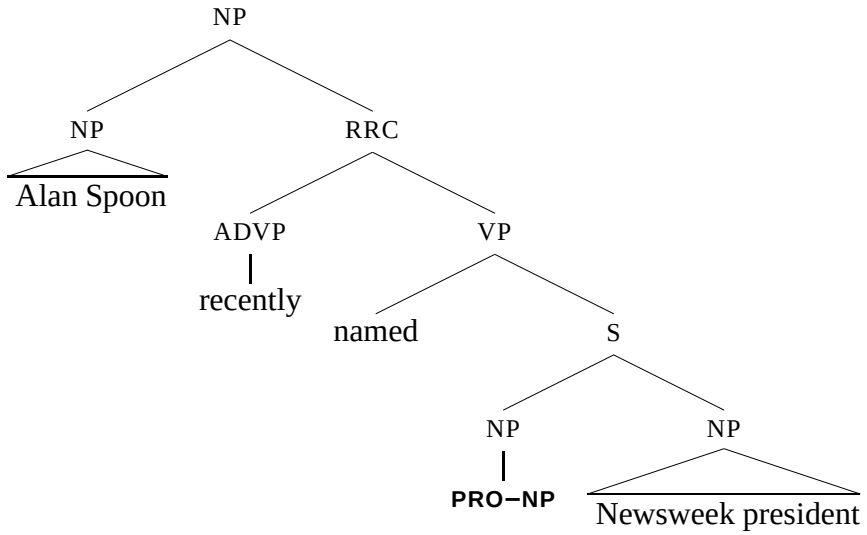
[4]Since the parser does not have to detect the EEs in this case, the precision, the recall, and the F-score are the same.



Judge Curry ordered the refunds to begin Feb. 1 and said that he wouldn't entertain any appeals or other attempts to block his order by Commonwealth Edison.

(wsj_0015.mrg)

Figure 4.7: A typical error the unlexicalised parser commits: proposing subject control instead of object control.



Alan Spoon, recently named Newsweek president, said Newsweek's ad rates would increase 5% in January. (wsj_0012.mrg)

Figure 4.8: A typical error the unlexicalised parser commits: misattachment of empty subjects in small clauses.

Finally, observe the improvement in PARSEVAL scores in the case of models handling EEs. As it turns out, incorporating NLDs into the parsing model greatly increases its accuracy, despite the higher ambiguity rate of the grammar. This shows that keeping the EEs is beneficial for retrieving not only predicate–argument but also phrase structure.

As the experiment with the INSERT model has shown, however, the unlexicalised parser is not able to both detect EEs and recover their antecedents, the bottleneck being detecting the sites. The parser would greatly benefit from the external knowledge where EEs occur in its input. In the following chapter we explore whether we can perform this task prior to parsing, i.e., in a preprocessing step using no explicit phrase structure information.

4.4 Related work

The literature on parsing with empty elements is very limited. Robert C. Moore (p.c.) confirms our experience with the INSERT model: he also observed a similar behaviour. Johnson and Kay (1994) discuss the problem of large (even infinite) search space, due to the possibly unbounded number of empty elements. They propose to limit this number using lexical information; they argue that the presence of EEs is always triggered by a lexical head. Although such a move would clearly restrict the search space, Johnson and Kay (1994) do not present any practical evaluation how successful their method is. Note that our INSERT model does not suffer from the problem of the unbounded number of EEs (the parser cannot fall into an infinite loop), still it is inefficient.

Non-local dependencies can be viewed as discontinuous structures. Plaehn (1999, 2000) reports similar efficiency problems with parsing with his unlexicalised probabilistic Discontinuous Phrase Structure Grammar.

Summary

This chapter centred around three main themes. First, we discussed various evaluation metrics we use throughout the dissertation. Second, we presented an algorithm which takes trees in the Tracebank format and recovers antecedents, i.e., it approximates the co-indexation scheme of the original PTB. Finally, we developed an unlexicalised PCFG parser which is able to insert EEs. This parser turned out to be inaccurate and inefficient. Interestingly, when the same parser is informed about the sites where EEs occur, it becomes fast and reliable: it recovers antecedents with 89.7% F-score (86.3% HD score). This shows that finding the

antecedent for a given EE is easy: even the simplest parsing model, an otherwise relatively inaccurate unlexicalised PCFG, can perform this task fairly reliably.

Chapter 5

The trace tagger

In the previous chapter, we have shown that even an unlexicalised PCFG parser does considerably well at detecting NLDs provided it is reliably informed about the sites where such NLDs occur. In this chapter, we address the question whether finding the sites for NLDs is possible prior to parsing, i.e., without explicit information about phrase structure.

The answer turns out to be positive, further supporting the claim that finding the majority of NLDs in English is not that difficult. Specifically, we propose a tagger, the *trace tagger*, that is able to find where EEs occur in the surface string with state-of-the-art accuracy. The tagger is a probabilistic finite-state transducer, returning the highest probability EE-sequence for each input sentence. The underlying probability model is a conditional maximum entropy model using both local and long-distance features.

This chapter[1] first shows how finding the sites for NLDs can be cast as a tagging problem. Then, we introduce the probability model and the contextual features employed therein. In the discussion, we analyse the performance of the tagger and the errors it makes, determine which features prove to be helpful for the task, and compare our results to previous work on the same task.

5.1 The tagger

Detecting empty elements can be regarded as a simple tagging task: we label words according to the existence and type of empty elements preceding them. For example, the word *Sasha* in the sentence

Sam said **COMP–SBAR** *Sasha snores.* (5.1)

[1]This chapter is an extension of Section 4 of (Dienes and Dubey 2003b). In particular, we use a different search method in the tagger and give detailed error and feature analyses.

gets the label EE=**COMP–SBAR**, whereas the word *Sam* is labelled with EE=* expressing the lack of an EE immediately preceding it. If a word is preceded by more than one EE, such as *to* in the following example, it is labelled with the concatenation of the two EEs, i.e., with EE=**COMP–WHNP_PRO–NP**:

> *It would have been too late* **COMP–WHNP PRO–NP** *to think about on Friday.* (5.2)

This task is very similar to named entity recognition (NER), where the problem is to detect whether a NP is a named entity and to classify named entities into four classes: persons, locations, organisations, and names not belonging to the three previous classes. NER is often viewed as a tagging task: NPs are tagged with one of five labels: four representing the four classes and the fifth reserved for NPs which are not named entities (Bikel *et al.* 1999, Tjong Kim Sang 2002, Tjong Kim Sang and de Meulder 2003).

Despite the similarities, there are certain differences which might make our task more difficult than NER. One of them is the extreme sparseness of the data. First, the tagset size is larger than for NER: disregarding the complex, concatenated tags, there are 44 different EE-types, as opposed to the 5 NER-tags. However, with the concatenated tags representing adjacent EEs, the cardinality of the tagset rises to 162. The number of the tags in itself would be no problem, since taggers for agglutinative languages manage tagsets of a magnitude higher cardinality (cf. e.g. Brants 1995, Hajič and Hladka 1998, Erjavec *et al.* 1999, Dienes and Oravecz 2000, Hajič 2000, Tufiş *et al.* 2000, Oravecz and Dienes 2002). The real problem is that more than 100 of the 162 tags occur less than 10 times in the entire training corpus of 1M words. In NER, on the other hand, even the least frequent tag occurs 5800 times in a corpus of approximately 255k words (Tjong Kim Sang and de Meulder 2003).

A further complication is the sparseness of the EEs themselves: in our development corpus WSJ00, out of the 46451 tokens only 3056 are preceded by one or more EEs. That is, approximately 93.5% of the words are labelled with the EE=* label. As a comparison, in the NER setting, only 83% of the words get the NONE tag (Tjong Kim Sang and de Meulder 2003).[2] Moreover, most of the EE-labels are adjacent to EE=* tags, i.e., the previous label does not help predicting the next one. These circumstances call for a conditional model. Indeed, a generative model using the EE-tags as hidden states would be in essence a unigram model, where the emission probabilities $P(word|\text{EE-}tag)$ are not very informative. To overcome the problem, one might try to incorporate some contextual information into the labels themselves (cf. Osborne 2000b), but this approach is rather limited, since it further increases the sparse data problem.

[2] In fact, only words within a NP can be named entities, which decreases the search space considerably: only 73% of such words get the NONE tag.

Another important difference between NER and the present task is the apparent non-local nature of the problem. Cues such as the occurrence of a WH-word in the sentence are important in detecting EEs. This situation calls for a model which is able to combine not necessarily independent local and non-local features in a principled way.

In this chapter, we present a tagger designed to overcome the above difficulties. The tagger learns the distribution of EEs applying machine learning techniques. Specifically, it is based on a conditional maximum entropy model, similar to the one employed by Ratnaparkhi (1996) for part-of-speech tagging. In the remainder of this section, after a brief introduction to maximum entropy models, we present the details of the tagger.

5.1.1 Maximum entropy models

Maximum entropy (ME) and other related models, such as Conditional Random Fields (Lafferty *et al.* 2001), have been reported to be successful in many supervised NLP tasks, including sentence boundary detection (Reynar and Ratnaparkhi 1997), POS-tagging (Ratnaparkhi 1996), named entity recognition (Malouf 2002b, Florian *et al.* 2003), parsing (Ratnaparkhi 1997), parse selection (Johnson *et al.* 1999, Riezler *et al.* 2002, Toutanova *et al.* 2002, Baldridge and Osborne 2003), text classification (Nigam *et al.* 1999), text segmentation (McCallum *et al.* 2000), sentence extraction (Osborne 2002), machine translation (Och and Ney 2002), etc. Their popularity and success might be attributed to three main features that make ME models well-suited for NLP applications. Unlike many popular machine learning approaches, such as Memory-Based Learning (Aha *et al.* 1991), Support Vector Machines (Vapnik 1995, 1998) or Neural Nets (Rumelhart *et al.* 1986), ME models explicitly model the probability distribution generating the data. Furthermore, they offer a principled way to combine various, not necessarily independent sources of information. Finally, the principle of maximum entropy is very attractive: it prescribes us to use the model which (best) accounts for the seen data but does not make further assumptions about unseen events.

Maximum entropy models were introduced into Natural Language Processing by members of the IBM T.J. Watson Research Center in the early 1990s through a number of influential articles (Lau *et al.* 1993, Berger *et al.* 1996, Rosenfeld 1996, Della Pietra *et al.* 1997). In the presentation below, we closely follow (Berger *et al.* 1996).

Let us start our introduction to maximum entropy models by defining what we mean by supervised learning. We assume here that the phenomenon we want to learn is controlled by a random process (it might be a deterministic one, though). Our task is to find out what this random process is, i.e., to learn the underlying probability distribution p. In order to achieve this goal, we observe how this

process behaves by collecting a (large) number of samples: $(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)$. Here, x_i is the raw data and y_i is the (linguistic) structure associated with it. For example, x_i can be a sentence, and y_i is the correct parse tree. In our present setting, x_i is a sentence with a word marked as the target word, y_i is the EE-label of the target word. We call our set of samples the *training data*. The training data can be summarised by its empirical probability distribution $\tilde{p}$, which is defined as

$$\tilde{p}(x,y) = \frac{\text{number of times } (x,y) \text{ occurs in the sample}}{N} = \frac{f(x,y)}{N} \tag{5.3}$$

where $f(x,y)$ is the frequency count of (x,y) in the sample.

Recall that our task is to find the underlying probability distribution p that generates the sample, that is, we are interested is $p(x,y)$ for all possible x,y pairs. If our sample is very-very large (i.e., $N \to \infty$), the empirical distribution $\tilde{p}$ is the same as the true probability distribution p. Unfortunately, the amount of training data is very far from being large enough in most of the cases. In practice, we only see a small fraction of possible (x,y) pairs: typically, a particular pair does not occur in the sample at all (especially if x is taken to be a sentence) or occurs at most a few times (usually once). This problem is called *data sparseness*: we do not have enough data to estimate the true probability distribution. That is, the observed empirical distribution $\tilde{p}$ is very far from the underlying distribution p. Should we take the empirical distribution to predict the behaviour of the random process, we could not generalise to unseen cases: they have 0 probability according to $\tilde{p}$. That is, our model would *overfit* the training data.

In order to avoid these problems and be able to generalise from the sample, we decompose each data instance into smaller building blocks which, hopefully, show a less sparse behaviour. In order to achieve this goal, we assume that the data can be described in terms of (boolean) *indicator features* (or simply *features*) capturing relevant aspects of the data with respect to the task at hand. For instance, we might notice that **PRO–NP** EEs tend to precede infinitival *to*. In this case, we define a binary feature capturing this observation:

$$f(x,y) = \begin{cases} 1 & \text{if } y = \textbf{PRO–NP} \text{ and the target word is } \textit{to} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Similarly, we can define many other features that we conjecture to carry important information.

With the help of the features $f_1,\ldots,f_n$, we can describe those aspects of the data that are relevant for the task and discard the rest. In other words, indicator features impose a partition on the training data, where similar instances differing only in irrelevant aspects are assumed to be in the same partition. Although

our training data is too small to reliably estimate the whole distribution, it might be large enough to estimate the relevant aspects of the data.[3] That is, our small building blocks, the indicator features, occur often enough to be representative of their relevance, which is expressed by the expected value $\tilde{p}(f)$ of each feature f according to the empirical distribution $\tilde{p}$:

$$\tilde{p}(f_i) \stackrel{\text{def}}{=} \sum_{x,y} \tilde{p}(x,y)\, f_i(x,y) \tag{5.5}$$

For instance, we might find that the expected value of the feature in (5.4) indicating that **PRO–NP** occurs before *to* is higher than the expected value of the feature capturing the occurrence of the same EE before the word *house.*

Since we assume that the sample reasonably estimates the relevant aspects the data, the expected value of each feature f_i according to the underlying model p has to be the same as the expected value according to the empirical distribution:

$$\sum_{x,y} \tilde{p}(x,y)\, f_i(x,y) = \sum_{x,y} p(x,y)\, f_i(x,y) \tag{5.6}$$

This means that we can constrain our search: the true model can only be among those models that satisfy this condition (for each feature f_i).

Although this approach is very attractive, we face a serious problem: calculating the right-hand side of the above equation requires us to sum over all possible (x,y) pairs. In most NLP tasks, it is impossible to perform this summation. For instance, if x stands for sentences and y for a possible syntactic analysis, the calculation of the right hand side would require us to sum over all possible analyses of all possible sentences.[4]

The above constraint is, in fact, very stringent: it requires the model to best account for both the training sample and the data we did not observe. That is, we expect it to be an accurate model of the *joint* probability distribution $p(x,y)$. Fortunately, in NLP we are more often interested in the *conditional* distribution: $p(y|x)$. For instance, when we encounter a sentence, we would like to know the conditional probabilities of possible parses given the sentence in order to rank them, and we are rarely interested in the marginal distribution $p(x)$, i.e., in the probability of the sentence itself.[5] This allows us to relax the constraints: the un-

[3] In fact in many applications, this is not the case, and it is beneficial to apply smoothing to the resulting model (Chen and Rosenfeld 1999, Curran and Clark 2003). We discuss the smoothing technique we use in Section 5.1.3.

[4] Sampling might be helpful to overcome the problems (e.g. Abney 1997), although it might not be feasible in real systems (Johnson *et al.* 1999).

[5] The most notable exception is the noisy-channel model (Shannon 1948), where we are interested in the *language model* as well, i.e., in the marginal distribution. Johnson (2001) compares the accuracy of joint and conditional models in parsing and POS-tagging and finds that the joint model outperforms the conditional one, showing that marginal probabilities might still be important.

derlying distribution p should only account for the *seen* training data (Lau *et al.* 1993, Rosenfeld 1996):

$$\tilde{p}(f) \stackrel{\text{def}}{=} \sum_{x,y} \tilde{p}(x,y) f(x,y) = \sum_{x} \tilde{p}(x) \sum_{y} p(y|x) f(x,y) \stackrel{\text{def}}{=} p(f) \tag{5.7}$$

This constraint requires us to sum over all possible analyses for sentences in the training data which is much more feasible than summing over all possible sentences (although this might still be intractable, Osborne 2000a).

To recapitulate, we can reformulate the task of finding the underlying probability distribution p as follows. Suppose that we are given n features $f_1, \ldots, f_n$ which we assume to be relevant for describing the process. We are looking for a model p, which satisfies the constraint in Eq. (5.7) for each feature f_i. This model is a member of the following set of probability distributions:

$$\mathcal{C} = \{p \mid p(f_i) = \tilde{p}(f_i) \quad \forall i \in \{1,2,\ldots,n\}\} \tag{5.8}$$

This set is not empty ($\tilde{p} \in \mathcal{C}$); it might, however, be very large, even infinite. Which distribution should we chose from it?

If we have no prior knowledge about the distribution, it is reasonable to choose the *most uniform* one accounting for the training data, that is, the distribution which agrees with our observations but does not assume anything that is not known. For instance, if we observe that our features f_1 and f_2 behave in exactly the same way in our training data, there is no basis for selecting a distribution which predicts that they behave differently on unseen data points. This principle is the *maximum entropy principle*, which advises us to choose the least complex hypothesis that can account for the seen data.

Uniformity can be expressed in terms of conditional entropy, defined by

$$H(p) \stackrel{\text{def}}{=} -\sum_{x,y} \tilde{p}(x) p(y|x) \log p(y|x) \tag{5.9}$$

The most uniform distribution is the one that maximises the conditional entropy. That is, we search for the distribution $p^{\star}_{\text{ME}} \in \mathcal{C}$ with maximum entropy:

$$p^{\star}_{\text{ME}} = \operatorname*{argmax}_{p \in \mathcal{C}} H(p) \tag{5.10}$$

It can be shown that the above definition is sound in that $p^{\star}_{\text{ME}}$ is well-defined: there is always a unique distribution in any constrained set $\mathcal{C}$ which maximises the entropy.

This approach has several nice properties. First, the optimal $p^{\star}_{\text{ME}}$ has the following parametric form:

$$p^{\star}_{\text{ME}}(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{i=1}^{n} \lambda_i f_i(x,y)\right) \tag{5.11}$$

with appropriate λ_i's, where $Z(x)$ is a normalising factor, defined as:

$$Z(x) = \sum_y \exp\left(\sum_{i=1}^{n} \lambda_i f_i(x,y)\right) \tag{5.12}$$

That is, in this model, each feature is associated with a *weight*, and the logarithm of the probability of an event is proportional to the linear combination of the indicator features:

$$\log p^\star_{\text{ME}}(y|x) \propto \sum_i \lambda_i f(x,y) \tag{5.13}$$

for any fixed x. The family of probability models with such a property is called *log-linear models*. We refer to the set of log-linear models with indicator features $f_1, \ldots, f_n$ as $\mathcal{L}$.

Another important characteristics of the maximum entropy model $p^\star_{\text{ME}}$ is that it happens to be the model in the family $\mathcal{L}$ of log-linear models that best accounts for the training data; that is, it is the model that maximises the likelihood of the training data. Formally, the log-likelihood of the empirical distribution $\tilde{p}$ as predicted by the probability distribution p is defined as:

$$L(p) \stackrel{\text{def}}{=} \sum_{x,y} \tilde{p}(x,y) \log p(y|x) \tag{5.14}$$

Denote $p^\star_{\text{ML}}$ the model in $\mathcal{L}$ that maximises the log-likelihood, that is:

$$p^\star_{\text{ML}} \stackrel{\text{def}}{=} \operatorname*{argmax}_{p \in \mathcal{L}} L(p) \tag{5.15}$$

With these definitions, $p^\star_{\text{ME}} = p^\star_{\text{ML}}$, i.e., the maximum entropy model in $\mathcal{C}$ is the model in $\mathcal{L}$ that maximises the likelihood of the training sample.

Since the maximum entropy model happens to be a log-linear model as well, it is very easy to calculate the conditional probability $p^\star_{\text{ME}}(y|x)$ once we know the feature weights $\lambda_1, \ldots, \lambda_n$. But how can we determine the weights? Unfortunately, there is no closed-form solution in the general case, therefore we have to rely on numerical optimisation methods. The log-likelihood function L, however, behaves well in $\mathcal{L}$: it is convex, hence all local maxima are global maxima as well, i.e., there is a unique global maximum of the likelihood function.[6] This means that several general purpose optimisation methods can be applied to the problem; Malouf (2002a) gives a comparison of a handful of such algorithms.

An algorithm specially tailored to log-linear models is the Generalised Iterative Scaling (GIS) algorithm of Darroch and Ratcliff (1972), summarised in Figure 5.1. It starts with setting the parameters to 0 and calculating the expected

[6] This does not mean that there is a unique parameter set maximising the likelihood function. In fact, there might be many of them (see Bancarz and Osborne 2002).

1 set $C = \max_{x,y} \sum_{i=1}^{n} f_i(x,y)$
2 set $t = 0$
3 **foreach** $i = 1, \ldots, n$
4 set $\lambda_i^{(0)} = 0$
5 calculate $\tilde{p}(f_i)$
6 calculate $L(p^{(0)})$
7 **do**
8 **foreach** $i = 1, \ldots, n$
9 calculate $p^{(t)}(f_i)$
10 set $\lambda^{(t+1)} = \lambda^{(t)} + \frac{1}{C} \log \frac{\tilde{p}(f_i)}{p^{(t)}(f_i)}$
11 calculate $L(p^{(t+1)})$
12 increment t
13 **while** $(L(p^{(t)}))$ not converges

Figure 5.1: The Generalised Iterative Scaling algorithm.

value of each feature according to the empirical distribution $\tilde{p}$ (lines 4–5). Then, in line 6, it calculates the log-likelihood $L(p^{(0)})$ of the training data according to the initial uniform distribution. Lines 7–13 contain the iterative part of the algorithm. In each iteration step (t), we calculate the expected value of each feature according to the probability distribution $p^{(t)}$, using the formula in Eq. (5.7) (line 9). Then, each feature weight λ_i is updated in line 10 by an amount proportional to the ratio of our objective $\tilde{p}(f_i)$ and the expected value according to the current model $p^{(t)}(f_i)$. Intuitively, in each iteration step, we change the model parameters so that the expected value of each feature according to our model gets closer to what we observe in the training data. The algorithm terminates when the log-likelihood of the data according to our model no longer improves (or at least when the improvement is under a threshold). Note that the original algorithm of Darroch and Ratcliff (1972) requires a slack feature f_{n+1} which ensures that $C = \sum_{i=1}^{n+1} f_i(x,y)$, for all (x,y) pairs. Curran and Clark (2003), however, show that this feature is not necessary.

Computationally the most expensive part of the algorithm is calculating the expected value of each feature, which requires a double loop: for each training instance x, we have to loop through all possible underlying structures y.[7] That is, the worst case complexity of each iteration is $\mathcal{O}(nNY)$, where N is the size of the

[7] By possible underlying structures we mean both observed and unobserved ones. For instance, if the underlying structure is a syntactic analysis of the given sentence, we have to loop through *all* possible parse trees.

training sample, n is the number of indicator features and Y is the average number of possible analyses. In practice, training time might be prohibitive when we have many indicator features (n is large) and/or many possible structures (Y is large).[8]

5.1.2 Features

In the ME approach, indicator features translate linguistic aspects of the data to machine-learnable structures. Therefore, the task of designing the right feature set is the most crucial part of the learning process, and requires a human expert. The relevance and quality of the features have an immediate effect on the learner's capability to capture the necessary generalisations. However, one has to take not only the linguistic side into account but also the characteristics of the learning algorithm. Too many and too specific features result in *overfitting* the training data and lead to decreased performance in the general case. Therefore, a fine balance is to be attained: one has to select a relatively small amount of important features.

How does this happen in practice? Usually,[9] we do not design individual features, but rather *feature templates* (see, e.g. Ratnaparkhi 1996). Suppose, for instance, that we hypothesise that the target word might be of importance for deciding whether it is preceded by an EE or not (e.g. *to* is often preceded by **PRO–NP**). In order to encode this intuition, we define a template: `current_word=X`, where `X` is taken to be a variable. Then, while preparing our training data, we instantiate several features by substituting the variable `X` with the current word:

`current_word=`*to*
`current_word=`*house*
...

We call each such feature a *basefeature*. The basefeatures are still not the features the training algorithm manipulates. The indicator features discussed in the previous section are pairs in the form of $\langle label, basefeature \rangle$:

f_1: EE=* and `current_word=`*to*
f_2: EE=**PRO–NP** and `current_word=`*to*
...
f_{101}: EE=* and `current_word=`*house*
f_{102}: EE=**PRO–NP** and `current_word=`*house*
...

[8] Goodman (2002) proposes a speedup for the GIS algorithm; unfortunately the memory requirement of the new algorithm is generally too high to be feasible.

[9] In what follows, we assume that we essentially have a labelling (tagging) task, where each word is assigned a label from a relatively small fixed set.

These features are usually automatically instantiated before the learning process; the training algorithm, then, associates a weight with each such feature.

Now, let us turn our attention to the task at hand: detecting where EEs occur in the input sentence. In the present setting, we employ two main types of features – local ones (Tables 5.1 and 5.2) and non-local ones (Table 5.3).

Local features represent information about the words and their POS-tags occurring in a window of 5 words around the target word (the $(i+1)$th word in the sentence). This five-word window corresponds to a trigram model, where we consider both left and right trigrams around the focus word at the same time. Table 5.1 shows the "lexical" features, adopted from (Ratnaparkhi 1996), which encode the surface form of the surrounding words. The motivation for using these features is straightforward: many EEs co-occur with only a small set of trigger words. For example, the empty SBAR complementiser (**COMP–SBAR**) mostly occurs after words of utterance, such as *said, says, explain*, etc. Similarly, no empty complementiser can be followed by *what*, and they only very rarely occur before the word *that*.

Although lexical features are often helpful, they prove to be too specific in many other cases. Therefore, we also make use of the POS information of our input (we assume POS-tagged input). The features encoding POS-tags in a window of 5 words around the target word are less fine-grained than the lexical features, but they are capable of capturing interesting generalisations (Table 5.2). For example, many **NP–NP** traces representing passives are preceded by the sequence of some form of *be* and the past participle of some (transitive) verb. Similarly, the EE representing empty units (**UNIT**) always comes after a $-sign followed by some numeric expressions, such as *123* or *billion*. Clearly, lexical features cannot capture this generalisation, since the identity of the number is completely irrelevant and the same number does not necessarily occur in the training set. Moreover, data sparseness due to the high number of lexical entries prevents us from using bigrams of words, whereas features representing bigrams of POS-tags do not cause any specific problem.

Finally, the observation that EE labels are generally preceded by the label "no-empties" (EE=*) calls for a feature template encoding the identity of the previous predicted label ($l_{i-1} = X$).

Local features, useful in the majority of the cases (cf. Section 5.3), are sometimes very restrictive. Frequently, interesting cues lie outside a window of five words, or even if they are within this window, they do not span a contiguous interval. Therefore, we employ further features, summarised in Table 5.3, to capture these cues. All of these features are implemented as regular expressions, hence the trace tagger can still be viewed as a finite-state machine.

$w_i = X$; $w_{i-1} = X$; $w_{i+1} = X$
X is a prefix of w_i, $\|X\| \leq 4$
X is a suffix of w_i, $\|X\| \leq 4$
w_i contains a number
w_i contains uppercase character
w_i contains hyphen

Table 5.1: Local lexical features at position $i+1$. X is a variable representing a word.

$pos_i = X$; $pos_{i-1} = X$; $pos_{i+1} = X$
$pos_{i-1}pos_i = XY$
$pos_{i-2}pos_{i-1}pos_i = XYZ$
$pos_i pos_{i+1} = XY$
$pos_i pos_{i+1} pos_{i+2} = XYZ$

Table 5.2: Local POS-related features at position $i+1$. X, Y, Z are variables representing POS-tags.

Target	Matching regexp	Explanation
NP–NP	BE (RB|RBR)* VBN __	passive
{**NP–NP** / **PRO-NP**}	__ RB* *to* RB* VB	to-infinitive
	N [,:] __ RB* VBG	gerund
COMP–SBAR	(V|,) __ *!that** (MD|V)	lookahead for *that*
WH–NP	!IN {WP / WDT / **COMP–WHNP**} **!WH–NP*** V __	lookback for pending **WH–NP**s
WH–ADVP	WRB **!WH–ADVP*** V **!WH–ADVP*** __ [.,:]	lookback for pending **WH–ADVP** before a verb

Table 5.3: Non-local binary feature templates; the EE-site is indicated by __

passive

This feature is the generalisation of the feature discussed above capturing a frequent site for passive **NP–NP**. The feature fires when a form of *be* is followed by a past participle, separated by optional adverbs. Indeed, in many cases, the passive verb is modified by adverbs, such as in the examples:

`is (VIS) n't (RB) expected (VBN)` **NP–NP** (5.16)
`are (VARE) more (RBR) easily (RB) rejected (VBN)` **NP–NP**

to-infinitive

The feature captures the generalisation that in control constructions, the **PRO** is followed by an infinitival *to*, which might be separated by adverbs from both the EE and the infinitive. Examples are:

NP–NP `To (INF) further (RBR) load (VB)` the stakes... (5.17)
PRO-NP `Just (RB) to (INF) say (VB) ...`

gerund

Similarly, many gerundial clauses start with adverbs. The regular expression matches a frequent environment where gerunds occur, such as in the examples:

`...schedule (NN) , (,)` **PRO–NP** `resulting (VBG)` in... (5.18)
`...ideas (NNS) , (,)` **NP–NP** `frequently (RB) putting (VBG)...`

lookahead for *that*

This feature checks whether a complementiser *that* occurs between the current verb and the next finite verb. The goal is to prevent the model from proposing a **COMP–SBAR** label if there is an overt complementiser.

lookback for pending WH–NP

With the aid of this feature template, we can verify if the sentence is a relative clause or an embedded question with object extraction. Examples are:

a project **COMP–WHNP** `he heads (VBZ)` **WH–NP** (5.19)
`which (WDT) most Americans previously had (VHAD)` **WH–NP**

lookback for pending WH–ADVP

This regular expression matches frequent sites of extracted adverbs. It searches for a WH-adverb (such as *how*, *where*, etc.) preceding the current

main verb. In many cases, **WH–ADVP**s immediately precede punctuation, although not necessarily the first one after the main verb of the clause:

```
when (WRB) other foreign countries, notably Britain,        (5.20)
are acquiring more American assets WH–ADVP .
```

5.1.3 The model

Training

Even with very careful design of the feature set, ME models tend to overfit the training data (Berger *et al.* 1996, Chen and Rosenfeld 1999, Curran and Clark 2003), therefore it is customary to apply some kind of smoothing or feature selection algorithm to increase the generalisation capability of the model. Two widely applied methods are simple frequency cutoff (e.g. Ratnaparkhi 1996) and imposing a Gaussian prior on the weights (Chen and Rosenfeld 1999). Although a Gaussian prior, when compared to simple cutoff, is reported to produce slightly superior models in NLP settings (Curran and Clark 2003), it requires a fairly expensive optimisation of the variance(s) associated with the features. Apart from its simplicity, cutoff smoothing has a further advantage: it actually reduces the number of features, hence it speeds up both training and testing and decreases memory consumption. These considerations led us to apply frequency cutoff to our tagging model.

Having decided to use a frequency cutoff to improve and/or speed up the model, there are two important questions to address: (i) what do we mean by cutoff? and (ii) what is the right cutoff value? As for the first question, we have two options: we discard either low-frequency basefeatures or low frequency features (i.e., $\langle label, basefeature \rangle$ pairs). The difference is subtle: deleting an infrequent basefeature means that this particular aspect of the data is not important or reliable for making a choice. On the other hand, deleting individual features means that the given label–basefeature combination is unlikely, although other labels are more likely to co-occur with the given basefeature. Note that deleting infrequent features also results in deleting infrequent basefeatures. In the experiments, we take the approach of deleting infrequent features, even if the corresponding basefeature is frequent.

Another problem is determining the right cutoff value. Increasing the cutoff value has two effects: first, it decreases the number of (possibly uninteresting) features and labels, hence it leads to a better and more compact model. On the other hand, ignoring too many features results in losing interesting aspects of the data and, ultimately, in the deterioration of the model. Figures 5.2 and 5.3 show the effects of changing the cutoff value on the accuracy of the resulting model

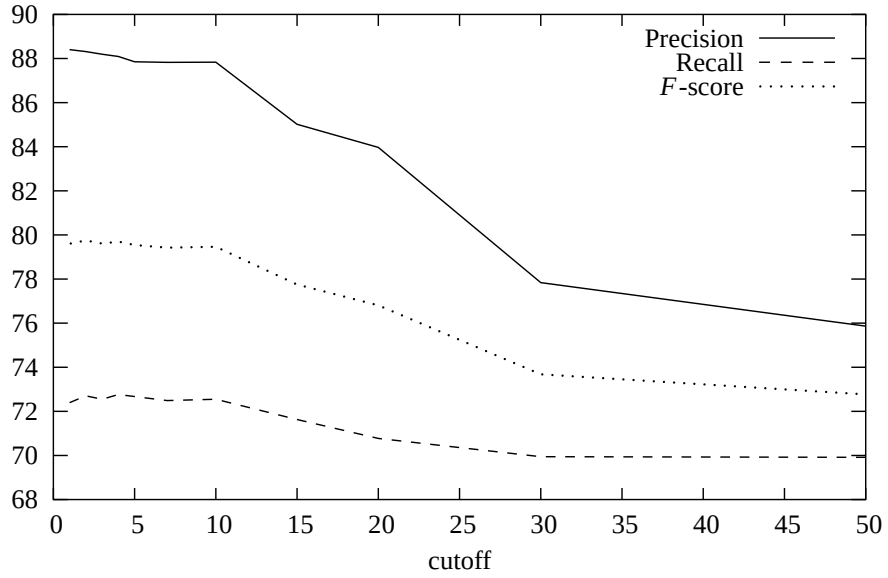


Figure 5.2: Effects of frequency cutoff on the accuracy of the trace tagger.

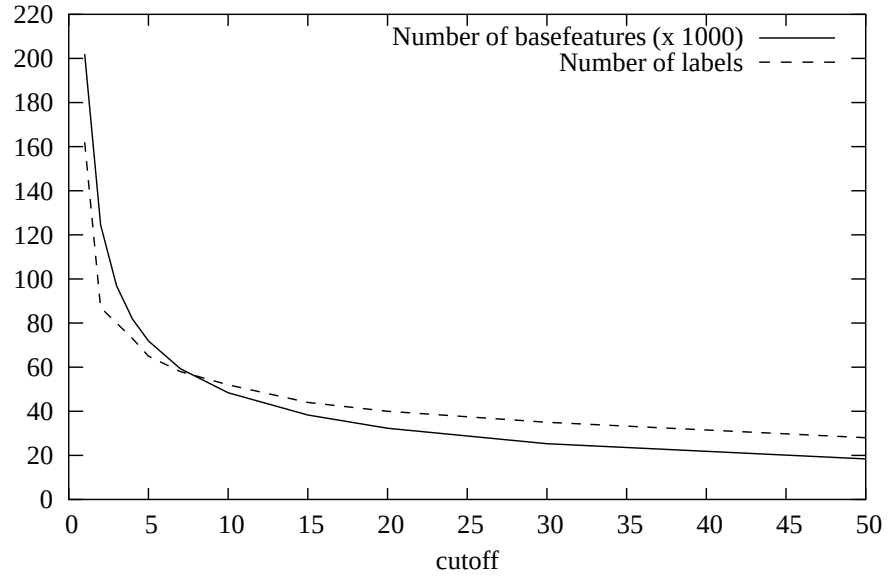


Figure 5.3: Effects of frequency cutoff on the number of basefeatures and labels.

(evaluated on the development data, WSJ00), and on the number of basefeatures and the number of labels. Cutoff values between 0 and 10 have no dramatic effect on the accuracy of the resulting model, whereas the cutoff value of 10 practically halves the number of features and labels, speeding up training and testing by a factor of 2. With larger cutoff values, however, the resulting models start to deteriorate relatively fast. Therefore, in the experiments below, we set the cutoff value to 10.

In the experiments described here, we use the standard training sections of the Wall Street Journal corpus (WSJ02–21). After extracting the necessary information from the training data, we applied our own implementation of a slightly modified version of the Generalised Iterative Scaling (GIS) algorithm (for details, see Curran and Clark 2003).

Testing

Our tagger is very similar to Ratnaparkhi's POS-tagger (Ratnaparkhi 1996), with one important difference: the input is POS-tagged text. In the experiments reported here, we use the correct POS-tags from the Treebank. The tagger, then, searches for the best label sequence (i.e., the best EE-sequence) given the sentence that maximises the joint (or equivalently the conditional) probability of the input and the sequence:

$$\operatorname*{argmax}_{l_1 \ldots l_n} p(l_1 \ldots l_n, \langle w_1, t_1 \rangle, \ldots \langle w_n, t_n \rangle) = \operatorname*{argmax}_{l_1 \ldots l_n} p(l_1 \ldots l_n | \langle w_1, t_1 \rangle, \ldots \langle w_n, t_n \rangle) \quad (5.21)$$

where l_is stand for EE-labels, w_is for words and t_is for POS-tags. Now, we approximate the above probability with the product of probabilities of a given label l_i occurring in a given context c_i, where the context is represented by the features described in Section 5.1.2:

$$p(l_1 \ldots l_n | \langle w_1, t_1 \rangle, \ldots \langle w_n, t_n \rangle) \approx \prod_{i=1}^{n} p(l_i | c_i) \quad (5.22)$$

Note that we assume independence between the contexts: the proper terms should be $p(l_i | c_1 \ldots c_i)$. Our contexts, however, capture (a large part of) previous contexts and under a trigram Markov assumption, they are in fact a reasonable approximation of the true probabilities.

The tagger performs a variant of beam search: k-best breadth first search. At each point of time we maintain the k-best paths (label sequences) leading to the given word. Although k-best breadth first search is not guaranteed to find the

Model	strict	(Johnson 2002)
Baseline (unigram)	24.9	33.4
Baseline (trigram)	69.0	74.4
Trace tagger	79.6	**83.2**
(Johnson 2002)	–	79.0

Table 5.4: The performance of trace detectors (labelled F-score) on the strict and Johnson's (2002) metrics.

highest probability path, it is fairly adequate in this setting, since our feature set enforces mostly local decisions. Exceptions are the two "lookback" features: they express interdependence between non-adjacent labels.[10]

In the implementation of the search algorithm, we use a priority queue of maximal length k for storing competing analyses. The queue is implemented as a singly linked sorted list, and since k is small, we employ incremental search to insert new elements into the list. Therefore, the worst-case complexity of our tagger is $\mathcal{O}(nlk^2)$, where n is the sentence length, l is the number of labels. In the experiments here, we used $k = 3$ setting, where k was optimised on the development set (WSJ00).

5.2 Results

In the present section, we report the performance of the trace tagger on the standard test section, Section 23 of the Wall Street Journal corpus. We use the evaluation metrics for detecting EEs described in Section 4.3.2: an EE is correctly detected if the type of the EE is correct and it occurs between the same words as in the gold standard.

The performance of the trace tagger is compared to two baseline models as well as to (Johnson 2002). One of the baselines is the most natural one: we independently select the most probable label for each ⟨word,POS-tag⟩ pair; this model amounts to a unigram hidden Markov model (HMM). The second baseline we employ is slightly more sophisticated: we apply an HMM-tagger (Brants 2000). However, as we argued in the previous section, one has to encode some contextual information in the label itself, otherwise the hidden states would be uninformative. Therefore, in this model, labels (hidden states) are ⟨POS-tag,EE-tag⟩ pairs.

[10] Because of the existence of such features, standard Viterbi-search is not applicable. Dienes and Dubey (2003b) try to approximate the probabilities with a Viterbi-like search (which treats paths differing only in lookback information as equivalent), and report marginally worse performance (with 0.5% worse overall F-score).

EE	Frequency	Prec. Here	Rec. Here	*F*-score Here	*F*-score Johnson
LABELLED	3864	87.8	72.8	79.6	–
UNLABELLED	3864	96.4	80.0	87.5	–
WH/PRO/TOP	2634	85.8	74.3	**79.6**	–
NP–NP	1148	88.1	80.2	84.0	–
COMP–SBAR	545	93.9	76.9	84.6	84.0†
WH–NP	508	91.8	75.2	**82.7**	81.0
PRO–NP	477	72.4	69.2	70.7	–
UNIT	388	99.7	93.5	**96.5**	92.0
TOP–S	277	93.9	88.5	**91.0**	88.0
WH–ADVP	171	80.4	45.6	58.2	56.0
COMP–WHNP	107	67.8	37.4	48.2	47.0
NP/PRO–NP	1625	91.5	84.6	**87.9**	82.0

Table 5.5: EE-detection results on WSJ23 and comparison with (Johnson 2002) (where applicable).

In principle, this setting would imply that the tagger has to find both the correct EE-labels and the correct POS-tags. To avoid such an unfair competition, the input "words" are concatenations of words and POS-tags. Table 5.4 summarises the results. Note that Johnson (2002) uses a slightly different evaluation scheme: he does not distinguish bound and unbound **PRO**s (**NP–NP**s and **PRO–NP**s). Table 5.4 shows results according to both metrics, where applicable.

According to our stricter metrics, the tagger achieves 87.5% unlabelled and 79.6% labelled *F*-score, clearly outperforming the baseline systems. Under Johnson's (2002) metrics, the overall labelled *F*-score increases to 83.2%, which is around 4% higher than the best reported result on the same task (Johnson 2002). These results amount to 97.96 – 98.13% tagging accuracy.

5.3 Discussion

The success of the trace detector is surprising, especially when compared to Johnson's algorithm which employs hand-written patterns on the output of a state-of-the-art parser. The tagger can reliably detect NLD sites without explicit knowledge of the phrase structure. This shows that, in English, non-local dependencies mostly occur at well-defined sites, where local cues are generally strong.

Interestingly, even the trigram baseline model performs quite competitively on the task: it can detect EEs with 74.4% labelled F-score under Johnson's (2002) metric, which is only 4.6% worse than Johnson's more sophisticated approach using a parser. More importantly, the shallow approaches outperform the unlexicalised parser with quite a large margin (by $20-30\%$ F-score, cf. Table 4.1) on the EE-detection task. What is the explanation of their success?

One important factor, as we will see in Chapter 7, is lexicalisation. Indeed, the taggers described here model the frequencies of ⟨word,EE-label⟩ and ⟨POS-tag,EE-label⟩ pairs, whereas this information is not available for the unlexicalised parser. The other factor that plays an important role is the presence of long-distance cues, from which the parser cannot benefit.

It is interesting to look at the system's performance on a case by case basis. Table 5.5 summarises the precision and recall for the detection of the most frequent EE-types (the remaining cases add up to around 6% of the EEs). One striking result is the considerably large gap between precision and recall of the tagger, a behaviour Johnson (2002) also reports. Thus, we can refine our claim with respect to detecting NLDs in English: most of the EEs occur at sites which are strongly marked on the surface. The detection of these EEs is relatively easy requiring no explicit phrase structure information. However, approximately one quarter of the NLDs are truly difficult cases without no local and lexical cue.

Table 5.5 also compares Johnson's (2002) results and the performance of the trace tagger. Generally, the tagger outperforms the parser+post-processing architecture in the majority of the cases, but the difference is the most striking in the case of the most frequent categories: the tagger is surprisingly accurate at detecting **PRO**s, although it has problems telling uncontrolled and controlled **PRO**s apart. The difference in categories with lower frequency is less prominent, although the shallow approach is be more accurate at detecting empty units (**UNIT**) and topicalised sentences (**TOP–S**) as well.

5.3.1 Error analysis

This section discusses the errors the trace tagger makes. Such a discussion is important for two reasons. First, it might suggest further features helping trace detection and, thus, might contribute to the further improvement of the system. The second benefit of such an analysis is that it gives further insights into the distribution of EEs and NLDs in English by determining the occurrences where the shallow approach fails due to either the lack of local cues or local and non-local ambiguities. In order to avoid unfair optimisation on the test data, we carried out the error analysis on the development section (WSJ00).

NP–NP and PRO–NP

This group, covering around 50% of the EEs, proves to be the most challenging for the trace tagger as well as Johnson's (2002) approach. Generally, as the combined score in Table 5.5 shows, the tagger is fairly reliable in finding **PRO**s, but it cannot accurately distinguish them. The problem is that the binding NP might be very far from the EE-site, which makes it invisible for the model.

Another common source of error is reduced relative clauses. The lack of the copular *be*, which is the most important cue in the case of passive constructions, prevents the tagger from detecting the subject of such clauses:

> *The company said Mr. Stronach will personally direct the restructuring,* **NP–NP** *assisted* **NP–NP** *by Manfred Ginglm, president and chief executive.* (5.23)

Further cases where the tagger could not detect the EE properly include prepositional verbs (the tagger wrongly predicts the EE before the preposition), *to be to* constructions and coordination:

> *The declaration by Economy Minister Nestor Rapanelli is believed to be the first time such an action has been called for* **NP–NP** *by an Argentine official of such stature.* (5.24)

> *Michael Basham, deputy assistant secretary for federal finance, said the Treasury may wait until late Monday or even early Tuesday to announce whether the autions are* **NP–NP** *to be rescheduled.* (5.25)

> *Saudi Arabia, for its part, has vowed to enact a copyright law compatible with international standards and* **NP–NP** *to apply the law to computer software as well as to literary works, Mrs. Hills said.* (5.26)

WH–NP and COMP–WHNP

The tagger detects extracted **WH–NP**s with high precision. Recall, however, is considerably lower. A closer inspection of errors reveals that the tagger finds the extracted WH-subjects easily, whereas extracted WH-objects prove to be more difficult to detect. Indeed, in the case of subject **WH–NP**s local cues are very strong: the EE almost always occurs before a verb and after the complementiser *who* (followed by optional adverbs). One important exception is infinitival subject relative clauses without an overt complementiser, such as:

The first Champagne [COMP–WHNP] [WH–NP] *to crack that price barrier was the 1979 Salon de Mesnil Blanc de Blancs.* (5.27)

In such cases the tagger generally predicts a **PRO** instead of the correct structure. Around 7% of the **WH–NP**s and one third of the **COMP–WHNP**s belong to this class.

As for object **WH–NP**s, their site is generally less strongly marked on the surface: frequently, the complementiser is missing (i.e., it is also an empty element **COMP–WHNP**) and the only cue is the subcategorisation frame of the preceding verb – information which is only implicitly available for the tagger. Examples, such as

Documents filed with the Securities and Exchange Commission on the pending spinoff disclosed that Cray Research Inc. will withdraw the almost $100 million in financing [COMP–WHNP] *it is providing the new firm* [WH–NP] *if Mr. Cray leaves or if the product-design project he heads is scrapped.* (5.28)

emerges as being very difficult for the tagger.

Clearly, the detection of object **WH–NP**s and **COMP–WHNP**s are strongly tied: the detection of one of them entails the detection of the other. The low accuracy of the **COMP–WHNP** detection indicates the difficulty of the problem. Around two thirds of the **COMP–WHNP** cases and 15% of the **WH–NP**s occur in such a construction.

The problem is further complicated when the **WH–NP** is extracted over clause boundaries, as illustrated by the following sentences:

. . . a point [COMP–WHNP] *he wants* [WH–NP] *to make . . .* (5.29)
. . . this law [COMP–WHNP] *he tried to pass* [WH–NP] .

. . . and the $2.25 billion [COMP–WHNP] *Northeast says its bid is worth* [WH–NP] . (5.30)

TOP–S and COMP–SBAR

The system is generally very reliable at finding sites for extracted sentences, since most of these sites occur after verbs of utterance, such as *said, says, explained*, etc. However, when these words are followed by a noun phrase, the tagger can no longer decide, whether the following NP is the subject of the verb (**TOP–S**), starts a new subordinate clause with an empty complementiser (**COMP–SBAR**) or is simply an adjunct of the main verb. Compare the following three sentences, especially the similarity of the local cues around the underlined word *said*.

Although the purchasing managers' index continues to indicate a slowing economy, it isn't signaling an imminent recession, said **TOP–S** *Robert Bretz, chairman of the association's survey committee and director of materials management at Pitney Bowes Inc., Stamford, Conn.* (5.31)

Michaels Stores Inc., which owns and operates a chain of specialty retail stores, said **COMP–SBAR** *October sales rose 14.6% to $32.8 million from $28.6 million a year earlier.* (5.32)

Sea Containers, a Hamilton, Bermuda-based shipping concern, said Tuesday that it would sell $1.1 billion of assets and use some of the proceeds to buy about 50% of its common shares for $70 apiece. (5.33)

Detecting the correct EEs in these cases would require a deeper understanding of the sentence structure.

WH–ADVP

Extracted WH-adverbs have the worst detection results among the frequent EE-types, even though they occur in very specific environments. First, most clauses containing a **WH–ADVP** start with a WH-adverbial complementiser, such as *how*, *why*, *when*, etc. This cue is captured by the feature performing a lookback for such words. Moreover, a closer look at the data reveals that **WH–ADVP**s mostly precede a punctuation mark, generally either a comma or a full stop. Why is the accuracy of the tagger so low on such items, even though they occur at sites with strong cues?

One complication in detecting **WH–ADVP**s is their different syntactic status: previously discussed EEs were all complements, whereas WH-adverbs are typically adjuncts, hence they are mostly optional and subject to weaker lexical selection. Furthermore, according to the Penn Treebank representation scheme, WH-adverbs are preceded by complements and overt adjuncts. Therefore, in addition to hypothesising a **WH–ADVP**, the tagger has to find both the *head* and the *end* of the corresponding verb-phrase. As the following example shows, the head and the dependent **WH–ADVP** as well as the corresponding WRB might be separated by several intermediate clauses (i.e., verbs which, in principle, could also be heads) and commas:

The bank stocks got a boost when Connecticut Bank & Trust and Bank of New England said they no longer oppose pending legislation that would permit banks from other regions to merge with Connecticut and Massachusetts banks **WH–ADVP**. (5.34)

> . . . *when* *other foreign countries, notably Britain, are acquiring more American assets* **WH–ADVP** . (5.35)

The tagger, unaware of phrase constructions, cannot detect these structures accurately, hence the low precision and recall figures. This EE-type is relatively rare (4.5% of all EEs); on the one hand, this means that it is difficult to reliably estimate their probability distribution, which is another reason for the low score the tagger achieves on detecting them. On the other, their sparsity entails that their impact on the overall score is also limited.

5.3.2 Feature analysis

In the previous section we explored which cases are difficult for the tagger and why. As a complementary approach, we also investigate which features contribute most to the accuracy of the tagger. Such an analysis reveals the relevant aspects of the data and help determine important features which could eventually be incorporated into other systems aiming at detecting NLDs as well.

In the present section, we approach the problem in two different ways. First, we carry out *performance analysis*: we study the improvement of the accuracy of the tagger when we gradually turn on groups of feature templates. Second, using a ME-tagger allows us to take a more analytical approach: we can investigate the *weights* associated with features. Informally, in a ME-model, a high positive weight corresponds to strong positive correlation between the basefeature and the label, whereas a low negative weight suggests strong negative correlation.

Table 5.6 gives the results of the experiments with different feature sets carried out on the development data (WSJ00). We isolated three large groups of the features: (i) POS-related features, generated by the templates described in Table 5.2; (ii) lexical features (LEX, Table 5.1); and (iii) long-distance features (LDF, Table 5.3). We ran experiments using five different combinations of the three groups.

The results are remarkable. First, the tagger using only POS-related features, i.e., no lexical or long-distance information, performs surprisingly well and man-

Model	F-score
LEX only	68.9
POS only	71.3
POS+LDF	73.4
POS+LEX	77.4
All	79.5

Table 5.6: Performance analysis of several feature types.

ages to achieve 71.3% labelled F-score. Even this simple model outperforms the baseline models (cf. Table 5.4), although it has no information about lexical ⟨word,EE-label⟩ probabilities. This shows that cues at the POS-level are very strong for the detection of many NLDs. The model achieves accuracy comparable to the full-fledged tagger on detecting **NP–NP**s, **WH–NP**s, **UNIT**s and **TOP–S**s, whereas the performance on other EE-types is considerably lower. Handling **WH–ADVP**s proves to be very difficult for this model, achieving 7.3% F-score on detecting them.

Lexicalisation turns out to be equally useful for the task. The tagger employing lexical features only (in a window of three words!) achieves 68.9% labelled F-score. The strength of the model lies in detecting **COMP–SBAR**s and **TOP–S**s with high accuracy (83.5% and 88.0% F-score, respectively), showing that these EEs occur only after a well-defined set of words. Interestingly, the detection results for the EEs substituting NPs (**PRO–NP**, **NP–NP** and **WH–NP**) is $5-7\%$ lower than in the POS-only case, demonstrating that these elements are in many cases lexically bound, but in other cases POS-information (and larger window size) facilitates their detection.

The combination of lexical and POS-information brings the strengths of the two models together: the F-score rises to 77.4%, with most of the EEs almost as accurately detected as by the tagger having all features turned on, with the exception of the **WH–ADVP**s: this particular model attains only 8% F-score on them.

Long-distance features remedy this situation by raising the accuracy on **WH–ADVP**s to 58%. These features also improve the detection of the other EEs by around 1%. Interestingly, when combined with the POS-features only, long-distance features have a greater relative impact, showing that both lexical features and long-distance features capture some overlapping characteristics of the data. In fact, in certain cases, they seem to act against each other: the score for **WH–ADVP**s is 7% higher when lexical features are not switched on.

In summary, performance analysis reveals that information in a relatively small window (5 words) already contains valuable cues for the EE-detection task. POS-features and lexical features are equally important, capturing both similar and different aspects of the data. The relative importance of the long-distance features is considerably smaller (they improve the overall performance by around 2%), but in the case of **WH–ADVP**s their contribution is more dramatic.

We can refine our feature analysis by looking at the weights assigned to individual features. However, inspecting feature weights in itself is not sufficient: infrequent features tend to show a more extreme behaviour than frequent ones. Therefore, we weighted the feature weights by the relative frequency of the given features. This number is the *expected weight* of the given feature indicating how much this particular feature contributes to the probability of events on average.

In general, weights for long-distance features (cf. Table 5.3) tend to have a high absolute value. In fact, they often have a low negative value when the associated label is EE=*, whereas a high positive value when the label is their target label. This shows that these features are indeed important in detecting EEs; however, they are very specialised, and therefore relatively infrequent, hence their relatively small contribution to the overall score. Another interesting, and somewhat unexpected, observation is the relatively strong dispreference for EEs following words containing uppercase letters.

In the remaining of this section, we summarise the most important correlation patterns for the most frequent EE-types. In the following, the feature `0pos` represents the POS-tag of the word preceding the EE-site, whereas the feature `1pos` stands for the POS-tag of the following word. The complex feature `01pos` thus denote the POS-tag sequence enclosing the EE, whereas `-10pos` the two preceding POS-tags. Similarly, `0lex` and `1lex` refer to the preceding and following words, respectively. The suffix and prefix features (`suf1`, `suf2`,... and `pref1`, `pref2`,...) apply to the preceding word.

NP–NP and PRO–NP

In general, the long-distance features targeted at passive, infinitival, and gerundial constructions show high correlation with the occurrence of these EEs. There is one exception: **PRO–NP** EEs show relatively strong negative correlation with passive constructions, indicating that the model properly captured the generalisation that the patient of the verb undergoing passivisation has to occur within the sentence. POS-related features also play an important role in detecting **PRO**s with the right context having slightly stronger influence than the left one. Table 5.7 contains the first few most important features for this class:

WH–NP

Table 5.7 summarises the most important features in detecting extracted WH noun phrases. A key observation is the model's specialisation towards subject **WH–NP**s; object **WH–NP**s do not seem to occur in characteristic environments. The only exception is the long-distance feature performing a lookback (`wh`=yes), which captures cues for detecting extracted objects. It is also interesting to see how the model has generalised over triggering WH-words: the feature capturing the two-letter prefix of the previous word (`pref2`=*wh*) is the second most important cue for this class.

COMP–SBAR

Interestingly, neither the long-distance lookahead feature for non-occurrence of the word *that* (`that`=no) nor the feature capturing the word pre-

NP–NP	PRO–NP	WH–NP
1pos=TO	1pos=VBG	0pos=WDT
toinf=yes	toinf=yes	pref2=*wh*
1lex=*to*	1pos=TO	pref1=*w*
pass=yes	1lex=*to*	wh=yes
0pos=VBN	pos=VBN	suf3=*hat*
1pos=VBG	01pos=IN+VBG	0pos=WP
suf2=*ed*	suf2=*ed*	suf4=*that*
suf1=*d*	-10pos=NN+VBN	0lex=*that*
01pos=,+VBG	01pos=VBN+IN	pref4=*that*
01pos=VBD+TO	-10pos=NNS+VBN	

Table 5.7: The most important features for detecting **NP–NP**, **PRO–NP** and **WH–NP**.

ceding the EE (*said* in many cases) proved to be the most important feature in detecting empty SBAR complementisers. Instead, the feature expressing that the following word should be a pronoun comes first in the list (Table 5.8). In fact, the pronoun *it* has a strong a bias towards co-occurring with **COMP–SBAR**s. Similarly, many empty complementisers are followed by the determiner *the*. The following list describes the most important cues in order. Observe the generalisation pref2=*sa* from the trigger words: *said*, *says* and *say*.

COMP–WHNP

This item shows very similar behaviour to **COMP–SBAR**s; indeed, both of them have similar syntactic functions. **COMP–WHNP**s are, however, not lexically anchored, therefore the most important features try to capture subordinate clause boundary (Table 5.8). This task is fairly difficult, therefore there are only three features that really show strong correlation with the occurrence of empty WH–complementisers (see previous table): **COMP–WHNP**s tend to occur between a noun and a pronoun (in this order).

TOP–S

This category tends to occur at the end of the sentence, following the words *says* or *said*. Table 5.8 shows the most important features for this class of EEs.

WH–ADVP

Detecting adverbial WH-sites proves to be one of the most challenging cases for the tagger (along with the detection of null WH-operators). Feature

COMP–SBAR	COMP–WHNP	TOP–S
1pos=PRP	1pos=PRP	pref2=*sa*
pref2=*sa*	01pos=NN+PRP	-1lex=*”*
0lex=*said*	01pos=NNS+PRP	-1pos=”
pref3=*sai*		1lex=*.*
suf3=*aid*		1pos=.
suf2=*id*		01pos=VBD+.
01pos=VBD+PRP		012pos=VBD+.+*EOS*
pref1=*s*		-10pos=”+VBZ
1lex=*it*		-210pos=,+”+VBZ
0pos=VBD		0pos=VBZ
that=no		pref4=*says*
1lex=*the*		lex=*says*

Table 5.8: The most important features for detecting **COMP–SBAR**, **COMP–WHNP** and **TOP–S**.

weights also indicate the complexity of the task: there are only three features which emerge as showing relatively strong correlation with the occurrence of this particular EE. The most important one of them is the long-distance lookback feature for WH-adverbs such as *how*, *when*, etc. Indeed, the performance analysis has also shown that including this feature improves detection accuracy of **WH–ADVP**s from 7% to 58%.

WH–ADVP
wrb=yes
1pos=.
1lex=*.*

5.4 Related work

Employing a tagger to detect EEs or syntactic structure in general, shows similarities with the supertagging approach of Srinivas and Joshi (1999) (cf. also Clark 2002). Indeed, the success of both models is due to the presence of local cues which partly determine syntactic structure. Our task, i.e., finding EEs, is a subtask in supertagging: finding the right elementary tree entails finding NLDs. Our prediction is that supertagging would also greatly benefit from employing the features that prove to be useful for the trace tagger.

Another task which is similar is named entity recognition (e.g. Bikel *et al.* 1999). NER was the shared task of the 2002 and 2003 Conferences on Natural Language Learning (CoNLL). The papers, (Tjong Kim Sang 2002, Tjong Kim Sang and de Meulder 2003), introducing the task give a good overview of the field. Maximum entropy models are very popular among the competing systems; conditional models are reported to systematically outperform generative (HMM) models (Malouf 2002b, Florian *et al.* 2003).

Buchholz (2002) also uses a shallow approach to recover local and non-local dependencies. The input of her memory-based learner is chunked and POS-tagged text. She reports 67.34% F-score according to our HD metrics (cf. Section 4.1), with high (80.92%) precision and low (57.68%) recall.

Summary

In this chapter, we have shown that detecting empty elements is indeed feasible without explicit knowledge of phrase structure, i.e., before parsing. We have developed a maximum entropy tagger, the trace tagger, which detects NLD-sites accurately. In English, a relatively small window of five words contains reasonable amount of information to determine where EEs occur. Both lexical and POS-related features have proven to be useful for the task. The most difficult cases for the tagger are telling controlled and uncontrolled **PRO**s apart and detecting WH-traces in adjunct position (mainly **WH−ADVP**s).

Chapter 6

The combined architecture

In the previous two chapters, we showed that (i) parsing with EEs is only feasible if the unlexicalised parser knows where EEs occur in its input, and (ii) a tagger can provide this information fairly reliably prior to parsing. In this chapter,[1] we validate our two-step approach by combining the tagger and the parser. The combination method is a simple pipeline: the best hypothesis of the tagger is the input to the parser. We compare our results to Johnson's (2002).

In this chapter, we also report experiments with a state-of-the-art lexicalised parser, presented in (Collins 1997). Interestingly, although the lexicalised parser is substantially more accurate at constructing phrase structure, the difference between the unlexicalised and the lexicalised parsers is small when it comes to antecedent recovery. We discuss the reasons for such a behaviour.

6.1 The unlexicalised parser

Theoretically, the 'best' way to combine the trace tagger and the parsing algorithm would be to build a unified probability model. However, the nature of the models are quite different: the finite-state model is conditional, taking the words as given. The parsing model, on the other hand, is generative, treating the words as an unlikely event. There is a reasonable basis for building the probability models in different ways. As we argued in Section 5.1, a generative model for the tagger is not appropriate: since most of the words are tagged as EE=*, the hidden states of such a generative model would be uninformative. Although conditional parsing algorithms do exist, they are reported to be difficult to train using large corpora (Johnson 2001). Therefore, we employ a simple, non-probabilistic, architecture here: the input to the parser is the output of the tagger, and the parser

[1] This chapter builds on both (Dienes and Dubey 2003a) and (Dienes and Dubey 2003b).

		INSERT	PERFECT	TAGGER
EE det.	all	44.3*	100	79.6
(F-score)	**WH/PRO/TOP**	47.1*	100	79.6
ANTE rec.	all	37.7*	89.7	72.9
(F-score)	**WH/PRO/TOP**	38.0*	89.5	70.4
HD accuracy	**WH/PRO/TOP**	42.7*	86.3	67.8
Time (s/sent)		54.8	8.5	8.9
Chart (edges/sent)		2376k	327k	339k
PARSEVAL	≤ 100	75.4*	78.0	75.7
(F-score)	≤ 40	75.8*	79.3	77.0
Surface deps.		81.8*	82.5	81.0
All deps.		80.5*	82.7	80.9
Missed sents.		39.9%	0.6%	0.8%

Table 6.1: EE detection, antecedent recovery, HD score, parsing times, and missed parses for the unlexicalised parser (INSERT, PERFECT and TAGGER models).

treats the suggestions of the tagger as 100% correct. We refer to this model as the TAGGER model. Chapter 8 discusses a more sophisticated combination method.

The unlexicalised parser is the same as the PERFECT model of Section 4.3.1, i.e., it does not insert EEs and treats the existing EEs in its input as separate words. For the combined system, we report NLD-related scores (for antecedent recovery and HD-relations) as well as parsing time, size of the search space measured by the number of edges on the chart, and standard PARSEVAL scores.

Results

The results of the experiment are summarised in Table 6.1. The combined approach proves to be feasible: parsing time is only a little higher than in the case of the PERFECT model, and the antecedent recovery score is 5% better than Johnson's (2002) results. In the case of the semantically interesting EEs, the difference between Johnson's and our results increases to around 8%.

*These results are reported on sentences the parser could parse.

Discussion

The most important result of the experiments presented here is the success of the combined model, showing that the appropriate combination of simple shallow modules can efficiently and successfully solve a task which is generally considered to require more sophisticated tools. The most important effect of the trace tagger is the reduction of the search space the parser has to explore. The parser, using the hypotheses of the tagger, explores only 14% of the search space of the INSERT model, which results in a sixfold speedup. Due to the decrease in the size of the search space, beam search is more reliable at filtering edges, and the TAGGER model is both more robust and more accurate than the INSERT model.

Our approach has several potential weaknesses which could have led to its failure. An important problem with the architecture presented here is the rigidity of the combination method. Indeed, the parser proves to be very sensitive to tagging errors: whenever the trace tagger commits an error, it turns out to be difficult for the parser to recover and produce a reasonable tree. In Chapter 8, we present a novel probabilistic framework for the combination which overcomes this problem keeping the search space of the parser relatively small.

Another critical point is that we work with a very simple parser, an unlexicalised PCFG model, even though lexicalised parsers reportedly perform better than their unlexicalised counterparts (e.g. Magerman 1995, Charniak 1997, 2000, Collins 1997, Ratnaparkhi 1997, etc.). In the next section, we present an experiment using a lexicalised parser in the same setting as here.

Interestingly, when it comes to detecting non-local dependencies, our unlexicalised model still outperforms Johnson's (2002) pattern matching algorithm, who employs a state-of-the-art lexicalised parser. There are several crucial differences between the two approaches which render our system more accurate despite the use of a less accurate parser. First, we employ a parsing model which, albeit unlexicalised, is designed to capture non-local dependencies. Johnson, on the other hand, adopts a parser which does not handle NLDs.

The second difference, which follows from the previous one, is that we detect NLDs partly before, partly during parsing, whereas Johnson (2002) attempts to do everything after the parser has finished its job. It turns out, however, that the parser Johnson (2002) employs tends to get confused when trying to analyse sentences with NLDs, and, as a consequence, the pattern triggering the detection of an EE does not even occur. Indeed, Johnson (2002) reports a 9% decrease in EE-detection score when he applies his pattern matching algorithm to the output of the parser (as opposed to the perfect trees).

Third, Johnson (2002) offers one algorithm for both detecting EEs and recovering antecedents. Our approach, on the other hand, hinges on the insight that recovering antecedents requires structural information, whereas the occurrence of

EE	Frequency	Prec.	Rec.	*F*-score	
		Here	Here	Here	Johnson
ALL	3864	80.4	66.6	**72.9**	68.0
WH/PRO/TOP	2643	76.0	65.6	**70.4**	62.7†
NP–NP	1148	69.9	62.1	**65.8**	60.0
COMP–SBAR	545	93.2	78.0	84.9	84.0†
WH–NP	508	91.1	74.6	82.0	81.0
PRO–NP	477	69.9	67.1	**68.5**	50.0
UNIT	388	99.7	92.5	**96.0**	92.0
TOP–S	277	87.7	85.2	86.5	87.0
WH–ADVP	171	76.2	45.0	56.6	56.0
COMP–WHNP	107	68.3	38.3	49.1	47.0

Table 6.2: Antecedent recovery results for the unlexicalised TAGGER model and comparison with Johnson (2002).

EEs is mainly restricted by local and lexical constraints. Therefore, we have developed separate specialised modules for the two tasks, each of which outperforms Johnson's (2002) all-in-one approach on the corresponding tasks.

Finally, though maybe most importantly, the combined TAGGER model presented here makes use of lexical information while recovering NLDs. Although Johnson (2002) mentions that a simple lexicalisation method does not improve the results of his algorithm, we have seen in the previous chapter (Section 5.3) that lexicalisation plays an important role in EE detection: adding lexical features to the model improves the detection score by around 6%. Indeed, there are many different interacting cues at different levels which facilitate NLD recovery: the occurrence of grammatical function words (infinitival *to*, WH-adverbs, etc.), lexical preferences (whether a verb is transitive or tends to co-occur with EEs, like the words *said*, *explained*, etc.), local POS-sequences and structural (syntactic) information. Johnson (2002) cannot incorporate all of these cues.

Table 6.2, comparing the scores for antecedent recovery attained by the unlexicalised TAGGER model and by Johnson's (2002) algorithm, reveals that the most important weakness of Johnson's model is telling controlled and uncontrolled **PRO**s (**NP–NP**s and **PRO–NP**s) apart. Indeed, even in the case of perfect trees (!), Johnson (2002) reports 63% and 57% *F*-score for **NP–NP** and **PRO–NP**, respectively. These results are considerably worse than the scores achieved by our TAGGER model, which does not assume prior knowledge of phrase structure. As we argued in Section 3.2.1, the main problem is the lack of lexicalisation: cer-

tain verbs (e.g. *means*, *involves*) favour uncontrolled **PRO–NP**s while others (e.g. *begins*, *starts*) show a strong preference for controlled **NP–NP** traces. Johnson's (2002) unlexicalised pattern matching algorithm is unable to incorporate this information.

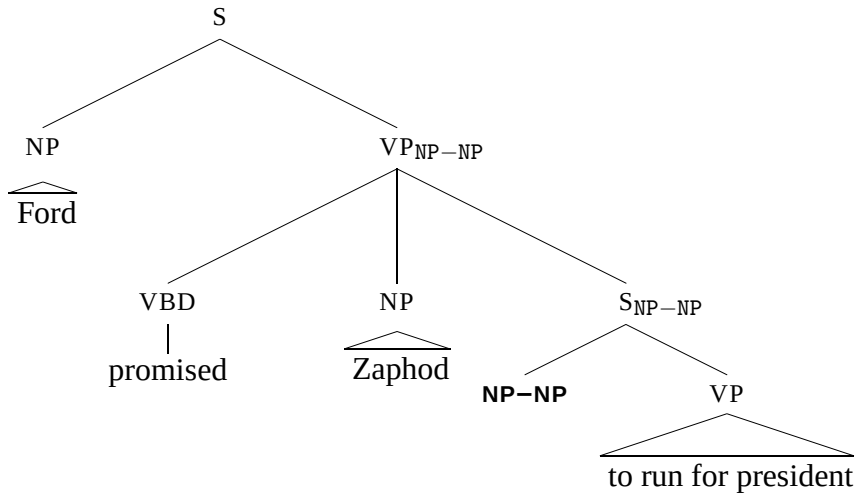
Although we do lexicalise the unlexicalised parser to some extent when combining it with the trace tagger, antecedent recovery might benefit from further lexicalisation, i.e., from a lexicalised parser. Specifically, as we already discussed it in Section 4.3.3, the unlexicalised parser cannot distinguish subject and object control. Compare the structures in Figure 6.1, which are almost identical, except for one label: in the case of subject control, the matrix verb is dominated by a $\text{VP}_{\text{NP}-\text{NP}}$ nonterminal label, whereas in the case of object control, the dominating nonterminal label is a VP. The choice between the two alternatives is exclusively determined by the matrix verb. This information is not available for the unlexicalised parser. A lexicalised parser, on the other hand, is able to make use of this cue. Therefore, we expect a lexicalised parser to be substantially more accurate at antecedent recovery – at least, for **NP–NP** traces. To test this hypothesis, we present experiments with a state-of-the-art lexicalised parser (Collins 1997, 1999).

6.2 Collins's parser

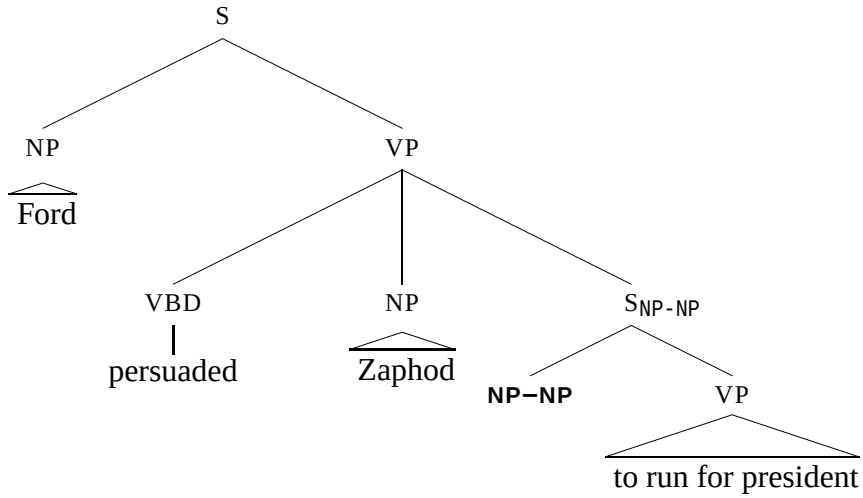
We have seen that the preprocessing approach advocated here already outperforms Johnson's (2002) post-processing approach by a margin of 5% on antecedent detection, even though we employ a fairly inaccurate (in terms of PARSEVAL scores) unlexicalised PCFG parser. In order to further improve the results, we substitute the unlexicalised parser with a state-of-the-art lexicalised one, with Collins's (1997) Model 2. On the PARSEVAL metrics, this model outperforms the unlexicalised parser by around 15% F-score.

In this section, we test the models with the lexicalised parser under two conditions: under the PERFECT and the TAGGER ones. As before, we use the simple pipeline architecture: the EEs are treated as separate words and the parser takes the input to be 100% correct, regardless of whether the information about the EE-sites comes from the gold standard or from the tagger. We defer the detailed description of the parsing model to the next chapter.

We employ the parser described in (Collins 1999). The parser itself requires only minor engineering modifications to incorporate handling EEs, such as allowing a larger number of nonterminals and removing hard-coded constants. The training algorithm, which extracts the necessary information from a treebank, on the other hand, demands a number of small adjustments to handle EEs properly. Furthermore, the original extraction algorithm used by Collins (1999) is barely



(a) Subject control.



(b) Object control.

Figure 6.1: Problems for the unlexicalised model: subject vs. object control

		NOTRACE	PERFECT	TAGGER
EE-det.	all	–	100	79.6
(F-score)	**WH/PRO/TOP**	–	100	79.6
ANTE-rec.	all	–	90.5[2]	74.3
(F-score)	**WH/PRO/TOP**	–	90.4	72.3
HD accuracy	**WH/PRO/TOP**	–	89.0	71.3
Time (s/sent)		1.8	1.6	1.6
Chart (edges/sent)		14k	12k	13k
Fully constructed edges (per sent)		434k	307k	331k
PARSEVAL	≤ 100	88.0	88.2	86.3
(F-score)	≤ 40	88.5	88.8	87.1
Surface deps.		90.7	90.0	88.6
All deps.		88.2	90.5	88.2

Table 6.3: Finding NLDs with a lexicalised parser: the PERFECT and the TAGGER models.

documented, therefore our system, when tested under the same circumstances as the original Model 2 in (Collins 1997, 1999), is slightly less accurate: it achieves 88.0% F-score on the PARSEVAL metrics. The parser uses beam search (Goodman 1998) to prune its search space, with the beam set to 10000.

In the experiments, we report NLD-accuracy as well as PARSEVAL scores, parsing time and the size of the search space. The latter is measured by the number of fully constructed edges and by the size of the chart, i.e., the number of edges remaining after the beam search.

Results

The main results of the experiments with the lexicalised parser are summarised in Table 6.3. Lexicalisation does improve the results, but only by a margin of around $1-2\%$ (cf. Table 6.1), even in the case where the perfect information about the occurrence of EEs is available for the parser (PERFECT model). The lexicalised

[2] Dienes and Dubey (2003a) report a slightly higher F-score, because they fail to discount pseudo-attachments.

EE	Frequency	Prec.	Rec.	F-score		
		Here	Here	Here	UNLEX	Johnson
ALL	3864	81.8	67.9	**74.3**	72.9	68.0
WH/PRO/TOP	2643	77.9	67.5	**72.3**	70.4	62.7†
NP–NP	1148	74.9	68.3	**71.5**	65.8	60.0
COMP–SBAR	545	93.9	76.9	84.6	84.9	84.0 †
WH–NP	508	89.6	73.4	80.7	**82.0**	81.0
PRO–NP	477	72.4	69.2	**70.7**	68.5	50.0
UNIT	388	99.7	93.6	96.5	96.0	92.0
TOP–S	277	83.9	79.1	81.4	**86.4**	87.0
WH–ADVP	171	73.2	41.5	53.0	56.6	56.0
COMP–WHNP	107	67.8	37.4	48.2	49.1	47.0

Table 6.4: Antecedent recovery results for the lexicalised TAGGER model and comparison with Johnson (2002) and with the unlexicalised TAGGER model.

TAGGER model outperforms Johnson's (2002) approach by 6.3% on the task of antecedent recovery; in the case of semantically interesting NLDs, the difference rises to almost 10%. Finally, PARSEVAL scores are only marginally higher under the PERFECT condition than in the case of the model which does not handle NLDs at all (NOTRACE model), whereas they are considerably worse when the trace tagger is applied as a preprocessor (TAGGER model).

Discussion

The most unexpected result of the experiments presented in this section is the surprisingly small difference between the lexicalised and the unlexicalised models with respect to antecedent recovery: even though the lexicalised parser is considerably more accurate according to the PARSEVAL metrics, it only outperforms the unlexicalised parser by $1-2\%$ when it comes to handling NLDs. A closer look at the results (Table 6.4) shows that the increase is only due to better recovery scores for controlled **PRO**s (**NP–NP**): lexicalisation, in the case of the TAGGER models, improves the score for this particular EE by approximately 6%.

When comparing the scores for antecedent recovery as well as for recovering head–dependent relations (HD) in the case of the PERFECT models (Table 6.5), we see a similar behaviour: as predicted, the lexicalised parser is more accurate at finding the antecedents for **NP–NP** traces. Observe, however, that the lexicalised PERFECT model is around 10% worse at recovering the antecedents for **WH–ADVP** gaps. Examining the errors in the development set reveals that their main cause is

EE	Frequency	Ante. rec.[3]		HD score[3]	
		LEX	UNLEX	LEX	UNLEX
ALL	3864	90.5	89.7	87.7	85.9
WH/PRO/TOP	2634	90.4	89.5	89.0	86.3
NP–NP	1148	86.0	81.7	85.1	79.4
COMP–SBAR	545	100.0	100.0	97.4	96.0
WH–NP	508	96.9	96.7	95.9	94.5
PRO–NP	477	100.0	100.0	96.2	92.9
UNIT	388	100.0	100.0	87.9	89.9
TOP–S	277	90.6	91.3	89.5	87.0
WH–ADVP	171	84.8	93.6	83.6	88.9
COMP–WHNP	107	100.0	100.0	98.1	100.0

Table 6.5: Antecedent recovery and HD scores for the lexicalised and unlexicalised PERFECT models.

the total confusion of the parser: the **WH–ADVP** elements get attached to the wrong phrases and their antecedent WHADVP or **COMP–WHADVP** often ends up in a different clause. Why is the lexicalised parser less accurate than the unlexicalised one in these cases? Why is the overall performance of the unlexicalised model so close to the lexicalised one?

We see two main factors that come into play here. First, we argued in the previous section as well as in Section 5.3 for the importance of lexicalisation in the task of EE detection. Collins's parser models two types of statistics: monolexical ones and bilexical dependencies. While the parser is shown to only marginally benefit from modelling bilexical dependencies (Gildea 2001), monolexical statistics prove to be very important. For instance, Klein and Manning (2003) present a lightly lexicalised parser incorporating this kind of lexical information only; their parser is more accurate than early lexicalised ones. Similarly, as far as monolexical statistics are concerned in the cases crucial for EE detection, we implicitly lexicalise our unlexicalised parser by using a lexicalised preprocessor (or the oracle). Consequently, the difference is relatively small between the lexicalised and the unlexicalised models.[4]

The other reason for the relatively small benefit of using the lexicalised model can be attributed to data sparseness. Using `gap+` variables increases the number

[3]Since the parser does not have to detect the EEs in this case, the precision, the recall, and the F-score are the same.

[4]The latter model would be better called lightly lexicalised.

of nonterminals considerably (by a factor of 7.1), making it more difficult to reliably estimate the parameters for the model. While this poses no problem for the unlexicalised model, adding lexicalisation enhances the sparseness of the training data, especially in the case of infrequent EEs, such as **WH–ADVP** or **TOP–S**. Furthermore, as we will see in Chapter 7, lexical information greatly restricts the rules available for the parser. The lack of available alternatives may drive the lexicalised parser towards a suboptimal (global) solution. In the case of the unlexicalised model, however, even the part of the grammar restricted by the tagger is large enough to allow the parser to attach and thread the EEs correctly.

Another effect of data sparseness is that handling EEs does not seem to improve the lexicalised parser as far as phrase structure and surface dependencies are concerned: even the PERFECT model attains lower scores on the measure evaluating its accuracy for recovering surface dependencies (although the PARSEVAL scores are marginally higher). The reason for such a behaviour is that the presence of EEs dramatically restrict the rules the parser can apply, and given the low frequency of EEs, the appropriate rule might not even occur in the training corpus at all. Even if the necessary rule appears, it is usually infrequent and thus the estimation for its probability is unreliable. This constellation drives the parser towards a suboptimal solution.

When combining the trace tagger and the lexicalised parser, the drop in both PARSEVAL and surface dependency scores is more remarkable: the TAGGER model performs $1.5 - 2\%$ worse according to these measures. Apart from the lack of available rules, the main reason is the rigid combination of the parser and the tagger: whenever the tagger makes a mistake, the parser cannot properly incorporate the (wrong) hypothesis into the parse tree, and is, thus, forced to come up with a parse tree which contains more errors. We conjecture that this effect of the mistakes the tagger makes is present in the case of the unlexicalised models as well, but they benefit more from the implicit lexicalisation discussed above.

Note, however, that having lower PARSEVAL scores than in the case of the NOTRACE model is not a problem in itself: although we lose some accuracy as far as phrase structure and surface dependencies are concerned, we also gain non-local dependency information which is unavailable to us when applying the NOTRACE model. Indeed, when it comes to detecting both surface and deep (non-local) dependencies, the TAGGER model is as accurate as the NOTRACE model.

Finally, observe the reduction of the search space the parser has to explore when it is informed about NLD-sites: in the case of the TAGGER model, the parser constructs 25% fewer edges than in the case of the NOTRACE approach. The size of the chart also decreases by around 10%, which results in a 12% speedup. The PERFECT model shows a similar behaviour.

In summary, lexicalisation does improve the results for recovering NLDs, but only by a relatively small margin. There are two reasons for such a behaviour.

First, handling NLDs increases the number of the parameters we have to estimate and therefore aggravates the sparse data problem. To alleviate this problem, improved smoothing techniques are required, the investigation of which is beyond the scope of the present dissertation. The second main cause of the relatively modest improvement lies in the rigid combination of the trace tagger and the parser. In the next two chapters, we explore two possible remedies: in Chapter 7, we test whether we need the preprocessor at all or the lexicalised parser in itself is able to detect NLDs. As we will see, it is feasible to detect NLDs with a lexicalised parser, but the results suggest that the parser might still benefit from consulting the trace tagger, provided they are integrated in a more appropriate way. In Chapter 8, we present a novel architecture to achieve this goal.

Summary

This chapter has presented a simple architecture combining the trace tagger and the parser: the parser considers the hypotheses of the trace tagger correct and constructs a parse tree without inserting further empty elements. Although this simple architecture is very rigid and makes the parser very sensitive to tagging errors, it still outperforms Johnson's (2002) approach on antecedent recovery by 5% even if we employ an unlexicalised parser. Lexicalisation improves the scores by 1.5%.

We have discussed experiments both with a relatively inaccurate unlexicalised and a state-of-the-art lexicalised parser. Interestingly, even though the lexicalised parser is much more accurate at building phrase structure, it does only marginally better when it comes to recovering non-local dependencies. The main reason, we have argued, is that the two COMBINED models are not that different: we implicitly lexicalise the unlexicalised parser by the preprocessing step, which makes extensive use of lexical information. Data sparseness and the rigid combination with the trace tagger have also been claimed to prevent the lexicalised parser from improving the results by a larger margin.

Chapter 7

Finding NLDs with a lexicalised parser

In this chapter,[1] we investigate whether a lexicalised parser suffers from the same efficiency and accuracy problems as the unlexicalised one (cf. Chapter 4). In Section 5.3 we argued that the success of the trace tagger is to a large extent due to lexicalisation: the tagger takes into account frequencies of ⟨word,EE-label⟩ and ⟨POS-tag,EE-label⟩ pairs, whereas this information is not available for an unlexicalised parser. Furthermore, in Section 6.2 we saw that a lexicalised parser only marginally improves the scores when combined with the tagger. We claim that the reason is, again, due to the lexicalised nature of the tagger: the lexicalised parser contributes no further information to the model. In fact, we claimed that lexicalisation might also hinder the parser when it comes recovering NLDs: it restricts the number of available choices so drastically that the parser cannot incorporate an EE, especially in the case of the TAGGER model when tagger makes a mistake.

However, this also suggests that a lexicalised parser would not suffer from the same problems as the unlexicalised one when trying to detect EEs. Indeed, if lexicalised parsing with EEs is feasible, it might be possible that it even outperforms the trace tagger, since a parser can make use of explicit phrase structure information.

In this chapter, we investigate these claims. Although Collins (1997) presents a model (Model 3) that can detect one EE-type (**WH–NP**), there is no probabilistic phrase-structure parsing model that can handle all EE-types. Therefore, in this chapter, we generalise Collins's approach to several EE-types (Model 4), and test the new parser in the same settings as we used for the unlexicalised INSERT model in Chapter 4.

[1] This chapter is the extension of Section 5 of Dienes and Dubey (2003a).

7.1 Model 3

Since our parsing model that handles several EE-types is a generalisation of Collins's (1997) Model 3, we present Model 3 in this section. First, we sketch the probability model and, then, briefly describe the parsing algorithm. Finally, we discuss the problems with Model 3. The first section closely follows (Collins 1999, pp. 168–175).

7.1.1 The probability model

In the following, we adopt Collins's notation. In general, a lexicalised context free rule has the following form:

$$P(h) \rightarrow L_n(l_n)\ldots L_1(l_1)H(h)R_1(r_1)\ldots R_m(r_m), \tag{7.1}$$

where upper-case letters stand for nonterminals, and lower-case ones represent ⟨headword,head-POS⟩ pairs (lexicalisation). P is referred to as the parent, H as the head, whereas L_i and R_j are left and right modifiers, respectively.

In his generative model, Collins defines the probability of the rule as:

$$p(L_n(l_n)\ldots L_1(l_1)H(h)R_1(r_1)\ldots R_m(r_m)|P(h)) \tag{7.2}$$

This probability can be expanded using the chain rule:

$$\begin{aligned} &p(L_n(l_n)\ldots L_1(l_1)H(h)R_1(r_1)\ldots R_m(r_m)|P(h)) = \\ &p_h(H|P(h)) \cdot \\ &\prod_{i=1}^{n} p_l(L_i(l_i)|L_1(l_1)\ldots L_{i-1}(l_{i-1}),P(h),H) \cdot \\ &\prod_{j=1}^{m} p_r(R_j(r_j)|L_1(l_1)\ldots L_n(l_n),R_1(r_1)\ldots R_{j-1}(r_{j-1}),P(h),H) \end{aligned} \tag{7.3}$$

To handle the extreme sparsity of the data, Collins assumes independence between the generation of the modifiers, i.e., he approximates this generation process as a unigram Markov process. Hence, the terms in Eq. (7.3) are simplified as:

$$\begin{aligned} &p(L_n(l_n)\ldots L_1(l_1)H(h)R_1(r_1)\ldots R_m(r_m)|P(h)) \approx \\ &\qquad \approx p_h(H|P(h)) \cdot \prod_{i=1}^{n} p_l(L_i(l_i)|P(h),H) \cdot \prod_{j=1}^{m} p_r(R_j(r_j)|P(h),H) \end{aligned} \tag{7.4}$$

Note, however, that the unigram Markov process would give too high a probability to too long sequences. To avoid this behaviour, Collins also generates STOP symbols L_{n+1} and R_{m+1} after the generation of the left and right modifiers (cf. also Brants 2000); once the STOP symbols are inserted, no more modifier is generated.

In summary, the generation of the RHS given the LHS of a rule (7.1) is viewed as a three-step process (Model 1):

1. Generate the head constituent H with probability $p_h(H|P(h))$.
2. Generate the left modifiers $L_1(l_1)...L_{n+1}(l_{n+1})$ as a unigram Markov process with probability $\prod_{i=1}^{n+1} p_l(L_i(l_i)|P(h),H)$, where L_{n+1} is the STOP symbol.
3. Generate the right modifiers $R_1(r_1)...R_{m+1}(r_{m+1})$ as a unigram Markov process with probability $\prod_{j=1}^{m+1} p_r(R_j(r_j)|P(h),H)$, where R_{m+1} is the STOP symbol.

Due to the independence assumption between the generation of the modifiers, this model fails to capture the difference between transitive, intransitive and ditransitive verbs, as well as the difference between complements and adjuncts. Clearly, some of the modifiers are obligatory, therefore the Markov process should not stop before generating all of them (complements). To handle this problem, Collins associates two subcategorisation frames (LC and RC) with each head: one for the left and one for the right modifiers (Model 2). The subcat frames are multisets of nonterminal labels the head requires as obligatory complements. In this model, complements have different non-terminal labels from adjuncts (NP–C and NP, respectively). Whenever a complement compatible with the subcat frame is generated, it is removed from the frame. The probability of generating a STOP symbol is 0 if the corresponding subcat frame is not empty. In summary, the probability model is extended as:

$p_h(H\|P(h)) \cdot$	generate head	(7.5)
$p_{lc}(\mathrm{LC}^{(1)}\|P(h),H) \cdot$	generate initial left subcat	
$p_{rc}(\mathrm{RC}^{(1)}\|P(h),H) \cdot$	generate initial right subcat	
$\prod_{i=1}^{n+1} p_l(L_i(l_i)\|P(h),H,\mathrm{LC}^{(i)}) \cdot$	generate left modifiers (and remove complements from subcat frame)	
$\prod_{j=1}^{m+1} p_r(R_j(r_j)\|P(h),H,\mathrm{RC}^{(j)})$	generate right modifiers (and remove complements from subcat frame)	

This model is further improved by incorporating **WH–NP** traces (Model 3). Model 3 assumes a similar gap-threading approach to the one we introduced in Section 2.3: gap information is encoded in the non-terminal labels; we discuss the differences in Section 7.2.2. Now, suppose that the LHS of a rule contains a

gap. There are three ways the gap information can be passed to the RHS: (i) the head takes the gap, (ii) one of the left modifiers takes the gap, or (iii) one of the right modifiers takes the gap. To model this choice, Collins introduces one more parameter G, which can have three values: **Head**, **Left** or **Right**. These values determine how the gap is passed on to the dependent. If G=**Head**, the head nonterminal in the RHS also has to contain a `gap` variable. In the case of **Left** or **Right**, a `gap` feature is added to the left or right subcat frame. This `gap` feature can disappear from the subcat frame if either a generated nonterminal contains a `gap` variable or a **TRACE** symbol is generated. The **TRACE** symbol also counts as an NP complement, therefore, it removes not only the `gap` feature but also a NP–C from the subcat frame. In summary, the generation of the rule is the following:

$p_h(H\|P(h))\cdot$	generate head	(7.6)
$p_g(G\|P(h),H)\cdot$	generate gap (G=**H,L,R**) if P has gap	
$p_{lc}(\mathrm{LC}^{(1)}\|P(h),H)\cdot$	generate initial left subcat & add `gap` if G=**L**	
$p_{rc}(\mathrm{RC}^{(1)}\|P(h),H)\cdot$	generate initial right subcat & add `gap` if G=**R**	
$\prod_{i=1}^{n+1} p_l(L_i(l_i)\|P(h),H,\mathrm{LC}^{(i)})\cdot$	generate left modifiers (and remove complements and `gap` from subcat frame)	
$\prod_{j=1}^{m+1} p_r(R_j(r_j)\|P(h),H,\mathrm{RC}^{(j)})$	generate right modifiers (and remove complements and `gap` from subcat frame)	

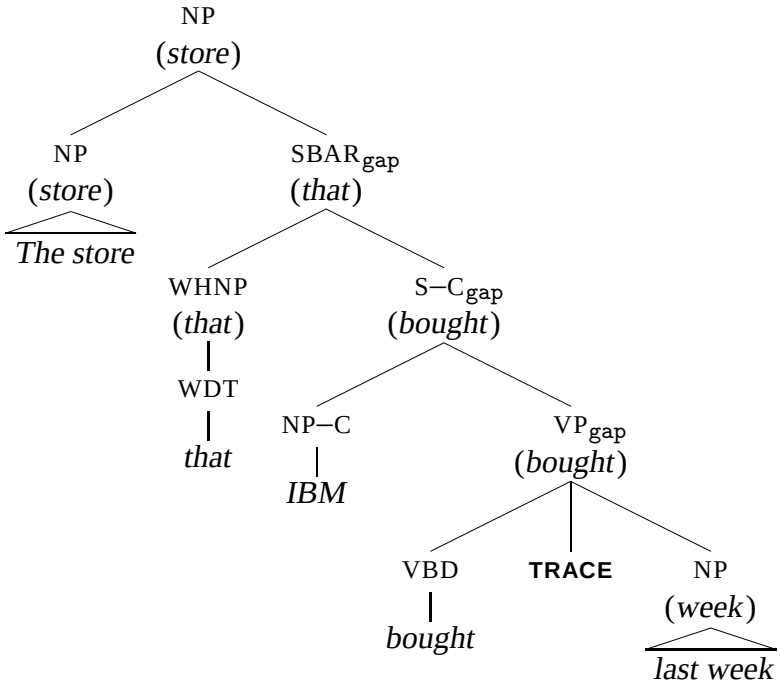
Thus, the probability of rule (2) in Figure 7.1 is calculated as follows:

$$\begin{aligned}
&p_h(\mathrm{WHNP}|\mathrm{SBAR},\textit{that})\cdot p_g(\mathbf{R}|\mathrm{SBAR},\textit{that},\mathrm{WHNP})\cdot \\
&p_{lc}(\{\}|\mathrm{SBAR},\textit{that},\mathrm{WHNP})\cdot p_{rc}(\{\mathrm{S{-}C}\}|\mathrm{SBAR},\textit{that},\mathrm{WHNP})\cdot \\
&p_r(\mathrm{S{-}C}_{\texttt{gap}}(\textit{bought})|\mathrm{SBAR},\textit{that},\mathrm{WHNP},\{\mathrm{S{-}C},\texttt{gap}\})\cdot \\
&p_r(\texttt{STOP}|\mathrm{SBAR},\textit{that},\mathrm{WHNP},\{\})\cdot p_l(\texttt{STOP}|\mathrm{SBAR},\textit{that},\mathrm{WHNP},\{\})
\end{aligned} \tag{7.7}$$

Observe how the right subcat is extended with the `gap` feature and how this feature is removed when the child S–C(`gap`) is generated. Similarly, the probability for rule (4) is estimated as follows:

$$\begin{aligned}
&p_h(\mathrm{VP}|\mathrm{VB},\textit{bought})\cdot p_g(\mathbf{R}|\mathrm{VP},\textit{bought},\mathrm{VB})\cdot \\
&p_{lc}(\{\}|\mathrm{VP},\textit{bought},\mathrm{VB})\cdot p_{rc}(\{\mathrm{NP{-}C}\}|\mathrm{VP},\textit{bought},\mathrm{VB})\cdot \\
&p_r(\textbf{TRACE}|\mathrm{VP},\textit{bought},\mathrm{VB},\{\mathrm{NP{-}C},\texttt{gap}\}) \\
&p_r(\mathrm{NP{-}C}(\textit{week})|\mathrm{VP},\textit{bought},\mathrm{VB},\{\})\cdot \\
&p_r(\texttt{STOP}|\mathrm{VP},\textit{bought},\mathrm{VB},\{\})\cdot p_l(\texttt{STOP}|\mathrm{VP},\textit{bought},\mathrm{VB},\{\})
\end{aligned} \tag{7.8}$$

Note that the generation of **TRACE** removes both the `gap` feature and the {NP–C} requirements from the subcat frame.




```
(1) NP         -> NP    SBAR(gap)
(2) SBAR(gap)  -> WHNP  S-C(gap)
(3) S(gap)     -> NP-C  VP(gap)
(4) VP(gap)    -> VB    TRACE      NP
```

Figure 7.1: An example with `gap` features (Collins 1999, p. 176).

7.1.2 The parsing algorithm

Now, let us turn our attention to the parsing algorithm. Collins (1999) employs a CYK-style bottom-up chart parser to construct the parse trees. The most probable parse is found by dynamic programming augmented with beam search: subtrees for a given span are pruned from the search space if their probability falls below a threshold compared to the best probability edge for the span (cf. Goodman 1998). Two edges are the same with respect to dynamic programming, if

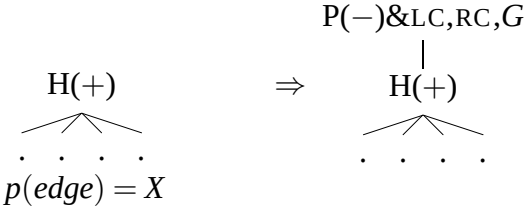
- they have the same nonterminal label (7.9)
- they span the same words
- their heads are the same
- both of them are either active or passive
- their left and right subcat frames are the same, respectively.

The parser constructs the syntactic structure through the following procedures:

`generate_unary`

This function generates a parent node for a passive edge and it also generates the corresponding left and right subcat frames as well as the gap variable, if necessary.

`generate_unary(edge E, label P)`



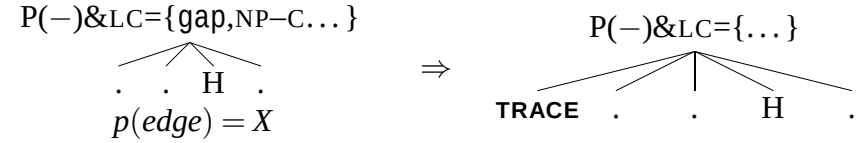
$p(edge) = X$

$$p(newedge) = X \cdot p_h(H|P,h) \cdot p_g(G|P,h,H) \cdot p_{lc}(\mathrm{LC}|P,h,H) \cdot p_{rc}(\mathrm{RC}|P,h,H)$$

`generate_trace`

This procedure is responsible for generating traces as either a left or a right modifier, provided the subcat frames allow gaps. If a trace is generated, an NP–C and the `gap` feature is removed from the subcat frame.

`generate_trace(edge E)`



$$p(newedge) = X \cdot p_l(\textbf{TRACE}|P,h,H,\mathrm{LC})$$

`generate_TOP`

This function is very similar to `generate_unary`, but it also checks whether the edge spans the whole sentence.

`generate_STOP`

This function generates `STOP` symbols on both sides, provided the subcat frames are empty.

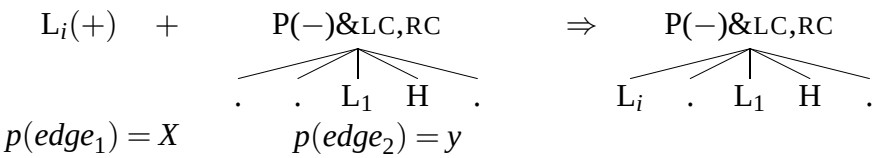
`generate_STOP(edge E)`

$$p(newedge) = X \cdot p_l(\texttt{STOP}|P,h,H,\mathrm{LC}) \cdot p_r(\texttt{STOP}|P,h,H,\mathrm{RC})$$

`join_2_edges`

This function attaches a passive edge as a child to an active one, while checking whether the edge fits the subcat frame. The function comes in three versions depending on whether the head is to the left or to the right and, if it is to the left, whether there is coordination or not.

`join_2_edges(edge E1, edge E2)`



$$p(newedge) = X \cdot Y \cdot p_l(L_i(l_i)|P,h,H,\mathrm{LC})$$

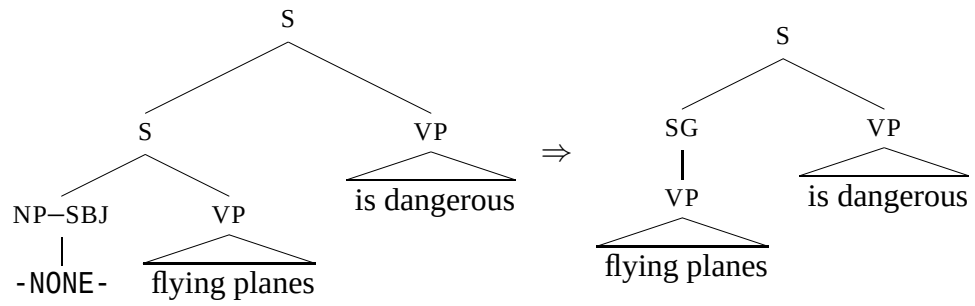


Figure 7.2: The treatment of **PRO** subjects in Model 3 (Collins 1999, p. 184).

7.1.3 Discussion

Although Model 3 only deals with **WH–NP** traces explicitly, Collins (1999) introduces another mechanism to handle **PRO–NP**s and **NP–NP**s in subject position. Whenever a clause has an empty subject, the EE is removed and the subjectless clause gets an SG label instead of the original S label (cf. Figure 7.2). In principle, we could use the SG labels to detect **PRO** subjects. Note, however, that this approach neutralises the distinction between controlled and uncontrolled **PRO**s: it is impossible to determine whether the EE has an antecedent or not.

The main problem with Model 3 is that it is designed to handle **WH–NP** traces only. Specifically, the following five problems should be remedied in order to generalise the framework for a wider range of EEs:

(i) Model 3 expects all EEs to have antecedents, though some do not have any (e.g. **PRO–NP**);

(ii) it cannot handle multiple *types* of EEs;

(iii) it does not allow multiple *instances* of EEs at a node;

(iv) it expects all EEs to be complements, though some are not (e.g. **WH–ADVP**);

(v) it cannot model EEs with dependents, for example **COMP–...**.

Figure 7.3 illustrates the case of an EE having no antecedent, where there is no need for threading the `gap` feature (this representation is very similar to the SG-notation). Figure 7.4 shows examples with several different EE-types. Observe the node label $S_{WH-NP+NP-NP}$ representing multiple instances of gaps. Model 3 cannot handle these cases at all: it does not allow more than one `gap` variable on a nonterminal. Moreover, it uses only one type to represent gaps, which prevents it from telling which phrase is the antecedent of the **NP–NP** and which one is the antecedent of the **WH–NP** trace (cf. Section 2.3).

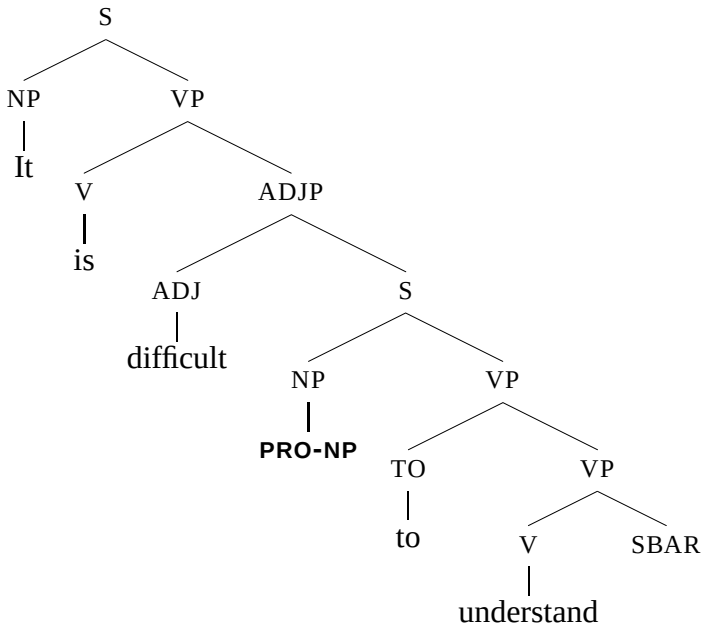


Figure 7.3: An EE with no antecedent.

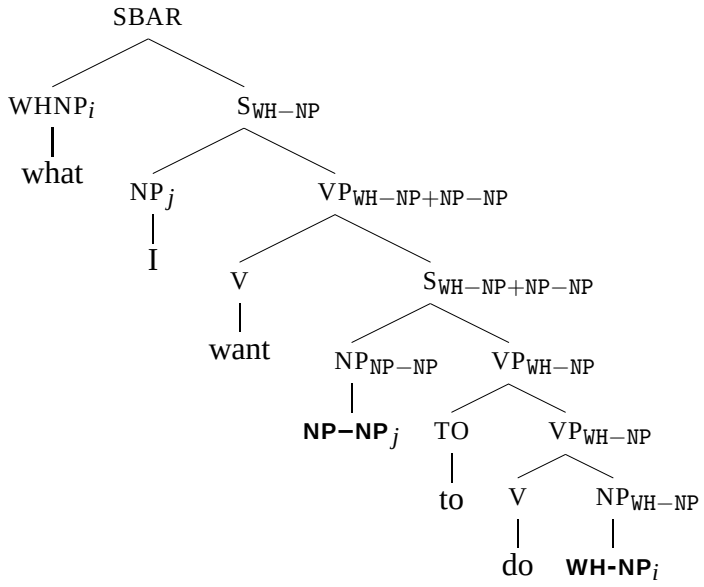


Figure 7.4: Multiple *types* and *instances* of EEs.

The lack of any distinction between adjunct and complement EEs is also a serious problem: when an adjunct EE is generated (e.g. **WH–ADVP**), there is no corresponding ADVP–C in the subcat frame. Clearly, this distinction should also be handled in a general model. Finally, EEs with complements are also problematic; this problem, however, is more fundamental than the others, because the parser is head-driven, that is, a head must be generated before any of its dependents are attached. In the case of non-head EEs, the parser has to generate an EE only if a head constituent requires it. In the case of empty heads, however, there is no such restriction: the parser should hypothesise (several types of) empty heads everywhere in the input string; this behaviour would tremendously increase the search space and slow down the parser.

7.2 Model 4

The problems with Model 3 call for a generalised treatment of EEs. In this section, we propose an extension of the model to accommodate several different EE-types. This model, Model 4, solves the first four problems discussed in the previous section. In Section 7.2.2, we discuss how some heuristics can overcome the fifth problem.

7.2.1 The probability model

Model 3 employs one `gap` feature to represent one instance of one EE-type. In the generalised model, we have to deal with several types and possibly multiple instances. In fact, gaps behave in a very similar way as complements: (i) the head determines the list of possible EEs, and (ii) whenever a child containing pending gaps is generated, the gaps are to be removed from the list. Therefore, we extend the model with the left and right "*gapcategorisation frames*" (*gapcat* frames): they are multisets of gaps required by the head. Gapcat frames are generated in a fashion similar to subcat frames and they replace the gap variable G of Model 3. We assume conditional independence between the generation of the subcat and the gapcat frames as well as between the probabilities for the left and the right gapcat frames. Therefore, in the new model, the probability of the RHS is estimated as:

$$
\begin{array}{ll}
p_h(H|P(h)) \cdot & \text{generate head} \\
p_{lgc}(\mathrm{LGC}^{(1)}|P(h),H) \cdot & \text{generate initial left gapcat} \\
p_{rgc}(\mathrm{RGC}^{(1)}|P(h),H) \cdot & \text{generate initial right gapcat} \\
p_{lc}(\mathrm{LC}^{(1)}|P(h),H) \cdot & \text{generate initial left subcat} \\
p_{rc}(\mathrm{RC}^{(1)}|P(h),H) \cdot & \text{generate initial right subcat} \\
\prod_{i=1}^{n+1} p_l(L_i(l_i)|P(h),H,\mathrm{LC}^{(i)},\mathrm{LGC}^{(i)}) \cdot & \text{generate left modifiers (and remove complements and EEs from subcat/gapcat frames)} \\
\prod_{j=1}^{m+1} p_r(R_j(r_j)|P(h),H,\mathrm{RC}^{(j)},\mathrm{RGC}^{(j)}) & \text{generate right modifiers (and remove complements and EEs from subcat/gapcat frames)}
\end{array}
\tag{7.10}
$$

In this model, we encode in each nonterminal label the types and the number of pending EEs dominated by the nonterminal (cf. Section 2.3). Whenever a child with pending EEs is generated, the corresponding EEs are removed from the gapcat frame. Generating a STOP symbol when the gapcat (and the subcat) frame is not empty receives null probability. Thus, for example, the probability of the rule

$$
\mathrm{VP}_{\mathtt{WH-NP}}(\textit{to}) \quad \rightarrow \quad \mathrm{TO}(\textit{to}) \quad \mathrm{VP}_{\mathtt{WH-NP}}(\textit{do}) \tag{7.11}
$$

in Figure 7.4 is estimated as follows:

$$
\begin{array}{l}
p_h(\mathrm{TO}|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to}) \cdot \\
p_{lgc}(\{\}|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to},\mathrm{TO}) \cdot \\
p_{rgc}(\{\mathtt{WH-NP}\}|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to},\mathrm{TO}) \cdot \\
p_{lc}(\{\}|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to},\mathrm{TO}) \cdot \\
p_{rc}(\{\mathrm{VP{-}C}\}|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to},\mathrm{TO}) \cdot \\
p_l(\mathtt{STOP}|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to},\mathrm{TO},\{\},\{\}) \cdot \\
p_r(\mathrm{VP{-}C}_{\mathtt{WH-NP}}(\textit{do})|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to},\mathrm{TO},\{\mathrm{VP{-}C}\},\{\mathtt{WH-NP}\}) \cdot \\
p_r(\mathtt{STOP}|\mathrm{VP}_{\mathtt{WH-NP}},\textit{to},\mathrm{TO},\{\},\{\})
\end{array}
\tag{7.12}
$$

Observe how the generated VP complement removes items from both the subcat and the gapcat frames. The following example shows how the probability is calculated when an EE is passed down on the head:

$$
\mathrm{S}_{\mathtt{WH-NP}}(\textit{want}) \quad \rightarrow \quad \mathrm{NP}(\textit{I}) \quad \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}(\textit{want}) \tag{7.13}
$$

$$
\begin{aligned}
&p_h(\mathrm{VP}_{\mathtt{WH-NP+NP-NP}} | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}) \cdot \\
&p_{lgc}(\{\} | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}, \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}) \cdot \\
&p_{rgc}(\{\} | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}, \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}) \cdot \\
&p_{lc}(\{\mathrm{NP{-}C}\} | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}, \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}) \cdot \\
&p_{rc}(\{\} | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}, \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}) \cdot \\
&p_l(\mathrm{NP{-}C}(I) | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}, \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}, \{\mathrm{NP{-}C}\}, \{\}) \cdot \\
&p_l(\mathtt{STOP} | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}, \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}, \{\}, \{\}) \cdot \\
&p_r(\mathtt{STOP} | \mathrm{S}_{\mathtt{WH-NP}}, \textit{want}, \mathrm{VP}_{\mathtt{WH-NP+NP-NP}}, \{\}, \{\})
\end{aligned}
\tag{7.14}
$$

Note the empty gapcat frames and the difference between the number of gaps on the parent and the head-child: this indicates that the antecedent of one of the EEs (**NP–NP**) should be deposited among the children of the parent (NP(I) in this case). As a final example, the probability for the rule:

$$
\mathrm{S}\textit{(to)} \quad \rightarrow \quad \textbf{PRO–NP} \quad \mathrm{VP}\textit{(to)} \tag{7.15}
$$

in Figure 7.3 is calculated as:

$$
\begin{aligned}
&p_h(\mathrm{VP} | \mathrm{S}, \textit{to}) \cdot \\
&p_{lgc}(\{\mathtt{PRO-NP}\} | \mathrm{S}, \textit{to}, \mathrm{VP}) \cdot \\
&p_{rgc}(\{\} | \mathrm{S}, \textit{to}, \mathrm{VP}) \cdot \\
&p_{lc}(\{\mathrm{NP{-}C}\} | \mathrm{S}, \textit{to}, \mathrm{VP})) \cdot \\
&p_{rc}(\{\} | \mathrm{S}, \textit{to}, \mathrm{VP}) \cdot \\
&p_l(\textbf{PRO–NP} | \mathrm{S}, \textit{to}, \mathrm{VP}, \{\mathrm{NP{-}C}\}, \{\mathtt{PRO-NP}\}) \cdot \\
&p_l(\mathtt{STOP} | \mathrm{S}, \textit{to}, \mathrm{VP}, \{\}, \{\}) \cdot \\
&p_r(\mathtt{STOP} | \mathrm{S}, \textit{to}, \mathrm{VP}, \{\}, \{\})
\end{aligned}
\tag{7.16}
$$

In this case, the right gapcat frame is not empty, although there is no `gap` feature on the parent nonterminal: the new model allows the generation of gaps with such parents as well. Furthermore, observe how the generation of the **PRO–NP** trace removes both the complement NP–C and `PRO-NP` requirements from the corresponding frames. Were the EE an adjunct (e.g. a **WH–ADVP**), the subcat frame would not have changed: the new model allows manipulating the subcat and the gapcat frames independently.

7.2.2 The parsing algorithm

Although the probability model has changed considerably, the parsing algorithm requires only minor modifications to accommodate the new model. First, on each

edge, we maintain two additional multisets: the left and the right gapcat frames. To decrease the sparsity of the data, we distinguish only six different EE-types in the gapcats (as opposed to the 44 possible types): **NP–NP**, **WH–NP**, **PRO–NP**, **TOP–S**, **WH–ADVP**, and **OTHER**, the latter category containing all the EEs which are not in the other five classes. With respect to the dynamic programming, we extend the notion of equivalent edges to incorporate gapcat frames: two edges do not compete with each other if their gapcat frames differ (cf. 7.9). Furthermore, we make use of a table containing which items each nonterminal removes from the gapcat frame (e.g. $S_{\texttt{WH-NP+NP-NP}}$ removes a **WH–NP** and an **NP–NP** item).

One EE-type can show various different syntactic behaviour: even if an EE occurs mostly in complement positions, sometimes it is an adjunct. In order to allow for such a variation, we internally distinguish these EEs from each other: for the parser (and, in fact, for the probability model as well) an adjunct EE is different from a complement EE, even if their types are the same (e.g. **WH–ADVP**).

In the parsing algorithm itself, all functions should be modified to accommodate the left and right gapcat values in the probability model. Four functions require further minor adjustments. First, the function `generate_unary` should also generate the gapcats; the update rule for the inside probability changes to:

$$\begin{aligned} p(newedge) = & p(oldedge) \cdot p_h(H|P,h) \cdot \\ & p_{lgc}(\text{LGC}|P,h,H) \cdot p_{rgc}(\text{RGC}|P,h,H) \cdot \\ & p_{lc}(\text{LC}|P,h,H) \cdot p_{rc}(\text{RC}|P,h,H) \end{aligned} \tag{7.17}$$

Second, `generate_STOP` is modified to test whether the gapcat frames are empty. Third, `generate_trace` should generate all EE-types: in each call it loops through all (internal) EE-types and attempts to add each in turn to the original edge. It also consults the table mentioned above to remove complement EEs from the subcat frame. Finally, whenever a passive edge with pending gaps is combined with an active edge, `join_2_edges` removes the pending gaps from the gapcat frame of the active edge.

Notes

Although Model 4 is able to handle a wider range of EEs amounting to 68% of all EE-tokens, it cannot deal with all types. The missing cases are (i) pseudo-attachment and ellipsis, (ii) empty complementisers (**COMP–...**), and (iii) empty units (**UNIT**). The proper treatment of pseudo-attachment and ellipsis is beyond the scope of the present dissertation; fortunately, they only account for approximately 4% of the EE-tokens.

The other two types, however, are much more frequent: they amount to 28% of all EEs. Recall that the reason for not integrating them into the parsing model

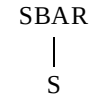
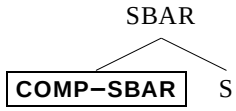
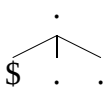
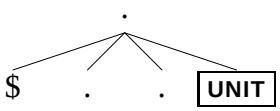
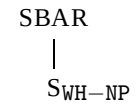
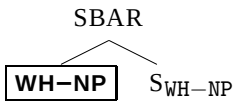
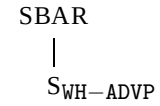
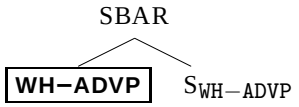
EE type	structure without EE	structure with EE
COMP–SBAR		
UNIT		
COMP–WHNP		
COMP–WHADVP		

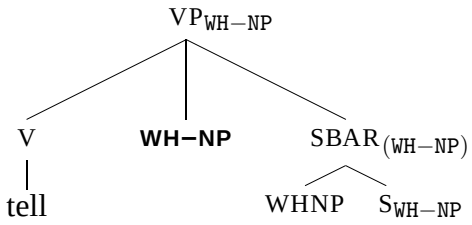
Table 7.1: Detecting empty complementisers and units.

is that these EEs are heads, i.e., the context-free rules in a head-driven parsing regime cannot impose any restrictions on generating them. However, the majority of such EEs can be easily found in a post-processing step, once the phrase structure is built: Johnson (2002) reports 96% and 92% F-score for detecting empty SBAR complementisers (**COMP–SBAR**) and empty units (**UNIT**) when the correct parse tree is known. Table 7.1 shows when and how a parse tree should be modified in order to accommodate these two types, which amount to 24% of all EEs. Our heuristics achieves 94.6% F-score for detecting **UNIT**s when knowing the perfect tree, whereas the F-score for **COMP–SBAR** is 97.8%.

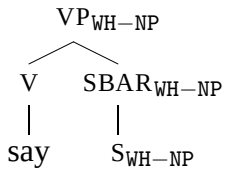
The majority of the remaining 4% is **COMP–WHNP**s and **COMP–WHADVP**s. Due to the distinction between different types of `gap+` variables on the nonterminal labels, these two types are also easy to detect (cf. Table 7.1): our heuristics finds them with 100% F-score, when applied to the perfect trees with the gaps removed. Observe that our technique to handle these gaps is essentially the same as the pattern matching approach proposed by Johnson (2002).

Finally, note the difference between Collins's and our approach for threading the `gap` variables: Collins (1997, 1999) threads the gaps one level higher, i.e., he also marks the SBAR nonterminal as having a pending **WH–NP** (compare Figures 7.5 and 7.4). There are several reasons why we do not adopt his approach. First, such an approach would defeat the parsing algorithm: now we can remove the pending gaps from the gapcat frames, whenever we encounter a nonterminal

(a) *Who did you tell what you don't like?*



(b) *Who did you say you don't like?*



(c) *Who did you tell that you don't like her?*

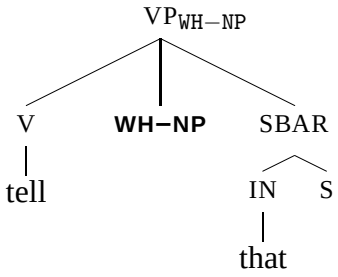


Figure 7.5: Difficult distinctions for the gap-threading approach of Collins (1997, 1999). In (a), the gap+ variable in brackets indicates where our gap-threading approach differs from Collins's one: in this case we would not have a gap+ variable on the nonterminal.

with gap features. Had we threaded the gaps one level higher, we could not distinguish SBARs with real pending EEs (in case of long WH- or NP-movement) and the ones that should deposit their traces on their daughters.

Second, adopting Collins's approach would unnecessarily complicate the antecedent recovery algorithm (cf. Section 4.1): it would require the algorithm to look one level higher, to the grandparent node, to decide whether an EE should or should not be deposited. For example, when depositing a **WH–NP** in Figure 7.4, we would have to check whether the parent of the SBAR constituent has a gap variable or not. Figure 7.5 shows two cases which would be difficult to distinguish. In (a) we have two **WH–NP**s, both of them participating in short movement; (b) contains only one such EE in long movement. In order to recover the antecedent properly, it is not enough to look at the gap variables on the nonterminals: the algorithm should examine the parse tree more carefully. In our gap threading approach, the SBAR constituent does not contain a gap variable in Figure 7.5a, hence the two cases are easy to tell apart.

The final problem with Collins's gap-threading approach concerns the probability estimation as well as the linguistic insights: the SBAR constituents in Figures 7.5a and 7.5c behave in the same way with respect to their heads, therefore their probability estimation should be the same.

7.3 Experiments

We trained and tested Model 4 using the same training and test data as in previous chapters. The input to the parser is POS-tagged text with the perfect tags from the Treebank. The parser is set up to insert EEs; henceforth, we refer to this approach as the lexicalised INSERT model. The EEs the parser is not able to handle are detected using the heuristics discussed above.

Apart from standard NLD-related metrics (EE detection, antecedent recovery, and HD-relations), we measure parsing time, the number of constructed edges, the size of the chart and parse accuracy. Table 7.2 summarises the results. We also compare Model 4 to the preprocessing approach using the lexicalised parser (cf. Section 6.2) and to Johnson's (2002) post-processing approach.

In summary, Model 4 proves to be feasible: it does not suffer from the problems we experienced in the case of the unlexicalised parser (Section 4.3), although it needs more time and explores a larger search space than the preprocessing approach. Moreover, the INSERT model is outperformed by the trace tagger on detecting EEs by a margin of $2.5 - 3\%$. On the other hand, Model 4 achieves the same accuracy for antecedent recovery. PARSEVAL scores also improve but are still lower than for Model 2.

		NOTRACE (Model 2)	TAGGER (Model 2)	INSERT (Model 4)
EE-det.	all	–	79.6	76.7
(F-score)	**WH/PRO/TOP**	–	79.6	77.3
ANTE-rec.	all	–	74.2	73.2
(F-score)	**WH/PRO/TOP**	–	72.3	72.3
HD accuracy	**WH/PRO/TOP**	–	71.3	72.9
Time (s/sent)		1.8	1.6	2.9
Chart (edges/sent)		14k	13k	25k
Fully constructed edges (per sent)		434k	331k	694k
PARSEVAL	≤ 100	88.0	86.3	87.3
(F-score)	≤ 40	88.5	87.1	87.9
Surface deps.		90.7	88.6	89.2
All deps.		88.2	88.2	88.6

Table 7.2: Finding NLDs with a lexicalised parser; comparison with the TAGGER and the NOTRACE.

EE	Freq.	Prec.	Rec.	*F*-score		
		Here	Here	Here	Tagger	Johnson
LABELLED ALL	3864	80.8	73.0	76.7	**79.6**	–
UNLABELLED ALL	3864	93.4	84.3	**88.6**	87.5	–
WH/PRO/TOP	2643	80.3	74.5	77.3	**79.6**	–
NP–NP	1148	83.5	78.8	81.1	**84.0**	–
COMP–SBAR	545	80.9	83.1	82.0	**84.6**	84.0
WH–NP	508	83.5	80.5	82.0	82.7	81.0
PRO–NP	477	65.0	60.8	62.8	**70.7**	–
UNIT	388	94.4	91.2	92.8	**96.5**	92.0
TOP–S	277	95.0	96.4	**95.7**	91.0	88.0
WH–ADVP	171	63.9	53.8	58.4	58.2	56.0
COMP–WHNP	107	51.2	41.2	45.6	**48.2**	47.0

Table 7.3: EE-detection results for Model 4 and comparison with Johnson (2002) and the trace tagger.

7.4 Discussion

One of the principal results of the experiment with Model 4 is that it does not show the same behaviour as the unlexicalised INSERT model. It is indeed feasible to detect EEs with a *lexicalised* parser: although the parser explores a larger search space and requires more time to parse, it only fails in the case of 11 sentences ($< 0.5\%$) and parsing time is still relatively low. This shows that lexicalisation is very important for restricting the search space when it comes to detecting NLDs (cf. also Johnson and Kay 1994). Note, however, that Model 4 only inserts a subset of EEs, namely the ones that do not have dependents. Is it possible that EEs with dependents are the major cause of the infeasible behaviour of the unlexicalised INSERT model? The answer turns out to be negative: when the unlexicalised model tries to insert **WH** and **PRO** traces only, it still fails in the case of 35.1% of the sentences. Therefore, we conclude that lexicalisation is the most prominent factor for the success of both Model 4 and the trace tagger.

Although Model 4 proves to be feasible, it still requires around 80% more time to arrive at its solutions than the preprocessing approach. The slowdown is clearly due to the larger search space it has to explore: the number of both the fully constructed edges and the edges on the chart is twice as high as in the case of the preprocessing approach.

EE	Frequency	Prec.	Rec.	*F*-score		
		Here	Here	Here	TAGGER	Johnson
ALL	3864	77.2	69.7	73.2	**74.3**	68.0
WH/PRO/TOP	2643	75.1	69.7	72.3	72.3	–
NP–NP	1148	74.5	70.3	**72.3**	71.5	60.0
COMP–SBAR	545	80.9	83.2	82.0	**84.6**	84.0
WH–NP	508	82.4	79.5	81.0	80.7	81.0
PRO–NP	477	65.0	60.8	62.8	**70.7**	50.0
UNIT	388	94.4	91.2	92.8	**96.5**	92.0
TOP–S	277	87.5	88.8	**88.2**	81.4	87.0
WH–ADVP	171	61.8	52.0	**56.5**	53.0	56.0
COMP–WHNP	107	51.2	41.1	45.6	**48.2**	47.0

Table 7.4: Antecedent recovery results with Model 4 and comparison with Johnson (2002) and the preprocessing approach using Model 2.

Moreover, the parser is not able to detect EEs as reliably as the tagger: the EE-detection scores for Model 4 are $2.5-3\%$ worse than those of the tagger. A closer look at the results for individual EE-types (Table 7.3) shows that the tagger is better at distinguishing **PRO–NP**s and **NP–NP**s. On the other hand, the parser detects topicalised sentences (**TOP–S**) more accurately than the tagger. This shows that the parser can make use of the whole sentence structure when deciding whether to insert an EE or not.

Interestingly, the lexicalised INSERT model, although less accurate at EE detection, performs the same on the antecedent-recovery metrics as the TAGGER model, and outperforms it on the HD metrics. A look at the individual results (Table 7.4) casts some light on the situation: although the tagger finds more EEs, the parser is better at recovering the antecedents for its own hypotheses. Indeed, antecedent recovery scores of the parser are higher for all EE-types with antecedents. Another benefit of Model 4 over the TAGGER model is its higher parsing accuracy: it improves on the preprocessing architecture by 1% *F*-score according to the PARSEVAL metrics. Furthermore, Model 4 fails to parse only 11 sentences, whereas the preprocessing approach cannot produce an analysis for 24 sentences.

Observe that Model 4 achieves somewhat lower scores on the PARSEVAL metric as well as on detecting surface dependencies than Model 2. We attribute this behaviour to sparser data: since the size of the nonterminal set is 7.1 times larger, we have to estimate more parameters from the same amount of data. Furthermore, as we argued in the previous chapter, EEs, due to their low frequency, restrict the

set of available rules. In the worst case, the rule necessary to complete the correct structure does not appear in the training set. When evaluated on all dependencies, however, Model 4 does outperform Model 2.

Comparing Models 3 and 4 reveals that Model 4 is around 3.7% less accurate at detecting **WH–NP** traces, and the antecedent recovery score is also 3.1% lower for this particular EE. The explanation is again data sparseness; the price of detecting all EEs is that we lose some accuracy on certain constructions when compared to specialised models, because we have to partition our data into smaller sets, which makes the probability estimation less reliable.

Finally, note that Model 4 makes an important simplifying assumption, as far as bilexical dependencies are concerned: it does not model these dependencies for NLDs. In particular, the lexical head of an EE is taken to be *TRACE* instead of the head of the co-indexed constituent. There is a good reason for this simplification: generally, the real headword is not yet available when the parser generates an EE, therefore a bottom-up parser cannot take it into account (but see Hockenmaier 2003b). Fortunately, this simplification causes no problem in practice: Gildea (2001) shows that incorporating bilexical information improves Collins's (1997) model by only 0.5% (in terms of the PARSEVAL score).

In summary, Model 4 has both positive and negative characteristics. It is slower than the preprocessing approach and less accurate at detecting EEs. This suggests that it could benefit from consulting the trace tagger. On the other hand, it is more robust and can recover antecedents more reliably that the TAGGER model. Moreover, the parse trees are more accurate according to the PARSEVAL measures. We claim that the reason for this behaviour is the rigid combination method of the TAGGER model. Can we combine the trace tagger and the parser in a more appropriate way to bring the strengths of both systems together? In the next chapter, we present a novel probabilistic architecture which achieves this goal.

7.5 Related work

There is relatively little work in the parsing literature specialising on handling NLDs; most research in this area is closely tied up with developing efficient parsing algorithms for deeper grammar formalisms, such as HPSG, LFG, TAG or CCG – the review of these approaches is beyond the scope of the present discussion. In general, however, the major bottleneck in using these formalisms for large-scale parsing is their limited coverage and lack of robustness (Uszkoreit 2002). They usually can only parse a subset of our test set, which hinders comparison with the present work.

To our knowledge, the only probabilistic approaches to explicitly handle a wide range of NLDs with a parser is presented by Clark *et al.* (2002) and Hock-

enmaier (2003b), who use Combinatorial Categorial Grammar as their underlying linguistic framework. Interestingly, their models face similar problems as ours: they are much slower than the model not handling EEs explicitly (Hockenmaier and Steedman 2002b) and they also achieve slightly lower scores for surface dependencies. Unfortunately, both papers use a theory-dependent definition of non-local dependency, which is not compatible with our definition. In particular, they treat auxiliary constructions and coordination as non-local dependencies, which forbids straightforward comparison.

Summary

In this section, we have proposed a generalisation of Collins's (1997) Model 3 which can handle a wider range of EEs. We have also discussed reliable heuristics to handle the majority of the remaining EEs not integrated into the parser. The lexicalised model does not suffer from the efficiency and accuracy problems of the unlexicalised model, showing the importance of lexicalisation for the task. We have compared the new model with the lexicalised TAGGER approach of Section 6.2 and found that it outperforms the preprocessing approach on recovering antecedents, although it does worse on detecting EEs. It is also more robust and more accurate at detecting phrase structure than the TAGGER model, although it does not reach the robustness and accuracy of Model 2 of Collins (1997, 1999), due to sparser data and a much larger search space.

Chapter 8

A general architecture for combining external modules with a parser

In the previous section, we observed an interesting situation: although the TAGGER model outperforms the lexicalised INSERT model by $2.5-3\%$ on EE detection, it does no better at recovering antecedents for the semantically interesting cases. Such a situation arises due to the lack of proper communication between the tagger and the parser: when the tagger makes a mistake or suggests an EE which is hard to incorporate into the parse tree, the parser gets confused or even fails. The principal problem is the lack of a closer (probabilistic) integration of the two modules: since the parser regards the output of the tagger as 100% reliable, it cannot override the decisions the tagger makes.

In this chapter, we propose a general probabilistic architecture for combining different external modules with a parser. The modules return their *k*-best hypotheses with their probability distribution and the parser takes this guidance into account in order to arrive at an optimal solution. This approach has not only the benefit of increasing accuracy, but it also speeds up the parser by reducing the search space it has to explore.

After discussing the motivations for designing such a combined architecture, we develop the probability model in Section 8.2.1. In Section 8.2.2, we present the required modifications to the parsing algorithm of Section 7.2. To evaluate the framework, we present two experiments: first, in Section 8.3, we explore how the model fares on detecting NLDs. Second, we illustrate the success of the approach when combining an NP chunker with the parser.

8.1 Motivation

Stochastic parsing models assign probabilities to syntactic structures, typically to trees (or, in general, to directed graphs). The task of a probabilistic parser, then, is to find the maximum probability structure T that is compatible with the input:

$$\underset{T \in \mathcal{T}(S)}{\operatorname{argmax}}\, p(T) \tag{8.1}$$

where $\mathcal{T}(S)$ is a set of trees (structures) compatible with the input sentence S.

The problem is, however, that we do not know the probability $p(T)$. Indeed, this probability depends on innumerable factors, both linguistic and extra-linguistic ones. For example, it might depend on the topic of the conversation, our mood, our interlocutor, even on the weather outside (if it is raining, we tend to use the word *sun* more often). Even if we assume that only linguistic factors are at play, there are many more of them that we can reasonably incorporate into the model: the data (especially the relatively small amount of annotated data) is too scarce for deriving reliable probability distributions (*sparse data problem*). Therefore, a parsing model cannot use the "true" probability $p(T)$, but has to *estimate* it from a corpus.

The quality of the parsing model greatly depends on the quality of this estimation. Parsing models, however, are general-purpose models: they perform POS-disambiguation, chunking, prepositional phrase attachment, word sense disambiguation (at least as far as subcat frames are concerned), etc. While it is reasonable to solve all these tasks in one architecture, specialised models might use a different, more specific estimation technique (and even more data) and, thus, might have a better estimation for parts of the structure. For example, our trace tagger estimates the sites of NLDs more reliably. Similarly, a module specialised in NP chunking might outperform the parser in finding chunks. The parser would potentially benefit from incorporating external information from such modules.

Another potential benefit of combining external modules and a parser is higher parsing speed. As we have seen in Section 4.3.1, the parser is much faster if it is informed about the sites where NLDs occur. The speedup is the result of a much smaller search space: the parser no longer has to postulate EEs basically everywhere, it only has to explore the fraction of the search space that is compatible with the hypotheses of the trace tagger. Similarly, one would expect higher speed when combining the parser with an NP chunker: the parser would only have to build structures which do not cross NP boundaries.

In this chapter, we develop a novel architecture for combining external modules with a statistical parser, achieving the two goals discussed above: the combined system is more accurate and faster than the parser alone. In the design, we aim at satisfying the following criteria:

flexibility the parser may select suboptimal solutions according to the probability model of the external module (second/third/...-best hypotheses);

independence if one module changes, the other does not have to be retrained;

generality a wide range of external modules can be combined with the parser; the only requirement is that they return their *k*-best hypotheses, accompanied with their estimated probabilities;

simplicity the parser needs only minor modifications allowing straightforward implementation.

In the next section, we describe the architecture and the modifications in the parsing algorithm required for accommodating the changes. The subsequent sections present two experiments with the new system.

8.2 The architecture

In this section, we develop a general probabilistic architecture which combines several external modules with a parser. The modules are required to return their *k*-best hypotheses for the input, associated with their probabilities. In order to keep the presentation simple, we describe how *one* such module can be integrated with the parser: the generalisation of the approach to several modules is straightforward. First, we focus on the probability model, then we describe how a (bottom-up) parser can be implemented to accommodate the combination.

In this section we use the concept of *(in)compatibility* of the hypotheses provided by the external module and by the parser. The actual definition of compatibility is task dependent (cf. Sections 8.3 and 8.4), but in general, a hypothesis of the external module is incompatible with a given subtree if they exclude each other: there is no possible parse tree (independent of the grammar) which can accommodate both. For example, the hypotheses of the trace tagger and the parser are incompatible, if the parser predicts an EE where the trace tagger does not (or vice versa).

8.2.1 The probability model

In what follows, we use $p(T)$ to represent the "true" probability of a parse tree T, that is, the probability given the state of the world. As we argued in the previous section, this probability can only be estimated in practice; $p_p(T)$ denotes the probability estimation according to the parser's probability model. Our task

is to improve on this estimation, making use of the information the external module provides; $p_c(T)$ denotes the (improved) estimation according to the combined architecture.

Our first observation is that the maximal probability tree also maximises the joint probability $p(T,S)$ for a given sentence S and, since S is fixed, the conditional probability $p(T|S)$ as well ($p(T,S) = p(S)p(T|S)$):

$$\operatorname*{argmax}_{T\in\mathcal{T}(S)} p(T) = \operatorname*{argmax}_{T\in\mathcal{T}(S)} p(T,S) = \operatorname*{argmax}_{T\in\mathcal{T}(S)} p(T|S) \tag{8.2}$$

where $\mathcal{T}(S)$ denotes the set of parse trees for the given sentence S. In practice, the parser tries to find the tree with maximal probability according to its own estimate, that is, the tree which maximises $p_p(T|S)$.

Now, suppose that $\mathcal{X}$ is a partition of the event space, that is, it is a set of mutually exclusive events that cover the whole probability mass:

$$\sum_{x\in\mathcal{X}} p(x) = 1, \text{ and } p(x_i,x_j) = 0 \text{ for all } x_i \neq x_j (\in \mathcal{X}). \tag{8.3}$$

Then, using the *law of total probability*, $p(T|S)$ can be rewritten as

$$p(T|S) = \sum_{x\in\mathcal{X}} p(T,x|S) = \sum_{x\in\mathcal{X}} p(x|S)p(T|x,S) \tag{8.4}$$

We can regard the elements of $\mathcal{X}$ as all possible analyses for the sentence according to the external approach. For example, $\mathcal{X}$ can stand for all possible EE sequences or all possible NP-chunk sequences. At a first sight, it is not clear how such a choice could help us: the set $\mathcal{X}$ can be very large and we do not know the probabilities $p(x|S)$ and $p(T|x,S)$. However, it is reasonable to assume that our external module returns a fairly reliable estimation for $p(x|S)$, at least for the k-best hypotheses (henceforth we use $p_e(x|S)$ to indicate the probability distribution according to the probability model of the external module). We also take k to be large enough, so that that the bulk of the probability mass is distributed among the k-best hypotheses of the external module, the set of which is henceforth referred to as $\mathcal{X}_k$. Then, we can estimate $p(T|S)$ by replacing the sum of all possible external analyses by the sum of the k-best analyses in Eq. (8.4):

$$p(T|S) \approx \sum_{x\in\mathcal{X}_k} p_e(x|S)p(T|x,S) \tag{8.5}$$

In the above formula, however, there is still an unknown term, $p(T|x,S)$, which seems to be even more difficult to estimate than the original term $p(T|S)$. Note, however, that certain trees are not compatible with x, that is, T and x cannot co-occur. For instance, if T contains an empty element where it is not hypothesised

by the external module, then the two hypotheses are incompatible, thus, the corresponding probability $p(T|x,S)$ is 0.

Now, the only remaining question is: what to do when the structure is compatible with x? Without further simplifying assumptions, we exacerbate the problem: the probability $P(T|x,S)$ is more difficult to estimate since a corpus annotated with phrase structure trees does not suffice: we have to take into account how the external module analyses this corpus. As a consequence, the sparse data problem is further aggravated. In order to remedy the situation, we propose to use the parser's probability estimate $p_p(T|S)$ whenever the tree is compatible with x, that is, we assume independence between the parser and the external module, whenever T and x are compatible:

$$p(T|x,S) \approx \begin{cases} 0 & \text{if } T \text{ is incompatible with } x \\ p_p(T|S) & \text{if } T \text{ is compatible with } x \end{cases} \tag{8.6}$$

With this approximation, Eq. (8.5) can be considerably simplified. First, we only have to take into account the hypotheses $x \in \mathcal{X}$ of the external module compatible with the parse tree T; $\mathcal{C}(T)$ denotes the set of these hypotheses. In the case of these hypotheses, however, the estimation in Eq. (8.5) contains a common term $p_p(T|S)$. Therefore, it can be stated in a much simpler form: we define the probability $p_c(T|S)$ of a parse tree T given the sentence S according to the combined probability model as

$$p_c(T|S) \stackrel{\text{def}}{=} \frac{1}{Z(S)} \cdot p_p(T|S) \cdot \sum_{x \in \mathcal{C}(T) \cap \mathcal{X}_k} p_e(x|S) \tag{8.7}$$

where

$$Z(S) \stackrel{\text{def}}{=} \sum_{T \in \mathcal{T}(S)} \left(p_p(T|S) \cdot \sum_{x \in \mathcal{C}(T) \cap \mathcal{X}_k} p_e(x|S) \right) \tag{8.8}$$

is a normalising factor ensuring that p_c be a proper probability distribution given the sentence S.

One way to interpret Eq. (8.7) is to regard the external module as a weighted *filter* on the parser: it filters out parse trees that are incompatible with the module's hypotheses. Note that if all the hypotheses are compatible with a given tree, we get back the probability according to the original parsing model. Another way to look at the above equation is to treat the normalised sum as a *prior* encoding the knowledge of the external module.

As before, the parser searches for the syntactic structure maximising the conditional probability given the sentence. The difference, however, is that it takes the combined estimation p_c into account, instead of p_p. Since $Z(S)$ is fixed for a

given sentence S, the search criterion may be further simplified:

$$\operatorname*{argmax}_{T \in \mathcal{T}(S)} p_c(T|S) = \operatorname*{argmax}_{T \in \mathcal{T}(S)} \left(p_p(T|S) \cdot \sum_{x \in \mathcal{C}(T) \cap \mathcal{X}_k} p_e(x|S) \right) \tag{8.9}$$

An additional minor complication is that our parsing models presented in this dissertation are generative, that is, they return the joint probability $p_p(T,S)$, as opposed to the conditional probability $p_p(T|S)$. To get the conditional probability, one has to renormalise the joint distribution by a factor of $p_p(S)$, the calculation of which requires summing over all possible trees T yielding the sentence S, i.e., over each element of the set $\mathcal{T}(S)$, which might be very large. Fortunately, $p_p(S)$ is constant (given S), therefore, it does not affect the rank of a given tree. That is, $p_c(T|S)$ and $p_c(T|S) \cdot p_p(S)$ are maximised by the same tree T, hence we can use the joint probability $p_p(T,S)$ in the above formula.

In summary, in the new model, we search for the optimal syntactic structure $\hat{T}$, defined by

$$\hat{T} \stackrel{\text{def}}{=} \operatorname*{argmax}_{T \in \mathcal{T}(S)} \left(p_p(T,S) \cdot \sum_{x \in \mathcal{C}(T) \cap \mathcal{X}_k} p_e(x|S) \right), \tag{8.10}$$

where

- p_p is the probability distribution according to the parser's model,
- p_e is the probability distribution according to the external module,
- $\mathcal{T}(S)$ is the set of syntactic structures T that yield the sentence S,
- $\mathcal{X}_k$ is the set of the k-best hypotheses according to the external module, and
- $\mathcal{C}(T)$ is the set of external hypotheses compatible with the parse tree T.

8.2.2 The general parsing algorithm

In this section, we discuss the modifications the parsing algorithm requires to incorporate the changes to the probability model. As we shall see, the proposed architecture allows straightforward integration with a bottom-up parser. In what follows, we treat $p_p(T,S)$ as the *inside* probability of a subtree T.

First, note that the estimation in Eq. (8.7) can also be used for substructures, not only for fully specified trees. A second observation is that incompatibility is a monotonic relation: if a substructure is incompatible with an external hypothesis x, then every structure containing the given substructure is also incompatible with x. Specifically, if a substructure is incompatible with all external hypotheses,

it cannot be used in any trees, i.e., we can safely discard it from the search space. This is how the combined model achieves search space reduction.

Now, suppose that the bottom-up parser creates a new edge T combining $T_1, \ldots, T_n$, using the appropriate rule $N \rightarrow N_1 \ldots N_n$. The inside probability of this edge according to the new model is

$$p_c(T|S) = \frac{1}{Z(S)} \cdot p_p(T,S) \cdot \sum_{x \in \mathcal{C}(T) \cap \mathcal{X}_k} p_e(x|S) \tag{8.11}$$

where $Z(S)$ is a constant (given the sentence S) and $p_p(T,S)$ is the inside probability according to the old model, calculated as

$$p_p(T,S) = p_p(N \rightarrow N_1 \ldots N_n|N) \cdot \prod_{i=1}^{n} p_p(T_i,S) \tag{8.12}$$

Therefore, the only additional complication the novel estimate introduces is the calculation of the sum. In fact, once we know the set of compatible external hypotheses, $\mathcal{C}(T) \cap \mathcal{X}_k$, this calculation is trivial. Determining the compatibility set, on the other hand, might add a considerable amount of overhead, depending on the task. Note, however, a crucial property of the model: since incompatibility is monotonic, the set of compatible hypotheses with T is a subset of the compatibility set of T_i for all $i = 1 \ldots n$. That is, this set is a subset of the intersection of all compatibility sets corresponding to $T_1 \ldots T_n$:

$$\mathcal{C}(T) \subseteq \bigcap_{i=1}^{n} \mathcal{C}(T_i) \tag{8.13}$$

Consequently, we might reasonably expect that the parser has to determine compatibility for only a couple of hypotheses, and thus the overhead introduced by this calculation is compensated for by the filtering effect.

In our present implementation, we extended the parsing algorithm of Section 7.2 to incorporate the new combined architecture. On each edge, we maintain an additional data structure, the set of compatible hypotheses $\mathcal{C}(T)$, as a bitvector. This facilitates fast calculation of an upper bound for the compatibility set $\mathcal{C}(T) \cap \mathcal{X}_k$. Whenever the parser modifies a structure or builds a new one, it calls a function `check_compatible` which determines this set.

An important question to consider is when to call this function. In general, structure building routines in a statistical parser are fairly expensive and they hardly discard edges before they are completely built. Compatibility check, however, removes edges without building them entirely. Therefore, it should be executed as early in the derivation as possible, before additional calculations and

actual probability lookup (which turns out to be the most expensive operation of the parser). In this way it can exercise its pruning power to a greater extent.

Further optimisation can be achieved by noticing that not each operation can change the set of compatible hypotheses. Therefore, in these cases, there is no need for the expensive calculations. However, this optimisation step is specific to the given module; we will discuss it separately in Sections 8.3 and 8.4. The definition of equivalence of two edges with respect to dynamic programming is also task specific; we will return to it in the relevant sections.

In summary, we have shown that the new probability model is straightforward to integrate into the original PCFG parsing model. Although checking the compatibility of a subtree and an external hypothesis might be expensive, we predict that the parser performs this operation relatively infrequently, introducing only limited computational overhead. On the positive side, the new probability model allows discarding edges (even if they have high probability according to p_p), which reduces the search space and, as a consequence, parsing time. We expect this effect to counter the extra time needed for the calculations to determine the compatibility set.

8.3 Finding NLDs

In the previous section, we have outlined a general probabilistic architecture to combine external modules with a parser. Our main claim is that a parser benefits from the external information: the combined system is both faster and more accurate. To substantiate this claim, we test the new architecture on two tasks: on NLD detection and baseNP chunking. In the present section, we explore the combination of the trace tagger with the parser. In the next section, we use a simple baseNP chunker to guide the parser.

8.3.1 Setup

In the following experiments, we combine the trace tagger described in Chapter 5 and the parser of Section 7.2. The tagger returns its k best hypotheses; in the experiments below, we set $k = 3$, since larger k does not improve the accuracy of the tagger (cf. Section 5.1.3). Both the parser and the tagger are trained independently on the standard training set of the Treebank (Sections 02–21).

The parser is modified to accommodate the new architecture as described in the previous section. One important issue concerns the definition of compatibility. There are several more permissive definitions, but we take the strictest approach: a substructure T is compatible with an EE-sequence x, if and only if the EEs in T and x match exactly. Exact match is counted in a similar way as in the case of

Action	Compatibility?
`generate_unary`	NO
`generate_trace`	OPT
`generate_STOP`	NO
`generate_TOP`	YES
`join_2_edges`	YES

Table 8.1: Actions of the parser and compatibility check.

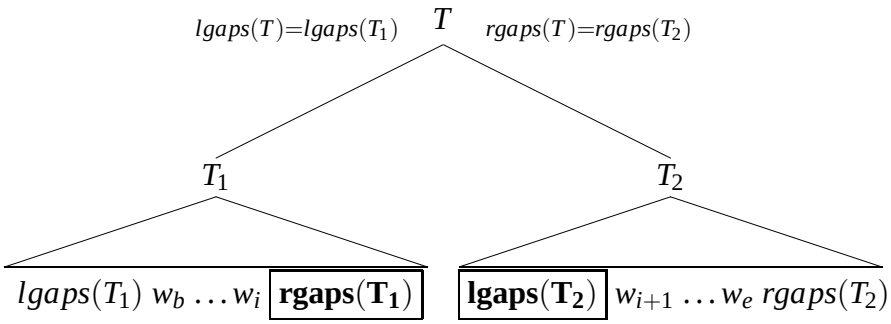


Figure 8.1: Checking compatibility when executing `join_2_edges`.

the evaluation metric for EE detection: EEs should occur between the same words with the same label. That is, the sets of EEs between two adjacent content words should be the same in T as in x. More lenient definitions could substitute exact match with subset relations.

Now, let us turn our attention to the actual implementation. Table 8.1 summarises the actions of the parser as presented in Section 7.2. How can we check whether a subtree (edge) T is compatible with a hypothesis using dynamic programming? Our invariant is that the compatibility of gaps inside this edge are verified. If we maintain this invariant throughout the derivation, gaps inside the sentence are checked by the end of the parsing process. After we constructed edges with the start symbol TOP that span the whole sentence, we also have to test whether the sets of gaps at the beginning and at the end of the sentence are compatible with the hypotheses of the trace tagger. This last step can be done when the function `generate_TOP` is executed.

None of actions of the parser, with the exception of `join_2_edges`, change the span of a subtree, hence they cannot affect the invariant. Consequently, there is no need to call the function `check_compatible` while executing these actions. On the other hand, when the parser joins two edges T_1 and T_2, gaps on the right boundary of the left edge ($rgaps(T_1)$) and on the left boundary of the right edge ($lgaps(T_2)$) are covered by the new edge T (Figure 8.1). Therefore, the compat-

ibility of these EEs should be verified, that is, the set of EEs occurring between words w_i and w_{i+1} according to some external hypothesis should be the same as $rgaps(T_1) \cup lgaps(T_2)$. If, for each edge, we maintain the two sets *rgaps* and *lgaps*, this test is trivial.

The parsing algorithm can be further optimised if we call a permissive version of the function `check_compatible` whenever a new EE is generated: the generated gap should be in the EE-set according to some external hypothesis. This test prevents the parser from generating EEs not compatible with any hypothesis at an early stage. This additional process, however, does not change the set of possible syntactic structures: edges covering incompatible EEs would be discarded later in the derivation during strict compatibility check. Table 8.1 summarises the above observations.

A final issue to address is the equivalence of two edges with respect to the dynamic programming. We say that two edges are equivalent if they behave in the same way concerning the context-free rules (cf. 7.9) as well as the function `check_compatible`. The latter entails that the sets of the left and right gaps, respectively, should be the same for the two edges. That is, two edges T_1 and T_2 are equivalent with respect to the dynamic programming, if the following conditions hold:

(8.14)

- T_1 and T_2 have the same nonterminal label;
- they span the same words;
- their heads are the same;
- both of them are either active or passive;
- their left and right subcat frames are the same, i.e., $lc(T_1) = lc(T_2)$ and $rc(T_1) = rc(T_2)$;
- their left and right gapcat frames are the same, i.e., $lgc(T_1) = lgc(T_2)$ and $rgc(T_1) = rgc(T_2)$; and
- *their left and right gapsets are the same, i.e.,* $lgaps(T_1) = lgaps(T_2)$ *and* $rgaps(T_1) = rgaps(T_2)$.

8.3.2 Experiments

In order to compare our architecture with the TAGGER and the INSERT models, we used the same settings as in Chapters 6 and 7. We measured the accuracy for EE detection and antecedent recovery, parsing time, the size of the chart and PARSEVAL-scores. The results are summarised in Table 8.2. On NLD-related tasks, the COMBINED model clearly outperforms both the TAGGER and the INSERT models by a considerable margin. When compared to Johnson's (2002)

		TAGGER	INSERT	COMBINED
EE det.	all	**79.6**	76.7	**79.6**
(F-score)	WH/TOP/PRO	79.6	77.3	**81.3**
ANTE rec.	all	74.3	73.3	**75.8**
(F-score)	WH/TOP/PRO	72.3	72.3	**75.8**
HD accuracy	WH/TOP/PRO	71.3	72.9	75.5
Time (s/sent)		**1.6**	2.9	2.4
Chart (edges/sent)		**13k**	25k	19k
Fully constructed edges (per sent)		**331k**	694k	469k
PARSEVAL	≤ 100	86.3	**87.3**	87.1
(F-score)	≤ 40	87.1	**87.9**	87.7
Surface deps.		88.6	89.2	**89.6**
All deps.		88.2	88.6	**89.2**

Table 8.2: Comparison of the TAGGER and INSERT models with the COMBINED model on finding NLDs.

results, the COMBINED model achieves approximately 14% higher F-score on antecedent recovery for the semantically important EEs (7.8% higher score for all EEs).

As predicted, the size of the search space lies between that of the TAGGER and of the INSERT models. Although PARSEVAL scores for the COMBINED model are still 0.2% lower than the scores for the standalone parser, the former system is more accurate at recovering surface dependencies.

We further investigated the pruning effects of the COMBINED model by varying the beam and plotting the accuracy for EE detection and antecedent recovery, as well as parsing time and the size of the chart (Figures 8.2–8.4).

8.3.3 Discussion

The most important result of the experiments is that the COMBINED architecture outperforms both the TAGGER and the INSERT models. Indeed, the parser manages to incorporate the information from the shallow module to improve its own accuracy. In particular, it outperforms the parser-only architecture on both EE

EE	Frequency	Precision		Recall		F-score	
		COMB	INS	COMB	INS	COMB	INS
ALL	3864	**82.5**	77.2	70.1	69.7	**75.8**	73.3
WH/PRO/TOP	2643	**81.9**	75.1	70.6	69.7	**75.8**	72.3
NP–NP	1148	**78.9**	74.5	71.6	70.3	**75.7**	72.3
COMP–SBAR	545	78.6	**80.9**	83.7	83.1	81.7	82.0
WH–NP	508	**91.6**	82.4	77.8	79.5	**84.1**	81.0
PRO–NP	477	**74.7**	65.0	**68.1**	60.8	**71.3**	62.8
UNIT	388	94.4	94.4	91.2	91.2	92.8	92.8
TOP–S	277	87.8	87.5	85.9	**88.8**	86.9	**88.2**
WH–ADVP	171	**85.9**	61.8	46.2	**52.0**	**60.1**	56.5
COMP–WHNP	107	**70.4**	51.1	35.5	**41.1**	**47.2**	45.6

Table 8.3: Antecedent recovery for the COMBINED and the INSERT models.

detection and antecedent recovery by around $3-4\%$, while HD accuracy is also 2.6% better.

Table 8.3 compares antecedent recovery scores for the COMBINED and INSERT models. The most important effect of the trace tagger is the increase in precision, which improves for almost all EEs. Indeed, the overall precision is 5% higher, and in the case of the semantically interesting EEs, the difference attains almost 7%. Recall, on the other hand, did not improve so dramatically, although it is a bit higher on average.

The increase in precision affects the semantically important EEs to the greatest extent. In particular, the scores for both controlled and uncontrolled **PRO**s improve considerably: the COMBINED model achieves 4.4% higher precision and 3.4% higher F-score on **NP–NP**s than the INSERT model. The differences in precision and in F-score for **PRO–NP**s are 9.7% and 8.7%, respectively. In the case of this EE, recall also improves considerably, by 7.3%. These results are, in fact, not surprising, since the trace tagger is already more accurate at detecting these EEs than the INSERT model. A further improvement concerns unbounded dependencies: the F-score for both **WH–NP**s and **WH–ADVP**s rises by 3.1% and 3.6%, respectively.

Interestingly, the new architecture also manages to improve on the results of the tagger when detecting the EEs it is designed to detect (**TOP–S**, **WH–...**, **NP–NP**

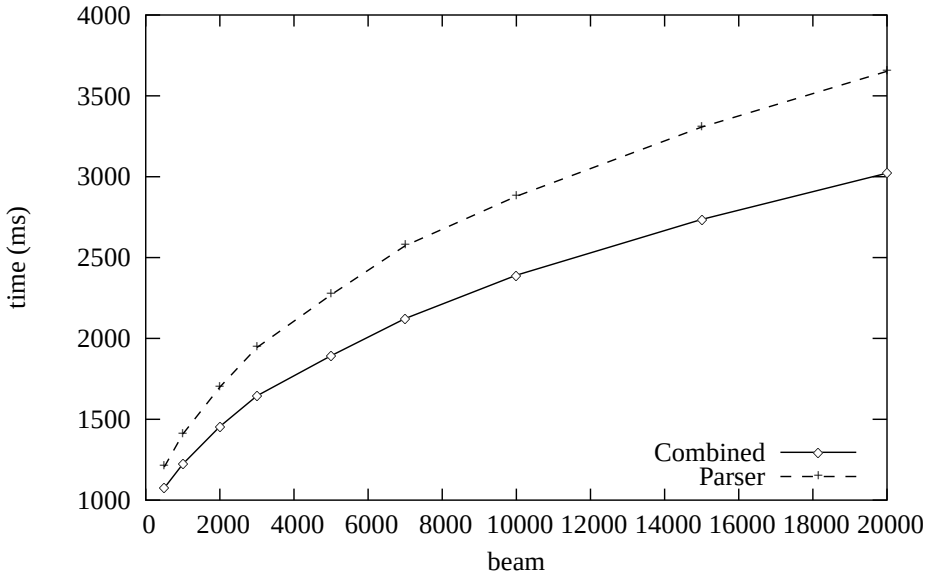


Figure 8.2: Parsing time as a function of the size of the beam.

and **PRO–NP**):[1] the labelled F-score increases by 1.7% on these items. This indicates that, in certain cases, the parser chooses the second or third best hypothesis of the trace tagger. As a consequence, both precision and recall increases (by 2.0% and 1.4%, respectively). Moreover, the COMBINED model no longer suffers from the rigidity of the TAGGER model: the number of failed sentences decreases (from 24 to 5, whereas the parser of the INSERT model cannot parse 11 sentences), and the resulting trees are of better quality (the overall PARSEVAL scores increase by 0.8%).

Another indication of the flexibility of our approach emerges when the parser is integrated with a shallow module of worse quality. Specifically, we used the tagger with only POS-related features switched on (cf. Tables 5.2 and 5.6 in Chapter 5). Recall that this particular tagger detected EEs with 71.3% labelled F-score, which means that the TAGGER approach with the rigid combination scheme could maximally achieve this score. The COMBINED architecture, however, manages to outperform it by 5.3% (!), achieving 76.6% labelled F-score on the EE-detection task. This suggests that further improvement of the tagger could further ameliorate the combined architecture.

An additional benefit of the integrated approach is its speedup over the INSERT model: the search space of the parser is efficiently pruned by the external mod-

[1] Recall that all the other EEs are inserted based on the parse tree, using very simple heuristics (cf. Section 7.2.2); indeed, scores for **COMP–SBAR**s and **UNIT**s are $2-3\%$ worse here than for the tagger.

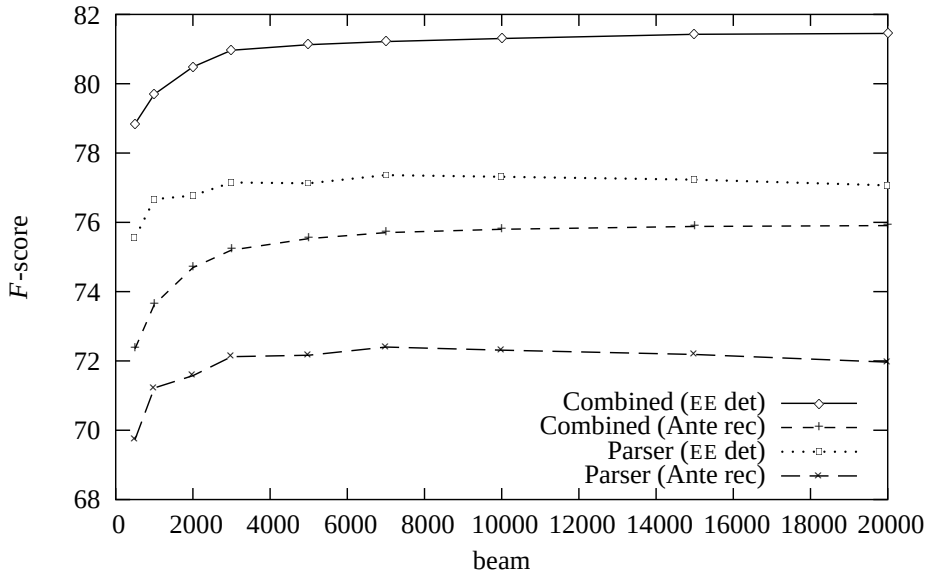


Figure 8.3: NLD-accuracy as a function of the size of the beam.

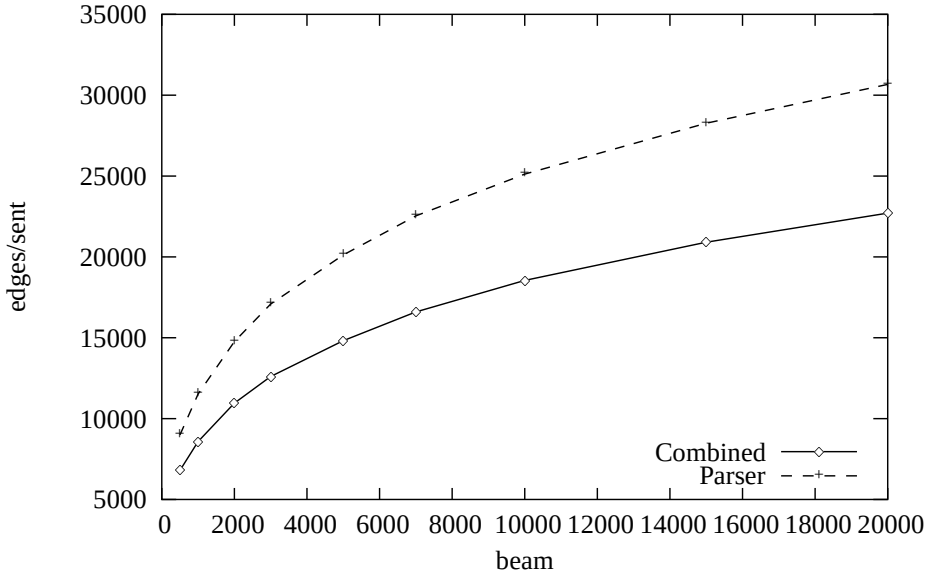


Figure 8.4: Number of edges on the chart as a function of the size of the beam.

ule. Indeed, parsing time decreases by around 17% despite the extra calculations verifying compatibility requires (cf. Figure 8.2). Moreover, the size of the chart (after beam search) is only 76% of the original chart, and the COMBINED model still manages to detect NLDs more accurately. The pruning effect of the shallow module is even more underlined when looking at the number of fully constructed edges: the new architecture reduces their number by one third. On the other hand, the COMBINED model behaves as predicted with respect to the TAGGER model: it is more robust and more accurate, but it explores a larger part of the search space and, thus, it is slower.

A final improvement the integrated architecture achieves concerns the quality of the edges on the chart (cf. Figures 8.2–8.4). Figure 8.3 shows that, for example, the COMBINED model attains the same result on antecedent recovery with the beam of 1000 as the parser-only approach with the widest beam. With the beam set to 1000, the chart contains only 8579 edges (roughly one third of the size of the chart in the case of the INSERT model with the widest beam of 20000) and, on average, requires 1.1 second to parse a sentence (although it fails on around 49 sentences, i.e., roughly on 2% of the sentences). This shows that the probability estimate for the edge probabilities is better: the high probability edges are the ones needed to detect NLDs.

The only metric where the INSERT model outperforms the COMBINED one, albeit only by a small margin, is the PARSEVAL measure. Nevertheless, since the score for surface dependencies is higher for the latter model than for the INSERT model, we do not regard this as a problem. In fact, this discrepancy between the PARSEVAL and dependency scores underline the difference between phrase structure and dependency structure (cf. also Carroll *et al.* 2002); our goal is to recover the latter.

In summary, the experiments presented in this section support our claims: the COMBINED model is both faster and more accurate than the INSERT model, showing that incorporating information from external modules can indeed improve the parser. The new architecture no longer suffers from the rigidity of the combination: the COMBINED model is more robust and more accurate than the TAGGER model. Are these results specific to the task of finding NLDs, or do they translate to other tasks as well? In the next section, we present another experiment with similar results.

8.4 BaseNP chunking

8.4.1 Motivation

Dividing a sentence into non-overlapping phrases is called text chunking:

$[_{VP}$ Dividing $][_{NP}$ a sentence $][_{PP}$ into $][_{NP}$ non-overlapping phrases $][_{VP}$ is called $][_{NP}$ text chunking $]$ (8.15)

The usefulness (and psychological plausibility) of chunking as a preprocessing step prior to parsing was first emphasised by Abney (1991). In his model, a chunker divides the sentence into various types of chunks, and the parser attaches these chunks to each other. The advantage of this approach lies in higher parsing speed: the parser does not need to build the internal structure for the chunks. Moreover, chunking also reduces the ambiguity rate the parser has to deal with, since it excludes edges from the search space crossing chunk boundaries.

Applying a machine learning method to the general chunking task was first proposed by Ramshaw and Marcus (1995), who use a transformation-based learner (Brill 1992, 1995). They also cast the task as a tagging problem: words are labelled with the tags **I**, **O** or **B**, depending on whether they are *inside*, *outside* or at the *beginning* of a phrase, respectively. The sentence above gets the following tag sequence according to this scheme:

Dividing$_{\mathbf{I-VP}}$ a$_{\mathbf{I-NP}}$ sentence$_{\mathbf{I-NP}}$ into$_{\mathbf{I-PP}}$ non-overlapping$_{\mathbf{I-NP}}$ phrases$_{\mathbf{I-NP}}$ is$_{\mathbf{I-VP}}$ called$_{\mathbf{I-VP}}$ text$_{\mathbf{I-NP}}$ chunking$_{\mathbf{I-NP}}$ (8.16)

Note that the **B** tag is only used if a word starts a phrase immediately following a phrase of the same type. Tjong Kim Sang and Veenstra (1999) experiment with other tagging schemes for chunking. Chunking was a shared task at the CoNLL-2000 conference; Tjong Kim Sang and Buchholz (2000) give a good overview of various approaches to chunking. An observation of special importance here is that most of these approaches incorporate *trigram* information into their models.

A special chunking task is baseNP chunking, first introduced by Church (1988). BaseNPs, also called non-recursive NPs, are noun phrases which themselves do not contain a NP, except perhaps in a specifier position (a possessive NP, e.g., *John's book*). As Collins (1999) notes, baseNPs considerably differ from other phrases, at least as far as their representation in the PTB is concerned. First, the head in a baseNP has a different role than in other phrases: in many cases, a non-recursive NP has the structure `NPB->NN NN NN` where the first noun modifies the second instead of the third one, which is the head (e.g. *pet food volume*, *vanilla ice cream*, etc.). The second difference is the strongly marked nature of the left boundary of a baseNP: after having generated a determiner, in general, the parser should stop generating left modifiers. Other phrases do not show a similar behaviour.

Recall that Collins (1997, 1999) defines a unigram Markov model to generate dependents of a given head and parent (cf. Section 7.1.1). However, this model is inadequate when it comes to baseNPs, since it cannot properly incorporate the specialties mentioned above. Therefore, Collins (1999) proposes a different probability model for baseNPs, where the generation of a (left) modifier depends on

not the headword but on the previously generated (left) modifier. Specifically, in the case of baseNPs, Eq. (7.4) is modified as:

$$p(L_n(l_n)\dots L_1(l_1)H(h)R_1(r_1)\dots R_m(r_m)|\text{NPB}(h)) \approx$$
$$\approx p_h(H|\text{NPB},h)\cdot\prod_{i=1}^{n} p_l(L_i(l_i)|\text{NPB},L_{i-1}(l_{i-1}))\cdot$$
$$\prod_{j=1}^{m} p_r(R_j(r_j)|\text{NPB},R_{j-1}(r_{j-1})) \quad (8.17)$$

where $L_0 = R_0 = H$ and $l_0 = r_0 = h$.

This model essentially is a bigram Markov model. However, as noted above, most approaches to chunking use trigrams instead of bigrams (Tjong Kim Sang and Buchholz 2000). Consequently, one could expect higher chunk accuracy when employing a trigram model. In this section, we will explore such a combination: we combine a trigram baseNP chunker and the parser using the general probabilistic architecture. Our baseline is the standalone parser, which finds the correct baseNP chunks with precision 93.8% and recall 94.2% (94.0% F-score).

8.4.2 The chunker

As discussed above, (baseNP) chunking can be viewed as a tagging task. In fact, it is very easy to adapt our maximum entropy tagger developed in Section 5.1 to handle the new tagging (chunking) task; we only have to modify the feature templates: since baseNP chunking does not require long distance features, we only use trigram lexical and POS-related features (see Tables 5.1 and 5.2 on page 97).

In chunking, the previous tags have a larger impact than in the case of detecting EEs. Therefore, we employ features encoding the two previous labels (l_{-1}, l_{-2} and their concatenation, $l_{-2}l_{-1}$). Wider context, however, requires more search. Thus, we widen the beam and do 8-best search (as opposed to the 3-best search we use in the trace tagger). Features occurring less than 5 times are removed from the event space.

When measured on the standard test set of Ramshaw and Marcus (1995), our chunker achieves 92.4% precision, 92.9% recall and 92.7% F-score.[2] In the following experiment, however, we use a much larger training set and make use of the correct POS-tags. Therefore, the performance of the chunker rises considerably: it attains 94.4% precision and 94.7% recall (94.5% F-score) on our test set (WSJ23).

[2]For a comparison with other approaches, visit `http://lcg-www.uia.ac.be/~erikt/research/np-chunking.html`.

8.4.3 The setup

On the baseNP chunking task, our simple chunker outperforms the parser (Model 2) by 0.5% F-score, therefore, we can expect some improvement when combining the external (shallow) module with the parser. In the present setting, the chunker returns its 8-best hypotheses. Both the parser and the chunker are trained independently on the standard training set (WSJ02–21).

As before, the crucial question is how we define compatibility of a partial parse tree with a chunk sequence. This problem, however, turns out to be more complicated this time than it is in the case of finding NLDs. The reason is evident: even though the chunker is implemented as a tagger, in the combination with the parser, we not only have to verify the presence or absence of some elements, but we also have to *compare structures*. As we will see shortly, it is straightforward to implement a function which tests whether the parser's baseNP hypotheses are compatible with the chunker's suggestions. On the other hand, ensuring that the parser hypothesise a baseNP wherever the chunker suggests one is more complicated.

First, let us turn our attention to the former problem and design a compatibility check procedure which is called whenever a parser suggests a baseNP. Clearly, every word dominated by a baseNP nonterminal should be inside a baseNP. This can be easily verified whenever we attach a new word (phrase) to the baseNP. Moreover, in the case of left modifiers, for example, the previous attachment cannot be the beginning of a baseNP:[3] a sequence of `...I B I...` is impossible *within* a non-recursive NP. These operations can be performed whenever two edges are joined (`join_2_edges`), and they ensure that all hypothesised baseNPs are also parts of baseNPs according to the chunker.

It might happen, however, that the parser proposes shorter baseNPs than the chunker. Therefore, whenever the parser generates `STOP` symbols to terminate a baseNP, we have to make sure that the words at the edges according to the parser are also at the edges according to the chunker. This can be done when the action `generate_STOP` is executed. In summary, the parsing algorithm requires the following extensions in order to guarantee that the baseNP hypotheses according to the parser form a subset of the chunker's hypotheses:

`generate_STOP`
: Check whether the left edge of the baseNP is the beginning of a baseNP according to the chunker. Similarly, at the right edge, check for the end of a baseNP.

`join_2_edges`
: If the parent is NPB, then verify that (i) the modifier being at-

[3] BaseNPs containing a possessive NP require a special treatment.

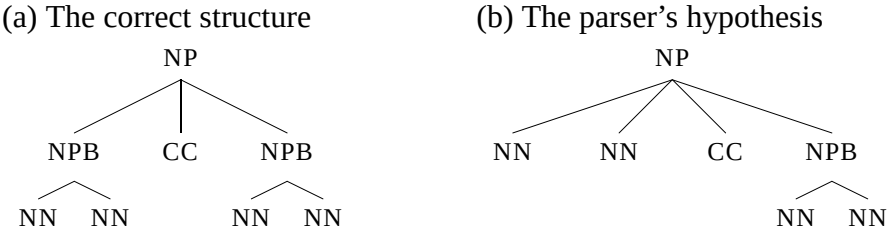


Figure 8.5: A problematic case for checking compatibility.

> tached is inside a baseNP according to the chunker; and (ii) the previously attached modifier is not at the beginning (or end if it is to the right of the head) of a baseNP.

The problem of ensuring that the parser finds all baseNPs is more difficult. Figure 8.5 illustrates the complications: the parser fails to assign the correct substructure to the first two nouns. How to spot such incompatibilities? The safest solution would be to inspect the whole the parse tree whenever the TOP symbol is generated. However, this could be too late to test compatibility: by then, the correct alternative might have been discarded from the search space. Moreover, such a late test would not improve the efficiency of the parser. Another approach could be performing compatibility check when a word is attached to a phrase: the problem is, however, that baseNPs might dominate virtually any phrases, apart from an NP. That is, when we attach a word to, say, an ADJP, we do not know yet whether this ADJP is in a baseNP or not, and, as a consequence, we cannot test whether this word is in a baseNP or not.

However, it turns out that Figure 8.5 shows a typical problem: the parser wrongly attaches a *terminal* to a NP and not to a NPB. It is easy to avoid such a situation: before the parser tries to join a terminal with a NP, it has to check whether the word in question is outside a baseNP according to the chunker. The required modifications are summarised below.

`generate_unary`
`join_2_edges`
> If the parent is a NP and the dependent is a terminal, check whether the latter is outside a baseNP according to the chunker.

Checking compatibility with the baseNP chunker requires no additional data structures and no modifications to the definition of equivalence of edges with respect to the dynamic programming.

		CHUNKER	PARSER	COMBINED
Chunks	precision	94.4	93.8	**94.7**
	recall	94.7	94.2	**94.9**
	F-score	94.5	94.0	**94.8**
Time (s/sent)		0.03	1.8	**1.1**
Chart (edges/sent)		–	14k	**7k**
Fully constructed edges (per sent)		–	434k	**148k**
PARSEVAL	≤ 100	–	88.0	**88.3**
(F-score)	≤ 40	–	88.5	**88.7**
Surface deps.		–	90.7	**91.0**

Table 8.4: Comparision of the parser and the combined architecture on baseNP chunking.

8.4.4 Experiments

In the experiments, the parser (which does *not* insert NLDs this time) and the chunker are trained independently on the standard training set and the combined architecture is tested on WSJ23. We report chunk and parse accuracy, as well as the size of the chart and parsing time. The results are summarised in Table 8.4. In summary, the combination improves both chunk and parse accuracy of the parser, whereas parsing time drops considerably by 40%, due to the smaller number of constructed edges.

8.4.5 Discussion

The combination of the chunker and the parser has the most dramatic effect on the number of constructed edges: even though both chunk and parse accuracy improve, the number of fully constructed edges in the combined model is around one third of their number in the original parser. Accordingly, the chart is much smaller: around half the size as for the original model. This shows that the quality of the constructed edges is much better: with fewer hypotheses, the combined architecture attains better performance. Clearly, the most important benefit of the combination is early filtering of the edges: the original parser hypothesises many phrases that cut across baseNP boundaries, which results in constructing a large number of unnecessary edges.

It is interesting to note that the combined architecture also manages to outperform the chunker on the chunking task. As in the case of recovering NLDs, this result shows that the parser indeed selects second/third/...-best hypotheses from the chunker's suggestions.

8.5 Discussion

In the previous sections, we have shown that the COMBINED models work in practical situations. However, there is a step which can be criticised from a theoretical point of view: the independence assumption of Eq. (8.6). Indeed, this assumption could have led to the failure of the approach. On the practical side, however, there are several reasons why it is useful and even necessary. First, without the independence assumption, the parser and the external module would no longer be independent: changing the external module would entail retraining the parser. Moreover, it is not clear how the probability $p(T|x,S)$ could be reasonably estimated: in the training data, we always have the perfect hypotheses, thus we cannot learn the mistakes of the external module. Furthermore, an approach solving the above problem would need much more training data, a requirement that is hard to achieve. Therefore, problematic as is from theoretical point of view, the independence assumption is necessary to combine the models in a feasible way.

Another question is whether the probabilistic integration is important at all: do the prior probabilities $p_e(x_i|S)$ actually contribute to the accuracy of the system? Otherwise, we could have an even simpler combination, assuming a flat prior or even no prior at all. We reran both experiments using both a flat prior and no prior: the resulting models proved to be marginally worse and slower than the probabilistic combination. This observation shows that knowing the prior distribution does improve the system, but the main effect of the combined approach is filtering. As a consequence, the new architecture allows the combination of non-probabilistic external modules with the parser, by simply assuming a uniform probability distribution on the output of the external modules.

Another possible problem is the definition of compatibility. In the examples above, the notion of compatibility is straightforward. There might occur cases, however, when it turns out to be more complex. For example, combining a top-down approach with the bottom-up parser might be tricky: the compatibility check should be postponed until a fairly late stage in the parsing process. In fact, even in the combination of the trace tagger and the parser, we face the same problem: we cannot decide whether the hypotheses of the parser and the tagger are compatible until we try to combine the edges neighbouring the EE-site (`join_2_edges`). Nevertheless, the architecture only hinges on the *monotonicity* of the incompatibility relation: if a sub-structure at a point is incompatible with an external hypothe-

sis, then all structures containing this sub-structure are incompatible. In the worst case, incompatibility can be defined on full parses only: as a consequence, the architecture would not do any filtering while parsing, but it would remain consistent.

Although we presented the architecture as an external module helping a bottom-up parser, it can be stated more generally. First, it is not tied to any parsing algorithm or even parsing formalism. Actually, it is not tied to parsing, either. The formulation in Eq. (8.7) can be applied in a wide variety of different settings. The only important requirement is that the compatibility set $\mathcal{C}(T')$ for each structure T' should be a subset of $\mathcal{C}(T)$, where T is a substructure of T'. Therefore the most general formulation would be as follows:

- Denote $\mathcal{X}_k = \{x_1, \ldots, x_k\}$ a set of mutually exclusive events with a probability distribution p_e, so that $\sum_{i=1}^{k} p_e(x_i) \approx 1$ (the external module; the events stand for external hypotheses).

- Denote $\mathcal{T} = \{t_1, \ldots\}$ a set with a partial ordering relation $<$ and a probability distribution p_p (the parser; t_is are substructures, $t_i < t_j$ iff t_j can be reached from t_i with a finite number of actions by the parser).

- Let $c : \mathcal{T} \to \mathcal{P}(\mathcal{X}_k)$ be a function, so that $c(t_j) \subseteq c(t_i) \subseteq \mathcal{X}_k$ whenever $t_i < t_j$ (the monotonic compatibility function).

- Then we estimate the probability of a structure t_i as

$$p_c(t_i) = \frac{1}{Z} \cdot p_p(t_i) \cdot \sum_{x \in c(t_i)} p_e(x) \tag{8.18}$$

This formulation makes the approach very general and applicable in several stochastic NLP systems aiming at combining different sources of information. Note, however, that the roles of the parser and of the external module are not symmetric: the external module is regarded as a preprocessor, guiding the parser. This design choice hinges on the assumption that parsing is more time-consuming than getting the external hypotheses. As a consequence, information mainly flows from the preprocessor towards the parser. Nevertheless, since the parser sometimes selects solutions suboptimal according to the external module's probability model, there is a limited flow of information from the parser towards the external module. The development of an architecture where the two modules have equal roles is deferred for future work.

8.6 Related work

The underlying idea of employing shallow modules as a preprocessing step before parsing to improve accuracy, coverage and/or parsing speed is very common: most statistical parsers use POS-tagged input so that they can fall back to the POS-tags supplied whenever they do not have a reasonable estimate for it. Ratnaparkhi (1997, 1998) integrates his tagger (Ratnaparkhi 1996) into a parsing model, in order to avoid errors due to the parser's commitment to one POS-sequence. Supertagging generalises the idea: the search space of the parser is reduced by initially assigning a small number of elementary syntactic structures to each word (Srinivas and Joshi 1999, Clark 2002, Clark *et al.* 2002). Note, however, that these approaches all assume that the preprocessor is basically a tagger: there is one label associated with each word.

Work generalising to one-to-many correspondence between input structures and words is relatively scarce in the field of statistical parsing. Collins (1996, 1997) uses a different probability model for baseNPs, which basically amounts to incorporating a baseNP chunker into the parsing model. Ratnaparkhi (1997, 1998) also integrates a general chunker into his parsing model. Brants (1999a,b) generalises the idea by using not only base chunks but basically chunks of arbitrary depth. These chunks act as filters on partial context-free trees: they prune unlikely sequences of partial structures. These approaches, however, fail to generalise beyond chunks.

In order to improve robustness and decrease parsing speed, integrating external (shallow) modules with a symbolic (deep) parser has been attracting increasing interest as well. Prins and van Noord (2001) combine *n*-best hypotheses of part-of-speech tagger with Alpino, a wide-coverage HPSG parser. The tagger acts as a filter on the parser. Daum *et al.* (2003) employ both a tagger and a chunker as sources of filter constraints in a constraint based framework. Crysmann *et al.* (2002) propose a general, non-probabilistic architecture to integrate shallow and deep NLP modules using a multilevel XML annotation scheme. In this framework, Frank *et al.* (2003) combine a probabilistic topological parser (Becker and Frank 2002) with a symbolic HPSG parser (Callmeier 2000). They use various confidence scores derived from the stochastic parser to facilitate the search of the deep parser. However, they do not design a general framework to use such scores.

Summary

This chapter has presented a new general probabilistic architecture to combine different external modules with a parser. We have shown through two experiments that the new architecture both improves the accuracy of the model and speeds up

parsing by reducing the search space of the parser. Another important characteristics of this architecture is that the parser does not require re-training when a new module is added. Finally, in certain cases, the combined architecture improves on the accuracy of the external module as well, showing mutual interaction between the parser and the external module.

Chapter 9

Conclusions

The main thesis presented in this dissertation is that non-local dependencies in English can be efficiently and accurately recovered by an appropriate combination of shallow approaches. In particular, we have shown that (i) it is possible to detect the majority of NLD-sites with state-of-the-art accuracy without explicit knowledge of phrase structure; and (ii) even an unlexicalised PCFG parser can find the non-local dependents once it is informed whether a head participates in such a construction. These insights have led us to propose a two-level architecture where a finite-state machine provides a parser with the information about NLD-sites. In order to make the parser more robust with respect to the errors the finite-state machine commits, we have developed a probabilistic framework in which the preprocessor imposes soft constraints on the parser and thus efficiently guides it towards an optimal solution without forcing it to stick to erroneous decisions of the preprocessor.

Although our last system achieves the best reported results on identifying non-local dependencies in English (see Table 9.1), the problem is far from being solved. Indeed, accuracy, especially recall, should be improved. There are several avenues for future research that might prove useful for improving the scores. We plan to incorporate more, especially non-local and external, information. It is also essential to fight data sparseness. One line of attack might be reducing the number of parameters the parser has, for instance, by using a lightly lexicalised parser or even a finite state machine. Another possible way to go about the problem is making use of unlabelled data, by either applying co-training or unsupervised learning techniques.

The themes underlying the present research point further than the actual task of recovering NLDs. One of the most salient issues has been the reduction of the search space the parser has to explore. We have argued throughout the dissertation that reliable reduction of the search space improves both the accuracy and the efficiency of the parser. No doubt, preprocessing has always been regarded as a means of search space reduction. The present work, however, extends the standard

		UNLEX			LEX				
		NOTRACE	TAGGER	PERFECT	NOTRACE	TAGGER	INSERT	COMBINED	PERFECT
EE det.	all	–	79.6	*100*	–	79.6	76.7	79.6	*100*
(F-score)	**WH/PRO/TOP**	–	79.6	*100*	–	79.6	77.3	81.3	*100*
ANTE rec.	all	–	72.9	*89.7*	–	74.3	73.3	75.8	*90.5*
(F-score)	**WH/PRO/TOP**	–	70.4	*89.5*	–	72.3	72.3	75.8	*90.4*
HD accuracy	**WH/PRO/TOP**	–	67.8	*86.3*	–	71.3	72.9	75.5	*89.0*
Time (s/sent)		*9.3*	8.9	*8.5*	*1.8*	1.6	2.9	2.4	*1.6*
Chart (edges/sent)		*368k*	339k	*327k*	*14k*	13k	25k	19k	*12k*
PARSEVAL	≤ 100	*72.2*	75.7	*78.0*	*88.0*	86.3	87.3	87.1	*88.2*
(F-score)	≤ 40	*73.7*	77.0	*79.3*	*88.5*	87.1	87.9	87.7	*88.8*
Surface deps.		*79.3*	81.0	*82.5*	*90.7*	88.6	89.2	89.6	*90.0*
All deps.		*77.1*	80.9	*82.7*	*88.2*	88.2	88.6	89.2	*90.5*

Table 9.1: Summary of the models.

approaches in several respects. First, our architecture defines a more flexible interface between the preprocessor and the parser, where the parser can select from multiple hypotheses of the preprocessor and take the preprocessor's probability estimation into account. Second, the architecture allows incorporating not only shallow but any type of information that might help the parser. Finally, it opens up the possibility of combining probabilistic and symbolic approaches.

We have claimed that this architecture is general enough to accommodate many tasks and the combination of several modules. Although we have successfully applied it in two experiments, exploring its usefulness and possible limitations requires further research. Even in parsing, there are many possible modules that can eventually help the parser, from a POS-tagger or a general chunker to systems specialised in named-entity recognition, word-sense disambiguation, resolving PP-attachment ambiguity, etc. Another interesting line of future research is providing the parser with higher-level information about the dialogue structure or world knowledge in order to prevent it from proposing readings that the dialogue system cannot handle or that are unlikely according to the state of the dialogue/world (cf. Gabsdil 2004).

Our second underlying theme concerns system design. One approach to attack a difficult problem is to employ a powerful apparatus to solve it; let us call this design top-down design. The other approach, the one we advocate here, is the opposite: start with a simple framework and after understanding why it fails, extend it to cover the problem at hand. We call this design technique bottom-up design. Although we might eventually end up with the same powerful system as in the case of the top-down approach, there are three main reasons why we believe the bottom-up design is superior.

The first reason concerns parsing technology and efficiency. In the search for an adequate system, we might stop our hill-climb earlier and find a simpler and often more efficient architecture with the same (or even better) accuracy as the more complicated one. In a statistical NLP setting, simplicity frequently entails fewer parameters, which can be estimated more reliably than in the case of a complicated model with more parameters. That is, by adopting the bottom-up design technique, we make better use of the existing labelled data. Moreover, simpler models are usually easier to estimate from unlabelled data.

Second, employing a powerful framework prevents us from gaining important insights into the nature of the task at hand, since a more involved system offers many possible solutions to a problem. Simpler approaches, on the other hand, allow for a more restricted set of solutions, and thus force us to thoroughly investigate the causes of the difficulties. This way we can acquire a deeper understanding of the task, and of language in general.

Third, the bottom-up approach naturally supports the combination of various simple modules. Modularity is useful in many respects. Modular systems are

easier to design, implement, debug and maintain. They might also allow parallel computing and thus speed up the system. A further benefit of modularity is the independence of the modules: they can be separately updated if a more accurate version is available. Finally, a modular design permits the combination of a wide variety of information sources, and it might be an adequate model of human sentence processing.

The success of many relatively simple approaches to difficult tasks such as word sense disambiguation, statistical machine translation, or even parsing, supports the bottom-up design. Ultimately, we believe that many "hard" tasks can be successfully tackled with relatively simple systems, especially when combined with statistical approaches. This dissertation has been an illustration of this claim.

Appendix A

A detailed inventory of empty elements

This appendix gives a detailed inventory of EEs as they are used in the Penn Treebank. We follow the annotation guidelines (Bies *et al.* 1995) in the present discussion and the examples are from the same source unless otherwise indicated. In the presentation below, however, we use the notation of Chapter 2.

Recall that the PTB annotation scheme distinguishes the following types of empty nodes:

T	trace of A′-movement, including parasitic gaps
(NP *)	arbitrary **PRO**, controlled **PRO** and trace of A-movement
PPA	pseudo-attach: permanent predictable ambiguity
RNR	pseudo-attach: right node raising
ICH	pseudo-attach: interpret constituent here
EXP	pseudo-attach: expletives
?	placeholder for ellipsed material
0	null complementiser, including null WH-operator
U	empty unit

Table 2.2 on page 31 shows how these labels map to our labels for empty elements.

A.1 WH–... and TOP-S: traces of A′-movement

The use of this EE-marker loosely corresponds to A′-movement (cf. Haegeman 1991), although it also covers parasitic gaps. There are two main types of such EEs: (i) traces of topicalisation and (ii) traces of WH-movement. S-level constituents in these constructions are always topicalised, whereas lower level constituents (NPs, ADVPs, PPs, etc.) tend to occur in WH-extraction. The PTB annotation scheme uses *T*-n to represent these kinds of EEs, regardless of their type. In order to have more mnemonic names, we use a different label (**TOP–S**) for topicalised sentences. Note further that some non-sentential constituents do occur in topicalised positions; they are nevertheless represented with **WH–...** EEs here. Finally, it is important to emphasise that these EEs always have an antecedent. Table A.1 shows the distribution of the various EEs in this group as observed in the development set (WSJ00): **WH–NP**, **TOP–S**, **WH–ADVP**, and **WH–PP** are far the most common EEs of this type. There are a handful other **WH–...** traces in the training set, but they do not occur in the development section. Below, we give an exhaustive description where **WH–...** and **TOP–S** gaps occur in the Treebank.

Type	Number	
	WSJ00	train
WH–NP	438	8661
TOP–S	233	4141
WH–ADVP	120	2560
WH–PP	24	447
WH–ADJP	0	72
WH–VP	0	58
All	815	15939

Table A.1: The distribution of various **WH–...** and **TOP–S** traces in WSJ00 and the training set.

A.1.1 WH–NP

This EE-type represents NP-traces either undergoing A′ movement (i.e., when the NP ends up in a non-argument position, cf. Haegeman 1991, Chapter 7) or being a parasitic gap (cf. e.g. Pollard and Sag 1994, Section 4.5). Table A.2 at the end of this section gives detailed statistics on the distribution of **WH–NP** traces in WSJ00.

WH-questions

The NP is extracted either from an argument (A.1a) or a non-argument (A.1b) position. It also includes vacuous subject movement, such as in (A.1c) (example taken from `wsj_0041.mrg`).

(a)
```
(SBARQ (WHNP-1 what
       (SQ are
           (NP-SBJ you )
           (VP thinking
               (PP-CLR about
               (NP WH−NP-1)))))
       ? )
```
(A.1)

(b)
```
(SBARQ (WHNP-1 Which day)
       (SQ did
           (NP-SBJ you)
           (VP get
               (ADVP-DIR there)
               (NP-TMP WH−NP-1)))
        ? )
```

(c)
```
(SBARQ (WHNP-1  Who)
       (S (NP-SBJ WH−NP-1)
          (VP 's
              (VP telling
                  (NP the truth))))
       ? )
```

Relative clauses

Relative clauses are adjoined to a NP and get an SBAR label (as opposed to the SBARQ label of the WH-questions). There are three main types of such constructions: with an overt *that* or a WH-complementiser (A.2a), with a null complementiser (in this case the null complementiser represented by the empty element **COMP–WHNP** is taken to be the antecedent; cf. A.2b), and infinitival relatives (which are, in fact, subcases of relatives without overt complementiser; A.2c and d). In the cases of overt complementisers and infinitival relatives, the extracted NP is either a subject or an object in the embedded clause; null WH-complementisers in finite subclauses are allowed only in object extraction.

(a)
```
(NP (NP answers)
    (SBAR (WHNP-1 that/which)
          (S (NP-SBJ we)
```
(A.2)

```
                (VP encountered
                    (NP WH−NP1)))))
```

(b)
```
(NP (NP answers)
       (SBAR (WHNP1 COMP−WHNP)
             (S (NP-SBJ we)
                (VP encountered
                    (NP WH−NP1)))))
```

(c)
```
(NP (NP a movie)
       (SBAR (WHNP1 COMP−WHNP)
             (S (NP-SBJ PRO−NP)
                (VP to
                    (VP see
                        (NP WH−NP1)))))))
```

(d)
```
(NP (NP bloodhounds)
       (SBAR (WHNP1 COMP−WHNP)
             (S (NP-SBJ WH−NP1)
                (VP to
                    (VP trail
                        (NP the assassins))))))
```

Fronting

Generally, fronted arguments are represented in the PTB as in (A.3a), with the corresponding EE co-indexed with the topicalised constituent. There are, however, some additional complications with fronting. First, as illustrated in (A.3b), if the fronted argument is left-dislocated, i.e., it is associated with a resumptive pronoun, no EE is involved. Furthermore, if the fronted NP is an adjunct (A.3c), no trace is involved as long as the adjunct originates in the same clause. On the other hand, adjunct NPs originating in a lower clause leave a trace (A.3d).

(a) (A.3)
```
(S (NP-TPC1 This)
   (NP-SBJ every man)
   (VP contains
       (NP WH−NP1)
       (PP-LOC-CLR within
                  (NP him))))
```

(b)
```
(S (NP-TPC John)
   ,
   (NP-SBJ I)
   (VP like
```

```
          (NP him)
          (NP-ADVP a lot)))
```

(c)
```
(S (NP-TMP Yesterday)
   (NP-SBJ John)
   (VP called
       (NP us)))
```

(d)
```
(S (NP-TMP-TPC1 Yesterday)
   (NP-SBJ I)
   (VP think
      (S (NP-SBJ John)
         (VP called
             (NP us)
             (NP-TMP WH−NP1)))))
```

Tough-constructions

In the case of *tough*-constructions (cf. e.g. Pollard and Sag 1994, Section 4.4), the empty element is co-indexed with a null WH-operator (A.4).

```
(S (NP-SBJ Cars)                                                    (A.4)
   (VP are
       (ADJP-PRD tough
          (SBAR (WHNP1 COMP−WHNP)
                (S (NP-SBJ PRO−NP)
                   (VP to
                       (VP pay
                           (PP-CLR for
                               (NP WH−NP1)))))))))
```

Parasitic gaps

A parasitic gap occurs whenever an extracted NP is an argument of two or more verbs (cf. e.g. Pollard and Sag 1994, Section 4.5). The PTB annotation scheme represents parasitic gaps by introducing two **WH–NP** gaps with the same index (A.5a). Coordination is handled in a similar fashion (A.5b).

(a)
```
(SBARQ (WHNP1 Which papers)                                         (A.5)
       (SQ did
           (NP-SBJ2 you)
           (VP file
               (NP WH−NP1)
               (PP without
```

```
                    (S-NOM (NP-SBJ NP−NP2)
                           (VP reading
                               (NP WH−NP1)))))))
```

(b)
```
(VP tells
    (NP you)
    (SBAR (WHNP1  what)
          (S (NP-SBJ  the characters)
             (VP are
                 (VP (VP thinking
                         (NP WH−NP1)
                     and
                     (VP feeling
                         (NP WH−NP1)))))))))
```

Type		all	subject	object
WH-questions		9	9	0
Relatives		427	332	95
	that	331	293	38
	empty	63	–	63
	infinitival	33	29	4
Fronting		0	0	0
Tough		2	–	2
Parasitic		0	0	0
All		438	341	97

Table A.2: The distribution of the **WH–NP** traces in WSJ00.

A.1.2 TOP–S

These EEs represent topicalised subclauses if they precede their matrix verb. This constellation mostly occurs with quotations in the PTB. The tree in (A.6a) shows the basic case. If the quotation is discontinuous, the interrupting material is annotated as a parenthetical (PRN) and the EE is co-indexed with the whole sentence (A.6b). Finally, note that the annotation scheme requires an EE only if (part of) the quotation appears before the matrix verb (cf. A.6c).

(A.6)

(a)
```
(S ‘‘
   (S-TPC1 (NP-SBJ Willie)
           (VP caught
               (NP the ball)))
   ’’
   (NP-SBJ Casey)
   (VP said
       (S TOP−S1))
   .)
```

(b)
```
(S1 ‘‘
    (NP-SBJ Willie)
    ’’
    (PRN ,
         (S (NP-SBJ Casey)
            (VP said
                (S TOP−S1)))
         ,)
    ’’
    (VP caught
        (NP the ball))
    .
    ’’)
```

(c)
```
(S (NP-SBJ Casey)
   (VP said
       ’’
       (S (NP-SBJ Willie)
          (VP caught
              (NP the ball))))
   .
   ’’)
```

A.1.3 WH−ADVP, WH−PP, etc.

These EEs occur in very similar syntactic constructions as **WH−NP**s. There is a further construction specific to **WH−ADVP**s which we refer to as *so*-constructions (A.7a and b). Table A.3 shows the distribution of these EEs in WSJ00.

(A.7)

(a)
```
(SINV (ADVP-PRD-TPC1 So)
      (VP is
          (ADVP-PRD WH−ADVP1)
      (NP its balance sheet)))
```

(b) (SINV (ADVP-LOC-PRD-TPC$_1$ Here)
(VP are
(ADVP-LOC-PRD **WH−ADVP**$_1$)
(NP the figures)))

Type		**WH−ADVP**	**WH−PP**
WH-questions		2	0
Relatives		112	18
	overt comp.	82	17
	infinitival	30	1
Fronting		0	6
So-constructions		6	–
All		120	24

Table A.3: The distribution of the **WH−ADVP** and **WH−PP** traces in WSJ00.

A.2 NP−NP and PRO−NP: traces of NP-movement, controlled and arbitrary PRO

These types of EEs are the most frequent ones in the PTB corpus; many phenomena are indeed analysed in terms of NP-movement and **PRO**s. The most important difference between the two types of EEs (**NP−NP** and **PRO−NP**) is whether they are co-referential with a NP within the sentence or not. The original annotation scheme represents both with a *, the only difference being the presence or absence of the index indicating co-reference. In order to avoid confusion, we use two different labels: **NP−NP** stands for the co-indexed version (*-n), whereas **PRO−NP** represents the type without co-indexation (*). Generally, both of them participate in similar syntactic constructions, therefore we present them together below, indicating restrictions on their occurrence in the given environments. Table A.4 at the end of this section summarises the distribution of the **NP−NP** and **PRO−NP** traces.

A.2.1 Passives

Although the treatment of passivisation in terms of movement, or even accounting for it as a syntactic process, is disputed (e.g. Sag 1982, Pollard and Sag 1994),

the PTB annotation scheme follows the transformational grammarian approach to passivisation. Under this view, passivisation is a syntactic process which involves the movement of the (direct) object of the verb from object to subject position. The (surface) subject is co-indexed with its trace. The NP undergoing movement is the object of either a verb (A.8a) or a preposition (A.8b). These constructions also include "small clauses" (Haegeman 1991, p. 50f) with passive verbs (A.8c, a simplified example from `wsj_0021.mrg`).

(a) (A.8)
```
(S (NP-SBJ1 John)
    (VP was
        (VP hit
            (NP NP−NP1)
            (PP by
                (NP-LGS a ball)))))
```

(b)
```
(S (NP-SBJ1 (NP kid 's) cars)
    (VP are
        (ADVP-TMP often)
        (VP paid
            (PP-CLR for
                    (NP NP−NP1))
            (PP by
                (NP-LGS their parents)))))
```

(c)
```
(S (NP-SBJ the country)
    (VP wants
        (S (NP-SBJ1 half the debt)
           (VP forgiven
               (NP NP−NP1)))))
```

A.2.2 Reduced relatives

These cases might as well be viewed as passive constructions. However, the EE is never co-indexed with the corresponding NP, which "reflects an understanding of the relationship between the NP and the reduced relative as post-modification rather than predication" (Bies *et al.* 1995, p. 70).

(a) (A.9)
```
(NP (NP an agreement)
    (VP signed
        (NP PRO−NP)
        (PP by
            (NP-LGS everyone))))
```

A.2.3 Subjects of infinitival clauses

There are several syntactic constructions where the subject of an infinitival clause is a **PRO**, which is either co-indexed with a NP in the sentence or is not controlled. These EEs always appear under an NP–SBJ node.

VP and ADJP complement clauses

These instances include raising and control constructions, affecting both verbal (A.10a–c) and adjectival complements (A.10e). Also, infinitival complements of semi-auxiliaries (*supposed to, ought to, have to, about to*, etc.) are represented as having an empty subject (A.10d). The subject of the infinitival clause is co-indexed with the controlling NP.

(a) Raising (A.10)

```
(S (NP-SBJ1 Everyone)
   (VP seems
       (S (NP-SBJ NP−NP1)
          (VP to
              (VP dislike
                  (NP Drew Barrymore))))))
```

(b) Object control

```
(S (NP-SBJ Ford)
   (VP persuaded
       (NP1 Zaphod)
       (S (NP-SBJ NP−NP1)
          (VP to
              (VP run
                  (PP-CLR for
                          (NP president)))))))
```

(c) Subject control

```
(S (NP-SBJ1 Zaphod)
   (VP promised
       (NP Ford)
       (S (NP-SBJ NP−NP1)
          (VP to
              (VP run
                  (PP-CLR for
                          (NP president)))))))
```

(d) Semi-auxiliaries

```
(S (PP Of (NP course))
   ,
   (NP-SBJ1 regulators)
   (VP would
       (VP have
           (S (NP-SBJ NP−NP1)
              (VP to
                  (VP approve
                      (NP (NP Columbia 's)
                          reorganization)))))))
```

(e) ADJP complement clauses

```
  (S (NP-SBJ1 This climb)
     (VP is
         (ADJP-PRD likely
            (S (NP-SBJ NP−NP1)
               (VP to
                   (VP be
                       (ADJP-PRD difficult)))))))
```

Adverbials

Examples include, for instance, purpose clauses (A.11a), semi-complement clauses (A.11b), resultative clauses (A.11c), etc. Again, the subjects are co-indexed with the appropriate NP.

(a) purpose clause (A.11)

```
(S (NP-SBJ1 The public)
   (VP did n't
       (VP come
           (PP-DIR to
                   (NP the market))
           (S-PRP (NP-SBJ NP−NP1)
                  (VP to
                      (VP play
                          (NP a game)))))))
```

(b) semi-complement clause

```
(S (NP-SBJ_1 Skilled ringers)
   (VP use
       (NP their wrists)
       (S-CLR (NP-SBJ NP−NP_1)
              (VP to
                  (VP advance or retard
                      (NP the next swing))))))
```

(c) resultative clause

```
(S (NP-SBJ_1 (NP London 's)
             Financial Times 100-share index)
   (VP shed
       (NP 40.4 points)
       (S-ADV (NP-SBJ NP−NP_1)
              (VP to
                  (VP finish
                      (PP-CLR at
                              (NP 2149.3)))))))
```

Infinitives inside NPs

These include complement clauses of nouns (A.12a) and subjects of infinitival relative clauses (A.12b). These empty subjects are not co-indexed with a NP, therefore they are represented with **PRO–NP** gaps. If it is the subject of the infinitival relative clause that undergoes WH-extraction, the subject **WH–NP** is co-indexed with the (empty) WH-complementiser (A.2d).

(a)
```
(NP (NP John 's)
    decision
    (S (NP-SBJ PRO−NP)
       (VP to
           (VP leave))))
```
(A.12)

(b)
```
(NP (NP a manual)
    (SBAR (WHNP_1 0)
          (S (NP-SBJ PRO−NP)
             (VP to
                 (VP write
                     (NP WH−NP_1))))))
```

Further examples with **PRO**s as subjects of infinitival clauses involve imperative subjects (A.13a) and subjects of *tough*-constructions (A.13b, cf. also A.4). Here, the subject is an arbitrary **PRO** (**PRO–NP**).

(A.13)

(a)
```
(S (NP-VOC Kris)
   ,
   (NP-SBJ PRO−NP)
   (VP go
       (ADVP-DIR home))
   !)
```

(b)
```
(S (PP for (NP Zaphod))
   ,
   (NP-SBJ that steak)
   (VP is
       (ADJP-PRD ready
          (SBAR (WHNP1 0)
                (S (NP-SBJ PRO−NP)
                   (VP to
                       (VP eat
                           (NP WH−NP1)))))))))
```

A.2.4 Subjects of participial clauses and gerunds

The annotation scheme does not attempt to make a (theory-based) distinction between gerunds and other present participles. *-ing* clauses are labelled as a VP when a complement of (various forms of) *be*, as an S–NOM in subject and prepositional object positions, as an S if a verbal complement, and as an S–ADV/TMP/LOC/etc. if the clause is a verbal or sentential modifier. The *-ing* clauses labelled as S (i.e., S, S–NOM and S–ADV/etc.) require either an overt or a null subject. In the case of null subjects, the co-referent is co-indexed with the EE, "if it is clear to the annotator that the two are coreferent" (Bies *et al.* 1995, p. 70). Examples include:

VP complements

They may occur both with and without co-indexed empty subjects (A.14a and b, respectively).

(A.14)

(a)
```
(S (NP-SBJ1 I)
   (VP stopped
       (S (NP-SBJ NP−NP1)
          (VP eating
              (NP chocolate)))
```

```
            (PP-PRP for
                    (NP Lent))))
```

(b)
```
(S (NP-SBJ A Texas legislator)
   (VP proposes
       (S (NP-SBJ PRO−NP)
          (VP color-coding
              (NP (NP (NP drivers ')
                      licenses)
                  (PP of
                      (NP drug offenders)))))))
```

PP complements

"Null subjects of gerund complements of PP modifiers of NPs are coindexed only if there is a particularly strong coindexed interpretation or the PP appears to be part of some «fixed phrase»" (Bies *et al.* 1995, p. 98).

(a)
```
(S (NP-SBJ the company)                                          (A.15)
   (VP has
       (NP (NP no intention)
           (PP of
               (S-NOM (NP-SBJ PRO−NP)
                      (VP tapping
                          (NP its short-term
                                 bank lines)))))
       (PP-TMP for
               (NP (NP a good part)
                   (PP of
                       (NP 1990))))))
```

(b)
```
(S (PP In
       (NP (NP addition)
           (PP to
               (S-NOM (NP-SBJ NP−NP1)
                      (VP having
                          (NP high price-earnings
                                 ratios))))))
   ,
   (NP-SBJ1 most)
   (VP pay
       (NP puny dividends)))
```

Adverbials

They may occur as VP-level and sentence level adjuncts (A.16a and b).

(A.16)

(a)
```
(S (NP-SBJ_1 She)
   (VP left
       ,
       (S-ADV
          (NP-SBJ_2 NP−NP_1)
          (VP offended
              (NP NP−NP_2)
              (PP by (NP-LGS their remarks))))))
```

(b)
```
(S (NP-SBJ_1 Borough Presidents)
   ,
   (SBAR-ADV while
      (S (NP-SBJ NP−NP_1)
         (VP retaining
             (NP (NP membership)
                 (PP-LOC in
                    (NP (NP the Board)
                        (PP of
                            (NP Estimate))))))))
   ,
   (VP lose
       (NP their housekeeping functions)))
```

Subjects

The gerundial clause gets an S–NOM–SBJ label and its **PRO** subject is mostly not co-indexed (A.17a) even if its referent is in the sentence (*you* in A.17b). Some examples involve controlled **PRO** subjects, though (A.17c, simplified example from `wsj_0766.mrg`).

(A.17)

(a)
```
(S (S-NOM-SBJ (NP-SBJ PRO−NP)
              (VP Taking
                  (NP Iwo Jima)))
   (VP was
       (NP-PRD no easy feat)))
```

(b)
```
(S (S-NOM-SBJ (NP-SBJ PRO−NP)
              (VP Eating
                  (NP chocolate)))
   (VP is
```

```
            (ADJP-PRD good
                      (PP for
                          (NP you)))))
```

(c)
```
(S (NP-SBJ_1 she)
   (VP admits
       (SBAR that
             (S (S-NOM-SBJ
                    (NP NP−NP_1)
                    (VP venturing
                        (PP inside the house)
                (VP was
                    n't
                    (NP such a great idea))))))))
```

A.2.5 Subjects of *as*- and *than*-clauses

A **PRO** is used as a "placeholder" subject in clauses introduced by *than* or *as* if they lack an overt subject. The **PRO** subject may be arbitrary (A.18a–c) as well as controlled (A.18d). Note the wide variety of the constructions: comparatives (A.18a and b), sentential adjuncts (A.18c), verbal complements (A.18d).

(a) (A.18)
```
(S But
   (NP-SBJ there)
   (VP may
       (VP be
           (NP-PRD (NP less)
                   (ADVP-LOC there)
                   (SBAR than
                         (S (NP-SBJ PRO−NP)
                            (VP meets
                                (NP the eye))))))))
```

(b)
```
(NP (NP as little)
    (SBAR as
          (S (NP-SBJ PRO−NP)
             (VP is
                 (ADJP-PRD consistent ...)))))
```

(c)
```
(S (NP-SBJ Primerica)
   ,
   (SBAR-ADV as
             (S (NP-SBJ PRO−NP)
                (VP expected)))
```

```
        ,
        (ADVP also)
        (VP acquired ...))

(d) (NP (NP Items)
        (VP listed
            (NP1 PRO−NP)
            (PP-CLR as
                (S-NOM (NP-SBJ NP−NP1)
                       (VP being
                           (PP-PRD in
                               (NP short supply)))))))
```

Type		NP−NP	PRO−NP
Passive		380	–
	VP	377	–
	PP	3	–
Reduced relative		–	152
	VP	–	150
	PP	–	2
Subjects		607	274
	infinitival	383	146
	participial	220	120
	than/as	4	8
All		987	426

Table A.4: The distribution of **NP−NP** and **PRO−NP** in WSJ00.

A.3 PSEUDO−. . . : pseudo-attach

The PTB annotation scheme uses the pseudo-attach function to represent various syntactic constructions: (i) predictable structural ambiguity, (ii) shared constituents, (iii) discontinuous phrases, and (iv) extraposed clauses. These empty elements are the most difficult to handle: unlike in the case of other EEs, they can be dominated by virtually any nonterminal label, the antecedent typically does not c-command the EE (cf. Section 2.3), and they often pose a problem for existing linguistic theories as well. Fortunately, they are relatively infrequent, constituting

only 3.8% of the EEs in the development section (WSJ00). Below is an exhaustive list of pseudo-attach constructions. In the annotation scheme, we do not distinguish them from each other, and use the **PSEUDO–...** label uniformly. Pseudo-attachment always involves co-indexation. Table A.5 at the end of this section summarises the distribution of various pseudo-attach constructions in WSJ00.

A.3.1 Structural ambiguity

To represent this kind of pseudo-attach, the original PTB annotation scheme uses the label **PPA**, which stands for "permanent predictable ambiguity." This label is used only in the "cases in which one cannot tell even from context where a constituent should be attached. The default is to attach the constituent at the more likely site [...] and then to pseudo-attach it at all other plausible sites" (Bies *et al.* 1995, p. 102). In the example below (A.19), for instance, the phrase *on the printer* might be attached to either *the forms*, *the class or the forms*, or it could modify directly the verb *changes* as a PP-adverbial. Note that **PPA**s are very rare in the Treebank, since "finding potential ambiguities is difficult and time-consuming, especially when reading in context" (*ibid.*, p. 103).

```
(S (NP-SBJ PRO−NP)                                              (A.19)
   (VP Use
       (NP this option)
       (SBAR-TMP (WHADVP2 when)
          (S (NP-SBJ the operator)
             (VP changes
                 (NP (NP (NP the class)
                         or
                         (NP (NP the forms)
                             (PP-LOC1 on
                                (NP the printer))))
                     (PP-LOC *PPA*1))
                 (PP-LOC *PPA*1)
                 (ADVP-TMP WH−ADVP2))))))
```

A.3.2 Shared constituents

These cases involve shared constituents in right node raising constructions, i.e., where a constituent should be interpreted simultaneously at more than one place. The original annotation scheme uses the label **RNR** to indicate these cases.

```
(S But                                                          (A.20)
   (NP-SBJ2 our outlook)
```

```
(VP (VP has
        (VP been
            (ADJP-PRD *RNR*1)))
    ,
    and
    (VP continues
        (S (NP-SBJ NP−NP2)
           (VP to
               (VP be
                   (ADJP-PRD *RNR*1)))))
    ,
    (ADJP-PRD1 defensive)))
```

A.3.3 Discontinuous structures

This type of pseudo-attach is the most common in the Treebank, and it is used to indicate a constituency of elements separated by intervening material (discontinuous phrases). Hence its label: *ICH*, which stands for "interpret constituent here." Typical use of this EE-type involves heavy shift and other adjunction phenomena.

(a) (A.21)

```
(S (NP-SBJ (NP a young woman)
           (SBAR *ICH*1))
   (VP entered
       (SBAR1 (WHNP2 whom)
              (S (NP-SBJ she)
                 (PP-TMP at
                         (ADVP once))
                 (VP recognized
                     (NP WH−NP2)
                     (PP-CLR as
                        (NP Jemima Broadwood)))))))
```

(b)

```
(S (NP-SBJ2 (NP Nothing)
            (PP *ICH*1))
   (VP was
       (S (NP-SBJ3 NP−NP2)
          (VP to
              (VP be
                  (VP seen
                      (NP NP−NP3)))))
       (PP1 but
            (NP water))))
```

A.3.4 *It*-extraposition

This EE-type is used in cases where the clausal subject of a sentence is extraposed and is replaced by an expletive *it*. The PTB annotation scheme refers to these instances with the label *EXP* (expletive).

```
(S (NP-SBJ (NP It)                                          (A.22)
           (SBAR *EXP*$_1$))
   (VP is
       (ADJP-PRD clear)
       (PP to
           (NP me))
       (SBAR$_1$ that
              (S (NP-SBJ this message)
                 (VP is
                     (ADJP-PRD unclear))))))
```

Type	Number
PPA	2
RNR	22
ICH	75
EXP	28
All	127

Table A.5: The distribution of various pseudo-attachments in WSJ00.

A.4 ELLIPSIS–...: placeholder for ellipsed material

This EE, labelled as *?* in the original scheme, acts as a placeholder for an ellipsed predicate or a part thereof. Although, in general, this EE is used by the annotators as a last resort, there are some constructions characteristic of **ELLIPSIS**. Note that even if the missing material represented by this EE is often identical to another constituent in the sentence, they are never co-indexed. Below we give a list of common constructions involving **ELLIPSIS**.

Comparative deletion

The ellipsed part of a complement clause is marked with **ELLIPSIS–...**. The EE might be dominated by nodes labelled with a –PRD functional tag (A.23a), but also by an NP (A.23b), by a VP (A.23c), or even by a clausal S label (A.23d).

(A.23)

(a)
```
(S (NP-SBJ John)
   (VP is
       (ADJP-PRD (ADJP sillier)
          (SBAR than
             (S (NP-SBJ I)
                (VP am
                    (ADJP-PRD ELLIPSIS−ADJP)))))))
```

(b)
```
(S (NP-SBJ the Controller)
   (VP will
       (VP have
           (NP (NP the opportunity)
               (PP for
                   (NP (NP greater usefulness)
                       (PP to
                           (NP good government))
                       (SBAR than
                           (S (NP-SBJ he)
                              (VP has
                                  (NP ELLIPSIS−NP)
                                  (ADVP-TMP now)
                              )))))))))
```

(c)
```
(S (NP-SBJ Bill)
   (VP eats
       (NP (NP more hotdogs)
           (SBAR than
                 (S (NP-SBJ Mary)
                    (VP does
                        (VP ELLIPSIS−VP)))))))
```

(d)
```
(S (NP-SBJ the steel strike)
   (VP lasted
       (ADVP-TMP (ADVP much longer)
          (SBAR than
                (S (NP-SBJ he)
                   (VP anticipated
                       (SBAR COMP−SBAR
                         (S ELLIPSIS−S)))))))))
```

Missing VP after auxiliaries

Examples of ellipsed VPs after auxiliaries include VPs in the second conjunct (A.24a), VPs in subordinate clauses (A.24b) and in *as do* constructions (A.24c).

(A.24)

(a)
```
(S (S (NP-SBJ She)
      (ADVP-TMP rarely)
      (VP sings))
   ,
   so
   (S (NP-SBJ I)
      (VP do n't
          (VP think
              (SBAR COMP-SBAR
                 (S (NP-SBJ she)
                    (VP will
                        (VP ELLIPSIS-VP
                           (NP-TMP tonight)))))))))
```

(b)
```
(S (NP-SBJ PRO-NP)
   (VP Call
       (S (NP-SBJ it)
          (ADJP-PRD anecdotal))
       (SBAR-ADV if
                 (S (NP-SBJ you)
                    (VP will
                        (VP ELLIPSIS-VP))))))
```

(c)
```
(S (NP-SBJ (NP Warner)
           and
           (NP Mr. Azoff))
   (VP declined
       (NP comment)
       ,
       (SBAR-ADV as
                 (SINV did
                       (NP-SBJ MCA)
                       (VP ELLIPSIS-VP)))))
```

A.5 COMP-...: empty complementisers

This label is inserted in SBAR constituents without overt complementisers. It comes in two flavours: (i) null WH-operators (**COMP-WH...**) and (ii) subordinators for tensed complement clauses (**COMP-SBAR**). Neither of them has an antecedent; the original PTB annotation scheme represents both with the label `0`. Table A.6 summarises the distribution of various **COMP-...** EEs in the development section (WSJ00).

A.5.1 COMP−SBAR

It introduces most tensed complement clauses without an overt complementiser. The clause can be a complement of adjectives (A.25a), a verbal complement (A.25b) or a nominal complement (A.25c). A very frequent occurrence of **COMP−SBAR** involves quotations, where the annotation appears to be rather unreliable. Consider, for instance, the examples in (A.25d) and (A.25e): they appear to have similar structure, nevertheless (A.25d) does not contain a **COMP−SBAR** for no apparent reason.

(A.25)

(a)
```
(S (NP-SBJ I)
   (VP 'm
       (ADJP-PRD sure
           (SBAR COMP−SBAR
              (S (NP-SBJ he)
                 (VP 'll
                     (VP be
                         (ADVP-LOC-PRD here)
                         (NP-TMP any minute)))))))))
```

(b)
```
(S (NP-SBJ I)
   (VP believe
       (SBAR COMP−SBAR
             (S (NP-SBJ you)
                (VP are
                    (ADJP-PRD ELLPSIS−ADJP))))))
```

(c)
```
(S (PP in
       (NP the event
           (SBAR COMP−SBAR
              (S (NP-SBJ Congress)
                 (VP does
                     (VP provide
                         (NP (NP this increase)
                             (PP in
                                 (NP federal funds)
                             ))))))))
   ,
   (NP-SBJ the State Board)
   (VP should ...))
```

(d) *"The refund pool . . . may not be held hostage through another round of appeals," Judge Curry said.* (wsj_0015.mrg)

```
(S (S-TPC-1 (NP-SBJ the refund pool)
            (VP may not
            ... ))
   (NP-SBJ Judge Curry)
   (VP said
       (S TOP-S-1)))
```

(e) *"In Asia, as in Europe, a new order is taking shape," Mr. Baker said.* (wsj_0043.mrg)

```
(S (S-1 In Asia ...)
   (NP-SBJ Mr. Baker)
   (VP said
       (SBAR COMP-SBAR
             (S TOP-S-1))))
```

A.5.2 COMP–WH...

These empty WH-operators introduce two types of relative clauses: finite relative clauses without an overt WH-complementiser and infinitival relative clauses. They are always antecedents of the corresponding **WH–...** traces.

Finite relatives

The null complementiser is labelled **COMP–WHNP** if it corresponds to *who, that* or *which* (A.26a), whereas it is labelled as **COMP–WHADVP** if it stands for *where, when, why* or *how* (A.26b).

(a)
```
(NP (NP the bird)                                    (A.26)
    (SBAR (WHNP-1 COMP-WHNP)
          (S (NP-SBJ I)
             (VP saw
                 (NP WH-NP-1)))))
```

(b)
```
(NP (NP the place)
    (SBAR (WHADVP-1 COMP-WHADVP)
          (S (NP-SBJ I)
             (VP put
                 (NP the book)
                 (ADVP-PUT WH-ADVP-1)))))
```

Infinitival relatives

Here **COMP–WHNP** is used for extracted NP subjects and objects (A.27a), whereas the empty WH-operator is labelled **COMP–WHADVP** when it can be paraphrased as *in which, for which, at which,* etc. (A.27b).

(A.27)

(a)
```
(NP (NP a movie)
    (SBAR (WHNP1 COMP–WHNP)
          (S (NP-SBJ PRO–NP)
             (VP to
                 (VP see
                     (NP WH–NP1))))))
```

(b)
```
(S (NP-SBJ That)
   (VP 's
       (NP-PRD (NP a good way)
          (SBAR (WHADVP1 COMP–WHADVP)
             (S (NP-SBJ PRO–NP)
                (VP to
                    (VP keep
                        (ADJP-PRD warm)
                        (ADVP-MNR WH–ADVP1)))))))))
```

Type	Number
COMP–SBAR	456
COMP–WHNP	98
COMP–WHADVP	30
All	584

Table A.6: The distribution of empty complementisers (**COMP–...**) in WSJ00.

A.6 TU: empty units

This empty element, represented in the original annotation scheme with *U*, "marks the interpreted position of a unit symbol, such as $, # (British pounds), FFr (French francs), C$, US$, HK$, A$, M$, S$, and NZ$. It may also appear after % or even *cents*, when convenient" (Bies *et al.* 1995, p. 80). The "interpreted position" is where the symbol would appear when the text is pronounced

(A.28a–c). There is one notable exception shown in (A.28d). Observe the somewhat unintuitive choice of annotation: generally, the numbers and the currency sign are attached immediately to the NP, unless they are followed by the words *million*, *billion*, etc. (A.28b) or they are modified or participate in a conjunction (A.28c).

(A.28)

(a) `(NP C$ 5 UNIT)`

(b) `(NP (QP  C$ 5 million) UNIT)`

(c) `(NP (QP between $ 5 and $ 15) UNIT)`

(d)
```
(NP a
    (ADJP $ 5-a-share UNIT)
    increase)
```

List of abbreviations

DOP	Data-Oriented Parsing
EE	empty element
EM	Expectiation Maximisation
GIS	Generalised Iterative Scaling
HD	head–dependent
HMM	hidden Markov model
LFG	Lexical-Functional Grammar
LHS	left hand side (of a context free rule)
ME	Maximum Entropy
NER	Named Entity Recognition
NLD	non-local dependency
NLP	Natural Language Processing
(P)CFG	(Probabilistic) Context-Free Grammar
(P)DPSG	(Probabilistic) Discontinuous Phrase Structure Grammar
PTB	Penn Treebank
RHS	right hand side (of a context free rule)
TAG	Tree-Adjoining Grammar
TIG	Tree-Insertion Grammar
TSG	Tree-Substitution Grammar
UBG	Unification-Based Grammar
WSJ23	Section 23 of the Wall Street Journal corpus

Citation Index

Bibliography

Steven ABNEY (1991). Parsing by chunks. In Robert BERWICK, Steven ABNEY and Carol TENNY, (eds.), *Principle-Based Parsing*. Kluwer Academic Publishers, Dordrecht.

Steven P. ABNEY (1997). Stochastic attribute-value grammars. *Computational Linguistics*, **23**(4):597–618.

David W. AHA, Dennis KIBLER and Marc K. ALBERT (1991). Instance-based learning algorithms. *Machine Learning*, **6**(1):37–66.

Jason BALDRIDGE and Miles OSBORNE (2003). Active learning for HPSG parse selection. In *Proceedings of the 7th Conference on Natural Language Learning (CoNLL-2003)*, 17–24. Edmonton, Canada.

Timothy BALDWIN, Emily M. BENDER, Dan FLICKINGER, Ara KIM and Stephan OEPEN (2004). Road-testing the English Resource Grammar over the British National Corpus. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC2004*. Lisbon, Portugal.

Iain BANCARZ and Miles OSBORNE (2002). Improved Iterative Scaling can yield multiple globally optimal models with radically differing performance levels. In *Proceedings of the 19th International Conference on Computational Linguistics*, 43–49. Taipei, Taiwan.

Markus BECKER and Anette FRANK (2002). A stochastic topological parser for German. In *Proceedings of the 19th International Conference on Computational Linguistics*, 71–77. Taipei, Taiwan.

Adam L. BERGER, Stephen A. DELLA PIETRA and Vincent J. DELLA PIETRA (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, **22**(1):39–71.

Ann BIES, Mark FERGUSON, Karen KATZ and Robert MACINTYRE (1995). *Bracketting Guidelines for Treebank II style Penn Treebank Project*. Linguistic Data Consortium.

Daniel M. BIKEL, Ralph M. WEISCHEDEL and Richard SCHWARTZ (1999). An algorithm that learns what's in a name. *Machine Learning*, **34**(1–3):211–231.

Ezra BLACK, Frederick JELINEK, John LAFFERTY, David M. MAGERMAN, Robert L. MERCER and Salim ROUKOS (1992a). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*. Harriman, NY.

Ezra BLACK, John LAFFERTY and Salim ROUKOS (1992b). Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 185–192. Newark, DE.

Don BLAHETA and Eugene CHARNIAK (2000). Assigning function tags to parsed text. In *Proceedings of the 1st Meeting of the North American Chapter of the Association For Computational Linguistics*, 234–240. Seattle, WA.

James P. BLEVINS (1990). *Syntatcic complexity: Evidence for discontinuity and multidomination*. Ph.D. thesis, University of Massachusetts.

Rens BOD (1992). A computational model for performance: Data Oriented Parsing. In *Proceedings of the 15th International Conference on Computational Linguistics*, 855–859. Nantes, France.

Rens BOD (1995a). *Enriching Linguistics with Statistics: Performance Models of Natural Language*. Academische Pers, The Netherlands. Ph.D. thesis.

Rens BOD (1995b). The problem of computing the most probable tree in Data-Oriented Parsing and Stochastic Tree Grammars. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, 104–111. Dublin, Ireland.

Rens BOD (2000). Parsing with the shortest derivation. In *Proceedings of the 18th International Conference on Computational Linguistics*, 69–75. Saarbrücken, Germany.

Rens BOD (2001). What is the minimal set of fragments which obtains maximum parse accuracy? In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 66–73. Toulouse, France.

Rens BOD (2003). An efficient implementation of a new DOP model. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, 19–26. Budapest, Hungary.

Rens BOD and Ronald KAPLAN (1998). A probabilistic corpus-driven model for Lexical-Functional Analysis. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguisticsand the 17th International Conference on Computational Linguistics*, 145–151. Montreal, Canada.

Rens BOD and Ronald KAPLAN (2003). A DOP model for Lexical-Functional Grammar. In Rens BOD, Remko SCHA and Khalil SIMA'AN, (eds.), *Data-Oriented Parsing*, CSLI Publications. University of Chicago Press.

Remko BONNEMA (2003). Probability models for DOP. In Rens BOD, Remko SCHA and Khalil SIMA'AN, (eds.), *Data-Oriented Parsing*, CSLI Publications. University of Chicago Press.

Taylor L. BOOTH and Richard A. THOMPSON (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers*, **C-22**(5):442–450.

Gosse BOUMA, Gertjan VAN NOORD and Robert MALOUF (2001). Alpino: Wide-coverage computational analysis of Dutch. In Walter DAELEMANS, Khalil SIMA'AN, Jorn VEENTSTRA and Jakub ZAVREL, (eds.), *Computational Linguistics in The Netherlands 2000*, 45–59. Rodopi, Amsterdam.

Thorsten BRANTS (1995). Tagset reduction without information loss. In *Proceedings of 33rd Annual Meeting of the Association for Computational Linguistics*, 287–289. Cambridge, MA.

Thorsten BRANTS (1999a). Cascaded Markov Models. In *Proceedings of 9th Conference of the European Chapter of the Association for Computational Linguistics*, 117–125. Bergen, Norway.

Thorsten BRANTS (1999b). *Tagging and Parsing with Cascaded Markov Models*, vol. 6 of *Saarbrücken Dissertations in Computational Linguistics and Language Technology*. DFKI&Saarland University, Saarbrücken.

Thorsten BRANTS (2000). TnT – A statistical part-of-speech tagger. In *Proceedings of the 6th Applied Natural Language Processing Conference ANLP-2000*, 224–231. Seattle, WA.

Joan BRESNAN (1976). Evidence for a theory of unbounded transformations. *Linguistic Analysis*, **2**:353–399.

Chris BREW (1995). Stochastic HPSG. In *Proceedings of 7th Conference of the European Chapter of the Association for Computational Linguistics*, 83–89. Dublin, Ireland.

Eric BRILL (1992). A simple rule-based part of speech tagger. In *Proceedings of the 3rd Conference on Applied Natural Language Processing*. Trento, Italy.

Eric BRILL (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, **21**:543–565.

Ted BRISCOE (1987). Deterministic parsing and unbounded dependencies. In *Proceedings of the 3rd Conference of the European Chapter of the Association for Computational Linguistics*, 211–217. Copenhagen, Denmark.

Ted BRISCOE and John CARROLL (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, **19**(1):25–59.

Norbert BRÖKER (1997). *Eine Dependenzgrammatik zur Kopplung heterogener Wissensysteme auf modallogischer Basis*. Ph.D. thesis, Philosophische Fakultät, Albert-Ludwigs-Universität, Freiburg, Germany.

Sabine BUCHHOLZ (2002). *Memory-Based Grammatical Relation Finding*. Ph.D. thesis, Tilburg University.

Harry BUNT (1991). Parsing with discontinuous phrase structure grammar. In Masaru TOMITA, (ed.), *Current issues in parsing technology*, 49–63. Kluwer Academic Publishers, Dordrecht, Boston, London.

Harry BUNT (1996). Formal tools for describing and processing discontinuous constituency structure. In Harry BUNT and Arthur VAN HORCK, (eds.), *Discontinuous Constituency*, 63–83. Mouton de Gruyter, Berlin, New York.

Harry BUNT, Jan THESINGH and Ko VAN DER SLOOT (1987). Discontinuities in trees, rules and parsing. In *Proceedings of the 3rd Conference of the European Chapter of the Association for Computational Linguistics*, 203–211. Copenhagen, Denmark.

Miriam BUTT, Tracy KING, Maria-Eugenia NIÑO and Frédérique SEGOND (1999). *A Grammar Writer's Cookbook*. No. 95 in CSLI Lecture Notes. CSLI Publications, Stanford, CA.

Aoife CAHILL, Mairead MCCARTHY, Josef VAN GENABITH and Andy WAY (2002). Automatic annotation of the Penn-Treebank with LFG F-structure information. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation, LREC2002*. Las Palmas, Spain.

Ulrich CALLMEIER (2000). PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, **6**(1):99–108.

Glenn CARROLL and Eugene CHARNIAK (1992). Two expeirments on learning probabilistic dependency grammars from corpora. Tech. Rep. CS-92-16, Department of Computer Science, Brown University, Providence, RI.

John CARROLL, Ted BRISCOE and Antonio SANFILIPPO (1998). Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st International Conference on Language Resources and Evaluation, LREC1998*, 447–454. Granada, Spain.

John CARROLL, Anette FRANK, Dekang LIN, Detlef PRESCHER and Hans USZKOREIT, (eds.) (2002). *Proceedings of the LREC Workshop "Beyond PARSEVAL"*. Las Palmas, Spain.

John CARROLL, Guido MINNEN and Ted BRISCOE (1999). Corpus annotation for parser evaluation. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora (LINC'99)*, 35–41. Bergen, Norway.

John CARROLL and David WEIR (1997). Encoding frequency information in lexicalized grammars. In *Proceedings of the 5th ACL/SIGPARSE International Workshop on Parsing Technologies*, 8–17.

John CARROLL and David WEIR (2003). Encoding frequency information in stochastic parsing models. In Rens BOD, Remko SCHA and Khalil SIMA'AN, (eds.), *Data-Oriented Parsing*, CSLI Publications. University of Chicago Press.

Eugene CHARNIAK (1993). *Statistical Language Learning*. The MIT Press, Cambridge, MA.

Eugene CHARNIAK (1996). Tree-bank grammars. In *AAAI/IAAI, Vol. 2*, 1031–1036.

Eugene CHARNIAK (1997). Statistical techniques for natural language parsing. *AI Magazine*.

Eugene CHARNIAK (2000). A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association For Computational Linguistics*, 132–139. Seattle, WA.

Stanley F. CHEN and Ronald ROSENFELD (1999). A Gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Carnegie Mellon University.

Zhiyi CHI and Stuart GEMAN (1998). Estimation of probabilistic context-free grammars. *Computational Linguistics*, **24**(2):299–305.

David CHIANG (2000). Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, 456–463. Hong Kong, China.

David CHIANG (2003). Statistical parsing with an automatically extracted tree adjoining grammar. In Rens BOD, Remko SCHA and Khalil SIMA'AN, (eds.), *Data-Oriented Parsing*, CSLI Publications. University of Chicago Press.

Mahesh V. CHITARO and Ralph GRISHMAN (1990). Statistical parsing of messages. In *Proceedings of the DARPA Speech and Natural Language Workshop*, 263–266. Hidden Valley, PA.

Noam CHOMSKY (1957). *Syntactic Structures*. Mouton, The Hague.

Noam CHOMSKY (1965). *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.

Noam CHOMSKY (1977). On Wh-movement. In Peter CULICOVER, Thomas WASOW and Adrian AKMAJIAN, (eds.), *Formal Syntax*, 71–132. Academic Press, New York.

Kenneth CHURCH and Ramesh PATIL (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, **8**(3–4):139–149.

Kenneth Ward CHURCH (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceeding of the 2nd Conference on Applied Natural Language Processing*, 136–143. Texas, Austin.

Stephen CLARK (2002). A supertagger for Combinatory Categorial Grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks*, 19–24. Venice, Italy.

Stephen CLARK, Julia HOCKENMAIER and Mark STEEDMAN (2002). Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 327–334. Philadelphia, PA.

Michael COLLINS (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the 8th Conference of the European Chapter of the Association for Computational Linguistics*, 16–23. Madrid, Spain.

Michael COLLINS (1999). *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

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

Michael COLLINS and Nigel DUFFY (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 263–270. Philadelphia, PA.

Michael John COLLINS (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 184–191. Santa Cruz, CA.

Ann COPESTAKE and Dan FLICKINGER (2000). An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000)*. Athens, Greece.

Michael COVINGTON (1990a). A dependency parser for variable-word-order languages. Tech. Rep. Research Report AI-1990-01, Artificial Intelligence Programs, The University of Georgia, Athens, GA.

Michael COVINGTON (1994). Discontinuous dependency parsing of free and fixed word order: Work in progress. Tech. Rep. Research Report AI-1994-02, Artificial Intelligence Programs, The University of Georgia, Athens, GA.

Michael A. COVINGTON (1990b). Parsing discontinuous constituents in dependency grammar. *Computational Linguistics*, **16**(4):234–236.

Berthold CRYSMANN, Anette FRANK, Bernd KIEFER, Stephan MÜLLER, Günter NEUMANN, Jakub PISKORSKI, Ulrich SCHÄFER, Melanie SIEGEL, Hans USZKOREIT, Feiyu XU, Markus BECKER and Ulrich HANS-KRIEGER (2002). An integrated architecture for deep and shallow processing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 441–448. Pittsburgh, PA.

James R. CURRAN and Stephen CLARK (2003). Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 91–98. Budapest, Hungary.

Walter DAELEMANS, Jakub ZAVREL, Ko VAN DER SLOOT and Antal VAN DEN BOSCH (2002). Timbl: Tilburg memory based learner v.4.3 reference guide. Tech. rep., ILK.

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

Michael DAUM, Kilian A. FOTH and Wolfgang MENZEL (2003). Constraint based integration of deep and shallow parsing techniques. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 99–106. Budapest, Hungary.

Stephen DELLA PIETRA, Vincent DELLA PIETRA and John LAFFERTY (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**:380–393.

A. P. DEMPSTER, N. M. LAIRD and D. B. RUBIN (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistics Society*, **39**(B):1–38.

Péter DIENES and Amit DUBEY (2003a). Antecedent recovery: Experiments with a trace tagger. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, 33–40. Sapporo, Japan.

Péter DIENES and Amit DUBEY (2003b). Deep syntactic processing by combining shallow methods. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 431–438. Sapporo, Japan.

Péter DIENES, Alexander KOLLER and Marco KUHLMANN (2003). Statistical A* dependency parsing. In *Prospects and Advances in the Syntax/Semantics Interface*, 85–89. Nancy, France.

Péter DIENES and Csaba ORAVECZ (2000). Bottom-up tagset design from maximally reduced tagset. In *Proceedings of the Workshop on Linguistically Interpreted Corpora, LINC2000*, 42–47.

Christy DORAN, Dania EGEDI, Ann Beth HOCKEY, B. SRINIVAS and Martin ZAIDEL (1994). The XTAG system – a wide coverage grammar for English. In *Proceedings of the 17th International Conference on Computational Linguistics*, 922–928. Kyoto, Japan.

Denys DUCHIER and Ralph DEBUSMANN (2001). Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 180–187. Toulouse, France.

Andreas EISELE (1994). Towards probabilistic extensions of constraint-based grammars. Computational Aspects of Constraint-Based Linguistic Description II, pp. 321. DYANA-2 Deliverable R1.2.B.

Jason EISNER (1996a). An empirical comparison of probability models for dependency grammar. Tech. rep., University of Pennsylvania. IRCS-96-11.

Jason EISNER (1996b). Three new probabilistic models for dependency parsing: an exploration. In *Proceedings of the 16th International Conference on Computational Linguistics*, 340–345. Copenhagen, Denmark.

Tomaž ERJAVEC, Sašo DŽEROSKI and Jakub ZAVREL (1999). Morphosyntactic tagging of Slovene: Evaluating PoS taggers and tagsets. Tech. Rep. 8018, Department of Intelligent Systems, Jožef Stefan Institute, Ljubljana.

Yehuda N. FALK (2003). The English auxiliary system revisited. In *Proceedings of the LFG03 Conference*, 184–204. Albany, NY.

Radu FLORIAN, Abe ITTYCHERIAH, Hongyan JING and Tong ZHANG (2003). Named entity recognition through classifier combination. In *Proceedings of the 7th Conference on Natural Language Learning (CoNLL-2003)*, 168–171. Edmonton, Canada.

Eva Wai-man FONG and Dekai WU (1995). Learning restricted probabilistic link grammars. In *Proceedings of IJCAII Workshop on New Approaches to Learning for Natural Language Processing*, 49–56. Montréal, Canada.

Anette FRANK, Markus BECKER, Berthold CRYSMANN, Bernd KIEFER and Ulrich SCHÄFER (2003). Integrated shallow and deep parsing: TopP meets HPSG. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 104–111. Sapporo, Japan.

Malte GABSDIL (2004). Combining acoustic confidences and pragmatic plausibility for classifying spoken chess move instructions. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*. Cambridge, MA.

Haim GAIFMAN (1965). Dependency systems and phrase-structure systems. *Information and Control*, **8**(3):304–337.

Gerald GAZDAR (1981). Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, **12**(2):155–184.

Gerald GAZDAR, Ewan KLEIN, Geoffrey PULLUM and Ivan SAG (1985). *Generalized Phrase Structure Grammar*. Blackwell, Oxford, UK.

Stuart GEMAN and Mark JOHNSON (2002). Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 279–286. Pittsburgh, PA.

Daniel GILDEA (2001). Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 167–202. Pittsburgh, PA.

Joshua GOODMAN (1996). Efficient algorithms for parsing the DOP model. In *Proceedings of the 1996 Conference on Empirical Methods in Natural Language Processing*, 143–152. Philadelphia, PA.

Joshua GOODMAN (1997). Probabilistic feature grammars. In *Proceedings of the 5th International Workshop on Parsing Technologies (IPWT'97)*. Boston, MA.

Joshua GOODMAN (1998). *Parsing inside-out*. Ph.D. thesis, Harvard University.

Joshua GOODMAN (2002). Sequential conditional Generalized Iterative Scaling. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 9–16. Philadelphia, Pennsylvania.

Joshua GOODMAN (2003). Efficient parsing of DOP with PCFG-reductions. In Rens BOD, Remko SCHA and Khalil SIMA'AN, (eds.), *Data-Oriented Parsing*, CSLI Publications. University of Chicago Press.

Dennis GRINBERG, John LAFFERTY and Daniel SLEATOR (1995). A robust parsing algorithm for link grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies (IPWT'95)*, 111–125. Prague, Czech Rep. Also available as Carnegie Mellon University Computer Science technical report CMU-CS-95-125.

Maurice GROSS (1964). The equivalence of models of language used in the fields of mechanical translation and information retrieval. *Information Storage and Retrieval*, **2**:43–57.

Liliane HAEGEMAN (1991). *Introduction to Government and Binding Theory*. Blackwell, Oxford, UK and Cambridge, USA.

Robin HAIGH, Geoffrey SAMPSON and Eric ATWELL (1988). Project APRIL – A progress report. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, 104–112. Buffalo, NY.

Jan HAJIČ (2000). Morphological tagging: Data vs. Dictionaries. In *Proceedings of 1st Meeting of the North American Chapter of the Association For Computational Linguistics*, 94–101. Seattle, WA.

Jan HAJIČ and Barbora HLADKA (1998). Tagging inflective languages: prediction of morphological categories for a rich structured tagset. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguisticsand the 17th International Conference on Computational Linguistics*, 483–490. Montreal, Canada.

Gilbert HARMAN (1963). Generative grammars without transformation rules: a defense of phrase structure. *Language*, **39**:597–616. Reprinted in Walter J. Savitch, Emmon Bach, William Marsh and Gila Safran-Naveh (eds), *The Formal Complexity of Natural Language* 87-116. Dordrecht, Holland; D. Reidel: 1987.

Philip HARRISON, Steven ABNEY, Ezra BLACK, Dan FLICKINGER, Ralph GRISHMAN, Claudia GDANIEC, Donald HINDLE, Robert INGRIA, Mitch MARCUS, Beatrice SANTORINI and Tomek STRZALKOWSKI (1991). Evaluating syntax performance of parser/grammars of English. In *Proceedings of the Workshop on Evaluating Natural Language Processing Systems*, 71–78. Berkley, CA.

David G. HAYS (1964). Dependency theory: A formalism and some observations. *Language*, **40**:511–525.

James HENDERSON (2003). Neural network probability estimation for broad coverage parsing. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 131–138. Budapest, Hungary.

Derrick HIGGINS (2003). A machine-learning approach to the identification of wh gaps. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 99–102. Budapest, Hungary.

Julia HOCKENMAIER (2003a). *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, Institute for Communicating and Collaborative Systems, School of Informatics, University of Edinburgh.

Julia HOCKENMAIER (2003b). Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 359–366. Sapporo, Japan.

Julia HOCKENMAIER and Mark STEEDMAN (2002a). Acquiring compact lexicalised grammars from a cleaner Treebank. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC2000)*, 1974–1981. Las Palmas, Spain.

Julia HOCKENMAIER and Mark STEEDMAN (2002b). Models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 335–342. Philadelphia, PA.

Richard A. HUDSON (1990). *English Word Grammar*. Blackwell, Oxford.

Rebecca HWA (1998). An empirical evaluation of probabilistic tree insertion grammars. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguisticsand the 17th International Conference on Computational Linguistics*, 557–563. Montreal, Canada.

Frederik JELINEK, John LAFFERTY, David M. MAGERMAN, Robert L. MERCER, Adwait RATNAPARKHI and Salim ROUKOS (1994). Decision tree parsing using a hidden derivation model. In *Proceedings of the 1994 Human Language Technology Workshop*, 272–277. DARPA.

Valentin JIJKOUN (2003). Finding non-local dependencies: Beyond pattern matching. In *Proceedings of the ACL2003 Student Research Workshop*, 37–43. Sapporo, Japan.

Mark JOHNSON (1985). Parsing with discontinuous constituents. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, 127–132. Chicago, IL.

Mark JOHNSON (1998). PCFG models of linguistic tree representations. *Computational Linguistics*, **24**(4):613–632.

Mark JOHNSON (2001). Joint and conditional estimation of tagging and parsing models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 314–321. Toulouse, France.

Mark JOHNSON (2002). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 136–143. Philadelphia, PA.

Mark JOHNSON (2003). Learning and parsing stochastic unification-based grammars. In Bernhard SCHÖLKOPF and Manfred K. WARMUTH, (eds.), *Learning Theory and Kernel Machines*, 671–683. Springer Verlag. Proceedings of the 16th Annual Conference on Computational Learning Theory, COLT 2003, and the 7th Kernel Workshop, Kernel 2003, held in Washington, DC in August 2003.

Mark JOHNSON, Stuart GEMAN, Stephen CANON, Zhiyi CHI and Stefan RIEZLER (1999). Estimators for stochastic 'unification-based' grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 535–541. College Park, MD.

Mark JOHNSON and Martin KAY (1994). Parsing and empty nodes. *Computational Linguistics*, **20**(2):289–300.

Aravind K. JOSHI, Leon S. LEVY and Masako TAKAHASHI (1975). Tree adjunct grammars. *Journal of Computer and System Sciences*, **10**(1):136–163.

Aravind K. JOSHI and Yves SCHABES (1997). Tree-adjoining grammars. In G. ROZENBERG and A. SALOMAA, (eds.), *Handbook of Formal Languages*, 69–124. Springer Verlag, Berlin, New York.

Aravind K. JOSHI and Bangalore SRINIVAS (1994). Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th International Conference on Computational Linguistics*, 154–160. Kyoto, Japan.

Ronald M. KAPLAN and Joan BRESNAN (1982). Lexical-functional grammar: A formal system for grammatical representations. In Joan BRESNAN, (ed.), *The Mental Representation of Grammatical Relations*, 173–281. The MIT Press, Cambridge, MA.

Ronald M. KAPLAN and John T. MAXWELL III (1995). An algorithm for functional uncertainty. In Mary DALRYMPLE, Ronald M. KAPLAN, John T. MAXWELL III and Annie ZAENEN, (eds.), *Formal Issues in Lexical-Functional Grammar*, no. 47 in CSLI Lecture Note Series, 177–197. CSLI Publications.

Martin KAY (1986 (1980)). Algorithm schemata and data structures in syntactic processing. In Barbara J. GROSZ, Karen SPARCK-JONES and Bonnie Lynn WEBBER, (eds.), *Readings in Natural Language Processing*, 35–70. Morgan Kaufmann, Los Altos, CA.

Bernd KIEFER and Hans-Ulrich KRIEGER (2000). A context-free approximation of head-driven phrase structure grammar. In *Proceedings of the 6th International Workshop on Parsing Technologies*, 49–80. Trento, Italy.

Bernd KIEFER and Hans-Ulrich KRIEGER (2002). A context-free approximation of head-driven phrase structure grammar. In Stephan OEPEN, Dan FLICKINGER, Jun-ichi TSUJII and Hans USZKOREIT, (eds.), *Collaborative Language Engineering. A Case Study in Efficient Grammar-Based Processing*. CSLI Publications, Stanford, CA.

Bernd KIEFER, Hans-Ulrich KRIEGER and Detlef PRESCHER (2002). A novel disambiguation method for unification-based grammars using probabilistic context-free approximations. In *Proceedings of the 19th International Conference on Computational Linguistics*, 439–445. Taipei, Taiwan.

Dan KLEIN and Christopher D. MANNING (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 423–430. Sapporo, Japan.

Alexander KOLLER and Kristina STRIEGNITZ (2002). Generation as dependency parsing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 17–24. Philadelphia, PA.

Christian KORTHALS (2002). Corpus induction of dependency grammar word order rules. In *Proceedings of the Workshop an Linguistic Corpora and Logic Based Grammar Formalisms*, 38–51. Utrecth, The Netherlands.

Brigitte KRENN, Wojciech SKUT and Thorsten BRANTS (1998). Construction of a linguistically interpreted German newspaper corpus. In *Proceedings of the Conference on Language Resources and Evaluation (LREC1998)*, 705–711. Granada, Spain.

John LAFFERTY, Andrew MCCALLUM and Fernando PEREIRA (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, 282–289. Morgan Kaufmann, San Francisco, CA.

John LAFFERTY, Daniel SLEATOR and Davy TEMPERLEY (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, 89–97. Cambridge, MA.

Raymond LAU, Ronald ROSENFELD and Salim ROUKOS (1993). Adaptive language modelling using the Maximum Entropy approach. In *Proceedings ARPA Human Language Technologies Workshop*, 81–86. Princeton, NJ.

Dekang LIN (1995). A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1420–1427. Montreal, Canada.

David M. MAGERMAN (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 276–283. Cambridge, MA.

David M. MAGERMAN and Mitchell P. MARCUS (1991). Pearl: A probabilistic chart parser. In *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics*, 15–20. Berlin, Germany.

David M. MAGERMAN and Carl WEIR (1992). Efficiency, robustness, and accuracy in Picky chart parsing. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 40–47. Newark, DE.

Robert MALOUF (2002a). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Workshop on Natural Language Learning (CoNLL-2002)*, 49–55. Taipei, Taiwan.

Robert MALOUF (2002b). Markov models for language-independent named entity recognition. In *Proceedings of Proceedings of the 6th Workshop on Natural Language Learning (CoNLL-2002)*, 155–158. Taipei, Taiwan.

Christopher D. MANNING and Hinrich SCHÜTZE (2001). *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA. & London, UK.

Mitchell MARCUS, Grace KIM, Mary Ann MARCINKIEWICZ, Robert MACINTYRE, Ann BIES, Mark FERGUSON, Karen KATZ and Britta SCHASBERGER (1994). The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the ARPA Human Language Technology Workshop*, 110–115.

Mitchell P. MARCUS, Beatrice SANTORINI and Mary Ann MARCINKIEWICZ (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, **19**(2):313–330.

John T. MAXWELL III and Ronald M. KAPLAN (1995a). The interface between phrasal and functional constraints. In Mary DALRYMPLE, Ronald M. KAPLAN, John T. MAXWELL III and Annie ZAENEN, (eds.), *Formal Issues in Lexical-Functional Grammar*, no. 47 in CSLI Lecture Note Series, 403–429. CSLI Publications.

John T. MAXWELL III and Ronald M. KAPLAN (1995b). A method for disjunctive constraint satisfaction. In Mary DALRYMPLE, Ronald M. KAPLAN, John T. MAXWELL III and Annie ZAENEN, (eds.), *Formal Issues in Lexical-Functional Grammar*, no. 47 in CSLI Lecture Note Series, 381–401. CSLI Publications.

Andrew MCCALLUM, Dayne FREITAG and Fernando PEREIRA (2000). Maximum entropy Markov models for information extraction and segmentation. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML 2000)*, 591–598. Stanford, CA.

Igor A. MEL'ČUK (1988). *Dependency syntax: theory and practice*. SUNY Series in Linguistics. State University of New York Press, Albany.

Stefan MÜLLER (1996). The Babel-system—an HPSG Prolog implementation. In *Proceedings of the 4th International Conference on the Practical Application of Prolog*, 263–277. London, UK.

Radford M. NEAL (1993). Probabilistic inference using Markov chain Monte Carlo methods. Tech. rep., Dept. of Computer Science, University of Toronto.

Peter NEUHAUS and Norbert BRÖKER (1997). The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the 8th Conference of the European Chapter of the Association for Computational Linguistics*, 337–343. Madrid, Spain.

Günter NEUMANN (2003). A data-driven approach to Head-driven Phrase-Structure Grammar. In Rens BOD, Remko SCHA and Khalil SIMA'AN, (eds.), *Data-Oriented Parsing*, CSLI Publications. University of Chicago Press.

Kamal NIGAM, John LAFFERTY and Andrew MCCALLUM (1999). Using maximum entropy for text classification. In *Proceedings of the IJCAI-99 Workshop on Machine Learning for Information Filtering*, 61–67. Stockholm, Sweden.

Joakim NIVRE (2002). Two models of stochastic dependency grammar. Tech. rep., Växjö University: School of Mathematics and Systems Engineering. MSI Report 02118.

Gertjan VAN NOORD (1991). Head corner parsing for discontinuous constituency. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 114–121. Berkeley, CA.

Franz Josef OCH and Hermann NEY (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 295–302. Philadelphia, PA.

Stephan OEPEN, Ezra CALLAHAN, Dan FLICKINGER and Christopher D. MANNING (2002a). LinGO Redwoods. a rich and dynamic treebank for HPSG. In *Proceedings of the Beyond PARSEVAL Workshop of the 3rd LREC Conference*. Las Palmas, Spain.

Stephan OEPEN, Dan FLICKINGER, Kristina TOUTANOVA and Christopher D. MANNING (2002b). LinGO Redwoods. a rich and dynamic treebank for HPSG.

In *Proceedings of the Workshop on Treebanks and Linguistic Theories (TLT)*, 139–145. Sozopol, Bulgaria.

Stephan OEPEN, Kristina TOUTANOVA, Stuart SHIEBER, Christopher D. MANNING, Dan FLICKINGER and Thorsten BRANTS (2002c). The LinGO Redwoods treebank: Motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics*, 1253–1257. Taipei, Taiwan.

Csaba ORAVECZ and Péter DIENES (2002). Efficient stochastic Part-of-Speech tagging for Hungarian. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC2002)*, 710–717. Las Palmas, Spain.

Miles OSBORNE (2000a). Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the 18th International Conference on Computational Linguistics*, 586–592. Saarbrücken, Germany.

Miles OSBORNE (2000b). Shallow parsing as part-of-speech tagging. In *Proceedings of the 4th Conference on Computational Language Learning (CoNLL-2000) and the 2nd Learning Language in Logic Workshop (LLL– 2000)*, 145–147. Lisbon, Portugal.

Miles OSBORNE (2002). Using maximum entropy for sentence extraction. In *Proceedings of the ACL 2002 Workshop on Automatic Summarization*, 1–8. Philadelphia, CA.

Miles OSBORNE and Ted BRISCOE (1997). Learning stochastic categorial grammars. In *Proceedings of the 1st Conference on Natural Language Learning (CoNLL-1997)*, 80–87. Madrid, Spain.

Fernando PEREIRA and Yves SCHABES (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 128–135. Newark, DE.

Oliver PLAEHN (1999). *Probabilistic Parsing with Discontinuous Phrase Structure Grammar*. Master's thesis, Saarland University.

Oliver PLAEHN (2000). Computing the most probable parse for a Discontinuous Phrase Structure Grammar. In *Proceedings of the 6th International Workshop on Parsing Technologies*, 195–206. Trento, Italy.

Carl J. POLLARD and Ivan A. SAG (1994). *Head-driven Phrase Structure Grammar*. University of Chicago Press.

Judita PREISS (2003). Using grammatical relations to compare parsers. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 291–298. Budapest, Hungary.

Robbert PRINS and Gertjan VAN NOORD (2001). Unsupervised POS-tagging improves parsing accuracy and parsing efficiency. In *Proceedings of the International Workshop on Parsing Technologies 2001*, 154–165. Beijing, China.

Derek PROUDIAN and Carl POLLARD (1985). Parsing head-driven phrase structure grammar. In *Proceedings of the 23th Annual Meeting of the Association for Computational Linguistics*, 167–171. Chicago, IL.

Owen RAMBOW and Aravind JOSHI (1994). A processing model for free word order languages. In Jr. CLIFTON C., L. FRAZIER and K. RAYNER, (eds.), *Perspectives on sentence processing*, 267–302. Lawrence Erlbaum Associates.

Lance RAMSHAW and Mitch MARCUS (1995). Text chunking using transformation-based learning. In David YAROVSKY and Kenneth CHURCH, (eds.), *Proceedings of the 3rd Workshop on Very Large Corpora*, 82–94. Association for Computational Linguistics, Somerset, New Jersey.

Adwait RATNAPARKHI (1996). A Maximum Entropy Part-of-Speech tagger. In *Proceedings of the 1996 Conference on Empirical Methods in Natural Language Processing*, 133–142. Philadelphia, PA.

Adwait RATNAPARKHI (1997). A linear observed time statistical parser based on Maximum Entropy models. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing*, 1–10. Providence, RI.

Adwait RATNAPARKHI (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

Mike REAPE (1991). Parsing bounded discontinuous constituents: Generalisations of some common algorithms. In Ton VAN DER WOUDEN and Wietske SIJTSMA, (eds.), *Computational Linguistics in the Netherlands. Papers from the First CLIN-meeting.*, 103–132. Utrecht University-OTS, Utrecht, The Netherlands.

Philip RESNIK (1992). Probabilistic tree-adjoining grammars as a framework for statistical natural language processing. In *Proceedings of the 15th International Conference on Computational Linguistics*, 418–424. Nantes, France.

Jeffrey C. REYNAR and Adwait RATNAPARKHI (1997). A Maximum Entropy approach to identifying sentence boundaries. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, 16–19. Washington, D.C.

Stefan RIEZLER (1999). *Probabilistic Constraint Logic Programming*. Ph.D. thesis, Universität Tübingen. AIMS Report, 5(1), IMS, Universität Stuttgart.

Stefan RIEZLER, Tracy H. KING, Ronald M. KAPLAN, Richard CROUCH, John T. MAXWELL III and Mark JOHNSON (2002). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 271–278. Philadelphia, PA.

Stefan RIEZLER, Detlef PRESCHER, Jonas KUHN and Mark JOHNSON (2000). Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, 480–487. Hong Kong, China.

Jane J. ROBINSON (1970). Dependency structures and transformational rules. *Language*, **46**:259–285.

Ronald ROSENFELD (1996). A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, **10**(2):187–228.

John Robert ROSS (1967). *Constraints on Variables in Syntax*. Ph.D. thesis, MIT. Published as *Infinite Syntax!*, Ablex, Norton, NJ, 1986.

David E. RUMELHART, James L. MCCLELLAND and THE PDP RESEARCH GROUP (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, MA.

Ivan SAG (1982). A semantic theory of 'NP movement' dependencies. In Pauline JACOBSON and Geoffrey K. PULLUM, (eds.), *The Nature of Syntactic Representation*, 427–465. Reidel, Dordrecht.

Geoffrey SAMPSON (1986). A stochastic approach to parsing. In *Proceedings of the 11th International Conference on Computational Linguistics*, 151–155. Bonn, Germany.

Christer SAMUELSSON (2000). A statistical theory of Dependency Syntax. In *Proceedings of the 18th International Conference on Computational Linguistics*, 684–690. Saarbrücken, Germany.

Remko SCHA (1990). Taaltheorie en taaltechnologie; competence en performance. In R. DE KORT and G.L.J. LEERDAM, (eds.), *Computertoepassingen in de Neerlandistiek*, 7–22. LVVN, Almere.

Remko SCHA, Rens BOD and Khalil SIMA'AN (1999). A memory-based model of syntactic analysis: Data Oriented Parsing. *Journal of Empirical and Theoretical Artificial Intelligence (JETAI)*, **11**(3).

Yves SCHABES (1992). Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 15th International Conference on Computational Linguistics*, 425–432. Nantes, France.

Yves SCHABES, Michal ROTH and Randy OSBORNE (1993). Parsing the Wall Street Journal with the Inside-Outside algorithm. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics*, 341–347. Utrecht, The Netherlands.

Yves SCHABES and Stuart SHIEBER (1994). An alternative conception of Tree-Adjoining derivation. *Computational Linguistics*, **20**(1):91–124.

Yves SCHABES and Richard C. WATERS (1995). Tree insertion grammar: a cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, **21**(4):479–513.

Yves SCHABES and Richard C. WATERS (1996). Stochastic lexicalized tree-insertion grammar. In Harry BUNT and Masaru TOMITA, (eds.), *Recent Advances in Parsing Technology*, 281–294. Kluwer Academic Press, London, England.

Helmut SCHMID (2002). A generative probability model for unification-based grammars. In *Proceedings of the 19th International Conference on Computational Linguistics*, 884–890. Taipei, Taiwan.

Gerold SCHNEIDER (2003a). Extracting and using trace-free functional dependencies from the Penn Treebank to reduce parsing complexity. In Joakim NIVRE and Erhard HINRICHS, (eds.), *Proceedings of Treebanks and Linguistic Theories (TLT)*, vol. 9 of *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, 153–164. Växjö University Press, Växjö, Sweden.

Gerold SCHNEIDER (2003b). Learning to disambiguate syntactic relations. *Linguistik online*, **17**(05/03):117–135.

Gerold SCHNEIDER (2003c). A low-complexity, broad-coverage probabilistic Dependency Parser for English. In *Proceedings of the Student Workshop of HLT-NAACL 2003*. Edmonton, Cananda.

Peter SGALL, Eva HAJIČOVÁ, Jarmila PANENOVÁ and J. L. MEY, (eds.) (1986). *The meaning of the Sentence in its Semantic and Pragmatic Aspects*. Academia, Prague, Czechoslovakia.

Claude SHANNON (1948). A mathematical theory of communication. *The Bell System Technical Journal*, **27**:379–423,623–656.

Beau SHEIL (1976). Observations on context-free parsing. In *Proceedings of the 6th International Conference on Computational Linguistics*.

Khalil SIMA'AN (1996). Computational complexity of probabilistic disambiguation by means of Tree Grammars. In *Proceedings of of the 16th International Conference on Computational Linguistics*, 1175–1180. Copenhagen, Denmark.

Khalil SIMA'AN (2003). Computational complexity of disambiguation under DOP1. In Rens BOD, Remko SCHA and Khalil SIMA'AN, (eds.), *Data-Oriented Parsing*, CSLI Publications. University of Chicago Press.

Daniel SLEATOR and Davy TEMPERLEY (1991). Parsing English with a link grammar. Tech. Rep. CMU-CS-91-196, Department of Computer Science, Carnegie Mellon University.

Bangalore SRINIVAS (1997). *Complexity of Lexical Descriptions and its Relevance to Partial Parsing*. Ph.D. thesis, University of Pennsylvania.

Bangalore SRINIVAS and Aravind JOSHI (1999). Supertagging: An approach to almost parsing. *Computational Linguistics*, **25**(2):237–265.

Mark STEEDMAN (1996). *Surface Structure and Interpretation*. The MIT Press, Cambridge, MA., London, UK.

Mark STEEDMAN (2000). *The Syntactic Process*. The MIT Press, Cambridge, MA.

Erik F. TJONG KIM SANG (2002). Introduction to the CoNLL–2002 shared task: Language-independent named entity recognition. In *Proceedings of the 6th Workshop on Natural Language Learning (CoNLL-2002)*, 155–158. Taipei, Taiwan.

Erik F. TJONG KIM SANG and Sabine BUCHHOLZ (2000). Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of 4th Conference on Computational Language Learning (CoNLL-2000) and the Second Learning Language in Logic Workshop (LLL– 2000)*, 127–132. Lisbon, Portugal.

Erik F. TJONG KIM SANG and Fien DE MEULDER (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the 7th Conference on Natural Language Learning (CoNLL-2003)*, 142–147. Edmonton, Canada.

Erik F. TJONG KIM SANG and Jorn VEENSTRA (1999). Representing text chunks. In *Proceedings of 9th Conference of the European Chapter of the Association for Computational Linguistics*, 173–179. Bergen, Norway.

Kristina TOUTANOVA, Christopher D. MANNING, Stuart M. SHIEBER, Dan FLICKINGER and Stephan OEPEN (2002). Parse disambiguation for a rich HPSG grammar. In *First Workshop on Treebanks and Linguistic Theories (TLT2002)*, 253–263. Sozopol, Bulgaria.

Dan TUFIŞ, Péter DIENES, Csaba ORAVECZ and Tamás VÁRADI (2000). Principled hidden tagset design for tiered tagging of Hungarian. In *Proceedings of the Second International Conference on Language Resources and Evaluation, LREC2000*, 1421–1426. Athens, Greece.

Hans USZKOREIT (2002). New chances for deep linguistic processing. In *Proceedings of the 19th International Conference on Computational Linguistics*, xiv–xxvii. Taipei, Taiwan.

Vladimir VAPNIK (1995). *The Nature of Statistical Learning Theory*. Springer, Berlin, Germany.

Vladimir VAPNIK (1998). *Statistical Learning Theory*. Wiley, Chichester, UK.

K. VIJAY-SHANKER (1987). *A Study of Tree-Adjoining Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

K. VIJAY-SHANKER and David J. WEIR (1994). The equivalence of four extensions of Context-Free Grammars. *Mathematical Systems Theory*, **27**:511–546.

Adriaan VAN WIJNGAARDEN, B. J. MAILLOUX, J. E. L. PECK, Cornelis H. A. KOSTER, Michel SINTZOFF, C. H. LINDSEY, Lambert G. L. T. MEERTENS and R. G. FISKER (1975). Revised report on the algorithmic language ALGOL 68. *Acta Informatica*, **5**:1–235.

XTAG RESEARCH GROUP (2001). A lexicalized tree adjoining grammar for english. Tech. Rep. IRCS-01-03, IRCS, University of Pennsylvania.

Daniel ZEMAN (1998). A statistical approach to parsing of Czech. *Prague Bulletin of Mathematical Linguistics*, 29–37.

Daniel ZEMAN (2002). Can subcategorization help a statistical dependency parser? In *Proceedings of the 19th International Conference on Computational Linguistics*, 1156–1162. Taipei, Taiwan.