

Parsing with a Compacted Treebank Grammar

Alexander Krotov



Doctor of Philosophy

Department of Computer Science, University of Sheffield

01 July 2001

Abstract

Large corpora annotated with tree structure such as the Penn Treebank provide a basis for automatic grammar extraction. Parsing with *treebank grammars* extracted this way yields good parsing accuracy, but the grammar size is very large (and thus the time and memory requirements are high).

In this work I attempt to reduce the grammar size, or *compact* the grammar without losing its coverage or parsing accuracy. Treebank grammar compaction eliminates some longer, ‘redundant’ rules replacing them with an equivalent combination of shorter rules. ‘Rule parsing’ is used to determine which rules are redundant.

In its simplest, ‘naive’ form, grammar compaction eliminates more than 90% of rules, but the parsing accuracy is poor. When rule probabilities are taken into account only 9% of all rules can be eliminated but the parsing accuracy is not affected significantly. The compaction rate can be improved even further, up to 31% using ranking. In this case recall improves, but precision is lower.

Combined with thresholding (removing infrequent rules), the compaction rate can be improved further, to 62% (or to 71% with a gain in recall and loss in precision). Thresholding achieves higher reduction rates, but at the cost of grammar coverage and lower parsing accuracy.

Acknowledgements

First, I would like to thank my supervisor, Yorick Wilks. He introduced me to statistical natural language processing when I first came to Sheffield, and made numerous valuable comments throughout this work, especially towards the end.

The members of my committee, Matt Fairtlough, Robert Gaizauskas and Mark Hepple have closely followed my progress and their feedback was important for completing this dissertation. Robert Gaizauskas suggested grammar extraction from the Penn Treebank and later carried out a detailed investigation of the extracted grammar. I used his programs to produce the initial treebank grammar for the compaction experiments. Mark Hepple was actively involved in parsing and evaluation. I used his parser to produce the experiment results reported in this work.

Thanks to Andreas Stolcke and Satoshi Sekine I was able to experiment with grammar compaction during the early stages of this work. They developed statistical parsers which I used before an in-house parser was implemented.

Thanks to David Levy and Yorick Wilks I was involved in CONVERSE project. Work on CONVERSE allowed me to use my knowledge in parsing in a practical application. Bobby Batacharia and Roberta Catizone were also on the CONVERSE team.

Many people in Sheffield were involved in discussions that contributed to this dissertation. In particular, I would like to thank Matthew Hurst, Kyo Kageura, Chunyu Kit and Paul McKeivitt.

Last, but not least, I thank my family and close friends for their valuable support.

Contents

1	Introduction	1
2	Context-Free Parsing	8
2.1	Part-of-speech Tagging	9
2.2	Context-Free Grammars	11
2.3	Parsing with Context-Free Grammars	11
2.3.1	Chart parsing	12
2.3.2	Earley parsing	14
2.3.3	LR(k) and Tomita parsing	17
2.4	Parsing natural language	18
3	Elements of Statistical NLP	21
3.1	Corpora	21
3.2	Hidden Markov Models	23
3.2.1	Training HMMs	24
3.3	Probabilistic Grammars	26
3.4	Training a grammar with the Inside-Outside Algorithm	27
4	Contemporary Research in Statistical Parsing	30
4.1	Probabilistic CFG Parsing	31
4.1.1	Stolcke – Probabilistic Earley Parsing	31
4.1.2	Sekine – Parsing with Two Nonterminals	33

4.1.3	Hepple – Memory-efficient Chart Parsing	34
4.2	Treebank Grammars	36
4.2.1	Sharman – P-ID/LP grammars	36
4.2.2	Gaizauskas – Treebank Grammar Extraction	37
4.2.3	Charniak – Parsing with a Treebank Grammar	37
4.2.4	Shirai – Japanese Treebank Grammar	38
4.3	Data-Oriented Parsing (DOP)	39
4.4	Johnson – PCFG models of linguistic tree representations	41
5	Extracting the Penn Treebank Grammar	42
5.1	Rule Extraction – Early Experiments	43
5.1.1	First version of the Penn Treebank	43
5.1.2	Matching Algorithm	44
5.1.3	Results	45
5.1.4	Problems with the Penn Treebank	45
5.1.5	Conclusions	46
5.2	Second Release of the Penn Treebank	47
5.2.1	Improvements in the Second Release	47
5.2.2	Modified Extraction Algorithm	49
5.2.3	Extracting the Grammar	51
6	Compacting the Penn Treebank Grammar	54
6.1	Motivation	55
6.2	Growth of the Rule Set	56
6.3	Rule Growth and Partial Bracketting	57
6.4	Grammar Compaction	59
6.5	Proof of Order Independence	61
6.6	Experiments	63
6.7	Retaining Linguistically Valid Rules	63

6.8	Conclusions	66
7	Parsing and Evaluation	68
7.1	Parsing System	69
7.2	Parser Evaluation	71
7.3	Test Data	75
8	Experiments	76
8.1	Summary of Earlier Results	77
8.2	Grammar and Sample Generation	79
8.2.1	Overview	79
8.2.2	Storing Rule Parse Charts	79
8.2.3	Probabilistic Compaction	81
8.2.4	Test Samples	82
8.3	Results	83
8.3.1	Thresholding	83
8.3.2	Probabilistic Compaction	83
8.3.3	Effect of removing unknown linguistic constituents	85
8.3.4	Effect of selecting the top sentence constituent	86
8.4	Summary	86
9	Conclusions	89
A	Tags used in the Penn Treebank	92
B	Programs Used for Evaluation the Grammar Compaction Method	97
	References	97

List of Tables

2.1	Sample LR table	18
2.2	Illustration of an LR derivation	19
6.1	Results of the Grammar Compaction Experiment	62
6.2	Preliminary results on parsing with the linguistically compacted grammar	66
7.1	Parser Evaluation Measures	74
8.1	Effect of simple thresholding	84
8.2	Effect of Varying Thresholds	84
8.3	Evaluating Probabilistic Compaction	85
8.4	Effect of removing X and FRAG	86
8.5	Evaluating Probabilistic Compaction using the Best Parse	87
A.1	Syntactic Tags	92
A.2	Part-of-Speech Tags	93
A.3	Functional Tags	95

List of Figures

2.1	A small grammar used as a sample	12
2.2	A sample parse tree	12
2.3	Illustration of chart parsing	14
2.4	Illustration of Earley parsing	16
3.1	A Markov Model	24
3.2	A Hidden Markov Model	25
3.3	Illustration of inside and outside probabilities	29
5.1	Sample text from the Penn Treebank I	43
5.2	Sample parse tree	44
5.3	A sample of merged text from the Penn Treebank	49
5.4	Tag simplification illustrated	52
5.5	Rate of Growth of the Rule Size	53
6.1	Rule Set Growth for Penn Treebank I	56
6.2	Rule Set Growth for Penn Treebank II	57
6.3	Compacted Grammar Size	64
8.1	Grammar size growth versus compacted grammar size	78
8.2	Parsing accuracy using probabilistically compacted grammars (preliminary results)	78
8.3	Evaluation of the grammar compaction method	80

8.4	Ranking rules by compaction ratio	82
-----	---	----

Chapter 1

Introduction

Natural Language Processing (NLP) traces its origins back to 1956, when the Dartmouth Conference on Artificial Intelligence was held as well as the Second Conference on Mechanical Translation. Computers view text merely as a sequence of characters, however natural language text contains much more information (e.g., the meanings of individual words, phrase structure elements, and sentences). One of the primary tasks of NLP is to augment natural language text (viewed as a string of characters) with a representation that a computer can understand. For example, given the string `the cat sat on the mat`, the computer will need to know that the first three letters are a definite article `the`, the first seven letters are the noun phrase `the cat`. `Sat` is the main verb of the sentence, and it is connected with its subject `the cat`, object `the mat` and a preposition `on`. `The cat` is a kind of an `animal`. The computer will also need to know that `sit` prefers animate subjects (i.e., `the mat` does not normally `sit` on `the cat`). This list can be extended much further, since the meanings of sentences and their parts connect with the parts of other sentences. Also, words can have different meanings. `Cat`'s other noun meanings include `cat o' nine tails`. `Cat` can be a verb meaning `beat with the cat o' nine tails`. However, this meaning occurs infrequently.

At the lowest level, NLP has to deal with the language *syntax*, i.e., individual words and phrase elements. This is the most explored area of NLP and the primary focus of this work.

Early natural language systems such as BASEBALL (Green et al., 1963), SIR (Raphael, 1968), STUDENT (Bobrow, 1968) and PARRY (Colby, 1973) matched input against a set of patterns or regular expressions. Patterns were later replaced with grammars, first context-free and then other formalisms were designed to take into account more linguistic information (e.g., ATNs, DCG, FUG, HPSG, (Woods, 1970; Pereira and Warren, 1980; Kay, 1985; Gazdar et al., 1982)). Parsing a natural language sentence involves designing a grammar and using a parsing algorithm to derive the parse tree for this sentence. Most parsing algorithms are based on chart parsing (Kay, 1963). Other areas of NLP include *semantics*, *discourse interpretation*, *intentions* and *language generation*. Grosz, Sparck-Jones, and Webber (1986) contains a comprehensive collection of papers on NLP.

Semantics deals with the interpretation of syntactic structures: i.e. **the cat** is not just a noun but also an **animal**. An **animal** is a kind of a **living being**. In the example above, the meaning of the **the cat** merges with the meaning of the main verb **sat** and the location **on the mat** giving rise to the meaning of the complete sentence. The meaning of a sentence or its elements also depends on the *domain*, *context* and *task* of the semantic system. *Domain* is the part of the world knowledge that the semantic system is aware of. In a financial context, **tumble** is likely to mean a **sharp drop** (i.e., **the stock prices tumbled**), whereas used in a home appliance catalogue **tumble** associates with a **tumble dryer**. *Context* is the place of a given sentence among other sentences, the place and time this sentence is being said, and the beliefs and intentions of all participants of the discourse. The *task* of a semantic system is what it is used for, e.g., a conversational companion, the language component of a multimodal virtual reality system, a question-answering system, etc.

People speak in sequences of sentences, and to derive the meaning of an individual sentence one may need to refer to its *context*. *Discourse interpretation* deals with augmenting the meaning of an individual sentence using the information derived from the entire sequence, and possibly external world knowledge (such as time and location). In particular, meanings of most pronouns can be resolved only if the previous sentences are taken into

account. For example, if `the neighbour's cat just walked in my kitchen` is followed by `he must have smelled fish on the table`, `he` in the second sentence refers to `the neighbour's cat` in the first sentence. The meaning of `he` cannot be resolved using the second sentence alone. The first sentence is needed and external world knowledge is also helpful (the neighbour invited me over a couple of days ago, and she told me her cat is male).

People do not always mean exactly what they say. For example, a question `do you know what time it is?` does not require a yes or no answer as syntax may suggest. Instead, most people would expect either the current time or a negative response. Answering `yes` is grammatically correct, but does not make much sense. Taking into account the beliefs and desires of all participants of a discourse helps to understand how people's *intentions* can be translated into natural language. This area of NLP is closely related to Artificial Intelligence (AI) and philosophy.

Language generation systems take semantic representation as an input and produce the corresponding natural language string. This process is not as straightforward as it may seem (i.e., substituting character strings into existing grammatical patterns), since people do not speak using fully grammatical sentences. Instead, they use idioms and sometimes break grammar rules (e.g., `we don't need no education`). Fully grammatical output may sound too computerised, and inserting a certain amount of “fluff” (which may not carry any specific meaning) makes text more fluid and human-like. Certain expressions such as `how are you?` have well-established meanings and do not require syntactic decomposition. `How are you?` is a greeting and not a question.

In this work I investigate applying statistical methods to syntactic processing. Statistical methods have revived in recent NLP for a number of reasons. One is the failure of purely rule-based approaches to provide robust natural language recognition. Also, a tremendous increase in computing power during the last decade has allowed computationally complex statistical methods in speech and language to be implemented to work in real time. The basis of statistical work, *corpora*, i.e., large amounts of computerised text (or speech), raw

and annotated, were available in English back in the early 1960s, i.e. Brown corpus (Francis and Kučera, 1982), and more recently (Marcus, Santorini, and Marcinkiewicz, 1993; Marcus et al., 1994). Corpora are also available in other languages such as Japanese (EDR, 1994) and German (Skut et al., 1997).

The main focus of this dissertation is *treebank grammars*, grammars that can be extracted directly from a parsed corpus. Treebank grammars are easy to obtain and promising parsing accuracy based on them have been shown in Charniak (1996). However, in Krotov, Gaizauskas, and Wilks (1994) we reported a significant growth of the number of rules in the extracted grammar and no tendency for this growth to slow down or approach a limit. A similar pattern of growth was observed in a more thorough investigation of the PTB grammar carried out in Gaizauskas (1995).

Parsing with a very large number of rules demands a considerable amount of memory and time. The grammar we extracted from the Wall Street Journal portion of the Penn Treebank consists of more than 17,000 rules. A traditional parser using context-free rules may not be able to handle such a large grammar, and only recently efficient algorithms have been developed for parsing with large grammars (Moore, 2000) or Mark Hepple’s parser described in this thesis. One may also want to reduce the size of the extracted grammar to make it more useful for applications. An easy way to do this is thresholding, i.e., removing less frequently occurring rules. Thresholding achieves a considerable reduction in size without significant loss in parsing accuracy. However, thresholding (especially with a high cut-off) does not retain grammar coverage, i.e., thresholded grammar will not be able to parse all sentences that the original grammar can parse.

Another option is to *compact* the rule set. There are many long rules in the Penn Treebank, and it is quite possible that many of these long rules can be replaced with trees consisting of shorter rules from the same rule set. Such ‘redundant’ rules can therefore be safely removed from the grammar. This is the idea behind a compaction algorithm, which removes any rules that can be ‘parsed’ with other rules. The process of ‘parsing rules’ (defined later) identifies ‘redundant’ rules and is similar to context-free parsing of strings.

A large portion of this work is devoted to investigating a compaction algorithm. The first results from compacting the Penn Treebank are very promising: the 17,000+ rule set compacts to 1,667 rules, less than 10% of its original size. Also, the size of the compacted grammar seems to be approaching some limit, though more parsed data is necessary to confirm this hypothesis.

Parsing with the compacted grammar is much faster than parsing with its uncompact version. However, the straightforward compaction method does not take into account rule frequencies. Therefore, parsing with such a small grammar produces a different kind of parse: the parsed structure of redundant rules does not appear and is replaced with the structure consisting of a large number of short and infrequent rules. Consequently, parsing accuracy of the fully compacted grammar is poor.

To alleviate this problem, I suggest using rule frequencies in the compaction process. This modified, linguistic compaction algorithm does not affect parsing accuracy or grammar coverage, but the compaction rate is much lower (9% as opposed to over 90% with the full, ‘naive’ compaction). In a further modification to the compaction algorithm, *ranked* linguistic compaction produces a better compaction rate (31%), and a better parsing recall (but lower precision). This is the best compaction rate I can achieve at the moment. It may be that many long and infrequent rules encode useful linguistic information, and thus cannot be removed.

Combined with simple thresholding, this compaction rate rises in the end to 62% (or to 71% with higher recall, but lower precision), suggesting the likely utility for a compaction algorithm. For a partially word-order free language such as Korean, an idea similar to compaction was explored in Lee et al. (1997). They modify their grammar formalism so that it favours shorter rules and deeper tree analyses. A significant decrease in grammar size and parsing time is reported with little or no loss in parsing accuracy. Lee’s results indicate that treebank grammar compaction may work even better for a partially word-order free language.

Grammar compaction can be easily extended to the Data-Oriented Parsing (DOP) ap-

proach (Bod, 1996; Bod and Scha, 1996; Bod, Bonnema, and Scha, 1996) if one were to parse the subtrees instead of rules. At present, DOP achieves high parsing accuracy, but at the cost of tremendous computational complexity. Good results are reported, but on small samples, and sentences with average length of less than 5 words (Bod, Bonnema, and Scha, 1996). If a considerable reduction were achieved in compacting DOP grammars, this would certainly make DOP parsing more efficient. It would be interesting to see whether this is the case.

Parsing with the compacted grammar delivers comparable if not better accuracy compared to other non-lexicalised approaches using probabilistic context-free grammars (Sekine and Grishman, 1995; Shirai, Tokunaga, and Tanaka, 1995; Charniak, 1996). However, the best parsing results at present (Collins, 1997; Ratnaparkhi, 1997; Charniak, 2000) use lexical data and non-context free statistical models. Nonetheless, context-free grammars are widely used in natural language systems. For example, a small (and slightly modified) version of the Penn Treebank grammar was used in LaSIE, an information extraction (IE) system (Gaizauskas et al., 1995). Also, the Microsoft speech recognition engine accepts context-free grammar as a model for constraining recognition. In particular, probabilistic context-free grammar (PCFG) is a very simple model of natural language and recent research (Johnson, 1998b) shows that ‘it seems reasonable to first attempt to better understand the properties of PCFGs themselves.’ Compaction of treebank grammars is an attempt to reduce grammar size by removing redundancy contained within the grammar itself. Combined with simple thresholding, a significant reduction (and thus better parsing accuracy) can be achieved.

This work is organised as follows. Chapter 2 reviews syntactic parsing: part of speech tagging, context-free grammars and a number of parsing algorithms. Chapter 3 discusses corpora and parsing-related elements of statistical NLP: Hidden Markov Models and probabilistic grammars. It is followed by Chapter 4 which discusses contemporary research in probabilistic parsing.

The rest of the dissertation describes our investigations of grammars extracted from the

Penn Treebank (PTB). The PTB is the largest publicly-available, manually parsed corpus of English. Chapter 5 describes our early experiments in grammar extraction from the first release of the PTB. A significant growth in the number of extracted rules was observed, therefore suggesting a need to compact the extracted rule set. Chapter 6 describes the compaction algorithm, and how it performs on the Penn Treebank grammar. Compacting the PTB rule set raises the question of retaining linguistically valid rules as well as evaluating the resulting grammar with a parser. Retaining linguistically valid rules is discussed at the end of Chapter 6, along with some preliminary evaluation results.

The resulting grammar is evaluated by parsing a sample of text. Parsing and evaluation is discussed in Chapter 7. Chapter 8 summarises the experimental results presented earlier, describes the set of programs used for the final set of experiments and presents the complete evaluation results.

Finally, Chapter 9 presents some final remarks and suggests directions for further research.

Chapter 2

Context-Free Parsing

Early parsers (King, 1983) were built upon Chomsky's transformational model and attempted to take into account semantic information. At present, there is a wide variety of parsing schemata and algorithms. (Lin, 1994) presents an efficient, broad coverage parser based on a government-binding (GB) theory (Chomsky, 1981; Haegeman, 1991), while (Harkema, 2000) developed a theoretical framework for parsing with Minimalist Grammars (Chomsky, 1995), inspired by the GB theory.

(Maxwell and Kaplan, 1993) explored interaction of phrasal and functional constraints in complex grammar formalisms such as Lexical-Functional Grammar (LFG) (Kaplan and Bresnan, 1982), Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1982), Functional Unification Grammar (FUG) (Kay, 1985) and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987). (Oepen and Carroll, 2000) report practical results in using local ambiguity packing for parsing with HPSG.

XTAG project at the University of Pennsylvania (Doran et al., 1997; Sarkar, 2000), see also www.cis.upenn.edu/xtag is a wide-scale effort involving development of not only a large grammar (and a parser), but also the tools necessary to develop this grammar such as morphological analyser, POS tagger, syntactic and tree databases, and a sophisticated visual X-interface.

Vijay-Shanker, Weir, and Rambow (1995) defined a new D-Tree Grammar formalism

based on the work on Tree-Adjoining Grammars (TAGs) and designed a polynomial time Earley-style parser for D-Tree Grammars. Eisner and Satta (1999) designed two algorithms which reduced parsing complexity of bilexical context-free grammars from $O(n^5)$ to $O(n^4)$. (Johnson, 1998a) describes how a context-free grammar can be approximated with a finite-state machine using a series of grammar transforms.

Statistical syntactic parsers (e.g., (Collins, 1997; Charniak, 2000)) use simpler models (i.e., head-driven parsing, maximum entropy, Markov models) and carry out syntactic analysis, also called *shallow parsing*. Ambiguity in such parsers is resolved using lexical information and statistical methods (i.e., by observing patterns in the text) rather than a set of complex rules.

In this chapter I discuss parsing with context-free grammars (CFGs). CFG is the “least common denominator” of many complex grammar formalisms (such as GPSG, FUG and HPSG), can be separated into the set of context-free phrasal constraints and attribute-value phrasal constraints, as (Maxwell and Kaplan, 1993) note.

Section 2.1 talks about part-of-speech tagging, a stage preceding parsing. Section 2.2 discusses context-free grammars and their modifications. Section 2.3 presents chart parsing, Earley parsing and algorithms that do not use a chart such as the LR(k) and Tomita algorithms. Finally, Section 2.4 discusses some issues related to parsing natural language with context-free grammars, outlining some deficiencies of purely rule-based systems and suggesting a need for richer language models.

2.1 Part-of-speech Tagging

Part-of-speech (POS) tagging deals with assigning a correct part of speech tag (e.g., noun, adjective) to each word in a sentence. POS tagging is an independent task, but is often carried out immediately before parsing. High accuracy (95%+) has been achieved using a number of methods. Choice of an appropriate set of tags is influenced by the available corpora. The Penn Treebank tagset (Marcus, Santorini, and Marcinkiewicz, 1993) is used in most of the recent research in English (see Table A.2).

Early taggers, e.g., Cherry (1978), used a dictionary and a set of rules to assign tags to words. A rule-based tagger starts with assigning tags to the ‘known’ words and then applies a number of rules in turn in order to assign tags to the remaining words. Cherry (1978) reports 95% accuracy using his tagger.

Hidden Markov Models (HMMs, see Section 3.2) have been used very successfully for POS tagging (Jelinek, 1985; Church, 1988; Kupiec, 1992). HMM taggers are trained on a large tagged corpus and predict each tag based on the word being tagged and its $n - 1$ predecessors and their tags (n -gram model). Trigrams are normally used together with smoothing techniques which help to account for shorter dependencies and less frequently occurring n -grams. Once the training data is collected, the Viterbi algorithm is used to tag words. HMM taggers achieve 96-97% accuracy.

Transformation-based tagging (Brill, 1995) takes the rule-based approach further by learning tagging rules from a tagged corpus. Transformation-based learning captures more complex dependencies between words and tags than the n -gram model. Therefore, it requires considerably fewer parameters for training while achieving similar accuracy.

Recent taggers based on hand-coded rules, e.g. English Constraining Grammar Parser (EngCG) developed by Voutilainen (1995) capture even more complex dependencies between words and achieve higher accuracy, above 99%. However, as Samuelsson and Voutilainen (1997) note, these results have been seriously questioned for example by Church (1992), who argues that even in manual tagging linguists disagree on the correct analysis of about 3% of the words. Thus, reporting accuracies above the 97% upper bound does not make sense.

Maximum entropy (Ratnaparkhi, 1996) and memory-based models (Daelemans et al., 1996; van Halteren, Zavrel, and Daelemans, 1998) have also been used for POS tagging. They achieve even further improvement in tagging accuracy compared to other statistical taggers, up to about 98%.

Finally, recent statistical parsers (Magerman, 1995; Collins, 1996; Collins, 1997) carry out tagging and parsing tasks simultaneously.

2.2 Context-Free Grammars

Chomsky's work on formal languages (Chomsky, 1963) has provided the basis for most of the early work in parsing natural language. Augmented Transition Networks (ATNs) (Woods, 1970) extended the generative power of context-free grammars and combined it with the simplicity of a finite-state automaton. In the 80s the context-free grammar framework was extended further taking into account more linguistic information. Formalisms such as Functional Unification Grammar (FUG) (Kay, 1985), Definite Clause Grammar (DCG) (Pereira and Warren, 1980) and Generalized Phrase Structure Grammar (GPSG) (Gazdar et al., 1982) were developed.

In this work I consider only the simplest form of context-free grammar, as defined in formal language theory, i.e., a set of rewrite rules of the form $A \rightarrow B_1, B_2, \dots, B_n$ where A is a nonterminal and B_i are either nonterminals or terminals. Nonterminals are syntactic tags (such as noun phrase, verb phrase) and terminals are part of speech tags or words (for lexicalised approaches). This rule indicates that nonterminal A can be replaced with the string B_1, \dots, B_n . Each B_i , if it is a nonterminal and a left-hand side (LHS) of a different rule can be expanded in turn, thus generating parse derivations. One nonterminal is labelled as a top-level nonterminal, usually S , indicating that this nonterminal should be at the top of all valid string derivations.

To illustrate the parsing algorithms presented in the following section, I will use the short grammar in Figure 2.1. The tagset is consistent with the one used in the Penn Treebank (see Table A.1 and A.2). Such grammar can generate only one string: *DT NN VBD IN DT NN* (e.g., “the cat sat on the mat”), but it is sufficient to explain parsing algorithms. Figure 2.2 shows the graphic representation of this string.

2.3 Parsing with Context-Free Grammars

Given a grammar, a *parsing* algorithm is needed to create the tree (or several trees) that can be used to analyse a given string. Chart parsing and its varieties are commonly used for

$$\begin{aligned}
 S &\rightarrow NP VP & (1) \\
 NP &\rightarrow DT NN & (2) \\
 VP &\rightarrow VBD PP & (3) \\
 PP &\rightarrow IN NP & (4)
 \end{aligned}$$

Figure 2.1: A small grammar used as a sample

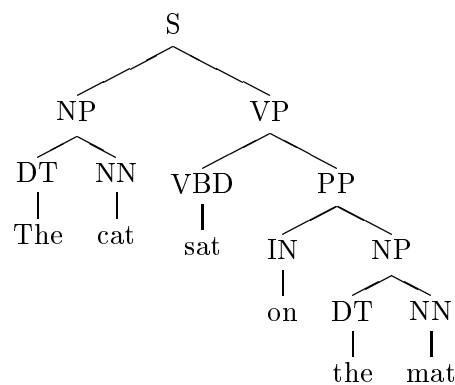


Figure 2.2: A sample parse tree

context-free parsing. In this section I present some basics of chart parsing, Earley parsing, as a kind of chart parser and parsing algorithms that do not use charts such as the LR(k) and Tomita algorithm. LR(k) parsers are restricted to a certain type of grammar, but offer a significant reduction in parsing time and memory usage. LR(k) algorithms are commonly used in compilers (e.g., see Johnson (1975)). Tomita's algorithm is a modification of LR(k) algorithm which can use any kind of context-free grammar.

2.3.1 Chart parsing

Chart parsing was originally proposed in Kay (1963), Younger (1967), and Kasami (1967). Kay used charts for parsing natural language. Younger and Kasami proved the $O(n^3)$ time complexity of the parsing algorithm, where n is the number of symbols in the input string. Sikkel (1997) presents an extensive overview of parsing algorithms and schemata presently available, including many varieties of chart parsing. In this section I will describe one of the simpler, but more commonly used form of chart parsing, similar to the one presented

in (Gazdar and Mellish, 1989).

A parse chart consists of *edges* of the form $A : (i, j)$. This edge indicates that nonterminal A covers the substring (i, j) of the input, i.e., the substring spanning from the i -th to the j -th ‘gap’ between the symbols in the original input string. An i -th ‘gap’ occurs immediately after the i -th input symbol, and the 0-th ‘gap’ is just before the first symbol (see Figure 2.3). The part-of-speech tagger ‘seeds’ the chart with lexical edges, one for each word. A parsing algorithm then traverses the chart finding ways to combine edges. Two or more edges can be combined if they cover neighboring substrings and there is a grammar rule which has all nonterminals involved on the right-hand side in the right sequence. In other words, edges $B_k : (i_k, j_k), k = 1 .. n$ can be combined if $j_k = i_{k+1}$ for all $k = 1 .. n - 1$ and there is a rule $A \rightarrow B_1, \dots, B_n$ in the grammar. In this case, a new edge $A : (i_1, j_n)$ can be added to the chart. This process is repeated until an edge which covers the entire sentence with a top-level nonterminal is found ($S : (1, n)$), or no more edges can be added to the chart. S is the top-level nonterminal, and n is the number of characters in the input. If $S : (1, n)$ is found, all parses of the input string can be found by traversing the chart. If not, the input string cannot be parsed with the given grammar.

To make edge combination easier, many parsers use *incomplete* edges of the form:

$$(i, j) : A \rightarrow B_1, \dots, B_m \cdot C_1, \dots, C_n$$

where (i, j) is the substring covered by the edge, and

$$A \rightarrow B_1, \dots, B_m, C_1, \dots, C_n$$

is a rule in the grammar. The dot indicates that all B_i are already covered by the edge and all C_j are yet to be completed. A *complete edge*

$$(i, j) : A \rightarrow B_1, \dots, B_m \cdot$$

is equivalent to the edge $A : (i, j)$ as described in the previous paragraph. The combination rule for incomplete edges is now as follows. A complete edge

$$(i, j) : X \rightarrow B_1, \dots, B_m \cdot$$

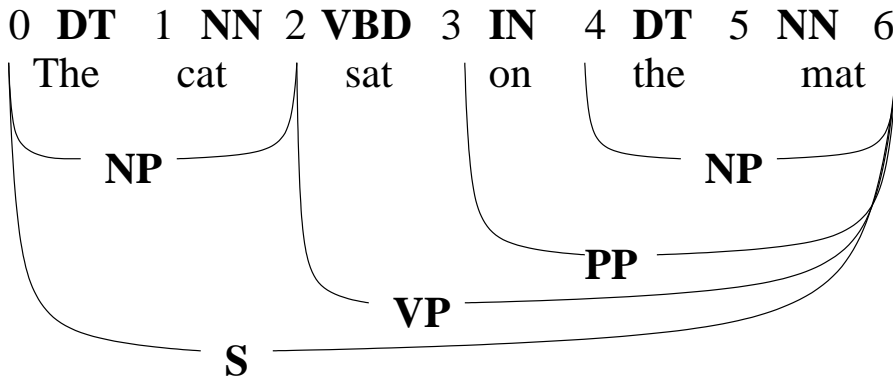


Figure 2.3: Illustration of chart parsing

can be combined with an incomplete edge

$$(j, k) : A \rightarrow C_1, \dots, C_n \cdot X \ D_1, \dots, D_p$$

producing an edge

$$(i, k) : A \rightarrow C_1, \dots, C_n \ X \cdot \ D_1, \dots, D_p$$

Chart parsing algorithms differ in the way they generate edges. Bottom-up algorithms such as the CKY algorithm start with lexical edges of length 1 representing individual word or part of speech tokens and combine shorter edges into longer ones. If an edge covering the entire input string with the top-level nonterminal exists, the string is parsable. Top-down algorithms, such as Earley algorithm start with an incomplete edge

$$(0, 0) : \rightarrow \cdot \ S$$

attempting to complete S , the top-level nonterminal. For a description of some parsing algorithms, bottom-up and top-down, see Gazdar and Mellish (1989).

2.3.2 Earley parsing

Earley's algorithm (Earley, 1968; Earley, 1970) is a top-down chart parsing algorithm that can handle unary and null-production rules, i.e. rules of the form $A \rightarrow B$ and $A \rightarrow \epsilon$ where B is a nonterminal and ϵ is an empty string. Such rules usually need special handling because

of possible infinite loops. Stolcke (1995) modified the Earley algorithm for use in natural language parsing. The algorithm proceeds left-to-right, generating a set of edges for each element of the input string x_1, \dots, x_n . The parser starts with the initial edge $(0, 0) : \rightarrow \cdot S$ and then for each new input token x_j repeats three steps in turn: prediction, scanning and completion. When this is complete, the parser looks for the edge $(0, n) : \rightarrow S \cdot$, and if it is found, the sentence is parsable. The parse tree can then be regenerated from the chart.

Prediction generates new states for potential expansion. For each state

$$(i, j) : A \rightarrow B_1, \dots, B_n \cdot C \ D_1, \dots, D_p$$

where C is a nonterminal and $C \rightarrow E_1, \dots, E_q$ is a rule in the grammar prediction adds a state

$$(j, j) : C \rightarrow \cdot E_1, \dots, E_q$$

Scanning moves the dot in the incomplete edges if the next character matches the current input symbol. For each state

$$(i, j) : A \rightarrow B_1, \dots, B_n, \cdot c \ D_1, \dots, D_p$$

where c matches the current input x_j scanning creates a state

$$(i, j + 1) : A \rightarrow B_1, \dots, B_n c \cdot \ D_1, \dots, D_p$$

Completion combines complete edges with the incomplete ones. For each pair of edges

$$(i, j) : A \rightarrow B_1, \dots, B_n \cdot$$

and

$$(k, i) : C \rightarrow D_1, \dots, D_p, \cdot A \ E_1, \dots, E_q$$

completion adds an edge

$$(k, j) : C \rightarrow D_1, \dots, D_p A \cdot \ E_1, \dots, E_q$$

Figure 2.4 illustrates Earley parsing with the grammar from Figure 2.3.

Initial state $(0, 0) : \rightarrow . S$ **Token 1**

Predicted

 $(0, 0) : S \rightarrow . NP VP$ $(0, 0) : NP \rightarrow . DT NN$

Scanned

 $(0, 1) : NP \rightarrow DT . NN$ **Token 2** $(0, 2) : NP \rightarrow DT NN .$

Completed

 $(0, 2) : S \rightarrow NP . VP$ **Token 3**

Predicted

 $(2, 2) : VP \rightarrow . VBD PP$

Scanned

 $(2, 3) : VP \rightarrow VBD . PP$ **Token 4**

Predicted

 $(3, 3) : PP \rightarrow . IN NP$

Scanned

 $(3, 4) : PP \rightarrow IN . NP$ **Token 5**

Predicted

 $(4, 4) : NP \rightarrow . DT NN$

Scanned

 $(4, 5) : NP \rightarrow DT . NN$ **Token 6** $(4, 6) : NP \rightarrow DT NN .$

Completed

 $(3, 6) : PP \rightarrow IN NP .$ $(2, 6) : VP \rightarrow VBD PP .$ $(0, 6) : S \rightarrow NP VP .$ $(0, 6) : \rightarrow S .$

Figure 2.4: Illustration of Earley parsing

2.3.3 LR(k) and Tomita parsing

Instead of using a chart to track all possible derivations, LR parsers proceed left-to-right keeping the string of all processed symbols on the stack. When a new symbol is encountered, the parser decides whether it is acceptable, and if it is, whether to *shift* this symbol onto the stack or *reduce* some elements on the stack using a grammar rule and replace elements with the left-hand side of the rule. While doing this, the parser keeps track of its current *state*.

For a certain class of grammars, LR(k) grammars, the shift/reduce decision can be made based only on the current state, the current input symbol, and k following symbols in the input string (k is often 1). Standard algorithms exist to determine whether a given grammar is LR(k) and to create *action* and *goto* tables (see, for example, (Aho and Ullman, 1972)). Given these tables, the parser can process a string in linear time as opposed to n^3 for chart-based algorithms. LR(k) algorithm is often used in compilers due to its efficiency. An automatic compiler generator is described in Johnson (1975). YACC creates action and goto tables for a given grammar, reports any shift/reduce conflicts and provides a standard parser which uses the generated tables. The action and goto tables for the sample grammar used in this chapter are given in Table 2.1, and a sample LR derivation is in Table 2.2. Briscoe and Carroll (1993) used a version of the LR(k) algorithm for parsing natural language.

The class of LR(k) grammars is limited; however, such grammars can model computer languages well, but cannot describe many phenomena in natural language, such as the ambiguity of prepositional phrase attachment. For example, in the sentence “I saw a man with a telescope”, “with a telescope” can attach to either “a man” or to “I saw a man”. A grammar that can handle this ambiguity is not LR(k), the ambiguity causes shift/reduce conflict. Tomita (1986) modified LR algorithm so that it can use any context-free grammar. The action table for Tomita algorithm contains all shift/reduce conflicts and, when such a conflict is encountered during parsing, the parsing process splits into two threads, each of which is followed independently. See Tomita (1984) for an example of a parse. The

worst case time complexity can be potentially exponential (Kipps, 1989; Johnson, 1989). However, Kipps (1989) presents a modification of Tomita algorithm requiring only $O(n^3)$ time. He also discusses two subclasses of context-free grammars that can be recognised in $O(n^2)$ and $O(n)$ time (using this algorithm). Shirai and others (1995; 1997) used Tomita's algorithm for parsing Japanese, and Briscoe and Carroll (1993) used a variant of Tomita's algorithm for parsing English.

Action Table						Goto Table			
State	DT	NN	VB	IN	\$	S	NP	VP	PP
0	s1						3	11	
1		s2							
2			r2					10	
3			s4						
4				s5					9
5	s6						8		
6		s7							
7					r2				
8					r4				
9					r3				
10					r1				
11					acc				

Table 2.1: Sample LR table

2.4 Parsing natural language

Martin, Church, and Patil (1981) present an extensive overview of parsing natural language with context-free grammars. *Ambiguity* is reported to be one of the main problems. In other words, most natural language sentences have many possible derivations and it is not always possible to choose the correct derivation using the grammar alone. Extra-grammatical knowledge, such as semantic information, is necessary to make this choice. The number of possible derivations may grow exponentially with the length of the input.

One source of ambiguity is prepositional phrase attachment. For example, (Martin,

Stack	Next word	Next action
0	The(DT)	shift1
0 DT 1	cat(NN)	shift2
0 DT 1 NN 2	sat(VBD)	reduce2
0 NP 3	sat(VBD)	shift4
0 NP 3 VBD 4	on(IN)	shift5
0 NP 3 VBD 4 IN 5	the(DT)	shift6
0 NP 3 VBD 4 IN 5 DT 6	mat(NN)	shift7
0 NP 3 VBD 4 IN 5 DT 6 NN 7	\$	reduce2
0 NP 3 VBD 4 IN 5 NP 8	\$	reduce4
0 NP 3 VBD 4 PP 9	\$	reduce3
0 NP 3 VP 10	\$	reduce1
0 S 11	\$	accept

Table 2.2: Illustration of an LR derivation

Church, and Patil, 1981) reports 455 parses for the sentence:

List the sales of products produced in 1973 with the products produced in 1972.

Sampson, Haigh, and Atwell (1988) and Fujisaki et al. (1989) originally suggested using probabilities to disambiguate all possible sentence analyses. Probabilistic parsing usually requires a training corpus, and ranks analyses based on the frequency of occurrence of certain patterns in the training data. The basics of stochastic (or probabilistic) context-free grammars (SCFGs or PCFGs) and parameter estimation are laid out in Booth (1969) and Booth and Fu (1975) and will be described in Section 3.3. Most probabilistic parsers using PCFG use CKY algorithm. However, Stolcke (1995) presented a version of Earley algorithm suitable for *probabilistic* parsing.

Most natural language parsers are also limited to a specific domain. Wide-coverage grammars exist, e.g., Briscoe et al. (1987), but the parsing performance reported is not as high as with those used for limited domains only. When a domain-specific parser is used to analyse sentences in a different domain, the performance is likely to suffer. Specifically, Prospero, a commercial parser (Oakes, 1994), is very good at parsing newspaper articles, but

when it is applied in a dialogue environment (Batacharia et al., 1999), some constructions are not parsable. For example, Prospero cannot parse sentences like “don’t you like school?” since such sentences do not normally occur in formal writing. In order to be able to parse this sentence (and some other constructions) the sentences which need special handling are pre-selected, modified so that parser can accept them, and the output of the parser is modified again.

Chapter 3

Elements of Statistical NLP

In this chapter I present elements of statistical natural language processing relevant to parsing. For an extensive coverage of statistical NLP see Charniak (1993) and Manning and Schütze (1999). First, in Section 3.1 I discuss some currently available corpora such as the Brown Corpus and the Penn Treebank. Section 3.2 talks about Hidden Markov Models (HMMs) which are essentially finite-state automata augmented with transition probabilities. HMMs have been used successfully in speech recognition and for part-of-speech tagging. HMMs are usually trained on a large corpus using inside-outside probabilities. Context-free grammars can be augmented with probabilities as well as finite-state automata. In Section 3.3 I present the basics of probabilistic (or stochastic) context-free grammars (PCFGs or SCFGs). The inside-outside algorithm can be adapted to work for PCFGs, but it is much more computationally complex than the algorithm for HMMs. Some approaches using the inside-outside algorithm for training grammars and then parsing are presented in Section 3.4.

3.1 Corpora

Corpora, or collections of annotated and computerised text, are the basis for most work in Statistical NLP. Most of the currently available corpora are in English, but collections in

other languages exist (e.g., Japanese (EDR, 1994), Dutch, or German (Skut et al., 1997)). EDR corpus of Japanese is very similar to the Penn Treebank, with one of the main differences being that there are no nonterminal tags. Sentences are annotated with the tree structure not containing nonterminal tag labels. So, Shirai and others (1995; 1997) automatically assign nonterminal tags while extracting the grammar rules. Their approach is very similar to the one described in this work, however, they only perform shallow, 1-level parsing when compacting the grammar. It is not yet clear how the treebank grammar extraction and compaction work can be applied to corpora for free word order languages such as German. Treebank grammar approach relies heavily on context-freeness of the grammar, while in (Skut et al., 1997) there are crossing branches encoding non-context free dependencies.

One of the earliest (and most well known) corpora is the Brown corpus (Francis, 1964; Francis and Kučera, 1982), a collection of American written English used in 1961. It consists of just over 1 million words, annotated with part of speech tags. The corpus is divided into 500 samples of roughly 2000 words each. There are 15 distinct domains ranging from editorial press to humor.

Later, the LOB corpus (Garside, Leech, and Sampson, 1987) was annotated with syntactic structure as well as part-of-speech tags. The SUSANNE corpus consists of a 130,000-word subset of the Brown Corpus (US written English), annotated in a different scheme (Sampson, 1995). A small portion of SUSANNE was used by Carroll, Briscoe, and Sanfilippo (1998) for evaluating the new parsing scheme.

In the Penn Treebank (PTB) (Marcus, Santorini, and Marcinkiewicz, 1993; Marcus et al., 1994; Bies et al., 1995) syntactic annotation was extended and modified, and also the size of the corpus was increased (about 1.25 million part-of-speech tokens in the Wall Street Journal portion alone). The second release of the Penn Treebank has become a standard for much of the modern US work in statistical NLP (Magerman, 1995; Collins, 1996; Collins, 1997). Research presented in this thesis is also based on the Penn Treebank.

In British English, the Spoken English Corpus (Knowles, Williams, and Taylor, 1996;

Knowles, Winchmann, and Alderson, 1996) contains 52,000 words of British English speech, developed jointly by the Linguistics Department at Lancaster and IBM UK Scientific Centre between 1984 and 1991. ICE-GB is the first one of the International Corpora of English (ICE) to be completed. It is made up of 200 written and 300 spoken texts, consisting together of about one million words. The corpus is fully annotated with parse tree structures, containing 83,394 parse trees, of which 59,640 are in the spoken part. Information on (ICE-GB) can be found at www.ucl.ac.uk/english-usage/ice-gb/index.htm.

3.2 Hidden Markov Models

Markov Models (Markov, 1913) were introduced as early as 1913 in attempts to analyse written text, but they did not become widely known until Hidden Markov Models (HMMs) were used for speech recognition (Ferguson, 1980). HMMs are also successfully used for part-of-speech tagging (Jelinek, 1985; Church, 1988; Kupiec, 1992).

Essentially, a Markov Model is a set of states and transitions between these states. Each transition has a transition probability and a symbol attached. The symbol can be viewed as either input or output symbol, depending on the application. Two states are marked as the ‘start’ and ‘end’ states. A Markov Model can accept (or generate) strings by traversing the diagram from the start to the end state. The probability of a string is the product of probabilities of all transitions involved in this string’s derivation. The sum of probabilities of all transitions coming out of a given state should equal 1. For each state and symbol pair there is only one corresponding transition. However, this is not the case for a Hidden Markov Model. There may be several transitions corresponding to a given state and symbol pair. In this case, there is no longer a single path through the diagram – it may split into two or more paths which can split in turn (and then merge if they come back to the same state). An algorithm implementing an HMM has to follow all paths. The probability of a string is the sum of probabilities of each path (which is a product of all probabilities of all transitions involved in this path). For a formal definition of an HMM see, for example, (Charniak, 1993).

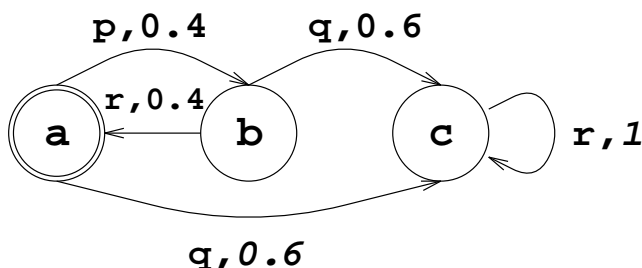


Figure 3.1: A Markov Model

Figure 3.1 and 3.2 show a simple Markov Model and an HMM. a and c are the start and end states, respectively. Both of these models can generate a string pq , but a conventional Markov model can follow only one path: abc , whereas a HMM can follow two paths: abc and aac . The probability of deriving pq is $0.4 \cdot 0.6 = 0.24$ for a traditional model, and $0.4 \cdot 0.6 + 0.2 \cdot 0.4 = 0.32$ for an HMM. Given an HMM and an input string, the Viterbi algorithm (Viterbi, 1967) determines the most probable path through the system.

In part-of-speech tagging, the next tag is predicted based on n previous tags, where n is often 2. Such a model is called a *trigram* model, because a triple of tags (or a trigram) is the basic unit used in tagging. The parameters of a trigram (or n -gram) model are estimated from a large corpus using frequencies of occurrence of certain n -grams or a training algorithm such as the Baum-Welch algorithm (Baum, 1972). Once the parameters are estimated, the tagging task is carried out using the Viterbi algorithm. Many HMM-based taggers also use smoothing techniques which use unigram, bigram and trigram probabilities with different weights. Smoothing is needed to account for less frequently occurring n -grams and shorter syntactic dependencies.

3.2.1 Training HMMs

Using frequencies of occurrence of certain n -grams for estimating parameters of an HMM model is simple, but a rather crude method. Using the Maximum Likelihood framework, one attempts to find the parameters of a model maximising the probability of the training corpus. There is no algorithm to implement the Maximum Likelihood framework for

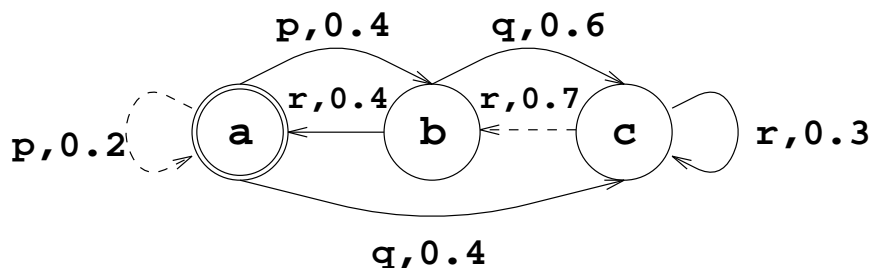


Figure 3.2: A Hidden Markov Model

HMMs. However, the Baum-Welch algorithm (Baum, 1972) approaches a local maximum, and for HMMs it is usually sufficient. However, when the Baum-Welch algorithm is used for probabilistic context-free grammars, it is not quite as effective (see Section 3.4).

It is beyond the scope of this work to give a complete description of the Baum-Welch algorithm (for details, see Charniak (1993)). Here, I will only present the essentials which are as follows. Baum-Welch algorithm is an iterative algorithm which improves the probability of a model after each step. When incremental improvements become small, the estimation stops. The local maximum (or a saddle point) approximated this way depends on the initial set of parameters, which could be random or guessed using the training data. During each iteration step, *forward* and *backward* probabilities are calculated inductively. These probabilities have many useful properties and are essentially the ‘building blocks’ for further derivations. The forward probability $\alpha_i(k)$ is the probability of arriving at a certain state i after first k input symbols have been processed. The backward probability $\beta_i(k)$ is a probability of leaving a certain state i and processing the last k symbols of input. Using forward and backward probabilities, the parameters of the model can then be re-estimated, by recalculating the number of times each transition is traversed when the training data is supplied to the HMM.

3.3 Probabilistic Grammars

As mentioned earlier, in Section 2.4, purely rule-based parsing produces many parses and no easy way to tell which parse is the best. One way to rank different parses is by using probabilities. Fujisaki et al. (1989) (and many others) have successfully used probabilistic context-free grammars (PCFGs) for parsing natural language. The basics of PCFGs were originally defined in Booth (1969), and there is some work on grammar inference in Booth and Fu (1975).

A PCFG is a context-free grammar augmented with a probability attached to each rule. For each nonterminal A the sum of probabilities of all rules which have this nonterminal on the left-hand side should equal 1:

$$\sum_{\lambda_i, \text{ where } A \rightarrow \lambda_i \text{ is a rule in the grammar}} P(A \rightarrow \lambda_i) = 1$$

The probability of a certain derivation is the product of all probabilities of all rules involved in this derivation. If a string has several derivations, its probability is the sum of the probabilities of all derivations. Different derivations of a given string are ranked based on their probabilities, and the derivation with the highest probability (or n best derivations) is used for further processing. Also, the sum of probabilities of all derivations of all strings that can be generated by a given grammar is equal 1 (this is easy to prove by induction, starting from the head nonterminal of the grammar and given the condition above that the sum of probabilities for all rules with a certain nonterminal on the left-hand side is equal 1).

Parsing with PCFGs is usually done with the Viterbi algorithm (Viterbi, 1967), which, for context-free grammars, is a variation of CKY algorithm. Also, Stolcke (1995) adapted the Earley algorithm to use probabilities. In the Viterbi algorithm, each edge has a probability attached, and each time the parser regenerates this edge (this happens because many different derivations are possible), the ‘new’ probability is compared with the ‘old’ probability, and if the ‘new’ probability is higher, it replaces the ‘old’ one.

3.4 Training a grammar with the Inside-Outside Algorithm

The Maximum Likelihood framework can be used for training PCFGs as well as for training HMMs. The equivalents of forward and backward probabilities are the *inside* and *outside* probabilities, demonstrated graphically in Figure 3.3. The algorithm for deriving inside and outside probabilities and using them to reestimate the parameters of a PCFG is described in Kupiec (1991) and Charniak (1993). In this section I present three approaches to using the Maximum Likelihood framework to train PCFGs. Although a local maximum can be achieved, the resulting grammar is not always usable for parsing. As Carroll and Charniak (1992b) note, 300 different initial grammars converged to 300 different local maximums. Also, the *inside-outside algorithm* is considerably more computationally complex than the Baum-Welch algorithm.

Lari and Young (1990) consider a grammar consisting of rules in Chomsky Normal Form (CNF): either $A \rightarrow a$ or $A \rightarrow BC$, where a is a terminal, and A, B, C are nonterminals. The derivations and the parsing procedure become much easier. However, such a grammar uses considerably more parameters. The grammar is pre-trained using an HMM and the Baum-Welch algorithm and then the resulting HMM is converted to a PCFG. This considerably reduces the speed of conversion without using too much computer power.

The inside-outside algorithm was augmented with two techniques: constraining and a grammar minimisation algorithm. Constraining assigns a nonterminal to each terminal so that ‘greedy’ symbols do not take on several nonterminals and thus produce an enormous number of rules. The grammar minimisation algorithm finds these ‘greedy’ symbols, removes them from the grammar, and randomises the probabilities of under-represented stochastic processes.

This approach was tested on a ‘toy’ grammar producing strings of palindromes ($\{xy|x$ is a mirror image of $y\}$) from an alphabet of two symbols a, b . Results were compared with those obtained using an HMM trained with the Baum-Welch algorithm. A significant decrease in entropy was observed. However, a three-fold excess of nonterminals is needed to ensure an adequate distribution of nonterminals to each of the hidden processes. Given the

$O(n^3)$ complexity (n is the number of nonterminals) of the inside-outside algorithm this is potentially crippling.

Pereira and Schabes (1992) modified the inside-outside algorithm to use bracketed sentences representing the syntactic structure of the sentence so that the algorithm converges in fewer iterations. On a palindrome language similar to the one Lari and Young (1990) used this algorithm converged very quickly to the ‘almost optimal’ grammar for this language.

The ATIS corpus (which is a part of the Penn Treebank) was then used for training and testing. A crossing brackets measure (see Chapter 7 for details) was used to evaluate the quality of the resulting parses. 90% non-crossing bracket measure rates well compared to 35% using the conventional inside-outside algorithm. However, no *precision* or *recall* figures are reported. Grammar learning using partially bracketed corpus rather than raw text converges faster and to a better grammar.

Carroll and Charniak (1992a) applied the inside-outside algorithm to learning a dependency grammar which is a formalism equivalent to a CFG. The inside-outside algorithm produced 300 different local minima for 300 different starting grammars and also it tends to ‘memorise’ longer rules. Then, they augmented learning process using ‘pseudo-negative’ examples. These are the sentences that can be generated by the grammar, but do not occur in the training corpus with the similar frequency. At each step of training, the algorithm generates a few sentences and compares the calculated probability of each sentence with the frequency of its occurrence. In process of determining whether a certain sentence is a pseudo-negative example, Chebyshev’s equation is used. If a sentence is a pseudo-negative example, all rules forming this sentence are removed from the grammar one-by-one (but only if the removal of each rule reduces the cross-entropy, to make sure the fundamental rules such as $S \rightarrow NP VP$ remain in the grammar). Also, Carroll and Charniak use the *incremental learning* approach, instead of the traditional *batch* approach to further augment the learning process. In incremental learning, the corpus is reread from the beginning at each step of iteration (as opposed to ‘feeding’ one sentence at a time). This achieves faster convergence to a better grammar at the cost of tremendous computational complexity. Use

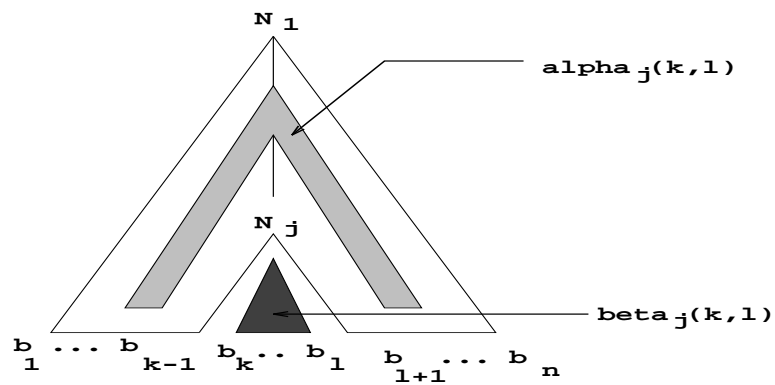


Figure 3.3: Illustration of inside and outside probabilities

of a 'sliding frame' is proposed to reduce the complexity.

Briscoe and Waegner (1992) show that Baum-Welch grammar learning combined with linguistic constraints is a powerful technique for extending coverage of PCFGs, yet PCFGs are not very good in selecting the correct parse.

Chapter 4

Contemporary Research in Statistical Parsing

In this chapter I present some modern work in statistical parsing related to parsing with probabilistic context-free grammars (PCFGs), treebank grammar extraction, data-oriented parsing (DOP), and linguistically motivated tree transformations. First, Section 4.1 reviews three approaches to parsing with PCFGs: by Stolcke, Sekine, and Hepple. All of these parsers have been used to conduct grammar compaction experiments presented in this dissertation. Treebank grammar approaches, described in Section 4.2 use corpora annotated with syntactic structure to extract the underlying grammar. Parsing with extracted grammars achieves high parsing accuracy, but the grammar size is very large. The grammar compaction approach presented in this dissertation extends grammar extraction by attempting to reduce the size of the extracted grammar. The Data-Oriented Parsing (DOP) approach briefly described in Section 4.3 takes probabilistic CFG parsing further by considering tree chunks as building blocks for sentence derivations. The DOP approach can achieve high parsing accuracy, but at the cost of tremendous computational complexity. Section 4.4 describes the work by Johnson who investigates how parse tree transformations can affect the PCFG extracted from the resulting trees. Indeed, he achieves up to 8% improvement in precision and recall while parsing with the extracted grammar.

4.1 Probabilistic CFG Parsing

4.1.1 Stolcke – Probabilistic Earley Parsing

Stolcke (1994a; 1994b; 1995) augmented the Earley’s parsing algorithm to use probabilities and to handle arbitrary context-free rules, especially unary and null-production rules.

Left recursion and unit productions can cause Earley’s algorithm to generate an infinite number of edges. However, for stochastic grammars individual probabilities become smaller and smaller, and their sum converges to a limit. Stolcke has shown that such a limit exists and how to calculate it. This is done by using matrices, and, according to Stolcke (1995) the algorithm is as follows:

Matrices Two matrices are needed: one for the left recursion and the other one for the unit productions. Their size is equal to the number of nonterminals in the grammar. In both cases, an initial matrix is first created, and then its *closed* form is calculated using the following equation:

$$Cl = (I - In)^{-1}, \quad (4.1)$$

where I is the unit matrix, Cl is the closed form, and In is the initial form. Formally, Cl is the reflexive, transitive closure of In .

In initial matrices each element corresponds to a pair of nonterminals. This element is filled with the value of the relation function: the left-corner relation for left recursion and the unit-production relation for unit productions.

1. Left Recursion

The probabilistic left-corner relation applied to two nonterminals X and Y is the total probability of rules in grammar that have X as the head and the right hand side of which begins with Y :

$$L(X, Y) = \sum_{X \rightarrow Y\lambda \in G} P(X \rightarrow Y\lambda)$$

2. Unit Productions

Probabilistic unit relation applied to two nonterminals X and Y is the probability of the rule $X \rightarrow Y$ if it exists, 0 otherwise:

$$U(X, Y) = P(X \rightarrow Y)$$

If there is no left recursion and no unit productions in the grammar, both matrices in the formulae become unit matrices.

Basic Operations: Prediction, Scanning and Completion Edges are now augmented with their forward and inner probabilities (α and γ respectively). The forward probability of an edge is similar to that defined in Section 3.2 on the inside-outside algorithm, and the inner probability of an edge is its ‘total’ probability, i.e., the sum of probabilities of all derivations of this edge. The three steps of Earley’s algorithm now look as follows: (λ and μ are sequences of terminals and nonterminals and i and k are positions in the chart)

- **Scanning**

Scanning is not affected by left-recursion and unit-production, thus no matrices are needed: the probabilities are just copied.

$$(k, i) : \quad X \rightarrow \lambda.a\mu \quad [\alpha, \gamma]$$

where $a = x_i$ add the edge

$$(k, i + 1) : \quad X \rightarrow \lambda a.\mu \quad [\alpha, \gamma]$$

- **Prediction**

For all

$$(k, i) : \quad X \rightarrow \lambda.Z\mu \quad \text{add the edge} \quad (i, i) : \quad Y \rightarrow .\mu \quad [\alpha', \gamma']$$

for all rules $Y \rightarrow \mu$ such that $R_L(Z, Y) \neq 0$, and

$$\alpha' += \alpha \cdot R_L(Z, Y) \cdot P(Y \rightarrow \mu)$$

$$\gamma' = P(Y \rightarrow \mu)$$

- **Completion**

For all pairs

$$\left. \begin{array}{l} (j, i) : \quad Y \rightarrow \mu. \quad [\alpha'', \gamma''] \\ (k, j) : \quad X \rightarrow \lambda.Z\mu \quad [\alpha, \gamma] \end{array} \right\} \text{add the edge } (k, i) : X \rightarrow \lambda Z.\mu \quad [\alpha', \gamma']$$

where Y, Z are such that $R_U(Y, Z) \neq 0$, and $Y \rightarrow \mu$ is not a unit production. Then

$$\alpha' + = \alpha \cdot \gamma'' \cdot R_U(Z, U)$$

$$\gamma' + = \gamma \cdot \gamma'' \cdot R_U(Z, U)$$

4.1.2 Sekine – Parsing with Two Nonterminals

Sekine and Grishman (1995) and Sekine (1998) extract a grammar from the Penn Treebank using an algorithm similar to the one described in this thesis. However, they simplified the grammar by using only two nonterminals in order to produce simpler structures, S and NP . Any other nonterminals are removed and the structure is flattened. For example, if the following subtree occurs in the Penn Treebank:

(S NP (VP (VP VBX (ADJP JJ) CC (VP VBX NP))))

then Sekine's extraction algorithm will produce the following rule:

S -> NP VBX JJ CC VBX NP

This approach produces a large number of long rules: 32,296 distinct rules have been extracted from 94% of the Penn Treebank (23,386 for S and 8,910 for NP).

To cope with the large number of rules, Sekine developed a chart parser which can handle a large grammar. The key technique is to factor grammar rules with common prefixes. For example, for all edges beginning with DT there is one index entry, which points to other indexes, for edges beginning with $DT NN$, $DT NNX$, etc. This way the amount of storage (per edge) is significantly reduced, and also searching for edges is more efficient.

Results are reported on a sample of 1,989 sentences in terms of Parseval metric (see Black (1991) and Chapter 7). 33.9% of sentences in the sample have no crossing brackets with the standard tree, and recall and precision are respectively 73.43% and 72.61%. This compares well with rule-based parsers: Black (1993) cited the best non-crossing score using traditional grammars as 41% and the average of several systems as 22%.

Despite the parser's efficiency, the grammar still had to be thresholded (i.e., rules occurring less than 2 times were removed). Otherwise, the parser runs out of memory on long sentences (about 15% of the sample). Also, in the newer release of his parser Sekine uses 5 nonterminals instead of 2, so that less flat structures are produced.

4.1.3 Hepple – Memory-efficient Chart Parsing

Hepple's parser (developed at Sheffield University but undocumented) was written in response to the demand for a parser that can handle a large grammar (10,000+ rules) parsing long strings (up to 40 tags) of part-of-speech tags. Memory and time are two major limitations for such parser, and several techniques were used to reduce resource requirements to the minimum.

The overall algorithm is bottom-up, left-to-right and uses active edges. The edges are carefully enumerated so that when an inactive edge is added to the chart any active edges that may combine with it are already present. The grammar is pre-compiled into a network similar to the one used by Sekine, factoring grammar rules with common prefixes on the right-hand side (this is described in Section 4.1.2). Each inactive edge is stored standardly, including their span, grammar category, Viterbi (maximum) probability, and the sequence of daughter edges which were used to build it. The particular combination of the daughter edges refer to the Viterbi derivation (the one with the maximum probability) of the inactive edge.

Because the grammar has been pre-compiled into a transition network, active edges can be stored very efficiently, by referring to their span, a sequence of the daughter nodes and a pointer to the right node in the grammar network (i.e., a rule and a position of the dot

within that rule). In the C implementation of the parser, active edges are stored in an array indexed by the span. Each array entry refers to a binary-branching structure which allows efficient access. All stored edges are indexed by the pointer to the grammar to allow fast search.

For edge combinations, active edges are not used directly. Instead, all possibilities for extending active edges are pre-computed, indexing them by the span and a specific following daughter category. This is possible as long as the restrictions on the edge enumeration are in place. If so, all active edges that can possibly be added at a given stage are already in the chart. The *successors table* is then computed and used for further combinations.

The time and memory efficiency achieved with this parser are tremendous. We are able to parse strings of 40 part-of-speech tokens with a grammar consisting of 15,420 rules (see Chapter 8 and (Krotov et al., 1999)). A full search of the parse tree is performed, resulting in 800,000+ active edges for the longest sentences. The system runs on a Linux workstation with a 400 MHz Pentium II and 256 Mb of memory. It takes about 3 hours to parse a 2,245 sentence sample (and less for smaller grammars, proportionally to the grammar size). The average sentence length is about 21 word tokens. We are not aware of any other system achieving a comparable accuracy, time efficiency and at the same time conducting a *full* search of the parse tree. Moore (2000) also presents an efficient left-corner parser using the Penn Treebank grammar. However, his test size is much smaller (30 sentences, which takes 27 seconds to parse), and so is the average sentence length of the sample (5.7 word tokens). Therefore, Moore's results are not directly comparable to Hepple's.

The time and memory issue is traditionally solved by pruning the search tree (e.g., using beam search, (Paeseler, 1990)). In our case, full search is necessary for grammar compaction, for which a modified version of the language parser is used. Caraballo and Charniak (1998), developed the figures of merit to eliminate edges, based on a scoring function which may be rather computationally complex. Parser performance can be improved by varying parameters of this model. Caraballo and Charniak's method can also handle *n*-best parsing.

4.2 Treebank Grammars

Treebank grammar learners acquire a grammar by extracting it from a corpus annotated with syntactic structure (a treebank). This approach achieves good parsing accuracy with relatively little effort, but a large hand-annotated corpus is needed. Section 4.2.1 describes an early attempt by Sharman, Jelinek, and Mercer (1990) to extract P-ID/LP grammars which was tested successfully on a small corpus. However, the scale of this experiment is very small. Following early work in extracting a grammar from the Penn Treebank Krotov, Gaizauskas, and Wilks (1994), Gaizauskas (1995) carried out a detailed investigation of the extraction process. Some modifications of the input trees were necessary to make the resulting program usable for parsing, but no formal parsing experiments were conducted. The extraction procedure finally adopted is described in Section 4.2.2. The grammar extracted using this method was used for the experiments described in this thesis. Charniak (1996) extracts a grammar in a similar way and uses it in a chart parser to achieve good parsing accuracy. His method is described in Section 4.2.3. Finally, Section 4.2.4 presents work by Shirai, Tokunaga, and Tanaka (1995) in extracting a grammar from a Japanese treebank (the EDR corpus).

4.2.1 Sharman – P-ID/LP grammars

Sharman, Jelinek, and Mercer (1990) have shown good results in acquiring a P-ID/LP grammar from a treebank and parsing with this grammar. However, to achieve a reasonable speed of computation, the number of terminals was reduced by mapping several terminals into one. The P-ID/LP grammar used by the authors is the probabilistic version of the ID/LP grammar where simple context-free rules are split in two orthogonal sets of rules: immediate dominance (ID) and linear precedence (LP) rules. The assumption is that dominance and precedence relations are independent. A P-ID/LP grammar was determined by observation (what is call “extraction” in this dissertation) from Associated Press (AP) newswire material containing about one million words. The corpus was available in the form of a treebank with about 45,000 different phrase types marked by manual parsers. A vari-

ant of a CKY parser was used to compute the probability of each sentence. Part-of-speech tagging was performed as well as actual parsing.

To increase the processing speed both terminal and nonterminal sets were reduced: the nonterminal set from 64 to 16, and the terminal set from 264 to 100. Several (non)terminals were mapped into one to convert the sentences to the new format. The test set used is small — only 42 sentences chosen at random, subject to restriction that the sentence length was less than 30 words. 18 sentences were correctly parsed, and 19 were “similarly” parsed — this is a 88% accuracy. Some causes of the imperfect parses were due to the choice of non-terminal symbols. The treebank symbols were insufficiently specific and their number was reduced for efficiency.

4.2.2 Gaizauskas – Treebank Grammar Extraction

Gaizauskas (1995) presents a detailed investigation of extracting the grammar from the Penn Treebank and the resulting grammar. The extraction process was modified to account for the suffix tags and extra constituents such as `-NONE-` (see Section 5.2.2 for details). 17,534 rules were extracted from 49,208 sentences containing 1,253,013 part-of-speech tokens.

4.2.3 Charniak – Parsing with a Treebank Grammar

In parallel with our efforts in Sheffield (Krotov, Gaizauskas, and Wilks, 1994; Gaizauskas, 1995), Charniak (1996) conducted parsing experiments on a grammar extracted from the Penn Treebank. The grammar was extracted in a way similar to the one described in Gaizauskas (1995) with an addition of *right-branching bias*, a procedure to readjust rule probabilities to favor right-branching structures. Adding a right-branching bias improves parsing accuracy (precision and recall) by approximately 2-3%. English is traditionally recognised as a right-branching language.

The results are similar to ours: 15,953 rules were extracted from the training corpus of roughly 1,000,000 words (a slightly smaller training corpus was used compared to (Gaizauskas, 1995), and so the number of extracted rules is also smaller). Only 6,785 of

these rules occurred more than once. Removing rules that occur once does not affect parsing accuracy significantly, and so the smaller grammar is used to produce parsing results. PARSEVAL (Black, 1991) precision and recall of 82% and 80% are reported. This accuracy is very impressive considering no lexical information was used (i.e., the parser takes a string of part-of-speech tags as input). Ambiguity is noted to be a major issue, due mainly to the size and the arbitrary nature of treebank grammars. Figures of merit are used to improve the parser's efficiency by reducing the number of edges that need to be processed.

4.2.4 Shirai – Japanese Treebank Grammar

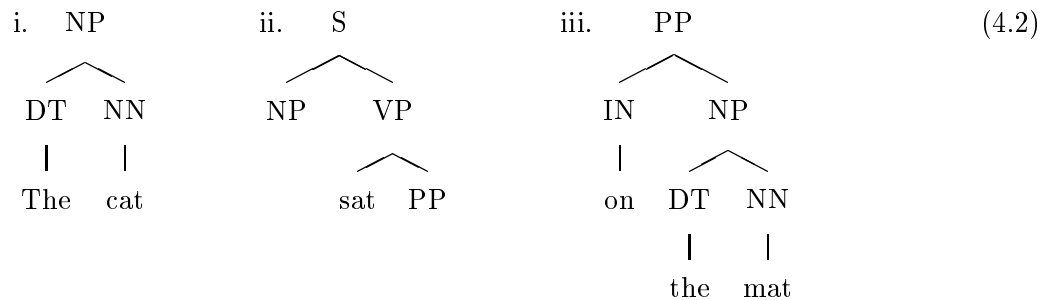
Shirai and others (1995; 1997) conducted experiments in extracting a treebank grammar for Japanese from the EDR corpus (EDR, 1994). The extraction algorithm is similar to that described in Krotov, Gaizauskas, and Wilks (1994), Gaizauskas (1995) and Charniak (1996). One of the primary differences is that EDR does not have nonterminal tags, so sentences are annotated with tree structures, but nonterminal nodes are unlabelled. Shirai derives nonterminal symbols from the last terminal covered by the nonterminal's span, excluding punctuation marks and end markers. This is motivated by head-final nature of Japanese language. For example, if a nonterminal covers a string ending with *meisi* (noun), it is assigned the *meisi-ku* label. Probabilities are estimated standardly, using rule counts. Some linguistic knowledge specific to Japanese is incorporated in the extraction algorithm. For example, compound words are converted into a right-branching tree structure in order to suppress unnecessary parse trees. For the same reason, part-of-speech tags corresponding to punctuation symbols and to postpositions are subcategorised.

Shirai also carries out grammar compaction similar to the compaction described in Chapter 6. However, Shirai's algorithm is shallow, as it takes only one level of depth into account when deciding whether certain rules should be eliminated. The grammar compaction algorithm presented in this thesis considers all possible combinations by which a rule can be expanded with other rules using a modified version of the context-free parsing algorithm.

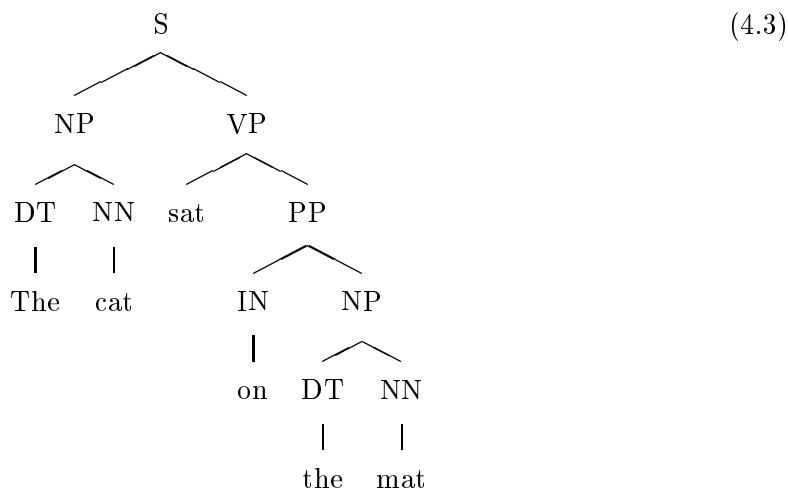
2,219 rules were extracted from 180,000 sentences. On a training set of 20,000 sentences the Tomita GLR parser (Tomita, 1986) achieves 62% recall and 74% precision (using an algorithm comparable to the one in PARSEVAL (Black, 1991)). 92% of all test sentences can be parsed with the extracted grammar. Many sentences could not be parsed because of memory limitations. An average of 10^9 possible parses per sentence is reported.

4.3 Data-Oriented Parsing (DOP)

The Data-Oriented parsing approach, developed by Bod, Scha and Bonnema (Bod, 1992; Bod, 1993a; Bod, 1993b; Bod, 1995a; Bod, 1996; Bod and Scha, 1996; Bod, Bonnema, and Scha, 1996; Bonnema, Bod, and Scha, 1997) goes a step beyond context-free grammars by considering subtrees as if they were grammar rules. Subtrees can be combined to derive the entire sentence parse. Context-free rules are subtrees of depth 1. For example, three subtrees



can be combined to produce the entire sentence



The DOP grammar learner uses treebanked text (i.e., text annotated with syntactic structure) to collect all possible subtrees used in the corpus. Due to the number of possible combinations, the number of subtrees is much larger than the number of rules extracted by a treebank grammar learner (which is already very large). Therefore, the computational complexity of the DOP method is tremendous. Also, the number of ways a parse of a sentence (e.g., (4.3)) can be split into chunks grows exponentially with the size of the parse tree (the tree can be either split or not at each one of n non-leaf nodes giving 2^n possible combinations). Some DOP variations limit the depth of extracted subtrees to improve efficiency.

Indeed, Sima'an (1996) shows that parsing with DOP is NP-hard. Essentially, this rules out using DOP for any large scale practical application. Bod shows good accuracy in using DOP for parsing, but the test sentences from ATIS or OVIS corpus are short (5 words, (Bod and Scha, 1996)). The test sample size is also small (100 sentences as opposed to 2,000+ sentences for most Penn Treebank-related approaches). Collins (1999b) criticises DOP approach because there is no efficient parsing algorithm and the training and test sample size is small. Goodman (1996; 1998) presents an efficient parsing algorithm for a variety of DOP which runs 500 times faster than the algorithm presented in (Bod, 1995b).

4.4 Johnson – PCFG models of linguistic tree representations

Johnson (1998b) argues that PCFGs are simple models for describing language and their simplicity makes it worth investigating their properties further, even though they do not capture as many dependencies as more complex models such as the ones described in (Collins, 1999a; Charniak, 2000).

Even though the context-free formalism is not sufficient for describing natural language (Shieber, 1985), the statistical independence assumption encoded in the PCFG is in fact stronger than the context-free assumption. For example, Johnson shows that two PCFG models of PP attachment, one which is an extension of a CFG model and the other which is derived from non-CFG sources perform similarly.

Johnson studies how varying the tree representation model annotating a sample of text affects the PCFG extracted from such sample. He starts with a sample of trees for section 22 of the WSJ corpus, and suggests six alternative tree representation models: maximum likelihood parse trees using the PCFG induced from those trees, and five models produced using transformation-dettransformation process using simple node relabelling.

Indeed, tree transformations affect parsing with the extracted PCFG, adding up to 8% to the precision and recall figures, and making PCFG parsing comparable with parsing using other, best broad coverage models.

Chapter 5

Extracting the Penn Treebank Grammar

This chapter presents the results of our early experiments in extracting a grammar from the Penn Treebank (PTB) (Marcus, Santorini, and Marcinkiewicz, 1993; Marcus et al., 1994). Section 5.1 discusses the first phase of the Penn Treebank project, and presents our first results in rule extraction (also reported in Krotov, Gaizauskas, and Wilks (1994)). We have observed a significant growth in the number of rules and no tendency for this rate of growth to slow down. Such growth suggests the need for reducing the size of the extracted grammar. One way is by removing less frequent rules, or *thresholding* the grammar. The other way is grammar compaction, discussed in detail in Chapter 6 and Krotov and others (1997; 1998; 1999). Grammar compaction retains grammar coverage without a significant loss in parsing accuracy. Section 5.2 discusses the improvements made in the second release of the PTB and how this affected our extraction algorithm.

Parsed text	POS tagged text
((S	[The/DT new/JJ rate/NN]
(NP The new rate)	will/MD be/VB
will	[payable/JJ Feb./NP 15/CD]
(VP be	./.
(ADJP payable)	
(NP Feb. 15))	
.)	

Figure 5.1: Sample text from the Penn Treebank I

5.1 Rule Extraction – Early Experiments

5.1.1 First version of the Penn Treebank

The Penn Treebank is a large annotated corpus of English created at the University of Pennsylvania (Marcus, Santorini, and Marcinkiewicz, 1993; Marcus et al., 1994). It consists of 4.5 million words of American English, the largest proportion of which consists of materials from the Wall Street Journal. During the first phase of the project (1989–1992), this corpus was annotated for part-of-speech information. And over half of it was annotated with skeletal syntactic structure.

The Penn Treebank I contains two sets of files: POS tagged files and skeletally parsed files. Samples of each kind of file are shown in Figure 5.1. POS tagged files consist of lists of terminal tokens (i.e., words and punctuation) and their part-of-speech tags. Noun phrase groupings are also included, but we do not use these for grammar extraction. The skeletally parsed files consist of Lisp-like sentence structures (or parse trees) with syntactic constituents delimited with parentheses. These structures do not include any POS information. However, the terminal content of the parse trees should match that of the corresponding POS files. To match the terminal tokens in the parse trees with their corresponding tags we use the matching algorithm described in the next section.

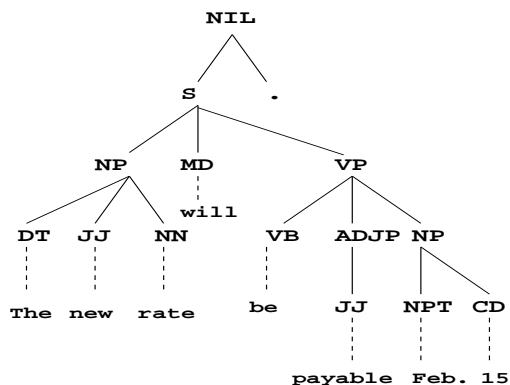


Figure 5.2: Sample parse tree

5.1.2 Matching Algorithm

The matching algorithm replaces the words in the parsed structure with their terminal tags. Consider the sentence shown in Figure 5.1. The algorithm first opens the tagged and parsed files and then matches each token from the parsed file (i.e., a word token or a terminal tag such as DT) with the terminal in the tagged file. If there is no match, we look up the current symbol in the set of nonterminals (and if the current symbol is a nonterminal, we can skip it). If there is still no match, we print an error message. We leave nonterminal tags unchanged, but replace the terminal tokens (i.e., words and punctuation) with their corresponding POS tags. The resulting parse structure consisting of nonterminal tags and POS tags in place of words is then used for grammar extraction.

For instance, **S** does not match **The** but is a nonterminal symbol. However, **The** matches the next token in the tagged file and is consequently replaced with DT. Similarly, **new** is replaced with JJ, and so on. The resulting parse tree is as follows (or, in graphical form, see Figure 5.2):

```
((S (NP DT JJ NN ) MD (VP VB (ADJP JJ) (NP NPT CD))) .)
```

From the resulting structure we extract all context-free productions that are used to derive it. This is done recursively, starting from the top node. The rules extracted from the sample above are as follows:

$$\begin{array}{lll}
 NIL \rightarrow S & . & NP \rightarrow DT JJ NN \quad NP \rightarrow NPT CD \\
 S \rightarrow NP MD VP & & ADJP \rightarrow JJ \quad VP \rightarrow VB ADJP NP
 \end{array}$$

This process is repeated over all sentences in the corpus storing all distinct context-free productions (or rules). For each rule we count the number of occurrences. This is used later for assigning a probability to each rule. We do this in a standard manner, by dividing the rule count by the number of times its left-hand side (LHS) nonterminal occurs as a LHS in the entire extracted set.

5.1.3 Results

We implemented the matching algorithm and the rule extraction algorithm as described above. For the test set, we chose to use the Wall Street Journal articles, because other subsets of the Penn Treebank are too small and too specific. From a corpus of 45,059 terminal tokens we extracted 2,733 rules.

A simple Prolog chart parser similar to the one described in (Gazdar and Mellish, 1989) was used to test the plausibility of the extracted grammar. For efficiency, we used thresholding to reduce the number of rules. This parser could parse short sentences from the PTB and would often come up with the same sentence structure as marked up in the treebank.

At this point we stopped and decided not to proceed further mainly because of the inconsistencies in the Penn Treebank data files we encountered. These inconsistencies would break the matching algorithm more often than we had anticipated. We also observed that the grammar underlying the treebank varied from one set of files to another.

5.1.4 Problems with the Penn Treebank

Because of the preliminary nature of the Penn Treebank I some problems were bound to emerge. We mention these problems not to be critical — for without the Penn Treebank our investigation would have been impossible — but to indicate difficulties that can be

expected. At the time this work was done, we had hoped that when the next release of the Penn Treebank project was completed, the contents of this section will no longer be true.¹

The problems encountered can be split into three different categories. First, POS tagging and skeletal parsing were carried out as separate tasks. Consequently, there are inevitable mismatches between two sets of files. For example, curly brackets are coded *LCB* in a parse file as opposed to { in a POS file. Also, terminal entries containing a slash / are shown in full in POS files, but in parsed files everything before a slash is cut off. For example, 1/8 is coded as 1/8 / CD in a POS file, compared to 1 in a parsed file. This kind of problem causes us to make too many ad-hoc assumptions about the input files, making the matching algorithm too complex.

Secondly, the set of nonterminals used in the parsed files is larger than that presented in Marcus, Santorini, and Marcinkiewicz (1993). For example, non-documented nonterminals such as AUX, COMP, ORD, PRT were encountered in the process of extraction. In other cases, variants or misspellings of tags appeared (e.g., N, V).

Finally, the encoding conventions appear to be slightly different in different sets of files. Particularly, the encoding of the top-level constituent is inconsistent. There is no symbol for the root nonterminal so we added an artificial symbol S1 at the top of the tree. When this is done, some sentences have the production S1 -> S <fullstop> at the top of the tree, whereas other sentences have S1 -> S and S -> NP VP <fullstop>. In other words, the coders could not decide whether the fullstop should be included within the S nonterminal or whether it should be at the top level. We then end up with two sets of rules having S as their LHS: the ones with and without the fullstop at the end, which was very confusing.

5.1.5 Conclusions

In this early work, we implemented and evaluated the algorithm that extracts grammatical information from the Penn Treebank. We showed that the grammar we extracted can be

¹And indeed, Penn Treebank II provides 'combined' files which contain the sentence structure, word tokens and their matching tags. Thus, the matching algorithm as described is no longer necessary.

used in a straightforward probabilistic chart parser. This parser gave the correct analysis for short sentences.

At the time this work was done (1994) we could not implement an efficient parser for the extracted grammar, as did we know enough about statistical parsing. Instead, we used a trivial Prolog parser with a reduced grammar. Parsing with a large number of rules is computationally complex. One way to handle the efficiency is by using an intelligent grammar reduction algorithm. Such an algorithm would allow us to reduce the size of the extracted grammar while losing as little coverage as possible. We decided to attempt to reduce the grammar size rather than using a more efficient parsing technique (or pruning), because we expected to use treebank grammar extraction on a larger corpus (and thus the size of the extracted grammar would be larger also). Penn Treebank I is in many ways a preliminary release (compared to PTB II), and only a small portion of text is annotated with tree structure. Not losing coverage while reducing the grammar is important as we are trying to preserve as much information extracted from the treebank as possible.

Similar work was reported in Sharman, Jelinek, and Mercer (1990). However, Sharman simplified the task by reducing the number of nonterminals and using a P-ID/LP grammar that is easier to use in a parser. The resulting grammar is small and a parser using this grammar produces accurate sentence analyses.

5.2 Second Release of the Penn Treebank

The second release of the PTB is much more consistent in its content and and also a much larger proportion of the corpus is parsed, and therefore it is more suitable for the extraction task.

5.2.1 Improvements in the Second Release

Apart from removing inconsistencies and including more parsed text, two other major improvements were made: first, *merged* files were created, and also a new syntactic annotating scheme was implemented.

In merged, or combined files all words in the parsed structures are augmented with their part-of-speech tags. The matching algorithm similar to that described earlier is now carried out by the annotators, and the matching program is supplied on the PTB CD-ROM. A sample of merged text is shown in Figure 5.3. The extraction algorithm can then use merged files directly.

Following the first, preliminary release of the Penn Treebank, many of its users wanted richer forms of annotation than provided originally. Specifically, many users wanted to see some form of predicate-argument structure. In response to the users' requests, a new tagging scheme was implemented (Marcus et al., 1994). The new scheme contains a base set of tags as well as a set of *functional* tags. The base tags are similar to those used in PTB I and consist of 45 part-of-speech tags, 26 syntactic tags and 10 null element tags. See Tables A.1 and A.2 for the entire list of syntactic and part-of-speech tags. A set of null tags has been extended to account for phenomena such as wh-movement, passive, and the subjects of infinitival constructions.

Functional tags are new in the PTB II, and are attached to the syntactic tags with a hyphen. They are used to extend the meaning of the base tag, and indicate such functions as

- text category (e.g., headline -HLN, titles -TTL),
- grammatical function (e.g., adverbials -ADV, logical subjects in passives -LGS), and
- semantic role (e.g., direction -DIR, location -LOC, temporal -TMP).

The entire list of functional tags is presented in Table A.3.

Finally, semantic and null tags can be augmented with a numeric tag attached with either hyphen or an equals sign. Numeric tags provide a non-context free mechanism to attach clauses which are otherwise disconnected.

Consider a sentence from the PTB II in Figure 5.3. Here, -TMP (temporal) attached to the prepositional phrase indicates the semantic role of this PP. The null element *-2 is linked with NP-SBJ-2 ('the second ship') to show that this noun phrase is assumed in place

Merged text

```

( (S
  (S
    (NP-SBJ-2 (DT The) (JJ second) (NN ship) )
    (VP (VBZ is)
      (VP (VBN scheduled)
        (S
          (NP-SBJ-1 (-NONE- *-2) )
          (VP (TO to)
            (VP (VB be)
              (VP (VBN delivered)
                (NP (-NONE- *-1) )
                (PP-TMP-3 (IN in)
                  (NP (NN fall) (CD 1990) ))))))))
        (CC and)
        (S
          (NP-SBJ=2 (DT the) (JJ third) )
          (PP-TMP=3 (IN in)
            (NP (NN fall) (CD 1991) )))
          (. .) ))
  )
)

```

Figure 5.3: A sample of merged text from the Penn Treebank

of the null element for predicate-argument interpretation. Tags PP-TMP-3 and PP-TMP=3 indicate that these two prepositional phrases ('in fall 1990' and 'in fall 1991') are parallel, and the missing material in PP-TMP=3 can be recovered from that in PP-TMP-3.

5.2.2 Modified Extraction Algorithm

It is not clear whether all extra information encoded using the new annotation scheme is needed for extracting a context-free grammar. First, the numeric indices provide non-context free grammar information which cannot be used in a context-free parser.

Functional tags may be potentially useful in parsing. However, if all combinations of base and functional tags are treated as separate tags, we could have too many tags for a parser to cope with. Some base tags have two functional tags attached (e.g., PP-PRD-TPC).

Null elements are primarily used in the Penn Treebank so that they can be co-indexed with the relevant context in other clauses. This is done to make the predicate-argument

structure of the sentence clear. Since we cannot use numeric indices, null elements are not of much use to us either. Besides, null elements generate ϵ -production rules, which require special treatment in parsing, since they can cause a combinatorial explosion of the amount of search space required (Stolcke, 1995).

If null elements are removed, then the rules in which they occur may reduce to unary rules (i.e., those of the form `nonterminal` \rightarrow `nonterminal`). It is not clear how important such rules are to us, and they again require special treatment in parsing (Stolcke, 1995). For simplicity, and since we are not using semantic information encoded in the tree, we want to merge such rules with their mother nodes. In fact, many unary rules arise because of removal of the null elements in the tree underneath. But then, what tag should we assign to the resulting node? For instance, in Figure 5.3 removing the null tag `*-2` results in ϵ -production `NP` $\rightarrow \epsilon$, which is removed, generating the unary rule `S` \rightarrow `VP`. In this case, the ‘child’ `VP` tag describes the contents of its constituent better than the ‘mother’ `S` tag, because `S` also assumes the presence of the subject `NP` which was removed. Our model is too simple to hypothesise a potentially null noun phrase constituent. It seems that there are not very many unary rules explicitly coded in the PTB, and so most unary rules are caused by removal of null elements. But even for the unary rules encoded in the treebank moving the child node into its mother’s place seems appropriate (Gaizauskas, 1995).

The exact tag simplification procedure adopted is as follows (Gaizauskas, 1995):

1. eliminate all hyphen- or equals-attached suffix tags;
2. eliminate any constituents whose label is `-NONE-` (all null element tags in PTB II are uniformly treated as lexical tags occurring in a unary constituent whose label as `-NONE-`);
3. if as a result of 2. any constituent now has an empty set of children then remove this constituent (this may percolate recursively up the tree);
4. if any constituent now has a single child (either as a result of 3. or because it was initially created this way) then remove the constituent and then merge the child into

parent's position in the next higher level constituent (this has the effect of removing all unary rules whose right-hand sides are not word class tags).

The simplification process as applied to one of the *S* clauses in the sample in Figure 5.3 is shown step-by-step in Figure 5.4.

5.2.3 Extracting the Grammar

A grammar was extracted from the Wall Street Journal portion of the Penn Treebank using the scripts referenced in Gaizauskas (1995). From 49,208 sentences containing 1,253,013 part-of-speech tokens we extracted 17,534 rules. In line with our earlier experiments, the number is very large. Figure 5.5 shows the growth of the number of rules as a function of the number of sentences supplied to the extraction algorithm. As expected, the number of rules continued to grow with no obvious tendency to approach a limit. This suggests a need for compacting the extracted grammar.

Clause from the treebank:

```
(S
  (NP-SBJ-1 (-NONE- *-2) )
  (VP (TO to)
    (VP (VB be)
      (VP (VBN delivered)
        (NP (-NONE- *-1) )
        (PP-TMP-3 (IN in)
          (NP (NN fall) (CD 1990) ))))))))
```

after step 1:

```
(S
  (NP (-NONE- *) )
  (VP (TO to)
    (VP (VB be)
      (VP (VBN delivered)
        (NP (-NONE- *) )
        (PP (IN in)
          (NP (NN fall) (CD 1990) ))))))))
```

after step 2:

```
(S
  (NP )
  (VP (TO to)
    (VP (VB be)
      (VP (VBN delivered)
        (NP )
        (PP (IN in)
          (NP (NN fall) (CD 1990) ))))))))
```

after step 3:

```
(S
  (VP (TO to)
    (VP (VB be)
      (VP (VBN delivered)
        (PP (IN in)
          (NP (NN fall) (CD 1990) ))))))))
```

after step 4:

```
(VP (TO to)
  (VP (VB be)
    (VP (VBN delivered)
      (PP (IN in)
        (NP (NN fall) (CD 1990) ))))))))
```

Figure 5.4: Tag simplification illustrated

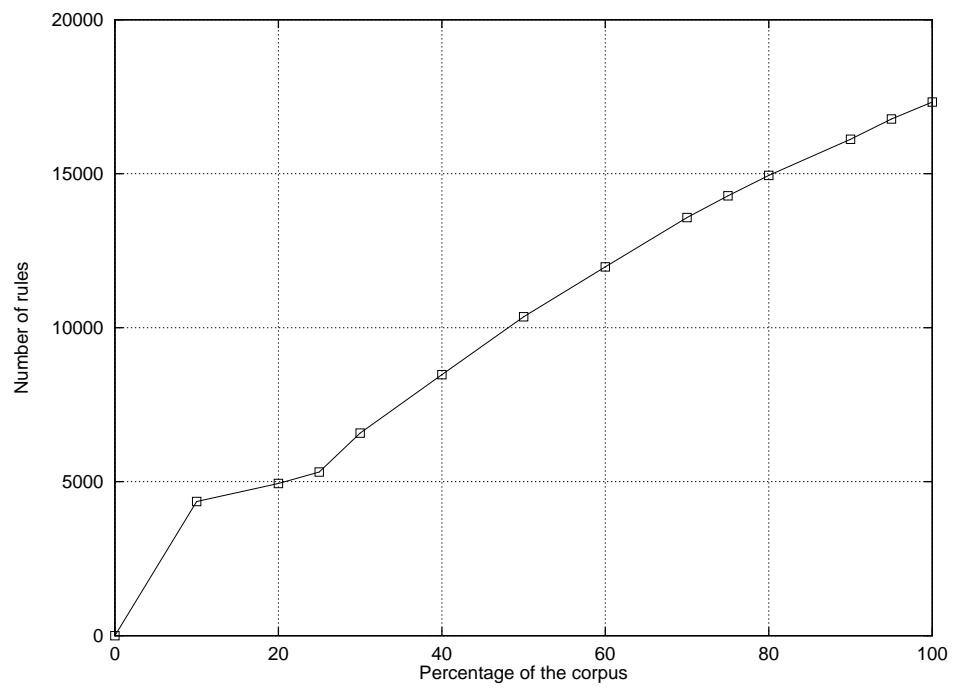


Figure 5.5: Rate of Growth of the Rule Size

Chapter 6

Compacting the Penn Treebank Grammar

As shown in Chapter 5, treebanks, such as the Penn Treebank (PTB), offer a simple approach to obtaining a broad coverage grammar: one can simply read the grammar off the parse trees in the treebank. While such a grammar is easy to obtain, the growth of the rule set with corpus size suggests that the derived grammar is far from complete and that much more treebanked text would be required to obtain a complete grammar, if one exists at some limit. However, we offer an alternative explanation of such growth in terms of the underspecification of structures within the treebank. This hypothesis is explored by applying an algorithm to *compact* the derived grammar by eliminating redundant rules – rules whose right hand sides can be parsed by other rules. The size of the resulting compacted grammar, which is significantly less than that of the full treebank grammar, is shown to approach a limit. However, such a compacted grammar does not yield very good accuracy figures. A version of the compaction algorithm taking rule probabilities into account is proposed, which is argued to be more linguistically motivated. This method can be used to give 12% reduction in grammar size without any change in parsing accuracy, and can produce a 28% reduction with some gain in recall, but a small loss in precision.

6.1 Motivation

The Penn Treebank (PTB) (Marcus et al., 1994) has been used for a rather simple approach to deriving large grammars automatically: one where the grammar rules are simply ‘read off’ the parse trees in the corpus, with each local subtree providing the left and right hand sides of a rule. Charniak (Charniak, 1996) reports precision and recall figures of around 80% for a parser employing such a grammar. In this paper we show that the huge size of such a treebank grammar (see below) can be reduced in size without appreciable loss in accuracy, and, in fact, an improvement in recall can be achieved.

Our approach can be generalized in terms of Data-Oriented Parsing (DOP) methods (see (Bonnema, Bod, and Scha, 1997)) with the tree depth of 1. However, the number of trees produced with a general DOP method is so large that Bonnema (Bonnema, Bod, and Scha, 1997) has to resort to restricting the tree depth, using a very domain-specific corpus such as ATIS or OVIS, and parsing very short sentences of average length 4.74 words. Our compaction algorithm can be easily extended for the use within the DOP framework but, because of the huge size of the derived grammar (see below), we chose to use the simplest PCFG framework for our experiments. Other nonlexicalised approaches include, for example, those by Mooney (Thompson, Mooney, and Tang, 1997) and Schabes et. al. (Schabes, Roth, and Osborne, 1993). Mooney parses very domain-specific sentences, whereas Schabes considers only binary-branching rules.

We are concerned with the nature of the rule set extracted, and how it can be improved, with regard both to linguistic criteria and processing efficiency. In what follows, we report the worrying observation that the growth of the rule set continues at a square root rate throughout processing of the entire treebank (suggesting, perhaps that the rule set is far from complete). Our results are similar to those reported in (Krotov, Gaizauskas, and Wilks, 1994). For the complete investigation of the grammar extracted from the Penn Treebank II see (Gaizauskas, 1995). We discuss an alternative possible source of this rule growth phenomenon, *partial bracketting*, and suggest that it can be alleviated by *compaction*, where rules that are redundant (in a sense to be defined) are eliminated from the grammar.

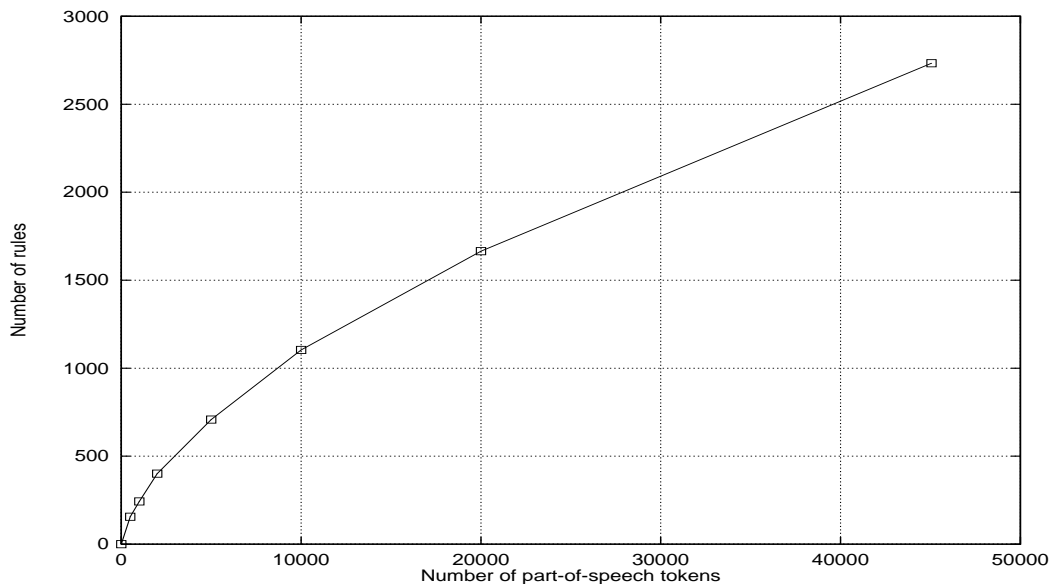


Figure 6.1: Rule Set Growth for Penn Treebank I

Our experiments on compacting a PTB treebank grammar resulted in two major findings: one, that the grammar can be compacted to about 8% of its original size, and the rule number growth of the compacted grammar stops at some point. The other is that a 12% reduction can be achieved with no loss in parsing accuracy, whereas a 28% reduction yields a gain in recall, but a loss in precision. Perhaps an optimum compaction proportion can be achieved if used together with an intelligent rule pruning algorithm.

6.2 Growth of the Rule Set

One could investigate whether there is a finite grammar that should account for any text within a class of related texts (i.e. a domain oriented sub-grammar of English). If there is, the number of extracted rules will approach a limit as more sentences are processed, i.e. as the rule number approaches the size of such an underlying and finite grammar.

As we reported in Krotov, Gaizauskas, and Wilks (1994), this expectation was not borne out by our experiments on PTB I. We extracted 2,700+ rules from 45,000+ part-of-speech tokens in the input. This rule number is not large in itself, since some contemporary speech

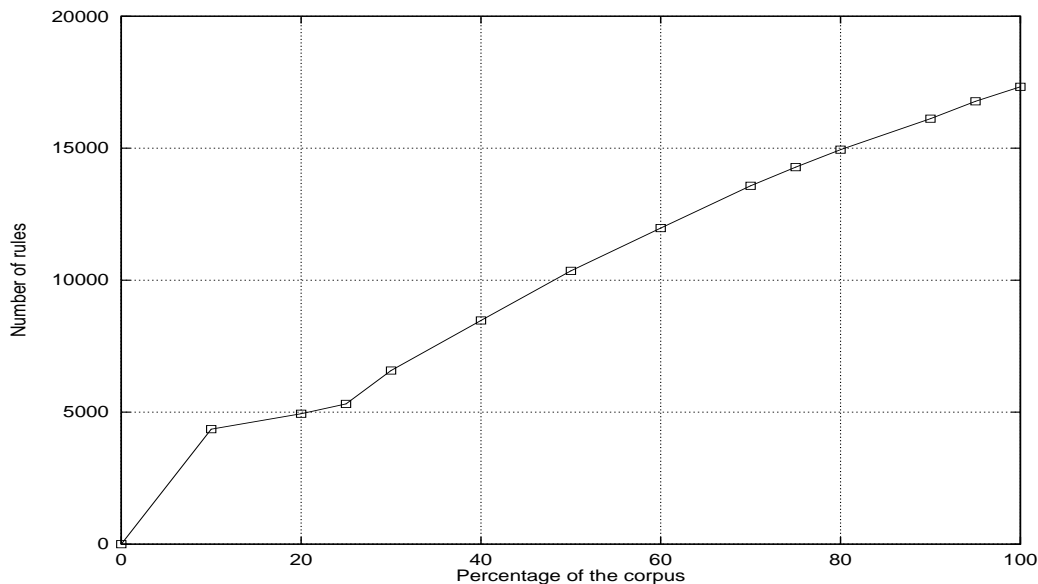


Figure 6.2: Rule Set Growth for Penn Treebank II

systems use much larger grammars. However, as shown in Figure 6.1, rule set growth proceeds with no sign that an asymptotic limit is approached toward the end of the corpus.

We had hoped that some approach to a limit would be seen using PTB II (Marcus et al., 1994), which is larger and more consistent for bracketting than PTB I. As shown in Figure 6.2, however, a similar pattern of growth is observed as for PTB I, where the rule number growth continues unabated even after more than 1 million part-of-speech tokens have been processed.

6.3 Rule Growth and Partial Bracketting

Why should the set of rules continue to grow in this way? Putting aside the possibility that natural languages do not have finite rule sets as Sampson (1987) suggests in his investigation of noun phrases extracted from LOB corpus, we can think of two possible answers. First, it may be that the full “underlying grammar” is much larger than the rule set that has so far been produced, requiring a much larger tree-banked corpus than is now available for its extraction. If this were true, then the outlook would be bleak for achieving near-

complete grammars from treebanks, given the resource demands of producing hand-parsed text. However, the radical incompleteness of grammar that this alternative implies seems incompatible with the promising parsing results that Charniak reports (Charniak, 1996). Also, Taylor, Grover, and Briscoe (1989) retest Sampson’s claim that there is no distinction between grammatical and non-grammatical constructions in the sample extracted from a treebanked corpus. This claim, if true, would preclude the possibility of creating a comprehensive grammar for automatic analysis of natural language. Taylor et. al. extracted a set of noun phrases from the same corpus as Sampson using a similar methodology. They corrected some of the noun phrases and analysed the entire sample using Alvey Natural Language Tools (ANLT) grammar, a wide-coverage hand-crafted grammar for English (Grover et al., 1987; Grover et al., 1989). Approximately 700 types of noun phrases were reduced to a much smaller number, between 36 and 54 of more general types, and the minimally modified ANLT grammar could parse 96.88% of the noun phrases in the test set correctly, showing that a hand-crafted grammar is capable of parsing a large portion of corpus of naturally occurring language.

Another answer is suggested by the presence in the extracted grammar of rules such as (6.1).¹ This rule is suspicious from a linguistic point of view, and we would expect that the text from which it has been extracted should more properly have been analysed using rules (6.2,6.3), i.e. as a coordination of two simpler NPs.

$$NP \rightarrow DT NN CC DT NN \quad (6.1)$$

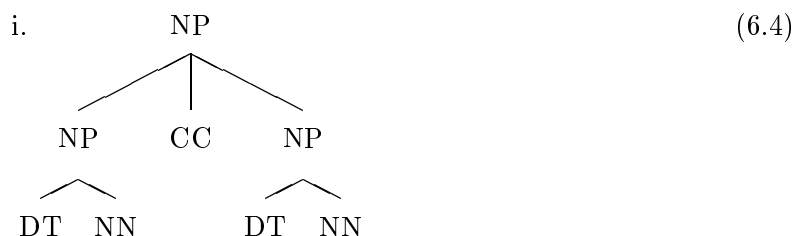
$$NP \rightarrow NP CC NP \quad (6.2)$$

$$NP \rightarrow DT NN \quad (6.3)$$

Our suspicion is that this example reflects a widespread phenomenon of *partial bracketting* within the PTB. Such partial bracketting will arise during the hand-parsing of texts, with (human) parsers adding brackets where they are confident that some string forms a given constituent, but leaving out many brackets where they are less confident of the constituent

¹See Table A.2 for the complete list of PTB POS tags.

structure of the text. This will mean that many rules extracted from the corpus will be ‘flatter’ than they should be, corresponding properly to what should be the result of using several grammar rules, showing only the top node and leaf nodes of some unspecified tree structure (where the ‘leaf nodes’ here are category symbols, which may be nonterminal). For the example above, a tree structure that should properly have been given as (6.4), has instead received only the partial analysis (6.5), from the flatter ‘partial-structure’ rule (6.1).



Even assuming some reasonable limit on the length for righthand side category sequences in a rule, the number of partial-structure rules that can be derived from even a relatively small ‘genuine’ underlying grammar is potentially enormous. Hence, on this alternative, continuing rule set growth does not imply incompleteness, i.e. it may be that nearly all the genuine rules are extracted in processing the corpus, but that the rule set continues to grow from the collection of more and more partial-structures that then become rules in our simple extraction process.

6.4 Grammar Compaction

The idea of partiality of structure in treebanks and their grammars suggests a route by which treebank grammars may be reduced in size, or *compacted* as we shall call it, by the elimination of partial-structure rules. A rule that may be eliminable as a partial-structure rule is one that can be ‘parsed’ (in the familiar sense of context-free parsing) using *other*

rules of the grammar. For example, the rule (6.1) can be parsed using the rules (6.2,6.3), as the structure (6.4) demonstrates. Note that, although a partial-structure rule should be parsable using other rules, it does not follow that every rule which is so parsable is a partial-structure rule that should be eliminated. There may be defensible rules which can be parsed. This is a topic to which we will return at the end of the paper (Sec. 6.7). For most of what follows, however, we take the simpler path of assuming that the parsability of a rule is not only necessary, but also sufficient, for its elimination.

Rules which can be parsed using other rules in the grammar are *redundant* in the sense that eliminating such a rule will *never* have the effect of making a sentence unparseable that could previously be parsed.² The idea of eliminating rules that are redundant in this sense has been independently developed (at roughly the same time as us) by (Shirai, Tokunaga, and Tanaka, 1995), and used in their work on the EDR corpus.

The algorithm we use for compacting a grammar is straightforward. A loop is followed whereby each rule R in the grammar is addressed in turn. If R can be parsed using other rules (which have not already been eliminated) then R is deleted (and the grammar *without* R is used for parsing further rules). Otherwise R is kept in the grammar. The rules that remain when all rules have been checked constitute the compacted grammar.

An interesting question is whether the result of compaction is independent of the order in which the rules are addressed. In general, this is not the case, as is shown by the following rules, of which (6.8) and (6.9) can each be used to parse the other, so that whichever is addressed first will be eliminated, whilst the other will remain.

$$B \rightarrow C \tag{6.6}$$

$$C \rightarrow B \tag{6.7}$$

$$A \rightarrow B B \tag{6.8}$$

$$A \rightarrow C C \tag{6.9}$$

²Thus, wherever a sentence has a parse P that employs the parsable rule R , it also has a further parse that is just like P except that any use of R is replaced by a more complex substructure, i.e. a parse of R .

Order-independence can be shown to hold for grammars that contain no *unary* or *epsilon* ('empty') rules, i.e. rules whose righthand sides have one or zero elements, as we prove in Section 6.5. The grammar that we have extracted from PTB II, and which is used in the compaction experiments reported in Section 6.6, is one that excludes such rules. Unary and sister rules were collapsed with the sister nodes, e.g. the structure (S (NP -NULL-) (VP VB (NP (QP ...))) .) will produce the following rules: S -> VP ., VP -> VB QP and QP -> ...³

6.5 Proof of Order Independence

In the previous section 6.4, we claimed that the result of applying the algorithm stated there to compact a grammar will be the same regardless of the order in which the rules of the grammar are addressed (i.e. parsed) *provided that* the grammar contains no *unary* or *epsilon* ('empty') rules, i.e. rules whose righthand sides have one or zero elements. The proof of this will be presented here.

We first require the lemma that for a grammar G that contains no unary or epsilon rules, a parse of a rule R under G cannot use R itself, unless the parse is *trivial*, i.e. consists *only* of R . This lemma holds because all rules are *branching* (i.e. have more than one RHS category). Hence, any parse that involves more than one rule use must have more leaf nodes than any single rule used within it, and so cannot be a parse of that rule.

We (non-standardly)⁴ define the *coverage* of a grammar to be the set of pairs (A, γ) (A a category, γ a string of categories) that can be parsed by the grammar, i.e. pairs (A, γ) such that the rules of the grammar can be used to build a tree whose topmost node is A and whose leaf nodes are γ . More formally:

$$\text{cov}(G) = \{(A, \lambda\mu\nu) \mid (A, \lambda\mu\nu) \in G \text{ or } ((A, \lambda B\nu) \in \text{cov}(G) \text{ and } (B, \mu) \in \text{cov}(G))\}$$

where $(A, \lambda\mu\nu) \in G$ indicates that grammar G contains a production $A \Rightarrow \lambda\mu\nu$

³See Chapter 5 for discussion.

⁴*coverage* as defined here is similar to the closure of the standard \Rightarrow relation in formal language theory. We are using nonterminal heads as well as strings, and allow nonterminals as well as terminals within strings.

Note that removing a redundant rule R from a grammar G (i.e. where R can be parsed within $G - \{R\}$) leaves the coverage unchanged, i.e. $cov(G) = cov(G - \{R\})$.

Now, consider two grammars G_1 and G_2 which are alternative intermediate stages (i.e. under different rule orders) in the course of compacting a grammar G that contains no unary or epsilon rules. Hence, $cov(G) = cov(G_1) = cov(G_2)$. Assume both grammars contain a rule R , which crucially is redundant for G_1 but not for G_2 (or the other way around: it suffices to swap indices), i.e. $cov(G_1) = cov(G_1 - \{R\})$ and R has a nontrivial parse within $G_1 - \{R\}$. Since $cov(G_1) = cov(G_2)$, the rules of $G_1 - \{R\}$ used to parse R must themselves be parsable (trivially *or* non-trivially) within G_2 . It follows (by combining those parses together) that R has a non-trivial parse within G_2 which (by the lemma) cannot contain R , and so R must be redundant for G_2 , contradicting our assumption. Hence, R must be redundant either for both G_1 and G_2 , or for neither, and so the order in which rules are addressed in compaction is irrelevant to the ultimate eliminability of any given rule, and hence for the overall result of compaction.

% of the corpus	Compacted Grammar Size	Uncompacted Grammar Size
10	1181	4358
20	1285	4940
30	1429	6578
40	1569	8476
50	1671	10357
60	1570	11973
70	1647	13577
80	1744	14945
90	1821	16120
100	1667	17329

Table 6.1: Results of the Grammar Compaction Experiment

6.6 Experiments

We conducted a number of compaction experiments for which we used two parsers: Stolcke’s BOOGIE (Stolcke, 1994b; Stolcke, 1995) and Sekine’s Apple Pie Parser (Sekine and Grishman, 1995). First, the complete grammar was parsed as described in Section 6.4. Results exceeded our expectations: the set of 17,529 rules reduced to only 1,667 rules, a better than 90% reduction.

To investigate in more detail how the compacted grammar grows, we conducted a third experiment involving a *staged* compaction of the grammar. Firstly, the corpus was split into 10% chunks (by number of files) and the rule sets extracted from each. The staged compaction proceeded as follows: the rule set of the first 10% was compacted, and then the rules for the next 10% added and the resulting set again compacted, and then the rules for the next 10% added, and so on. Results of this experiment are shown in Table 6.1 and Figure 6.3. At 50% of the corpus processed the compacted grammar size actually exceeds the level it reaches at 100%, and then the overall grammar size starts to go down as well as up. This reflects the fact that new rules are either redundant, or make “old” rules redundant, so that the compacted grammar size seems to approach a limit.

6.7 Retaining Linguistically Valid Rules

Even though parsable rules are redundant in the sense that has been defined above, it does not follow that they should always be removed. In particular, there are times where the flatter structure allowed by some rule may be more linguistically correct, rather than a simple case of partial bracketting. Consider, for example, the (linguistically plausible) rules (6.10,6.11,6.12). Rules (6.11) and (6.12) can be used to parse (6.10), but it should not be eliminated, as there are cases where the flatter structure it allows is more linguistically correct.

$$VP \rightarrow VB NP PP \tag{6.10}$$

$$VP \rightarrow VB NP \tag{6.11}$$

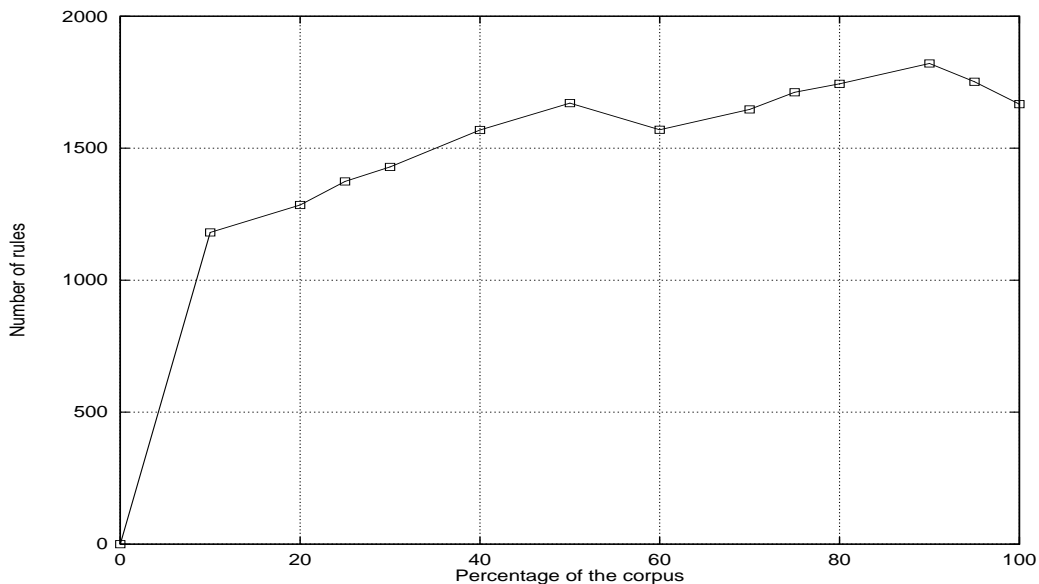


Figure 6.3: Compacted Grammar Size

$$NP \rightarrow NP PP \quad (6.12)$$

$$\begin{array}{cc}
 \text{i.} & \text{ii.} \\
 \begin{array}{c}
 \text{VP} \\
 \swarrow \quad \searrow \\
 \text{VB} \quad \text{NP} \\
 \swarrow \quad \searrow \\
 \text{NP} \quad \text{PP}
 \end{array} &
 \begin{array}{c}
 \text{VP} \\
 \swarrow \quad | \quad \searrow \\
 \text{VB} \quad \text{NP} \quad \text{PP}
 \end{array}
 \end{array} \quad (6.13)$$

We believe that a solution to this problem can be found by exploiting the data provided by the corpus. Frequency of occurrence data for rules which have been collected from the corpus and used to assign probabilities to rules, and hence to the structures they allow, so as to produce a *probabilistic* context-free grammar for the rules. Where a parsable rule is correct rather than merely partially bracketted, we then expect this fact to be reflected in rule and parse probabilities (reflecting the occurrence data of the corpus), which can be used to decide when a rule that *may* be eliminated *should* be eliminated. In particular, a rule should be eliminated only when the more complex structure allowed by other rules is more probable than the simpler structure that the rule itself allows. A number of alternatives arise for deciding the elimination of a rule in the case where it has more than one parse, as above.

Should the rule's own probability be measured against the maximum parse probability, or the sum of all (or n , for some n) parse probabilities? This is again an experimental issue, and similar options arise concerning the reassignment of an eliminated rule's probability value to the rules used for its own parse(s).

In the case where the grammar is to be used with a statistical parser that returns only the most probable parse, the elimination of rules on the above basis is justifiable independently of linguistic considerations. Thus, where any rule has a parse that is more probable than the rule itself, it follows that any parse of a sentence that employs that rule will not be the most probable parse that the statistical parser can construct and so eliminating the rule will be safe in terms of the parser's output, as well as being beneficial for efficiency.

The grammar compacting algorithm employing probabilistic compaction just described was implemented, but only a 10% compaction ratio was achieved. We then ranked the rules that are in the linguistically compacted set but not in the fully compacted set (about 80% of the grammar), based on the parse probability vs. rule probability ratio. We then took $x\%$ of the rules with lowest ranking, added them to the fully compacted set and evaluated the resulting grammar with different values assigned to $x\%$ ($x = 0$ corresponds to the fully compacted grammar, $x = 100$ to the linguistically compacted grammar).

The preliminary results of our experiments are presented in Table 6.2. They are derived using standard Parseval metrics of precision and recall (Black, 1991). That is, precision is the percentage of brackets in the test bracketing that also occur in the treebank bracketing. Recall is the percentage of brackets in the treebank bracketing that also occur in the test bracketing.

As one can see, the fully compacted grammar yields poor recall and precision figures. This can be because collapsing of the rules often produces too much substructure (hence lower precision figures) and also because many longer rules in fact encode valid linguistic information.

However, linguistic reduction achieves a 12% reduction without any loss in accuracy, and 28% reduction even yields higher recall. Our figures are higher than those reported

	Extracted	Compacted	Linguistically compacted grammars			
			70	80	90	100
Recall	88.41%	61.88%	88.03%	89.21%	88.56%	88.11%
Precision	87.41%	56.50%	82.47%	85.54%	86.71%	87.75%
Grammar size	16973	1417	10884	12228	13573	14917
reduction (% of full)	0%	92%	36%	28%	20%	12%

Table 6.2: Preliminary results on parsing with the linguistically compacted grammar

by Charniak due to the fact that we used shorter sentences for our samples (maximum length 20 tokens, average length 13.9 tokens). For a similar sample (with the sentence length restricted to 20 tokens), Charniak reports 87.3% precision and 84.9% recall. We also removed any rules containing **X** or **FRAG** from the extracted grammar, which resulted in a slightly better accuracy and a smaller grammar compared to the one reported in Section 6.4. **X** and **FRAG** were assigned to the tree constituents where annotators were unsure which label is appropriate, if any.

6.8 Conclusions

We see the principal results of PTB compaction experiments to be the following:

- the result showing continued square-root growth in the rule set extracted from the PTB II;
- the analysis of the source of this continued growth in terms of *partial bracketting* and the justification this provides for compaction via rule-parsing;
- the result that the compacted rule set *does* approach a limit at some point during staged rule extraction and compaction, after a sufficient amount of input has been processed;
- that, though the fully compacted grammar produces lower Parseval figures than the extracted grammar, a 12% reduction (without loss) can still be achieved by using

linguistic compaction, and 28% reduction yields a gain in recall, but a loss in precision.

The latter result in particular provides further support for the possible future utility of the compaction algorithm. Our method is similar to that used by Shirai (Shirai, Tokunaga, and Tanaka, 1995), but the principal differences are as follows. First, their algorithm does not employ full context-free parsing in determining the redundancy of rules, considering instead only direct composition of the rules (so that only parses of depth 2 are addressed). We have proved that the result of compaction is independent of the order in which the rules in the grammar are parsed in those cases involving 'mutual parsability' (discussed in Section 6.4), but Shirai's algorithm will eliminate both rules so that coverage is lost. Secondly, it is not clear that compaction will work in the same way for English as it does for Japanese. Our findings show that the linguistically compacted grammar can perform very well on English, and our results compare favourably to the 85% recall and 75% precision figures obtained by Shirai.

Chapter 7

Parsing and Evaluation

In Chapters 5 and 6 we have shown how a context-free grammar can be automatically extracted from the Penn Treebank. To evaluate the extracted grammar, we use it in a parsing system. Parsing with the treebank grammar places considerable demands on a parser due to the large number of rules in the grammar. We have used three parsers at different stages of our research, one described in Stolcke (1994b), one described in Sekine (1998), and an in-house parser developed by Mark Hepple. Section 7.1 describes our choice of parsers and important implementation details.

Parser evaluation is described in Section 7.1. Until recently, the PARSEVAL program described in Black (1991) was commonly used for evaluating parser output. PARSEVAL produces precision, recall and average crossing bracket figures which are commonly used to compare accuracy of different parsers. However, PARSEVAL ignores nonterminal labels while comparing parses. Sekine and Grishman (1997) developed the bracketing evaluation software, `evalb`, which extends the set of PARSEVAL metrics. `evalb` calculates labelled precision and recall as well as some other figures. State-of-the-art parsing systems (e.g., ones described in Collins (1996) and Ratnaparkhi (1997)) are now evaluated using labelled precision and recall. We report both labelled and unlabelled precision and recall in Chapter 8. These are obtained using `evalb`.

As well as evaluation measures we need to generate test data to supply as the input to the

parser. Syntactically annotated data from the Penn Treebank is used as a ‘gold standard’ against which the parser output is evaluated. And, of course, the test and training data have to be disjoint. Generating test data is described in Section 7.3.

7.1 Parsing System

Parsing with a grammar extracted from the Penn Treebank places considerable demands on the parser being used. This is because of the sheer size of the grammar (17,000+ rules), and the unpredictable nature of the extracted rule set. Younger (1967) presents a CFG parsing algorithm achieving q^3n^3 time complexity where n is the number of tokens in the string and q is the number of nonterminals in the grammar. Because of the computational complexity of context-free parsing and ambiguity issues most CFG-based systems use a smaller number of rules. When parsing with a large number of rules time and memory requirements are very important. Also, we cannot make any assumptions on the extracted grammar since the annotated sentences in the PTB can potentially contain any kind of rules. In probabilistic parsing, epsilon- and unary production rules may cause infinite loops. As shown in Stolcke (1995), special processing is required for such rules. Few parsers meet the above requirements. We used Stolcke’s and Sekine parsers until Mark Hepple developed an in-house parser.

Stolcke’s BOOGIE (Stolcke, 1994b) is a LISP-based system which allows the user to experiment with different kinds of probabilistic models including PCFGs and Hidden Markov Models, as well as various parsing algorithms. We used BOOGIE initially to observe a large compaction rate of the grammar extracted from the PTB. However, the LISP-based program was too slow to carry out the entire experiment and it also required too much memory.

Sekine’s Apple Pie chart parser (Sekine and Grishman, 1995) uses a very large grammar (32,000+ rules), and is implemented in C to take full advantage of the processor speed and efficient memory usage. It incorporates several techniques to store the grammar and the parse chart more efficiently, so that less time and memory is used. We used the Apple Pie

parser to compact the entire PTB grammar from 17,000+ rules to 1,667, and it only took about 30 minutes on a Sun SPARC 4. We also used the Apple Pie parser for our early experiments in evaluating the probabilistic compaction method, but then started running out of memory while parsing longer sentences. We also wanted to have better control over generating the parse chart and selecting the best parse.

Mark Hepple developed a chart parser in Prolog, which we used for generating the set of parse charts for grammar pre-compaction (see Section 8.2.2). His parser proceeds left-to-right on the input sentence, and uses a technique to eliminate some unnecessary parse edges. A considerable saving in the amount of memory required is achieved this way. Hepple then re-implemented this parser in C to achieve faster speed and more efficient memory usage. I adapted this parser for producing results presented in Chapter 8.

In all these parsers, the most probable sentence structure is considered to be the best one returned by the parser. The Viterbi algorithm is used to find the best sentence parse.¹ The Viterbi algorithm (Viterbi, 1967; Forney, 1973) was originally proposed for finding the best state sequence in a Hidden Markov Model. However, it can be applied to finding a best parse in chart parsing with a context-free grammar. I will now briefly describe the Viterbi algorithm for chart parsing.

First, each edge is augmented with a probability (which is the probability of its best derivation): $(i, j) : A[\gamma]$, where $(i, j) : A$ is an edge in the chart, defined similarly in Chapter 2, and γ is its associated probability. Then, the parser proceeds bottom-up, calculating probabilities of each new edge added to the chart (as the product of two edge probabilities and the rule probability). The new edge is added to the chart only if it is either a new edge or its probability is higher than of the one that already exists. In our notation, say an edge $(i, j) : A[\gamma_1]$ is in the chart, and an edge $(i, j) : A[\gamma_2]$ has just been produced by combining two other edges in the grammar. If $\gamma_2 > \gamma_1$, the edge $(i, j) : A[\gamma_2]$ replaces the old edge in the chart. Otherwise the chart remains unchanged. For any new edge, there is no other edge $(i, j) : A[.]$.

¹BOOGIE gives many options to the user, and Viterbi best parse is the one we used in our experiments.

The key idea here is that it is not necessary to store *all* derivations of the sentence to find the best one. It is sufficient to keep the best derivation for each edge in the chart (or each state in a Hidden Markov Model). The number of all possible parse derivations can grow exponentially, whereas the number of edges in the chart is limited to $\frac{n \times (n-1)}{2} \times N$, where n is the length of the sentence, and N is the number of nonterminals. One can prove by recursion that such process will eventually return the best parse of the sentence. However, if an arbitrary number of best parses (e.g. 5 best parses) is needed, full search of the parse tree will be required (or a form of beam search).

7.2 Parser Evaluation

Parser output is usually evaluated by comparing it against the ‘gold’ analysis of the set of sentences being parsed. The ‘gold’ analysis is typically human-annotated, and assumed to be correct. Treebanked corpora such as Penn Treebank provide an excellent source of test material.

We used two programs for evaluating our parsing system: PARSEVAL (Black, 1991) and `evalb` (Sekine and Grishman, 1997). Until recently, PARSEVAL was standardly used to evaluate performance of parsing systems. `evalb` is a newer program which produces a more extensive evaluation figures including, most importantly, labelled precision and recall. (Collins, 1996) used `evalb` to evaluate performance of his parser. We used PARSEVAL initially and then switched to `evalb` because it produced more extensive output, was easier to integrate, and ran much faster.

Both programs use a crossing brackets measure as the basis for evaluation. The crossing brackets measure is based on the fact that crossing sentence constituents, or brackets represent a significant mismatch between the ‘gold’ parse and the one produced by the system. Two brackets cross when they have elements in common and neither of the brackets is fully contained inside the other. For example, in the following two structures

Treebank analysis:

((The new rate) will (be payable (Feb. 15)))

and

Test analysis:

((The new rate) (will be) (payable (Feb. 15)))

brackets (be payable Feb. 15) and (will be) cross, because be is in both brackets, yet neither of the brackets is fully contained inside the other. On the other hand, (payable Feb. 15), though it is not a part of the treebank analysis, is fully contained in (be payable Feb. 15) and thus is not a crossing bracket.

The number of crossing brackets alone does not take into account the complexity of the sentence. One crossing bracket in the test analysis of a short sentence such as **cats eat fish** often means that the analysis is completely incorrect (see below). However, one crossing bracket in the analysis of a longer sentence such as the one presented above leaves room for parts of the sentence which are analysed correctly.

Correct analysis:

(Cats (eat fish))

and

Test analysis:

((Cats eat) fish)

PARSEVAL counts the number of crossing brackets for each pair of treebank and test analyses, and also calculates the precision and recall figures, which are defined as follows:

$$\text{Precision} = \frac{\text{number of test constituents exactly matching the treebank constituents}}{\text{total number of test constituents}}$$

$$\text{Recall} = \frac{\text{number of treebank constituents exactly matching the test constituents}}{\text{total number of treebank constituents}}$$

Precision and recall measures account for the complexity of the sentence, and for non-matching brackets that do not cross. For the first pair of analyses, the number of brackets

common to both analyses is 3, the number of brackets in the treebank analysis is 4, and the number of brackets in the test analysis is 5. Consequently, *recall* is $\frac{3}{4} = 75\%$, *precision* is $\frac{3}{5} = 60\%$, and *accuracy* (see below) is $\frac{4}{5} = 80\%$. There is only one crossing bracket. For the short sentence, precision, recall and accuracy are all 50% if the outermost bracket is taken into account, 0% if it is not.

Other statistics can be collected, such as accuracy (see (Charniak, 1996)), which counts the number of non-crossing brackets in the test sample:

$$\text{Accuracy} = \frac{\text{number of test constituents not crossing with the treebank constituents}}{\text{total number of test constituents}}$$

Apart from crossing brackets, it may be worth considering the ‘ideal’ evaluation criterion which looks for a perfect match of the parse structures. According to (Magerman, 1994), however, the internal consistency of the Penn Treebank (the first release) was estimated to be 23%. In other words, if two independent taggers marked the same sentence, the chance both analyses were the same is 23%. Naturally, if a parser is tested on the data from the Penn Treebank, one should not expect better performance on the complete match measure than 23%.

PARSEVAL program takes a set of test and standard bracketings and, for each sentence, calculates recall, precision and the number of crossing brackets. This data is then averaged over the entire sample. PARSEVAL also counts the number of sentences with no crossing brackets.

`evalb` calculates all measures PARSEVAL does, and also labelled precision and recall, number of sentences with 2 or fewer crossing brackets, number of sentences exactly matching the ‘gold’ structure, and tagging accuracy. `evalb` gives the user the option to evaluate part-of-speech tagging of the sentence as well as the parse structure.

Both PARSEVAL and `evalb` modify test and ‘gold’ parses before carrying out the calculations. PARSEVAL erases all word-external punctuation, null categories, instances of: auxiliaries, “not”, pre-infinitival “to” and possessive endings such as ‘s and ‘. Then it recursively eliminates all brackets enclosing a single constituent or nothing.

PARSEVAL also modifies the test parse if certain constructions are different from

Measure	PARSEVAL	evalb	Charniak
Recall	◇	◇	◇
Precision	◇	◇	◇
Recall(labelled)		◇	
Precision(labelled)		◇	
Accuracy			◇
No crossing	◇	◇	
≤ 2 crossing		◇	
Average crossing	◇	◇	
Exact match		◇	
Tagging accuracy		◇	

Table 7.1: Parser Evaluation Measures

the pre-defined standard. These constructions include extraposition, modification of noun phrases by post-head phrases such as PP, and certain sequences of prepositions. See Black (1991) for a detailed explanation and examples.

`evalb` removes all tree nodes whose top nonterminal is in a `DELETE_LABEL` list specified in the set of parameters. `DELETE_LABEL` usually includes the label `-NONE-` indicating a null constituent in the PTB. All nodes which contain nothing are also recursively removed. Then, `evalb` removes all functional tags attached to nonterminals (the ones prefixed with `-` or `=`). For example, `NP-SBJ` becomes `NP`.

Table 7.1 shows all measures produced by `PARSEVAL`, `evalb`, and those used in Charniak (1996).

Recently, Carroll, Briscoe, and Sanfilippo (1998) argue that constituent-based evaluation as in `PARSEVAL` and `evalb` does not give reliable results for non-treebank based parsers, i.e. those using alternatively created grammars. A technique for measuring parser accuracy using grammatical relations is proposed, and some preliminary results using this new technique are shown.

7.3 Test Data

I implemented a program `gen_sample` which generates a sample file from the Penn Treebank in PARSEVAL and `evalb` formats and the format suitable as an input to a parser. It takes the set of PTB file numbers as its input and returns the set of sample files. `gen_sample` was used to generate samples for our experiments described in Chapter 8.

For different evaluation procedures, we may need to filter the sample set. For example, one may want to know how a parser performs on sentences of different lengths. `gen_sample` will allow you to specify both a minimum and maximum sentence length in the sample.

The Penn Treebank also contains syntactic tags such as **X** and **FRAG** which do not carry much linguistic information. `gen_sample` offers an option to remove sentences containing **X** and **FRAG** from the sample. Also, not all analyses in the Penn Treebank have **S** as their head constituents. One may need to filter the sample so that it contains only analyses with **S** at the top.

Chapter 8

Experiments

In Chapter 5 we showed that a context-free grammar can be extracted from the parsed portion of the Penn Treebank. The size of this grammar turned out to be very large and growing as more sentences were processed by the extraction algorithm. Then, in Chapter 6 we demonstrated how this large grammar can be compacted to less than one-tenth of its original size. Also, the size of the compacted grammar seemed to stop growing. We evaluated this small compacted grammar in a Viterbi parser, but it does not yield very good parsing accuracy.

We then suggested the probabilistic version of the compaction algorithm. Preliminary evaluation results show that the extracted grammar *can* be compacted without loss in parsing accuracy.

In this chapter we extend results presented earlier by using a larger test set and exploring a number of options for grammar generation and filtering the test data. First, Section 8.1 summarises results obtained earlier. Section 8.2 discusses grammar generation. We show how the grammar compaction algorithm can be implemented more efficiently by first parsing *all* rules and storing the resulting parse trees (an idea that was suggested by Mark Hepple). These stored trees are then used to compact any subset of the fully extracted grammar. We then describe linguistic and ranked compaction algorithms, and describe the test data used in experiments

Section 8.3 presents complete evaluation results. Finally, Section 8.4 compares our results with other related research.

8.1 Summary of Earlier Results

The findings of our earlier experiments with the Penn Treebank grammar can be summarised as follows:

- a large number of extracted rules and an ongoing growth of this number which does not seem to approach a limit;
- compacting the large rule set to less than one-tenth of its original size without losing coverage;
- poor parsing accuracy of the fully compacted grammar;
- augmentation of the compaction method taking into account rule probabilities. There is no or little loss in parsing accuracy, but a much smaller compaction rate is achieved.

Rule growth is shown in Figure 8.1 together with the results of compaction. A significant rate of compaction can be easily seen in this figure. However, the fully compacted grammar did not perform well in a parser, so that the compaction method was enhanced using rule probabilities. The *probabilistic* compaction yields practically no loss in parsing accuracy, but the compaction ratio is much lower. Another variation of probabilistic compaction uses *ranking* of rules, and can produce any grammar varying in size from the fully compacted grammar to the probabilistically compacted grammar. The parameter x between 0 and 100 is used to distinguish between such grammars, with 0 corresponding to the fully compacted grammar, and 100 to the probabilistically compacted grammar. Using ranking helps to reduce the size of the extracted grammar a little more: with $x = 70$, there is a small gain in recall and a small loss in precision (see Figure 8.2).

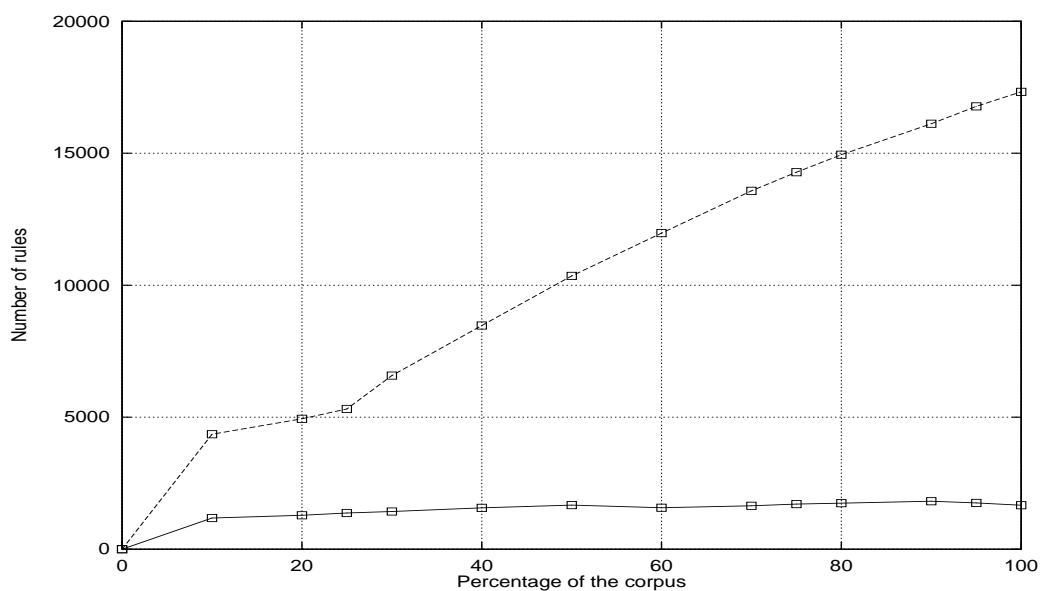


Figure 8.1: Grammar size growth versus compacted grammar size

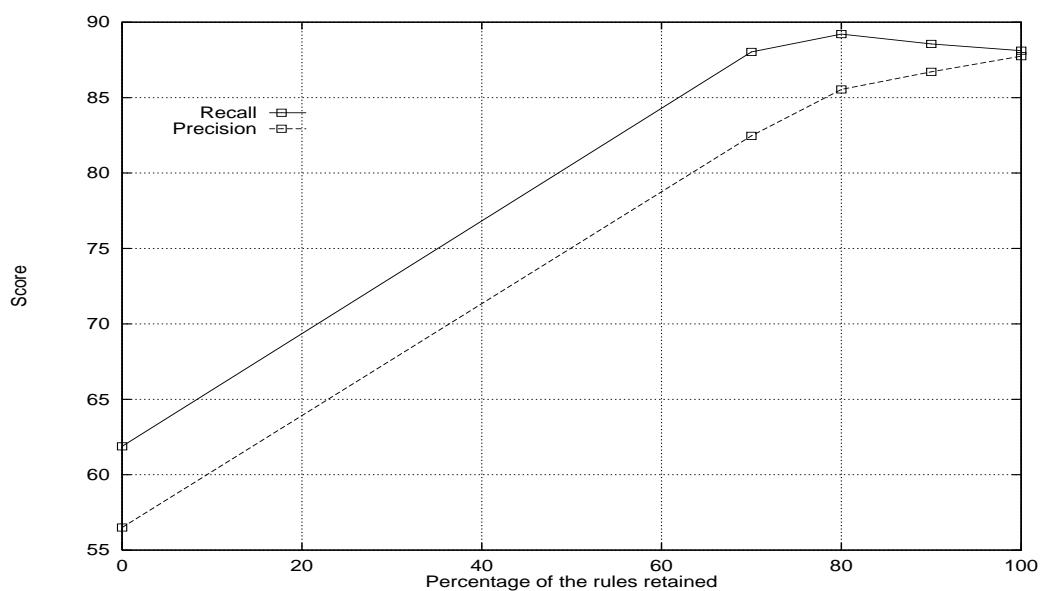


Figure 8.2: Parsing accuracy using probabilistically compacted grammars (preliminary results)

8.2 Grammar and Sample Generation

8.2.1 Overview

Evaluation of the treebank grammar compaction method consists of the following stages. First, a grammar is extracted from the Penn Treebank (or its subset) using scripts described in Gaizauskas (1995). Then, the extracted grammar is filtered, e.g., using thresholding by rule frequency. The filtered grammar is then compacted by parsing each rule with other rules. We also collect rule parse probabilities so that the probabilistically compacted grammars can be generated (with or without ranking). Rule parsing can be speeded up considerably by pre-parsing the entire extracted rule set and storing all parse charts. These charts can be used to generate the compacted version of any subset of the entire grammar without actually parsing the rules.

The compacted grammar is then used in a probabilistic parser (Sekine's or Hepple's) which produces the bracketed sentence structure. The bracketing is then evaluated using `Parseval` or `evalb`. I developed a simple program which generates test samples using any portion of the Penn Treebank (specified by the user) and filters them if necessary.

Figure 8.3 illustrates the evaluation process, and Appendix B lists programs used in evaluation.

8.2.2 Storing Rule Parse Charts

Compacting the Penn Treebank is a rather lengthy process: it involves parsing 17,000+ times with the grammar containing 17,000+ rules. For experiments, we need to filter the grammar (e.g., by thresholding or removing rules containing `X` or `FRAG`) and then compact the smaller version of this grammar. Reparsing the entire grammar every time a new experiment is run seems wasteful. Mark Hepple suggested that the entire parse chart for all rules can be stored while parsing the grammar extracted from the entire Penn Treebank. This collection of all parse charts contains all information needed to produce the compacted version of any subsets of the Penn Treebank full grammar. Such a collection was produced

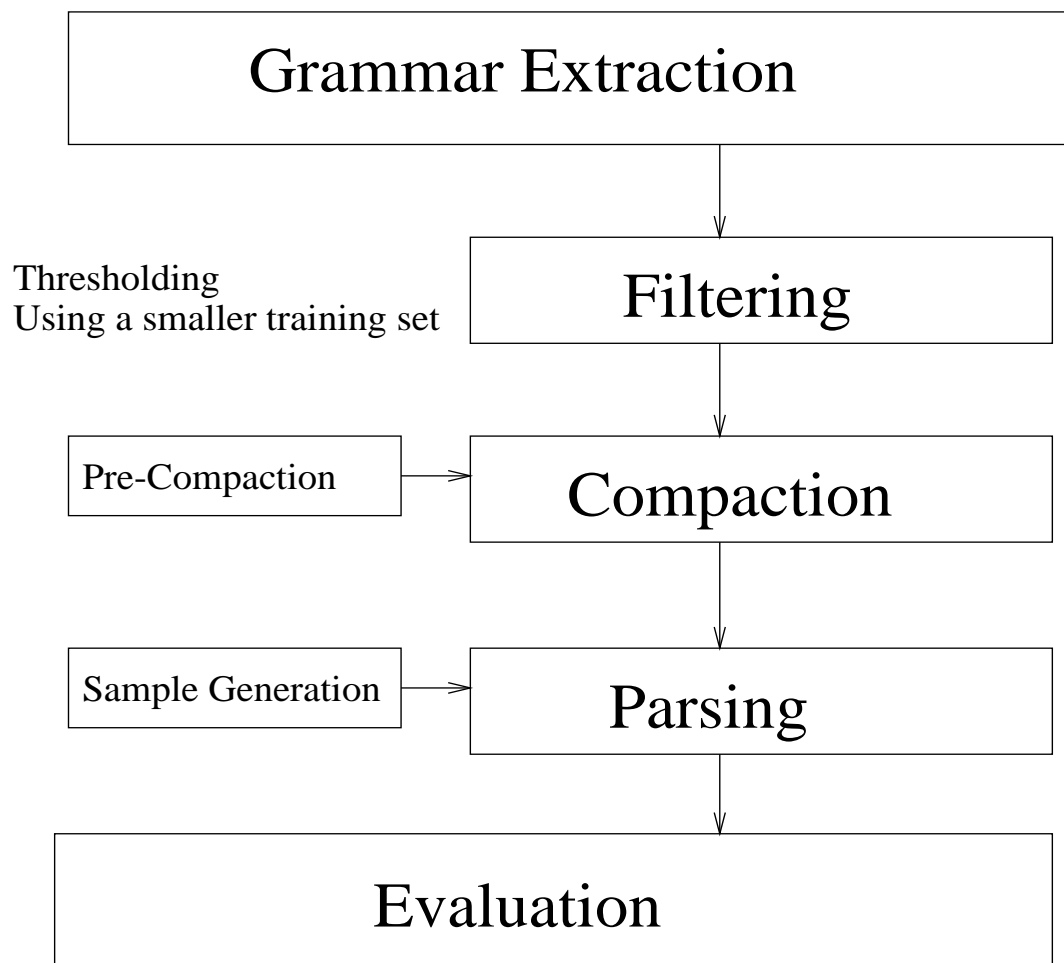


Figure 8.3: Evaluation of the grammar compaction method

with Mark Hepple's Prolog parser, taking about 400 megabytes of space. Each rule is identified with a unique number used in the name of an individual parse chart (for this rule), and within other parse charts. For example, the file `rule.3444` contains the parse chart for rule 3444, and 3444 is used in other parse charts in any place rule 3444 is used.

To compact a smaller version of the Penn Treebank grammar we use the following algorithm. First, we collect rule numbers for all rules *removed* from the full grammar (e.g., ones that do not fit the thresholding criteria). Then, we go through the collection of the parse charts removing all edges containing any of these numbers (or removed rules) in their derivations. As a result of this process, some rules may lose all of their edges. We then remove such rules as well. For other rules, we use only remaining edges for producing fully compacted, probabilistically compacted grammars and ranking. I implemented a Perl script `makegram` which reduces the full PTB grammar set and carries out full compaction and linguistic compaction as just described.

8.2.3 Probabilistic Compaction

In Section 6, we outlined several options for compacting the Penn Treebank grammar. In this section, we will show which options were chosen for the final batch of experiments and describe the compaction process in detail. The extracted grammar is compacted by parsing each rule with other rules as described earlier (using pre-stored parse charts). During this compaction we store the Viterbi probability of each redundant rule as well as its original probability obtained using counts. In case a rule is non-parsable (i.e., it is in the fully compacted grammar), its Viterbi probability is 0. We then calculate the ratio of these two probabilities (Viterbi probability divided by the original rule probability), and use this 'compaction' ratio to order (or rank) the rules. For all rules in the fully compacted grammar the compaction ratio is 0 (because Viterbi parse probability is 0). For all rules in the probabilistically compacted grammar the ratio is greater than 0 but less than 1 (because Viterbi probability is less than the rule probability). And for all other rules the compaction ratio is greater or equal to 1. Figure 8.4 illustrates ranking by compaction ratio.

The fully compacted grammar is small, but not very good for parsing, and the probabilistically compacted grammar yields good parsing accuracy, but is still quite large. By using ranking, we are trying to derive a grammar which is the trade-off between two extremes: it is smaller than the probabilistically compacted grammar and still produces good parsing accuracy. From all rules having compaction ratio between 0 and 1 we pick $x\%$ with the lowest ratio, append this set to the fully compacted grammar, and parse with the resulting grammar. See Figure 8.4 for illustration. Hopefully, varying $x\%$ will produce a smaller grammar yielding good parsing accuracy.

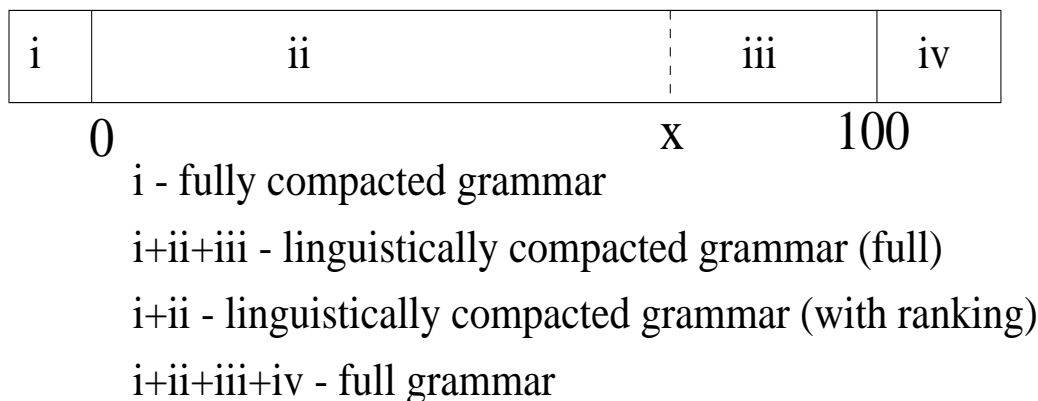


Figure 8.4: Ranking rules by compaction ratio

8.2.4 Test Samples

For preliminary experiments, small test samples were used (500 sentences). This was due to the limitations of the parser. We then used a more efficient parser.

For the set of experiments reported in this chapter, the test set consists of all text in directory 23 (of the WSJ portion of the PTB), filtered if necessary. By default, we filter by length (removing all sentences longer than 40 tokens), or, optionally, by removing sentences containing **X** and **FRAG** elements (see the next section). The training set consists of text in directories 02-21. (Collins, 1996) derives his test and training data from the same directories of the Penn Treebank. The size of the test sample is 2,245 sentences unfiltered,

2,197 sentences filtered (by removing X and FRAG).

8.3 Results

8.3.1 Thresholding

First, we wanted to see what results can be achieved with simple thresholding. Charniak (1996) reports that removing rules that only occur once does not significantly affect parsing accuracy, but reduces the grammar to less than a half of its original size. In Bod and Scha (1996) 1-occurring rules are called ‘hapaxes’ and removing hapaxes of depth 1 does not significantly affect parsing accuracies.

We observed exactly the same pattern in our experiments (see Table 8.1). The size of the grammar reduced from 15,420 rules to 6,517 rules with little change in parsing accuracy. We used the 1-thresholded grammar as a starting point for the rest of the experiments presented in this chapter.

We also wanted to see what results can be achieved by using other thresholds. We eliminate the rules rather crudely, by using only the rule count as the threshold parameter, disregarding the rule probability. Precision and recall for the grammars using thresholds of up to 100 is shown in Table 8.3.1. We also collected the coverage figures, i.e. how many sentences can still be parsed with the thresholded grammar. Unlike compaction, thresholding does not preserve grammar coverage, and the greater is the threshold, the more sentences cannot be parsed with the thresholded grammar. But the parsing accuracy of the thresholded grammar does not change significantly, even with large thresholds. However, the coverage deteriorates considerably. Only 1,587 out of 2,245 sentences can be parsed with 100-thresholded grammar, indicating 29% loss in coverage.

8.3.2 Probabilistic Compaction

Using the 1-thresholded grammar as a starting point, we first compacted it fully. The fully compacted grammar was much smaller than the original grammar, but produced poor

	Size	Reduction	Labelled		Unlabelled	
			Recall	Precision	Recall	Precision
Entire grammar	15,420		74.08%	77.43%	77.19%	80.67%
Thresholded grammar	6,517	58%	74.42%	76.99%	77.52%	80.20%

Table 8.1: Effect of simple thresholding

Threshold	Size	Reduction	Recall	Precision	Sent. parsed	Coverage Loss
0	15,420	0%	74.08%	77.43%	2,244	0.04%
1	6,517	58%	74.42%	76.99%	2,241	0.2%
2	4,508	71%	74.27%	76.36%	2,240	0.2%
3	3,675	76%	74.02%	75.87%	2,229	0.7%
5	2,736	82%	73.94%	75.29%	2,220	1.1%
10	1,776	88%	73.60%	74.02%	2,191	2.4%
15	1,418	91%	73.55%	73.51%	2,157	3.9%
20	1,209	92%	72.82%	72.06%	2,139	4.7%
30	933	94%	72.70%	71.55%	2,059	8.7%
50	660	96%	73.40%	71.22%	1,875	16.5%
70	540	96%	73.86%	71.66%	1,665	25.8%
100	443	97%	73.63%	70.67%	1,587	29.3%

Table 8.2: Effect of Varying Thresholds

	Size	Reduction	Labelled		Unlabelled	
			Recall	Precision	Recall	Precision
Uncompacted	6,517	0%	74.42%	76.99%	77.52%	80.20%
Fully compacted	1,218	81%	32.44%	19.34%	44.84%	26.73%
Probabilistically compacted grammars:						
20%	2,158	67%	59.81%	45.57%	65.11%	49.60%
40%	3,098	52%	68.73%	57.79%	72.44%	60.90%
60%	4,037	38%	73.51%	68.06%	77.01%	71.30%
70%	4,507	31%	75.07%	71.58%	78.38%	74.74%
80%	4,977	24%	75.72%	73.51%	79.03%	76.72%
90%	5,447	16%	75.19%	75.56%	78.38%	78.77%
100%	5,917	9%	74.54%	77.02%	77.65%	80.23%

Table 8.3: Evaluating Probabilistic Compaction

parsing accuracy. We then probabilistically compacted the 1-thresholded grammar with $x = 20, 40, 60, 70, 80, 90, 100$. In evaluating the resulting grammars, a pattern similar to the one observed in our earlier experiments was observed. With $x = 100$, there is no significant accuracy change, but the compaction ratio, is small. With $x = 70$, the compaction ratio is greater, recall of the compacted grammar is greater than that of the original grammar, but precision is lower. The entire set of results is presented in Table 8.3.

8.3.3 Effect of removing unknown linguistic constituents

In this experiment, we wanted to see if removing **X** and **FRAG** elements from the grammar and the test sample will affect parsing accuracy of the compacted grammar. **X** and **FRAG** non-terminal elements in the Penn Treebank denote parts of the sentence where the human annotators were uncertain as to what non-terminal symbol should be assigned. It is not clear whether it is worth keeping such symbols in the grammar, and in the sample. Such filtering yields a slightly smaller sample (2,197 sentences as opposed to 2,245) and slightly smaller 1-thresholded grammar (6,430 rules as opposed to 6,517). Probabilistic compaction was carried out, as in previous section and the obtained results are very similar to those with the uncompacted grammar (see Table 8.4).

	Size	Reduction	Labelled		Unlabelled	
			Recall	Precision	Recall	Precision
Uncompacted	6,430		74.74%	77.34%	77.72%	80.43%
Fully compacted	1,169	82%	33.18%	19.80%	44.84%	26.76%
Probabilistically compacted grammars:						
60%	3,966	38%	73.63%	68.17%	77.00%	71.29%
70%	4,432	31%	75.19%	71.70%	78.40%	74.75%
80%	4,899	24%	76.23%	74.01%	79.43%	77.12%
90%	5,365	17%	75.42%	75.81%	78.49%	78.89%
100%	5,831	9%	74.86%	77.37%	77.84%	80.45%

Table 8.4: Effect of removing X and FRAG

8.3.4 Effect of selecting the top sentence constituent

Another option we tried is selecting the best parse analysis with any head nonterminal for evaluation rather than best parse with S as the top node. Doing this gives more options for selecting the best sentence parse, improving the coverage in case a thresholded grammar is used. We have collected the data for all experiments presented in this chapter, using the best parse as opposed to limiting the choice to S at the top. However, a slight decrease in accuracy (2.5 - 4%, or even greater for smaller grammars) was observed consistently throughout the entire set of results. It is therefore better to use only the trees with S at the top for selecting the best parse. The results for probabilistically compacted grammar (using the best parse) are shown in Figure 8.5.

8.4 Summary

In this chapter, we carried out a thorough evaluation of the grammar compaction method. Most of our earlier findings were confirmed when evaluating with a larger sample. The fully compacted grammar is tiny, but its parsing accuracy is poor. Probabilistic compaction does not affect parsing accuracy considerably, but the compaction rate is small (12%). Using ranking helps to reduce the grammar size even further (compaction rate is 34%), obtaining a gain in recall and a loss in precision. These results are not affected by filtering grammar

	Size	Reduction	Labelled		Unlabelled	
			Recall	Precision	Recall	Precision
Uncompacted	6,517	0%	71.43%	73.88%	75.03%	77.60%
Fully compacted	1,218	81%	24.46%	14.59%	39.68%	23.67%
Probabilistically compacted grammars:						
20%	2,158	67%	52.39%	40.11%	59.49%	45.54%
40%	3,098	52%	62.46%	52.75%	67.59%	57.08%
60%	4,037	38%	69.06%	64.10%	73.36%	68.09%
70%	4,507	31%	71.14%	67.98%	75.13%	71.79%
80%	4,977	24%	71.91%	69.92%	75.58%	73.68%
90%	5,447	16%	71.99%	72.39%	75.66%	76.08%
100%	5,917	9%	71.53%	73.88%	75.12%	77.59%

Table 8.5: Evaluating Probabilistic Compaction using the Best Parse

and sample by removing any rules and sentences containing **X** and **FRAG** elements. Using all top sentence level constituents for selecting the best parse rather than restricting the top constituent to **S** negatively affects parsing accuracy. Therefore, only **S** as the top-level node should be used for selecting the best parse.

We have successfully used simple thresholding to improve grammar compaction results even further. Thresholding is applied to the grammar before compaction to obtain the best results. Removing rules that only occur once (1-thresholding) reduces the grammar to less than a half of its original size, but the parsing accuracy hardly changes nor does the grammar coverage. Using 1-thresholding in conjunction with grammar compaction yields 58% and 69% overall compaction rates (with and without loss in precision). Thresholding grammar beyond this limit decreases parsing accuracy slightly, but reduces coverage.

While our precision and recall results are still not quite as high as those obtained in Magerman (1995) and Collins (1996) (78/71% as opposed to 87/88% for labelled evaluation), we are not competing with them. Magerman and Collins use statistical pattern recognition (or variations) for parsing which is different from using the traditional context-free model. They are also using lexical information which we have not incorporated into our system yet. SPATTER-type parsing is relatively new, while context-free grammars are still used in many natural language systems. Given that CFGs are very easy to extract from parsed

corpora, it is worth investigating their properties further, as Johnson (1998b) noted.

However, our results compare favourably to those obtained using a context-free or similar model and no lexical information. Results reported in Charniak (1996) are comparable to ours (for an uncompact grammar), which is not surprising considering that method of extraction is practically the same (extraction and 1-thresholding). They also use right-branching bias in parsing taking into account right-branching nature of English language. Without bias, they report 80% precision, 77% recall. With bias, these figures rise to 82% and 80% respectively. Parseval was used for evaluation. Our precision and recall are 81% and 74% using `evalb` with ‘unlabelled’ option which is similar to, but not the same as Parseval evaluation. The difference in recall figures is probably due to the fact that Parseval carries out some sentence transformations before comparing brackets, and `evalb` does not. Charniak does not proceed any further than extracting and thresholding the PTB grammar. However, we suggested two ways to reduce the extracted grammar size one of which does not affect precision and recall significantly, and the other yields higher recall but lower precision.

Sekine and Grishman (1995) use flattened fragments of PTB trees with `S` and `NP` being left-hand sides. The number of rules is even greater than ours (32,296 distinct rules), and their Parseval precision and recall figures 73% and 73%, slightly lower than ours. In his thesis (Sekine, 1998) Sekine reports better accuracy using 5 nonterminals instead of 2. His best labelled precision and recall are 75% and 73%. Ours are 78% and 71%, so that the accuracy is similar. Like Charniak, Sekine does not try to reduce the size of his grammar.

Shirai, Tokunaga, and Tanaka (1997) carry out a similar compaction of the grammar extracted from the EDR corpus in Japanese. However, their compaction is shallow, using only rule parse trees of depth 1 as the basis for rule elimination. Their grammar contains 2,219 rules extracted from about 180,000 sentences, and it achieves 74% precision and 62% recall on a sample consisting of 20,000 sentences. This is using 30 best sentence parses, and not one. Their results are lower than ours, but the language and training and test corpora are different.

Chapter 9

Conclusions

In this work we started off with investigating the context-free rule-based approach to parsing natural language. Limitations of purely rule-based approaches suggested the need to use probabilities. Statistical techniques have been successfully applied to part-of-speech tagging and speech recognition, but only recently has significant progress been made in using statistical methods for parsing natural language.

Grammar compaction as described in this thesis augments the treebank grammar approach by reducing the number of rules without losing parsing accuracy. Reducing the number of rules improves parsing time and memory usage which is high for treebank grammars due to their size. The reduction rate achieved in this work is small (9% with no loss in accuracy, and 30% with a gain in recall but loss in precision), but a similar method applied to Korean achieves a much higher reduction rate, similar to the rate achieved with ‘naive’ compaction (Lee et al., 1997). This may be because Korean is partially word-order free. Shirai and others (1995; 1997) use a shallow version of compaction method in for extracting a Japanese treebank grammar. The compaction method is language-independent and does not encode any linguistic information. Other methods using treebank grammars (Sekine and Grishman, 1995; Charniak, 1996; Johnson, 1998b) incorporate some linguistic knowledge to augment either the grammar or parsing. Attempting to improve the efficiency of a best-first chart-based parser, Caraballo and Charniak (1996; 1998) devised a method

reducing the number of edges. However, this method may be very computationally complex and demand a considerable amount of CPU time. The parser used in this work reduces the number of edges at runtime using little CPU time.

There are a number of ways we can extend treebank grammar compaction approach. One way is to lexicalize the treebank grammars. For example, if we consider not only part-of-speech tags in the treebank grammar extraction process, but also some words – not all as this may make the number of rules very large, and compact the resulting grammar. Perhaps, at least initially, we can use a set of most frequently occurring words (and substitute the remaining words with their part-of-speech tags), and expand this set as much as computational power allows. This may allow us to achieve parsing accuracy similar to that shown by non-CFG lexicalised methods (Collins, 1997; Charniak, 2000).

Or, we can apply grammar compaction to the DOP subtrees. Indeed, context-free rules are DOP subtrees of depth 1. We can then straightforwardly extend the rule parsing algorithm to the DOP grammar. Then, since from the same amount of text we can extract many more DOP subtrees than CFG rules, and since, combinatorially, the DOP subtrees can be combined in many more ways than CFG rules, there is a better potential for the compaction algorithm to achieve better results. And, if we were ever to achieve a significant compaction rate for a DOP grammar, this would allow to use DOP method to parse longer sentences (and larger samples).

We can also use the compaction algorithm in conjunction with MDL- or any other bottom-up grammar learning. Kit (1996) shows that words and grammar rules can be learned from raw text, a sequence of either characters or part-of-speech tags. The MDL method does not use any grammatical information encoded by a human, but instead attempts to reduce the description length (DL) of the sample by looking for patterns in text and replacing them with automatically generated (nonterminal or word) symbols. This method learns some very sensible words and right hand sides of grammatical rules. However, it does not learn the meanings of the words nor the grammatical functions of the rules it produces. So, perhaps we can use the grammar extracted from the treebank (and

compact) to recover the left hand sides of the MDL-extracted rules, and give those rules some probabilistic bias in order to enhance the MDL search process. Or we can do it the other way: increase the probabilities of the treebank rules which can also be learned by the MDL method – that may enhance the compaction process. One of the main problems with the treebank grammar we encountered is “noise” in the extracted rules.

We believe that the treebank grammars contain a wealth of information which is not fully utilised, partly because of the tremendous size of the grammar (and thus computational complexity of parsing algorithms involved), and partly because of the “noise” in the data – mostly due to the inconsistencies of the annotators. The grammar compaction is only one way to address this problem – it reduces the grammar size and attempts to filter out some “noise” by “subsuming” longer rules which can be “parsed” using shorter rules.

Appendix A

Tags used in the Penn Treebank

Tag	Description
Clause level	
S	Simple declarative clause.
SBAR	Clause introduced by a (possibly empty) coordinating conjunction.
SBARQ	Direct question introduced by a wh-word or wh-phrase.
SINV	Inverted declarative sentence.
SQ	Inverted yes/no question, or main clause of a wh-question, following the wh-phrase in SBARQ.
Phrase level	
ADJP	Adjective Phrase
ADVP	Adverb Phrase
CONJP	Conjunction Phrase
FRAG	Fragment
INTJ	Interjection
LST	List marker

Table A.1: Syntactic Tags

Tag	Description
NAC	Not A Constituent, used to show the scope of certain prenominal modifiers within a noun phrase
NP	Noun Phrase
NX	Used within certain complex noun phrases to mark the head of the noun phrase
PP	Prepositional Phrase
PRN	Parenthetical
PRT	Particle
QP	Quantifier Phrase
RRC	Reduced Relative Clause
UCP	Unlike Coordinated Phrase
VP	Verb Phrase
WHADJP	Wh-adjective Phrase
WHADV	Wh-adverb Phrase
WHNP	Wh-noun Phrase
WHPP	Wh-prepositional Phrase
X	Unknown, uncertain, or unbracketable

Table A.1: Syntactic Tags

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential <i>there</i>

Table A.2: Part-of-Speech Tags

Tag	Description
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP\$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	<i>to</i>
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle

Table A.2: Part-of-Speech Tags

Tag	Description
VCN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP\$	Possessive wh-pronoun
WRB	Wh-adverb

Table A.2: Part-of-Speech Tags

Tag	Marks:
Text Categories	
-HLN	headlines and datelines
-LST	list markers
-TTL	titles
Grammatical Functions	
-CLF	true clefts
-NOM	non NPs that function as NPs
-ADV	clausal and NP adverbials
-LGS	logical subjects in passives
-PRD	non VP predicates
-SBJ	surface subject
-TPC	topicalized and fronted constituents
-CLR	closely related

Table A.3: Functional Tags

Tag	Marks:
Semantic Roles	
-VOC	vocatives
-DIR	direction & trajectory
-LOC	location
-MNR	manner
-PRP	purpose and reason
-TMP	temporal phrases

Table A.3: Functional Tags

Appendix B

Programs Used for Evaluation the Grammar Compaction Method

Here is the list of programs (mostly Perl scripts) we used for evaluation.

`makegram_lib`, `makegram_A`, `B`, ...

Thresholds and then parses a subset of the grammar extracted from the PTB using the set of pre-stored charts, generating the thresholded grammar, the fully compacted grammar, and the ranking file

`prob_B`, `C`, ... Generates a set of probabilistically compacted grammars from the ranking file generated by `makegram_*`.

`gen_sample` Generates the test sample and the corresponding ‘gold’ file (in PARSEVAL and `evalb` formats).

`mh_parse` Parses a sample using Mark Hepple’s parser.

`mh_proc` Processes the output of the parser by first converting it into the format acceptable to `evalb` or PARSEVAL, and then running `evalb` or PARSEVAL. If some sentences cannot be parsed, the corresponding lines are removed from the ‘gold’ file, so that the evaluation software works properly.

References

- Aho, Alfred V. and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall.
- Batacharia, Bobby, David Levy, Roberta Catizone, Alexander Krotov, and Yorick Wilks. 1999. CONVERSE – a companion with potential. In *Machine Conversations*, Kluwer International Series in Engineering and Computer Science, Vol. 511. Kluwer Academic Publishers.
- Baum, Leonard E. 1972. An inequality and associated maximisation technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8.
- Bies, Ann, Mark Ferguson, Karen Katz, and Robert MacIntyre, 1995. *Bracketing Guidelines for Treebank II Style*. Penn Treebank Project, January. Distributed on the Penn Treebank release 2 CD-ROM by the Linguistic Data Consortium.
- Black, Ezra. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In DARPA91 (DARPA91, 1991), pages 306–311.
- Black, Ezra. 1993. Parsing English by computer: The state of the art. In *Proceedings of ATR International Workshop on Speech Translation*.
- Bobrow, D. G. 1968. Natural language input for a computer problem-solving system. In Minsky (Minsky, 1968), pages 135–215.
- Bod, Rens. 1992. A computational model of language performance: Data Oriented Parsing. In *Proceedings of COLING'92*, pages 855–859, Nantes, France.
- Bod, Rens. 1993a. Monte Carlo parsing. In *Proceedings of Third International Workshop on Parsing Technologies*, Tilburg/Durbuy.
- Bod, Rens. 1993b. Using an annotated corpus as stochastic grammar. In *Proceedings of European Chapter of ACL'93*, Utrecht.

- Bod, Rens. 1995a. *Enriching Linguistics with Statistics: Performance Models of Natural Language*. Ph.D. thesis, University of Amsterdam. ILLC dissertation series 1995-14.
- Bod, Rens. 1995b. The problem of computing the most probable tree in Data-Oriented Parsing and Stochastic Tree Grammars. In *Proceedings of the Sixth Conference of the European Chapter of the ACL*, pages 37–44.
- Bod, Rens. 1996. Two questions about data-oriented parsing. In *Proceedings of Fourth Workshop on Very Large Corpora*, Copenhagen, Denmark.
- Bod, Rens, Remko Bonnema, and Remko Scha. 1996. A data-oriented approach to semantic interpretation. In *Proceedings of Workshop on Corpus-Oriented Semantic Analysis, ECAI-96*, Budapest, Hungary.
- Bod, Rens and Remko Scha. 1996. Data-oriented language processing, an overview. Technical Report LP-96-13, Institute for Logic, Language and Computation, University of Amsterdam. (cmp-lg/9611003).
- Bonnema, Remko, Rens Bod, and Remko Scha. 1997. A DOP model for semantic interpretation. In *Proceedings of European Chapter of the ACL*, pages 159–167.
- Booth, T. 1969. Probabilistic representation of formal languages. In *Proceedings of 10th Annual IEEE symposium on switching and automata theory*.
- Booth, T. L. and K. S. Fu. 1975. Grammatical inference: Introduction and Survey – parts I and II. *IEEE Transactions on Systems, Man and Cybernetics*, 5.
- Brill, E. 1995. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4):543–565, December.
- Briscoe, E., C. Grover, B. Boguraev, and J. Carroll. 1987. A formalism and environment for the development of a large grammar of English. In *Proceedings of 10th International Joint conference on Artificial Intelligence*, pages 703–708, Milan, Italy.

- Briscoe, Ted and John Carroll. 1993. Generalized probabilistic LR parsing of natural language corpora with unification-based grammars. *Computational Linguistics*, 19(1):25–60.
- Briscoe, Ted and Nick Waegner. 1992. Robust stochastic parsing using the inside-outside algorithm. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 30–53. AAAI.
- Caraballo, Sharon A. and Eugene Charniak. 1996. Figures of merit for best-first probabilistic chart parsing. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 127–132. University of Pennsylvania, May.
- Caraballo, Sharon A. and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.
- Carroll, Glenn and Eugene Charniak. 1992a. Learning probabilistic dependency grammars from labeled text. In *Working Notes, Fall Symposium Series*, AAAI, pages 25–32.
- Carroll, Glenn and Eugene Charniak. 1992b. Two experiments on learning probabilistic dependency grammars from corpora. Technical Report CS-92-16, Brown University, Department of Computer Science, March.
- Carroll, J., E. Briscoe, and A. Sanfilippo. 1998. Parser evaluation: a survey and new proposal. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 447–454, Granada, Spain.
- Charniak, Eugene. 1993. *Statistical Language Learning*. MIT Press, Cambridge, MA.
- Charniak, Eugene. 1996. Tree-bank grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1031–1036. MIT Press, August.
- Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 6th*

- ANLP Conference and the 1st Meeting of NAACL*, pages 132–139, Seattle, Washington, USA.
- Cherry, L. L. 1978. PARTS – a system for assigning word classes to English text. Technical Report Computing Science Technical Report No. 81, Bell Laboratories, Murray Hill, New Jersey, June.
- Chomsky, N. 1963. Formal properties of grammars. In R. R. Bush R. D. Luce and E. Galanter, editors, *Handbook of Mathematical Psychology*, volume II. Wiley, New York.
- Chomsky, N. 1981. *Lectures on Government and Binding*. Foris Publications, Cinnamison, USA.
- Chomsky, N. 1995. *The Minimalist Program*. MIT Press.
- Church, Kenneth. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of Second Conference on Applied Natural Language Processing*.
- Church, Kenneth. 1992. Current practice in part of speech tagging and suggestions for the future. In Simmons, editor, *Sbornik praci: In Honor of Henry Kučera*. Michigan Slavic Studies.
- Colby, Ken. 1973. Simulation of belief systems. In Schank and Colby, editors, *Computer Models of Thought and Language*. Freeman, San Francisco, CA.
- Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 184–191.
- Collins, Michael. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the ACL*, pages 16–23.
- Collins, Michael. 1999a. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

- Collins, Michael. 1999b. Review of Beyond Grammar: an Experience-based Theory of Language. *Computational Linguistics*, 25(3):440–444.
- Daelemans, Walter, Jakub Zavrel, Peter Berck, and Steven Gillis. 1996. MBT: A memory based part of speech tagger-generator. In *Proceedings of the Fourth Workshop on Very Large Corpora*, pages 14–27, Copenhagen, Denmark.
1991. *Fourth DARPA Speech and natural language workshop*, Pacific Grove, CA, February.
- Doran, C., B. Hockey, P. Hopely, J. Rosenzweig, A. Sarkar, B. Srinivas, F. Xia, A. Nasr, and O. Rambow. 1997. Maintaining the forest and burning out the underbush in XTAG. In *Workshop on Computational Environments for Grammar Development and Language Engineering (ENVGRAM)*, Madrid, Spain.
- Earley, Jay. 1968. *An efficient context-free parsing algorithm*. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.
- Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.
- Eisner, J. and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 457–464, University of Maryland.
- Ferguson, Jack. 1980. Hidden Markov Models for speech. In *IDA-CRD*, Princeton, NJ.
- Forney, G. D. 1973. The Viterbi algorithm. *Proceedings of IEEE*, 61(3).
- Francis, W. 1964. A standard sample of present-day English for use with digital computers. Technical Report Report to the U.S. Office of Education on Cooperative Research Project No. E-007, Brown University, Providence, Rhode Island.
- Francis, W. N. and H. Kučera. 1982. *Frequency Analysis of English usage: lexicon and grammar*. Houghton-Mifflin, Boston.

- Fujisaki, T., F. Jelinek, J. Cocke, and E. Black. 1989. A probabilistic parsing method for sentence disambiguation. In *Proceedings of the International Workshop on Parsing Technologies*, Pittsburgh.
- Gaizauskas, Robert. 1995. Investigations into the grammar underlying the Penn Treebank II. Research Memorandum CS-95-25, University of Sheffield.
- Gaizauskas, Robert, Takahiro Wakao, Kevin Humphreys, Hamish Cunningham, and Yorick Wilks. 1995. Description of the LaSIE as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, pages 207–220. Morgan Kaufmann.
- Garside, Roger, Geoffrey Leech, and Geoffrey Sampson, editors. 1987. *The Computational analysis of English: a corpus-based approach*. Longman.
- Gazdar, G., E. Klein, G. K. Pullum, and I. A. Sag. 1982. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, Massachusetts.
- Gazdar, Gerald and Chris Mellish. 1989. *Natural Language Processing in Prolog*. Addison-Wesley.
- Goodman, Joshua. 1996. Efficient algorithms for parsing the DOP model. In *Proceedings of Empirical Methods in Natural Language Processing*, Philadelphia.
- Goodman, Joshua. 1998. *Parsing Inside-Out*. Ph.D. thesis, Department of Computer Science, Harvard University.
- Green, B. F., A. K. Wolf, C. Chomsky, and K. Laughery. 1963. BASEBALL: An automatic question answerer. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, pages 207–216.
- Grosz, Barbara J., Karen Sparck-Jones, and Bonnie Lynn Webber. 1986. *Readings in Natural Language Processing*. Morgan Kaufmann, Los Altos, CA.
- Grover, C., E. J. Briscoe, J. Carroll, and B. Boguraev. 1987. The Alvey Natural Language Tools grammar. *Lancaster Working Papers in Linguistics*, 47.

- Grover, C., E. J. Briscoe, J. Carroll, and B. Boguraev. 1989. The ANLT grammar (2nd release). Technical Report No. 162, Computer Laboratory, Cambridge University.
- Haegeman, L. 1991. *Introduction to Government and Binding Theory*. Basil Blackwell Ltd.
- Harkema, Hank. 2000. A recognizer for minimalist grammars. In *Sixth International Workshop on Parsing Technologies*, Trento, Italy.
- Japan Electronic Dictionary Research Institute, 1994. *EDR Electronic Dictionary User's Manual (in Japanese)*, 2.1 edition.
- Jelinek, Frederick. 1985. Markov source modeling of text generation. In *The Impact of Processing Techniques on Communications. Volume E91 of NATO ASI Series*. M. Nijhoff, Dordrecht, pages 569–598.
- Johnson, M. 1989. The computational complexity of Tomita's algorithm. In *Proceedings of the First International Workshop on Parsing Technologies*, pages 203–208, Carnegie-Mellon University, Pittsburgh.
- Johnson, Mark. 1998a. Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In *Proceedings of COLING-ACL'98*, pages 619–623, Montreal.
- Johnson, Mark. 1998b. PCFG models of linguistic tree representation. *Computational Linguistics*, 24(4):613–632.
- Johnson, S. C. 1975. YACC – Yet Another Compiler-Compiler. Technical Report Computer Science Technical Report No. 32, Bell Laboratories, July.
- Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA, pages 173–281.
- Kasami, Tadao. 1967. A note on computing time for recognition of languages generated by linear grammars. *Information and Control*, 10:209–214.

- Kay, Martin. 1963. A parsing procedure. In *Information Processing 1962*. North Holland, Amsterdam.
- Kay, Martin. 1985. Parsing in Functional Unification Grammar. In *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*. Cambridge University Press, Cambridge, pages 251–278.
- King, Margaret, editor. 1983. *Parsing Natural Language*. Academic Press, London.
- Kipps, J. 1989. Analysis of tomita’s algorithm for general context-free parsing. In *Proceedings of the First International Workshop on Parsing Technologies*, pages 193–202, Carnegie-Mellon University, Pittsburgh.
- Kit, Chunyu. 1996. Current progress in learning phrase structure with the minimum description length principle. Technical Report CS-96-08, Department of Computer Science, University of Sheffield.
- Knowles, Gerry, Briony Williams, and Lita Taylor, editors. 1996. *A Corpus of Formal British English Speech*. Longman, London.
- Knowles, Gerry, Anne Winchmann, and Peter Alderson, editors. 1996. *Working with Speech: perspectives on research into the Lancaster/IBM Spoken English Corpus*. Longman, London.
- Krotov, Alexander, Robert Gaizauskas, and Yorick Wilks. 1994. Acquiring a stochastic context-free grammar from the Penn Treebank. In *Proceedings of Third Conference on the Cognitive Science of Natural Language Processing*, pages 79–86, Dublin.
- Krotov, Alexander, Mark Hepple, Rob Gaizauskas, and Yorick Wilks. 1997. Compacting the Penn Treebank grammar. Technical Report CS-97-04, Department of Computer Science, University of Sheffield.
- Krotov, Alexander, Mark Hepple, Robert Gaizauskas, and Yorick Wilks. 1998. Compacting the Penn Treebank Grammar. In *Proceedings of the 36th Annual Meeting of the Associa-*

- tion for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 699–703.
- Krotov, Alexander, Mark Hepple, Robert Gaizauskas, and Yorick Wilks. 1999. Evaluating Two Methods for Treebank Grammar Compaction. *Journal of Natural Language Engineering*, 5(4):374–394.
- Kupiec, Julian. 1991. A trellis-based algorithm for estimating the parameters of a hidden stochastic context-free grammar. In DARPA91 (DARPA91, 1991).
- Kupiec, Julian. 1992. Robust part-of-speech tagging using a hidden markov model. *Computer Speech and Language*, 6:225–242.
- Lari, K. and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–36.
- Lee, K. J., J.-H. Kim, Y. S. Han, and G. C. Kim. 1997. Restricted representation of phrase structure grammar for building a tree-annotated corpus of korean. *Natural Language Engineering*, 3:215–230. part 2 and 3.
- Lin, Dekang. 1994. PRINCIPAR – an efficient, broad-coverage, principle-based parser. In *COLING-94*, pages 482–488, Kyoto, Japan.
- Magerman, David. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the ACL*, pages 276–283.
- Magerman, David M. 1994. *Natural Language Processing as Statistical Pattern Recognition*. Ph.D. thesis, Stanford University.
- Manning, Christopher and Hinrich Schütze. 1999. *Foundation of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.
- Marcus, M., G. Kim, M.A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of ARPA Speech and Natural language workshop*.

- Marcus, M., B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.
- Markov, Andrei. 1913. An example of statistical investigation in the text of ‘Eugene Onegin’ illustrating coupling of ‘tests’ in chains. In *Proceedings of the Academy of Sciences, St. Petersburg*, volume 7 of VI, pages 153–162.
- Martin, W. A., K. W. Church, and R. S. Patil. 1981. Preliminary analysis of a breadth-first parsing algorithm: Theoretical and experimental results. Technical Report TR-261, MIT LCS.
- Maxwell, J. III and R. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590.
- Minsky, M., editor. 1968. *Semantic Information Processing*. MIT Press, Cambridge, MA.
- Moore, Robert. 2000. Improved left-corner chart parsing for large context-free grammars. In *Sixth International Workshop on Parsing Technologies*.
- Oakes, Mike. 1994. Parser version 1.0 for DOS. www.prosperosoftware.com/nls.html. Can also be found in the Natural Language Software Registry: www.dfki.de/cl/registry/apps/parser.html.
- Oepen, S. and J. Carroll. 2000. Ambiguity packing in constraint-based parsing – practical results. In *Proceedings of the First Conference of the North American Chapter of the Association for Computational Linguistics (NAACL’00)*, pages 162–169, Seattle, WA.
- Paeseler, A. 1990. Modification of Earley’s algorithm for speech recognition. In A. Waibel and K. F. Lee, editors, *Readings in Speech Recognition*. Morgan Kaufmann.
- Pereira, F. and D. H. Warren. 1980. Definite Clause Grammars for language analysis — a survey of the formalism and a comparison with Augmented Transition Networks. *Artificial Intelligence*, 13:231–278.

- Pereira, Fernando and Yves Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of DARPA Workshop*, pages 128–135, Harriman, NY.
- Pollard, Carl and Ivan Sag. 1987. *Information-Based Syntax and Semantics*, volume 13 of *CSLI Lecture Notes*. CSLI, Stanford.
- Raphael, B. 1968. SIR: Semantic Information Retrieval. In Minsky (Minsky, 1968), pages 33–45.
- Ratnaparkhi, Adwait. 1996. A maximum entropy part of speech tagger. In *Proceedings of ACL-SIGDAT Conference on Empirical Methods in Natural Language Processing*.
- Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of EMNLP-2*. cmp-lg/9706014.
- Sampson, G., R. Haigh, and E. Atwell. 1988. Project APRIL: a progress report. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 104–112, Buffalo, NY.
- Sampson, Geoffrey. 1987. Evidence against the ‘grammatical’/‘ungrammatical’ distinction. In W. Meijs, editor, *Corpus Linguistics and Beyond*. Rodopi, Amsterdam, pages 219–226.
- Sampson, Geoffrey. 1995. *English for the Computer*. Clarendon Press, Oxford, UK.
- Samuëllson, Christer and Atro Voutilainen. 1997. Comparing a linguistic and stochastic tagger. In *Proceedings of the 35th Meeting of the ACL and the 8th Meeting of the EAACL*, Madrid.
- Sarkar, A. 2000. Practical experiments in parsing using tree adjoining grammars. In *Proceedings of the Fifth Workshop on Tree Adjoining Grammars (TAG-5)*, Paris, France.

- Schabes, Yves, Michael Roth, and Randy Osborne. 1993. Parsing the Wall Street Journal with the inside-outside algorithm. In *Proceedings of the Sixth Conference of the European ACL*, pages 341–347.
- Sekine, Satoshi. 1998. *Corpus-Based Parsing and Sublanguage Studies*. Ph.D. thesis, New York University.
- Sekine, Satoshi and Ralph Grishman. 1995. A corpus-based probabilistic grammar with only two non-terminals. In *Proceedings of Fourth International Workshop on Parsing Technologies*, pages 216–223.
- Sekine, Satoshi and Ralph Grishman. 1997. Domain project report. Technical report, New York University.
- Sharman, R., F. Jelinek, and R. Mercer. 1990. Generating a grammar for statistical training. In *Proceedings of DARPA Speech and Natural Language Workshop*, pages 267–274.
- Shieber, Stuart M. 1985. Evidence against the Context-Freeness of natural language. *Language and Philosophy*, 8(3):333–344.
- Shirai, Kiyoyaki, Takenobu Tokunaga, and Hozumi Tanaka. 1997. Automatic extraction of japanese probabilistic context-free grammar from a bracketed corpus (in japanese). *Journal of Natural Language Processing*, 4(1):125–146.
- Shirai, Kiyooki, Takenobu Tokunaga, and Hozumi Tanaka. 1995. Automatic extraction of Japanese grammar from a bracketed corpus. In *Proceedings of Natural Language Processing Pacific Rim Symposium*, pages 211–216, Korea, December.
- Sikkel, K. 1997. *Parsing Schemata – A Framework for Specification and Analysis of Parsing Algorithms*. Springer-Verlag.
- Sima'an, Khalil. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of COLING'96*, Copenhagen, Denmark.

- Skut, Wojciech, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, Washington, DC.
- Stolcke, Andreas. 1994a. *Bayesian Learning of Probabilistic Language Models*. Ph.D. thesis, University of California, Berkeley.
- Stolcke, Andreas. 1994b. How to BOOGIE: A manual for Bayesian object-oriented grammar induction and estimation. Internal Memo, International Computer Science Institute, June.
- Stolcke, Andreas. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.
- Taylor, L., C. Grover, and T. Briscoe. 1989. The syntactic regularity of English Noun Phrases. In *Proceedings of the 4th European Meeting of the Association for Computational Linguistics*, pages 256–263, UMIST, Manchester, UK.
- Thompson, Cynthia A., Raymond J. Mooney, and Lappoon R. Tang. 1997. Learning to parse natural language database queries into logical form. In *Proceedings of the ML-97 workshop on Automata Induction, Grammatical Inference and Language Acquisition*.
- Tomita, Masaru. 1984. LR parsers for natural languages. In *COLING-84*, pages 354–357.
- Tomita, Masaru. 1986. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer, Boston.
- van Halteren, Hans, Jakub Zavrel, and Walter Daelemans. 1998. Improving data driven wordclass tagging by system combination. In *Proceedings of COLING-ACL'98*, pages 491–497, Montreal, Canada.
- Vijay-Shanker, K., D. Weir, and O. Rambow. 1995. Parsing D-tree grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies*, pages 252–259.

- Viterbi, J. 1967. Error bounds for convolutional codes and asymptotically optimum decoding algorithm. *IEEE Trans. Information Theory*, IT-13:260–269, April.
- Voutilainen, A. 1995. A syntax-based part of speech analyser. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics*, pages 157–164, Dublin.
- Woods, W. A. 1970. Transition network grammars for natural language analysis. *Communications of ACM*, 13:457–471.
- Younger, Daniel H. 1967. Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10:198–208.