# HEAD-DRIVEN STATISTICAL MODELS FOR NATURAL LANGUAGE PARSING

Michael Collins

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

1999

---

Professor Mitch Marcus
Supervisor of Dissertation

---

Professor Jean Gallier
Graduate Group Chair

# Acknowledgements

Mitch Marcus was a wonderful advisor. He gave consistently good advice, and allowed an ideal level of intellectual freedom in pursuing ideas and research topics. I would like to thank the members of my thesis committee — Aravind Joshi, Mark Liberman, Fernando Pereira and Mark Steedman — for the remarkable breadth and depth of their feedback. I had countless impromptu but influential discussions with Jason Eisner, Dan Melamed and Adwait Ratnaparkhi in the LINC lab. They also provided feedback on many drafts of papers and thesis chapters. Paola Merlo pushed me to think about many new angles of the research. Dimitrios Samaras gave invaluable feedback on many portions of the work. Thanks to James Brooks for his contribution to the work that comprises chapter 5 of this thesis.

The community of faculty, students and visitors involved with the Institute for Research in Cognitive Science at Penn provided an intensely varied and stimulating environment. I would like to thank them collectively. Some deserve special mention for discussions that contributed quite directly to this research: Breck Baldwin, Srinivas Bangalore, Dan Bikel, Mickey Chandresekhar, David Chiang, Christy Doran, Kyle Hart, Al Kim, Tony Kroch, Robert Macintyre, Max Mintz, Tom Morton, Martha Palmer, Jeff Reynar, Joseph Rosenzweig, Anoop Sarkar, Matthew Stone, Debbie Steinig, Ann Taylor, John Trueswell, Bonnie Webber, Fei Xia, and David Yarowsky. I would also like to thank Amy Dunn, Mike Felker and Betsy Norman for making administrative matters run smoothly.

There was also some crucial input from sources outside of Penn. In the summer of 1996 I worked at BBN Technologies: discussions with Scott Miller, Richard Schwartz and Ralph Weischedel had a deep influence on the research. Manny Rayner and David Carter from

SRI Cambridge supervised my Masters thesis at Cambridge: their technical supervision was the beginning of this research, and their enthusiasm and support was really what propelled me into working in computational linguistics.

Finally, and most importantly, thanks to my parents and my sister Sarah, for giving their unwavering support during and before this work.

# Abstract


HEAD-DRIVEN STATISTICAL MODELS FOR NATURAL LANGUAGE PARSING

Michael Collins

Supervisor: Professor Mitch Marcus

Statistical models for parsing natural language have recently shown considerable success in broad-coverage domains. Ambiguity often leads to an input sentence having many possible parse trees; statistical approaches assign a probability to each tree, thereby ranking competing trees in order of plausibility. The probability for each candidate tree is calculated as a product of terms, each term corresponding to some sub-structure within the tree. The choice of *parameterization* is the choice of how to break down the tree. There are two critical questions regarding the parameterization of the problem:

1. What linguistic objects (e.g., context-free rules, parse moves) should the model's parameters be associated with? I.e., How should trees be decomposed into smaller fragments?

2. How can this choice be instantiated in a sound probabilistic model?

This thesis argues that the locality of a lexical head's influence in a tree should motivate modeling choices in the parsing problem. In the final parsing models a parse tree is represented as the sequence of decisions corresponding to a head-centered, top-down derivation of the tree. Independence assumptions then follow naturally, leading to parameters that encode the X-bar schema, subcategorization, ordering of complements, placement of adjuncts, lexical dependencies, wh-movement, and preferences for close attachment. All of these preferences are expressed by probabilities conditioned on lexical heads.

The goals of the work are two-fold. First, we aim to advance the state of the art. We report tests on Wall Street Journal text showing that the models give improved accuracy over other methods in the literature. The models recover richer representations than previous approaches, adding the complement/adjunct distinction and information regarding wh-movement. Second, we aim to increase understanding of statistical parsing models. Each parameter type is motivated through tree examples where it provides discriminative information. An empirical study of prepositional phrase attachment ambiguity is used to investigate the effectiveness of dependency parameters for ambiguity resolution. A number of parsing models are tested, and we give a breakdown of their performance on different types of construction. Finally, we give a detailed comparison of the models to others in the literature.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Parsing is a fundamental problem in language processing for both machines and humans. Most natural language applications (such as Information Extraction, Machine Translation, or Speech Recognition) would almost certainly benefit from high-accuracy parsing. From a scientific standpoint, there is the question of how people interpret language: what knowledge is used, and exactly how this knowledge is applied in practice.

In its simplest form, the parsing problem involves the definition of an algorithm that maps any input sentence to its associated syntactic tree structure. This thesis takes a machine-learning approach to the problem. The *sentence* $\rightarrow$ *tree* function[1] is induced from a training set, a set of example *sentence–tree* pairs. A test set of *sentence–tree* pairs is used to evaluate the model's accuracy.

In common with several other approaches, we adopt a statistical method. The learning problem then becomes a task of estimating parameter values from training data. This thesis considers two critical questions regarding the parameterization of the problem:

1. What linguistic objects (e.g., context-free rules, parse moves) should the model's parameters be associated with? I.e., How should trees be broken down into smaller fragments?

2. How can this choice be instantiated in a sound probabilistic model?

[1] We assume a definition of the parsing problem where each sentence must be mapped to a *single* tree structure, even if this requires disambiguation using knowledge sources in addition to the grammar. This contrasts with another common definition of the parsing problem, where the task is to recover all syntactically well-formed trees for a sentence, without any disambiguation.

Our goals are two-fold. First, we aim to advance the state of the art, by reporting improved parsing accuracy over previous results. Second, we aim to increase understanding of the parsing problem, through a detailed analysis of our parsing models, and a detailed comparison to models proposed elsewhere in the literature.

## 1.1 An Overview of this Chapter

Section 1.2 gives practical motivation for parsing, describing the information represented by parse trees, with some example applications as further illustration. Section 1.3 then describes a major difficulty, ambiguity. There are several pathological factors when parsing in a broad domain: the grammar required for broad coverage is large; sentences are typically long; and many common types of ambiguity lead to an exponentially growing (with respect to the sentence length) number of analyses. The end result is that ambiguity becomes an astonishingly severe problem. Section 1.4 sketches previous work, in rule-based and statistical approaches to parsing.

Section 1.5 is the major part of this introduction, outlining the approach in this thesis. We define the choice of *parameterization* as the choice of how to break down parse trees, two questions then arising: (1) what linguistic objects (e.g., context-free rules, parse moves etc.) should the model's parameters be associated with?; (2) How can this choice be instantiated in a sound probabilistic model? We introduce two criteria for a parameterization: *discriminative power* (that the parameters should represent the data well) and *compactness* (that the parameters should represent the data concisely).

An example from the belief networks literature is then discussed: it has several important lessons for parsing. The *causality* of physical problems is an important motivation for the modeling choices in belief networks; reasoning about the causality of the problem is essential for a compact parameterization. We argue that the analogue of causality in the parsing case is *locality*. In particular, the locality of a lexical head's influence in a parse tree should be used to motivate modeling choices in statistical parsing models. The final model in this work has parameters reflecting a head's local domain of influence in the tree (i.e., the model has parameters that encode the X-bar schema, subcategorization,

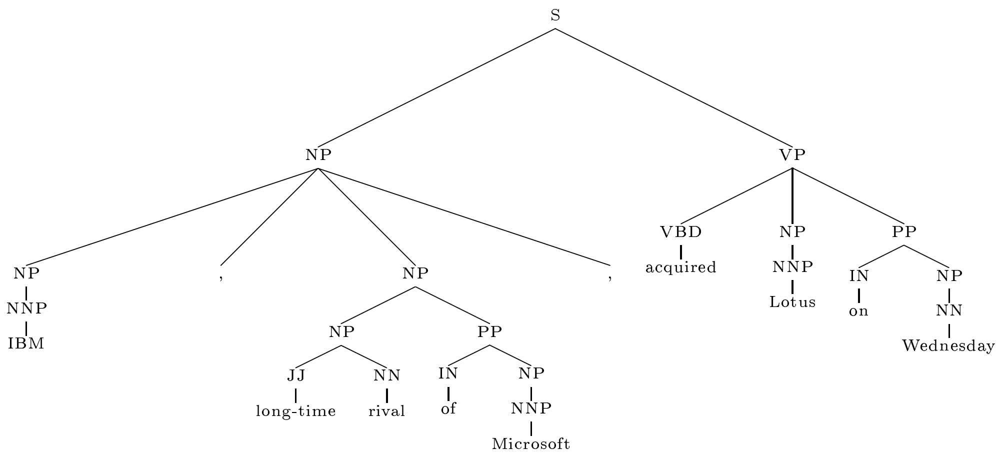


Figure 1.1: A Parse Tree

ordering of complements, placement of adjuncts, lexical dependencies, preferences for close attachment, and wh-movement; all preferences being expressed by parameters conditioned on head-words).

## 1.2 The Practical Motivation for Parsing

The tree in figure 1.1 represents several levels of information. The non-terminal directly above each word in the sentence is the part-of-speech for that word: for example, the tree indicates that "Lotus" is an `NNP` (proper noun), "acquired" is a `VBD` (past-tense verb), "long-time" is a `JJ` (adjective). The tree describes the hierarchical grouping of words into phrases: "IBM" is an `NP` (noun phrase), "IBM, long-time rival of Microsoft" is also an `NP`; "acquired Lotus on Wednesday" is a `VP` (verb phrase); and so on.

Finally, the tree represents grammatical relations between phrases or words. In a rule $\langle \texttt{S} \rightarrow \texttt{NP VP} \rangle$ the `NP` is the subject of the verb within the `VP` (by this rule, "IBM , long-time rival of Microsoft" is the subject of "acquired"). Similarly, $\langle \texttt{VP} \rightarrow \texttt{VB NP} \rangle$ represents an object-verb relationship ("Lotus" is the object of "acquired") and $\langle \texttt{VP} \rightarrow \texttt{VB} \ \ldots \ \texttt{PP} \rangle$ represents prepositional-phrase modification of a verb ("on Wednesday" modifies "acquired"). These syntactic roles allow us to directly read predicate-argument relations from the tree: that IBM, by virtue of being the subject, is doing the acquiring (rather than Microsoft); that Lotus, by virtue of being the object, is the acquiree; and so on.

A number of potential applications — Information Extraction (IE), Information Retrieval (IR), Machine Translation (MT), and Speech Recognition — serve to illustrate how this information could be useful. In an Information Extraction task, an NLP system fills a database with facts extracted from some group of documents. Given several years or decades of newswire text, a user might make the query "retrieve all *buyer–buyee* pairs where *buyer* is an acquiring company, and *buyee* is the company being bought". A parser together with a lexicon of verbs of buying/selling[2] could achieve much of this task.

High precision Information Retrieval is a related task. A query such as "retrieve all articles where Microsoft bought something" can be implemented, given a set of parsed articles, as a retrieval of all documents where *Microsoft* is the subject of verbs such as *buy*, *acquire* or *purchase*[3].

Machine translation (MT) is another application. One problem in MT is alignment: the description of how the word order in one language maps to the word order in another language. English and Japanese have quite different word orders, as in the following sentences:

| | |
|---|---|
| English: | IBM bought Lotus |
| Japanese: | *IBM Lotus bought* |

| | |
|---|---|
| English: | Sources said that IBM bought Lotus yesterday |
| Japanese: | *Sources yesterday IBM Lotus bought that said* |

On the surface, the correspondence between the two longer sentences looks quite complex. With parse structure, the mapping can be described by a simple recursive definition, each rule in the English grammar having a corresponding rule in Japanese. For example, the rule $\langle \texttt{VP} \rightarrow \texttt{VB NP} \rangle$ in English maps to the rule $\langle \texttt{VP} \rightarrow \texttt{NP VB} \rangle$ in Japanese (reflecting the fact that the object in English follows the verb, while in Japanese this order is reversed).

[2] The lexicon would need to specify for each lexical item its mapping from syntactic to semantic roles, containing entries such as {verb = *acquire/buy/purchase* ⇒ subject = Buyer, object = Buyee }, {verb = *sell* ⇒ PP headed by to = Buyer, object = Buyee }.

[3] Note that this implementation would *not* retrieve a document including the sentence in figure 1.1; a search based on simple co-occurrence or proximity of *Microsoft* and *buy/acquire/purchase* would retrieve a false-positive in this case.

Speech recognition is our final example. Current speech recognizers generally use a "trigram" language model to give a prior probability distribution over strings in a language. The prior probability of a sentence is calculated as a product of terms, each term corresponding to a window of three consecutive words. In the following example *of* is mis-recognized as *that*:

| | |
|---|---|
| Actual Utterance: | He is a resident of the U.S. and of the U.K. |
| Recognizer output: | He is a resident of the U.S. and *that* the U.K. |

In spite of being ungrammatical, the second sentence will receive high prior probability under a trigram model, as all triples of words that it contains are quite plausible. This explains the recognition error. The later models in this thesis (chapter 7) are not only parsing models, but also assign probabilities to strings in a language. In contrast to trigram models, they will give low probability to ungrammatical strings like the example, and will also capture statistical dependencies between words that fall outside the three-word window (for example, the subject–verb relation *IBM–acquired* in figure 1.1). (Model 2 of chapter 7 of this thesis assigns 78 times higher probability to the correct string in the above example. In contrast, a bigram model trained on the same data assigns over 10 times greater probability to the *incorrect* string, primarily because the bigram $\langle$`and that`$\rangle$ is around 15 times as frequent as $\langle$`and of`$\rangle$.)

## 1.3 A Major Problem: Ambiguity

Ambiguity is a major problem in parsing, and a primary motivation for statistical methods. A few causes of syntactic ambiguity (there are many others) are as follows:

- Part-of-speech (POS) ambiguity. For example, the word *saw* can be either a verb or a noun.

- Prepositional-Phrase attachment ambiguity. The sentence

  The woman saw the man with the telescope

  has at least two syntactic trees: one where the `PP` *with the telescope* modifies *man*, the other where it modifies *saw*.

- Coordination. In the phrase

  a program to promote safety in trucks and vans

  *vans* could be coordinated with *trucks*, *safety* or *program*.

The coordination example deserves further attention. There is clearly only one plausible interpretation: *vans* and *trucks* are coordinated. But there are several other structures that are syntactically well-formed: *vans* could be coordinated with *safety* or *program*; the preposition *in* could modify *safety*, *promote* or *program*. This gives at least 6 parse trees for the string[4]. All but one analysis is highly implausible from a semantic standpoint, but all 6 analyses will be recovered by a parser that is armed with syntactic information alone.

[Church and Patil 82] noted that PP attachment ambiguity leads to an exponential blow-up in the number of analyses for a sentence. A sequence $\langle$`VB NP PP*`$\rangle$ with $n$ `PP`s has $C_{n+1}$ analyses, where $C_{n+1}$ is the $(n+1)$'th Catalan number.[5] Other types of ambiguity show similar exponential behaviour. In newswire text, sentences are typically quite long: the average sentence length in Wall Street Journal (WSJ) is around 23 words, with 26% of sentences being over 30 words in length, and 7% being over 40 words. Moreover, a grammar with a very large number of rules is required for coverage of a broad domain. The combination of a large grammar, long sentences, and exponential factors leads to ambiguity being an astonishingly severe problem in broad domains such as Wall Street Journal.

[4] And there are almost certainly more analyses. *a program to promote* is an NP (as in *a program to promote is all we need*). *safety in trucks and vans* has two analyses as an NP. A wide-coverage grammar would need a rule `NP` $\rightarrow$ `NP NP` to cover appositive cases without a comma, as in *Suddenly, George Bush the pro-choice advocate became George Bush the abortionist* (an example from the Penn WSJ treebank). Thus there are two structures of the form [NP [NP *a program to promote*] [NP *safety in trucks and vans* ]], and still more analyses such as [NP [NP [NP *a program to promote*] [NP *safety* ]] [PP *in trucks and vans* ]].

[5] The $n$th Catalan number $C_n$ is $\frac{1}{n+1}\left(\begin{array}{c}2n\\n\end{array}\right)$.

## 1.4 Previous Approaches

### 1.4.1 Rule-Based Approaches

A standard approach to the parsing problem (see [Allen 87] for an overview) is as follows. A grammar is hand-crafted, often in some kind of unification formalism, often with a large amount of lexically specific information in the form of subcategorization information. Ambiguity is resolved through selectional restrictions (e.g., a lexicon might (1) specify that *eat* must take an object with the feature *+food*, and (2) specify which nouns in the lexicon have the *+food* feature). While perhaps feasible in a limited domain such as database query tasks (as in LUNAR [Woods 70] or SRI's system for the Air Travel Information System (ATIS) domain [Dowding et al. 93]), selectional restrictions run into several problems when scaled to wide-coverage tasks. First, the vocabulary size becomes so large that the sheer volume of information required becomes daunting[6]. Second, selectional restrictions have long been known in linguistics to have theoretical problems (e.g., see [McCawley 68]). While these problems may not come to light in a restricted domain, they are encountered frequently in a broad domain. At the very least, this implies that selectional restrictions should be encoded as soft preferences rather than hard constraints.

Additional problems are the large size of a broad-coverage grammar, and the fact that structural preferences[7] also come into play during disambiguation. Structural preferences interact in subtle ways with syntactic and semantic information, and are also best encoded as soft preferences.

It is difficult to find direct evidence of how successful these methods were on broad-coverage domains, as most systems were built and tested before treebanks became available to support evaluation. (Although see [Black et al. 93] section 1.2 for some relevant discussion.) We can, however, cite one source of indirect evidence. The Message Understanding

[6]We counted 24,444 distinct words in 40,000 sentences of WSJ text (a conservative estimate: we only counted words starting with lower-case letters, thereby excluding numbers and proper nouns). The problem is further compounded by the need for different entries for each word-sense of a word, increasing the volume of information required, and leading to difficult distinctions about what exactly constitutes a separate sense for a word.

[7]For example, close-attachment. The sentence *John was believed to have been shot by Bill* has two *semantically* plausible readings: *Bill* could have done either the shooting or the believing. The strong preference for the "shooting" reading is almost certainly due to a preference for attachment to the most recent verb.

Conferences (MUCs) evaluated NLP systems for information extraction from newswire articles. By the time of MUC-6 [MUC-6, 1995], none of the 5 best-performing systems (from BBN, Lockheed-Martin, NYU, SRA and SRI) used full parsing. At most, they used a partial parser, as in BBN's system; more often they used finite-state pattern-matching techniques as exemplified in [Appelt et al. 93]. Many of these sites originally attempted the MUC tasks with a full parser: see [Appelt et al. 93] and [Grishman 95] for descriptions of how two of the sites moved away from full parsing to finite-state methods.

### 1.4.2 Statistical Methods

In response to these difficulties, researchers began to investigate machine-learning approaches to the problem, primarily through statistical methods (with some notable exceptions, such as the rule-based learning methods of [Brill 93] or [Hermjakob and Mooney 97]). A "treebank" — a set of example sentence/parse-tree pairs — is annotated by hand and used to train a parsing model. Some part of the treebank is reserved as test data, being used to evaluate the model's accuracy.

Early work investigated the use of Probabilistic Context Free Grammars (PCFGs), but with rather disappointing results: as we will see later, a simple PCFG's failing is its lack of sensitivity to lexical information and structural preferences. Research then moved in several directions: towards models that had increased structural sensitivity; to partially supervised training algorithms; to probabilistic versions of lexicalized grammars; and to "history-based" models.

[Magerman 95] described the SPATTER parser (an extension of the work described in [Jelinek et al. 94]) applied to the Penn WSJ treebank [Marcus et al. 93]. This work represented a maturation of statistical parsing techniques, in several respects:

- It represented a major advance in the scale of the tasks undertaken by statistical parsers. All sentences up to 40 words in length were parsed, on a domain (WSJ) that is much less restricted than previously tested domains such as ATIS or the IBM computer manuals data.

- The parser was trained completely automatically from the treebank, with no requirement for a hand-crafted grammar.

- The results represented a major improvement over the accuracy for PCFGs: 84.5/84.0% precision/recall on section 23 of the Penn WSJ treebank. ([Charniak 97] later reported that a non-lexicalized PCFG scores around 72% averaged precision/recall on this task.)

- The model had parameters that conditioned heavily on lexical information, presumably accounting for much of its improvement over PCFG-based methods.

The SPATTER parser will serve as a benchmark for the work in this thesis. At its time of publication, it gave the best accuracy results reported on the Penn WSJ treebank — representing a substantial improvement over previous work. It embodies a very different approach from the one taken in this thesis; the contrast between the two methods raises several interesting issues.

## 1.5 This Thesis

This thesis considers alternative *parameterizations* for statistical parsing. By a *parameterization* we mean something quite explicit. Statistical parsing models assign a probability $Score(T, S)$ to each Tree-Sentence $(T, S)$ pair in a language[8]. The most likely tree for an input sentence $S$ is then defined as

$$T_{best}(S) = \arg\max_{T} Score(T, S) \tag{1.1}$$

Under this view the parsing problem is separated into two components: (1) the *model* is a function that defines a probability $Score(T, S)$ for each $(T, S)$ pair; (2) the *parser* is an algorithm that implements the search for $T_{best}$ for any input sentence $S$.

For the model to have a tractable number of parameters, a $(T, S)$ pair must be broken down into a set of "events" $\langle Event_1 ... Event_n \rangle$. $Score(T, S)$ is then calculated as a product

[8] $Score(T, S)$ could be either a joint probability $P(T, S)$ or a conditional $P(T|S)$; we use the term $Score$ to indicate neutrality between these possibilities.

of terms, each *Event* having a corresponding probability:

$$Score(T, S) = \prod_{i=1...n} Score(Event_i) \tag{1.2}$$

The choice of *parameterization* is the choice of how to break down the tree; in other words, what types of events to associate parameters with. There are many possible ways of breaking down trees. In a PCFG, parameters are associated with rules in the tree. In the SPATTER parser, parameters are associated with moves made by a parsing algorithm, each *Event* being a ⟨*parse-decision, context*⟩ pair (context is a set of features encoding information in the local area of the parse decision).

The choice of parameterization is central to the success of a parsing model. Two criteria are particularly important:

**Discriminative Power** The parameters should include the contextual information required for disambiguation decisions. If a parse tree is implausible because of a particular part of its structure, this should be reflected in the parameters associated with that parse tree.

PCFGs (in their simplest form) are an example of a model that fails in this respect. As we will see later, they are too insensitive to lexical information and structural preferences to provide a model with adequate discriminative power.

**Compactness** Given that it has adequate discriminative power, the model should have as few parameters as possible. The number of parameters in a model roughly determines the amount of training data required to train the model. Put another way, the compactness of a model will determine how close it comes to filling its full potential given the (almost certainly) limited amount of training data, and the consequent problem of under-training.

As illustration, take a model that associates probabilities with entire trees — each tree in a language is represented as a single *Event*, the full tree itself. This model has tremendous discriminative power, being able to model arbitrary properties of the tree. But it fails badly by the compactness criterion. It is unlikely that we would ever have enough training data to train a model of this type. Section 1.5.1 will give further illustration of the idea of compactness.

In short, a good parameterization should represent the data well, and it should represent the data concisely.[9]

This thesis addresses two questions regarding the parameterization of statistical parsing models:

1. What linguistic objects (e.g., context-free rules, parse moves etc.) should the model's parameters be associated with? I.e., How should trees be broken down into smaller fragments?

2. How can this choice be instantiated in a sound probabilistic model?

Before going into the details of these questions, we introduce an example from the Belief Networks literature that has some important lessons for the parsing problem.

### 1.5.1 An Example: Belief Networks and Causality

[Russell and Norvig 95] describe the following belief networks problem, which they cite as being originally due to Judea Pearl. A person has a house with a burglar alarm, and is at work. She has two neighbors, John and Mary, who are fairly reliable at calling her at work should the alarm go off. There are two possible triggers for the alarm: either a burglary or an earthquake. The task is to build a model that supports queries such as "Given that Mary has called, what is the probability that there was a burglary", or "Given that there is an earthquake, what is the probability that both John and Mary will call".

We can use 5 binary-valued random variables to model the problem: $A$ indicates whether or not the alarm has gone off; $E$ and $B$ indicate whether there was an earthquake or burglary respectively; $J$ and $M$ indicate whether John or Mary respectively have called. To support all possible inferences, we require a model of the joint probability $P(A, B, E, J, M)$. Marginal probabilities such as $P(B|M)$ or $P(J, M|E)$ can then be calculated.

[9]It is important to realise that we are not merely talking about the familiar trade-off between under and over-training. Different modeling choices can lead to parameterizations with the same discriminative power, but quite different compactness properties. Some models fail to capture important generalisations, unnecessarily fragmenting training data events, thereby being much less compact than they could be. Consequently, some models will take more training data to achieve the same performance as a more compact model; or equivalently, they will perform worse given the same amount of training data.

In the worst case, this distribution has 32 possible configurations and would require 31 parameters. However, we can build a more compact model using the following procedure:

1. Choose an ordering for the 5 variables — we will choose the order $\langle B, E, A, J, M\rangle$. Re-write the joint probability using the chain rule with this ordering:

$$P(A,B,E,J,M) = P(B)P(E|B)P(A|E,B)P(J|A,E,B)P(M|A,E,B,J) \quad (1.3)$$

2. For each of the 5 terms in equation 1.3, where possible make independence assumptions and thus reduce the number of parameters in the model. We make the following independence assumptions:

$$P(E|B) = P(E) \quad (1.4)$$

$$P(J|A,E,B) = P(J|A) \quad (1.5)$$

$$P(M|A,E,B,J) = P(M|A) \quad (1.6)$$

   The independence assumptions are justified through our knowledge of *causality* in the world. For example, it is reasonable to assume that there is no causal link between earthquakes and burglaries, and therefore that $P(E|B) = P(E)$.

These steps give a final solution to the model:

$$P(A,B,E,J,M) = P(B)P(E)P(A|E,B)P(J|A)P(M|A) \quad (1.7)$$

Providing that the independence assumptions are correct, the model fulfills the first criterion of fitting the data well. It also fulfills the second criterion, being quite compact: the model has 10 parameters as opposed to 31 for the unreduced model[10].

In general, the design of a belief net model is a three step process:

1. Choose the variables involved in the problem. In the previous example, we chose the variables $A$, $B$, $E$, $J$ and $M$.

2. Choose an ordering for the variables (in the example, $\langle B, E, A, J, M\rangle$) and rewrite the joint probability using the chain rule.

[10]This difference may not seem dramatic, but that is because there are only 5 variables in the example. Providing that sufficient independence assumptions can be made, there will generally be an exponential reduction in the number of parameters in a belief net with $n$ nodes, from $2^n - 1$ to $O(n)$ parameters.

3. Make independence assumptions for each of the terms in the chain of probabilities.

An important point is that a good choice of variable ordering (step 2) is critical to a good parameterization of the problem. To see this imagine we chose a different order, $\langle M, J, E, B, A\rangle$. In this case the chain rule gives:

$$P(A,B,E,J,M) = P(M)P(J|M)P(E|J,M)P(B|J,M,E)P(A|J,M,E,B) \qquad (1.8)$$

With equation 1.8, it turns out to be impossible to simplify any of the terms through reasonable independence assumptions[11]. The result is that each of the parameters is not reduced, and the model requires 31 parameters. The model will represent the same distribution as the model with 10 parameters (i.e., it has equal discriminative power), but it is much less compact. Note also that the parameters in the model are now quite counter-intuitive: it is difficult to judge what terms such as $P(A|J,M,E,B)$ or $P(J|M)$ correspond to in the world.

Two lessons emerge from this. For a compact parameterization of a problem:

1. The choice of variable ordering is crucial. With a good ordering, independence assumptions fall out naturally. With a bad ordering, it may be impossible to make independence assumptions.

2. A general guideline is that the choice of ordering should reflect the causality of the problem. $B$ and $E$ can both cause an alarm, and should therefore precede $A$ in the ordering; $A$ can cause John or Mary to call, and should therefore precede $J$ and $M$ in the ordering. The ordering $\langle B, E, A, J, M\rangle$ satisfies these constraints (as would some other orderings such as $\langle E, B, A, M, J\rangle$).

These lessons have strong parallels in the parsing case.

### 1.5.2 Modeling Parse Structures

Most statistical parsing models — including PCFGs, SPATTER, and the models in this thesis — fall within the framework of history-based models (originally applied to parsing

[11] As an example, $P(J|M) = P(J)$ is not a good independence assumption, as knowing that Mary has called increases the chances that the alarm has gone off, thereby increasing the chance that John has called. Similar arguments apply to the other terms.

by [Black et al. 92b][12]). The design of a history-based model involves three steps:

1. **Representation**. Choose how to represent parse trees. For example, choose the set of part-of-speech tags and non-terminal labels in the tree; choose whether or not to have lexical head-words attached to non-terminals; choose whether to represent words directly, or as their morphological stems, or as bit-strings derived through clustering techniques.

2. **Decomposition**. This step involves the definition of a one-to-one mapping between parse trees $T$ and decision sequences $\langle d_1...d_n\rangle$. The sequence $\langle d_1...d_n\rangle$ can be thought of as the sequence of moves that builds $T$ in some canonical order. The model defines either a joint probability $P(T,S)$ over all possible tree-sentence $(T,S)$ pairs, or a conditional probability $P(T|S)$ over all candidate trees for a particular sentence. Given a mapping between trees and decision sequences, the probability of a tree can be written either as

$$P(T|S) \quad = \quad \prod_{i=1...n} P(d_i|d_1...d_{i-1},S) \tag{1.9}$$

   or

$$P(T,S) \quad = \quad \prod_{i=1...n} P(d_i|d_1...d_{i-1}) \tag{1.10}$$

   In conditional models, a tree is usually associated with the sequence of decisions made by a particular parser in recovering the tree. In joint models, the decisions are usually the steps in some top-down derivation of the tree, for example the sequence of productions used in a left-most derivation of a context-free grammar.

3. **Independence Assumptions**. This step involves the definition of a function, $\Phi$, which groups decision sequences into equivalence classes, thereby reducing the number of parameters to manageable proportions. The final model is then one of the forms

$$P(T|S) \quad = \quad \prod_{i=1...n} P\left(d_i|\Phi\left(d_1...d_{i-1},S\right)\right) \tag{1.11}$$

[12] Although the term "history-based" has perhaps recently become strongly associated with conditional models that define $P(T|S)$, i.e., [Jelinek et al. 94, Magerman 95, Ratnaparkhi 97], they can also be used to create joint models defining $P(T,S)$. In fact, the paper that originally used this term — [Black et al. 92b] — described a joint model.

$$P(T,S) = \prod_{i=1...n} P\left(d_i | \Phi\left(d_1...d_{i-1}\right)\right) \tag{1.12}$$

The choice of $\Phi$ could either be made by hand, or automatically using a machine-learning technique such as decision trees (the SPATTER parser uses a conditional model in step (2), together with decision trees for automatic search for independence assumptions).

These three steps are highly analogous to the three steps in the design of belief networks. The choice of representation in parsing corresponds to the choice of variables in belief networks ($A$, $B$, $E$, $J$, and $M$ in the alarm example); the choice of decomposition in parsing corresponds to the choice of variable ordering in belief networks; the choice of independence assumptions is required in both model types.

We argue that the second step — the choice of decomposition — is critical for a successful parameterization of the parsing problem. This shouldn't be a surprise, given that its analogue in the belief networks case (the choice of variable ordering) was seen to have a critical effect in that example. With a good choice for step 2, it is easy to make independence assumptions that lead to a good parameterization of the problem. A bad choice for step 2 leads to linguistically implausible parameters: i.e., parameters that fail by either or both of the criteria of discriminative power and compactness.

The SPATTER model can now be evaluated in terms of the three steps. From the comments in the preface of [Magerman 95], it seems the reasoning behind the model was as follows. Linguistic expertise would be used in step 1 — a linguistic "expert" would pick all parse-tree features that might be useful for disambiguation. Decision trees would then be used for step 3, identifying the features that were actually useful for disambiguation.

On the surface this seems a very plausible approach; but we argue that its weakness is a lack of appreciation for the importance of step 2. SPATTER chose to represent a parse-tree as the sequence of decisions made in a bottom-up parse of the tree. The parameters in the model are then associated with $\langle$*parse-move, context*$\rangle$ pairs. When we analyse the model in more detail later in this thesis, we will see that in some cases parameters miss important disambiguating information, and in other cases they unnecessarily fragment training data. The parameters are deficient in terms of both discriminative power and compactness.

Our emphasis of the importance of step 2 leaves us with an important question. In the belief networks example, causality was used to motivate the choice of variable ordering and independence assumptions. In the parsing problem there is no physical process, and therefore no clear notion of causality. So what should motivate the choice of decomposition in parsing?

### 1.5.3 A Motivation for the Choice of Decomposition: Causality and Locality

In the belief networks example, causality was used to motivate the choice of variable ordering and independence assumptions. But what is the analogue of causality in the parsing case?

The key is that reasoning about causality in the physical world is closely related to reasoning about *locality*. The *domain of locality* of an event is the region of space-time that it can affect. By defining the domain of locality for an event we can reason about the causal influences that it can and can't exert. In the belief nets example, in making the independence assumption $P(E|B) = P(E)$ we are reasoning about the locality of the effects of a burglary: that a burglary's influence is limited, and that it certainly doesn't extend to causing earthquakes. Similarly, in deciding that $P(J|M, A, B, E) = P(J|A)$, we are reasoning about the domain of locality of events $M, A, B, E$: that the events $M, B, E$ are limited in their domain of locality, and do not directly influence the chance of John calling (for example, we are assuming that Mary does not run into John in the street, tell him that she's already called, and thereby dissuade him from calling).

Once we have equated causality with locality, the step to linguistics and parsing is a small one. Much of the work in linguistics focuses on conditions on structural locality. In particular, highly lexicalized formalisms such as LFG [Kaplan and Bresnan 82], TAG [Joshi 87], CCG [Steedman 96], HPSG [Pollard and Sag 94] and Minimalism/GB [Chomsky 95] stress the locality of the influence of lexical heads in a parse tree. Each word in a sentence affects a limited domain within the tree. TAG is a clear example — a lexical head has an associated elementary tree that directly represents any constraints associated with that head ([Frank 92] discusses these constraints extensively). Lexical entries in LFG, CCG and HPSG are also rich representations of the constraints associated with a head word. Chomsky's discussion of X-bar theory within Minimalism ([Chomsky 95] page 172) contains the following passage:

> An X-bar structure is composed of projections of heads selected from the lexicon. Basic relations, then, will involve the head as one term. Furthermore, the basic relations are typically "local".

Now take the example tree in figure 1.2(a). In our final models, the probability for this tree will be calculated as a product of terms, each term being a probability that is conditioned on one of the lexical items in the sentence. Thus each word in the sentence will have an associated set of probabilities. Take the parameters associated with *told* as an example. We will see in the next section that these parameters reflect the local influence of *told* in the parse tree. Figure 1.2(b) shows a sub-tree associated with *told*; we will take this to be the domain of locality for *told*. *told* will be responsible for the spine of this sub-tree, `S` $\rightarrow$ `VP` $\rightarrow$ `VBD`; for its subject, its object, an `NP` adjunct and the `SBAR` complement; and for the head-words of these constituents, *IBM, him, yesterday,* and *that.* To flip the analogy, we can think of *told* as having caused, or generated, all and only these parts of the tree in 1.2(a).

Our aim, then, should be to choose an order of decomposition that allows a parameterization that reflects the local influence of lexical heads. This leads to an immediate constraint on the decomposition of the tree: a lexical head must be generated before all structure that is dependent upon it. For example, *told* must be generated before the tree structure and other lexical items in the tree of figure 1.2(b). This constraint leads us to a head-centered derivation of the tree. Given this choice of decomposition, independence assumptions reflecting the domain of locality of each lexical head fall out naturally.

### 1.5.4 A Sketch of the Parameter Types

We now describe the different types of parameters in the final models of this thesis. (Chapter 7 gives an exact definition of the parameters; this section gives more of a sketch, omitting some details for the sake of conciseness. Chapter 3 gives detailed motivation for each

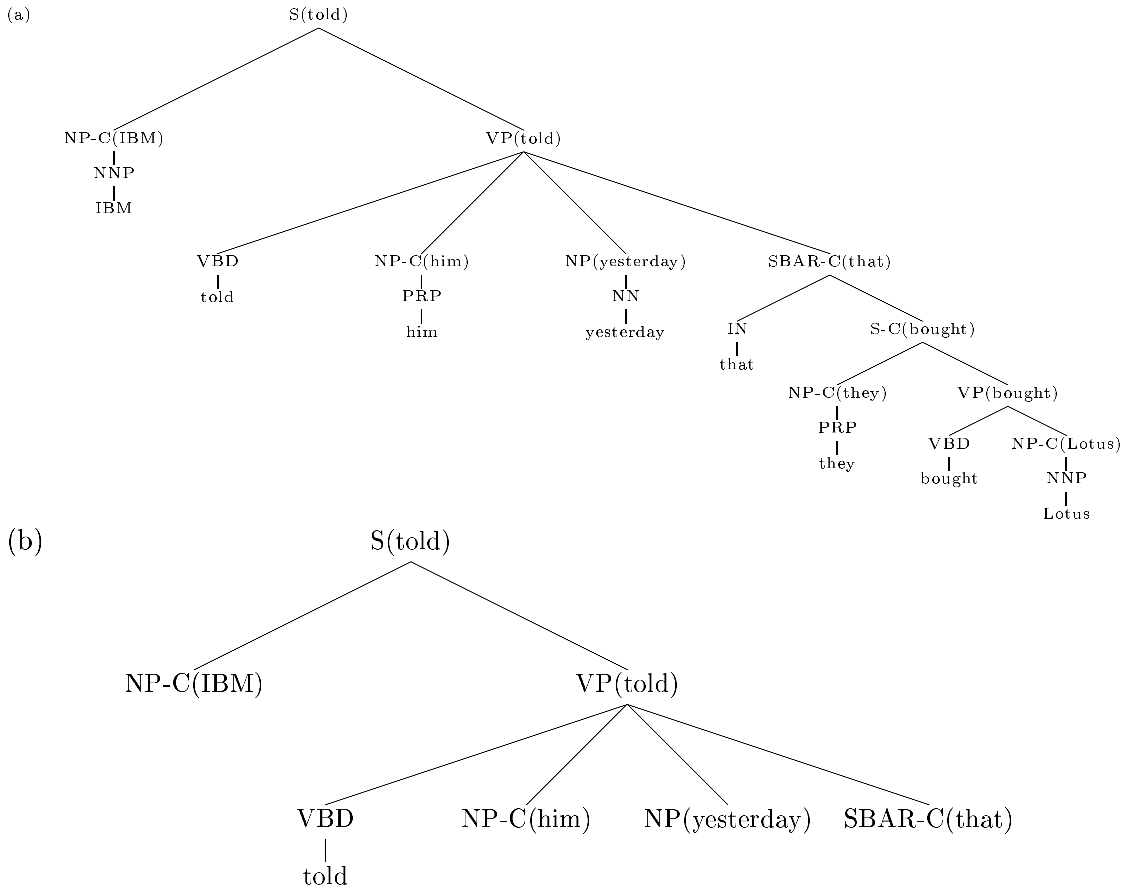



Figure 1.2: (a) A parse tree. Head-words for each non-terminal are shown in parentheses (for example, *told* is the head of the constituent `S(told)`). The `-C` tag indicates complements as opposed to adjuncts: *him* is a complement (object), *yesterday* is an adjunct (temporal modifier). (b) The domain of locality of *told* in the tree. Only these parts of the tree are directly dependent on *told*.

of these parameters in terms of their discriminative power.) For illustration, we will sketch the parameters associated with *told* in the tree of figure 1.2(a).

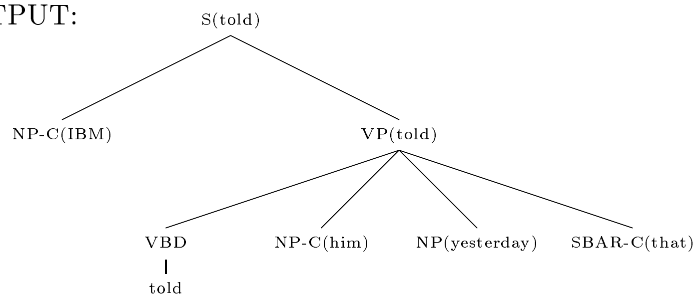
The model uses a history-based approach; a parse-tree is represented as the sequence of decisions in a canonical top-down, head-centered derivation of the tree. Each decision has an associated probability. The first decision in the derivation is a special move that chooses the top node of the tree. In the example, `S(told)` is generated:

INPUT: START OUTPUT: S(told)

This decision has probability $P(\text{S(told)} \mid \text{START})$.

The next part of the derivation — a sub-sequence of decisions — will contribute the probabilities conditioned on *told*. The input to this sub-derivation is the `S(told)` non-terminal that has just been generated. The output of the sub-derivation is a sub-tree with `S(told)` at its root:

INPUT: S(told) OUTPUT:




The output is built incrementally in a series of stages. Each stage contributes a different parameter type, namely probabilities corresponding to: (1) a choice of the X-bar spine of the sub-tree; (2) a choice of subcategorization frames; (3) a choice of the relative order of the complements, and the placement of adjuncts; (4) a choice of head words for the complements and adjuncts. Next, we will describe the stages of the derivation, and explain the probabilities that result from each stage.

Note that once the sub-derivation has generated the sub-tree associated with *told*, the non-terminals `NP-C(IBM)`, `NP-C(him)`, `NP(yesterday)` and `SBAR-C(that)` will recursively generate their own sub-trees, thereby contributing probabilities conditioned on *IBM*, *him*, *yesterday* and *that* respectively. Figure 1.3 shows the sub-derivations associated with other words in the sentence. We will now describe how the sub-derivation associated with *told* is broken down into a sequence of decisions.

**Head Projection Parameters**

INPUT: S(told) OUTPUT: S(told)
|
VP(told)
|
VBD
|
told

There are two decisions in this example involving the generation of the X-bar spine of the tree. The probability of each decision is given by a parameter specific to the headword *told*. $P($`VP`$|$`S,told`$)$ is the probability of an `S` node with *told* as its head-word taking a `VP` node as its head. $P($`VBD`$|$`VP,told`$)$ is the probability of a `VP` node with *told* as its head-word taking a `VBD` node as its head. The spine is then complete because `VBD` (unlike `VP` and `S`) is a part-of-speech tag.

Once their values are learned, these parameters encode the X-bar schema — that a verb projects upwards to a `VP` which in turn projects to an `S`, or that a noun projects up to an `NP`, and so on.

**Subcategorization Parameters**

INPUT: S(told)
|
VP(told)
|
VBD
|
told

OUTPUT: S(told)
|
$\{\text{NP-C}\}^{\text{VP(told)}}\{\}$
|
$\{\}^{\text{VBD}}\{\text{NP-C,SBAR-C}\}$
|
told

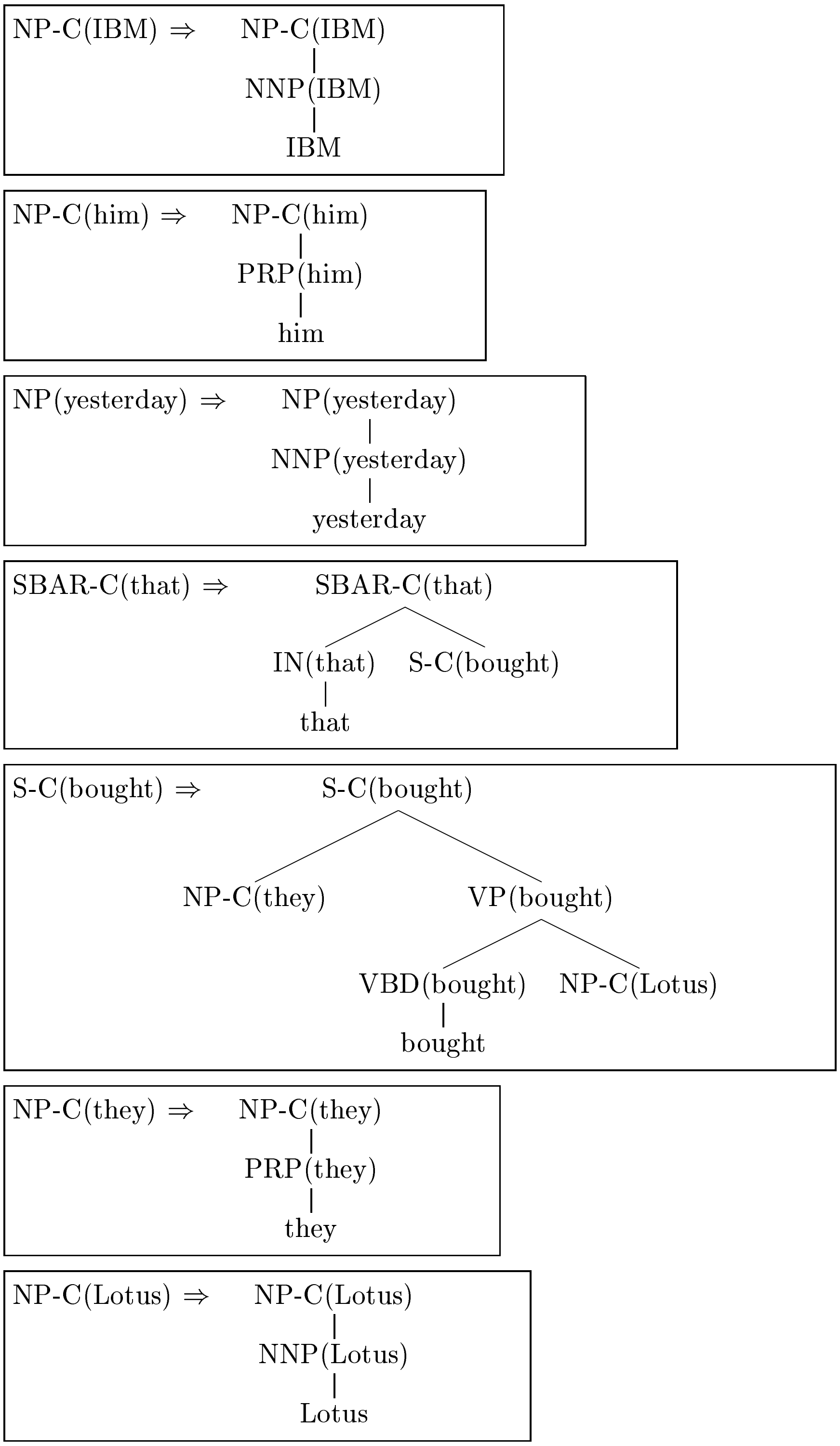



Figure 1.3: Sub-derivations for words other than *told* in the sentence. Each sub-derivation will contribute a set of probabilities conditioned on the lexical item that is the input to the sub-derivation.

In the next stage, subcategorization decisions are made. Left and right subcategorization frames are chosen to be added at each level of the tree. Thus there are four decisions to be made at this stage for the example. Again, each decision has an associated probability:

- $P(\{$`NP-C`$\}|$`parent=S,child=VP,told,LEFT`$)$ is the probability that *told* takes a single `NP` complement to its left, at the level where the parent non-terminal is `S` and the head-child non-terminal is `VP` (i.e., it is the probability of *told* taking a single subject).

- $P(\{\}|$`parent=S,child=VP,told,RIGHT`$)$ is the probability that *told* takes no complements to its right at the `S/VP` level.

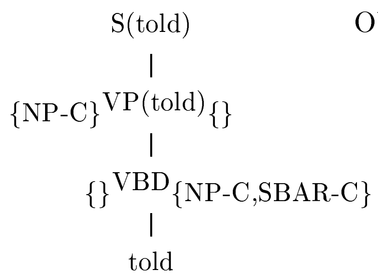
- $P(\{\}|$`parent=VP,child=VBD,told,LEFT`$)$ is the probability that *told* takes no complements to its left at the `VP/VBD` level.

- $P(\{$`NP-C,SBAR-C`$\}|$`parent=VP,child=VBD,told,RIGHT`$)$ is the probability that *told* takes both an `NP` and an `SBAR` complement to its right at the `VP/VBD` level. (Note that the subcategorization frame is an unordered multiset, so at this stage the relative order of the two complements is unspecified.)

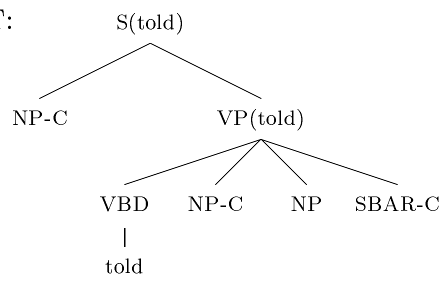
These parameters allow the model to learn a probability distribution over possible subcategorization frames for each entry in the lexicon: for example, the probability that any given verb will take a single subject (presumably a probability equal to 1); the probability of *told* taking `NP` and `SBAR` complements; or the probability of *give* taking two `NP` complements.

**Placement of Complements and Adjuncts**

INPUT:




OUTPUT:




Having chosen the subcategorization frames, decisions are made regarding the relative order of the different complements, and about whether the head takes any adjuncts. In the example, four sequences of modifier non-terminals are generated: the sequence ⟨NP-C⟩ to the left of the VP; a null sequence to the right of the VP; a null sequence to the left of the VBD; and the sequence ⟨NP-C NP SBAR-C⟩ to the right of the VBD. Each of these sequences is generated in a sequence of steps, each step having an associated probability.

As an example, figure 1.4 shows how the ⟨NP-C NP SBAR-C⟩ sequence is generated. The sequence is generated from inside to outside (i.e., the NP-C is generated first, the SBAR-C is generated last). At each point either a non-terminal or the STOP symbol, which terminates the sequence, is chosen with some probability. The probability is conditioned on the parent and child non-terminals (VP and VBD in figure 1.4), the head-word (told) and the direction relative to the head (RIGHT). The probability is also conditioned on the subcategorization frame, which keeps track of which subcategorization requirements have not yet been fulfilled. Initially this frame is {NP-C,SBAR-C}; by the end of the sequence all requirements are fulfilled and the frame is empty.

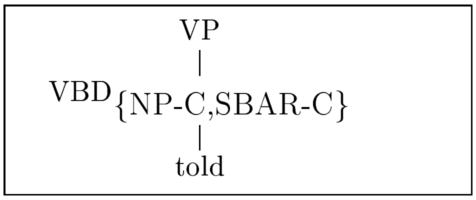
There are additional parameters associated with the three other sequences:

- $P$(NP-C|S,VP,{NP-C},told,LEFT). The probability of generating a (subject) NP-C to the left of the head at the S/VP level, given that this requirement hasn't yet been fulfilled (the {NP-C} conditioning variable indicates that the subject is still required).

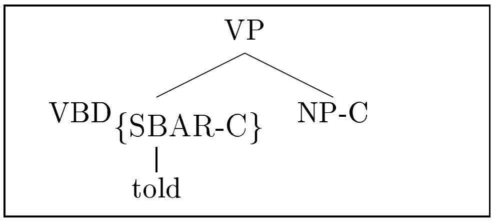
- $P$(STOP|S,VP,{},told,LEFT). The probability of terminating the sequence to the left of the head at the S/VP level, given that there are no requirements left in the subcategorization frame. (Note that adjuncts could also be generated at this point, extending the sequence.)

- $P$(STOP|VP,VBD,{},told,LEFT). The probability of terminating the sequence to the left of the head at the VP/VBD level, given that there are no requirements left in the subcategorization frame. (Note that adjuncts could also be generated at this point.)

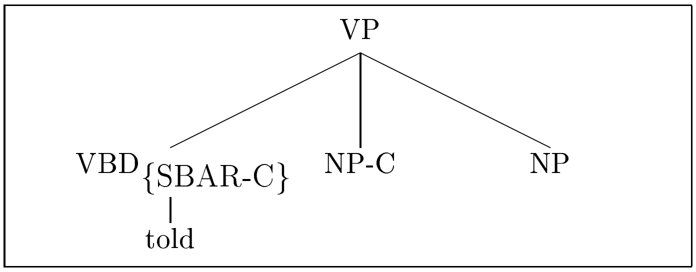
- $P$(STOP|VP,VBD,{},told,RIGHT). The probability of terminating the sequence to the right of the head at the VP/VBD level, given that there are no requirements left in the subcategorization frame. (Note that adjuncts could also be generated at this point.)



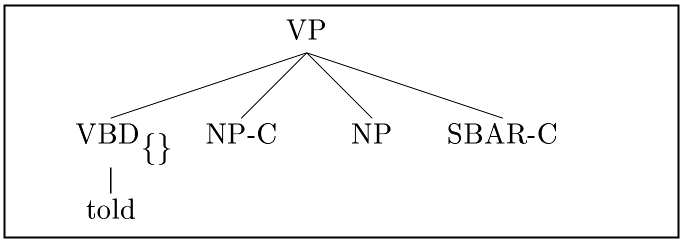

$\Downarrow P(\text{NP-C}|\text{VP, VBD, \{NP-C,SBAR-C\}, told, RIGHT})$




$\Downarrow P(\text{NP}|\text{VP, VBD, \{SBAR-C\}, told, RIGHT})$




$\Downarrow P(\text{SBAR-C}|\text{VP, VBD, \{SBAR-C\}, told, RIGHT})$




$\Downarrow P(\text{STOP}|\text{VP, VBD, \{\}, told, RIGHT})$

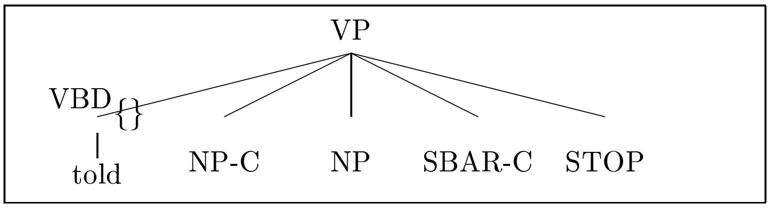
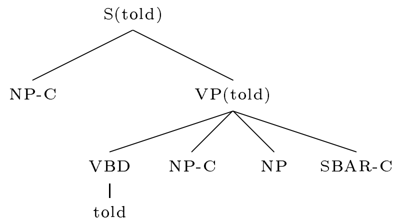



Figure 1.4: Generation of the ⟨`NP-C NP SBAR-C`⟩ sequence to the right of the `VBD`.

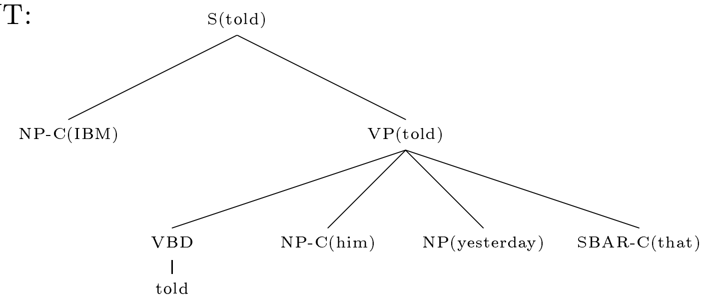
Thus these parameters encode the relative ordering of the complements (for example, that the NP-C object is closer to *told* than the SBAR-C complement); and they also encode the decision to take adjuncts, such as the NP between the NP-C and SBAR-C complements.

**Dependency Parameters**

INPUT:




OUTPUT:




Finally, a head-word is chosen for each modifier. $P(\texttt{IBM}|\texttt{told,S,VP,NP-C})$ is the probability of seeing *IBM* as the head-word of the NP-C in subject position (the triple $\langle\texttt{S,VP,NP-C}\rangle$ signifies the subject-verb relationship involved between the two words). There are similar parameters for the probability of seeing *him* as the head-word of the object, *yesterday* as the head-word of the NP adjunct, and *that* as the head-word of the SBAR-C complement.

[Hindle and Rooth 91] showed that dependency parameters could be a powerful source of disambiguating evidence for PP attachment ambiguities; this is one motive for generalizing this result to other relationships in the tree. Dependency parameters can be considered to be a probabilistic counterpart of selectional restrictions.

**Surface Distance Parameters**

The parameters can be modified to allow the model to learn that the dependent words for a particular head (e.g., {*IBM, him, yesterday, that*} for *told*) are likely to be placed close to the head in the surface string. This preference[13] is a direct reflection of a head-word's domain of locality in the surface ordering.

From a parsing point of view, these preferences mean that the *shot by Bill* analysis should get much higher probability than the *believed by Bill* analysis in the following example:

John was believed to have been shot by Bill

In a language production sense, these preferences reflect facts such as the following:

John was believed by Bill to have been shot

is much more likely to be uttered than

John was believed to have been shot by Bill

(given that the *believed by Bill* interpretation is intended in both cases).

For reasons of brevity we do not describe here how these preferences are represented in the parameterization of the parsing problem, merely noting that they are important. Precise details are given in the models of chapters 6 and 7.

### 1.5.5 Results

Thus far the motivation for our approach has been quite abstract. Our eventual goal is to build a parser of high accuracy. Empirical results should then be the central test of a model, and they also have the advantage of objectivity.

The results in chapter 7 show that the parser recovers constituents in section 23 of the Penn WSJ treebank with 88.3/88.0% precision/recall. At the time of writing, these were the best published results on this task. They represent a 25% relative error reduction over the results for SPATTER when trained and tested on the same data. The model achieves these results using quite simple (interpolated) estimation techniques, leading to a substantial reduction in training time over the SPATTER model. An implication is that the approach in this thesis is quite orthogonal to that of SPATTER, and a promising

[13] We mean a statistical preference, rather than a hard grammatical constraint. Analysis in chapters 6 and 7 of this thesis shows that dependent words are very likely to be placed close to the head they modify, at least in WSJ English.

area of future research would be to combine the strengths of the two methods, through a motivated choice of parse-tree decomposition (as in this thesis) together with more powerful estimation techniques (such as decision trees or maximum-entropy models, as used in [Ratnaparkhi 97]). We would argue that a further advantage of our models is that the parameters are linguistically intuitive, and therefore that it is easier to understand why the models work: this is important for future work in improving parse accuracy, and for the understanding of how the models fit with other areas of research such as psycholinguistics or linguistics.

### 1.5.6 A Summary of the Argument

We now briefly summarize the arguments of this section:

- Two criteria dictate the success of a parameterization of the parsing problem: discriminative power and compactness.

- We take a belief networks example. In belief networks there are three stages in the design of a model: (1) a choice of variables (or representation); (2) a choice of ordering of the variables, with a consequent expression of the chain rule; (3) a choice of independence assumptions, where terms in the chain rule are simplified.

- In the belief networks example, the second step (the choice of variable ordering) is crucial. With a good ordering, reasonable independence assumptions and a compact parameterization follow naturally. With a bad ordering, it may be impossible to make good independence assumptions.

- A general guideline is that the choice of variable ordering should reflect the *causality* of the problem; the resulting parameters also reflect the causality of the system being modeled.

- History-based models are a generalization of most statistical parsing models, and their design can also be considered to be a three stage process: (1) a choice of parse tree representation; (2) a choice of parse tree decomposition; (3) independence assumptions. These three stages are highly analogous to the belief networks problem.

- We argue that Step 2 in the design of a history-based model is critical for a good parameterization of the parsing problem. This leaves an open question: if causality is used to motivate step 2 in the belief networks problem, what motivates step 2 in the parsing problem?

- Reasoning about causality in the physical world is equivalent to reasoning about the locality of the effects of an event. Thus causality is directly related to locality. Many linguistic theories emphasize structural locality: in particular, lexicalized formalisms emphasize the locality of a head's influence in a parse tree. The locality of a lexical head's influence should be used to motivate the choice of decomposition in the parsing problem.

- The result is a head-centered decomposition of the parse tree. Independence assumptions then follow naturally, with the parameters reflecting a head's local domain of influence in the tree (i.e., the model has parameters that encode the X-bar schema, subcategorization, ordering of complements, placement of adjuncts, lexical dependencies, and preferences for close attachment; all preferences being expressed by parameters conditioned on head-words).

## 1.6 Overview

**Chapter 2** gives mathematical results that will be used throughout the thesis. It concentrates on two topics. First, the chapter discusses methods for defining probabilities over structured events such as sentences, sentence/tagged-sequence pairs, or parse trees. Second, it discusses estimation of parameter values from training data counts.

**Chapter 3** considers a series of alternative parameterizations for the parsing problem, in terms of their discriminative power. We begin with arguments for why PCFGs make poor models for statistical parsing: their failing is a lack of sensitivity to lexical information and structural preferences. We then give examples motivating the use of dependency parameters, features encoding the grammatical relations between words, subcategorization parameters, parameters encoding the preference for close-attachment, and the use of part-of-speech tags as word-class information.

**Chapter 4** considers previous work on statistical parsing.

**Chapter 5** considers a statistical method for the resolution of a specific (and relatively difficult) case of syntactic ambiguity, PP-attachment ambiguity. The method illustrates the power of dependency parameters for disambiguation: a method that considers the head-words involved in PP-attachment decisions resolves ambiguous cases with over 84% accuracy. (The major part of this chapter is joint work with James Brooks, having been originally described in [Collins and Brooks 95].)

**Chapter 6** describes a first attempt at building a full statistical parser that embodies many of the parameter types described in Chapter 3. It can be considered a direct attempt to generalize the PP-attachment model of Chapter 5 to the case of full parsing. Results are promising: the model recovers constituents with 85.7/85.3% precision and recall. There are, however, some mathematical problems with the model, that almost certainly impact performance, and also lead to problems with extending the model to include additional parameter types or to recover additional information. These problems lead us to the parsing models of the next chapter. (Much of the work in this chapter was originally described in [Collins 96].)

**Chapter 7** describes three models for statistical parsing: moving from a more mathematically motivated model with (almost) the same parameter types as the parser of chapter 6 (model 1); to a model that makes the argument/adjunct distinction and has subcategorization parameters (model 2); to a model that includes a treatment of wh-movement (model 3). Models 2 and 3 recover constituents with 88.3%/88.0% precision/recall. The chapter takes a closer look at the strengths and weaknesses of the parser by considering its accuracy on various different types of dependencies. (Much of the work in this chapter was originally described in [Collins 97].)

**Chapter 8** discusses the parser of chapter 7 in more detail. It takes a closer look at the close-attachment preferences; it considers the implicit assumptions about tree representation that the models make; and it gives a closer comparison of the models to other work on parsing the Penn WSJ treebank.

**Chapter 9** gives some thoughts on future work, while **Chapter 10** gives conclusions.

### 1.6.1 Reader's Guide

Chapter 7 describes the final parsing models of this work, Chapter 8 gives supporting discussion. These chapters should be quite self-contained; readers who are already familiar with previous work on statistical parsing may wish to skip straight to these chapters, and later refer to the earlier chapters for more detail.

Chapters 2 to 6 give background material supporting the final models. Specifically:

- Chapter 2 is intended for readers who have some background in probability/statistics and theory of automata, but who may not be so familiar with stochastic automata and their application to machine-learning problems. (Most of the work should, for example, be very familiar to readers from a speech recognition background, with the possible exceptions of the sections on probabilistic context-free grammars and history-based models.)

- Much of Chapter 3 should be familiar to readers who are well grounded in syntax/linguistics. Note, however, that the goals of statistical parsing (finding the single most likely parse for a sentence) lead to a slightly different emphasis in some cases. For example, many of the examples that are used to motivate different choices of representation involve syntactic ambiguity: these examples lead to a need to deal with some phenomena (e.g., close-attachment preferences) that are usually considered to be outside of the grammar.

- Chapter 4 aims to give a comprehensive literature review.

- Chapter 5 should be a good introduction to machine-learning approaches to ambiguity resolution in natural language. It also gives several useful experimental results concerning the use of dependency parameters. Chapter 6 gives further development of many of the intuitions behind the use of dependency parameters.

# Chapter 2

# Statistical Models

## 2.1 Introduction

This chapter gives mathematical background that is used throughout this thesis. Readers who are familiar with statistical approaches to machine learning may wish to move straight to the later sections, for example section 2.6 (Probabilistic Context Free Grammars) and section 2.7 (history-based models).

The first two sections contain introductory material:

- Section 2.2 gives some basic definitions from probability theory and mathematical statistics, and describes how supervised machine learning problems can be treated within a probabilistic framework.

- Section 2.3 gives a general discussion of the application of probabilistic methods to natural language problems. It first gives a general strategy for modeling structured events. It next gives a general solution of the maximum-likelihood parameter estimates for a class of models that is general enough to include almost all of the models described in this chapter. It finally gives two conditions for a model to be "well-formed". These conditions will be used to motivate many of the modeling choices proposed later in the chapter.

We then describe some specific techniques for modeling structured events:

- Sections 2.4, 2.5, 2.6 and 2.7 describe a number of model types, in increasing order of generality: Markov models for the definition of probabilities over sequences; Hidden Markov Models for probabilities over sequence/state-sequence pairs; Probabilistic Context-Free Grammars for probabilities over sentence-tree pairs; and finally "history-based" models as a generalization of the previous model types. In each case we prove that the model is "well-formed" under the definition in section 2.3.3, and we derive the maximum-likelihood estimates, using the result in section 2.3.2.

- Section 2.9 describes a variety of estimation methods that are refinements of maximum-likelihood estimators. Effective models for natural language tasks often have a very large number of parameters, leading to problems with extreme sparseness of the counts that are the basis of the parameter estimates. This section describes estimation methods that deal robustly with sparse data problems.

## 2.2 Probability Theory

We assume some standard definitions from probability theory (see, for example, [BD 77] for a comprehensive review). If the set $\zeta$ is a discrete event space, and $P$ is a probability distribution over this space, then: (1) $0 \leq P(A) \leq 1$ for all $A \in \zeta$; (2) $\sum_{A \in \zeta} P(A) = 1$.

In most examples the probability measure will be parameterized: i.e., $P$ will also be a function of some vector of parameters $\Theta$. We write the probability of event $A$ given some parameter setting $\Theta$ as $P(A \mid \Theta)$. The *parameter space* $\Omega$ is then the space $\{\Theta \mid P(A \mid \Theta)$ is a probability measure over $\zeta\}$.

As an example, take the case of tossing a coin that can appear as either heads (H) or tails (T), where the probability of it landing as heads is $p$. In this case:

- The event space $\zeta$ is the set $\{\text{H}, \text{T}\}$.

- The parameter vector $\Theta$ has a single component, $p$.

- The probability measure $P(A \mid \Theta)$ is defined as $p$ if $A = \text{H}$, $1 - p$ if $A = \text{T}$.

- The parameter space $\Omega$ is the set $[0, 1]$ ($p$ must take some real value between 0 and 1 for $P(A \mid \Theta)$ to be a probability measure).

### 2.2.1 Maximum Likelihood Estimation

Now assume we have a sample, a sequence of $n$ events $\mathbf{X} = \langle X_1, X_2 ... X_n \rangle$, drawn from $\zeta$. For example, we might toss the coin 5 times and see $\mathbf{X} = \langle$H,T,T,H,T$\rangle$. Given this sample, how do we calculate an estimate of $\Theta$, which we will call $\hat{\Theta}$?

A very general method is to use maximum likelihood estimation ([BD 77] chapter 3 introduces ML estimation; chapter 4 describes many of its properties). Assuming that the events are independent of each other, the likelihood function, $L$, is defined as

$$L(\mathbf{X} \mid \Theta) = \prod_{i=1...n} P(X_i \mid \Theta) \tag{2.1}$$

The maximum likelihood estimate $\hat{\Theta}_{ML}$ is the value of $\Theta$ (in the parameter space $\Omega$) that maximizes this likelihood function:

$$\hat{\Theta}_{ML} = \arg\max_{\Theta \in \Omega} L(\mathbf{X} \mid \Theta) \tag{2.2}$$

In the coin example, the likelihood of the sample $\mathbf{X} = \langle$H,T,T,H,T$\rangle$ is

$$L(\langle \text{H,T,T,H,T} \rangle \mid \Theta) = p(1-p)(1-p)p(1-p) = p^2(1-p)^3 \tag{2.3}$$

and the maximum likelihood estimate of $p$ in this case is

$$\hat{p}_{ML} = \arg\max_{p \in [0,1]} p^2(1-p)^3 = \frac{2}{5} \tag{2.4}$$

In general, for a sample of size $n$ with $h$ heads, it can be shown that $\hat{p}_{ML} = \frac{h}{n}$.

### 2.2.2 Notation

Before describing how probability theory can be used in a machine-learning context, we introduce some notation. In this chapter, $P$ is generally used to denote a probability distribution, for example $P(y|x)$ denotes the conditional probability of $y$ given $x$, $P(y,x)$ denotes the joint probability of $y$ and $x$. If the event spaces for $x$ and $y$ are $\mathcal{X}$ and $\mathcal{Y}$ respectively, the implication is that $\forall x \in \mathcal{X}$ $\sum_{y \in \mathcal{Y}} P(y|x) = 1$, and that $\sum_{x \in \mathcal{X},\, y \in \mathcal{Y}} P(y,x) = 1$.

We will also use $Score(x,y)$ to denote a probability associated with a pair $(x,y)$. The term $Score(x,y)$ can denote any one of the distributions $P(x,y)$, $P(y|x)$ or $P(x|y)$ — it is a neutral term, which will be useful when the exact nature of a model's parameters is underspecified. If $Score$ includes conditional probability notation, all variables to the right of "|" must be conditioned upon. For example, $Score(x, y|\Theta)$ could be any one of $P(x, y|\Theta)$, $P(y|x, \Theta)$, or $P(x|y, \Theta)$.

### 2.2.3 Probabilistic Approaches for Supervised Machine Learning Problems

The supervised machine-learning problems considered in this thesis take the following form. We assume the task is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is a set of possible inputs, $\mathcal{Y}$ is a set of possible outputs. Training data is a set of $n$ input-output pairs, $\langle x_1, y_1\rangle ... \langle x_n, y_n\rangle$ where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, and $y_i = f(x_i)$.

As an example, take the part-of-speech (POS) tagging problem (e.g., see [Church 88]):

- Define $\mathcal{V}$ to be a *vocabulary*, a set of possible words in a language. A sentence in the language is a sequence $\langle w_1, w_2 ... w_m\rangle$ where $m \geq 0$ and $w_i \in \mathcal{V}$. The input space $\mathcal{X}$ is then the set of all possible sentences in the language.

- Define $\mathcal{T}$ to be a set of possible part-of-speech tags. A *tag sequence* is a sequence $\langle t_1, t_2 ... t_m\rangle$ where $m \geq 0$ and $t_i \in \mathcal{T}$. The output space $\mathcal{Y}$ is then the set of all possible tag sequences.

- Training data is $n$ examples drawn from $\mathcal{X} \times \mathcal{Y}$, i.e. $n$ sentence/tag-sequence pairs.

- The learning task is then to induce a function from word sequences to tag sequences, $f : \mathcal{X} \rightarrow \mathcal{Y}$.

In probabilistic approaches the problem is transformed from directly learning a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, to learning a probability function $Score : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$. $Score(x, y)$ is either a conditional probability $P(y|x)$ or a joint probability $P(x, y)$. Having defined $Score$, $f(x)$ can be defined as the most likely member of $\mathcal{Y}$ under this probability distribution:

$$f(x) = \arg\max_{y \in \mathcal{Y}} Score(x, y) \tag{2.5}$$

Thus every candidate output for the input $x$ will have an associated score; the candidate sequences can be ranked in order of probability, moreover the candidate with the highest score can be chosen as the single most likely output (assuming that there is only one candidate with this highest score).

In a parameterized model a parameter vector $\Theta$ is an additional argument to the $Score$ function; the $Score$ function is now written as $Score(x, y|\Theta)$. This function, the model *structure*, is fixed, with all possible variation being described by the parameter space $\Omega$, the space of possible values for $\Theta$. In this framework the learning problem becomes a problem of setting the parameter estimates, $\hat{\Theta} \in \Omega$,[1] from the set of training examples.

In summary, within this framework the modeling task divides into 3 problems:

1. Defining the model structure, a function $Score(x, y|\Theta)$ with an associated parameter space $\Omega$.

2. Defining a parameter estimation method, i.e. a function from training samples $\langle x_1, y_1\rangle ... \langle x_n, y_n\rangle$ to parameter estimates, $\hat{\Theta}$. This function is usually strongly related to $Score(x, y|\Theta)$: for example, maximum likelihood estimation specifies that $\hat{\Theta} = \arg\max_{\Theta\in\Omega} \prod_{i=1...n} Score(x_i, y_i|\Theta)$.

3. Defining a search method: an algorithm that for any input $x$ will find the most likely output, $y_{best} = \arg\max_{y\in\mathcal{Y}} Score(x, y|\hat{\Theta})$.

### 2.2.4 A Note on the Definition of the Event Space

For simplicity we have assumed that the input and output spaces $\mathcal{X}$ and $\mathcal{Y}$ can be defined separately. The result is that $\mathcal{X} \times \mathcal{Y}$ will include some members $(x, y)$ that are ill-formed because $x$ and $y$ do not "match". For example, in the POS tagging case, some members of $\mathcal{X} \times \mathcal{Y}$ will be word/tag sequences with different lengths. In general, $Score$ will not be defined for these cases: from here on we will simply assume that for ill-formed inputs, $Score(x, y) = 0$.

[1] In general if $q$ is a parameter, we will write $\hat{q}$ to denote an estimate of that parameter's value.

## 2.3 Defining Probabilities over Structured Events: Some General Results

In the previous section we saw that a probabilistic approach to the machine learning problem can be reduced to 3 sub-problems: (1) defining the model structure; (2) defining an estimation method; and (3) designing a search algorithm. This section gives some very general methods and results for steps (1) and (2) in natural language problems. The following sections then describe specific approaches for probabilistic modeling of sequences, tagged sequences, and parse trees.

A major difficulty in probabilistic modeling of the NLP problems addressed in this chapter is the complexity of the event space. The event space in the coin-tossing example was relatively simple: it was a finite set, having only two members; and each event in the set was atomic. In NLP problems both the input space $\mathcal{X}$ and the output space $\mathcal{Y}$ can be of high dimensionality, with the members of these sets being complex structures (e.g. word or tag sequences, or tree structures). As illustration, take the following 3 examples:

**Sentences** Suppose there is a vocabulary $\Sigma$, a set of possible words in a language. Each event in a sample is a sentence drawn from the vocabulary $\Sigma$, i.e. a sequence of words $\langle w_1, w_2 ... w_n\rangle$ such that $n \geq 0$ and $w_i \in \Sigma$. The event space $\zeta$ is then the set $\Sigma^*$ (where $\Sigma^*$ is as defined in formal language theory, see [Hopcroft and Ullman 79] page 2).

This is the language modeling problem for speech recognition (e.g., see [Jelinek 90]).

**Tagged Sentences** Suppose that there is a set of possible words in a language, $\mathcal{V}$, and a set of possible tags, $\mathcal{T}$. Each event is a (sentence, tag-sequence) pair, i.e. a pair $(\langle w_1, w_2 ... w_n\rangle, \langle t_1, t_2 ... t_n\rangle)$ such that $n \geq 0$, $w_i \in \mathcal{V}$, and $t_i \in \mathcal{T}$.

**Parsed Sentences** Suppose that a context-free grammar $G$ defines a set of well-formed (sentence, tree) pairs in some language. The event space is then this set of (sentence, tree) pairs.

The complexity of the event space in these cases leads to a requirement for quite complicated parameterizations. The coin-tossing example involved a single parameter, $p$, with a simple function from events to probabilities ($P(H|p) = p$, $P(T|p) = 1 - p$). In the three previous examples we must define a probability measure over an infinite set of events. We must define this distribution using a finite number of parameters, moreover only as many parameters as can be reliably estimated from the training sample.

In the following sections we first describe a general strategy for modeling structured events: the idea of associating probabilities with sub-structures within individual events — for example associating probabilities with tag sub-sequences in the POS tagging problem, or with rules in the parsing problem. Next we give a general solution for the maximum-likelihood estimates for a class of models that includes practically all of the models proposed in this chapter. Finally, we give two conditions for a model to be "well-formed": (1) that a model $Score(x, y|\Theta)$ must either define a conditional or joint probability distribution; (2) that the maximum-likelihood estimates for the model should be derivable in closed form, or by some iterative solution (EM estimation and Maximum Entropy models both use iterative re-estimation techniques).

### 2.3.1 Defining the Model Structure: Associating Probabilities with Sub-Structures

This section addresses the problem of defining model structure, i.e. defining the function $Score(x, y|\Theta)$. As a straw man we first consider the simplest possible model. We choose a conditional probability model, so that $Score(x, y|\Theta) = P(y|x, \Theta)$. The parameter vector $\Theta$ lists a probability for each member of the set $\mathcal{X} \times \mathcal{Y}$, thus $\Theta$ is a vector with $|\mathcal{X}| \times |\mathcal{Y}|$ elements. The parameters of the model are estimated from training data using maximum-likelihood estimation, which gives parameter estimates $\hat{\Theta}$ such that

$$Score(x, y|\hat{\Theta}) = P(y|x, \hat{\Theta}) = \frac{Count(x, y)}{Count(x)} \tag{2.6}$$

($Count(x, y)$ is the number of times $\langle x, y \rangle$ is seen in the training sample, and $Count(x) = \sum_{y \in \mathcal{Y}} Count(x, y)$ is the number of times $x$ is seen in the training sample). With this model, finding the most likely output $y_{best}$ for an input $x$ amounts to a table look-up, simply choosing the output that has been seen with $x$ the most frequently in training data, i.e. defining $f(x) = \arg\max_{y \in \mathcal{Y}} Count(x, y)$.

Unfortunately this model will fail for all but the most trivial problems, due to the vast number ($|\mathcal{X}| \times |\mathcal{Y}|$) of parameters. At the very least, the model presupposes that every input $x$ seen in test data will have been seen at least once in training data: for inputs not seen in training data $Count(x, y) = Count(x) = 0$, and $P(y|x, \hat{\Theta})$ is undefined. In problems such as POS tagging or parsing, the proportion of sentences in test data that have also been seen in the training sample is typically extremely low, and the method will fail badly.

A model with far fewer parameters can be defined by associating parameters with *sub-structures* within the training and test events, rather than with entire events sampled from the set $\mathcal{X} \times \mathcal{Y}$. The score for an entire structure is then calculated as a product of probabilities: one probability for each sub-structure within the entire structure. In the parsing problem, where the output $y$ is a context-free tree, a natural step is to associate a probability with each rule in the grammar, rather than with each possible tree. In POS tagging, the commonly used HMM model (see [Church 88]) has two types of parameters: first, probabilities associated with tag subsequences (single tags, or pairs or triples of tags); second, probabilities associated with word-tag pairs. A trigram HMM would calculate the score for a sentence/tag-sequence pair as something similar to[2]

$$Score(\langle w_1 ... w_n \rangle, \langle t_1 ... t_n \rangle) = \prod_{i=1...n} Score(t_i, t_{i-1}, t_{i-2}) \prod_{i=1...n} Score(w_i, t_i) \tag{2.7}$$

This choice of representation reflects two linguistic assumptions: first, that some sequences of tags are much more frequent than others — i.e. that the parameters $Score(t_i, t_{i-1}, t_{i-2})$ carry useful information; second, that individual words have strong preferences for some tags over others — i.e., that $Score(w_i, t_i)$ also carries useful information.

Having chosen how to break down the structure, the next step is to precisely specify the parameters. In the POS tagging case there are a number of possibilities: $Score(w, t)$ could be a joint probability $P(w, t)$, or one of the conditional probabilities $P(w|t)$ or $P(t|w)$. $Score(t_i, t_{i-1}, t_{i-2})$ could be anything from a joint probability $P(t_i, t_{i-1}, t_{i-2})$ to one of the conditionals $P(t_i|t_{i-1}, t_{i-2})$, $P(t_{i-1}|t_i, t_{i-2})$, or $P(t_{i-2}|t_i, t_{i-1})$.

[2] $t_0$ and $t_{-1}$ are defined as some special $START$ tokens, padding the start of the sentence. We will see soon that an additional term, $P(STOP|t_{n-1}, t_n)$ is also required for the model to be well-formed.

When choosing between these different possibilities, the constraints described in section 2.3.3 (that the model should sum to one, and that the ML estimates should either be derivable in closed form, or by some iterative solution) eliminate many of the alternatives. A standard model is as follows:

- $Score(x, y|\Theta)$ is a joint distribution, with

  $$Score(\langle w_1 ... w_n \rangle, \langle t_1 ... t_n \rangle | \Theta) = P(\text{STOP}|t_n, t_{n-1}) \prod_{i=1...n} P(t_i|t_{i-1}, t_{i-2}) \prod_{i=1...n} P(w_i|t_i) \quad (2.8)$$

  Note there is an additional tag (the STOP symbol) and an associated probability of generating it (we will see later that this extra term is required for $Score$ to be a well-formed distribution).

Providing that the parameter values define conditional probability distributions, this model defines a joint distribution that sums to one. More formally,

**Iff**

1. $\forall t_2, t_3 \in \mathcal{T} \;\; \sum_{t_1 \in \mathcal{T} \cup \text{STOP}} P(t_1|t_2, t_3) = 1$
2. $\forall t \in \mathcal{T} \;\; \sum_{w \in \mathcal{V}} P(w|t) = 1$

**Then** $\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} Score(x, y|\Theta) = 1$

These results follow from a derivation of the equivalence of this model to a particular Hidden Markov Model (HMM). (See section 2.5.)

The next question is how the parameters should be estimated. We will see in section 2.3.2 that maximum-likelihood estimation gives

$$P(t_1|t_2, t_3) = \frac{Count(t_1, t_2, t_3)}{Count(t_2, t_3)} \quad (2.9)$$

$$P(w|t) = \frac{Count(w, t)}{Count(t)} \quad (2.10)$$

(Where $Count(x)$ is the number of times $x$ is seen in the training sample.)

### 2.3.2 Maximum-Likelihood Estimation in Structured Models

Having discussed the design of model structure, we now move on to parameter estimation. Practically all the models described in this chapter have a special form that leads to a simple solution for the maximum-likelihood parameter estimates. These models satisfy two criteria:

1. Say $\mathbf{\Theta} = \{p_1, p_2, ...p_n\}$ is the combination of $m$ multinomial distributions $\Omega_1, \Omega_2, ..\Omega_m$. Each $\Omega_i$ is a subset of the integers $\{1, 2, ..., n\}$ such that the $\Omega$s form a partition of $\{1, 2, 3..n\}$. $\{p_i | i \in \Omega_j\}$ are the parameters of the $j$th multinomial, so that

$$\sum_{i \in \Omega_j} p_i = 1 \tag{2.11}$$

   Let $\Omega^i$ be the multinomial that contains $p_i$.

2. The likelihood of the data can be written

$$L(\mathbf{X}|\mathbf{\Theta}) = \prod_{i \in \Omega_1} p_i^{C(i,\mathbf{X})} \prod_{i \in \Omega_2} p_i^{C(i,\mathbf{X})} ... \prod_{i \in \Omega_m} p_i^{C(i,\mathbf{X})} \tag{2.12}$$

   where $C(i, \mathbf{X})$ is the count of the event which corresponds to $p_i$ in a sample $\mathbf{X}$.

If these conditions are satisfied, maximizing 2.12 subject to the constraints in 2.11 gives maximum-likelihood estimates for each $p_i$ as

$$\hat{p}_{iML} = \frac{C(i, \mathbf{X})}{\sum_{j \in \Omega^i} Count(j, \mathbf{X})} \tag{2.13}$$

**Proof of 2.13**

We first define the log-likelihood function, $L'$:

$$L'(\mathbf{X}|\Theta) = \log L(\mathbf{X}|\Theta) = \log \prod_{i=1}^{n} p_i^{C(i,\mathbf{X})} = \sum_{i=1}^{n} C(i, \mathbf{X}) \log p_i \tag{2.14}$$

Maximizing 2.14 is equivalent to maximizing 2.12, and turns out to be more convenient. The maximization of 2.14 subject to the constraints in 2.11 can be transformed into an unconstrained maximization problem using Lagrange multipliers. We associate $m$ Lagrange multipliers $\lambda_1 ... \lambda_m$ with the $m$ constraints in 2.11. The unconstrained problem is then to maximize

$$\sum_{i=1}^{n} C(i, \mathbf{X}) \log p_i - \sum_{j=1...m} \lambda_j \sum_{k \in \Omega_j} p_k \tag{2.15}$$

Setting the partial derivatives of 2.15 with respect to each $p_i$ equal to zero gives $n$ simultaneous equations:

$$\frac{C(i, \mathbf{X})}{p_i} - \lambda_j = 0 \tag{2.16}$$

where $\lambda_j$ is the Lagrange multiplier associated with the multinomial $\Omega^i$ that contains $p_i$. Equivalently,

$$p_i = \frac{C(i, \mathbf{X})}{\lambda_j} \tag{2.17}$$

Finally, $\lambda_j$ can be eliminated by restating the constraints in 2.11:

$$\begin{aligned} \sum_{k \in \Omega^i} p_k &= 1 \\ \Rightarrow \sum_{k \in \Omega^i} \frac{C(k, \mathbf{X})}{\lambda_j} &= 1 \\ \Rightarrow \lambda_j = \sum_{k \in \Omega^i} C(k, \mathbf{X}) & \qquad (2.18) \end{aligned}$$

From 2.17 and 2.18

$$p_i = \frac{C(i, \mathbf{X})}{\sum_{j \in \Omega^i} C(j, \mathbf{X})} \tag{2.19}$$

**Example**

As an example, the HMM POS tagging model is of this special model form:

1. The parameter vector $\Theta$ consists of two types of parameters: $P(t_1|t_2, t_3)$ where $t_1 \in \mathcal{T} \cup STOP$ and $t_2, t_3 \in \mathcal{T} \cup START$; and $P(w|t)$ where $t \in \mathcal{T}$ and $w \in \mathcal{W}$. Thus $\Theta$ can be separated into $m = (|\mathcal{T}| + 1)^2 + |\mathcal{T}|$ subsets: $(|\mathcal{T}| + 1)^2$ corresponding to the multinomials $P(\cdot|t_2, t_3)$ and $|\mathcal{T}|$ corresponding to the multinomials $P(\cdot|t)$.

2. Say the training sample is $n$ word-sequence/tag-sequence pairs, where the $j$th sequence is of length $l_j$ and is written $\langle w_{j1} ... w_{j(l_j)} \rangle, \langle t_{j1} ... t_{j(l_j)} \rangle$. The likelihood of the $j$th sequence is

$$P(\mathrm{STOP}|t_{j_l}, t_{j_l - 1}) \prod_{i=1...j_l} P(t_{ji}|t_{j(i-1)}, t_{j(i-2)}) \prod_{i=1...j_l} P(w_{ji}|t_{ji}) \tag{2.20}$$

   The likelihood of the training sample is

$$\prod_{j=1...n} \left( P(\mathrm{STOP}|t_{j_l}, t_{j_l - 1}) \prod_{i=1...j_l} P(t_{ji}|t_{j(i-1)}, t_{j(i-2)}) \prod_{i=1...j_l} P(w_{ji}|t_{ji}) \right)$$

$$= \prod_{t_1 \in \mathcal{T} \cup STOP,\ t_2, t_3 \in \mathcal{T} \cup START} P(t_1|t_2,t_3)^{Count(t_1,t_2,t_3)} \prod_{t \in \mathcal{T},\ w \in \mathcal{V}} P(w|t)^{Count(w,t)} \tag{2.21}$$

$Count(t_1, t_2, t_3)$ is the number of times the parameter $P(t_1|t_2, t_3)$ is seen in the first product, i.e. the number of times the sequence $\langle t_1, t_2, t_3 \rangle$ is seen in training data. $Count(w, t)$ is the number of times the pair $P(w|t)$ is seen in the first product, i.e. the number of times $\langle w, t \rangle$ is seen in training data.

The general result in 2.13 can now be applied to give the familiar maximum-likelihood solutions for the parameter values:

$$\hat{P}_{ML}(t_1|t_2,t_3) = \frac{Count(t_1,t_2,t_3)}{\sum_{t_1 \in \mathcal{T} \cup STOP} Count(t_1,t_2,t_3)} = \frac{Count(t_1,t_2,t_3)}{Count(t_2,t_3)} \tag{2.22}$$

$$\hat{P}_{ML}(w|t) = \frac{Count(w,t)}{\sum_{w \in \mathcal{V}} Count(w,t)} = \frac{Count(w,t)}{Count(t)} \tag{2.23}$$

### 2.3.3 Two Conditions for Model Structures

In the POS tagging example we suggested the following model (equation 2.8, repeated here):

$$Score(\langle w_1...w_n \rangle, \langle t_1...t_n \rangle) = P(\text{STOP}|t_n, t_{n-1}) \prod_{i=1...n} P(t_i|t_{i-1}, t_{i-2}) \prod_{i=1...n} P(w_i|t_i) \tag{2.24}$$

In defining the parameters, and thereby the model structure, we hinted that 2.24 was one of the few possibilities. But why, for example, couldn't we instead choose a model such as the following?

$$Score(\langle w_1...w_n \rangle, \langle t_1...t_n \rangle) = P(\text{STOP}|t_n, t_{n-1}) \prod_{i=1...n} P(t_i|t_{i-1}, t_{i-2}) \prod_{i=1...n} P(t_i|w_i) \tag{2.25}$$

In a sense, the parameter type $P(t_i|w_i)$ is more intuitive than $P(w_i|t_i)$. In fact, one of the earliest papers on Markov taggers [Church 88] used 2.25 rather than 2.24.

As one criterion for the choice between competing models, we specify two conditions for a model structure to be "well-formed":

1. The model structure $Score(x, y|\Theta)$ for all values $\Theta \in \Omega$ should be either a conditional probability $P(y|x)$ (satisfying the constraint $\forall x \in \mathcal{X}\ \sum_{y \in \mathcal{Y}} Score(y, x) = 1$) or a joint probability $P(x, y)$ (satisfying the constraint $\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} Score(x, y) = 1$).

2. The maximum-likelihood estimates for the model should be derivable in closed form, or by some iterative solution (EM estimation and Maximum Entropy models both use iterative re-estimation techniques). Often this means that the model is of the form described in section 2.3.2, with the maximum likelihood estimate of $P(a|b)$ then being $\frac{Count(a,b)}{Count(b)}$.

   This second point is important because MLE's will be central to the estimation methods that we use. Even if we don't directly use MLE's, instead using one of the more robust methods described in section 2.9, the parameter estimates will generally be derived as a refinement of maximum-likelihood estimates.

Condition 1 alone is trivial to satisfy. Say, for example, we would like to define a joint distribution, and that we have a function $Score(x, y|\Theta)$ that is unfortunately *not* a distribution, i.e. $\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} Score(y, x|\Theta) \neq 1$. It is straightforward to define a function $Score'$ that *is* a distribution: if we define a normalization factor $Z(\Theta) = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} Score(y, x|\Theta)$ then $Score'(x, y|\Theta) = \frac{Score(x,y|\Theta)}{Z(\Theta)}$ is a distribution[3].

The problem with normalizing distributions in this way comes from condition 2. The maximum-likelihood estimate for a training sample $\langle x_1, y_1 \rangle ... \langle x_n, y_n \rangle$ is defined as

$$\hat{\Theta}_{ML} = \arg\max_{\Theta \in \Omega} \prod_{i=1...n} Score'(x_i, y_i|\Theta) = \arg\max_{\Theta \in \Omega} \prod_{i=1...n} \frac{Score(x_i, y_i|\Theta)}{Z(\Theta)} \tag{2.26}$$

$Z(\Theta)$ must be taken into account when maximizing this product, and may greatly complicate the process, often blocking a closed-form solution for the ML estimates. The addition of $Z(\Theta)$ means that the model is not of the type defined in section 2.3.2, and that the familiar relative frequency estimate $P(a|b) = \frac{Count(a,b)}{Count(b)}$ will most likely *not* be a maximum-likelihood estimate.

We can now return to the choice between 2.24 and 2.25 as models for POS tagging. Model form 2.24 sums to 1 as it stands, is of the form described in section 2.3.2, and

[3]When searching for the most likely output $y_{best}$ for some input $x$ under the distribution $Score'$, $Z(\Theta)$ will not complicate the process as it is a constant that can be ignored when ranking alternatives, and thus it does not need to be explicitly calculated.

therefore has ML estimates of the form $P(a|b) = \frac{Count(a,b)}{Count(b)}$. Model form 2.25 does not sum to 1 without a normalizing factor: with the normalizing factor the ML estimates are very difficult to derive (and probably don't even exist in closed form). The method in [Church 88] — the use of model form 2.25 with estimates derived from relative frequency estimates — cannot be justified as maximum-likelihood estimation. These theoretical deficiencies have been shown to give decreased performance of the model on real tasks: [Charniak et al. 93] reports accuracies of 96% vs. 95% for models 2.24 and 2.25 respectively on a POS tagging task.

### 2.3.4 Summary

In summary:

- The first task when designing a model is to choose how to break the members of the input-output space $\mathcal{X} \times \mathcal{Y}$ into smaller sub-events. For example, in the case of POS tagging, we chose to associate parameters with tag trigrams and word-tag pairs.

- The second task is to exactly specify the parameters in the model. In the POS tagging case, the parameters were $P(t_i|t_{i-1}, t_{i-2})$ and $P(w|t)$. When making this choice there are two guiding constraints: (1) that the overall model should define either a joint or conditional probability distribution over the space $\mathcal{X} \times \mathcal{Y}$ [4]; (2) that the maximum-likelihood estimates should be derivable in closed form, or by some iterative solution.

  Defining models that satisfy these criteria for complex event spaces is difficult, and is the major concern of this chapter. We will describe a number of techniques, in increasing order of generality, that can be applied to this type of problem: Markov models for probabilities over sequences; Hidden Markov Models for probabilities over sequence/state-sequence pairs (as in POS tagging); Probabilistic Context-Free Grammars for probabilities over sentence-tree pairs; and finally "history-based" models as a generalization of the previous model types.

[4] i.e. that the $Score$ associated with each member of $\mathcal{X} \times \mathcal{Y}$ gives either a joint distribution satisfying $\sum_{\substack{x \in \mathcal{X} \\ y \in \mathcal{Y}}} Score(x, y) = 1$, or a conditional distribution satisfying $\forall x \in \mathcal{X} \sum_{y \in \mathcal{Y}} Score(x, y) = 1$.

- The third task is that of *estimation*: given a training example and a model structure, how to calculate parameter values. For each of the model types described in this chapter we derive the maximum-likelihood estimate of the parameters. Section 2.9 describes more sophisticated estimation methods that smooth higher-order maximum-likelihood estimates with lower-order estimates.

## 2.4 Defining Sentence Probabilities Using Markov Processes

This section describes Markov models as a method for defining probabilities over sequences of symbols. These models are used in the parser of chapter 7, so we will discuss them quite extensively here. They are also the basis of Hidden Markov Models, described in the next section. Consider the following situation:

- $\mathcal{V}$ is a vocabulary, a set of words in a language.
- A sentence drawn from this vocabulary is a sequence $W = \langle w_1, w_2 ... w_n \rangle$ where $n \geq 0$, and $w_i \in \mathcal{V}$.
- We name the (infinite) set of all possible sentences $\zeta$ (this is the event space).

We would like to define a distribution $P$ over the event space $\zeta$. A simple way to define this probability distribution is to assume that the sentences of the language are generated by a Markov process. Our first step is to add a special $STOP$ symbol to the vocabulary, and to define $w_{n+1}$ in any sentence $W$ as this $STOP$ symbol (section 2.4.1 explains why this is necessary). The probability of any sentence can then be written using the chain rule of probabilities:

$$P(W) = P(\langle w_1 ... w_{n+1} \rangle) = \prod_{i=1...n+1} P(w_i \mid w_1 ... w_{i-1}) \tag{2.27}$$

The next step is to make an $m$th order Markov assumption: that the probability of a symbol depends only on the previous $m$ symbols ($w_0$ and $w_{-1}$ are taken to be some special START symbol):

$$\begin{aligned} P(w_i \mid w_1 ... w_{i-1}) &= P(w_i \mid w_{i-m} ... w_{i-1}) && (2.28) \\ P(W) &= \prod_{i=1...n+1} P(w_i \mid w_{i-m} ... w_{i-1}) && (2.29) \end{aligned}$$

For example, in the standard trigram model used in speech recognition [Jelinek 90], a 2nd order Markov assumption is used:

$$P(w_i \mid w_1 ... w_{i-1}) = P(w_i \mid w_{i-2} ... w_{i-1}) \tag{2.30}$$

$$P(W) = \prod_{i=1...n+1} P(w_i \mid w_{i-2} ... w_{i-1}) \tag{2.31}$$

An $m$th order Markov model requires $|\mathcal{V}|^{m+1}$ parameters. Providing that the parameters of the Markov process give well defined distributions, this model defines a probability distribution over the possible sentences in the language. More formally:

$$\left( \forall x, y \in \mathcal{V} \sum_{w \in \mathcal{V} \cup STOP} P(w \mid x, y) = 1 \right) \Longleftrightarrow \sum_{W \in \zeta} P(W) = 1 \tag{2.32}$$

(This is a slight simplification. There are actually further constraints on the parameter values for the model to sum to 1. Some parameter settings can lead to probability mass being lost to infinite length sequences. For example, a self-looping probability $P(a|a) = 1$ in a first order Markov chain will mean that there is some probability of never generating the STOP symbol. See for example [Thomason 86] pages 126–128 for conditions that exclude parameter values that lead to these problems.)

### 2.4.1 The Importance of the STOP Symbol

At first glance the addition of the $STOP$ symbol seems rather unnatural. This section gives justification for its inclusion. The suspicion that the $STOP$ symbol might be extraneous is compounded by many, perhaps a majority, of the references in language modeling for speech recognition omitting it[5]. The usual derivation in these descriptions of language modeling goes as follows. First, rewrite the probability of a word sequence using the chain rule of probabilities:

$$P(w_1, w_2, w_3 ... w_n) = \prod_{i=1...n} P(w_i | w_1 ... w_{i-1}) \tag{2.33}$$

Second, make Markov independence assumptions:

$$\prod_{i=1...n} P(w_i | w_1 ... w_{i-1}) = \prod_{i=1...n} P(w_i | w_{i-1} ... w_{i-m}) \tag{2.34}$$

[5] Maybe with good reason: either to considerably simplify the description at the cost of a small decrease in accuracy; or because they assume that the speech stream will not be decoded sentence by sentence, but instead as a continuous stream of potentially infinite length.

So what is the problem with this derivation?

The key point is that $n$, the sentence length, is variable. Equation 2.33 would be correct if the event space under consideration was the space of $n$-dimensional vectors $\mathcal{V}^n$: but the event space is instead the set of all strings in the language, $\mathcal{V}^*$. Writing the probability under consideration as $P(w_1, w_2, w_3...w_n)$ implies that $\mathcal{V}^n$ is the event space. To avoid this confusion we will write the probability of a sequence $\langle w_1, w_2, ...w_n\rangle$ as $P(\langle w_1, w_2, ...w_n\rangle)$: the angled braces imply that $\langle w_1, w_2, ...w_n\rangle$ is a sequence of variable length rather than an $n$-dimensional vector.

This criticism may seem pedantic, particularly in the case of speech recognition, where $n$ is often large and the $STOP$ probabilities may not be too significant. (In fact, if speech is not decoded sentence by sentence but is instead decoded as one steady stream then $n$ may become *very* large and the $STOP$ probabilities will become irrelevant: the language model becomes a Markov process that has zero probability of halting.) However, in our use of Markov processes in chapter 7 the sequences under consideration are typically of length 0, 1 or 2, and the $STOP$ probabilities are certainly important.

It is easy to give an example that illustrates the failings of equation 2.34:

- Assume $\mathcal{V} = \{a, b\}$, and therefore that $\zeta$ is $\{\epsilon$, `a`, `b`, `aa`, `bb`, `ab`, `bb` ...$\}$.
- Assume that we will model the probability over $\zeta$ with a 0'th order Markov process, with parameters $P(a) = P(b) = 0.5$.

We can now calculate the probability of several strings using the formula in equation 2.34: $P(\langle a\rangle) = 0.5$, $P(\langle b\rangle) = 0.5$, $P(\langle aa\rangle) = 0.5^2 = 0.25$, $P(\langle bb\rangle) = 0.25$ and so on. We already see from these 4 probabilities that the sum over the event space will be greater than 1: $P(\langle a\rangle) + P(\langle b\rangle) + P(\langle aa\rangle) + P(\langle bb\rangle) = 1.5$! An additional problem is that the probability of the empty string, $P(\langle\rangle)$, where $n = 0$, is undefined.

Now assume that we add the stop symbol, with the parameters of the Markov process modified to include this: e.g., $P(a) = P(b) = 0.25, P(STOP) = 0.5$. In this case we have $P(\langle STOP\rangle) = 0.5$, $P(\langle aSTOP\rangle) = 0.25 * 0.5 = 0.125$, $P(\langle bSTOP\rangle) = 0.25 * 0.5 = 0.125$, $P(\langle aaSTOP\rangle) = 0.25^2 * 0.5 = 0.03125$, $P(\langle bbSTOP\rangle) = 0.03125$ and so on. Thus far the sum of probabilities does not exceed 1, and the distribution is looking much better behaved. We can prove that the sum over all sequences is 1 by noting that the probability of any sequence of length $n$ is $0.25^n * 0.5$, and that there are $2^n$ sequences of length $n$, giving:

$$\begin{aligned} \sum_{W \in \zeta} P(W) &= \sum_{n=0}^{\infty} 2^n * 0.25^n * 0.5 \\ &= \sum_{n=0}^{\infty} 0.5^n * 0.5 \\ &= \sum_{n=0}^{\infty} 0.5^{n+1} \\ &= \sum_{n=1}^{\infty} 0.5^n \\ &= 1 \end{aligned} \tag{2.35}$$

In a 0'th order Markov process the distribution over lengths of strings is related directly to $P(STOP)$ — the probability of a string having length $n$ is the probability of generating $n$ non-$STOP$ symbols followed by the $STOP$ symbol:

$$P(length = n) = (1 - P(STOP))^n * P(STOP) \tag{2.36}$$

With higher order Markov processes, where the probability is conditioned on previously generated symbols, the conditional probability $P(STOP \mid w_{i-m}...w_{i-1})$ encodes the preference for certain symbols or sequences of symbols to end or not to end a sentence. For example, if we were building a bigram (1st order Markov) model of English we would expect the word *the* to end a sentence very rarely, and the corresponding parameter $P(STOP \mid the)$ to be very low. Without the $STOP$ symbol these kind of facts will not be encoded in the parameters. So we see that the $STOP$ symbol not only ensures that the probability distributions are well defined, but that it can also have a useful interpretation.

## 2.5 Defining Tagged-Sentence Probabilities Using Hidden Markov Processes

This section considers the definition of probability distributions over pairs of sequences. In general:

- The input space $\mathcal{X}$ is a set of sequences $\langle w_1 ... w_n \rangle$ where each $w_i$ is drawn from a set of "words" $\mathcal{V}$ (we refer to these sequences as word sequences, or sentences).
- The output space $\mathcal{Y}$ is a set of sequences $\langle t_1 ... t_n \rangle$ where each $t_i$ is drawn from a set of "tags" $\mathcal{T}$ (we refer to these as tag sequences).
- In order to define the mapping $f : \mathcal{X} \to \mathcal{Y}$ we define a distribution $Score : \mathcal{X} \times \mathcal{Y} \to [0, 1]$. Hidden Markov Models (HMMs), the topic of this section, can be used to define a joint probability $P(x, y|\Theta)$ over $\mathcal{X} \times \mathcal{Y}$.

As a first step in defining $P(x, y|\Theta)$, we break the probability into two terms:

$$P(\langle w_1, w_2 ... w_n \rangle, \langle t_1, t_2 ... t_n \rangle) = P(\langle t_1, t_2 ... t_n \rangle) \times P(\langle w_1, w_2 ... w_n \rangle \mid \langle t_1, t_2 ... t_n \rangle) \tag{2.37}$$

(This step is exact, being a direct consequence of the definition of conditional probability.) The two terms are then modeled separately. The probability distribution over tag sequences is defined using an $m$th order Markov model:

$$P(\langle t_1, t_2 ... t_n \rangle) = P(STOP \mid t_{n-m+1} ... t_n) \prod_{i=1...n} P(t_i | t_{i-m} ... t_{i-1}) \tag{2.38}$$

From the results in the previous section 2.38 defines a well-formed distribution over tag sequences providing that $\forall t_1 ... t_m \in \mathcal{T} \;\; \sum_{t \in \mathcal{T} \cup \{STOP\}} P(t | t_1 ... t_m) = 1$.

The second term is simplified by first using the chain rule, then by making the independence assumption that each word depends only on its corresponding tag:

$$\begin{aligned} P(\langle w_1, w_2 ... w_n \rangle \mid \langle t_1, t_2 ... t_n \rangle) &= \prod_{i=1...n} P(w_i \mid \langle w_1, w_2 ... w_{i-1} \rangle, \langle t_1, t_2 ... t_n \rangle) \\ &= \prod_{i=1...n} P(w_i \mid t_i) \end{aligned} \tag{2.39}$$

Substituting 2.38 and 2.39 in 2.37 gives the equation for an $m$th order HMM:

$$P(\langle w_1, w_2 ... w_n \rangle, \langle t_1, t_2 ... t_n \rangle) = P(STOP \mid t_{n-m+1} ... t_n) \prod_{i=1...n} P(t_i | t_{i-m} ... t_{i-1}) \prod_{i=1...n} P(w_i \mid t_i) \tag{2.40}$$

The property that this definition sums to 1 over the space $\mathcal{X} \times \mathcal{Y}$ comes directly from the fact that 2.38 and 2.39 sum to 1 over their respective event spaces (i.e., 2.38 sums to 1 over $\mathcal{Y}$, 2.39 sums to 1 over $\mathcal{X}$). The trigram tagger in 2.8 is a 2nd order HMM, so we have finally proved that it is a model that sums to 1 over its event space. Section 2.3.2 showed that the maximum-likelihood parameter estimates are

$$\begin{aligned}\hat{P}_{ML}(t|t_1...t_m) &= \frac{Count(t, t_1...t_m)}{Count(t_1...t_m)} \\ \hat{P}_{ML}(w|t) &= \frac{Count(w, t)}{Count(t)}\end{aligned} \tag{2.41}$$

An efficient algorithm for search for the highest probability tag sequence for a particular sentence — the Viterbi algorithm — exists for HMMs (e.g., see [Charniak 93]).

## 2.6 Probabilistic Context Free Grammars (PCFGs)

We now describe the use of Probabilistic Context Free Grammars (PCFGs) for modeling distributions over sentence/parse-tree pairs. The theory behind PCFGs will be important for the parsing models in chapter 7, so this section describes them in some detail: first giving basic definitions, next defining search algorithms, and finally deriving expressions for maximum-likelihood parameter estimates.

Probabilistic learning for parsing can be defined as follows:

- The input space $\mathcal{X}$ is a set of sequences $\langle w_1...w_n \rangle$ where each $w_i$ is drawn from a set of "words" $\mathcal{V}$ (we refer to these sequences as word sequences, or sentences).

- The output space $\mathcal{Y}$ is a set of parse trees generated by some context-free grammar. Each parse tree spans a member of $\mathcal{V}^*$ (i.e., each tree in $\mathcal{Y}$ has a member of $\mathcal{X}$ as its sequence of terminal symbols).

- In order to define the mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ we will define a distribution $Score$ : $\mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$. PCFGs can be used to define a joint probability $P(x, y|\Theta)$ over the space of possible sentence/parse-tree pairs.

### 2.6.1 Formal Definitions

A context-free grammar (e.g., see [Hopcroft and Ullman 79]) is usually defined as a 4-tuple G = (N, $\Sigma$, $P$, $S$) where

- N is a finite set of non-terminal symbols.

- $\Sigma$ is a finite set of terminal symbols.

- $P$ is a finite set of productions or rewriting rules. The rules in $P$ take the form $\alpha \to \beta$ where $\alpha \in N$ and $\beta \in \{N \cup \Sigma\}^*$.

- $S \in N$ is the starting symbol.

A probabilistic context-free grammar (PCFG) additionally has a probability associated with each rule in the grammar. We will write the probability associated with rule $\alpha \to \beta$ as $\mathcal{P}(\alpha \to \beta|\alpha)$. It is interpreted as the conditional probability of choosing the rule $\alpha \to \beta$, given that $\alpha$ is the non-terminal being rewritten in a derivation. If $D$ is a function assigning a probability to each member of $P$, a PCFG is a 5-tuple G = (N, $\Sigma$, $P$, $S$, $D$).

Given a PCFG, the probability for any context-free tree in the language is the product of probabilities for the rules that it contains. That is, if $T$ is a context-free derivation that involves $n$ rules of the form $\alpha_i \to \beta_i$,

$$P(T) = \prod_{i=1...n} \mathcal{P}(\alpha_i \to \beta_i|\alpha_i) \tag{2.42}$$

A PCFG also defines a probability distribution over strings. If $\mathcal{T}(S)$ is the set of trees whose surface string is $S$, then

$$P(S) = \sum_{T \in \mathcal{T}(S)} P(T) \tag{2.43}$$

Perhaps more importantly, given that we are attempting to define parsing models, a PCFG also defines the most likely tree for each string S, $T_{best}(S)$:

$$T_{best}(S) = \arg \max_{T \in \mathcal{T}(S)} P(T) \tag{2.44}$$

### 2.6.2 Conditions for Consistency

If $\mathcal{T}$ is the set of all possible trees in the context-free language underlying the PCFG, and $\mathcal{S}$ is the set of strings in the context-free language, then a PCFG should define probability distributions over the sets of possible trees and strings. That is,

$$\forall T \in \mathcal{T} \qquad \mathcal{P}(T) \geq 0 \tag{2.45}$$

$$\sum_{T \in \mathcal{T}} \mathcal{P}(T) \;=\; 1 \tag{2.46}$$

$$\forall S \in \mathcal{S} \qquad \mathcal{P}(S) \geq 0 \tag{2.47}$$

$$\sum_{S \in \mathcal{S}} \mathcal{P}(S) \;=\; 1 \tag{2.48}$$

Note that conditions 2.46 and 2.48 are equivalent ([Booth and Thompson 73] give conditions for 2.48 to be true, which we will take to also imply that 2.46 is true):

$$\sum_{S \in \mathcal{S}} \mathcal{P}(S) = \sum_{S \in \mathcal{S}} \sum_{T \in \mathcal{T}(S)} \mathcal{P}(T) = \sum_{T \in \mathcal{T}} \mathcal{P}(T) \tag{2.49}$$

Conditions 2.45 and 2.47 follow from all rule probabilities $\mathcal{P}(\alpha \rightarrow \beta|\alpha)$ being $\geq 0$.

[Booth and Thompson 73] prove that two conditions are required for what they call *consistency* (i.e., condition 2.48, and by implication 2.46):

1. For all non-terminals $\alpha \in N$,

   $$\sum_{(\alpha \rightarrow \beta) \in P} \mathcal{P}(\alpha \rightarrow \beta|\alpha) = 1 \tag{2.50}$$

   Thus consistency is the motivation for the probability associated with $\alpha \rightarrow \beta$ being $\mathcal{P}(\alpha \rightarrow \beta|\alpha)$, rather than another possibility such as a joint $\mathcal{P}(\alpha \rightarrow \beta)$, or the other conditional $\mathcal{P}(\alpha \rightarrow \beta|\beta)$.

   [Booth and Thompson 73] called PCFGs fulfilling this condition *proper*.

2. [Booth and Thompson 73] section V gives further conditions on the parameter values for consistency. This rules out problematic PCFGs such as a grammar that has a self-looping rule `NP` $\rightarrow$ `NP` with probability 1. In this case the self-loop means that some probability mass is lost to derivations that never terminate. Surprisingly, even grammars without self-loops can lose some probability mass to derivations that never terminate: for example, a PCFG with two rules `S` $\rightarrow$ `S S` (probability = 0.6) and `S` $\rightarrow$ `a` (probability = 0.4) has this problem.

Figure 2.1 gives an example PCFG, which is proper (and consistent). Figure 2.2 gives an example tree with its associated probabilities.

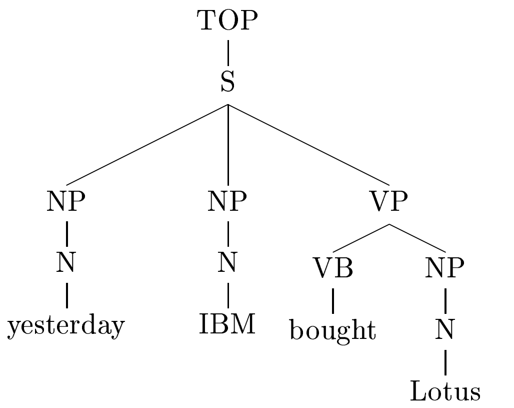
$N = \{\text{TOP, S, NP, VP, VB, NNP}\}$

$\Sigma = \{\text{gave, saw, U.S., IBM, yesterday}\}$

$S = \text{TOP}$

| Rules $P$ | Probabilities $D$ |
|---|---|
| `TOP` ⇒ `S` | 1.0 |
| `S` ⇒ `NP VP` | 0.8 |
| `S` ⇒ `NP NP VP` | 0.2 |
| `VP` ⇒ `VB NP` | 0.6 |
| `VP` ⇒ `VB NP NP` | 0.4 |
| `NP` ⇒ `N` | 1.0 |
| `VB` ⇒ gave | 0.6 |
| `VB` ⇒ bought | 0.4 |
| `N` ⇒ Lotus | 0.8 |
| `N` ⇒ IBM | 0.1 |
| `N` ⇒ yesterday | 0.1 |

Figure 2.1: A PCFG




$$
\begin{array}{lll}
\texttt{Probability =} & P(\texttt{TOP} \rightarrow \texttt{S}|\texttt{TOP}) & \times P(\texttt{N} \rightarrow \mathit{yesterday}|\texttt{N}) \\
& \times P(\texttt{S} \rightarrow \texttt{NP NP VP}|\texttt{S}) & \times P(\texttt{N} \rightarrow \mathit{IBM}|\texttt{N}) \\
& \times P(\texttt{VP} \rightarrow \texttt{VB NP}|\texttt{VP}) & \times P(\texttt{V} \rightarrow \mathit{bought}|\texttt{V}) \\
& \times P(\texttt{NP} \rightarrow \texttt{N}|\texttt{NP}) & \times P(\texttt{N} \rightarrow \mathit{Lotus}|\texttt{N}) \\
& \times P(\texttt{NP} \rightarrow \texttt{N}|\texttt{NP}) & \\
& \times P(\texttt{NP} \rightarrow \texttt{N}|\texttt{NP}) &
\end{array}
$$

Figure 2.2: A context-free tree, and its associated probability.

### 2.6.3 Search for the Highest Probability Tree

The search problem is to find the most likely tree, $T_{best}(S)$, for an input string $S$, where

$$T_{best}(S) = \arg\max_{T \in \mathcal{T}(S)} P(T) \tag{2.51}$$

One of the simplest methods is an extension of the CKY algorithm [Hopcroft and Ullman 79] — originally developed for context-free grammars — to PCFGs. The CKY algorithm assumes that the input grammar is in Chomsky Normal Form (CNF). In a CNF grammar every rule takes one of two forms:

- A $\rightarrow$ B C, where A, B, C are in $N$.
- A $\rightarrow$ a, where A is in $N$, a is in $\Sigma$.

For now we will assume that the PCFGs under consideration are in CNF.

The CKY algorithm for PCFGs is a dynamic programming algorithm that runs in $O(n^3|N|^3)$ time where $n$ is the number of words in the input sentence, $|N|$ is the number of non-terminals in the grammar. We assume the following data structures:

- Input: $n$ words $w_1 ... w_n$
- We assume that the $|N|$ non-terminals in the grammar have indices $1, 2, ...|N|$. Without loss of generality we take the first non-terminal to be the starting symbol.
- The central data structure is a dynamic programming array: $\pi[i, j, k]$ holds the maximum probability for a constituent with label $k$ spanning words $i...j$.
- The goal of the search is then to find $\pi[1, n, 1]$ (the maximum probability for any tree spanning the whole sentence, rooted in the starting symbol). Back-pointers in the dynamic programming array can be used to store the path leading to this goal (the highest probability tree).

The basis of the CKY algorithm is a recursive definition of the dynamic programming array:

- Base case:

for $i = 1...n, k = 1...|N|$,

if $k \rightarrow w_i$ is a rule in the grammar $\pi[i, i, k] = P(k \rightarrow w_i|k)$

else $\pi[i, i, k] = 0$

- Recursive case:

  $\pi[i, j, k] = \max\{\pi[i, m, k_1] * \pi[m + 1, j, k_2] * P(k \rightarrow k1 \;\; k2|k)\}$

  where the maximum is taken over $m$ such that $i \leq m \leq j - 1$ and $k1, k2$ such that $k \rightarrow k1 \;\; k2$ is in the grammar

Figure 2.3 gives pseudo code for an implementation of the algorithm. The only subtle point in the implementation is that when building a constituent of length $l$, all subconstituents of length less than $l$ must have already been built. The variable `s` (for span) in the pseudo-code ensures that constituents of length 2, 3, 4 ... n are built in that order.

### 2.6.4 Parameter Estimation

The next question is how to estimate the rule probabilities $\mathcal{P}(\alpha \rightarrow \beta|\alpha)$, given a training corpus. Assume that the training corpus consists of $n$ trees, $T_1...T_n$. Assume that each $T_i$ contains $r_i$ context-free rules, $\alpha_{ij} \rightarrow \beta_{ij}$ for $1 \leq j \leq r_i$. The likelihood of the corpus can be written as

$$\begin{aligned} L(corpus) &= \prod_{i=1...n} \mathcal{P}(T_i) \\ &= \prod_{i=1...n} \prod_{j=1...r_i} \mathcal{P}(\alpha_{ij} \rightarrow \beta_{ij}|\alpha_{ij}) \\ &= \prod_{(\alpha \rightarrow \beta) \in P} \mathcal{P}(\alpha \rightarrow \beta|\alpha)^{Count(\alpha \rightarrow \beta)} \end{aligned} \tag{2.52}$$

($Count(\alpha \rightarrow \beta)$ is the number of times the rule $\alpha \rightarrow \beta$ is seen in the first product: I.e., the number of times the rule is seen in training data.) This likelihood function shows that the model is of the general form described in section 2.3.2. If $\beta(\alpha)$ is the set $\{\beta \mid (\alpha \rightarrow \beta) \in P\}$ then it follows that the maximum-likelihood parameter estimates are

$$\mathcal{P}(\alpha \rightarrow \beta|\alpha) = \frac{Count(\alpha \rightarrow \beta)}{\sum_{\gamma \in \beta(\alpha)} Count(\alpha \rightarrow \gamma)} = \frac{Count(\alpha \rightarrow \beta)}{Count(\alpha)} \tag{2.53}$$

```
#initialisation
for all i,j,k
   p[i,j,k] = 0

#base case

for i = 1 ... n
   for k = 1 ... G
      if k -> wi is in grammar
         p[i,i,k] = P(k -> wi)

#recursive case

for s = 2 ... n
for i = 1 ... n-s+1
j = i+s-1
for m = i ... j-1
    for k = 1 ... G
    for k1 = 1 ... G
    for k2 = 1 ... G

    prob = p[i,m,k1] * p[m+1,j,k2] * P(k -> k1 k2)
    if (prob > p[i,j,k])

       p[i,j,k] = prob
       B[i,j,k] = {m,k1,k2}
```

Figure 2.3: Pseudo-code for the CKY algorithm for PCFGs. `p` is the dynamic programming array. `B` is an array of back-pointers allowing recovery of the highest probability tree.

## 2.7 History-Based Models

[Black et al. 92b] introduced what they called "history-based" models to natural language processing; later work such as [Jelinek et al. 94, Magerman 95, Ratnaparkhi 96, Ratnaparkhi 97] used this for parsing and tagging problems. The idea is to define a one-to-one mapping that maps each member of $\mathcal{X} \times \mathcal{Y}$ to a sequence of decisions $\langle d_1, d_2 ... d_n \rangle$. The joint probability of a member $(x, y)$ of $\mathcal{X} \times \mathcal{Y}$ is then written using the chain rule of probabilities as

$$P(x, y) = P(\langle d_1, d_2 ... d_n \rangle) = \prod_{i=1...n} P(d_i | d_1 ... d_{i-1}) \tag{2.54}$$

The conditioning context for each $d_i$, $\langle d_1, d_2 ... d_{i-1} \rangle$, is referred to as the "history", and is equivalent to some partially built structure.

The mapping between events in $\mathcal{X} \times \mathcal{Y}$ and decision sequences is achieved by defining a stochastic program that generates events in $\mathcal{X} \times \mathcal{Y}$. A stochastic program is an algorithm which at certain points makes a random choice between alternative decisions, according to some probability distribution [Koller, McAllester and Pfeffer 97]. The trace of the program can be represented as the sequence of decisions that is made; the probability of this decision sequence is the product of probabilities of the different decisions. If the program's probability of halting is 1,[6] then the program defines a distribution over decision sequences, and — given that there is one-to-one mapping from members of $\mathcal{X} \times \mathcal{Y}$ to decision sequences — it defines a distribution over $\mathcal{X} \times \mathcal{Y}$.

As it stands, associating a parameter $P(d_i | d_1 ... d_{i-1})$ with each possible prefix $\langle d_1, d_2 ... d_i \rangle$ would lead to a vast number of parameters. [Black et al. 92b] describe the use of a function $\Phi$ to group histories into equivalence classes, giving

$$P(x, y) = P(\langle d_1, d_2 ... d_n \rangle) = \prod_{i=1...n} P\left(d_i | \Phi(d_1 ... d_{i-1})\right) \tag{2.55}$$

As an example, figure 2.4 gives pseudo-code for a stochastic program that generates tree-sentence pairs. In this case the stochastic program generates left-most derivations of parse trees: a tree is represented as a sequence of decisions where each decision is the rule used to expand the left-most non-terminal at the current point in the derivation.

[6] Note that this property can often be quite difficult to prove.

For example, [S [NP [N I]] [VP [VB saw] [NP [N her]]]] would be represented as $\{\texttt{S} \rightarrow \texttt{NP VP},\ \texttt{NP} \rightarrow \texttt{N},\ \texttt{N} \rightarrow \texttt{I},\ \texttt{VP} \rightarrow \texttt{VB NP},\ \texttt{VB} \rightarrow \texttt{saw},\ \texttt{NP} \rightarrow \texttt{N},\ \texttt{N} \rightarrow \texttt{her}\}$.

If $\Phi(d_1...d_{i-1})) = \alpha$, where $\alpha$ is the left-most non-terminal in the partial tree defined by $\langle d_1...d_{i-1}\rangle$, then the stochastic program is equivalent to a PCFG, in that it generates trees with the distribution defined by a PCFG with parameters $P(\alpha \rightarrow \beta|\alpha)$. On the other hand, $\Phi$ could be extended to include arbitrary additional context in the partial tree defined by $d_1...d_{i-1}$ (for example, the parent of $\alpha$ in the tree, the symbol directly to the left of $\alpha$ in the tree, and so on). [Black et al. 92b] describe a method that uses decision trees to search for values of $\Phi$ that include additional context. So we see that while this history-based model includes PCFGs, it is also powerful enough to extend them in many ways.

For completeness, figures 2.5 and 2.6 give pseudocode for stochastic programs that generate sequences and sequence pairs, and states conditions for their equivalence to Markov and Hidden Markov Models respectively.

### 2.7.1 Conditional History-Based Models

History-based models can also be used to define conditional distributions $P(y|x)$ for $y \in \mathcal{Y}$ and $x \in \mathcal{X}$. [Ratnaparkhi 97, Magerman 95, Jelinek et al. 94] describe conditional models for parsing; [Ratnaparkhi 96] describes such a model for POS tagging. In conditional models the pair $x, y$ is again represented as a sequence of decisions, but the input $x$ is a conditioning variable:

$$P(y|x) = P(\langle d_1, d_2...d_n\rangle|x) = \prod_{i=1...n} P(d_i|\Phi(d_1...d_{i-1}, x)) \qquad (2.56)$$

For example, in [Ratnaparkhi 96], $d_1...d_n$ are the $n$ POS tags for a sentence of length $n$, and $\Phi(d_1...d_{i-1}, x)$ picks out the previous two tags, $d_{i-1}$ and $d_{i-2}$, as well as the words $w_{i-2}...w_{i+2}$. Chapter 4 describes the use of conditional history-based models for parsing [Magerman 95, Ratnaparkhi 97, Jelinek et al. 94] in some detail.

```
#d_i is the i'th decision made by the program -- each decision is the choice
#to use some rewrite rule from the grammar

#s_i is the i'th sentential form derived in a left-recursive derivation of
#the tree. s_0 is START (START is the starting symbol in the grammar);
#s_i can be derived from s_0 and d_1 ... d_n
```

$s_0$ = START;
i = 0;

while($s_i$ contains at least one non-terminal)
{

$\alpha$ = left-most non-terminal in $s_i$;
$d_i$ = choose a rewrite rule $\alpha \rightarrow \beta$ from the distribution $P(\alpha \rightarrow \beta | \Phi(d_1 ... d_{i-1}))$;

$s_{i+1} = s_i$ with $\alpha$ replaced by $\beta$;
i++;
}
return;

Figure 2.4: A stochastic program that generates trees. If $\Phi(d_1...d_{i-1}) = \alpha$ then this is equivalent to a PCFG.

```
#d_i is the i'th decision made by the program -- each decision is the choice
#to generate either a symbol from the vocabulary V, or the STOP symbol
```

i = 1;

while(TRUE)
{

$d_i$ = a word drawn from the distribution $P(word | \Phi(d_1...d_{i-1}))$;
if($d_i$ == STOP) return;
i++;

}

Figure 2.5: A stochastic program that generates sequences. If $\Phi(d_1...d_{i-1}) = d_{i-m}...d_{i-1}$ then this program is equivalent to an $m$th order Markov model.

```
#d_i is the i'th decision made by the program -- each decision for odd
#values of i is to either choose a tag from the set of tags T or the STOP
#symbol; for even values of i it is to choose a word from the vocabulary
```

i = 1;

while(TRUE)
{

$d_i$ = a tag drawn from the distribution $P(tag|\Phi_1(d_1...d_{i-1}))$;
if($d_i$ == STOP) return;
$d_{i+1}$ = a word drawn from the distribution $P(word|\Phi_2(d_1...d_i))$;
i+=2;

}

Figure 2.6: A stochastic program that generates sequence pairs. If $\Phi_1(d_1...d_{i-1}) = d_{i-2m+1}, d_{i-2m+3}...d_{i-1}$ and $\Phi_2(d_1...d_i) = d_i$ then this program is equivalent to an $m$th order Hidden Markov model.

## 2.8 Additional Topics in Statistical Models

### 2.8.1 Unsupervised Learning through the EM Algorithm

Model structures that define joint probability distributions $P(x, y|\Theta)$ can be trained in an unsupervised fashion using the expectation-maximization (EM) algorithm [Dempster, Laird and Rubin 77]. In unsupervised training the training data is a sequence of events $X = x_1...x_n$ drawn from $\mathcal{X}$. [Baker 79] describes an efficient algorithm for EM training of PCFGs; [Baum 71] describes the forward-backward algorithm for HMMs.

## 2.9 Estimation

We now consider estimation of multinomial parameters when data is sparse, through "smoothing" of maximum-likelihood (ML) estimates. This section is not intended to be a comprehensive review of smoothing techniques: there is a large amount of relevant literature on this subject (for example, see [Jelinek 90] and [Chen and Goodman 96]). Instead, the section first gives some general motivation for the need for smoothing; second, gives details of the specific techniques used later in this thesis.

### 2.9.1 The Sparse Data Problem

Practically all of the models in this chapter require estimation of multinomial parameters. In general, then, we assume that there is some set of parameters $P(Y|X_1, X_2, ...X_n)$ for which we require estimates. $Y$ is a random variable that can take any value from a set of possibilities $\mathcal{Y}$. $X_1, X_2...X_n$ are the conditioning variables, and are members of the sets $\mathcal{X}_1, \mathcal{X}_2, ...\mathcal{X}_n$ respectively. For conciseness we use $\mathbf{X}$ to denote the context $X_1, X_2...X_n$; $\mathbf{X}$ is a member of the set $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times ... \times \mathcal{X}_n$. In the models in this chapter, the maximum-likelihood (ML) estimate of $P$ has taken the form:

$$\hat{P}_{ML}(Y|\mathbf{X}) = \frac{Count(Y, \mathbf{X})}{Count(\mathbf{X})} \tag{2.57}$$

There are usually some severe problems with this estimate. In high-dimensional parameter spaces $Count(\mathbf{X})$ may be very low, or even 0, leaving 2.57 inaccurate or even undefined. This is the sparse data problem. For example, take the problem of estimating probabilities in a 2nd order Markov model where the size of the vocabulary $\mathcal{V}$ is 20,000 words. In this case there are $20,000^3 = 8 * 10^{12}$ parameter values to be estimated, so we might expect to need a corpus of at least $8 * 10^{12}$ words to estimate the parameters with any accuracy. To expect to have a corpus of this size is unrealistic.

A general strategy for dealing with this problem is to use ML estimates computed for *lower order* distributions. Lower order parameters are multinomials that are conditioned on sub-contexts of $\mathbf{X}$, rather than the entire context of conditioning variables. For example, if $n = 3$ then there are 7 lower order distributions: $P(Y|X_1)$, $P(Y|X_2)$, $P(Y|X_1, X_2)$, $P(Y)$ and so on. The utility of lower order estimates can be motivated if we consider the nature of estimation error more carefully.

### 2.9.2 Two Sources of Estimation Error

The analysis in this section is for binomial distributions, the special case where $|\mathcal{Y}| = 2$; however, the same principles apply to the estimation of multinomial parameters where $|\mathcal{Y}| > 2$.

Say $\hat{p}^n$ is an estimate of $p$, based on a sample size of size $n$. We define the expected error of an estimate $\hat{p}^n$ as follows:

$$Err\left(\hat{p}^n\right) = E_p\left[\left(\hat{p}^n - p\right)^2\right] \tag{2.58}$$

The $E_p$ operator refers to expected value with respect to the underlying probability $p$. ($\hat{p}^n$ is a random variable whose distribution depends on $p$: to see this note that $p$ defines a distribution over the set of possible samples of size $n$, and that each of these samples maps to a different value for $\hat{p}^n$. Hence we can calculate the expected value w.r.t. $p$ of functions of $\hat{p}^n$.)

For the purposes of this section we assume that the goal of an estimation method is to minimize the above definition of $Err$ (this is a common criteria for estimation in the mathematical statistics literature, as in [BD 77] chapter 4). There may be arguments for the use of other measures of estimation error (such as Kullback-Liebler distance), but the major intuition of this section — that there is a trade-off between bias and variance as two sources of error — will most likely apply to those cases also, even if the formal definitions and analysis differ.

We now note that $Err(\hat{p})$ is the sum of two components. We define $\breve{p}^n$ as $E_p[\hat{p}^n]$. Then there are two sources of error:

$$Err1\left(\hat{p}^n\right) = \left(\breve{p}^n - p\right)^2 \tag{2.59}$$

$$Err2\left(\hat{p}^n\right) = E_p\left[\left(\hat{p}^n - \breve{p}^n\right)^2\right] \tag{2.60}$$

It can be shown ([BD 77] pg. 117) that

$$Err\left(\hat{p}^n\right) = Err1\left(\hat{p}^n\right) + Err2\left(\hat{p}^n\right) \tag{2.61}$$

$\breve{p}^n - p$ is the bias of the estimate; $Err1$ is this value squared. $Err2$ is a measure of sampling error: with small sample sizes, the estimate $\hat{p}^n$ will with some probability vary from its average value $\breve{p}^n$. As $n$ increases $Err2$ will decrease: and $Err2$ goes to 0 as the sample size $n$ goes to $\infty$.

### A Compromise Between the Two Sources of Error

With these results in mind, we now return to the analysis of estimates of $P(Y|\mathbf{X})$ based on sub-contexts of $\mathbf{X}$. One result [BD 77] is that the ML estimate $\hat{P}_{ML}(Y|\mathbf{X})$ has a value of 0 for $Err1$. Unfortunately though, the sample size for this estimate may be very small, so the $Err2$ component of its error may be very large. In contrast, imagine that we define $\hat{P}(Y|\mathbf{X}) = \hat{P}_{ML}(Y)$, an ML estimate of $Y$ based on the empty sub-context of $\mathbf{X}$. This estimate will almost certainly have $Err1 > 0$, but its $Err2$ component will be much lower than that for $\hat{P}_{ML}(Y|\mathbf{X})$, as the sample size on which it is based is much larger.

Between these two extremes are estimates based on non-empty sub-contexts of $\mathbf{X}$, which will in general have decreasing $Err2$ but increasing $Err1$ as the conditioning subset gets smaller. We could imagine an estimation strategy where we simply chose a particular sub-context of $\mathbf{X}$, $\Phi(\mathbf{X})$, as the basis of the estimate, so that $\hat{P}(Y|\mathbf{X}) = \hat{P}_{ML}(Y|\Phi(\mathbf{X}))$. A sensible method would be to choose the $\Phi(\mathbf{X})$ that lead to the best compromise between $Err1$ and $Err2$ problems, thereby minimizing the expected error of the estimate. It turns out, however, that taking a weighted average of different ML estimates is better than simply picking a single ML estimate. This is the topic of the next section.

### 2.9.3 Linear Combinations of ML Estimates

Say we are defining an estimate of $P(Y|\mathbf{X})$. We define an $n$-level back-off strategy $\Phi = \Phi_1 ... \Phi_n$ as follows:

- Each $\Phi_i$ is a function that maps $\mathbf{X}$ to some sub-context of $\mathbf{X}$. We write this sub-context as $\Phi_i(\mathbf{X})$. Thus each $\Phi_i$ can be represented as a subset of the set of integers $\{1, 2, ...n\}$, and can take $2^n$ possible values.
- The $i$th level estimate $\hat{P}_i(Y|\mathbf{X})$ is defined as $\hat{P}_{ML}(Y|\Phi_i(\mathbf{X})) = \frac{Count(Y,\Phi_i(\mathbf{X}))}{Count(\Phi_i(\mathbf{X}))}$.

We stipulate a couple of constraints on possible forms for $\Phi$:

1. $\Phi_1(\mathbf{X})$ is sufficiently small for $\hat{P}_1(Y|\mathbf{X})$ to be defined in all cases; i.e., $\forall \mathbf{X} \in \mathcal{X}\ \ Count(\Phi_1(\mathbf{X})) > 0$. (Often this means that $\Phi_1(\mathbf{X})$ is the empty set.)
2. If $i < j$ then $\Phi_i(\mathbf{X}) \subset \Phi_j(\mathbf{X})$.

$\hat{P}_i$ for $1 \leq i \leq n$ now defines a sequence of estimates, with $Err1$ for each estimate decreasing with increasing $i$, and $Err2$ increasing with increasing $i$. So as we move through the estimates $\hat{P}_1 ... \hat{P}_n$ there is a trade-off between $Err1$ and $Err2$.

A linear combination of these estimates can be defined using a weight $\lambda_i$ associated with each back-off level. We define the $i$th level smoothed estimate $\tilde{P}_i$ recursively:

$$\begin{aligned} \tilde{P}_1 &= \hat{P}_1 \\ \tilde{P}_i &= \lambda_i \hat{P}_i + (1 - \lambda_i) \tilde{P}_{i-1} \quad \text{for } 1 < i \leq n \end{aligned} \tag{2.62}$$

The final estimate will be $\tilde{P}_n$, which incorporates all estimates $\hat{P}_1 ... \hat{P}_n$ that are well defined. Each $\lambda_i$ must take a value between 0 and 1 for each $\tilde{P}_i$ to define a distribution over $\mathcal{Y}$. The value of $\lambda_i$ can be interpreted as an indication of how much we trust $\hat{P}_i$ rather than the $(i-1)$th level smoothed estimate: a value of 1 means that we trust it completely, a value of 0 means we don't trust it at all. The next question is how $\lambda_i$ should be calculated.

### 2.9.4 Calculating Back-Off Weights

Once the back-off strategy $\Phi = \Phi_1 ... \Phi_n$ has been defined, the remaining question is how to pick each of the weights $\lambda_i$. At the very least, $\lambda_i$ should be 0 if $Count(\Phi_i(\mathbf{X})) = 0$, in which case $P_i$ is undefined (otherwise the final estimate $\tilde{P}_n$ will be undefined). Beyond this constraint there are a couple of methods that we will discuss: optimizing the likelihood of held-out data, and direct calculation of the value of $\lambda_i$ from various characteristics of the sample.

A theme that is common to both of these techniques is to make $\lambda_i$ dependent on $Count(\Phi_i(\mathbf{X}))$, the sample size on which $P_i$ is based. If $Count(\Phi_i(\mathbf{X})) = 0$, then $P_i$ is undefined and $\lambda_i$ *must* be 0. As $Count(\Phi_i(\mathbf{X}))$ increases, we would expect $P_i$ to be become more reliable, and to reflect this $\lambda_i$ should move towards its maximum possible value of 1.

#### Method 1: Optimizing the Likelihood of Held-out Data

[Jelinek 90] describes how the values for $\lambda_i$ can be calculated with the use of held-out data. The basic idea is to split the training data into two sub-sets: the first is used to calculate the different maximum-likelihood estimates $\hat{P}_i$; $\lambda_i$ then is chosen to optimize the likelihood of the second set of held-out data. [Jelinek 90] describes methods for optimization of the weights $\lambda_i$.

The dependence of $\lambda_i$ on $Count(\Phi_i(\mathbf{X}))$ is achieved through "bucketing". In the most extreme case, this means training a different value of $\lambda_i$ for each value of $Count(\Phi_i(\mathbf{X}))$. More likely, a different $\lambda_i$ is trained for each range of counts: for example, a different value might be trained for $Count(\Phi_i(\mathbf{X})) = 0, 1, 2, [3-5], [6-10], [11-20], [21-30], [30-50], [50-100], [100-\infty]$. The boundaries of these ranges are usually chosen so that there are as many buckets as possible, subject to the constraint that each bucket contains some minimum number of events required to estimate $\lambda_i$ robustly.

A method that makes more efficient use of training data is to rotate the held-out data: for example partitioning the training data into 10 equally size subsets, then estimating $\lambda_i$ values with each of the 10 subsets held-out, calculating the final value of $\lambda_i$ as the average of these 10 values.

#### Method 2: Direct Calculation of $\lambda_i$

Another method is to define $\lambda_i$ in terms of $Count(\Phi_i(\mathbf{X}))$:

$$\begin{aligned} \lambda_i &= 0 && \text{If } Count(\Phi_i(\mathbf{X})) = 0 \\ \lambda_i &= \frac{Count(\Phi_i(\mathbf{X}))}{Count(\Phi_i(\mathbf{X})) + C_i} && \text{If } Count(\Phi_i(\mathbf{X})) > 0 \end{aligned} \tag{2.63}$$

$C_i$ is a constant that can be optimized using held-out data. This definition satisfies the intuition that $\lambda_i$ should go to 1 as $Count(\Phi_i(\mathbf{X}))$ goes to $\infty$. The value for $C_i$ dictates the rate at which this asymptote is approached: the higher $C_i$'s value, the slower the convergence towards 1.

While this method has some of the properties we would expect, it does seem rather ad-hoc. However, there are a number of methods in the smoothing literature that justify this relationship between $\lambda_i$ and $Count(\Phi_i(\mathbf{X}))$, both theoretically [Witten and Bell 91] and empirically [Chen and Goodman 96].

An approach that we will use in the parser in chapter 7 is to calculate $C_i$ as a function of what we call the *diversity* of $\Phi_i(\mathbf{X})$, $D\left(\Phi_i(\mathbf{X})\right)$. The method is borrowed from [Bikel et al. 97], and has further motivation in [Witten and Bell 91]. The diversity is defined as

$$D\left(\Phi_i(\mathbf{X})\right) = |\mathcal{Y}(\Phi_i(\mathbf{X}))| \tag{2.64}$$

where

$$\mathcal{Y}(\Phi_i(\mathbf{X})) = \{y \mid Count(y, \Phi_i(\mathbf{X})) > 0\} \tag{2.65}$$

The diversity is the number of different outcomes that have been seen with the context $\Phi_i(\mathbf{X})$ in the training sample. 2.63 is then modified to be:

$$\begin{aligned} \lambda_i &= 0 && \text{If } Count(\Phi_i(\mathbf{X})) = 0 \\ \lambda_i &= \tfrac{Count(\Phi_i(\mathbf{X}))}{Count(\Phi_i(\mathbf{X}))+C\times D\left(\Phi_i(\mathbf{X})\right)} && \text{If } Count(\Phi_i(\mathbf{X})) > 0 \end{aligned} \tag{2.66}$$

$C$ is a constant that can also be optimized on held-out data. $\lambda_i$ is now sensitive to the diversity as well as the count of $\Phi_i(\mathbf{X})$. As the diversity increases, $\lambda_i$ decreases, and less trust is put in the estimate $\lambda_i$. The motivation behind this is that the diversity is a measure of how likely a new outcome is to appear in a test sample: the higher the diversity the more likely a novel event is to occur, and the less $\hat{P}_i$ should be trusted ($\hat{P}_i$ estimates the probability of any novel event as 0). Take an example where the sample size is 10: if out of those 10 events in the sample the same outcome was seen every time (i.e. $D = 1$), then we should be reasonably sure that novel events are unlikely to occur in test data, and $\lambda_i$ should be high. In contrast, if a different outcome was seen every time ($D = 10$), then we should be reasonably sure that novel events are likely to be seen in test data, and $\lambda_i$ should be correspondingly lower.

Empirical justification for this model comes from [Chen and Goodman 96]: they report very good performance for a method that is quite similar to 2.63. They define a function $OneCount$

$$OneCount\left(\Phi_i(\mathbf{X})\right) = |\mathcal{Y}(\Phi_i(\mathbf{X}))| \tag{2.67}$$

where

$$\mathcal{Y}(\Phi_i(\mathbf{X})) = \{y \mid Count(y, \Phi_i(\mathbf{X})) = 1\} \tag{2.68}$$

$\lambda_i$ is then defined as

$$\lambda_i = 0 \qquad \text{If } Count(\Phi_i(\mathbf{X})) = 0$$

$$\lambda_i = \frac{Count(\Phi_i(\mathbf{X}))}{Count(\Phi_i(\mathbf{X}))+B+C\times OneCount\big(\Phi_i(\mathbf{X})\big)} \qquad \text{If } Count(\Phi_i(\mathbf{X})) > 0 \tag{2.69}$$

So their measure of diversity, $OneCount$, is quite similar to ours, and will most likely be highly correlated; they also have an additional constant $B$ that can be optimized on a held-out data set.

# Chapter 3

# Some Alternative Parameterizations for Statistical Parsing

The previous chapter described some mathematical techniques for statistical modeling of natural language problems. This chapter gives additional background, considering alternative parameterizations for parsing. We give a sequence of proposals in increasing order of sophistication; each refinement is motivated through examples where the old parameterization fails, and the new method fixes the perceived deficiency.

We assume that designing a model structure is a two stage process. The first step is the choice of parameterization (the subject of this chapter). The second step in a realization of this representational choice in a precise mathematical model (using techniques described in the previous chapter). Of course, this is an idealization. The desire for a mathematically sound formulation will sometimes alter the parameterization (often in interesting ways). The model of chapter 6 is a first attempt to implement the parameter types of this chapter; chapter 7 gives final models with all of the parameter types.

## 3.1 A Definition of Parse-Tree Parameterization

We first define what we mean by the "parameterization" of a parse tree for probabilistic parsing. In the models in this thesis, the probability of a parse tree is defined as the product of several terms, each term being associated with an "event" in the parse tree (an event is some fragment of the parse tree):

$$\mathcal{P}(T,S) = \prod_{i=1...n} Score(Event_i) \tag{3.1}$$

For example, in the case of a PCFG, $Event_i$ is the $i$'th context free rule in the tree.

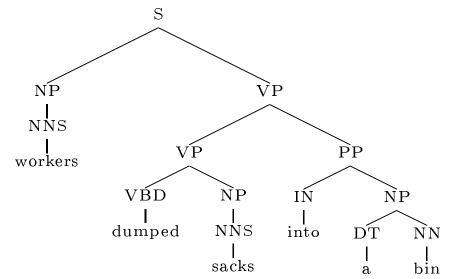
The parameterization of a parse tree is the choice of $Event_1 ... Event_n$ — in the case of a PCFG, the $n$ context-free rules in the tree. In other words, the choice of parameterization is the choice of how to break down trees: what linguistic objects to associate the parameters with.

The $Score$ for each event will be an estimate of some conditional probability, and in general will be directly related to the number of times the event has been seen in training data, $Count(Event_i)$. Typically, $Event_i$ will be split into two parts: $Context_i$ and $Prediction_i$, with $Score(Event_i) = \hat{P}(Prediction_i|Context_i)$ ($\hat{P}$ is an estimate of $P$). In the case of PCFGs, if $Event_i$ is the rule $\alpha_i \rightarrow \beta_i$ then $Context_i = \alpha_i$, $Prediction_i = \beta_i$ and $Score(Event_i) = \hat{P}(\beta_i|\alpha_i)$. [Booth and Thompson 73] showed that this choice of $Prediction_i$ and $Context_i$ for PCFGs leads to a consistent model; in general, the choice of $Prediction_i$ and $Context_i$ will be motivated by the desire for a "well-formed" model, where the idea of a "well-formed" model is defined in section 2.3.3.

In this chapter, however, we will ignore the precise definition of the $Score$ for each $Event$. The crucial point is that the utility of a particular parameterization can be investigated through its ability to discriminate between different trees. This discriminative ability is independent of the exact formula used to calculate the $Score$. For example, by noting that the two trees in figure 3.1 differ only by rules that make no mention of lexical information, we can see that a simple PCFG ignores lexical information when making PP attachment decisions. This ignorance holds regardless of how the score for the rules in the tree is calculated.

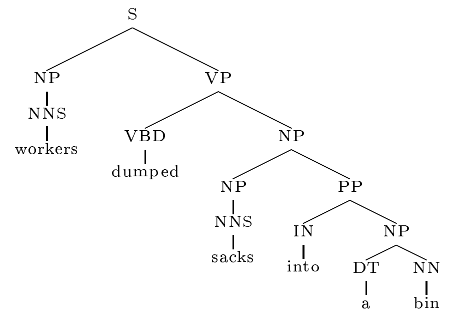
The first change, a shift from simple PCFGs to dependencies, is the most radical. From

(a)




| Rules |
|---|
| S → NP VP |
| NP → NNS |
| **VP → VP PP** |
| VP → VBD NP |
| NP → NNS |
| PP → IN NP |
| NP → DT NN |
| NNS → workers |
| VBD → dumped |
| NNS → sacks |
| IN → into |
| DT → a |
| NN → bin |

(b)




| Rules |
|---|
| S → NP VP |
| NP → NNS |
| **NP → NP PP** |
| VP → VBD NP |
| NP → NNS |
| PP → IN NP |
| NP → DT NN |
| NNS → workers |
| VBD → dumped |
| NNS → sacks |
| IN → into |
| DT → a |
| NN → bin |

Figure 3.1: A case of PP attachment ambiguity. (a) Verb attachment, the correct tree. (b) Noun attachment. Note that the trees share exactly the same set of rules, except (a) has a rule `VP → VP PP`, (b) has a rule `NP → NP PP`.

there we add progressively more features, motivating each additional piece of information through example trees where the previous parameterization can be seen to be problematic.

For convenience, throughout this chapter we will use the terms "parameterization" and "representation" interchangeably. This differs from our definition of representation in chapter 1, repeated here:

- **Representation**. Choose how to represent parse trees. For example, choose the set of part-of-speech tags and non-terminal labels in the tree; choose whether or not to have lexical head-words attached to non-terminals; choose whether to represent words directly, or as their morphological stems, or as bit-strings derived through clustering techniques.

By chapter 1's definition, the parsers of [Ratnaparkhi 97, Charniak 97, Goodman 97] and this thesis all use the same representation. By the definition of this chapter, they differ to varying degrees in their choice of parameterization, or representation. The stricter definition of representation, as used in this chapter, reflects our emphasis on the parameters of the model: that it is not sufficient to simply include features that may be useful for disambiguation, it is also crucial to consider how they are linked together in the final parameters of the model.

### 3.1.1 A Note about *Events* in this Chapter

In this chapter, for conciseness, we do not consider how the various parameters may interact — for example, the choice of subcategorization frame for a head will influence the dependencies that it takes.

Eventually, the models of chapter 7 will give a fully specified model where such interactions are captured. The parameterization is formed using a history–based model, where each event corresponds to a decision in a top-down derivation of the tree. The conditioning context for each decision is chosen so that interactions are modeled properly. In some cases this will result in the model conditioning on context that is not included in the events of this chapter. The specification of events in this chapter should therefore be taken as the "core", essential subset of information; the final parameters of chapter 7 will include additional context in some cases.

### 3.1.2 Parameterization Proposals: a Summary

The following sections will consider a sequence of proposals, summarized here:

**A simple PCFG** Each "simple" context-free rule in a parse-tree has an associated probability. Section 3.2 defines a simple PCFG as containing two types of rules: (1) rules $\mathtt{X} \rightarrow \mathtt{Y}$ where X is a POS tag and Y is a lexical item; (2) rules $\mathtt{X} \rightarrow \mathtt{Y}_1 \ldots \mathtt{Y}_n$ where each X and $\mathtt{Y}_i$ are drawn from a simple set of non-terminals such as VP, NP, or SBAR.

**Dependencies** A dependency is defined to be a relation between two words in a sentence (a *modifier* and a *head*), written $\langle modifier \rightarrow head \rangle$. A tree for a sentence with $n$ words contains $n$ dependencies between pairs of words; each dependency has an associated probability. Thus this proposal introduces probabilities associated with *pairs* of words in the sentence.

**Dependencies + Direction** A tree is represented as $n$ dependency relations $\langle modifier \rightarrow head, direction \rangle$. *direction* specifies the relative order of the *modifier* and *head* in the sentence.

**Dependencies + Direction + Relations** A tree is represented as $n$ dependency relations $\langle modifier \rightarrow head, direction, relation \rangle$. *relation* specifies the grammatical relation between the two words. It is formed by a triple of non-terminals taken from the tree. As an example, a dependency might be $\langle IBM \rightarrow acquired, \text{LEFT}, \langle \mathtt{NP,S,VP} \rangle \rangle$, signifying a relationship where: *IBM* is the modifier; *acquired* is the head; *IBM* appears to the left of *acquired*; and the relationship is $\langle \mathtt{NP,S,VP} \rangle$ (a triple of non-terminals representing a subject-verb relationship).

**Dependencies + Direction + Relations + Subcategorization** A tree is represented by $m$ subcategorization probabilities, in addition to the $n$ dependencies defined before. Subcategorization probabilities correspond to events of the form "What is the probability that *give* takes two NP complements to its right?", or "What is the probability that *give* takes a single subject NP to its left?".

**Dependencies + Direction + Relations + Subcategorization + Distance** The dependency events are extended to include a *distance* variable, some measure of the distance between the *modifier* and *head*. The distance is measured either over the surface string between the two words, or with some reference to the tree structure. It allows the model to differentiate dependencies between "close" vs. "distant" pairs of words; this is important for the model to learn preferences for close-attachment.

**Dependencies + Direction + Relations + Subcategorization + Distance + Parts-of-Speech** All events including lexical items are extended to include the POS tag associated with that lexical item. Statistics based on lexical items may be sparse; statistics based on POS tags can be used as a fall-back in cases of sparse data.

## 3.2 Parameterization Proposal 1: A Simple PCFG

**Parameterization Proposal 1.** *(A simple PCFG) A parse tree is represented as* $n$ *events* $Event_1 ... Event_n$*, where* $Event_i$ *is the i'th context-free rule in the tree, and the rules are "simple" (the idea of a "simple" rule is defined below).*

Much of the early work on statistical parsing of natural language focused on PCFGs as a formalism. Results, however, were rather disappointing — for example [Charniak 97] reports accuracy of 70.6/74.8% recall/precision[1] for a PCFG trained and tested on the Penn Wall Street Journal treebank [Marcus et al. 93]. In comparison, the models in this thesis reach over 88% precision and recall on the same task.

Inducing a PCFG directly from the Penn WSJ treebank, as in the non-lexicalized (rather than the lexicalized) model of [Charniak 97], leads to a grammar with the following properties:

- Each rule in the grammar is either of the form
  1. $X \rightarrow Y_1 ... Y_n$, where X and $Y_1 ... Y_n$ are non-terminals, and $n \geq 1$.
  2. $X \rightarrow x$, where $X$ is a non-terminal part of speech, and $x$ is a lexical item.
- The non-terminals in the grammar are a very simple, restricted set of labels `NP`, `VP`, `S`, `SBAR`, `PP` etc.

[1] Recall and precision are defined in section 6.4 of this work; informally, they refer to the accuracy in recovering constituents in a parse tree. A constituent is defined by its non-terminal, and the words it spans in the sentence.

$N$ = {TOP, S, NP, VP, VB, NNP}

$\Sigma$ = {gave, saw, U.S., IBM, yesterday}

$S$ = TOP

| Rules $P$ | Probabilities $D$ |
|---|---|
| TOP $\Rightarrow$ S | 1.0 |
| S $\Rightarrow$ NP VP | 0.8 |
| S $\Rightarrow$ NP NP VP | 0.2 |
| VP $\Rightarrow$ VB NP | 0.6 |
| VP $\Rightarrow$ VB NP NP | 0.4 |
| NP $\Rightarrow$ N | 1.0 |
| VB $\Rightarrow$ gave | 0.6 |
| VB $\Rightarrow$ bought | 0.4 |
| N $\Rightarrow$ Lotus | 0.8 |
| N $\Rightarrow$ IBM | 0.1 |
| N $\Rightarrow$ yesterday | 0.1 |

Figure 3.2: A simple PCFG

- The part-of-speech tags are also relatively restricted, distinguishing major category (noun, verb, preposition etc.), and some simple sub-categories (singular/plural/proper nouns, different verb inflections etc.). The part-of-speech tags do not make subcategorization distinctions for verbs (for example transitive vs. intransitive).

From here on we will refer to a PCFG that satisfies these properties as a “simple” PCFG. The grammar in figure 2.1, repeated in figure 3.2, is an example of a simple PCFG.

In an attempt to account for the poor parsing accuracy of simple PCFGs, we identify two major weaknesses in this parse tree representation:

1. Lack of sensitivity to lexical dependencies.
2. Lack of sensitivity to structural preferences such as preferences for right-branching or left-branching structures.

### 3.2.1 Lack of Sensitivity to Lexical Dependencies

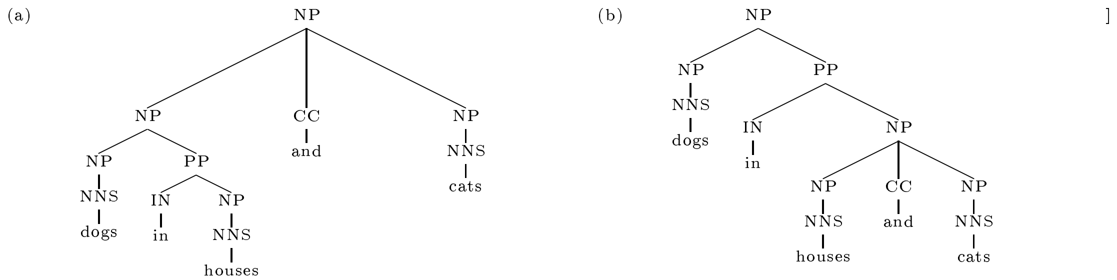
**Some Examples**

A major weakness of simple PCFGs is their lack of sensitivity to lexical information. To illustrate this, first take the example of prepositional phrase attachment ambiguity shown in figure 3.1. The important point is that the trees for the two analyses differ by only one rule: in the verb attachment case the tree has a rule $\texttt{VP} \rightarrow \texttt{VP PP}$, in the noun attachment case the differing rule is $\texttt{NP} \rightarrow \texttt{NP PP}$. Given that a PCFG assigns the probability for an entire tree as a product of rule probabilities, the attachment decision hinges on the probability for these two rules. If $\mathcal{P}(\texttt{VP} \rightarrow \texttt{VP PP}) > \mathcal{P}(\texttt{NP} \rightarrow \texttt{NP PP})$ then the verb-attachment tree will have higher probability. If $\mathcal{P}(\texttt{NP} \rightarrow \texttt{NP PP}) > \mathcal{P}(\texttt{VP} \rightarrow \texttt{VP PP})$ then the noun-attachment tree will have higher probability.

By this argument, the decision between the two structures depends only on the two rule probabilities, and has no dependence on the lexical items themselves. In fact, any sentence with the part-of-speech sequence $\langle\texttt{NNS VBD NNS IN DT NN}\rangle$ will be assigned a verb-attachment if $\mathcal{P}(\texttt{VP} \rightarrow \texttt{VP PP}) > \mathcal{P}(\texttt{NP} \rightarrow \texttt{NP PP})$, a noun-attachment otherwise. This is problematic in that the attachment clearly depends on the words involved: as an example, *workers sold sacks of a chemical* has the same POS sequence as *workers dumped sacks into a bin*, but involves a noun-phrase attachment.

Looking at statistics from the Penn treebank we can see how bad this non-lexical strategy is. Inspection of the training set of PP attachment ambiguities originally used in [Ratnaparkhi et al. 94] shows that 52% of attachments were to the noun. On this basis, we would expect the purely structural preference made by a PCFG to achieve an accuracy barely above chance. In contrast, methods which look at the 4 head-words involved in the pp-attachment ambiguity (e.g., `dumped sacks into bin`) can achieve up to 84% accuracy[2] (see the results in chapter 5). The lack of lexical sensitivity in PCFGs is decreasing accuracy by over 30% in this case.

[2]Even a method that looks at the preposition alone, making the most frequent attachment for each preposition, scores around 72% accuracy. In the example *into* can attach to both nouns and verbs, but is seen attached to verbs 92% of the time in [Ratnaparkhi et al. 94]'s training set.




| Rules |
|---|
| NP → NP CC NP |
| NP → NP PP |
| NP → NNS |
| PP → IN NP |
| NP → NNS |
| NP → NNS |
| NNS → dogs |
| IN → in |
| NNS → houses |
| CC → and |
| NNS → cats |

| Rules |
|---|
| NP → NP CC NP |
| NP → NP PP |
| NP → NNS |
| PP → IN NP |
| NP → NNS |
| NP → NNS |
| NNS → dogs |
| IN → in |
| NNS → houses |
| CC → and |
| NNS → cats |

Figure 3.3: A case of coordination ambiguity. Both analyses contain exactly the same set of rules, and will therefore receive equal probability.

There are many other cases of ambiguity that exhibit a similar sensitivity to lexical information. Examples are coordination, relative clause attachment, and noun-noun compounds. ([Lauer 95] describes a method that uses dependency information for disambiguation within noun-noun compounds.) Figure 3.3 shows an example of coordination. This case is even more extreme: the two possible analyses contain the same set of rules, and therefore receive equal probability under a simple PCFG. Again, lexical dependencies seem crucial: *dogs* and *cats* are far more likely to be coordinated than *houses* and *cats*.

### A General Result

The only lexical sensitivity that the simple PCFG has is to part-of-speech frequencies for different words, through the parameters $\mathcal{P}(\text{tag} \rightarrow \text{word} \mid \text{tag})$[3].

To see this, suppose we make the simplifying assumption that each word in the vocabulary has only one possible part of speech tag, i.e. one tag where $\mathcal{P}(\text{tag} \rightarrow \text{word} \mid \text{tag}) > 0$.

[3]For example, the word *saw* is seen 84 times in the Penn treebank: 4 times tagged as *NN*, 80 times tagged as *VBD*. *NN* is seen 163935 times in total, *VBD* is seen 37493 times. So the parameters $\mathcal{P}(\text{saw} \mid \text{VBD})$ and $\mathcal{P}(\text{saw} \mid \text{NN})$ will be vastly different: $\frac{\mathcal{P}(\text{saw} \mid \text{VBD})}{\mathcal{P}(\text{saw} \mid \text{NN})} = \frac{80}{37493} \times \frac{163935}{4} = 87.4$.

We define the function $tag(word)$ to be the one possible tag for a particular $word$. Given a sentence $\langle w_1...w_n \rangle$, all trees for this sentence which have probability greater than zero must then have the POS sequence $tag(w_1)...tag(w_n)$. Furthermore, all of these trees will have the same product of probabilities involving lexical items: $\prod_{i=1...n} \mathcal{P}(w_i \mid tag(w_i))$. When comparing different possible trees for an input sentence $w_1...w_n$, the lexical probabilities will be irrelevant and the ranking will be done solely on the basis of the rules in the tree. In conclusion, if any two sentences have the same tag sequence, they will have to receive the same highest-probability analysis.

### 3.2.2 Structural Preferences

A second weakness of simple PCFGs is their lack of sensitivity to structural information. Statistics in the Penn WSJ treebank show that there is a definite preference for right-branching structures, in that adverbials tend to attach to the most recent possible attachment site. Figures 3.4 and 3.5 show examples involving PP attachment to nouns and verbs respectively. In both cases the right-branching structure is seen around two-thirds of the time, but the competing structures contain identical rules so the simple PCFG fails to capture this preference. [Briscoe and Carroll 93] used similar examples to motivate the move from a simple PCFG to a probabilistic LR parser.

## 3.3 Dependency Parameterizations

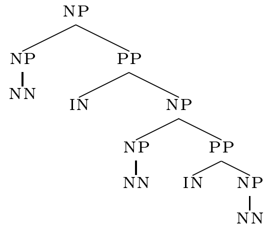
The second representation proposal, a shift to dependencies, is a radical move from the simple PCFG. It is largely motivated by a desire for increased sensitivity to lexical information.

### 3.3.1 Parameterization Proposal 2: Dependencies

**Parameterization Proposal 2.** *(Dependencies) A parse tree is represented as $n$ events $Event_1...Event_n$, where $Event_i$ is the dependency $w_i \rightarrow h_i$. $w_i$ is the i'th word in the sentence, $h_i$ can be any other word in the sentence or the $START$ symbol.*

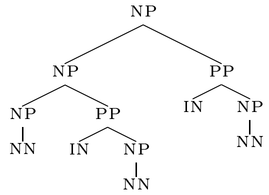
Note that a sentence with $n$ words leads to a parse tree with $n$ dependencies. In a

(a)




| Rules |
|---|
| NP → NP PP |
| NP → NN |
| PP → IN NP |
| NP → NP PP |
| NP → NN |
| PP → IN NP |
| NP → NN |

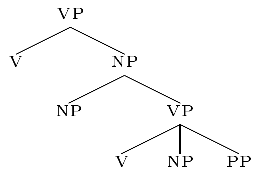
(b)




| Rules |
|---|
| NP → NP PP |
| NP → NN |
| PP → IN NP |
| NP → NP PP |
| NP → NN |
| PP → IN NP |
| NP → NN |

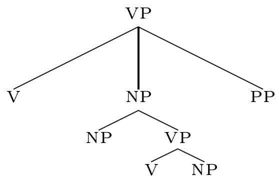
Figure 3.4: Two possible structures for the same sequence of POS tags. (a) is a right branching structure: the second prepositional phrase attaches to the closest noun. In (b) the second prepositional phrase attaches to the furthest noun. In Penn Wall Street Journal text structure (a) appears 68% of the time, (b) appears 32% of the time. However, both structures contain exactly the same set of rules, so the simple PCFG fails to distinguish between them.

(a)




| Rules |
|---|
| VP → V NP |
| NP → NP VP |
| VP → V NP PP |

(b)




| Rules |
|---|
| VP → V NP |
| NP → NP VP |
| VP → V NP PP |

Figure 3.5: Two possible structures for the same sequence of POS tags. (a) is a right branching structure: the second prepositional phrase attaches to the closest verb. In (b) the second prepositional phrase attaches to the furthest verb. In Penn Wall Street Journal text structure (a) appears 67% of the time, (b) appears 33% of the time. However, both structures contain exactly the same set of rules, so the simple PCFG fails to distinguish between them.

dependency $w_i \rightarrow h_i$ we will describe $w_i$ as the modifier, $h_i$ as the head. As an example, figure 3.6 gives two trees with their associated dependencies.

### 3.3.2 The Function from Trees to Sets of Dependencies

This section defines the function from a tree to its associated dependencies. There are two stages involved: first, the lexicalization of trees (i.e., the addition of a word to each non-terminal label in the tree); second, the derivation of $(n-1)$ dependencies from each rule with $n$ children.

#### Step 1: Lexicalization of Parse Trees

[Black et al. 92b, Jelinek et al. 94, Magerman 95] introduced *lexicalization* of non-terminals as a way of improving parsing accuracy. Each non-terminal in the tree is modified to also include a *head-word*, one of the words in the sentence. Headwords are assigned through a function that identifies the "head" of each rule in the grammar. The head is one of the child non-terminals in the rule. More precisely, the function $head(X \rightarrow Y_1...Y_n)$ returns a value $1 \leq h \leq n$, where $h$ is the index of the head (i.e. $Y_{head(X \rightarrow Y_1...Y_n)}$ is the head of the phrase). (Appendix A defines the function used in this thesis.)

For example, in the rule $\texttt{S} \rightarrow \texttt{NP VP}$ the VP would, by linguistic arguments, be the head of the phrase. In this case $head(\texttt{S} \rightarrow \texttt{NP VP}) = 2$.

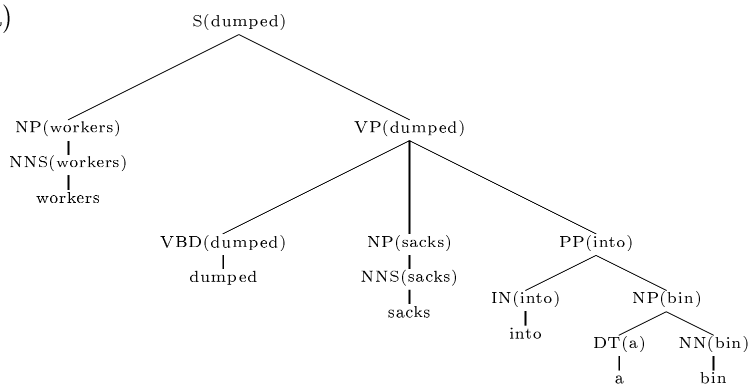
The head*word* for each non-terminal can then be defined recursively:

**base case** If a non-terminal $X$ is on the left-hand-side of a rule $X \rightarrow x$, where: 1) $X$ is a non-terminal part of speech; and 2) $x$ is a lexical item; then $headword(X) = x$.

**recursive case** If (1) a non-terminal $X$ is on the left-hand-side of a rule $X \rightarrow Y_1...Y_n$, where $Y_1...Y_n$ are non-terminals; and (2) $h = head(X \rightarrow Y_1...Y_n)$; then $headword(X) = headword(Y_h)$.

For example, take the rule $\texttt{S} \rightarrow \texttt{NP VP}$ in figure 3.7. The VP is the head of the phrase, and $headword(\texttt{S}) = headword(\texttt{VP}) =$ dumped.

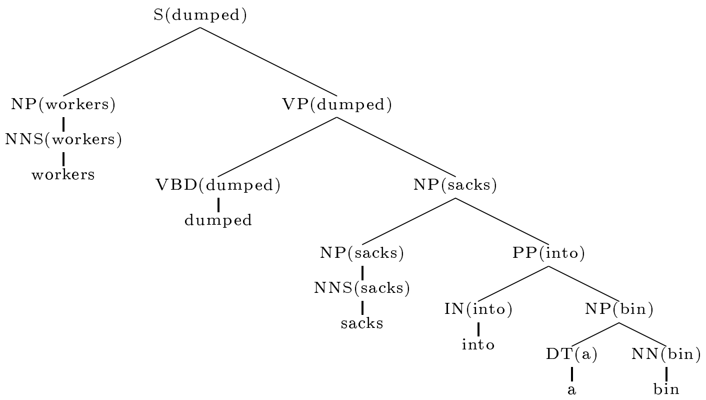
(a)




(b)

| Dependencies |
|---|
| workers → dumped |
| dumped → START |
| sacks → dumped |
| **into → dumped** |
| a → bin |
| bin → into |

a')

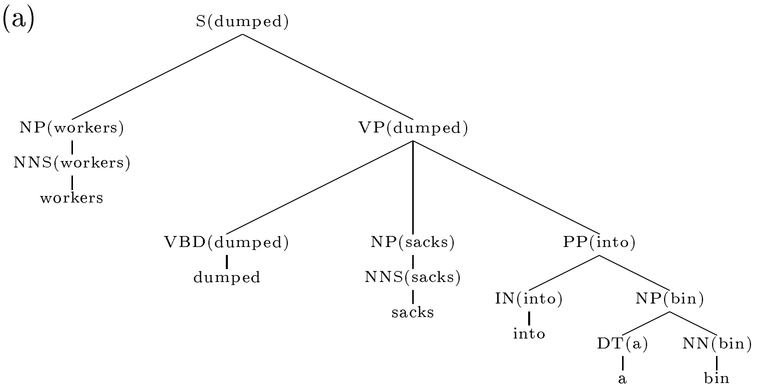



b')

| Dependencies |
|---|
| workers → dumped |
| dumped → START |
| sacks → dumped |
| **into → sacks** |
| a → bin |
| bin → into |

Figure 3.6: (a) a lexicalized tree. (b) a list of dependencies that the tree contains. a') a lexicalized tree with the PP attaching to the noun, and b') the dependencies that it contains.

(a)




(b)

| Rules |
|---|
| $\mathrm{S} \rightarrow \mathrm{NP}\ \underline{\mathrm{VP}}$ |
| $\mathrm{NP} \rightarrow \underline{\mathrm{NNS}}$ |
| $\mathrm{VP} \rightarrow \underline{\mathrm{VBD}}\ \mathrm{NP}\ \mathrm{PP}$ |
| $\mathrm{NP} \rightarrow \underline{\mathrm{NNS}}$ |
| $\mathrm{PP} \rightarrow \underline{\mathrm{IN}}\ \mathrm{NP}$ |
| $\mathrm{NP} \rightarrow \mathrm{DT}\ \underline{\mathrm{NN}}$ |

Figure 3.7: (a) a lexicalized tree: each non-terminal has an associated *headword* (shown in parentheses after the non-terminal). (b) a list of rules in the tree, with the *head* for each rule underlined. The definition of the head of each rule leads to the recovery of headwords: each non-terminal receives its headword from its head child.

**Step 2: Derivation of Dependencies from Lexicalized Trees**

Having defined the headword for each non-terminal in the tree, the next step is to identify a set of *dependencies* between words in the sentence. A dependency is a relationship between two word-tokens[4] in a sentence, a *modifier* and its *head*, which we will write as $modifier \rightarrow head$. The dependencies for a given tree are derived in two ways:

- Take each rule $X \rightarrow Y_1...Y_n$ such that: 1) $Y_1...Y_n$ are non-terminals; 2) $n \geq 2$; 3) $h = head(X \rightarrow Y_1...Y_n)$. Each rule contributes $(n-1)$ dependencies, namely $headword(Y_i) \rightarrow headword(Y_h)$ for $1 \leq i \leq n, i \neq h$.

- If $X$ is the root non-terminal in the tree, and $x$ is its headword, then $x \rightarrow$ START is a dependency.

For example, in the top tree in figure 3.6, the rule `VP` $\rightarrow$ `VBD NP PP` contributes two dependencies. In this case $n = 3$, $h = 1$, and the dependencies are $\langle headword(Y_2) \rightarrow$

[4]From here on we use the term *word* to mean *word-token*, at least when there is no danger of confusion.

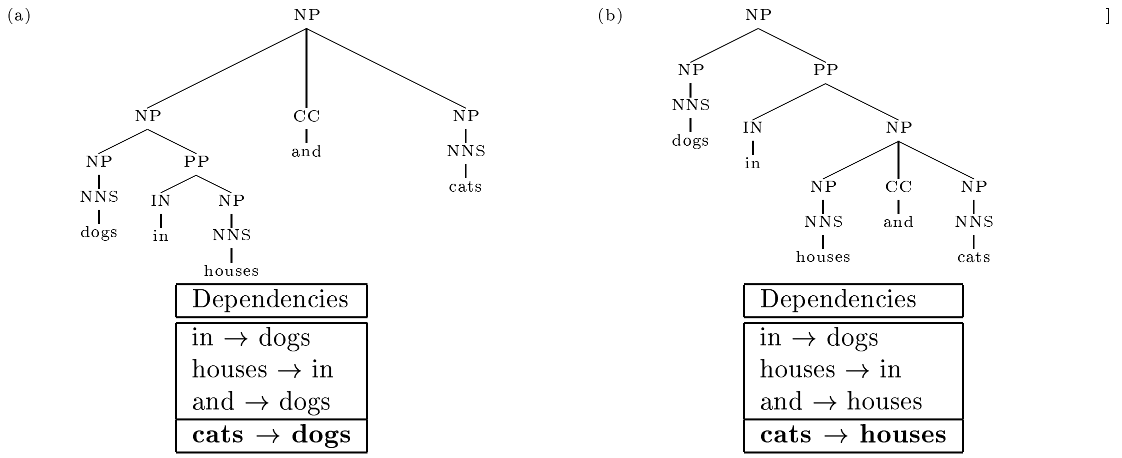



Figure 3.8: The case of coordination ambiguity revisited, using a dependency representation.

$headword(Y_1)\rangle = \langle sacks \rightarrow dumped\rangle$ and $\langle headword(Y_3) \rightarrow headword(Y_1)\rangle = \langle into \rightarrow dumped\rangle$. Figure 3.6 lists the entire set of dependencies in the tree.

### 3.3.3 The Motivation for Dependencies as a Representation

Representing trees as sets of dependencies rather than as a set of simple PCFG rules leads to very direct use of lexical information. This is the major strength of the dependency representation.

This point is best illustrated by a couple of examples. First, take the two trees in figure 3.6. The two trees differ by only one dependency: the decision between them will depend on the values of $Score(into \rightarrow dumped)$ and $Score(into \rightarrow sacks)$. Whereas in the simple PCFG the decision between the two structures hinged on rule probabilities that paid no attention to lexical items, the decision now depends on the three lexical items *dumped*, *sacks*, and *into*.

Figure 3.8 shows the dependency structures for the coordination example, originally shown in figure 3.3. The two trees differ by a couple of dependencies, but most importantly one has a dependency *cats* → *dogs*, whereas the other has a dependency *cats* → *houses*. (See Appendix A for a description of the head-rules used in this work, and how they deal with coordination. The first coordinated phrase is taken as the head of the entire phrase, with the second conjunct standing in a special coordination relationship to this head.)

### 3.3.4 Parameterization Proposal 3: Dependencies + Direction

**Parameterization Proposal 3.** *(Dependencies + Direction) A parse tree is represented as n events $Event_1 ... Event_n$, where $Event_i$ is the tuple $\langle w_i \rightarrow h_i, direction_i \rangle$. $w_i \rightarrow h_i$ is the i'th dependency, as before. $direction_i$ is* `L` *(for left) if $w_i$ precedes $h_i$ in the sentence,* `R` *(for right) if $w_i$ follows $h_i$.*

The need for a feature describing the relative order of the $modifier$ and $head$ is not surprising: especially in English, which has strong word order. Most dependencies are much more likely to occur in one direction than the other: a subject modifier is almost always seen to the left of the head it modifies, an `NP` complement to a preposition is always seen to the right of the preposition. The addition of the *direction* variable allows the model to take these facts into account.

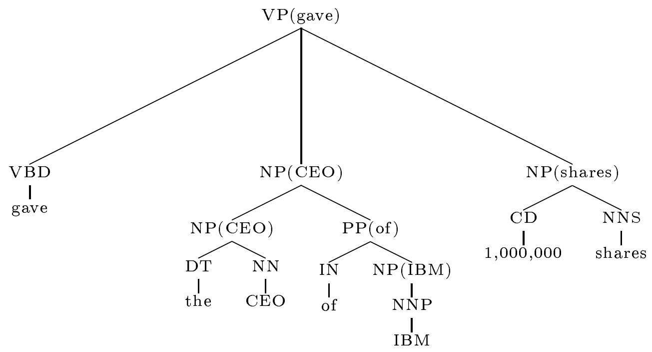
Figure 3.9 gives an extreme example of the utility of the *direction* feature. Without the *direction* feature parse 3.9(a) and parse 3.9(b) both contain plausible dependencies, and would receive roughly equal probability. With the *direction* feature the model will give much higher probability to parse 3.9(a), as $\langle of \rightarrow CEO, R \rangle$ should get much higher probability than $\langle of \rightarrow shares, L \rangle$ (prepositional phrases will virtually always be seen as post-modifiers to `NP`s in training data).

### 3.3.5 Parameterization Proposal 4: Dependencies + Direction + Relations

**Parameterization Proposal 4.** *(Dependencies + Direction + Relations) A parse tree is represented as n events $Event_1 ... Event_n$, where $Event_i$ is a $\langle w_i \rightarrow h_i, direction_i, relation_i \rangle$ tuple. $w_i \rightarrow h_i$ is the i'th dependency. $direction_i$ indicates the relative order of $w_i$ and $h_i$, as before. $relation_i$ is the grammatical relation associated with the dependency: it is a triple of non-terminals, $\langle modifier_i, parent_i, head_i \rangle$.*

Thus far we have ignored the non-terminals in the context-free tree when defining the dependency representation. This section introduces non-terminal information through a *relation* associated with each dependency. A relation is a triple of non-terminals defined in the following way:

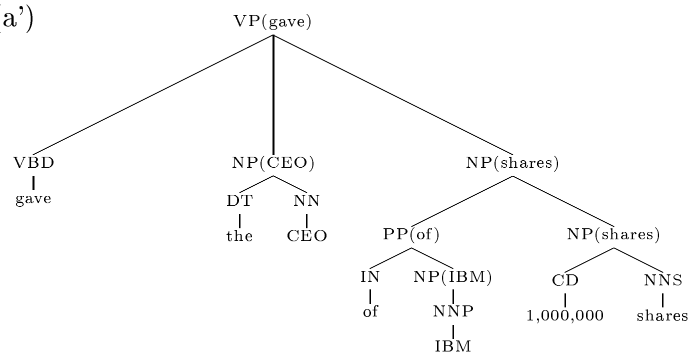
(a)




(b)

| Dependency | Direction |
|---|---|
| **of → CEO** | **R** |

(a')




(b')

| Dependency | Direction |
|---|---|
| **of → shares** | **L** |

Figure 3.9: Two tree fragments and their associated dependencies ((b) and (b') show the one dependency that differs between the two trees.) (a) should have relatively high probability; (a') should be very unlikely, as PPs can virtually never pre-modify `NP`s in English. Without the *direction* associated with each dependency, the two trees both contain plausible dependencies and the model assigns roughly equal probability to each.

- For each dependency $headword(Y_i) \rightarrow headword(Y_h)$ derived from rule $X \rightarrow Y_1 ... Y_n$, the associated *relation* is $\langle Y_i, X, Y_h \rangle$.
- If $X$ is the root non-terminal in the tree, and $x$ is its headword, then the dependency $x \rightarrow$ START has relation ⟨X, START, START⟩.

Figure 3.10 shows an example lexicalized tree with its associated dependencies and their relations. As an example of how a rule defines the relation for each dependency, take the rule VP $\rightarrow$ VBD NP PP in figure 3.10. It contributes two dependencies: $\langle headword(Y_2) \rightarrow headword(Y_1) \rangle = \langle sacks \rightarrow dumped \rangle$ and $\langle headword(Y_3) \rightarrow headword(Y_1) \rangle = \langle into \rightarrow dumped \rangle$. The associated relations are $\langle Y_2, X, Y_1 \rangle = \langle \texttt{NP,VP,VBD} \rangle$ and $\langle Y_3, X, Y_1 \rangle = \langle \texttt{PP,VP,VBD} \rangle$ respectively.

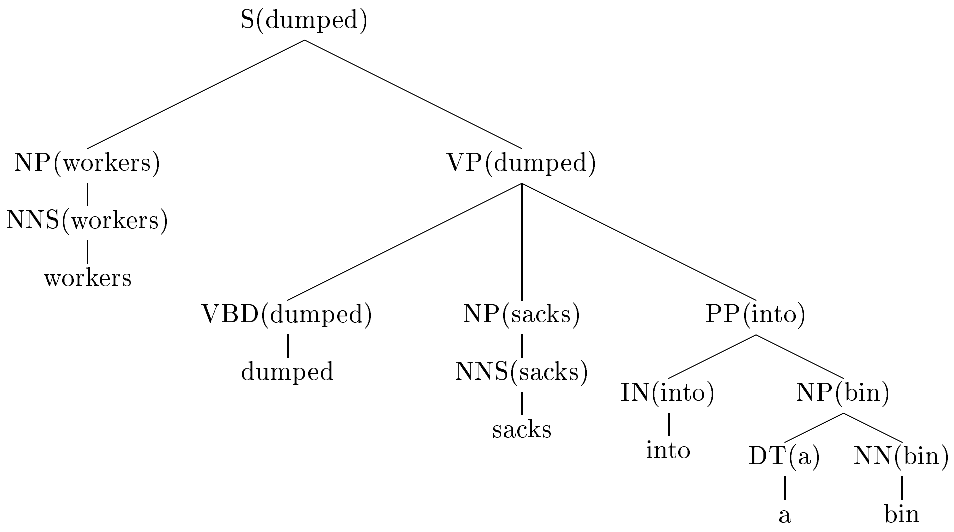
### Motivation for Relations

The inclusion of non-terminals is important for two reasons. First, if we are interested in recovering parse trees with the non-terminals, it is essential to include the non-terminals in the dependency representations. Otherwise, any lexicalized tree with the same headword structure, but arbitrary node labels, would have the same dependency structure and would therefore receive the same probability. Dependencies alone fail to discriminate between trees with different node labelings. With the relation triples, the mapping between a lexicalized tree and its associated set of dependencies is almost one-to-one, and different probabilities are assigned to different tree labelings.

There is a second reason for including the relation triples. Even if we were not concerned with recovering the tree non-terminals — recovering dependency structures alone was considered sufficient — the non-terminals would still be crucial in that they improve parsing accuracy. Whereas before a constituent was represented by its headword alone, it is now effectively represented by a ⟨ *headword, non-terminal* ⟩ pair. The addition of the non-terminal provides two pieces of additional information:

1. The major part-of-speech category for the word (noun, verb, preposition etc.). This is almost completely derivable from the non-terminal label (an `NP` almost always takes a noun as its head, a `PP` almost always takes a prepositional head, a `VP` or

(a)




(b)

| Dependencies | Directions | Relations |
|---|---|---|
| workers → dumped | L | ⟨ NP, S, VP ⟩ |
| dumped → START | — | ⟨ S, START, START ⟩ |
| sacks → dumped | R | ⟨ NP, VP, VBD ⟩ |
| into → dumped | R | ⟨ PP, VP, VBD ⟩ |
| a → bin | L | ⟨ DT, NP, NN ⟩ |
| bin → into | R | ⟨ NP, PP, IN ⟩ |

Figure 3.10: (a) a lexicalized tree. (b) a list of dependencies that the tree contains, with their direction and associated *relations*.

`S` almost always takes a verb as a head and so on). The importance of this POS information is due to the combination of two factors:

- POS tag ambiguity. A particular word may be ambiguous between several parts of speech. This means that a word can potentially be the head of quite different types of phrase.

  For example, *saw* can be verbal, as the head of a sentence such as *I saw the man*; or nominal, as in *the saw on the bench.*

- A head has strong restrictions about the POS type of its modifier. For example, a verb will take a subject to its left that is almost always an `NP`, and is never an `S`. The non-terminal information allows the head to specify this preference.

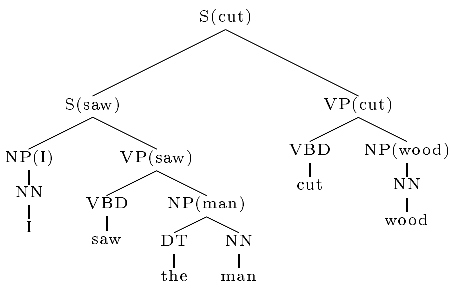
  For example, the verb "cut" may take a constituent headed by "saw" as a (subject) premodifier as long as it is the head of an `NP` (i.e., is the head of a phrase such as *the saw on the bench*), rather than the head of an `S` (as in *I saw the man*).

See figure 3.11 for an example of how these two characteristics may conspire to cause difficulties for a representation that lacks POS information.

2. The bar level for the constituent. A word with a particular part of speech may be the head of several different types of phrases: for example a verb may be the head of a single word constituent, a complete `VP`, a complete `S`, or even an `SBAR`; a noun may be the head of a phrase that is a bare noun, or may be the head of a complete `NP`. Beyond the simple POS distinctions, non-terminals make the further distinction between, for example, the noun (`NN`) *man* and the `NP` *the man in the park.*

   These distinctions are crucial, in that phrases with different bar levels are seen in quite different syntactic environments. A single noun can be a pre-modifier to another noun in a compound-nominal expression (e.g., as *cigarette* is in *cigarette filter*) but a full noun phrase can certainly not be a pre-modifier to another noun (e.g., as in *the (cigarette with a cigarette) filter*). See figure 3.12 for further illustration. As another example, a phrase headed by *to* can appear as a complement to a verb such as *force*

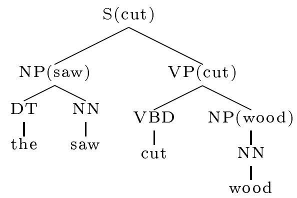
(a)




(b)

| Dependencies | Relations |
|---|---|
| I → saw | ⟨ NP, S, VP ⟩ |
| **saw → cut** | **⟨ S, S, VP ⟩** |
| the → man | ⟨ DT, NP, NN⟩ |
| man → saw | ⟨ NP, VP, VBD⟩ |
| cut → START | ⟨ S, START, START ⟩ |
| wood → cut | ⟨ NP, VP, VBD ⟩ |

(a')




(b')

| Dependencies | Relations |
|---|---|
| the → saw | ⟨ DT, NP, NN ⟩ |
| **saw → cut** | **⟨ NP, S, VP ⟩** |
| cut → START | ⟨ S, START, START ⟩ |
| wood → cut | ⟨ NP, VP, VBD ⟩ |

Figure 3.11: Two lexicalized trees, (a) and (a'), and the dependencies they contain, (b) and (b'). (a) should be a very unlikely parse: the dependency $\langle \textit{saw} \rightarrow \textit{cut}, \langle \texttt{S,S,VP} \rangle \rangle$ should get low probability. (a') is a likely parse, and contains a similar dependency $\langle \textit{saw} \rightarrow \textit{cut}, \langle \texttt{NP,S,VP} \rangle \rangle$. Without the non-terminal relations, both trees would contain the same dependency $\langle \textit{saw} \rightarrow \textit{cut} \rangle$, and the model would be unable to give low probability to (a) while giving high probability to (a').

providing it is an **S** rather than a **VP**.

### 3.3.6 Parameterization Proposal 5: Dependencies + Direction + Relations + Subcategorization

**Parameterization Proposal 5.** *(Dependencies + Direction + Relations + Subcategorization) A parse tree is represented as $n+m$ events $Event_1...Event_{n+m}$. Events $1...n$ are $\langle w_i \rightarrow h_i, direction_i, relation_i\rangle$ dependency tuples, as in proposal 4. Events $n+1...m$ are subcategorization frames. Each subcategorization frame is a* ⟨ `parent, head-child, head-word, direction, frame` ⟩ *tuple, where* `parent` *and* `head-child` *are non-terminals,* `head-word` *is a word,* `direction` *is either* `L` *or* `R`*, and* `frame` *is a multiset of non-terminals.*

The subcategorization frames associated with a tree are derived as follows:

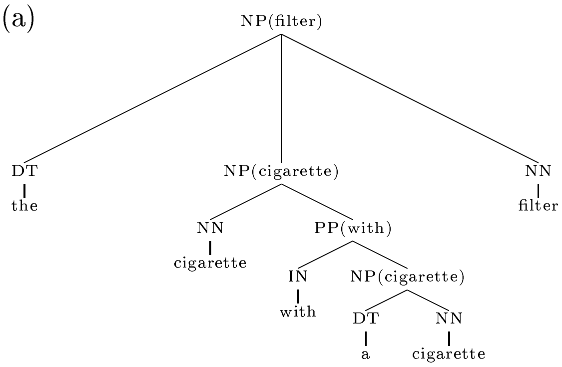
- We assume that the complement-adjunct distinction is made: that is, that a constituent can be identified as being a complement or an adjunct by inspection of its non-terminal label. (For the remainder of this section we will assume that complement non-terminals are marked with a `-C` suffix: `NP-C` would be an `NP` complement, `NP` would be an `NP` adjunct. Adding this distinction effectively doubles the number of non-terminals in the grammar.)
- Each rule $X \rightarrow Y_1...Y_n$ where $Y_1...Y_n$ are non-terminals, and $Y_h$ is the head non-terminal, contributes two subcategorization frames:

  1. $\langle X, Y_h, headword(X), L, frame_l\rangle$ where $frame_l$ is a multiset containing all complement non-terminals in the sequence $Y_1...Y_{h-1}$.
  2. $\langle X, Y_h, headword(X), R, frame_r\rangle$ where $frame_r$ is a multiset containing all complement non-terminals in the sequence $Y_{h+1}...Y_n$.

  For example, in the rule `S` $\rightarrow$ `NP-C VP` where `VP` is the head, and *dumped* is the headword of the phrase, the two subcategorization frames are ⟨`S,VP,dumped,L,`{`NP-C`}⟩ and ⟨`S,VP,dumped,R,`{}⟩.

Figure 3.13 shows a tree and its associated subcategorization frames.

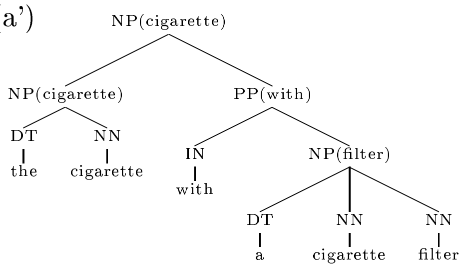
(a)




(b)

| Dependency | Relation |
|---|---|
| the → filter | DT, NP, NN |
| **cigarette → filter** | **NP, NP, NN** |
| with → cigarette | PP, NP, NP |
| a → cigarette | DT, NP, NN |
| cigarette → with | NP, PP, IN |

(a')

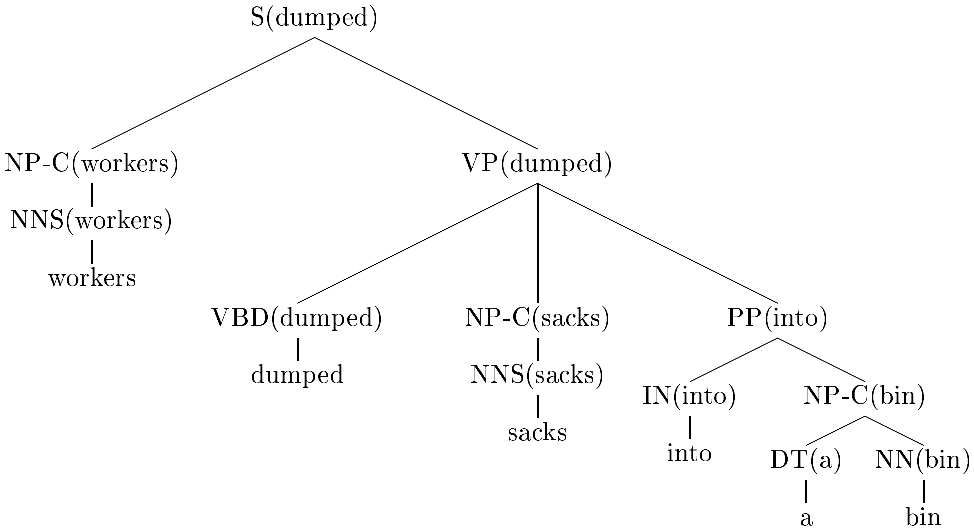



(b')

| Dependency | Relation |
|---|---|
| the → cigarette | DT, NP, NN |
| with → cigarette | PP, NP, NP |
| a → filter | DT, NP, NN |
| **cigarette → filter** | **NN, NP, NN** |
| filter → with | NP, PP, IN |

Figure 3.12: Two trees that contain a dependency $\langle cigarette \rightarrow filter \rangle$. Tree (a) should get low probability, as an `NP` cannot premodify a noun. In contrast, tree (a') should get relatively high probability. Without non-terminal relations the model fails to distinguish the dependencies $\langle cigarette \rightarrow filter, \langle$`NP, NP, NN`$\rangle\langle$ (low probability) and $\langle cigarette \rightarrow filter, \langle$`NN, NP, NN`$\rangle\langle$ (high probability), and cannot give low probability to (a) while giving high probability to (a').

(a)




(b)

| Rule | Associated Subcategorization Frames | |
|---|---|---|
| S $\rightarrow$ NP-C VP | ⟨`S,VP,dumped,L,`{`NP-C`}⟩ | ⟨ `S,VP,dumped,R,`{} ⟩ |
| VP $\rightarrow$ VBD NP-C PP | ⟨ `VP,VBD,dumped,L,`{} ⟩ | ⟨ `VP,VBD,dumped,R,`{`NP-C`} ⟩ |
| PP $\rightarrow$ IN NP-C | ⟨ `PP,IN,into,L,`{}⟩ | ⟨ `PP,IN,into,R,`{`NP-C`}⟩ |
| NP-C $\rightarrow$ NNS | ⟨ `NP-C,NNS,workers,L,`{}⟩ | ⟨ `NP-C,NNS,workers,R,`{}⟩ |
| NP-C $\rightarrow$ NNS | ⟨ `NP-C,NNS,sacks,L,`{}⟩ | ⟨ `NP-C,NNS,sacks,R,`{}⟩ |
| NP-C $\rightarrow$ DT NN | ⟨ `NP-C,NN,bin,L,`{}⟩ | ⟨ `NP-C,NN,bin,R,`{}⟩ |

Figure 3.13: (a) A lexicalized tree with the complement-adjunct distinction made. Complement non-terminals are marked with a `-C` suffix. (b) A list of the subcategorization frames associated with the tree (rules with a POS tag on their left hand side contribute no subcategorization frames, and are excluded from the table).

(a)

(a')

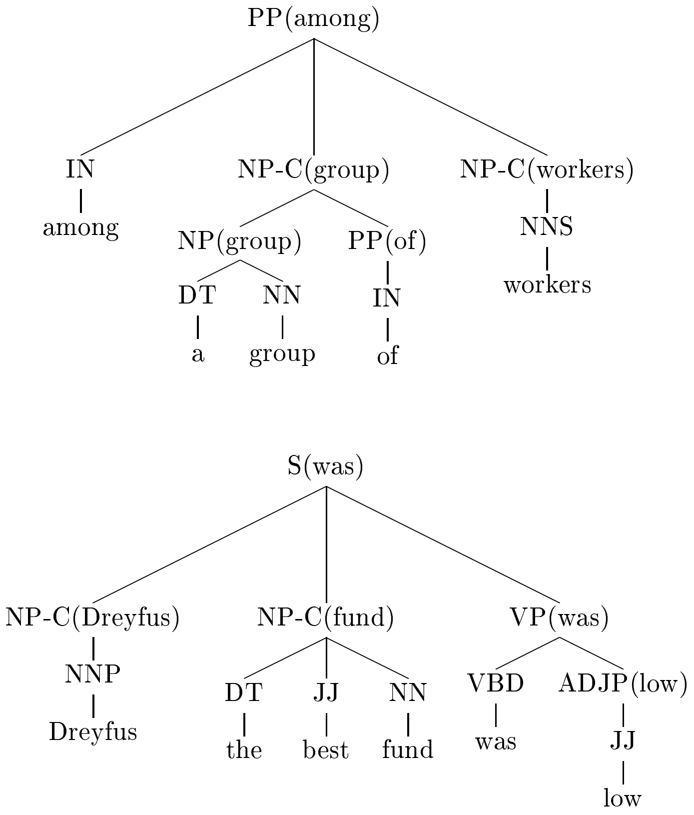



Figure 3.14: Two trees that should have low probability due to unlikely subcategorization frames. In each case the dependencies in the tree should all have high probability, and a representation based on dependencies alone will fail to give the trees low probability.

### The Motivation for Subcategorization

The need for subcategorization probabilities is not surprising, given that subcategorization is a major component of almost any syntactic theory. Figure 3.14 gives a couple of examples where subcategorization is needed to penalize bad parses. In both cases the trees have dependencies that are highly plausible, so that a representation based on dependencies alone will fail to give them low probability. The addition of subcategorization frames solves this problem: the tree in figure 3.14(a) will now have two very low probability frames $\langle$`PP,IN,among,R,`$\{$`NP-C, NP-C`$\}\rangle$ and $\langle$`PP,IN,of,R,`$\{\}\rangle$ (`PP`s in training data will almost always be seen with one complement, and very rarely if ever with zero or two complements); the tree in figure 3.14(a') will have a low probability frame $\langle$`S,VP,was,L,`$\{$`NP-C, NP-C`$\}\rangle$ (*was* will probably never be seen with two subjects in training data).

### 3.3.7 Parameterization Proposal 6: Dependencies + Direction + Relations + Subcategorization + Distance

**Parameterization Proposal 6.** *(Dependencies + Direction + Relations + Subcategorization + Distance) A parse tree is represented as* $n + m$ *events* $Event_1...Event_{n+m}$*. Events* $1...n$ *are* $\langle w_i \rightarrow h_i, direction_i, relation_i, distance_i \rangle$ *dependency tuples.* $w_i$*,* $h_i$*,* $direction_i$*, and* $relation_i$ *are as defined before.* $distance_i$ *is some function of the distance between* $w_i$ *and* $h_i$ *in the tree. Events* $n + 1...m$ *are subcategorization frames, as before.*

This section shows that adding a *distance* variable to each dependency allows the model to discriminate between parse trees with and without close attachment (i.e., to learn a preference for right or left branching structures). We first give an exact definition of the *distance* variable, and then give examples that motivate its utility. We also show how this distance measure can approximate subcategorization preferences, and actually greatly diminishes the need for subcategorization probabilities.

Finally, we give a second definition of distance. The first definition is that used by the parser in chapter 7. The second definition is used by the parser in chapter 6, and is a close approximation of the first distance measure, but with some significant failings that we will describe.

#### The First Definition of *distance*

To define the distance associated with each dependency we first define a *surface string* associated with each dependency:

- For each dependency $\langle headword(Y_i) \rightarrow headword(Y_h) \rangle$ derived from rule $X \rightarrow Y_1...Y_n$, such that $i < h$, the *surface string* associated with the dependency is the string indirectly dominated by non-terminals $Y_{i+1}...Y_{h-1}$ (the string is empty if $i = h - 1$).

- For each dependency $headword(Y_i) \rightarrow headword(Y_h)$ derived from rule $X \rightarrow Y_1...Y_n$, such that $i > h$, the *surface string* associated with the dependency is the string indirectly dominated by non-terminals $Y_{h+1}...Y_{i-1}$ (the string is empty if $i = h + 1$).

The *distance* feature is a function of the surface string associated with a dependency. The *distance* is a two bit string, with the following values:

**Bit 1** 1 if the string is empty, 0 if the string is non-empty. This feature indicates whether or not the modifier non-terminal is adjacent to the head.

**Bit 2** 1 if the string contains a verb, 0 if it does not contain a verb.

From these definitions there are three[5] possible values for the distance variable: `10`, which means that the head and modifier non-terminals are adjacent; `00`, which means they are non-adjacent, but there is no verb in the intervening string; `01`, which means that the intervening string does contain a verb. See figure 3.15 for example trees containing the same dependency with varying *distance* values.

### Motivation for the Distance Measure

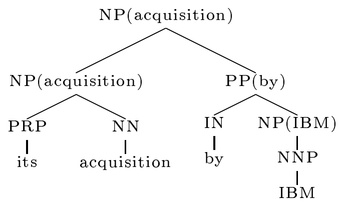
Figures 3.16 and 3.17 give examples where the distance measure is useful. In each case two competing trees differ by a single dependency, and the scores for these two discriminating dependencies are likely to be similar under the previous representation. The addition of the distance variable further differentiates the two dependencies, and allows the model to assign higher probability to dependencies involving close attachments.

### The Distance Measure as an Approximation of Subcategorization

If we now return to the example in figure 3.14(a) it becomes clear that the distance measure solves some of the problems concerning subcategorization. The second `NP` attaching to the preposition *among* gives a dependency $\langle workers \rightarrow among, \texttt{R}, \langle \texttt{NP,PP,IN} \rangle, 00 \rangle$, whereas the first `NP` attachment gives $\langle group \rightarrow among, \texttt{R}, \langle \texttt{NP,PP,IN} \rangle, 10 \rangle$. The distance variable differentiates between a dependency where the `NP` is/isn't adjacent to the preposition: the result is that $\langle workers \rightarrow among, \texttt{R}, \langle \texttt{NP,PP,IN} \rangle, 00 \rangle$ will get very low probability (a `PP` will almost never be seen with an `NP` modifier at distance `00`), and the parse tree will get a

[5] The fourth value, 11, is impossible because this would represent the contradiction that the string is both empty and contains a verb.

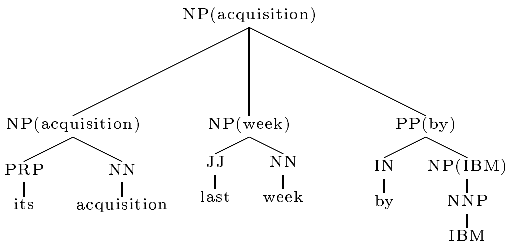
(a)




(b)

| Feature associated with *by* → *acquisition* | |
|---|---|
| Surface String | Distance |
| — | 10 |

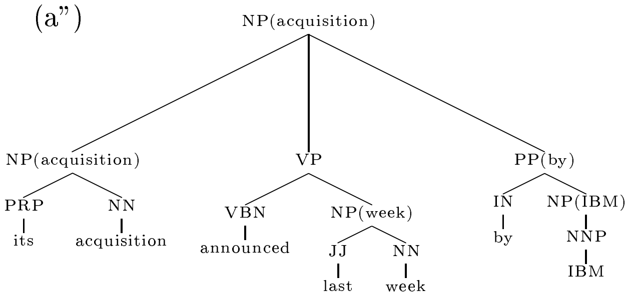
(a')




(b')

| Feature associated with *by* → *acquisition* | |
|---|---|
| Surface String | Distance |
| *last week* | 00 |

(a")

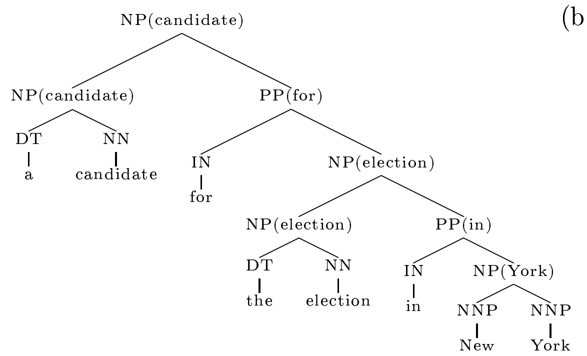



(b")

| Feature associated with *by* → *acquisition* | |
|---|---|
| Surface String | Distance |
| *announced last week* | 01 |

Figure 3.15: Three trees that contain a dependency $\langle by \rightarrow acquisition \rangle$. The surface string and distance features are shown for each dependency.

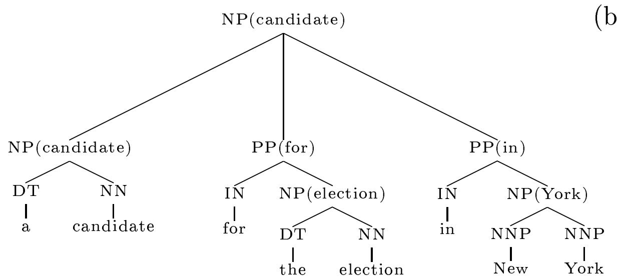
(a)




(b)

| Dependency | Distance |
|---|---|
| in → election | 10 |

(a')

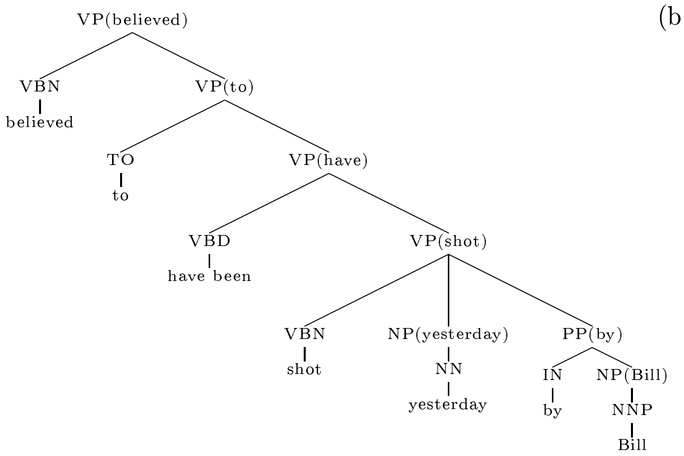



(b')

| Dependency | Distance |
|---|---|
| in → candidate | 00 |

Figure 3.16: Two competing trees which differ by a single dependency, $\langle in \rightarrow election \rangle$ vs. $\langle in \rightarrow candidate \rangle$. Without the distance variable, each dependency looks quite plausible and their probabilities are likely to be similar — the decision between the two structures will be a close one. With the distance variable the model can discriminate between the first attachment as close attachment (distance 10) and the second attachment as longer distance attachment (distance 00). (In fact, the right branching structure in (a) occurs about twice as often in the Penn WSJ treebank as the alternative structure in (a').)

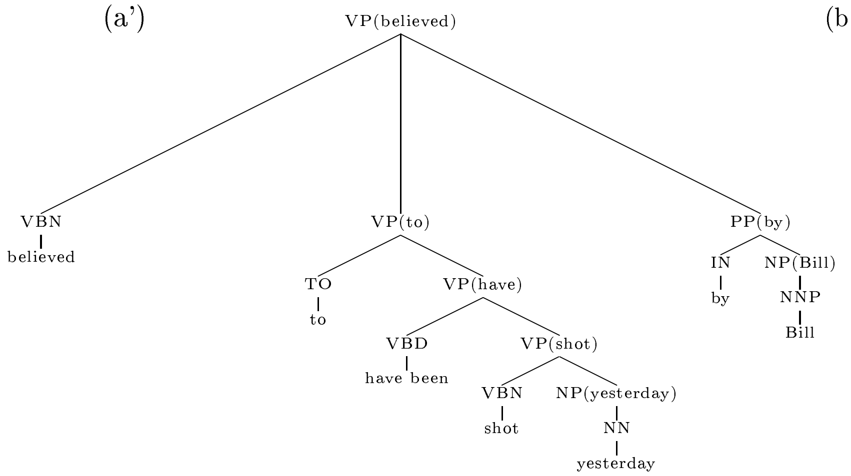
(a)




(b)

| Dependency | Distance |
|---|---|
| by $\rightarrow$ shot | 00 |

(a')




(b')

| Dependency | Distance |
|---|---|
| by $\rightarrow$ believed | 01 |

Figure 3.17: Two competing tree fragments which differ by a single dependency, $\langle$*by* $\rightarrow$ *shot*$\rangle$ vs. $\langle$*by* $\rightarrow$ *believed*$\rangle$. Without the distance variable, each dependency looks quite plausible and their probabilities are likely to be similar — the decision between the two structures will be a close one. With the distance variable the model can discriminate between the first attachment as an attachment that does not cross a verb (distance 00) and the second attachment as one that does cross a verb (distance 01). (In fact, the right branching structure in (a) occurs about 19 times as often in the Penn WSJ treebank as the alternative structure in (a').)

low score. In practice, the addition of distance allows the model to learn that a PP (almost always) subcategorizes for a single complement.

This result is not restricted to PPs alone. It also applies to other heads which take a single obligatory argument: the most common cases being a complementizer taking a sentential complement, or a transitive verb taking a single NP complement. Later, experimental results show that the distance and subcategorization features overlap a great deal in their utility, and that a model with distance but without subcategorization performs almost as well as a model that includes both distance and subcategorization.

### A Second Definition of *distance*

The first distance measure, just described, is used in the parser in chapter 7. We now describe a second measure that was used in the earlier parser in chapter 6. It can be considered an approximation of the first distance measure. The difference between the two distance measures was forced by the difference between the two probability models. Although the two distance measures are approximately the same, the second measure breaks down as a close-attachment or subcategorization preference in some cases. (The model structure in chapter 6 does not allow a natural incorporation of subcategorization probabilities, so the breakdown of distance as an approximation of subcategorization is particularly problematic.)

The second distance measure differs only in its definition of the *surface string* associated with a dependency:

- The surface string associated with a dependency $w_i \to h_i$ is the surface sequence of words between $w_i$ and $h_i$.

The distance measure is then defined as a function of the surface string, in the same way as before:

**Bit 1** 1 if the string is empty, 0 if the string is non-empty. This feature indicates whether or not the modifier word is adjacent to the head.

**Bit 2** 1 if the string contains a verb, 0 if it does not contain a verb.

Whereas the first definition of distance depended at least partially on the phrase structure of the tree, this second definition depends only on the surface position of the $head$ and $modifier$ in the surface string.

In some situations the two distance measures will give the same results: in fact, in figures 3.15, 3.16 and 3.17 this new distance measure will give the same results as the first distance measure for the highlighted dependencies.

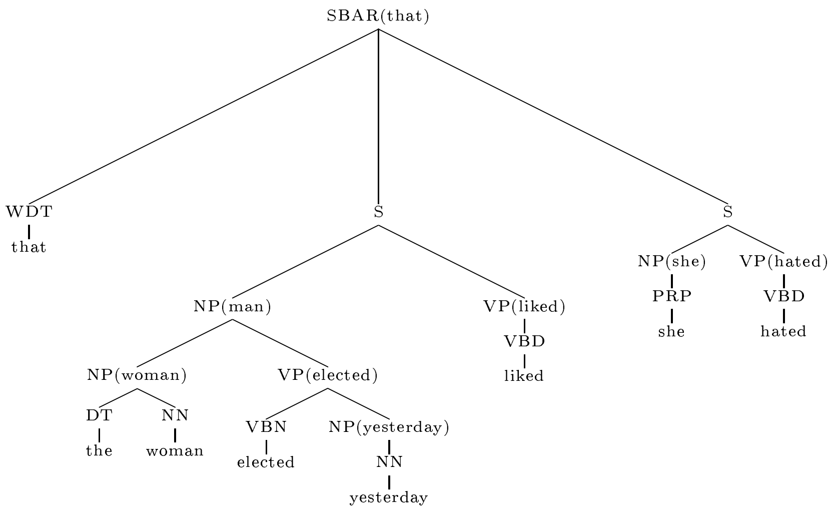
Consider, however, the tree in figure 3.14(a). With the first distance measure, the two dependencies involving the preposition *among* were $\langle workers \rightarrow among, R, \langle \texttt{NP, PP,IN} \rangle, 00\rangle$ and $\langle group \rightarrow among, R, \langle \texttt{NP,PP,IN} \rangle, 10\rangle$. In contrast *group* is *not* adjacent to *among* in the surface string, so the second distance measure gives the dependency $\langle group \rightarrow among, R, \langle \texttt{NP,PP,IN} \rangle, 00\rangle$: the distance variable now fails to differentiate the two dependencies and the approximation of subcategorization has broken down. This kind of problem lead to a refinement of the distance measure in chapter 6, through a redefinition of the surface string:

- The surface string associated with a dependency $w_i \rightarrow h_i$ is the surface sequence of words between $w_i$ and $h_i$, excluding words that appear as pre or post modifiers within non-recursive `NP`s.

By this definition, the premodifier *the* is excluded from the string between *among* and *group*, leading to a dependency with the "correct" distance measure: $\langle group \rightarrow among, R, \langle \texttt{NP,PP,IN} \rangle, 10\rangle$.

The redefinition of the surface string was not trivial to incorporate in the probabilistic model in chapter 6, involving a definition of a separate level for non-recursive `NP` recovery. We can see from the example in figure 3.18 that the second distance measure is still different from the first in some cases. In this example the difference leads to a failure to approximate the subcategorization fact that a complementizer takes a single `S` complement. The remaining differences between the two distance measures is no doubt one of the reasons that the chapter 7 parser performs better than the chapter 6 parser.

(a)




(b)

| Dependency | String1 | Distance1 | String2 | Distance2 |
|---|---|---|---|---|
| liked → that | — | 10 | *the woman elected yesterday* | 01 |
| hated → that | *the woman elected yesterday liked* | 01 | *the woman elected yesterday liked she* | 01 |

Figure 3.18: A tree, and the distance measure assigned by the first and second distance measures: String1, Distance1 are the features assigned by the first measure; String2, Distance2 are assigned by the second measure. Notice that the two distance measures give differing results, and importantly the first distance measure differentiates the `S` complement adjacent from the head from the second `S` complement. In contrast the second distance measure fails to differentiate the two dependencies, and will be forced to give similar probabilities to each, thereby failing to learn the subcategorization fact that a complementizer generally takes a single `S` complement.

### 3.3.8 Parameterization Proposal 7: Dependencies + Direction + Relations + Subcategorization + Distance + Parts-of-Speech

**Parameterization Proposal 7.** *(Dependencies + Direction + Relations + Subcategorization + Distance + Parts-of-Speech) A parse tree is represented as* $n+m$ *events* $Event_1 ... Event_{n+m}$*. Events* $1...n$ *are* $\langle w_i/tag(w_i) \rightarrow h_i/tag(h_i), direction_i, relation_i, distance_i \rangle$ *dependency tuples.* $w_i$*,* $h_i$*,* $direction_i$*,* $relation_i$ *and* $distance_i$ *are as defined before.* $tag(w_i)$ *and* $tag(h_i)$ *are the part of speech tags in the tree associated with words* $w_i$ *and* $h_i$ *respectively.*

In the final refinement of the parse tree representation, part of speech tags are added. The relation tuple already includes non-terminals associated with the words in a dependency, and therefore distinguishes the major part-of-speech category for the words in the dependency (as argued in section 3.3.5). The addition of parts of speech makes finer graded distinctions: for example distinguishing singular, plural and proper nouns, or distinguishing different verb forms.

The motivation for including parts of speech is tied to the estimation methods used for calculating parameter values. Without sparse data problems, the count of full dependency relations $Count(\langle w_i/tag(w_i) \rightarrow h_i/tag(h_i), direction_i, relation_i, distance_i \rangle)$ would be reliable enough to be the sole basis for estimation. But all such counts being reliable is extremely unlikely to be the case[6].

The estimation methods attack the sparse data problem by incorporating counts from subsets of this full representation (with probability theory solving the delicate problem of exactly how to combine these counts). These subset counts correspond to differing levels of generalization. For example: $Count($`IBM/NNP` $\rightarrow$ `bought/VBD, L,`$\langle$`NP,S, VP`$\rangle, 10)$ would be number of times *IBM* is seen as the subject of *bought*; $Count($`IBM/NNP` $\rightarrow$ `VBD, L,`$\langle$`NP,S, VP`$\rangle, 10)$ would be the number of times *IBM* is seen as the subject of any verb tagged `VBD`; $Count($`NNP` $\rightarrow$ `VBD, L,`$\langle$`NP,S, VP`$\rangle, 10)$ is the number of times a noun

[6]In the Penn WSJ treebank there are approximately 1,000,000 dependency events on which to base counts; with a (conservative) estimate of the vocabulary size as 20,000 words there would be $20,000^2 = 40,000,0000$ parameters even before considering the direction, distance and relation features. It's clear that the number of parameters vastly exceeds the number of training events, and that the sparse data problem is severe.

tagged as `NNP` is seen as the subject of a verb tagged `VBD`.

The addition of part of speech tags allows the model to learn useful generalizations about how whole part-of-speech classes behave. The subset counts will generally discard the (sparse) lexical items but retain the (much less sparse) POS tags. For example, $Count(\langle tag(w_i) \rightarrow tag(h_i), direction_i, relation_i, distance_i \rangle)$ may be quite reliable, and will encode many useful facts about how different parts-of-speech behave. For example, that singular or plural nouns are much more likely to take PP modifiers than proper nouns, or that a subject verb dependency must involve POS tags that agree for person and number, and so on.

## 3.4 Summary

This chapter initially showed that a "simple" PCFG has a couple of major failings: namely, a lack of sensitivity to lexical information and structural preferences. This motivated the move to dependencies as a representation. A tree for a sentence with $n$ words contains $n$ dependencies between pairs of words; associating probabilities with these dependencies leads to a model which is much more sensitive to lexical information. However, simple dependencies are not enough. We described a series of additions to the parameterization: word-order information, non-terminals encoding grammatical relations, subcategorization events, distance preferences, and the use of part-of-speech tags as word-class information. The remaining question is how to build a statistical model, using the techniques described in chapter 2, which includes all of these parameter types.

# Chapter 4

# Previous Work

## 4.1 Introduction

This chapter gives a broad overview of the literature on statistical parsing of natural language. We first give a brief history of the field, then describe each paper in detail. Note that chapter 8 of this thesis gives a more detailed comparison to previous work that is of the most direct relevance to the work in this thesis.

## 4.2 A Brief History of Probabilistic Parsing for Natural Language

### Goals of the field

When considering the work in this chapter, it is useful to bear in mind two distinct goals driving the research:

- The first is the topic of this thesis: i.e., building a parser that recovers linguistic structures with high accuracy. Thus the goal of this research is to maximize parse accuracy.
- The second is related to speech recognition: i.e., building a *language model*, a model that assigns probabilities to strings in a language. Trigram language models (e.g., see

[Jelinek 90]) have been very successful in speech recognition, but have clear weaknesses in modeling the grammaticality of strings, or capturing dependencies between pairs of words that are more than two words apart. Language models containing hierarchical or linguistic structure might lead to improved recognition performance. Thus this second goal is to build statistical grammars that give improved speech recognition performance over trigram models, or that give improved *perplexity*[1] results over trigram models.

The first goal is of most direct relevance to this thesis, but it is perhaps important to be aware of the second goal when analysing the underlying motivation for some of the work discussed in this chapter.

Both supervised and unsupervised methods have been pursued (supervised training uses a treebank of example sentence/tree pairs as training data, unsupervised training uses unanalysed text). While unsupervised training is clearly preferable, in that it removes the need for costly hand-annotation of a treebank, it is also generally acknowledged to be a much harder problem (at least for the goal of learning linguistically plausible structures).

**Availability of Treebank Data**

An important factor influencing research in the field has been the availability of parsed treebank corpora. This is needed for training supervised models, and for testing the parsing accuracy of both supervised and unsupervised models.

A major source of publicly available data is the Penn treebank [Marcus et al. 93]. The ATIS and Wall Street Journal (WSJ) sections of the first version of the treebank (TBI) became available in mid-to-late 1991. The ATIS and WSJ sections of the second version (TBII) became available in mid-to-late 1994 (this data is used in this thesis). TBII was superior to TBI in a couple of important respects: the annotations were checked more carefully, leading to greater consistency; and the annotation style was much improved, being sounder linguistically and having more information, in terms of semantic labelings and other indications of predicate-argument structure [Marcus et al. 94].

[1] Perplexity is a measure of how well a language model predicts some previously unseen data, and is at least perceived to be strongly related to speech recogniser performance; see [Jelinek 90] for details.

The lack of treebank data in the early years of statistical parsing research lead to a number of different training and evaluation approaches. Researchers at IBM [Black et al. 92a, Black et al. 92b, Jelinek et al. 94] had developed their own treebank corpora, for example on a computer manuals domain. Some researchers used the first version of the Penn treebank [Bod 93, Pereira and Schabes 92, Schabes et al 93]. Others developed their own training and test data, often by hand-selecting between multiple parses produced by an existing parser [Alshawi and Carter 94, Briscoe and Carroll 93, Carroll and Briscoe 95]. Several papers were theoretical, developing a probabilistic version of existing grammars such as TAGs or link grammars, often additionally deriving algorithms for parsing and unsupervised training [Resnik 92, Schabes 92, Schabes and Waters 93, Lafferty et al. 92], but not giving any evaluation of parsing accuracy.

**Perceived Problems with Probabilistic Context Free Grammars**

Probabilistic context free grammars (PCFGs) were a natural starting point for the research in statistical parsing of natural language. Their formal properties have been well understood since at least [Booth and Thompson 73]; efficient parsing algorithms are well known; and [Baker 79] describes the inside-outside algorithm, an efficient approach to EM parameter estimation [Dempster, Laird and Rubin 77], for unsupervised training.

Unfortunately, research suggested that PCFGs were poor models of language in several respects. In the unsupervised case the inside-outside algorithm was unsuccessful at inducing linguistically plausible structures (see, for example, [Pereira and Schabes 92]); neither did it lead to models that reduced perplexity for the language modeling task (almost certainly because PCFGs do not have the parameters corresponding to pairs or triples of words that make trigram models so successful). In the supervised training case, PCFGs were again perceived to be poor models, although more recent work [Charniak 97, Charniak 96] has shown that PCFGs can achieve at least respectable results on parsing the Penn WSJ corpus, version 2.

These perceived failures lead to areas of research that are described in the next three sections: (1) designing models with increased structural sensitivity; (2) the development of models containing parameters corresponding to lexical dependencies; and (3) the development of *history-based* models.

### Probabilistic Methods with Increased Structural Sensitivity

Several researchers looked at increasing the context-sensitivity of PCFGs, with encouraging results: e.g., [Magerman and Marcus 91, Magerman and Weir 92, Briscoe and Carroll 93, Bod 93]. [Brill 93] considered a rule-based learning model, again with more context sensitivity than a PCFG. Other researchers such as [Pereira and Schabes 92, Schabes et al 93, Black et al. 92a] considered partially supervised versions of the Inside-Outside algorithm: the idea being that treebanks such as TBI had relatively flat, underspecified trees, and that learning algorithms should be able to use this information while learning more detailed structure in an unsupervised manner.

All of these models retained PCFG's weakness of a lack of lexical sensitivity; the next two areas of research addressed this problem.

### Formalisms Including Lexical Dependencies

There were at least two reasons for developing models that included dependency parameters. First, researchers who were interested in language modeling for speech recognition realised that while trigram models might make poor *syntactic* models[2], the probabilities associated with pairs or triples of words were very useful when assigning probabilities to sentences in a language. It followed that for structured models to compete as language models, they would have to include such parameters (see for example [Lafferty et al. 92]). Second, and more importantly for the research in this thesis, research had suggested that dependency probabilities might be powerful sources of disambiguating information. [Hindle and Rooth 91] had shown their use in prepositional phrase disambiguation; and as early as 1990 arguments were made for the generalization of this method to full parsing (text taken from [Marcus 90]):

> Ken Church argued that parameterization on purely structural relations, such

[2] By syntactic models we mean models that distinguish grammatical from ungrammatical sentences, see the arguments in [Chomsky 57] for why Markov models fail in this respect.

as used in this and the previous paper would be strikingly less successful than parameterization on words, parameterizing perhaps (as in Hindle and Rooth's paper) on pairs of words in certain structural relations.

There were two types of work that attempted to generalize the method described in [Hindle and Rooth 91]. First, [Sekine et al 92, Alshawi and Carter 94, Jones and Eisner 92a] used an existing hand-crafted parser that recovered predicate-argument relations. Scores were added to the predicate-argument relations to rank different parses. The method in [Sekine et al 92] used unsupervised learning; [Alshawi and Carter 94, Jones and Eisner 92a] used supervised learning. The results were very promising: [Alshawi and Carter 94] report 89% exact match accuracy on the ATIS domain using the predicate-argument parameters alone, with 94% accuracy when other features were added.

A second line of research was to extend various lexicalized syntactic formalisms to the statistical case, as in stochastic tree adjoining grammars (STAGS) [Resnik 92, Schabes 92] and link grammars [Lafferty et al. 92]. A natural consequence of these formalisms was to give dependency parameters. Both [Schabes 92] and [Lafferty et al. 92] derive versions of the inside-outside algorithm for unsupervised training. These papers represent important work, as they give well-founded probability models including dependency parameters; but the lack of availability of training and test data means they are limited in their lack of evaluation, which inevitably leads to a lack of detail in the modeling choices, or to modeling choices that would most likely hurt parsing performance.

**History-Based Models**

A third line of research, history-based models, was developed by researchers at IBM [Black et al. 92a, Black et al. 92b, Jelinek et al. 94]. These models were characterized by two differences from simple PCFGs. First, the parse-tree representation was enriched in a couple of ways: non-terminal labels were extended to include information such as lexical items (head words), or semantic categories; and the conditioning context was extended to look at potentially all previously built structure, rather than just the non-terminal being expanded as in PCFGs. Second, more powerful machine-learning methods, in particular decision trees, were used for parameter estimation. The basic idea was to expand the conditioning features and context to (hopefully) include practically all sources of disambiguating information; then to use decision trees to learn exactly what features or combinations of features were actually important for parsing.

An important development in this research was a move from the use of a hand-crafted grammar in [Black et al. 92a, Black et al. 92b], to a model that was trained from a treebank alone in [Jelinek et al. 94].

### Statistical Models for Parsing the Penn WSJ Treebank

[Magerman 95] described the SPATTER parser — an extension of the model described in [Jelinek et al. 94] — applied to the Penn WSJ Treebank (version 2). The model was trained on 40,000 sentences, and tested on over 2,000 sentences. In many ways this work represented a maturation of the work in statistical parsing:

- It represented a major advance in the scale of the tasks undertaken by statistical parsers. All sentences up to 40 words in length were parsed, on a domain (WSJ) that is much less restricted than previous domains such as ATIS or the IBM computer manuals data.

- The parser was trained completely automatically from the treebank, with no requirement for a hand-crafted grammar.

- The results represented a major improvement over accuracy for PCFGs: 84.5/84.0% precision/recall on section 23 of the treebank ([Charniak 97] later reported that a non-lexicalized PCFG scores around 72% averaged precision/recall on this task).

- The model had parameters that conditioned heavily on lexical information, presumably accounting for much of its improvement over PCFG based methods.

The work in chapter 6 was originally published in [Collins 96], and was well underway by mid to late 1995; because of this, the work in [Magerman 95] was a crucial benchmark for the work. Since then, several other results have been reported on parsing the Penn WSJ treebank: [Ratnaparkhi 97] describes a history-based parser based on maximum entropy models; [Charniak 97, Goodman 97, Eisner 96] all describe methods that rely heavily on dependency probabilities. Thus [Magerman 95, Eisner 96, Ratnaparkhi 97, Charniak 97, Goodman 97] are all highly relevant to the work in this thesis, and are described briefly in section 4.8 of this chapter with a much more detailed comparison in chapter 8.

## 4.3 Five Categories of Previous Work

We now describe each of the papers in more detail. We divide previous work into five broad categories:

**Probabilistic Methods without Lexical Sensitivity** This section first describes recent work on parsing the Penn WSJ treebank using (non-lexicalized) PCFGs. It then goes on to describe refinements to PCFG models: partially supervised training algorithms, parameterizations that give increased structural sensitivity, and a few other topics.

**Rule-Based Learning Methods** This section describes work on applying transformational based learning to parsing, and decision tree learning of a deterministic shift-reduce parser.

**Ranking Parse Trees through Scores Associated with Semantic Tuples** The papers in this section use hand-written parsers that recover semantic tuples (tuples specifying pairs of head-words in some predicate-argument relationship) together with a syntactic parse. The papers describe methods of assigning scores to semantic tuples, thereby defining a function for ranking parse trees.

**Probabilistic Versions of Lexicalized Grammar Formalisms** This section describes work on probabilistic versions of grammar formalisms such as tree adjoining grammars, link grammars, lexicalized PCFGs, and head automata. This work is highly relevant to this thesis, as these formalisms include parameters corresponding to lexical dependencies.

**Previous Work on Probabilistic Parsing of the Penn WSJ Treebank** The work in this section is of most direct relevance to this thesis, as it describes statistical models that make strong use of lexical information, and have been evaluated on wide-coverage parsing (the WSJ treebank). This chapter gives a brief overview of this work; chapter 8 compares the work in this thesis to these papers in much greater detail.

## 4.4 Probabilistic Models without Lexical Sensitivity

### 4.4.1 Results for PCFGs on the Penn WSJ Treebank

**[Charniak 97, Charniak 96]**

Although there was much early work on PCFG-based parsing of natural languages, the most directly relevant result to this thesis is given in [Charniak 97]. [Charniak 97] describes a lexicalized PCFG model that we will discuss extensively in section 8.4.1, and also gives results for a non-lexicalized PCFG as a baseline. On sentences of length $\leq$ 100 words, the non-lexicalized PCFG scores 70.6% recall, 74.8% precision. This result is directly comparable to the results in this thesis, as the model was trained and tested on the same data, and evaluated using the same measures of accuracy.

[Charniak 96] also describes results for a PCFG trained and tested on the treebank. The results are less comparable: the training and test data differ from the sections used in this thesis, and evaluation is only given on the recovery of unlabeled constituents. The paper makes a number of important observations though. First, it shows that adding a structural bias to the grammar for right-branching structures gives a 2.4/2.7% improvement in recall/precision (although this change means that the model does not sum to one without renormalization, with the consequence that the relative frequency estimates used are not maximum-likelihood estimates; see section 2.3.3 of this work for more about this problem). Second, the paper considers the coverage problem in some depth. Inducing a PCFG directly from the Penn WSJ treebank inevitably leads to a grammar with imperfect coverage of test data: some test data sentences will require rules that have never been seen in training. [Charniak 96] argues that these problems are not harmful, we return to this point in section 8.3.

### 4.4.2 Partially Supervised Training of PCFGs

**[Chitrao and Grishman 90]**

[Chitrao and Grishman 90] describe a method that uses an existing parser — the PROTEUS parser — with EM unsupervised training of its rule probabilities (the model is a PCFG). They also describe a model with increased context-sensitivity, the expansion probability for a non-terminal being conditioned on its parent. The model was trained on 300 sentences, and tested on 140 sentences (these sentences were a subset of the 300 training sentences; this is not a problem because the method uses unsupervised training). They show a decrease in the number of incorrect parses from 44% without statistics to 26% with the use of statistics.

**[Pereira and Schabes 92]**

[Pereira and Schabes 92] describe an extension of the inside-outside algorithm [Baker 79] to the case where a *partially* bracketed corpus is used as (partially supervised) training data for a binary-branching PCFG. The benefits of this approach over the algorithm in [Baker 79] are two-fold: first, the algorithm is more efficient (in the limit, if the corpus contains fully-bracketed binary branching trees, the algorithm runs in $O(n)$ time, as opposed to $O(n^3)$ time for [Baker 79]); second, the grammar induced by the algorithm is partially constrained by the partial bracketings, thus it is likely to learn whatever information is in these partial bracketings, while learning more detailed structure within the bracketings in an unsupervised manner. Tests are made on the ATIS corpus: 90.36% bracketing accuracy (bracketing accuracy is the percentage of constituents that do not cross a constituent in the gold-standard parse) is reported for the [Pereira and Schabes 92] algorithm, as opposed to 37.35% accuracy for [Baker 79]. Thus the unsupervised method is shown to be very poor at inducing linguistic structure automatically, while the partially supervised method is shown to be constrained enough to give good results. Interestingly, the two induced grammars show very similar perplexity measurements on the corpus, in spite of giving radically different parsing accuracies.

### [Schabes et al 93]

[Schabes et al 93] describe the application of the method in [Pereira and Schabes 92] to the WSJ corpus. In this case the corpus is converted to completely binary branching trees, so the learning algorithm is completely constrained for bracketing, though it is free to learn non-terminal labelings in an unsupervised fashion. On sentences up to 15 words in length, they report bracketing accuracies of around 90%.

### [Black et al. 92a]

[Black et al. 92a] describe a statistical parsing model that assumes two resources: 1) a treebank of tree/sentence pairs for the domain under consideration; 2) a hand-crafted grammar with good coverage of this domain. The goal is to induce a PCFG model of the hand-crafted grammar (i.e. to estimate probabilities associated with the rules in the hand-crafted grammar). These parameters are estimated in a partially supervised way using the inside-outside algorithm on the treebank data: the trees in the treebank provide constraints in terms of bracketings and non-terminal labels (the learning algorithm is very similar to that in [Pereira and Schabes 92]). The method is shown to recover the correct parse 75% of the time on a computer manuals domain, with restrictions that 1) sentences in the test set are 7–17 words in length; and 2) sentences are comprised of a restricted vocabulary of the 3000 most frequently occurring words in the domain.

## 4.4.3 Methods with Increased Structural Sensitivity

### [Magerman and Marcus 91, Magerman and Weir 92]

[Magerman and Marcus 91] describe $\mathcal{P}$earl, a probabilistic parser; [Magerman and Weir 92] look more closely at efficient search strategies within $\mathcal{P}$earl, and give further empirical evaluation. $\mathcal{P}$earl's major departure from PCFG formalisms is to increase the context-sensitivity of the parameters in the model. In a PCFG each rule $A \rightarrow \beta$ has probability $P(A \rightarrow \beta | A)$; in $\mathcal{P}$earl the parameters are extended to condition on the rule $C \rightarrow \alpha A \gamma$ that generated $A$, and on the POS trigram $a_0 a_1 a_2$ in the sentence such that $a_1$ is the left-most word in the constituent. (The new parameters are written $P(A \rightarrow \beta | C \rightarrow \alpha A \gamma, a_0 a_1 a_2)$.)

Evaluation showed results of 88% accuracy in recovering the correct parse in the Voyager domain. The model did not use lexical information, parsing POS strings as input.

**[Briscoe and Carroll 93]**

[Briscoe and Carroll 93] describe a probabilistic model of LR parsing, based on the Alvey Natural Language Tools (ANLT) parser. Probabilities are associated with actions in an LR parse table, rather than with context-free rules: the model is more sensitive to structural distinctions than simple PCFGs, as the probability of applying a rule is conditioned on the LR state, not just the non-terminal being expanded. However, it is likely to have the same insensitivity to lexical information as simple PCFGs. [Briscoe and Carroll 93] note that simple PCFGs cannot encode structural preferences in examples such as compound nominal or prepositional phrase ambiguities (for example the analyses [[N N] N] vs. [N [N N]] contain the same rules, and therefore can't be distinguished by a simple PCFG), but that the LR probabilities can discriminate between these alternative analyses. The method is evaluated on 55 dictionary definitions of length up to 10 words, with a training set of 246 definitions. 41 (75%) of the definitions are parsed correctly. [Carroll and Briscoe 95] describe considerable progress in scaling the system. The system described there recovers at least one analysis for 80% of sentences in a diversified corpus; evaluation of 250 sentences that were covered showed labeled constituent recall/precision results of 82.9/83.9%.

**[Bod 93]**

[Bod 93] describes the application of *Data Oriented Parsing* (DOP) to the ATIS corpus. The DOP model can be viewed as a variant of Stochastic TAG [Resnik 92, Schabes 92] restricted to substitution only. The key distinguishing feature to DOP is how the parameters of this STAG are estimated from a corpus. Given a set of context-free trees in a treebank, there are multiple possible STAG derivations for each tree: the underlying derivation for a particular tree could involve sub-trees ranging in size from 1-level context-free productions, to an entire tree spanning the whole sentence. Thus the derivation underlying a tree is "hidden", and the parameter values cannot be estimated directly. [Bod 93] describes a method that first extracts all partial trees from a treebank, and then assigns probabilities to them in a way that gives a well-formed probability distribution over the space of possible derivations (the estimation method is perhaps rather ad-hoc — for example not maximizing the likelihood of the training set of example trees — but is probably quite robust; in effect the model smooths the probability for large tree fragments with probabilities for smaller trees). Given a test-data sentence, the probability of a candidate parse tree can be calculated as the sum of probabilities for derivations underlying the tree. Search for the highest probability tree is computationally expensive, given a requirement for summation over all derivations for each tree; [Bod 93] reports use of a Monte Carlo style algorithm to find the highest probability tree.

The model is interesting in that it radically extends the structural sensitivity of PCFGs to much larger tree fragments. The work in [Bod 93] does not include lexical information (parsing was done over POS strings), but there is nothing in principle to prevent extension of the model to lexicalized grammars. Unfortunately the computational complexity of parsing with the model may make it difficult to scale to more complex corpora than ATIS, or to the lexicalized case. [Bod 93] reports very good results (96% accuracy at recovering parse trees on the ATIS corpus). However, these results have not been replicated: see [Goodman 96] for a lengthy discussion. [Goodman 96] describes an efficient, but possibly approximate, implementation of DOP. He also reports results on ATIS for DOP, and compares to the PCFG method described by [Pereira and Schabes 92]: in these experiments DOP performs with moderately greater accuracy than a PCFG model. (Approximately 66.1% of sentences receive 0 crossing brackets for DOP, as opposed to 63.9% for a PCFG; this score is more lenient than the exact match criterion, so this performance clearly represents a substantial decrease from the 96% figure in [Bod 93]. In a data set that was cleaned up by Bod, both methods report better results: 86.1% and 79.2% zero crossing brackets for DOP and PCFG respectively.)

**[Sekine and Grishman 95]**

[Sekine and Grishman 95] describe a parser that uses rules with a large amount of structural detail. Only two non-terminals — `S` and `NP` — are used. Other non-terminals appear as intermediate structure associated with a rule. For example, the method might induce a rule $\texttt{S} \rightarrow \texttt{NP VBX JJ CC VBX NP}$ with associated structure [S NP [VP [VP VBX [ADJP JJ] CC [VP VBX NP]]]], and an associated probability $P(\text{rule,structure}|S)$. A part-of-speech tagging model is also integrated with the rule probabilities. The method is tested on the Penn WSJ treebank, with 33.9% of sentences receiving an analysis with no crossing brackets.

### 4.4.4 PCFG Parsing Algorithms for Different Evaluation Criteria

**[Goodman 96b]**

[Goodman 96b] describes different parsing algorithms for PCFGs that maximize the expected accuracy on different parsing metrics. Search for the highest probability parse under a PCFG model maximizes the probability of finding the correct parse for a sentence, but does not necessarily maximize the expected accuracy on other evaluation metrics such as crossing brackets, or labeled recall of constituents. Evaluation on the ATIS domain, with a PCFG induced directly from a treebank, shows that the different algorithms each give the best results for the particular evaluation metric they are tuned for.

### 4.4.5 The Effect of Annotation Style on PCFG Accuracy

**[Johnson 97]**

[Johnson 97] considers the effect of alternative tree representations on PCFG accuracy: in particular, how different representations for PP adjunction effect a PCFGs ability to distinguish between different parses. He gives results for a PCFG that are similar to those in [Charniak 97] (69.6%/73.5% recall/precision in recovering labeled constituents). He also shows that adding the parent of each non-terminal as conditioning information — that is, replacing $P(\alpha \rightarrow \beta|\alpha)$ with $P(\alpha \rightarrow \beta|\alpha, Parent(\alpha))$ where $Parent(\alpha)$ is the non-terminal dominating $\alpha$ — leads to an improvement to 79.3%/80.1% recall/precision. This modification was originally described in [Charniak and Carroll 94], which considered unsupervised training of a PCFG with $P(\alpha \rightarrow \beta|\alpha, Parent(\alpha))$ parameters, but did not give results for supervised training.

### 4.4.6 Representation of PCFG Rules as Markov Processes

**[Seneff 92]**

[Seneff 92] describes TINA, a natural language system for spoken language systems. The paper describes the implementation of a probabilistic parser, and its integration into a speech system. The work is of relevance to the work in this thesis, because the probability of a rule $P(X \rightarrow Y_1\ Y_2\ ...Y_n|X)$ is decomposed using a bigram Markov process, as $P(Y_1|X, START)P(STOP|X, Y_n)\prod_{i=2...n} P(Y_n|X, Y_{n-1})$. The models in chapter 7 also use Markov processes over non-terminal sequences to parameterize rule probabilities.

## 4.5 Rule-Based Learning Methods

**[Brill 93]**

[Brill 93] described the application of transformation based learning (TBL) to parsing. TBL has also been applied to POS tagging [Brill 95] and prepositional phrase attachment disambiguation [Brill and Resnik 94]. The method learns a set of rules, which are applied in sequence to give a parse for a sentence. The starting state is to have a completely right-branching, binary-branching tree for a sentence: each of the transformational rules can then change a local piece of structure, for example by transforming a bracketing [A [B C]] to [[A B] C]. A transformation can be triggered by a conditioning context, which is either a single tag, or a pair of tags. The method is sensitive to POS tags only, as the conditioning features do not include words. Training the model is achieved through a greedy search, at each iteration adding the rule that gives the greatest decrease in error rate to the list of rules. Results are given on the WSJ and ATIS corpora. On ATIS the method shows a slight improvement over [Pereira and Schabes 92], in spite of training on only 150 sentences ([Pereira and Schabes 92] trained on 700). The most comparable result to the results in this thesis are the percentage of sentences in WSJ of length 2-25 words with 0 crossing brackets, 29.2%. This result was with only 250 sentences of training data from TBI.

**[Hermjakob and Mooney 97]**

[Hermjakob and Mooney 97] describe a machine-learning method that induces the rules required by a deterministic shift-reduce parser. The model is history-based, in that a parse tree is represented as the sequence of decisions made by the parser. Recovery of a parse tree is then considered to be a series of classification problems: given the previously built structure, what should the next move by the shift-reduce parser be? The model uses a combination of ID3 decision trees (with some modifications), and decision lists, as the learning algorithm. Conditioning features include the previously built structure, together with a knowledge base that contains semantic information about the words in the lexicon, as well as subcategorization information. Decoding is deterministic, in that the decision trees return a single decision at each point of ambiguity, rather than returning a distribution over possibilities which could then be used in a probabilistic search.

Results for the method are given on Wall Street Journal text, training from 256 sentences. The Penn WSJ treebank is not used as training or test material, instead the authors construct their own corpus. Labeled Precision/Recall results of 89.8/89.6% accuracy are given; 1.02 crossings brackets per sentence are observed, with 56.3% of sentences having 0 crossing brackets. When comparing these figures to work on parsing the Penn WSJ treebank, a number of factors should be taken into account: 1) the test and training material is from a different set of sentences; 2) the domain is restricted, in that only sentences containing words from a vocabulary of the most frequent 3000 words in WSJ are included in the training/test data; 3) The annotation style is quite different from that of the Penn treebank (looking at figure 2 of [Hermjakob and Mooney 97], in some cases the tree is "flatter" than the treebank, in other cases it is more detailed, and the non-terminal labels are quite different). This difference in annotation style may substantially impact evaluation scores. In particular, the labeled precision/recall figures may be problematic, as the difference in annotation styles may lead to a quite different definition of what a constituent is[3]. Thus the crossing brackets figures may be the only point of reasonable comparison.

[3]As an example of an extreme case, if the POS tag or something similar for each word is included as a constituent, then approximately half the constituents will be recovered with over 98% accuracy; the presence or absence of POS tags as constituents will effect the labeled precision/recall figures drastically.

## 4.6 Ranking Parse Trees through Scores Associated with Semantic Tuples

### [Sekine et al 92]

[Sekine et al 92] describe a method that ranks parses by associating scores with `[head-word, syntactic relation, argument]` tuples. A parse tree is represented as a set of such tuples; each tuple has an associated score; the score for each candidate parse tree is calculated as a product of its tuple scores. The `syntactic relation` field can be either a direct syntactic relation, such as *subject* or *object*, or the identity of a preposition, such as *by* or *with*. Scores for tuples are calculated in an unsupervised manner: the method assumes an existing parser that will provide all analyses for a particular sentence, and an iterative algorithm is used to calculate the scores for tuples. While the method is heuristic — the tuple scores do not apparently have a direct probabilistic interpretation, and the model does not define a joint or conditional probability distribution over sentence-tree pairs — the method appears to be similar to the EM algorithm [Dempster, Laird and Rubin 77]. Evaluation was done on compound-nominal ambiguities alone, with over 70% accuracy in disambiguating these structures. In conclusion, while the method is heuristic in nature, and evaluation is fairly limited, the work is of great interest in a couple of respects. First, it proposes representing a parse tree as a set of `[head word, syntactic relation, argument]` tuples. Second, it uses an unsupervised training method.

### [Jones and Eisner 92a, Jones and Eisner 92b]

[Jones and Eisner 92a, Jones and Eisner 92b] describe a probabilistic parser applied to software testing documents. The probability of two constituents combining to form a new constituent in a bottom-up parse is calculated as the combination of two terms: a syntactic probability conditioned on the two non-terminals being joined; and a semantic probability conditioned on the two terms in the predicate-argument relation implied by the combination of the two constituents. Results showed that the parser recovered a parse for 77% of all test sentences, with the highest ranked parse being correct 90% of the time.

**[Alshawi and Carter 94]**

[Alshawi and Carter 94] describe a supervised technique for ranking parses in the ATIS domain. A parse tree is represented as: 1) a set of tuples that represent grammatical relations between words, in a similar way to [Sekine et al 92]; 2) the count of a number of other features appearing in the parse, for example the number of adjuncts, elliptical expressions, balanced conjunctions, and so on. An existing parser is used to generate all possible analyses for each sentence, and the correct analysis is selected by hand to give examples for supervised learning, and to provide a test set. The paper considers a number of ways of associating a score with each semantic tuple — the best method selects the correct parse 89.7% of the time. The second point in the paper is the combination of the different forms of evidence (semantic collocations, and counts of other features of the parse) using optimization methods based on hill climbing. Combining all features gives an accuracy of 94.3%.

## 4.7 Probabilistic Versions of Lexicalized Grammar Formalisms

A number of papers describe work on probabilistic versions of lexicalized syntactic formalisms. This work is important, as a natural result of the formalisms is to have parameters associated with lexical dependencies in parse trees. The work in this section is more theoretical, generally specifying a probability model, sometimes with a derivation of an inside-outside style training algorithm, but generally with no evaluation on a parsing task.

### 4.7.1 Stochastic Tree Adjoining Grammars

**[Resnik 92]**

[Resnik 92] describes a probabilistic model of Tree Adjoining Grammar (TAG). The model includes three types of parameters: $P_I(\alpha)$, the probability of tree $\alpha$ being the first tree in a derivation; $P_S(\beta|\alpha,\eta)$, the probability of substituting tree $\beta$ into tree $\alpha$ at node $\eta$; and $P_A(\beta|\alpha,\eta)$, the probability of adjoining tree $\beta$ into tree $\alpha$ at node $\eta$. Given that a tree in TAG contains at least one lexical item, the substitution and adjunction parameters are associated with pairs of words, their respective trees, and the grammatical relation that is involved. Hence the method is sensitive to many of the features (with the exception of right-branching preferences) that we proposed in chapter 3. A drawback of the method is that the inclusion of both trees as well as the grammatical relation will probably lead to a very large number of parameters. An easy way to solve this problem might be to: first, decompose the generation of the tree $\beta$ into a number of smaller steps; second, define smoothed estimates by defining a back-off hierarchy in the conditioning context for the parameters.

#### [Schabes 92]

[Schabes 92] describes a probabilistic model of TAG that is very similar to the model in [Resnik 92]. This paper additionally derives a method for unsupervised learning through the EM algorithm [Dempster, Laird and Rubin 77]: EM was originally derived for context-free grammars in [Baker 79], the algorithm in [Schabes 92] is an extension of this algorithm of this to Stochastic TAGs. Empirical results are given for unsupervised learning of the language $\{a^n b^n | n \geq 0\}$. The method is shown to converge quickly and correctly.

#### [Schabes and Waters 93]

[Schabes and Waters 93] describe what they call "Stochastic Context-Free Grammar". The formalism is equivalent in power to a context-free grammar, but its representation is the same as lexicalized TAG, with restrictions on the form of trees to ensure that the formalism is only context-free in power. The resulting formalism retains the parameters of [Resnik 92] and [Schabes 92], while allowing both parsing and EM training to be performed in $O(n^3)$ (as opposed to $O(n^6)$) time.

#### Formal Results for Stochastic TAG

Additional papers give further results and algorithms for Stochastic TAGs. [Sarkar 98] extends the conditions for the consistency of PCFGs given in [Booth and Thompson 73] to STAGs. [Nederhof et al 98] describes an algorithm for efficiently computing *prefix probabilities* for a STAG: efficient computation of these probabilities would allow incorporation of an STAG into a strictly incremental speech recogniser. [Nederhof et al 1998b] describe the computation of prefix probabilities in Stochastic Linear Indexed Grammars, a class of grammars that includes STAGs.

**[Joshi and Srinivas 94]**

[Joshi and Srinivas 94] introduced *Supertagging* as a method of stochastic parsing of TAGs; [Srinivas 97] gives more recent models and results. In supertagging, the recovery of the TAG tree associated with each word in the sentence is achieved through a trigram tagging model, as described in [Church 88]. The TAG tree for a word in a sentence can be regarded as a highly refined POS tag, with information about subcategorization for verbs, whether a noun is a pre-modifier or head, and so on. This increased detail of information also leads to increased ambiguity, words on average having 47 possible elementary trees, with the baseline method of simply choosing the most likely tree for each word giving 77% accuracy on the WSJ corpus (for simple POS tagging, this baseline is over 90%). A trigram tagger recovered supertags with 92% accuracy when trained from 1,000,000 words of the WSJ corpus (result from [Srinivas 97]). As a second step to parsing, a *Lightweight Dependency Analyser* (LDA) is used to link the elementary trees discovered by the supertagger to form a partial (or possibly complete) parse for the sentence.

### 4.7.2 Link Grammars

**[Lafferty et al. 92]**

[Lafferty et al. 92] describe a probabilistic version of Link grammar. Link grammar, introduced in [Sleator and Temperley 91], is similar to both categorial grammars [Wood 93] and lexicalized TAGS in many ways. The lexicon specifies a left and right *disjunct* for each word in a language, a disjunct being an ordered list of left or right complements/adjuncts that are required by the word (note that adjunction is not handled separately by the formalism, so a disjunct must list possible adjuncts as well as complements — this is a substantial difference from categorial grammars or TAG). [Lafferty et al. 92] describe a probabilistic model based on the top-down parsing algorithm in [Sleator and Temperley 91]: the model is generative, specifying a distribution over the space of parse/sentence pairs. They give an algorithm for unsupervised training of the model (similar to the inside-outside algorithm).

No evaluation of the model on a parsing task is given.

By taking a closer look at the parameterization of the model we can see that the method has potential for capturing many of the representational properties in chapter 3, but that there may be problems with the particular modeling choices made in the paper. The main parameters in the model are of the form $P(W, d, O|L, R, l, r)$, where $W$ is a word being generated; $L$ and $R$ are two potential words that could generate $W$ as a dependent; $O$ is an orientation specifying which of the two words $W$ could attach to (attaching to both words is a possibility); $d$ is the disjunct for $W$; and $l$, $r$ are the potential connection sites for $L$ and $R$ respectively. [Lafferty et al. 92] decompose this probability into three terms — $P(W|L, R, l, r)P(d|W, l, r)P(O|d, l, r)$ — implicitly choosing to generate the word $W$ first, followed by a choice of $W$'s disjunct, followed finally by a choice of which of $L$ and $R$ to attach to. We see an immediate problem with this: $W$ is generated before it is known which of $L$ or $R$ it will attach to. Thus the trigram probability cannot be reduced to a bigram involving the word that generates $W$, or cannot be smoothed by back-off to this bigram[4]. A more natural decomposition might be $P(O|d, l, r)P(W|L, R, l, r, O)P(d|W, l, r)$. In this case the parameter $P(W|L, R, l, r, O)$ can be backed off to $P(W|L)$ or $P(W|R)$ depending on the value of $O$, so the more predictive of the two bigrams can be used directly.

A second problem concerns the disjunct probability $P(d|W, l, r)$. Disjuncts in link grammar must list adjuncts as well as complements that will attach to a head-word, as well as the syntactic role that the word appears in. The original grammar formalism in [Sleator and Temperley 91] describes methods for concisely representing disjuncts, through the use of the OR connective for the representation of alternatives, and through the specification of optional or iterated members of a disjunct. This means that a large number (possibly an infinite number, given the iteration operator) of disjuncts can be specified in a single formula. Unfortunately, the parameter $P(d|W, l, r)$ must refer to prediction of a disjunct $d$ that fully specifies the complements and adjuncts required by $W$: $d$ cannot be a formula that represents a set of possible disjuncts. This will make the number of parameters of the form $P(d|W, l, r)$ very large, and will prevent the model from learning

[4] This is particularly problematic given that the "third", less relevant, word involved in the trigram may stand in a more-or-less arbitrary relationship to $W$ in the parse tree, see [Eisner 96] for further explanation.

generalizations (such that a particular verb is transitive, in spite of taking a varying number of adverbial modifiers; or that a particular noun tends to take particular modifiers, whether it is in subject, object, or some other syntactic position).

### 4.7.3 Lexicalized PCFGs

**[de Marcken 95]**

[de Marcken 95] describes experiments and analysis of unsupervised learning of PCFGs. He makes two very important points. First, he argues for a lexicalized formalism, where each non-terminal $Z$ has an associated head-word $z$, and the probability of a rule $Z'(z) \rightarrow Z(z)Y(y)$ is decomposed as the product of a rule and dependency probability, $P(Z' \rightarrow Z \ \ Y|Z')P(z|y, Y, Z)$. Thus the method includes dependency probabilities, and is similar to the work described in [Charniak 97], [Goodman 97], and this thesis (although modification of $P(Z' \rightarrow Z \ \ Y|Z')$ to $P(Z' \rightarrow Z \ \ Y|Z', z)$ would almost certainly lead to an improved model). He shows that the global maximum of the likelihood function for an artificial example is linguistically plausible with a lexicalized grammar, while with a simple PCFG the global maximum is not linguistically plausible. Second, he gives analysis on an example problem of an artificial corpus. This shows that even in a case where the global maximum of the likelihood function is a linguistically plausible grammar, the inside-outside algorithm is very likely to hill-climb to a local maximum that is *not* linguistically plausible.

### 4.7.4 Head Automata

**[Alshawi 96]**

[Alshawi 96] describes lexicalized head automata, a formalism for both parsing and machine translation. The formalism represents a parse tree through head-modifier relationships: each head has a sequence of left and right modifier words $\langle w_1 ... w_k \rangle, \langle w_{k+1} ... w_n \rangle$, with associated relations $\langle r_1 ... r_k \rangle, \langle r_{k+1} ... r_n \rangle$. The paper describes a probability model that is generative, defining a probability distribution over all possible parse trees. Algorithms for both parsing and generation are given; the paper also discusses how the models can be used for machine translation, and discusses the use of weights on transitions that are not probabilistic.

[Alshawi 96] describes five parameter types: $P(\downarrow, w'|w, r)$, the probability of seeing the word $w'$ as an $r$-dependent of word $w$; $P(m, q|r, \downarrow, w)$, the probability of starting in state $q$ of machine $m$, given that $w$ has just been generated as an $r$ dependent of some other word; $P(\leftarrow, q_i, r|q_{i-1}, m)$ and $P(\rightarrow, q_i, r|q_{i-1}, m)$, the probabilities of choosing to generate an $r$-dependent to the left/right of the head respectively, and to move to state $q_i$, given that the model is in state $q_{i-1}$ of machine $m$; $P(\square|q, m)$, the probability of stopping, given that the machine is in state $q$ of machine $m$. $P(\downarrow, w'|w, r)$ is similar to the parameter types $P_{L1}/P_{R1}$ in chapter 7 (see section 7.6.1 for a description of these parameters). $P(\leftarrow, q_i, r|q_{i-1}, m)$ and $P(\rightarrow, q_i, r|q_{i-1}, m)$ could be made to correspond to the $P_{L2}/P_{R2}$ parameters, if the pair $(q, m)$ is used to encode the conditioning variables such as non-terminals, subcategorization frames, the distance measure, and the head word. The head-projection and subcategorization probabilities could be encoded in $P(m, q|r, \downarrow, w)$ — in this case, $P(m, q|r, \downarrow, w)$ is a probabilistic choice of the entire X-bar structure, including subcategorization frames, associated with $w$.

### 4.7.5 Stochastic Attribute-Value Grammars

**[Brew 95]**

[Brew 95] describes a stochastic formulation of Head-Driven Phrase Structure Grammar (HPSG) [Pollard and Sag 94]. The model assigns probabilities to type-hierarchies, and thereby to HPSG's representation of syntactic structure. The paper concentrates on defining parameters over these feature structures, rather than specifying the precise nature of the parameters: it is not clear that a model could, or would, include dependency parameters, for example. The model is essentially specified by drawing a parallel between the hierarchical structures and a context-free grammar, then using a PCFG. A problem is noted with re-entrancy, where two values in a feature structure may be constrained to take the same value, but where the PCFG model loses probability mass to structures with different values of the feature.

**[Abney 97]**

[Abney 97] looks at assigning probabilities to attribute-value (AV) grammars. He notes that the method in [Brew 95] leads to a model that does not sum to 1 when summed over all well-formed structures, due to the re-entrancy problem. [Abney 97] also notes that, although the model can be normalized so that is sums to 1, in this case the parameter estimates defined by [Brew 95] are no longer justifiable as maximum-likelihood estimates. [Abney 97] describes the use of maximum-entropy estimation methods to define a model over AV grammars. The estimation techniques require an iterative algorithm with Monte-Carlo sampling: this may be computationally expensive. As [Abney 97] points out in his conclusion, the estimation method may also be of interest in context-free formalisms, as it allows parameters to be defined corresponding to arbitrary pieces of sub-structure within parse trees.

## 4.8 Previous Work on Parsing the Penn WSJ Treebank

The work described in this section is of very direct relevance to the work in this thesis. In chapter 8, we give a much more detailed analysis and comparison of the methods.

### 4.8.1 Formalisms Including Dependency Probabilities

**[Eisner 96]**

[Eisner 96] describes three models for statistical parsing of dependency formalisms; in addition [Eisner 96b] gives a fourth model, a more detailed description of the models, and more up-to-date results. The models vary from conditional probability models, to a generative model that is similar to model 1 in chapter 7 of this thesis. The best results for dependency accuracy on the test set (in [Eisner 96b]) are 92.6% accuracy. The model in chapter 6 was trained and tested on the same data, with an identical result (with the modest caveat that the chapter 6 parser used machine generated tags for this test, [Eisner 96b] used hand labeled tags). A more detailed comparison of Eisner's models to the models of this work is given in section 8.4.3.

**[Charniak 97]**

[Charniak 97] describes a probability model for a lexicalized PCFG. The probability of a lexicalized rule is decomposed into the product of two terms: 1) a probability that predicts the non-lexicalized part of a rule, conditioned on the parent non-terminal and its head-word; 2) a probability of generating the lexical head of each modifier in the rule, giving dependency probabilities. Several other refinements are given: conditioning on the non-terminal above the parent when predicting a rule; the use of automatically derived word-classes to smooth probabilities; the use of additional unsupervised training of parameter values. Results on the Penn WSJ treebank of 86.7/86.6% recall/precision are obtained. A more detailed comparison of Charniak's model to the models of this work is given in section 8.4.1.

**[Goodman 97]**

[Goodman 97] describes the use of probabilistic feature grammars. Each non-terminal in the grammar is represented as a set of feature-value pairs; the probability $P(X \rightarrow Y \;\; Z|X)$ of a rule $X \rightarrow Y \;\; Z$ is decomposed as incremental prediction of the feature values of $Y$ and $Z$. The formalism assumes binary branching rules (without loss of generality: a one-to-one mapping from n-ary rules to binary-branching rules is given). Experimental results are given on the Penn WSJ treebank with a non-terminal representation that includes the non-terminal label, head-word, head POS, distance features, and additional context in terms of modifier non-terminals generated at earlier stages in the derivation. Results on the Penn WSJ treebank of 84.8/85.3% recall/precision are given. A more detailed comparison of Goodman's model to the models of this work is given in section 8.4.4.

### 4.8.2 History-Based Models

**[Black et al. 92b]**

[Black et al. 92b] describe a *history-based* model for parsing, with results on a computer manuals domain. This work can be viewed as a progression of the work in [Black et al. 92a]. A hand-crafted grammar is again used in combination with a treebank; the paper describes a method for obtaining a training set of parses in the grammar's formalism using the treebank and the grammar. A history-based generative model is then trained with this converted treebank: the parse tree is modeled as the sequence of decisions in a top-down, left-most derivation of the tree. Each decision corresponds to the choice of a rule expansion, followed by selection of non-terminal features on each child of the rule.

A major change from [Black et al. 92a] is in the representation of non-terminals: each non-terminal has a syntactic category (e.g. NP), a semantic category (e.g. *Data*), and two head-words. Head-words are represented as bit-strings, which are derived automatically using the method in [Brown et al. 1992]. All probabilities are conditioned on features of the parent and grandparent non-terminal; the rule probability is estimated using a decision tree, other parameters are estimated using n-gram deleted interpolation methods. While the exact back-off order is not specified, the model clearly has the potential to include parameters similar to the model described in [Charniak 97]: i.e. the probability of expanding a rule given its lexical head, and head-modifier dependency relationships. Results show a 36.8% relative error reduction from the PCFG model in [Black et al. 92a].

**[Jelinek et al. 94]**

[Jelinek et al. 94] describe work on the computer manuals domain of [Black et al. 92a, Black et al. 92b]. In contrast to [Black et al. 92a, Black et al. 92b] the method does not require a hand-crafted grammar. Instead, a history-based conditional model is trained directly from a treebank. A syntactic tree is represented as the sequence of decisions in a bottom-up parse of the tree. The probability of each decision is based on surrounding context, and is estimated using a decision tree. There are three types of parameters: a model for POS tagging; a model for extending a node, i.e. building a child-parent arc in either a left, right or unary fashion; and a model for assigning non-terminal labels. Each parameter type is estimated using decision trees. Each node in the tree is represented as its non-terminal label, head-word, and POS tag for the head-word. Words are represented as bit-strings derived using the clustering method in [Brown et al. 1992]. An additional refinement of the model is to assume a distribution over possible bottom-up derivations of the tree, the parameters specifying this distribution being estimated using EM training.[5] Results show a 29% error reduction over performance of the model in [Black et al. 92a], in spite of the move to direct learning from a treebank as opposed to the use of a hand-crafted grammar.

**[Magerman 95]**

[Magerman 95] describes the SPATTER parser, a progression of the work in [Jelinek et al. 94]. SPATTER was applied to the computer manuals domain, and to the Penn WSJ treebank. This paper is crucial to the work in this thesis, as it describes the first result on the WSJ treebank that is significantly higher than the result for PCFGs, using a parser that conditions strongly on lexical information. Running the parser on section 23 of the treebank[6] gave results of 84.0%/84.3% recall/precision at recovering labeled constituents.

**[Ratnaparkhi 97]**

[Ratnaparkhi 97] describes a model that is also a history-based conditional model, associating probabilities with decisions made by a parser. The work differs from [Jelinek et al. 94, Magerman 95] in several significant ways: maximum-entropy models are used for estimation, rather than decision trees; words are represented directly, rather than as bit-strings (perhaps because of advantages of the maximum-entropy estimation technique); the derivation order is quite different, with separate stages for POS tagging and chunking; as a consequence of the different derivation order, the conditioning features for each decision are different; the search method uses a beam search which runs in expected linear time, a rather simpler strategy than that described in [Magerman 95]; the model does not use a hidden derivation model, instead there is a one-to-one mapping between parse trees and decision sequences. Results of 87.5/86.3% precision/recall on the WSJ treebank are obtained, a significant improvement over [Magerman 95]. [Ratnaparkhi 97] also gives results for recovery of the n-best parse trees: an oracle that could make an optimal choice from

[5] This leads to a considerably more complicated model for both training and decoding in comparison to fixing a simple single derivation model; and results in [Magerman 95, Ratnaparkhi 97], history-based models without hidden derivations, suggest that this additional complexity may not help accuracy.

[6] Thanks to David Magerman for allowing us to run these experiments.

the top 20 parse trees recovered in this way would score around 93% precision and recall, suggesting that the investigation of reranking schemes for n-best parses might be a fruitful line of research.

A more detailed comparison of the models of [Ratnaparkhi 97, Magerman 95] to the models of this work is given in section 8.4.2.

**[Chelba and Jelinek 98]**

[Chelba and Jelinek 98] describe a history based model that parses strictly left-to-right, and assigns a joint probability $P(T, S)$ to Tree-Sentence pairs. The main intention of the work is to build a language model for speech recognition. The model operates on binary-branching trees; the Penn WSJ treebank is converted to binary-branching trees centered around the heads of rules. The model uses both a *parsing* and a *prediction* model. Structure is built up incrementally from left to right. When control is with the parsing model there are three possible moves: either to join the right-most adjacent two trees with the left/right tree providing the head-word of the new constituent, or to choose not to join the structures, thereby passing control to the prediction model. The prediction model generates a word-tag pair with some probability, conditioned on the two previous head-words exposed in the parse. Thus the model provides trigram probabilities conditioned on head-words that may fall outside the three-word window of a usual trigram model. EM training is used to re-estimate the parameters of the model. The model achieves an 11% reduction in perplexity over a trigram model trained on the same data.

# Chapter 5

# Prepositional Phrase Attachment through a Backed-Off Model

The major part of this chapter is joint work with James Brooks, having been originally published as [Collins and Brooks 95]. It remains largely unchanged and when it differs we make a note at that point in the text. In the final discussion section we give more recent analysis, and describe some additional experiments that have implications for the parsing work in this thesis.

## 5.1 Introduction

Prepositional phrase attachment is a common cause of structural ambiguity in natural language. For example take the following sentence:

Pierre Vinken, 61 years old, *joined the board as a nonexecutive director.*

The PP 'as a nonexecutive director' can either attach to the NP 'the board' or to the VP 'joined', giving two alternative structures. (In this case the VP attachment is correct):

NP-attach: (joined ((the board) (as a nonexecutive director)))
VP-attach: ((joined (the board)) (as a nonexecutive director))

Work by [Ratnaparkhi et al. 94] and[Brill and Resnik 94] has considered corpus-based approaches to this problem, using a set of examples to train a model which is then used to make attachment decisions on test data. Both papers describe methods which look at the four head words involved in the attachment: the VP head, the first NP head, the preposition and the second NP head (in this case *joined*, *board*, *as* and *director* respectively).

This paper proposes a new statistical method for PP-attachment disambiguation based on the four head words.

## 5.2 Background

### 5.2.1 Training and Test Data

The training and test data were supplied by IBM, being identical to the data that was used in [Ratnaparkhi et al. 94]. Examples of verb phrases containing a (v np pp) sequence had been taken from the Wall Street Journal Treebank [Marcus et al. 93]. For each such VP the head verb, first head noun, preposition and second head noun were extracted, along with the attachment decision (1 for noun attachment, 0 for verb). For example, the verb phrase:

((joined (the board)) (as a nonexecutive director))

would give the quintuple:

0 joined board as director

The elements of this quintuple will from here on be referred to as the random variables $A$, $V$, $N1$, $P$, and $N2$. In the above verb phrase $A = 0$, $V = joined$, $N1 = board$, $P = as$, and $N2 = director$.

The data consisted of training and test files of 20801 and 3097 quintuples respectively. In addition, a development set of 4039 quintuples was also supplied. This set was used during development of the attachment algorithm, ensuring that there was no implicit training of the method on the test set itself.

### 5.2.2 Outline of the Problem

A PP-attachment algorithm must take each quadruple ($V = v$, $N1 = n1$, $P = p$, $N2 = n2$) in test data and decide whether the attachment variable $A = 0$ or 1. The accuracy of the algorithm is then the percentage of attachments it gets 'correct' on test data, using the $A$ values taken from the treebank as the reference set.

The probability of the attachment variable $A$ being 1 or 0 (signifying noun or verb attachment respectively) is conditional on the values of the words in the quadruple. In general, a probabilistic algorithm will make an estimate, $\hat{p}$, of this probability:

$$\hat{p}(A = 1|V = v, N1 = n1, P = p, N2 = n2) \tag{5.1}$$

For brevity this estimate will be referred to from here on as:

$$\hat{p}(1|v, n1, p, n2) \tag{5.2}$$

The decision can then be made using the test:

$$\hat{p}(1|v, n1, p, n2) \geq 0.5 \tag{5.3}$$

If this is true the attachment is made to the noun, if not then it is made to the verb.

### 5.2.3 Lower and Upper Bounds on Performance

When evaluating an algorithm it is useful to have an idea of the lower and upper bounds on its performance. Some key results are summarised in the table below. All results in this section are on the IBM training and test data, with the exception of the two 'average human' results.

| Method | Percentage Accuracy |
|---|---|
| Always noun attachment | 59.0 |
| Most likely for each preposition | 72.2 |
| Average Human (4 head words only) | 88.2 |
| Average Human (whole sentence) | 93.2 |

'Always noun attachment' means attach to the noun regardless of (v,n1,p,n2). 'Most likely for each preposition' means use the attachment seen most often in training data for the preposition seen in the test quadruple. The human performance results are taken from [Ratnaparkhi et al. 94], and are the average performance of 3 treebanking experts on a set of 300 randomly selected test events from the WSJ corpus, first looking at the four head words alone, then using the whole sentence.

A reasonable lower bound seems to be 72.2% as scored by the 'Most likely for each preposition' method. An approximate upper bound is 88.2%: it seems unreasonable to expect an algorithm to perform much better than a human.

## 5.3 Estimation based on Training Data Counts

### 5.3.1 Notation

We will use the symbol $f$ to denote the number of times a particular tuple is seen in training data. For example $f(1, is, revenue, from, research)$ is the number of times the quadruple $(is, revenue, from, research)$ is seen with a noun attachment. Counts of lower order tuples can also be made: for example $f(1, P = from)$ is the number of times $(P = from)$ is seen with noun attachment in training data, $f(V = is, N2 = research)$ is the number of times $(V = is, N2 = research)$ is seen with either attachment and any value of N1 and P.

### 5.3.2 Maximum-Likelihood (ML) Estimation

A maximum-likelihood method would use the training data to give the following estimation for the conditional probability:

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, n1, p, n2)}{f(v, n1, p, n2)} \tag{5.4}$$

Unfortunately sparse data problems make this estimate useless. A quadruple may appear in test data which has never been seen in training data. ie. $f(v, n1, p, n2) = 0$. The above estimate is undefined in this situation, which happens extremely frequently in a large vocabulary domain such as WSJ. (In this experiment about 95% of those quadruples appearing in test data had not been seen in training data.)

Even if $f(v, n1, p, n2) > 0$, it may still be very low, and this may make the above ML estimate inaccurate. Unsmoothed ML estimates based on low counts are notoriously bad in similar problems such as n-gram language modeling [Gale and Church 90]. However, later in this paper it is shown that estimates based on low counts are surprisingly useful in the PP-attachment problem.

### 5.3.3 Previous Work

[Hindle and Rooth 93] describe one of the first statistical approaches to the prepositional phrase attachment problem. Over 200,000 $(v, n1, p)$ triples were extracted from 13 million words of AP news stories. The attachment decisions for these triples were unknown, so an unsupervised training method was used (section 5.5.2 describes the algorithm in more detail). Two human judges annotated the attachment decision for 880 test examples, and the method performed at 80% accuracy on these cases. Note that it is difficult to compare this result to results on Wall Street Journal, as the two corpora may be quite different.

The Wall Street Journal Treebank [Marcus et al. 93] enabled both [Ratnaparkhi et al. 94] and [Brill and Resnik 94] to extract a large amount of supervised training material for the problem. Both of these methods consider the second noun, $n2$, as well as $v$, $n1$ and $p$, with the hope that this additional information will improve results.

[Brill and Resnik 94] use 12,000 training and 500 test examples. A greedy search is used to learn a sequence of 'transformations' which minimise the error rate on training data. A transformation is a rule which makes an attachment decision depending on up to 3 elements of the $(v, n1, p, n2)$ quadruple. (Typical examples would be 'If P=*of* then choose noun attachment' or 'If V=*buy* and P=*for* choose verb attachment'.) A further experiment incorporated word-class information from WordNet into the model, by allowing the transformations to look at classes as well as the words. (An example would be 'If N2 is in the *time* semantic class, choose verb attachment'.) The method gave 80.8% accuracy with words only, 81.8% with words and semantic classes, and they also report an accuracy of 75.8% for the metric of [Hindle and Rooth 93] on this data. (But note our later discussion in section 5.8.2.) Transformations (using words only) score 81.9%[1] on the IBM data used in this paper.

[Ratnaparkhi et al. 94] use the data described in section 5.2.1 of this paper: 20801 training and 3097 test examples from Wall Street Journal. They use a maximum entropy model which also considers subsets of the quadruple. Each sub-tuple predicts noun or verb attachment with a weight indicating its strength of prediction: the weights are trained to maximise the likelihood of training data. For example $(P = of)$ might have a strong

[1]Personal communication from Brill.

weight for noun attachment, while ($V = buy, P = for$) would have a strong weight for verb attachment. [Ratnaparkhi et al. 94] also allow the model to look at class information, this time the classes were learned automatically from a corpus. Results of 77.7% (words only) and 81.6% (words and classes) are reported. Crucially they ignore low-count events in training data by imposing a frequency cut-off which they describe as being "usually 3 to 5" in value.

## 5.4 The Backed-Off Estimate

[Katz 87] describes backed-off n-gram word models for speech recognition. There the task is to estimate the probability of the next word in a text given the (n-1) preceding words. The ML estimate of this probability would be:

$$\hat{p}(w_n|w_1, w_2....w_{n-1}) = \frac{f(w_1, w_2....w_n)}{f(w_1, w_2....w_{n-1})} \tag{5.5}$$

But again the denominator $f(w_1, w_2....w_{n-1})$ will frequently be zero, especially for large $n$. The backed-off estimate is a method of combating the sparse data problem. $\bar{f}(x)$ is defined as the *discounted* count for event $x$, where the Good-Turing method is used for discounting. The backed-off estimate is then defined recursively as follows[2]:

If $f(w_1, w_2....w_n) > c_1$

$$\hat{p}(w_n|w_1, w_2....w_{n-1}) = \frac{\bar{f}(w_1, w_2....w_n)}{f(w_1, w_2....w_{n-1})}$$

Else if $f(w_2, w_3....w_n) > c_2$

$$\hat{p}(w_n|w_1, w_2....w_{n-1}) = \alpha_1 \times \frac{\bar{f}(w_2, w_3....w_n)}{f(w_2, w_3....w_{n-1})}$$

Else if $f(w_3, w_4....w_n) > c_3$

$$\hat{p}(w_n|w_1, w_2....w_{n-1}) = \alpha_1 \times \alpha_2 \times \frac{\bar{f}(w_3, w_4....w_n)}{f(w_3, w_4....w_{n-1})}$$

Else backing-off continues in the same way.

[2][Collins and Brooks 95] gave a variation of the formulation in [Katz 87], for example stating the first constraint as $f(w_1, w_2....w_{n-1}) > c_1$ rather than $f(w_1, w_2....w_n) > c_1$

The idea here is to use estimates based on lower order n-grams if counts are not high enough to make an accurate estimate at the current level. The cut off frequencies ($c_1$, $c_2$....) are thresholds determining whether to back-off or not at each level: counts lower than $c_i$ at stage $i$ are deemed to be too low to give an accurate estimate, so in this case backing-off continues. ($\alpha_1$, $\alpha_2$,....) are normalisation constants which ensure that conditional probabilities sum to one.

The estimation of $\hat{p}(w_n|w_1, w_2....w_{n-1})$ is analogous to the estimation of $\hat{p}(1|v, n1, p, n2)$, and the above method can therefore also be applied to the PP-attachment problem. For example a simple method for estimation of $\hat{p}(1|v, n1, p, n2)$ would go from estimates of $\hat{p}(1|v, n1, p, n2)$ to $\hat{p}(1|v, n1, p)$ to $\hat{p}(1|v, n1)$ to $\hat{p}(1|v)$ to $\hat{p}(1)$. However a crucial difference between the two problems is that in the n-gram task the words $w_1$ to $w_n$ are sequential, giving a natural order in which backing off takes place: from $\hat{p}(w_n|w_1, w_2....w_{n-1})$ to $\hat{p}(w_n|w_2, w_3....w_{n-1})$ to $\hat{p}(w_n|w_3, w_4....w_{n-1})$ and so on. There is no such sequence in the PP-attachment problem, and because of this there are four possible triples when backing off from quadruples ($(v, n1, p)$, $(v, p, n2)$, $(n1, p, n2)$ and $(v, n1, n2)$) and six possible pairs when backing off from triples ($(v, p)$, $(n1, p)$, $(p, n2)$, $(v, n1)$, $(v, n2)$ and $(n1, n2)$).

A key observation in choosing between these tuples is that the preposition is particularly important to the attachment decision. For this reason only tuples which contained the preposition were used in backed off estimates: this reduces the problem to a choice between 3 triples and 3 pairs at each respective stage. Section 5.6.2 describes experiments which show that tuples containing the preposition are much better indicators of attachment.

The following method of combining the counts was found to work best in practice:

$$\hat{p}_{triple}(1|v, n1, p, n2) = \frac{f(1, v, n1, p) + f(1, v, p, n2) + f(1, n1, p, n2)}{f(v, n1, p) + f(v, p, n2) + f(n1, p, n2)} \tag{5.6}$$

and

$$\hat{p}_{pair}(1|v, n1, p, n2) = \frac{f(1, v, p) + f(1, n1, p) + f(1, p, n2)}{f(v, p) + f(n1, p) + f(p, n2)} \tag{5.7}$$

Note that this method effectively gives more weight to tuples with high overall counts. The method is equivalent to a weighted average of the three estimates, where the weight is proportional to the count on which the estimate is based. For example, 5.7 can be written as

$$\lambda_1 \frac{f(1,v,p)}{f(v,p)} + \lambda_2 \frac{f(1,n1,p)}{f(n1,p)} + \lambda_3 \frac{f(1,p,n2)}{f(p,n2)} \tag{5.8}$$

where $d = f(v,p) + f(n1,p) + f(p,n2)$, and

$$\lambda_1 = \frac{f(v,p)}{d} \quad \lambda_2 = \frac{f(n1,p)}{d} \quad \lambda_3 = \frac{f(p,n2)}{d} \tag{5.9}$$

Another obvious method of combination, a simple average[3], gives equal weight to the three tuples regardless of their total counts and does not perform as well.

The cut-off frequencies must then be chosen. A surprising difference from language modeling is that a cut-off frequency of 0 is found to be optimum at all stages. This effectively means however low a count is, still use it rather than backing off a level.

### 5.4.1 Description of the Algorithm

The algorithm is then as follows:

1. **If**[4] $f(v,n1,p,n2) > 0$

$$\hat{p}(1|v,n1,p,n2) = \frac{f(1,v,n1,p,n2)}{f(v,n1,p,n2)}$$

2. **Else if** $f(v,n1,p) + f(v,p,n2) + f(n1,p,n2) > 0$

$$\hat{p}(1|v,n1,p,n2) = \frac{f(1,v,n1,p) + f(1,v,p,n2) + f(1,n1,p,n2)}{f(v,n1,p) + f(v,p,n2) + f(n1,p,n2)}$$

3. **Else if** $f(v,p) + f(n1,p) + f(p,n2) > 0$

$$\hat{p}(1|v,n1,p,n2) = \frac{f(1,v,p) + f(1,n1,p) + f(1,p,n2)}{f(v,p) + f(n1,p) + f(p,n2)}$$

[3]eg. A simple average for triples would be defined as

$$\hat{p}_{triple}(1|v,n1,p,n2) = \frac{\frac{f(1,v,n1,p)}{f(v,n1,p)} + \frac{f(1,v,p,n2)}{f(v,p,n2)} + \frac{f(1,n1,p,n2)}{f(n1,p,n2)}}{3}$$

[4]At stages 1 and 2 backing off was also continued if $\hat{p}(1|v,n1,p,n2) = 0.5$. ie. the counts were 'neutral' with respect to attachment at this stage.

4. **Else if** $f(p) > 0$

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, p)}{f(p)}$$

5. **Else** $\hat{p}(1|v, n1, p, n2) = 1.0$ (default is noun attachment).

The decision is then:

If $\hat{p}(1|v, n1, p, n2) \geq 0.5$ choose noun attachment.

Otherwise choose verb attachment.

Note that this back-off method differs slightly from the method in [Katz 87], in that the denominator count rather than the numerator is conditioned upon by each **If** statement, and the numerator counts are not discounted.

## 5.5 Results

The figure below shows the results for the method on the 3097 test sentences, also giving the total count and accuracy at each of the backed-off stages.

| Stage | Total Number | Number Correct | Percent Correct |
|---|---|---|---|
| Quadruples | 148 | 134 | 90.5 |
| Triples | 764 | 688 | 90.1 |
| Doubles | 1965 | 1625 | 82.7 |
| Singles | 216 | 155 | 71.8 |
| Defaults | 4 | 4 | 100.0 |
| Totals | 3097 | 2606 | 84.1 |

### 5.5.1 Results with Morphological Analysis

In an effort to reduce sparse data problems the following processing was run over both test and training data:

- All 4-digit numbers were replaced with the string 'YEAR'.
- All other strings of numbers (including those which had commas or decimal points) were replaced with the token 'NUM'.

- The verb and preposition fields were converted entirely to lower case.
- In the n1 and n2 fields all words starting with a capital letter followed by one or more lower case letters were replaced with 'NAME'.
- All strings 'NAME-NAME' were then replaced by 'NAME'.
- All verbs were reduced to their morphological stem using the morphological analyser described in [Karp et al. 94].

These modifications are similar to those performed on the corpus used by [Brill and Resnik 94].

The result using this modified corpus was 84.5%, an improvement of 0.4% on the previous result.

| Stage | Total Number | Number Correct | Percent Correct |
|---|---|---|---|
| Quadruples | 242 | 224 | 92.6 |
| Triples | 977 | 858 | 87.8 |
| Doubles | 1739 | 1433 | 82.4 |
| Singles | 136 | 99 | 72.8 |
| Default | 3 | 3 | 100.0 |
| Totals | 3097 | 2617 | 84.5 |

### 5.5.2 Comparison with Other Work

Results from [Ratnaparkhi et al. 94], [Brill and Resnik 94] and the backed-off method are shown in the table below[5]. All results are for the IBM data. These figures should be taken in the context of the lower and upper bounds of 72.2%–88.2% proposed in section 5.2.3.

| Method | Percentage Accuracy |
|---|---|
| [Ratnaparkhi et al. 94] (words only) | 77.7 |
| [Ratnaparkhi et al. 94] (words and classes) | 81.6 |
| [Brill and Resnik 94] (words only) | 81.9 |
| Backed-off (no processing) | 84.1 |
| Backed-off (morphological processing) | 84.5 |

[5]Results for [Brill and Resnik 94] with words and classes were not available on the IBM data

## 5.6 A Closer Look at Backing-Off

### 5.6.1 Low Counts are Important

A possible criticism of the backed-off estimate is that it uses low count events without any smoothing, which has been shown to be a mistake in similar problems such as n-gram language models. In particular, quadruples and triples seen in test data will frequently be seen only once or twice in training data.

An experiment was made with all counts less than 5 being put to zero,[6] effectively making the algorithm ignore low count events. In [Ratnaparkhi et al. 94] a cut-off 'between 3 and 5' is used for all events. The training and test data were both the unprocessed, original data sets. The results were as follows:

| Stage | Total Number | Number Correct | Percent Correct |
|---|---|---|---|
| Quaduples | 39 | 38 | 97.4 |
| Triples | 263 | 243 | 92.4 |
| Doubles | 1849 | 1574 | 85.1 |
| Singles | 936 | 666 | 71.2 |
| Defaults | 10 | 5 | 50.0 |
| Totals | 3097 | 2526 | 81.6 |

The decrease in accuracy from 84.1% to 81.6% is clear evidence for the importance of low counts.

### 5.6.2 Tuples with Prepositions are Better

We have excluded tuples which do not contain a preposition from the model. This section gives results which justify this.

The table below gives accuracies for the sub-tuples at each stage of backing-off. The accuracy figure for a particular tuple is obtained by modifying the algorithm in section 5.4.1 to use only information from that tuple at the appropriate stage. For example for $(v, n1, n2)$, stage 2 would be modified to read

[6] Specifically: if for a subset $x$ of the quadruple $f(x) < 5$, then make $f(x) = f(1, x) = f(0, x) = 0$.

If $f(v, n1, n2) > 0$,

$$\hat{p}(1|v, n1, p, n2) = \frac{f(1, v, n1, n2)}{f(v, n1, n2)}$$

All other stages in the algorithm would be unchanged. The accuracy figure is then the percentage accuracy on the test cases where the $(v, n1, n2)$ counts were used. The development set with no morphological processing was used for these tests.

| Triples | | Doubles | | Singles | |
|---|---|---|---|---|---|
| Tuple | Accuracy | Tuple | Accuracy | Tuple | Accuracy |
| n1 p n2 | 90.9 | n1 p | 82.1 | p | 72.1 |
| v p n2 | 90.3 | v p | 80.1 | n1 | 55.7 |
| v n1 p | 88.2 | p n2 | 75.9 | v | 52.7 |
| v n1 n2 | 68.4 | n1 n2 | 65.4 | n2 | 47.4 |
| | | v n1 | 59.0 | | |
| | | v n2 | 53.4 | | |

At each stage there is a sharp difference in accuracy between tuples with and without a preposition. Moreover, if the 14 tuples in the above table were ranked by accuracy, the top 7 tuples would be the 7 tuples which contain a preposition.

## 5.7 Conclusions

The backed-off estimate scores appreciably better than other methods which have been tested on the Wall Street Journal corpus. The accuracy of 84.5% is close to the human performance figure of 88% using the 4 head words alone. A particularly surprising result is the significance of low count events in training data. The algorithm has the additional advantages of being conceptually simple, and computationally inexpensive to implement.

## 5.8 Further Discussion

### 5.8.1 Results with Limited Context

The parsers of chapters 6 and 7 use lexical information in making PP attachment decisions, but are effectively restricted to conditioning on the contexts $(N1, P)$ and $(V, P)$ and their subsets. To measure the impact on accuracy of this reduced context, we implemented the following modified algorithm:

1. **If** $f(v,p) + f(n1,p) > 0$

$$\hat{p}(1|v,n1,p,n2) = \frac{f(1,v,p) + f(1,n1,p)}{f(v,p) + f(n1,p)}$$

2. **Else if** $f(p) > 0$

$$\hat{p}(1|v,n1,p,n2) = \frac{f(1,p)}{f(p)}$$

3. **Else** $\hat{p}(1|v,n1,p,n2) = 1.0$ (default is noun attachment).

The resulting accuracy was 83.0%. This result is encouraging, in that the drop in accuracy from 84.1% is not too significant.

### 5.8.2 Results for Hindle and Rooth's Method

Although Hindle and Rooth's method was originally developed for unsupervised training, we can design a supervised model that has very similar parameters. Importantly, the parser in chapter 7 also uses very similar parameters in the case of PP attachment. For this reason, we would like to measure the accuracy of this Hindle and Rooth's model, and see how it compares to the PP attachment models in this chapter.

The model can be formulated as follows. It will define a joint probability, $P(A, V, N1, P, N2)$. The attachment decision is "Noun" if $P(A = noun, V, N1, P, N2) \geq P(A = verb, V, N1, P, N2)$, "Verb" otherwise. The joint probability is first re-written using the chain rule:

$$P(A, V, N1, P, N2) = P(V)P(N1)P(A|V, N1)P(P|A, V, N1)P(N2|P, A, V, N1) \quad (5.10)$$

Next, we make the following independence assumptions:

$$\begin{aligned} P(A|V, N1) &= P(A|N1) && (5.11) \\ P(P|A, V, N1) &= P(P|A, V) \text{ if A = Verb, } P(P|A, N1) \text{ otherwise} && (5.12) \\ P(N2|P, A, V, N1) &= P(N2) && (5.13) \end{aligned}$$

The terms $P(V)$, $P(N1)$ and $P(N2)$ do not involve $A$, and can therefore be discarded when making the attachment decision. The final decision is then "Noun" if $P(A = noun|N1)P(P|A = noun, N1) \geq P(A = verb|N1)P(P|A = verb, V)$. This model form is similar to Hindle and Rooth's. It has similar probabilities of seeing the preposition given the noun/verb, and $P(A = noun|N1)$ is the equivalent of Hindle and Rooth's stop probability.

The probabilities were smoothed by holding a count of 1 out for the backed off estimate, in each case backing off to ignore either $N1$ or $V$. The results for this model on the IBM test set were 81.3% accuracy.

# Chapter 6

# A Statistical Parser Based on Bigram Lexical Dependencies

The major part of this chapter was originally published as [Collins 96]. It remains largely unchanged and when it differs we make a note at that point in the text. In the final discussion section we give more recent analysis in light of the mathematical results and representation proposals of chapters 2 and 3.

## 6.1 Introduction

The previous chapter showed that lexical information is crucial for prepositional phrase attachment decisions, and it follows that lexical information is also useful when resolving other cases of ambiguity such as coordination, relative clause modification, noun-noun compounds, and so on. However, many early approaches to probabilistic parsing (see the previous work discussion in chapter 4) conditioned probabilities on non-terminal labels and part of speech tags alone. The SPATTER parser [Magerman 95, Jelinek et al. 94] was the first probabilistic parser to use lexical information for parsing wide-domain text, and recovered labeled constituents in Wall Street Journal text with above 84% accuracy – a dramatic improvement over the 70.6%/74.8% recall/precision figures for a non-lexicalized probabilistic context-free grammar (a result quoted in [Charniak 97]).

This paper describes a new parser which is much simpler than SPATTER, yet performs at least as well when trained and tested on the same Wall Street Journal data. The method uses lexical information directly by modeling head-modifier[1] relations between pairs of words. In this way it is similar to the previous work described in sections 4.7 and 4.8.1 of this work.

## 6.2 The Statistical Model

The aim of a parser is to take a tagged sentence as input (for example Figure 6.1(a)) and produce a phrase-structure tree as output (Figure 6.1(b)). A statistical approach to this problem consists of two components. First, the *statistical model* assigns a probability to every candidate parse tree for a sentence. Formally, given a sentence $S$ and a tree $T$, the model estimates the conditional probability $P(T|S)$. The most likely parse under the model is then:

$$T_{best} = \arg\max_{T} P(T|S) \tag{6.1}$$

Second, the *parser* is a method for finding $T_{best}$. This section describes the statistical model, while section 6.3 describes the parser.

The key to the statistical model is that any tree such as Figure 6.1(b) can be represented as a set of **baseNPs**[2] and a set of **dependencies** as in Figure 6.1(c). We call the set of baseNPs $B$, and the set of dependencies $D$; Figure 6.1(d) shows $B$ and $D$ for this example. For the purposes of our model, $T = (B, D)$, and:

$$P(T|S) = P(B, D|S) = P(B|S) \times P(D|S, B) \tag{6.2}$$

$S$ is the sentence with words tagged for part of speech. That is, $S = <(w_1, t_1), (w_2, t_2)...(w_n, t_n)>$. For POS tagging we use the maximum-entropy tagger described in [Ratnaparkhi 96]. The tagger performs at around 97% accuracy on Wall Street Journal Text, and is trained on the first 40,000 sentences of the Penn Treebank [Marcus et al. 93].

Given $S$ and $B$, the **reduced sentence** $\bar{S}$ is defined as the subsequence of $S$ which is formed by removing punctuation and reducing all baseNPs to their head-word alone. Thus

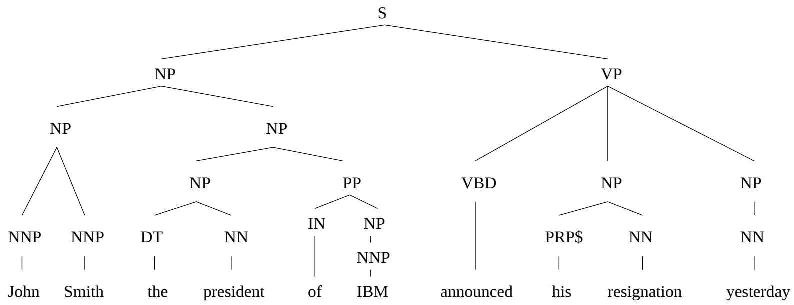
[1] By 'modifier' we mean the linguistic notion of either an argument or adjunct.

[2] A baseNP or 'minimal' NP is a non-recursive NP, i.e. none of its child constituents are NPs. The term was first used in [Ramshaw and Marcus 95].

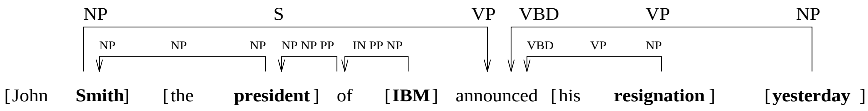
(a)

John/NNP Smith/NNP, the/DT president/NN of/IN IBM/NNP, announced/VBD his/PRP$ resignation/NN yesterday/NN .

(b)




(c)




(d)

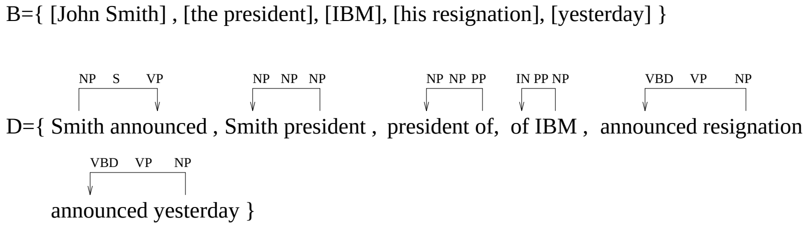



Figure 6.1: An overview of the representation used by the model. **(a)** The tagged sentence; **(b)** A candidate parse-tree (the correct one); **(c)** A dependency representation of (b). Square brackets enclose baseNPs (heads of baseNPs are marked in bold). Arrows show modifier $\rightarrow$ head dependencies. Section 6.2.1 describes how arrows are labeled with non-terminal triples from the parse-tree. Non-head words within baseNPs are excluded from the dependency structure; **(d)** $B$, the set of baseNPs, and $D$, the set of dependencies, are extracted from (c).

the reduced sentence is an array of word/tag pairs, $\bar{S} =< \langle \bar{w}_1, \bar{t}_1 \rangle, \langle \bar{w}_2, \bar{t}_2 \rangle ... \langle \bar{w}_m, \bar{t}_m \rangle >$, where $m \leq n$. For example, for Figure 6.1(a)

**Example 1.**

$$\begin{aligned} \bar{S} = < \quad & \langle Smith, NNP \rangle, \langle president, NN \rangle, \langle of, IN \rangle, \langle IBM, NNP \rangle, \\ & \langle announced, VBD \rangle, \langle resignation, NN \rangle, \langle yesterday, NN \rangle \quad > \end{aligned}$$

Sections 6.2.1 to 6.2.4 describe the dependency model. Section 6.2.5 then describes the baseNP model, which uses bigram tagging techniques similar to [Ramshaw and Marcus 95, Church 88].

### 6.2.1 The Mapping from Trees to Sets of Dependencies

The dependency model is limited to relationships between words in **reduced** sentences such as Example 1. The mapping from trees to dependency structures is central to the dependency model. It is defined in two steps:

**1.** For each constituent $P \rightarrow < C_1 ... C_n >$ in the parse tree a simple set of rules[3] identifies which of the children $C_i$ is the 'head-child' of $P$. For example, `NN` would be identified as the head-child of `NP` $\rightarrow$ `<DET JJ JJ NN>`, `VP` would be identified as the head-child of `S` $\rightarrow$ `<NP VP>`. Head-words propagate up through the tree, each parent receiving its head-word from its head-child. For example, in `S` $\rightarrow$ `<NP VP>`, `S` gets its head-word, *announced*, from its head-child, the `VP`.

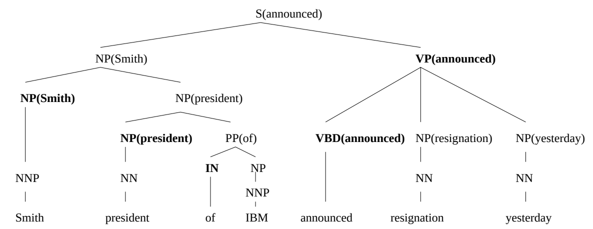



Figure 6.2: Parse tree for the reduced sentence in Example 1. The head-**child** of each constituent is shown in bold. The head-**word** for each constituent is shown in parentheses.

**2.** Head-modifier relationships are now extracted from the tree in Figure 6.2. Figure 6.3 illustrates how each constituent contributes a set of dependency relationships. `VBD`

[3]The rules are specified in Appendix A. These rules are also used to find the head-word of baseNPs, enabling the mapping from $S$ and $B$ to $\bar{S}$.

is identified as the head-child of `VP` $\rightarrow$ <`VBD NP NP`>. The head-words of the two NPs, *resignation* and *yesterday*, both modify the head-word of the `VBD`, *announced*. Dependencies are labeled by the modifier non-terminal, `NP` in both of these cases, the parent non-terminal, `VP`, and finally the head-child non-terminal, `VBD`. The triple of non-terminals at the start, middle and end of the arrow specify the nature of the dependency relationship – <`NP,S,VP`> represents a subject-verb dependency, <`PP,NP,NP`> denotes prepositional phrase modification of an `NP`, and so on[4].

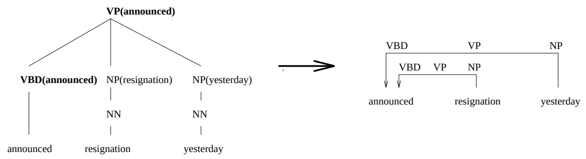



Figure 6.3: Each constituent with $n$ children (in this case $n = 3$) contributes $n - 1$ dependencies.

Each word in the reduced sentence, with the exception of the sentential head 'announced', modifies exactly one other word. We use the notation[5]

$$AF(j) = (h_j, R_j) \tag{6.3}$$

to state that the $j$th word in the reduced sentence is a modifier to the $h_j$th word, with relationship $R_j$[6]. $AF$ stands for 'arrow from'. $R_j$ is the triple of labels at the start, middle and end of the arrow. For example, $\bar{w}_1 = Smith$ in this sentence, and $\bar{w}_5 = announced$, so $AF(1) = (5,$<`NP,S,VP`>$)$.

$D$ is now defined as the m-tuple of dependencies: $D = \{(AF(1), AF(2)...AF(m)\}$. Note that the tree $T$ can be recovered from its associated set of dependencies, $D$. The linear order of the words in the sentence, combined with the relation labels on the arrows, describes how the dependencies combine to form constituency structure. For example, *resignation*, *announced* and *Smith* are all dependent on *announced* in figure 6.2, with relationships

[4] The triple can also be viewed as representing a semantic predicate-argument relationship, with the three elements being the type of the argument, result and functor respectively. This is particularly apparent in Categorial Grammar formalisms [Wood 93], which make an explicit link between dependencies and functional application.

[5] We preserve the notation from the original publication of this chapter, i.e., [Collins 96]. This differs from the notation in chapter 3 of this thesis, which would write the dependency as $\langle w_j \rightarrow w_{h_j}, R_j \rangle$.

[6] For the head-word of the entire sentence $h_j = 0$, with $R_j$=<Label of the root of the parse tree >. So in this case, $AF(5) = (0, < S >)$.

$\langle$`NP, VP, VBD`$\rangle$, $\langle$`NP, VP, VBD`$\rangle$, and $\langle$`NP, S, VP`$\rangle$ respectively. Only one structure over the order *Smith announced resignation yesterday* could have produced these dependencies. The model assumes that the dependencies are independent, so that:

$$P(D|S,B) = \prod_{j=1}^{m} P(AF(j)|S,B) \tag{6.4}$$

### 6.2.2 Calculating Dependency Probabilities

This section describes the way $P(AF(j)|S,B)$ is estimated. The same sentence is very unlikely to appear both in training and test data, so we need to back-off from the entire sentence context. We believe that lexical information is crucial to attachment decisions, so it is natural to condition on the words and tags. Let $\mathcal{V}$ be the vocabulary of all words seen in training data, $\mathcal{T}$ be the set of all part-of-speech tags, and $\mathcal{TRAIN}$ be the training set, a set of reduced sentences. We define the following functions:

- $C\left(\langle a,b\rangle, \langle c,d\rangle\right)$ for $a,c \in \mathcal{V}$, and $b,d \in \mathcal{T}$ is the number of times $\langle a,b\rangle$ and $\langle c,d\rangle$ are seen in the same reduced sentence in training data.[7] Formally,

$$C\left(\langle a,b\rangle, \langle c,d\rangle\right) = \sum_{\substack{\bar{S} \in \mathcal{TRAIN} \\ k,l=1..|\bar{S}|,\, l \neq k}} h\left(\bar{S}[k] = \langle a,b\rangle, \bar{S}[l] = \langle c,d\rangle\right) \tag{6.5}$$

where $h(x)$ is an indicator function that is 1 if $x$ is true, 0 if $x$ is false.

- $C\left(R, \langle a,b\rangle, \langle c,d\rangle\right)$ is the number of times $\langle a,b\rangle$ and $\langle c,d\rangle$ are seen in the same reduced sentence in training data, and $\langle a,b\rangle$ modifies $\langle c,d\rangle$ with relationship $R$. Formally,

$$C\left(R, \langle a,b\rangle, \langle c,d\rangle\right) = \sum_{\substack{\bar{S} \in \mathcal{TRAIN} \\ k,l=1..|\bar{S}|,\, l \neq k}} h(\bar{S}[k] = \langle a,b\rangle, \bar{S}[l] = \langle c,d\rangle, AF(k) = (l,R)) \tag{6.6}$$

- $F(R\,|\,\langle a,b\rangle, \langle c,d\rangle)$ is the probability that $\langle a,b\rangle$ modifies $\langle c,d\rangle$ with relationship $R$, given that $\langle a,b\rangle$ and $\langle c,d\rangle$ appear in the same reduced sentence. The maximum-likelihood

[7] Note that we count multiple co-occurrences in a single sentence, e.g. if $\bar{S} = (<a,b>, <c,d>, <c,d>)$ then $C(<a,b>, <c,d>) = C(<c,d>, <a,b>) = 2$.

estimate of $F(R \mid \langle a,b\rangle \,,\, \langle c,d\rangle\,)$ is:

$$\hat{F}(R \mid \langle a,b\rangle \,,\, \langle c,d\rangle\,) = \frac{C(R,\, \langle a,b\rangle \,,\, \langle c,d\rangle\,)}{C(\,\langle a,b\rangle \,,\, \langle c,d\rangle\,)} \tag{6.7}$$

We can now make the following approximation:

$$\begin{gathered} P(AF(j) = (h_j, R_j) \mid S, B) \approx \\ \frac{\hat{F}(R_j \mid \langle \bar{w}_j, \bar{t}_j\rangle \,,\, \langle \bar{w}_{h_j}, \bar{t}_{h_j}\rangle\,)}{\sum_{k=1..m, k\neq j, p\epsilon\mathcal{P}} \hat{F}(p \mid \langle \bar{w}_j, \bar{t}_j\rangle \,,\, \langle \bar{w}_k, \bar{t}_k\rangle\,)} \end{gathered} \tag{6.8}$$

where $\mathcal{P}$ is the set of all triples of non-terminals. The denominator is a normalising factor which ensures that

$$\sum_{k=1..m, k\neq j, p\epsilon\mathcal{P}} P(AF(j) = (k,p) \mid S, B) = 1$$

From (6.4) and (6.8):

$$\begin{gathered} P(D|S,B) \approx \\ \prod_{j=1}^{m} \frac{\hat{F}(R_j \mid \langle \bar{w}_j, \bar{t}_j\rangle \,,\, \langle \bar{w}_{h_j}, \bar{t}_{h_j}\rangle\,)}{\sum_{k=1..m, k\neq j, p\epsilon\mathcal{P}} \hat{F}(p \mid \langle \bar{w}_j, \bar{t}_j\rangle \,,\, \langle \bar{w}_k, \bar{t}_k\rangle\,)} \end{gathered} \tag{6.9}$$

The denominator of (6.9) is constant, so maximising $P(D|S,B)$ over $D$ for fixed $S, B$ is equivalent to maximising the product of the numerators, $\mathcal{N}(D|S,B)$. (This considerably simplifies the parsing process):

$$\mathcal{N}(D|S,B) = \prod_{j=1}^{m} \hat{F}(R_j \mid \langle \bar{w}_j, \bar{t}_j\rangle \,,\, \langle \bar{w}_{h_j}, \bar{t}_{h_j}\rangle\,) \tag{6.10}$$

### 6.2.3 The Distance Measure

An estimate based on the identities of the two tokens alone is problematic. Additional context, in particular the relative order of the two words and the distance between them, will also strongly influence the likelihood of one word modifying the other. For example consider the relationship between 'sales' and the three tokens of 'of' in the following sentence

**Example 2.** *Shaw, based in Dalton, Ga., has annual* **sales of** *about $ 1.18 billion, and has economies* **of** *scale and lower raw-material costs that are expected to boost the profitability* **of** *Armstrong 's brands, sold under the Armstrong and Evans-Black names .*

In this sentence 'sales' and 'of' co-occur three times. The parse tree in training data indicates a relationship in only one of these cases, so this sentence would contribute an estimate of $\frac{1}{3}$ that the two words are related. This seems unreasonably low given that 'sales of' is a strong collocation. The latter two instances of 'of' are so distant from 'sales' that it is unlikely that there will be a dependency.

This suggests that distance is a crucial variable when deciding whether two words are related. It is included in the model by defining an extra 'distance' variable, $\Delta$, and extending $C$, $F$ and $\hat{F}$ to include this variable. For example, $C(\langle a,b\rangle, \langle c,d\rangle, \Delta)$ is the number of times $\langle a,b\rangle$ and $\langle c,d\rangle$ appear in the same sentence at a distance $\Delta$ apart. (6.11) is then maximised instead of (6.10):

$$\mathcal{N}(D|S,B) = \prod_{j=1}^{m} \hat{F}(R_j \,|\, \langle \bar{w}_j, \bar{t}_j\rangle, \langle \bar{w}_{h_j}, \bar{t}_{h_j}\rangle, \Delta_{j,h_j}) \tag{6.11}$$

A simple example of $\Delta_{j,h_j}$ would be $\Delta_{j,h_j} = h_j - j$. However, other features of a sentence, such as punctuation, are also useful when deciding if two words are related. We have developed a heuristic 'distance' measure which takes several such features into account. The current distance measure $\Delta_{j,h_j}$ is the combination of 6 features, or questions (we motivate the choice of these questions qualitatively – section 6.4 gives quantitative results showing their merit):

**Question 1** Does the $h_j$th word precede or follow the $j$th word? English is a language with strong word order, so the order of the two words in surface text will clearly affect their dependency statistics.

**Question 2** Are the $h_j$th word and the $j$th word adjacent? English is largely right-branching and head-initial, which leads to a large proportion of dependencies being between adjacent words [8]. Table 6.1 shows just how local most dependencies are.

**Question 3** Is there a verb between the $h_j$th word and the $j$th word? Conditioning on the exact distance between two words by making $\Delta_{j,h_j} = h_j - j$ leads to severe sparse data problems. But Table 6.1 shows the need to make finer distance distinctions than just whether two words are adjacent. Consider the prepositions 'to', 'in' and 'of' in the following sentence:

[8] For example in '(John (likes (to (go (to (University (of Pennsylvania)))))))' all dependencies are between adjacent words.

| Distance | 1 | $\leq 2$ | $\leq 5$ | $\leq 10$ |
|---|---|---|---|---|
| Percentage | 74.2 | 86.3 | 95.6 | 99.0 |

Table 6.1: Percentage of dependencies vs. distance between the head words involved. These figures count baseNPs as a single word, and are taken from WSJ training data.

| Number of verbs | 0 | <=1 | <=2 |
|---|---|---|---|
| Percentage | 94.1 | 98.1 | 99.3 |

Table 6.2: Percentage of dependencies vs. number of verbs between the head words involved.

**Example 3.** *Oil stocks* **escaped** *the brunt of Friday 's selling and several were able to post gains , including Chevron , which* **rose** *5/8* **to** *66 3/8* **in** *Big Board composite trading* **of** *2.4 million shares .*

The prepositions' main candidates for attachment would appear to be the previous verb, 'rose', and the baseNP heads between each preposition and this verb. They are less likely to modify a more distant verb such as 'escaped'. Question 3 allows the parser to prefer modification of the most recent verb – effectively another, weaker preference for right-branching structures. Table 6.2 shows that 94% of dependencies do not cross a verb, giving empirical evidence that question 3 is useful.

**Questions 4, 5 and 6**

- Are there 0, 1, 2, or more than 2 'commas' between the $h_j$th word and the $j$th word? (All symbols tagged as a ',' or ':' are considered to be 'commas').
- Is there a 'comma' immediately following the first of the $h_j$th word and the $j$th word?
- Is there a 'comma' immediately preceding the second of the $h_j$th word and the $j$th word?

People find that punctuation is extremely useful for identifying phrase structure, and the parser described here also relies on it heavily. Commas are not considered to be words or modifiers in the dependency model – but they do give strong indications about the parse structure. Questions 4, 5 and 6 allow the parser to use this information.

### 6.2.4 Sparse Data

The maximum likelihood estimator in (6.7) is likely to be plagued by sparse data problems – $C(\langle\bar{w}_j,\bar{t}_j\rangle,\langle\bar{w}_{h_j},\bar{t}_{h_j}\rangle,\Delta_{j,h_j})$ may be too low to give a reliable estimate, or worse still it may be zero leaving the estimate undefined. We use a backing-off strategy (similar to the method in Chapter 5) to smooth these probabilities.

There are four estimates, $E_1$, $E_2$, $E_3$ and $E_4$, based respectively on: 1) both words and both tags; 2) $\bar{w}_j$ and the two POS tags; 3) $\bar{w}_{h_j}$ and the two POS tags; 4) the two POS tags alone.

$$E_1 = \tfrac{\eta_1}{\delta_1} \quad E_2 = \tfrac{\eta_2}{\delta_2} \quad E_3 = \tfrac{\eta_3}{\delta_3} \quad E_4 = \tfrac{\eta_4}{\delta_4} \tag{6.12}$$

where[9]

$$\begin{aligned}
\delta_1 &= C(\langle\bar{w}_j,\bar{t}_j\rangle,\langle\bar{w}_{h_j},\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) \\
\delta_2 &= C(\langle\bar{w}_j,\bar{t}_j\rangle,\langle\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) \\
\delta_3 &= C(\langle\bar{t}_j\rangle,\langle\bar{w}_{h_j},\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) \\
\delta_4 &= C(\langle\bar{t}_j\rangle,\langle\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) \\
\eta_1 &= C(R_j,\langle\bar{w}_j,\bar{t}_j\rangle,\langle\bar{w}_{h_j},\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) \\
\eta_2 &= C(R_j,\langle\bar{w}_j,\bar{t}_j\rangle,\langle\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) \\
\eta_3 &= C(R_j,\langle\bar{t}_j\rangle,\langle\bar{w}_{h_j},\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) \\
\eta_4 &= C(R_j,\langle\bar{t}_j\rangle,\langle\bar{t}_{h_j}\rangle,\Delta_{j,h_j})
\end{aligned} \tag{6.13}$$

Estimates 2 and 3 compete – for a given pair of words in test data both estimates may exist and they are equally 'specific' to the test case example. Chapter 5 suggests the following way of combining them, which favours the estimate appearing more often in training data:

---

9

$$C(\langle\bar{w}_j,\bar{t}_j\rangle,\langle\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) = \sum_{x\epsilon\mathcal{V}} C(\langle\bar{w}_j,\bar{t}_j\rangle,\langle x,\bar{t}_{h_j}\rangle,\Delta_{j,h_j})$$

$$C(\langle\bar{t}_j\rangle,\langle\bar{t}_{h_j}\rangle,\Delta_{j,h_j}) = \sum_{x\epsilon\mathcal{V}}\sum_{y\epsilon\mathcal{V}} C(\langle x,\bar{t}_j\rangle,\langle y,\bar{t}_{h_j}\rangle,\Delta_{j,h_j})$$

where $\mathcal{V}$ is the set of all words seen in training data; the other definitions of $C$ follow similarly.

$$E_{23} = \frac{\eta_2 + \eta_3}{\delta_2 + \delta_3} \tag{6.14}$$

This gives three estimates: $E_1$, $E_{23}$ and $E_4$, a similar situation to trigram language modeling for speech recognition [Jelinek 90], where there are trigram, bigram and unigram estimates. [Jelinek 90] describes a deleted interpolation method, which combines these estimates to give a 'smooth' estimate, and the model uses a variation of this idea:

**If** $E_1$ **exists**, i.e. $\delta_1 > 0$

$$\hat{F}(R_j \mid \langle \bar{w}_j, \bar{t}_j \rangle, \langle \bar{w}_{h_j}, \bar{t}_{h_j} \rangle, \Delta_{j,h_j}) =$$
$$\lambda_1 \times E_1 + (1 - \lambda_1) \times E_{23} \tag{6.15}$$

**Else If** $E_{23}$ **exists**, i.e. $\delta_2 + \delta_3 > 0$

$$\hat{F}(R_j \mid \langle \bar{w}_j, \bar{t}_j \rangle, \langle \bar{w}_{h_j}, \bar{t}_{h_j} \rangle, \Delta_{j,h_j}) =$$
$$\lambda_2 \times E_{23} + (1 - \lambda_2) \times E_4 \tag{6.16}$$

**Else**

$$\hat{F}(R_j \mid \langle \bar{w}_j, \bar{t}_j \rangle, \langle \bar{w}_{h_j}, \bar{t}_{h_j} \rangle, \Delta_{j,h_j}) = E_4 \tag{6.17}$$

[Jelinek 90] describes how to find $\lambda$ values in (6.15) and (6.16) which maximise the likelihood of held-out data. We have taken a simpler approach, namely:

$$\lambda_1 = \frac{\delta_1}{\delta_1 + 1}$$

$$\lambda_2 = \frac{\delta_2 + \delta_3}{\delta_2 + \delta_3 + 1} \tag{6.18}$$

These $\lambda$ values have the desired property of increasing as the denominator of the more 'specific' estimator increases. We think that a proper implementation of deleted interpolation is likely to improve results, although basing estimates on co-occurrence counts alone has the advantage of reduced training times.

### 6.2.5 The BaseNP Model

The overall model would be simpler if we could do without the baseNP model and frame everything in terms of dependencies. However, the baseNP model is needed for two reasons. First, while adjacency between words is a good indicator of whether there is some relationship between them, this indicator is made substantially stronger if baseNPs are reduced to a single word. Second, it means that words internal to baseNPs are not included in the co-occurrence counts in training data. Otherwise, in a phrase like 'The Securities and Exchange Commission closed yesterday', pre-modifying nouns like 'Securities' and 'Exchange' would be included in co-occurrence counts, when in practice there is no way that they can modify words outside their baseNP.

The baseNP model can be viewed as tagging the gaps between words with $S(tart)$, $C(ontinue)$, $E(nd)$, $B(etween)$ or $N(ull)$ symbols, respectively meaning that the gap is at the start of a $BaseNP$, continues a $BaseNP$, is at the end of a $BaseNP$, is between two adjacent $baseNP$s, or is between two words which are both not in $BaseNPs$. We call the gap before the $i$th word $G_i$ (a sentence with $n$ words has $n-1$ gaps). For example,

[ John Smith ] [ the president ] of [ IBM ] has announced [ his resignation ] [ yesterday ] $\Rightarrow$

John **C** Smith **B** the **C** president **E** of **S** IBM **E** has **N** announced **S** his **C** resignation **B** yesterday

The baseNP model considers the words directly to the left and right of each gap, and whether there is a comma between the two words (we write $c_i = 1$ if there is a comma, $c_i = 0$ otherwise). Probability estimates are based on counts of consecutive pairs of words in **unreduced** training data sentences, where baseNP boundaries define whether gaps fall into the $S$, $C$, $E$, $B$ or $N$ categories. The probability of a baseNP sequence in an unreduced sentence $S$ is then:

$$\prod_{i=2...n} \hat{P}(G_i \mid w_{i-1}, t_{i-1}, w_i, t_i, c_i) \tag{6.19}$$

The estimation method is analogous to that described in the sparse data section of this paper. The method is similar to that described in [Ramshaw and Marcus 95, Church 88], where baseNP detection is also framed as a tagging problem.

### 6.2.6 Summary of the Model

The probability of a parse tree $T$, given a sentence $S$, is:

$$P(T|S) = P(B, D|S) = P(B|S) \times P(D|S, B)$$

The denominator in Equation (6.9) is not actually constant for different baseNP sequences, but we make this approximation for the sake of efficiency and simplicity. In practice this is a good approximation because most baseNP boundaries are very well defined, so parses which have high enough $P(B|S)$ to be among the highest scoring parses for a sentence tend to have identical or very similar baseNPs. Parses are ranked by the following quantity[10]:

$$\hat{P}(B|S) \times \mathcal{N}(D|S, B) \tag{6.20}$$

Equations (6.19) and (6.11) define $\hat{P}(B|S)$ and $\mathcal{N}(D|S, B)$. The parser finds the tree which maximises (6.20) subject to the hard constraint that dependencies cannot cross.

### 6.2.7 Some Further Improvements to the Model

This section describes two modifications which improve the model's performance.

• In addition to conditioning on whether dependencies cross commas, a single constraint concerning punctuation is introduced. If for any constituent `Z` in the chart `Z` $\rightarrow$ `<..X Y..>` two of its children `X` and `Y` are separated by a comma, then the last word in `Y` must be directly followed by a comma, or must be the last word in the sentence. In training data 96% of commas follow this rule. The rule also has the benefit of improving efficiency by reducing the number of constituents in the chart.

• The model we have described thus far takes the single best sequence of tags from the tagger, and it is clear that there is potential for better integration of the tagger and parser. We have tried two modifications. First, the current estimation methods treat occurrences of the same word with different POS tags as effectively distinct types. Tags can be ignored when lexical information is available by defining

$$C\,(a, c) = \sum_{b,d \epsilon \mathcal{T}} C\,(\,\langle a, b\rangle\,,\,\langle c, d\rangle\,) \tag{6.21}$$

[10]In fact we also model the set of unary productions, $U$, in the tree, which are of the form $P \rightarrow < C_1 >$. This introduces an additional term, $\hat{P}(U|B, S)$, into (6.20).

where $\mathcal{T}$ is the set of all tags. Hence $C(a,c)$ is the number of times that the words $a$ and $c$ occur in the same sentence, ignoring their tags. The other definitions in (6.13) are similarly redefined, with POS tags only being used when backing off from lexical information. This makes the parser less sensitive to tagging errors.

Second, for each word $w_i$ the tagger can provide the distribution of tag probabilities $P(t_i|S)$ (given the previous two words are tagged as in the best overall sequence of tags) rather than just the first best tag. The score for a parse in equation (6.20) then has an additional term, $\prod_{i=1}^{n} P(t_i|S)$, the product of probabilities of the tags which it contains.

Ideally we would like to integrate POS tagging into the parsing model rather than treating it as a separate stage. This is an area for future research.

| MODEL | $\leq$ 40 Words (2245 sentences) | | | | |
|---|---|---|---|---|---|
| | LR | LP | CBs | 0 CBs | $\leq$ 2 CBs |
| (1) | 84.9% | 84.9% | 1.32 | 57.2% | 80.8% |
| (2) | 85.4% | 85.5% | 1.21 | 58.4% | 82.4% |
| (3) | 85.5% | 85.7% | 1.19 | 59.5% | 82.6% |
| (4) | 85.8% | 86.3% | 1.14 | 59.9% | 83.6% |
| SPATTER | 84.6% | 84.9% | 1.26 | 56.6% | 81.4% |

| MODEL | $\leq$ 100 Words (2416 sentences) | | | | |
|---|---|---|---|---|---|
| | LR | LP | CBs | 0 CBs | $\leq$ 2 CBs |
| (1) | 84.3% | 84.3% | 1.53 | 54.7% | 77.8% |
| (2) | 84.8% | 84.8% | 1.41 | 55.9% | 79.4% |
| (3) | 85.0% | 85.1% | 1.39 | 56.8% | 79.6% |
| (4) | 85.3% | 85.7% | 1.32 | 57.2% | 80.8% |
| SPATTER | 84.0% | 84.3% | 1.46 | 54.0% | 78.8% |

Table 6.3: Results on Section 23 of the WSJ Treebank. **(1)** is the basic model; **(2)** is the basic model with the punctuation rule described in section 6.2.7: **(3)** is model (2) with POS tags ignored when lexical information is present; **(4)** is model (3) with probability distributions from the POS tagger. **LR/LP** = labeled recall/precision. **CBs** is the average number of crossing brackets per sentence. **0 CBs,** $\leq$ 2 **CBs** are the percentage of sentences with 0 or $\leq$ 2 crossing brackets respectively.

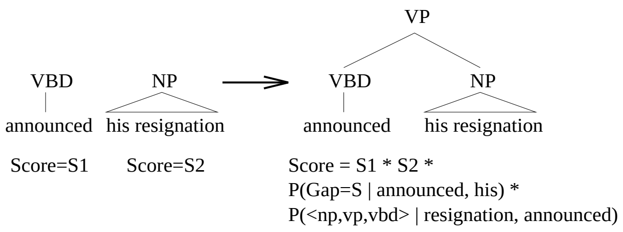



Figure 6.4: Diagram showing how two constituents join to form a new constituent. Each operation gives two new probability terms: one for the baseNP gap tag between the two constituents, and the other for the dependency between the head words of the two constituents.

## 6.3 The Parsing Algorithm

The parsing algorithm is a simple bottom-up chart parser. There is no grammar as such, although in practice any dependency with a triple of non-terminals which has not been seen in training data will get zero probability. Thus the parser searches through the space of all trees with non-terminal triples seen in training data. Probabilities of baseNPs in the chart are calculated using (6.19), while probabilities for other constituents are derived from the dependencies and baseNPs that they contain. A dynamic programming algorithm is used: if two proposed constituents span the same set of words, have the same label, head, and distance from the head to the left and right end of the constituent, then the lower probability constituent can be safely discarded. Figure 6.4 shows how constituents in the chart combine in a bottom-up manner.

## 6.4 Results

The parser was trained on sections 02 - 21 of the Wall Street Journal portion of the Penn Treebank [Marcus et al. 93] (approximately 40,000 sentences), and tested on section 23 (2,416 sentences). For comparison SPATTER [Magerman 95, Jelinek et al. 94] was also tested on section 23. We use the PARSEVAL measures [Black et al. 91] to compare performance:

$$\textbf{Labeled Precision} = \frac{number\ of\ correct\ constituents\ in\ proposed\ parse}{number\ of\ constituents\ in\ proposed\ parse}$$

$$\textbf{Labeled Recall} = \frac{number\ of\ correct\ constituents\ in\ proposed\ parse}{number\ of\ constituents\ in\ treebank\ parse}$$

| Distance Measure | Lexical Information | LR | LP | CBs |
|---|---|---|---|---|
| Yes | Yes | 85.0% | 85.1% | 1.39 |
| Yes | No | 76.1% | 76.6% | 2.26 |
| No | Yes | 80.9% | 83.6% | 1.51 |

Table 6.4: The contribution of various components of the model. The results are for all sentences of $\leq 100$ words in section 23 using model (3). For 'no lexical information' all estimates are based on POS tags alone. For 'no distance measure' the distance measure is Question 1 alone (i.e. whether $\bar{w}_j$ precedes or follows $\bar{w}_{h_j}$).

**Crossing Brackets** = number of constituents which violate constituent boundaries with a constituent in the treebank parse.

For a constituent to be 'correct' it must span the same set of words (ignoring punctuation, i.e. all tokens tagged as commas, colons or quotes) and have the same label[11] as a constituent in the treebank parse. Four configurations of the parser were tested: **(1)** The basic model; **(2)** The basic model with the punctuation rule described in section 6.2.7; **(3)** Model (2) with tags ignored when lexical information is present, as described in 6.2.7; and **(4)** Model (3) also using the full probability distributions for POS tags. We should emphasize that test data outside of section 23 was used for all development of the model, avoiding the danger of implicit training on section 23. Table 6.3 shows the results of the tests. Table 6.4 shows results which indicate how different parts of the system contribute to performance.

### 6.4.1 Performance Issues

All tests were made on a Sun SPARCServer 1000E, using 100% of a 60Mhz SuperSPARC processor. The parser uses around 180 megabytes of memory, and training on 40,000 sentences (essentially extracting the co-occurrence counts from the corpus) takes under 15 minutes. Loading the hash table of bigram counts into memory takes approximately 8 minutes.

Two strategies are employed to improve parsing efficiency. First, a constant probability

[11] SPATTER collapses `ADVP` and `PRT` to the same label, for comparison we also removed this distinction when calculating scores.

threshold is used while building the chart – any constituents with lower probability than this threshold are discarded. If a parse is found, it must be the highest ranked parse by the model (as all constituents discarded have lower probabilities than this parse and could not, therefore, be part of a higher probability parse). If no parse is found, the threshold is lowered and parsing is attempted again. The process continues until a parse is found.

Second, a beam search strategy is used. For each span of words in the sentence the probability, $P_h$, of the highest probability constituent is recorded. All other constituents spanning the same words must have probability greater than $\frac{P_h}{\beta}$ for some constant beam size $\beta$ – constituents which fall out of this beam are discarded. The method risks introducing search-errors, but in practice efficiency can be greatly improved with virtually no loss of accuracy. Table 6.5 shows the trade-off between speed and accuracy as the beam is narrowed.

| Beam Size $\beta$ | Speed Sentences/minute | LR | LP | CBs |
|---|---|---|---|---|
| 1000 | 118 | 84.9% | 85.1% | 1.39 |
| 150 | 166 | 84.8% | 85.1% | 1.38 |
| 20 | 217 | 84.7% | 85.0% | 1.40 |
| 3 | 261 | 84.1% | 84.5% | 1.44 |
| 1.5 | 283 | 83.7% | 84.1% | 1.48 |
| 1.2 | 289 | 83.5% | 83.9% | 1.50 |

Table 6.5: The trade-off between speed and accuracy as the beam-size is varied. Model (3) was used for this test on all sentences $\leq 100$ words in section 23.

## 6.5 Further Discussion

### 6.5.1 Representational Issues

If we examine this chapter's model in the light of the representation proposals of chapter 3, a number of points become clear:

- The model contains all of the representation proposals, with the exception of subcategorization frames. (Unfortunately, the model structure as it stands does not readily allow the addition of subcategorization.)

- Fortunately, as argued in section 3.3.7, the distance variable gives a close approximation to subcategorization (this almost certainly saves the model). Unfortunately, the distance measure is the second one described in section 3.3.7, and therefore breaks down — particularly as a model of subcategorization — in some cases.
- The model is limited to conditioning on properties of the surface string alone, rather than on any previously built structure. This is a quite severe limitation (and precludes the incorporation of subcategorization and wh-movement, two refinements in the next chapter's model).

### 6.5.2 Mathematical Issues

There are also a few mathematical problems with the model:

- The normalization factor in equation 6.8 means that the estimate cannot be justified as maximum-likelihood estimation; see section 2.3.3 for more discussion of this problem. Furthermore, even with this normalization factor the model is still deficient, due to structures with crossing dependencies receiving some probability.
- The model has no principled way of dealing with unary rules, currently a probability $P(Rule \mid word, tag)$[12] is multiplied into the parse score for each unary rule, but this is rather ad-hoc.

### 6.5.3 Summary

In summary, this chapter's model includes most of the representation proposals in chapter 3, and this is its strength. Its main weaknesses are some mathematical problems, and a model structure that prevents the representation of subcategorization, wh-movement, or a "correct" distance measure.

[12] The probability, given that a particular (word,tag) pair is seen in a sentence, that a unary rule *Rule* is seen with (word,tag) as a head somewhere in the parse tree.

# Chapter 7

# Three Generative, Lexicalized Models for Statistical Parsing

## 7.1 Introduction

The problems with the previous chapter's model, described in section 6.5, lead us to a new approach to the parsing problem. The models in this chapter define a joint probability $P(T, S)$ over Tree-Sentence pairs. A history-based model is used: a parse tree is represented as a sequence of decisions, the decisions being made in a head-centered, top-down derivation of the parse tree. Representing a parse-tree in this way allows independence assumptions that lead to parameters conditioned on lexical heads: head-projection, subcategorization, complement/adjunct placement, dependency, distance, and wh-movement (gap propagation) parameters.

We first describe three parsing models based on this approach, giving results on the Penn WSJ treebank. We then give a detailed breakdown of the results, in terms of accuracy on different kinds of constituents and attachment ambiguities — the intention being to get a better idea of the parser's strengths and weaknesses. The next chapter gives a much fuller discussion of the modeling choices made, their influence on accuracy, and also gives comparisons to related work.

The three new models are:

- In **Model 1** we show how to extend Probabilistic Context Free Grammars (PCFGs) to lexicalized grammars in a way that results in a quite similar model to that described in Chapter 6. Most importantly, it again has parameters corresponding to dependencies between pairs of head-words. We also show how to incorporate the "distance" measure into these models, by generalizing the model to a history-based approach. The advantages of the new model over the work of the previous chapter are:

- The model is not deficient (i.e., $\sum P(T,S) = 1$); Unary rules are handled in a quite natural way by the model.
- The distance measure is slightly different, and is improved. For example, the adjacency variable now corresponds directly to right-branching structures.
- The model shows an accuracy improvement over the models in Chapter 6 of 1.5%/1.9% recall/precision. The later models (2 and 3) show an overall improvement of 2.2%/2.4% recall/precision.
- The models in Chapter 6 were constrained to conditioning on features of the surface string alone, whereas the models in this chapter can potentially condition on any (previously generated) structure: we effectively make use of this in models 2 and 3.
- Part-of-speech tagging is naturally incorporated into the model.
- The model defines a joint probability measure $P(T,S)$, and can therefore be used as a language model for applications such as speech recognition or machine translation. It also means that it can be trained in an unsupervised manner using the EM algorithm [Dempster, Laird and Rubin 77], unlike models that only define conditional probabilities $P(T \mid S)$.

• In **Model 2**, we extend the parser to make the complement/adjunct distinction, which will be important for most applications using the output from the parser (for example, distinguishing "IBM" as a complement, from "yesterday", a temporal adjunct, in "yesterday IBM bought Lotus").

  Model 2 is also extended to have parameters corresponding directly to probability distributions over subcategorization frames for head-words; this leads to an improvement in accuracy.

• In **Model 3** we give a probabilistic treatment of wh-movement, which is derived from the analysis in Generalized Phrase Structure Grammar [Gazdar et al. 95]. The output of the parser is now enhanced to show trace co-indexations in wh-movement cases. The parameters in this model are interesting in that they correspond directly to the probability of propagating GPSG-style *slash* features through parse trees, potentially allowing the model to learn island constraints.

In summary, the work in this chapter makes two advances over previous models. First, Model 1 performs significantly better than the model in Chapter 6, and Models 2 and 3 give further improvements — our final results are 88.3/88.0% constituent precision/recall, an average improvement of 2.3% over results in Chapter 6. Second, the parsers in Chapter 6 and [Charniak 97, Ratnaparkhi 97, Goodman 97, Magerman 95, Jelinek et al. 94] produce trees without information about wh-movement or subcategorization. Most NLP applications will need this information to extract predicate-argument structure from parse trees.

### 7.1.1 Probabilistic Context-Free Grammars

Probabilistic context-free grammars are the starting point for the models in this chapter. For this reason we briefly recap the theory behind non-lexicalized PCFGs, before moving to the lexicalized case. (See chapter 2 for a full discussion of PCFG models.)

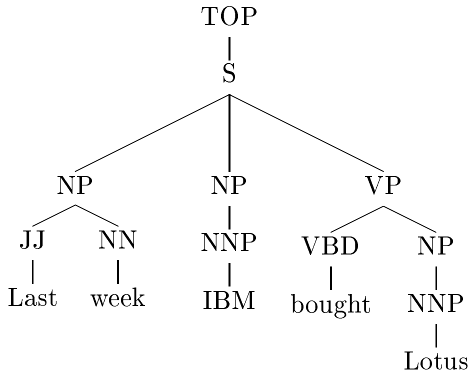
In general, a statistical parsing model defines the conditional probability, $\mathcal{P}(T \mid S)$, for each candidate parse tree $T$ for a sentence $S$. The parser itself is an algorithm that searches for the tree, $T_{best}$, that maximises $\mathcal{P}(T \mid S)$. A generative model uses the observation that maximising $\mathcal{P}(T, S)$ is equivalent to maximising $\mathcal{P}(T \mid S)$: [1]

$$T_{best} = \arg\max_T \mathcal{P}(T \mid S) = \arg\max_T \frac{\mathcal{P}(T,S)}{\mathcal{P}(S)} = \arg\max_T \mathcal{P}(T,S) \tag{7.1}$$

$\mathcal{P}(T, S)$ is then defined by attaching probabilities to a top-down derivation of the tree. Each context-free rule is of the format $LHS \rightarrow RHS$ ($LHS$ stands for "Left hand side", $RHS$ stands for "Right hand side"). In a PCFG, for a tree derived by $n$ applications of context-free rules $LHS_i \rightarrow RHS_i$, $1 \leq i \leq n$,

$$\mathcal{P}(T,S) = \prod_{i=1..n} \mathcal{P}(RHS_i \mid LHS_i) \tag{7.2}$$

[1] $\mathcal{P}(S)$ is constant, hence maximising $\frac{\mathcal{P}(T,S)}{\mathcal{P}(S)}$ is equivalent to maximising $\mathcal{P}(T, S)$.




| Internal rules | | | Lexical rules | | |
|---|---|---|---|---|---|
| TOP | -> | S | JJ | -> | Last |
| S | -> | NP NP VP | NN | -> | week |
| NP | -> | JJ NN | NNP | -> | IBM |
| NP | -> | NNP | VBD | -> | bought |
| VP | -> | VBD NP | NNP | -> | Lotus |
| NP | -> | NNP | | | |

Figure 7.1: A non-lexicalized parse tree, and a list of the rules it contains.

The rules are either internal to the tree, where $LHS$ is a non-terminal and $RHS$ is a string of one or more non-terminals; or lexical, where $LHS$ is a part-of-speech tag and $RHS$ is a word. See figure 7.1 for an example.

A central problem in PCFGs is to define the conditional probability $\mathcal{P}(RHS \mid LHS)$ for each rule $LHS \to RHS$ in the grammar. A simple way to do this is to take counts from a treebank and then to use the maximum likelihood estimate

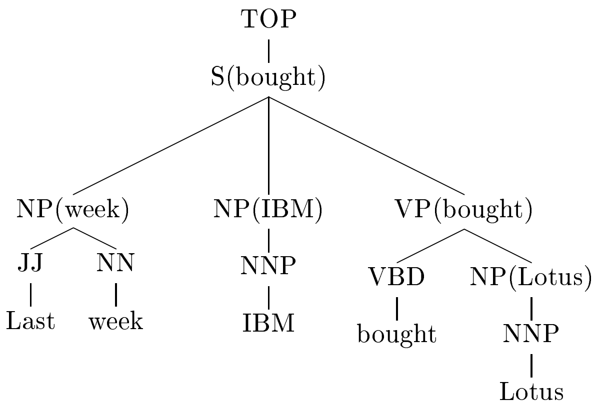
$$\mathcal{P}(RHS \mid LHS) = \frac{Count(LHS \to RHS)}{Count(LHS)} \tag{7.3}$$

### 7.1.2 Lexicalized PCFGs

A PCFG can be lexicalized[2] by associating a word $w$ and a part-of-speech (POS) tag $t$ with each non-terminal $X$ in the tree. See figure 7.2 for an example tree.

The PCFG model can be applied to these lexicalized rules and trees in exactly the same way as before. Whereas before the non-terminals were simple, for example "S" or "NP", they are now extended to include a word and part-of-speech tag, for example "S(bought,VBD)" or "NP(IBM,NNP)". Thus we write a non-terminal as $X(x)$, where $x = \langle w, t \rangle$, and $X$ is a constituent label. Formally, nothing has changed, we have just vastly increased the number of non-terminals in the grammar (by up to a factor of $|\mathcal{V}| \times |\mathcal{T}|$,

[2] We find lexical heads in Penn treebank data using the rules described in Appendix A.




```
TOP          ->  S(bought)
S(bought)    ->  NP(week)      NP(IBM)     VP(bought)
NP(week)     ->  JJ(Last)      NN(week)
NP(IBM)      ->  NNP(IBM)
VP(bought)   ->  VBD(bought)   NP(Lotus)
NP(Lotus)    ->  NNP(Lotus)
```

Figure 7.2: A lexicalized parse tree, and a list of the rules it contains. For brevity we omit the POS tag associated with each word.

where $|\mathcal{V}|$ is the number of words in the vocabulary, and $|\mathcal{T}|$ is the number of part of speech tags).

While nothing has changed from a formal point of view, the practical consequences of expanding the number of non-terminals quickly become apparent when attempting to define a method for parameter estimation. The simplest solution would be to use the maximum-likelihood estimate as in equation 7.3, for example estimating the probability of `S(bought) -> NP(week) NP(IBM) VP(bought)` as

$$\begin{gathered}\mathcal{P}(\texttt{NP(week) NP(IBM) VP(bought)} \mid \texttt{S(bought)}) = \\ \frac{Count(\texttt{S(bought)} \rightarrow \texttt{NP(week) NP(IBM) VP(bought)})}{Count(\texttt{S(bought)})}\end{gathered} \tag{7.4}$$

But the addition of lexical items makes the statistics for this estimate very sparse: the count for the denominator is likely to be relatively low, and the number of outcomes (possible lexicalized *RHS*s) is huge, meaning that the numerator is very likely to be zero. Predicting the whole lexicalized rule in one go is too big a step.

One way to overcome these sparse data problems is to break down the generation of the $RHS$ of each rule into a sequence of smaller steps, such that:

1. The steps are small enough for the parameter estimation problem to be tractable (providing that smoothing techniques are used to mitigate remaining sparse data problems).
2. The independence assumptions made are linguistically plausible. Model 1 uses a decomposition where parameters corresponding to lexical dependencies are a natural result, and also incorporates a preference for right-branching structures through conditioning on "distance"; Model 2 extends the decomposition to include a step where subcategorization frames are chosen probabilistically; Model 3 handles wh-movement by adding parameters corresponding to $slash$ categories being passed from the parent of the rule to one of its children, or being discharged as a trace.

## 7.2 Model 1

### 7.2.1 The Basic Model

This section describes how the generation of the $RHS$ of rule is broken down into a sequence of smaller steps in model 1. The first thing to note is that each rule in a lexicalized PCFG has the form[3]:

$$P(h) \rightarrow L_n(l_n)...L_1(l_1)H(h)R_1(r_1)...R_m(r_m) \tag{7.5}$$

$H$ is the head-child of the phrase, which inherits the head-word $h$ from its parent $P$. $L_1...L_n$ and $R_1...R_m$ are left and right modifiers of $H$. Either $n$ or $m$ may be zero, and $n = m = 0$ for unary rules. Figure 7.2 shows a tree which will be used as an example throughout this chapter. We will extend the left and right sequences to include a terminating `STOP` symbol, allowing a Markov process to model the left and right sequences. Thus $L_{n+1} = R_{m+1} = \texttt{STOP}$. (See section 2.4 for a discussion of the use of Markov models for probabilities over sequences.)

For example, in `S(bought,VBD)` $\rightarrow$ `NP(week,NN) NP(IBM,NNP) VP(bought,VBD)`:

| | | |
|---|---|---|
| $n = 2$ | $m = 0$ | P = `S` |
| H = `VP` | $L_1$ = `NP` | $L_2$ = `NP` |
| $L_3$ = `STOP` | $R_1$ = `STOP` | h = $\langle$`bought, VBD`$\rangle$ |
| $l_1$ = $\langle$`IBM, NNP`$\rangle$ | $l_2$ = $\langle$`week, NN`$\rangle$ | |

The probability of a rule can be rewritten (exactly) using the chain rule of probabilities:

$$\begin{aligned}
&\mathcal{P}(L_{n+1}(l_{n+1})...L_1(l_1)H(h)R_1(r_1)...R_{m+1}(r_{m+1}) \mid P(h)) = \\
&\mathcal{P}_h(H \mid P(h)) \times \\
&\prod_{i=1...n+1} \mathcal{P}_l(L_i(l_i) \mid L_1(l_1)...L_{i-1}(l_{i-1}), P(h), H) \times \\
&\prod_{j=1...m+1} \mathcal{P}_r(R_j(r_j) \mid L_1(l_1)...L_{n+1}(l_{n+1}), R_1(r_1)...R_{j-1}(r_{j-1}), P(h), H)
\end{aligned} \tag{7.6}$$

(the subscripts $h$, $l$ and $r$ are used to denote the head, left modifier and right modifier parameters respectively).

[3]With the exception of the top rule in the tree, which has the form `TOP` $\rightarrow$ `H(h)`.

Next, we make the assumption that the modifiers are generated independently of each other,

$$\mathcal{P}_l(L_i(l_i) \mid L_1(l_1)...L_{i-1}(l_{i-1}), P(h), H) = \mathcal{P}_l(L_i(l_i) \mid P(h), H) \quad (7.7)$$

$$\mathcal{P}_r(R_j(r_j) \mid L_1(l_1)...L_{n+1}(l_{n+1}), R_1(r_1)...R_{j-1}(r_{j-1}), P(h), H) = \mathcal{P}_r(R_j(r_j) \mid P(h), H) \quad (7.8)$$

In summary, the generation of the $RHS$ of a rule such as (7.5), given the $LHS$, has been decomposed into three steps[4]:

1. Generate the head constituent label of the phrase, with probability $\mathcal{P}_H(H \mid P, h)$.
2. Generate modifiers to the left of the head with probability $\prod_{i=1..n+1} \mathcal{P}_L(L_i(l_i)|P, h, H)$, where $L_{n+1}(l_{n+1}) =$ `STOP`. The `STOP` symbol is added to the vocabulary of non-terminals, and the model stops generating left modifiers when it is generated.
3. Generate modifiers to the right of the head with probability $\prod_{i=1..m+1} \mathcal{P}_R(R_i(r_i)|P, h, H)$. $R_{m+1}(r_{m+1})$ is defined as `STOP`.

For example, the probability of the rule `S(bought) -> NP(week) NP(IBM) VP(bought)` would be estimated as

$$\mathcal{P}_h(\texttt{VP} \mid \texttt{S,bought}) \times \mathcal{P}_l(\texttt{NP(IBM)} \mid \texttt{S,VP,bought}) \times$$
$$\mathcal{P}_l(\texttt{NP(week)} \mid \texttt{S,VP,bought}) \times \mathcal{P}_l(\texttt{STOP} \mid \texttt{S,VP,bought}) \times$$
$$\mathcal{P}_r(\texttt{STOP} \mid \texttt{S,VP,bought})$$

### 7.2.2 History-Based Models

The next section describes the addition of distance to the model, but first we extend the model to be "history-based", an extension that the distance model uses. ([Black et al. 92b] introduced history-based models for parsing; section 2.7 discusses these models.) Equations 7.7 and 7.8 made the independence assumption that each modifier is generated independently of the others. In general, though, each modifier could depend on any function

[4] An exception is the first rule in the tree, `TOP` $\rightarrow$ `H(h)`, which has probability $P_{TOP}(H, h|TOP)$

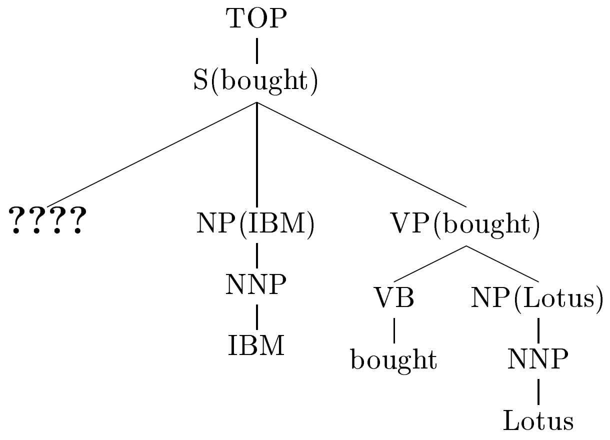



Figure 7.3: A partially completed tree derived depth-first. ???? marks the position of the next modifier to be generated — it could be a Non-terminal/head-word/head-tag triple, or the `STOP` symbol. The distribution over possible symbols in this position could be conditioned on *any* previously generated structure, i.e., any structure appearing in the figure.

$\Phi$ of the previous modifiers, head/parent category and head word.

$$\begin{aligned}
&\mathcal{P}_l(L_i(l_i) \mid L_1(l_1)...L_{i-1}(l_{i-1}), P(h), H) = \\
&\mathcal{P}_l(L_i(l_i) \mid \Phi\,(L_1(l_1)...L_{i-1}(l_{i-1}), P(h), H))
\end{aligned} \tag{7.9}$$

$$\begin{aligned}
&\mathcal{P}_r(R_j(r_j) \mid L_1(l_1)...L_{n+1}(l_{n+1}), R_1(r_1)...R_{j-1}(r_{j-1}), P(h), H) = \\
&\mathcal{P}_r(R_j(r_j) \mid \Phi\,(L_1(l_1)...L_{n+1}(l_{n+1}), R_1(r_1)...R_{j-1}(r_{j-1}), P(h), H))
\end{aligned} \tag{7.10}$$

In equations 7.7 and 7.8, $\Phi$ was chosen to ignore everything but $P$, $H$ and $h$.

Furthermore, if the top down derivation order is fully specified, then the probability of generating a modifier can be conditioned on *any* of the structure that has been previously generated. The remainder of this chapter assumes that the derivation order is depth-first — that is, each modifier recursively generates the sub-tree below it before the next modifier is generated. Figure 7.3 gives an example that illustrates this.

### 7.2.3 Adding Distance to the Model

The models in Chapter 6 showed that the distance between words standing in head-modifier relationships was important, in particular that it is important to capture a preference for right-branching structures (which *almost* translates into a preference for dependencies between adjacent words), and a preference for dependencies not to cross a verb.

Thus far the model has assumed that the modifiers are generated independently of each other (equations 7.7 and 7.8). Distance can be incorporated into the model by increasing the amount of dependence between the modifiers. If the derivation order is fixed to be depth-first, as in the general case of history-based models discussed in the preceding section, the model can condition on any structure *below* the preceding modifiers.

For the moment we exploit this by making the approximations

$$\mathcal{P}_l(L_i(l_i) \mid H, P, h, L_1(l_1)...L_{i-1}(l_{i-1})) = \mathcal{P}_l(L_i(l_i) \mid H, P, h, distance_l(i-1)) \tag{7.11}$$

$$\mathcal{P}_r(R_i(r_i) \mid H, P, h, R_1(r_1)...R_{i-1}(r_{i-1})) = \mathcal{P}_r(R_i(r_i) \mid H, P, h, distance_r(i-1)) \tag{7.12}$$

where $distance_l$ and $distance_r$ are functions of the surface string below the previous modifiers. (see figure 7.4). The distance measure is similar to that in Chapter 6, a vector with the following 2 elements: (1) is the string of zero length? (Allowing the model to learn a preference for right-branching structures); (2) does the string contain a verb? (Allowing the model to learn a preference for modification of the most recent verb).[5] See section 3.3.7 for motivation for the distance measure; see section 8.1 for further discussion of the distance measure within the models of this chapter.

[5]In the models described in [Collins 97], there was a third question concerning punctuation: (3) Does the string contain 0, 1, 2 or $>2$ commas? (where a comma is anything tagged as "," or ":"). The model described in this chapter has a cleaner incorporation of punctuation into the generative process, as described in section 7.5.3.

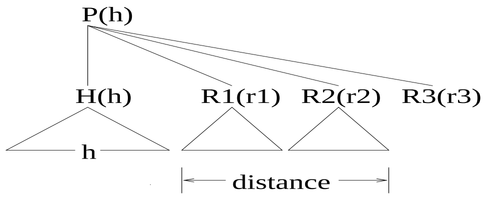



Figure 7.4: The next child, $R_3(r_3)$, is generated with probability $\mathcal{P}(R_3(r_3) \mid P, H, h, distance_r(2))$. The *distance* is a function of the surface string below previous modifiers $R_1$ and $R_2$. In principle the model could condition on any structure dominated by $H$, $R_1$ or $R_2$ (or, for that matter, on any structure previously generated elsewhere in the tree).

## 7.3 Model 2: The complement/adjunct distinction and subcategorization

The tree in figure 7.2 illustrates the importance of the complement/adjunct distinction. It would be useful to identify "IBM" as a subject, and "Last week" as an adjunct (temporal modifier), but this distinction is not made in the tree, as both `NP`s are in the same position[6] (sisters to a VP under an S node). From here on we will identify complements[7] by attaching a "-C" suffix to non-terminals. Figure 7.5 shows the tree in figure 7.2 with added complement markings.

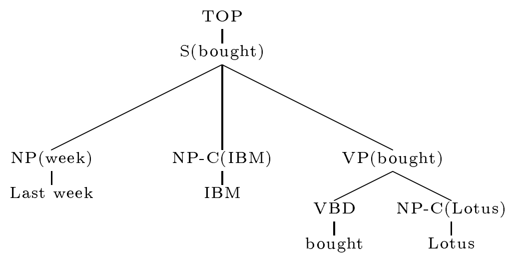



Figure 7.5: A tree with the "-C" suffix used to identify complements. "IBM" and "Lotus" are in subject and object position respectively. "Last week" is an adjunct.

A post-processing stage could add this detail to the parser output, but we give two reasons for making the distinction while parsing:

[6] Except "IBM" is closer to the VP, but note that "IBM" is also the subject in "IBM last week bought Lotus".

[7] We use the term *complement* in a broad sense that includes both complements and specifiers under the terminology of Government and Binding.

1. Identifying complements is complex enough to warrant a probabilistic treatment. Lexical information is needed — for example, knowledge that "week" is likely to be a temporal modifier. Knowledge about subcategorization preferences — for example that a verb takes exactly one subject — is also required. (For example, "week" can sometimes be a subject, as in *Last week was a good one*, so the model must balance the preference for having a subject against the relative improbability of "week" being the head-word of a subject.)

   These problems are not restricted to `NP`s, compare "The spokeswoman said (`SBAR` that the asbestos was dangerous)" vs. "Bonds beat short-term investments (`SBAR` because the market is down)", where an `SBAR` headed by "that" is a complement, but an `SBAR` headed by "because" is an adjunct.

2. Making the complement/adjunct distinction while parsing may help parsing accuracy. The assumption that complements are generated independently of each other often leads to incorrect parses. See figure 7.6 for further explanation.

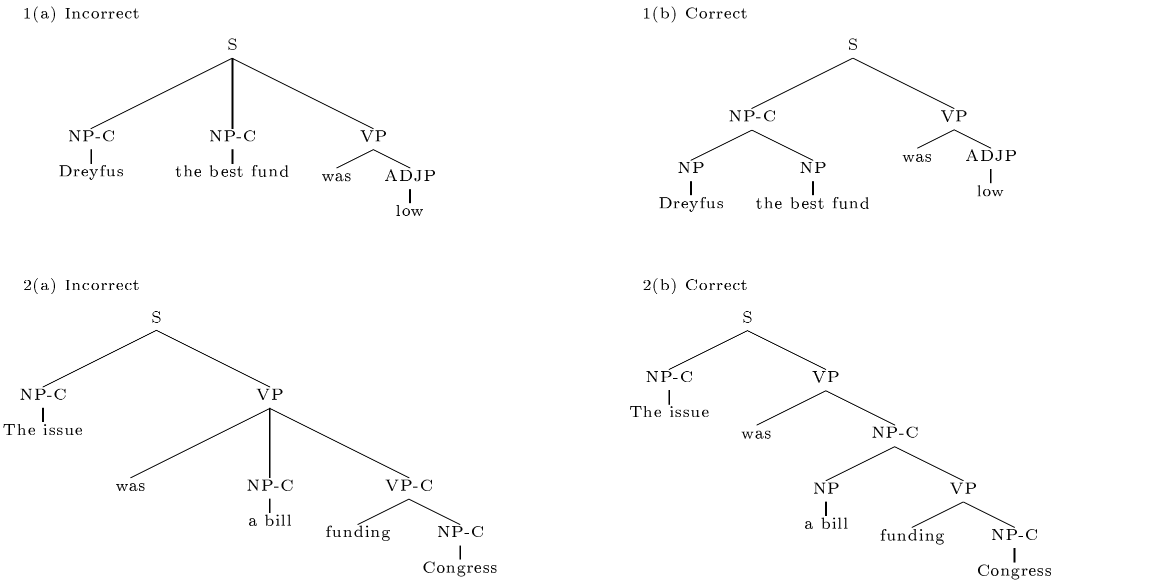



Figure 7.6: Two examples where the assumption that modifiers are generated independently of each other leads to errors. In (1) the probability of generating both "Dreyfus" and "fund" as subjects, $\mathcal{P}(\texttt{NP-C(Dreyfus)} \mid \texttt{S,VP,was}) * \mathcal{P}(\texttt{NP-C(fund)} \mid \texttt{S,VP,was})$ is unreasonably high. (2) is similar: $\mathcal{P}(\texttt{NP-C(bill),VP-C(funding)} \mid \texttt{VP,VB,was}) = \mathcal{P}(\texttt{NP-C(bill)} \mid \texttt{VP,VB,was}) * \mathcal{P}(\texttt{VP-C(funding)} \mid \texttt{VP,VB,was})$ is a bad independence assumption.

### Identifying Complements and Adjuncts in the Penn Treebank

We add the "-C" suffix to all non-terminals in training data that satisfy the following conditions:

1. The non-terminal must be: (1) an `NP`, `SBAR`, or `S` whose parent is an `S`; (2) an `NP`, `SBAR`, `S`, or `VP` whose parent is a `VP`; or (3) an `S` whose parent is an `SBAR`.
2. The non-terminal must *not* have one of the following semantic tags: `ADV`, `VOC`, `BNF`, `DIR`, `EXT`, `LOC`, `MNR`, `TMP`, `CLR` or `PRP`. See [Marcus et al. 94] for an explanation of what these tags signify. For example, the `NP` "Last week" in figure 7.2 would have the `TMP` (temporal) tag; and the `SBAR` in "(`SBAR` because the market is down)", would have the `ADV` (adverbial) tag.

In addition, the first child following the head of a prepositional phrase is marked as a complement.

### Probabilities over Subcategorization Frames

Model 1 could be retrained on training data with the enhanced set of non-terminals, and it might learn the lexical properties which distinguish complements and adjuncts ("IBM" vs "week", or "that" vs. "because"). However, it would still suffer from the bad independence assumptions illustrated in figure 7.6. To solve these kinds of problems, the generative process is extended to include a probabilistic choice of left and right subcategorization frames:

1. Choose a head $H$ with probability $\mathcal{P}_H(H \mid P, h)$.
2. Choose left and right subcat frames, $LC$ and $RC$, with probabilities $\mathcal{P}_{lc}(LC \mid P, H, h)$ and $\mathcal{P}_{rc}(RC \mid P, H, h)$. Each subcat frame is a multiset[8] specifying the complements that the head requires in its left or right modifiers.
3. Generate the left and right modifiers with probabilities $\mathcal{P}_l(L_i, l_i \mid H, P, h, distance_l(i-1), LC)$ and $\mathcal{P}_r(R_i, r_i \mid H, P, h, distance_r(i-1), RC)$ respectively. Thus the subcat requirements are added to the conditioning context. As complements are generated they are removed from the appropriate subcat multiset. Most importantly, the probability

[8] A multiset, or bag, is a set which may contain duplicate non-terminal labels.

of generating the STOP symbol will be 0 when the subcat frame is *non-empty*, and the probability of generating a complement will be 0 when it is not in the subcat frame; thus all and only the required complements will be generated.

The probability of the phrase S(bought) -> NP(week) NP-C(IBM) VP(bought) is now:

$$\mathcal{P}_h(\texttt{VP} \mid \texttt{S,bought}) \times \mathcal{P}_{lc}(\{\texttt{NP-C}\} \mid \texttt{S,VP,bought}) \times \mathcal{P}_{rc}(\{\} \mid \texttt{S,VP,bought}) \times$$
$$\mathcal{P}_l(\texttt{NP-C(IBM)} \mid \texttt{S,VP,bought}, \{\texttt{NP-C}\}) \times \mathcal{P}_l(\texttt{NP(week)} \mid \texttt{S,VP,bought}, \{\}) \times$$
$$\mathcal{P}_l(\texttt{STOP} \mid \texttt{S,VP,bought}, \{\}) \times \mathcal{P}_r(\texttt{STOP} \mid \texttt{S,VP,bought}, \{\})$$

Here the head initially decides to take a single NP-C (subject) to its left, and no complements to its right. NP-C(IBM) is immediately generated as the required subject, and NP-C is removed from $LC$, leaving it empty when the next modifier, NP(week) is generated. The incorrect structures in figure 7.6 should now have low probability because $\mathcal{P}_{lc}(\{\texttt{NP-C,NP-C}\} \mid \texttt{S,VP,bought})$ and $\mathcal{P}_{rc}(\{\texttt{NP-C,VP-C}\} \mid \texttt{VP,VB,was})$ are small.

## 7.4 Model 3: Traces and Wh-Movement

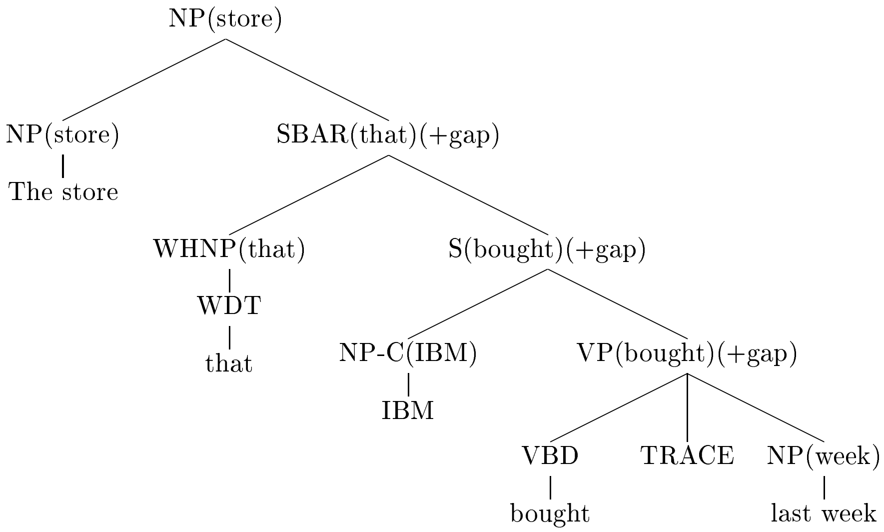
Another obstacle to extracting predicate-argument structure from parse trees is wh-movement. This section describes a probabilistic treatment of extraction from relative clauses. Noun phrases are most often extracted from subject position, object position, or from within PPs:

**Example 1.** *The store (*SBAR *that* TRACE *bought Lotus)*

**Example 2.** *The store (*SBAR *that IBM bought* TRACE*)*

**Example 3.** *The store (*SBAR *that IBM bought Lotus from* TRACE*)*

It might be possible to write rule-based patterns that identify traces in a parse tree. However, we argue again that this task is best integrated into the parser: the task is complex enough to warrant a probabilistic treatment, and integration may help parsing accuracy. A couple of complexities are that modification by an SBAR does not always involve extraction (e.g., "the fact (SBAR that besoboru is played with a ball and a bat)"),




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

Figure 7.7: A *+gap* feature can be added to non-terminals to describe wh-movement. The top-level `NP` initially generates an `SBAR` modifier, but specifies that it must contain an `NP` trace by adding the *+gap* feature. The gap is then passed down through the tree, until it is discharged as a `TRACE` complement to the right of *bought*.

and it is not uncommon for extraction to occur through several constituents, (e.g., "The changes (`SBAR` that he said the government was prepared to make `TRACE`)").

The second reason for an integrated treatment of traces is to improve the parameterization of the model. In particular, the subcategorization probabilities are smeared by extraction. In examples 1, 2 and 3 above 'bought' is a transitive verb; but without knowledge of traces, example 2 in training data will contribute to the probability of 'bought' being an intransitive verb.

Formalisms similar to GPSG [Gazdar et al. 95] handle wh-movement by adding a *gap* feature to each non-terminal in the tree, and propagating gaps through the tree until they are finally discharged as a trace complement (see figure 7.7). In extraction cases the Penn treebank annotation co-indexes a `TRACE` with the `WHNP` head of the `SBAR`, so it is straightforward to add this information to trees in training data.

Given that the LHS of the rule has a gap, there are 3 ways that the gap can be passed down to the RHS:

**Head** The gap is passed to the head of the phrase, as in rule (3) in figure 7.7.

**Left, Right** The gap is passed on recursively to one of the left or right modifiers of the head, or is discharged as a `TRACE` argument to the left/right of the head. In rule (2) it is passed on to a right modifier, the `S` complement. In rule (4) a `TRACE` is generated to the right of the head `VB`.

We specify a parameter $\mathcal{P}_G(G \mid P, h, H)$ where $G$ is either **Head**, **Left** or **Right**. The generative process is extended to choose between these cases after generating the head of the phrase. The rest of the phrase is then generated in different ways depending on how the gap is propagated: In the **Head** case the left and right modifiers are generated as normal. In the **Left, Right** cases a *gap* requirement is added to either the left or right SUBCAT variable. This requirement is fulfilled (and removed from the subcat list) when a trace or a modifier non-terminal which has the *+gap* feature is generated. For example, Rule (2), `SBAR(that)(+gap) -> WHNP(that) S-C(bought)(+gap)`, has probability

$$\mathcal{P}_h(\texttt{WHNP} \mid \texttt{SBAR,that}) \times \mathcal{P}_G(\texttt{Right} \mid \texttt{SBAR,WHNP,that}) \times \mathcal{P}_{LC}(\{\} \mid \texttt{SBAR,WHNP,that}) \times$$
$$\mathcal{P}_{RC}(\{\texttt{S-C}\} \mid \texttt{SBAR,WHNP,that}) \times \mathcal{P}_R(\texttt{S-C(bought)(+gap)} \mid \texttt{SBAR,WHNP,that}, \{\texttt{S-C,+gap}\}) \times$$
$$\mathcal{P}_R(\texttt{STOP} \mid \texttt{SBAR,WHNP,that}, \{\}) \times \mathcal{P}_L(\texttt{STOP} \mid \texttt{SBAR,WHNP,that}, \{\})$$

Rule (4), `VP(bought)(+gap) -> VB(bought) TRACE NP(week)`, has probability

$$\mathcal{P}_h(\texttt{VB} \mid \texttt{VP,bought}) \times \mathcal{P}_G(\texttt{Right} \mid \texttt{VP,bought,VB}) \times \mathcal{P}_{LC}(\{\} \mid \texttt{VP,bought,VB}) \times$$
$$\mathcal{P}_{RC}(\{\texttt{NP-C}\} \mid \texttt{VP,bought,VB}) \times \mathcal{P}_R(\texttt{TRACE} \mid \texttt{VP,bought,VB}, \{\texttt{NP-C, +gap}\}) \times$$
$$\mathcal{P}_R(\texttt{NP(week)} \mid \texttt{VP,bought,VB}, \{\}) \times \mathcal{P}_L(\texttt{STOP} \mid \texttt{VP,bought,VB}, \{\}) \times$$
$$\mathcal{P}_R(\texttt{STOP} \mid \texttt{VP,bought,VB}, \{\})$$

In rule (2) `Right` is chosen, so the $+gap$ requirement is added to $RC$. Generation of `S-C(bought)(+gap)` fulfills both the `S-C` and $+gap$ requirements in $RC$. In rule (4) `Right` is chosen again. Note that generation of `TRACE` satisfies both the `NP-C` and $+gap$ subcat requirements.

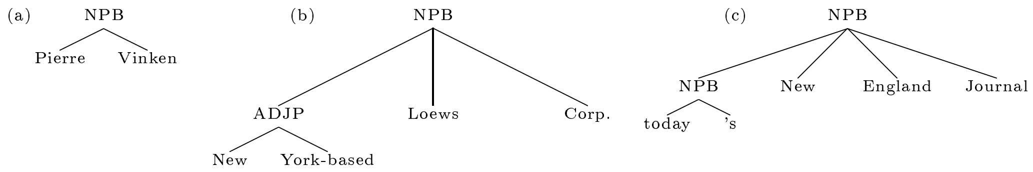



Figure 7.8: Three examples of structures with baseNPs

## 7.5 Special Cases

Sections 7.2 to 7.4 described the basic framework for the parsing models in this chapter. This section describes a handful of special cases: modifications to the basic models that are linguistically motivated, and give an increase in accuracy.

### 7.5.1 Non-recursive NPs

We define non-recursive `NP`s (from here on referred to as baseNPs, and labeled "NPB" rather than "NP") as `NP`s that do not directly dominate an `NP` themselves, unless that `NP` is a possessive `NP` (i.e. it directly dominates a POS-tag "POS"). Figure 7.8 gives some examples. BaseNPs deserve special treatment for three reasons:

- The boundaries of baseNPs are often strongly marked: particularly the start points of baseNPs, which are often marked with a determiner or another distinctive item such as an adjective. Because of this, the probability of generating the `STOP` symbol should be greatly increased when the previous modifier is, for example, a determiner. As they stand, the independence assumptions in the models lose this information. The probability of `NPB(dog) -> DT(the) NN(dog)` would be estimated as

$$\mathcal{P}_h(\texttt{NN} \mid \texttt{NPB,dog}) \times \mathcal{P}_l(\texttt{DT(the)} \mid \texttt{NPB,NN,dog}) \times$$
$$\mathcal{P}_l(\texttt{STOP} \mid \texttt{NPB,NN,dog}) \times \mathcal{P}_r(\texttt{STOP} \mid \texttt{NPB,NN,dog})$$

  In making the independence assumption

$$\mathcal{P}_l(\texttt{STOP} \mid \texttt{DT(the)},\texttt{NPB,NN,dog}) = \mathcal{P}_l(\texttt{STOP} \mid \texttt{NPB,NN,dog})$$

  the model will fail to learn that the `STOP` symbol is very likely to follow a determiner. As a result, the model will assign unreasonably high probability to `NP`s such as [`NP` *yesterday the dog*] in sentences such as [*yesterday the dog barked*].

- The annotation standard in the treebank leaves the internal structure of baseNPs underspecified. For example, both *pet food volume* (where *pet* modifies *food* and *food* modifies *volume*) and *vanilla ice cream* (where both *vanilla* and *ice* modify *cream*) would have the structure `NPB -> NN NN NN`. Because of this, there is no reason to believe that modifiers within `NPB`s are dependent on the head rather than the previous modifier. In fact, if it so happened that a majority of phrases were like *pet food volume*, then conditioning on the previous modifier rather than the head would be preferable.
- In general it is important (in particular for the distance measure to be effective) to have different non-terminal labels for what are effectively different X-bar levels. See section 8.2.2 for further discussion.

For these reasons the following modifications were made to the models:

- The non-terminal label for baseNPs is changed from `NP` to `NPB`. For consistency, whenever an `NP` is seen with no pre or post modifiers, an `NPB` level is added. For example, `[S [NP the dog] [VP barks] ]` would be transformed to `[S [NP [NPB the dog] ] [VP barks ] ]`. These "extra" `NPB`s are removed before scoring the output of the parser against the treebank.
- The independence assumptions are different when the parent non-terminal is an `NPB`. Specifically, equations 7.11 and 7.12 are modified to be

$$\mathcal{P}_l(L_i(l_i) \mid H, P, h, L_1(l_1)...L_{i-1}(l_{i-1})) = \mathcal{P}_l(L_i(l_i) \mid P, L_{i-1}(l_{i-1})) \tag{7.13}$$

$$\mathcal{P}_r(R_i(r_i) \mid H, P, h, R_1(r_1)...R_{i-1}(r_{i-1})) = \mathcal{P}_r(R_i(r_i) \mid P, R_{i-1}(r_{i-1})) \tag{7.14}$$

The modifier and previous-modifier non-terminals are always adjacent, so the distance variable is constant and is omitted. For the purposes of this model, $L_0(l_0)$ and $R_0(r_0)$ are defined to be $H(h)$. The probability of the previous example is now

$$\mathcal{P}_h(\texttt{NN} \mid \texttt{NPB,dog}) \times \mathcal{P}_l(\texttt{DT(the)} \mid \texttt{NPB,NN,dog}) \times$$
$$\mathcal{P}_l(\texttt{STOP} \mid \texttt{NPB,DT,the}) \times \mathcal{P}_r(\texttt{STOP} \mid \texttt{NPB,NN,dog})$$

Presumably $\mathcal{P}_l(\texttt{STOP} \mid \texttt{NPB,DT,the})$ will be very close to 1.

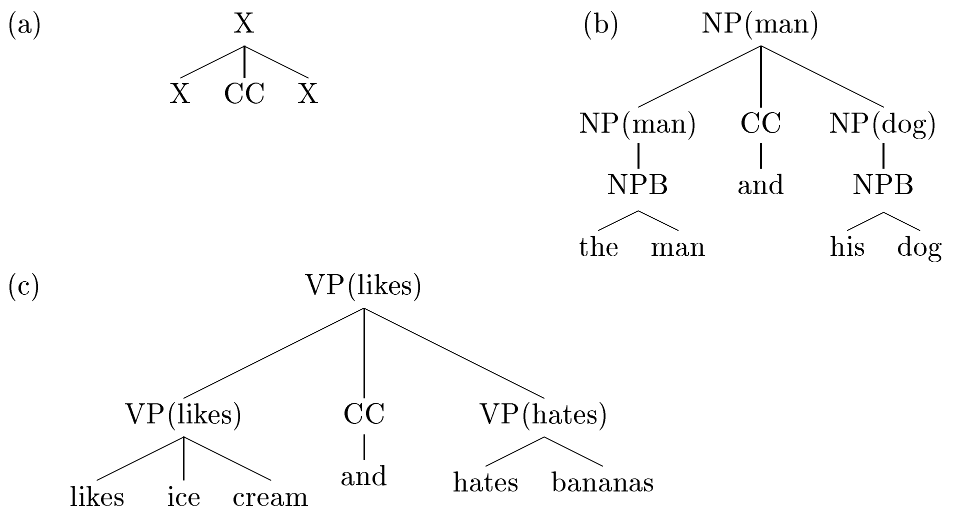



Figure 7.9: (a) the generic way of annotating coordination in the treebank. (b) and (c) show specific examples (with baseNPs added as described in section 7.5.1). Note that the first item of the conjunct is taken as the head of the phrase.

### 7.5.2 Coordination

Coordination constructions are another example where the independence assumptions in the basic models fail badly (at least given the current annotation method in the treebank). Figure 7.9 shows how coordination is annotated in the treebank.[9] To use an example to illustrate the problems, take the rule `NP(man) -> NP(man) CC(and) NP(dog)`, which has probability

$$\mathcal{P}_h(\texttt{NP} \mid \texttt{NP,man}) \times \mathcal{P}_l(\texttt{STOP} \mid \texttt{NP,NP,man}) \times$$
$$\mathcal{P}_r(\texttt{CC(and)} \mid \texttt{NP,NP,man}) \times \mathcal{P}_r(\texttt{NP(dog)} \mid \texttt{NP,NP,man}) \times$$
$$\mathcal{P}_r(\texttt{STOP} \mid \texttt{NP,NP,man})$$

The independence assumptions mean that the model fails to learn that there is always exactly one phrase following the coordinator (`CC`). The basic probability models will give much too high probability to unlikely phrases such as `NP -> NP CC` or `NP -> NP CC NP NP`. For this reason we alter the generative process to allow generation of both the coordinator and the following phrase in one step; instead of just generating a non-terminal at each step, a non-terminal and a binary-valued `coord` flag are generated. `coord`=1 if there is a

[9]See Appendix A for a description of how the head rules treat phrases involving coordination.

coordination relationship. For the preceding example this would give probability

$$\begin{aligned}
&\mathcal{P}_h(\texttt{NP} \mid \texttt{NP,man}) \times \mathcal{P}_l(\texttt{STOP} \mid \texttt{NP,NP,man}) \times \\
&\mathcal{P}_r(\texttt{NP(dog), coord=1} \mid \texttt{NP,NP,man}) \qquad (7.15)\\
&\mathcal{P}_r(\texttt{STOP} \mid \texttt{NP,NP,man}) \times \mathcal{P}_{cc}(\texttt{CC,and} \mid \texttt{NP,NP,NP,man,dog})
\end{aligned}$$

There is now a new type of parameter, $P_{cc}$, for the generation of the coordinator word and POS-tag. The generation of `coord=1` along with `NP(dog)` in the example implicitly requires generation of a coordinator tag/word pair through the $P_{cc}$ parameter. The generation of this tag/word pair is conditioned on the two words in the coordination dependency (`man` and `dog` in the example), and the label on their relationship (`NP,NP,NP` in the example, representing `NP` coordination).

The `coord` flag is implicitly 0 when normal non-terminals are generated, for example the phrase `S(bought) -> NP(week) NP(IBM) VP(bought)` now has probability

$$\begin{aligned}
&\mathcal{P}_h(\texttt{VP} \mid \texttt{S,bought}) \times \mathcal{P}_l(\texttt{NP(IBM),coord=0} \mid \texttt{S,VP,bought}) \times \\
&\mathcal{P}_l(\texttt{NP(week),coord=0} \mid \texttt{S,VP,bought}) \times \mathcal{P}_l(\texttt{STOP} \mid \texttt{S,VP,bought}) \times \\
&\mathcal{P}_r(\texttt{STOP} \mid \texttt{S,VP,bought}) \qquad (7.16)
\end{aligned}$$

### 7.5.3 Punctuation

This section describes our treatment of "punctuation" in the model, where "punctuation" is used to refer to words tagged as a comma or colon. Previous work — the models described in chapter 6 and the earlier version of these generative models described in [Collins 97] — conditioned on punctuation as surface features of the string, treating it quite differently from lexical items. In particular, the model in [Collins 97] failed to generate punctuation, a deficiency of the model. This section describes how punctuation is integrated into the generative models.

Our first step, for consistency, is to raise punctuation as high in the parse trees as possible. Punctuation at the beginning or end of sentences is removed from the training/test

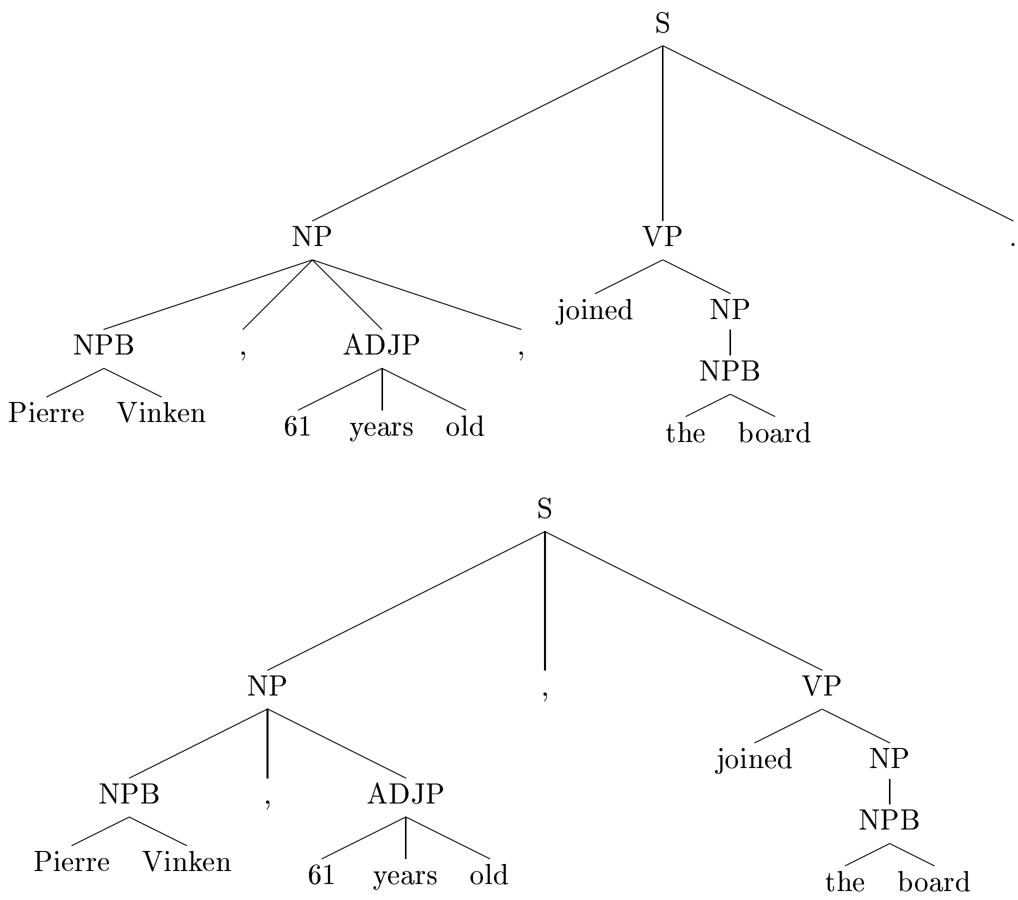



Figure 7.10: A parse tree before and after the punctuation transformations

data altogether. All punctuation items apart from those tagged as comma or colon (i.e. items tagged " " or .) are removed altogether. These transformations mean that punctuation *always* appears between two non-terminals, as opposed to appearing at the end of a phrase. See figure 7.10 for an example.

Punctuation is then treated in a very similar way to coordination: our intuition is that there is a strong dependency between the punctuation mark and the following phrase. Punctuation is therefore generated with the following phrase through a *punc* flag which is similar to the *coord* flag (a binary-valued feature equal to 1 if a punctuation mark is generated with the following phrase).

Under this model, NP(Vinken) -> NPB(Vinken) ,(,) ADJP(old) would have probability

$$
\begin{aligned}
&\mathcal{P}_h(\texttt{NPB} \mid \texttt{NP,Vinken}) \times \mathcal{P}_l(\texttt{STOP} \mid \texttt{NP,NPB,Vinken}) \times \\
&\mathcal{P}_r(\texttt{ADJP(old),coord=0,punc=1} \mid \texttt{NP,NPB,Vinken}) \times \\
&\mathcal{P}_r(\texttt{STOP} \mid \texttt{NP,NPB,bought}) \times \\
&\mathcal{P}_p(\texttt{, ,} \mid \texttt{NP,NPB,ADJP,Pierre,old}) \qquad (7.17)
\end{aligned}
$$

$\mathcal{P}_p$ is a new parameter type for generation of punctuation tag/word pairs. The generation of punc=1 along with ADJP(old) in the example implicitly requires generation of a punctuation tag/word pair through the $P_p$ parameter. The generation of this tag/word pair is conditioned on the two words in the punctuation dependency (Pierre and old in the example), and the label on their relationship (NP,NPB,ADJP in the example.)

### 7.5.4 Sentences with empty (PRO) subjects

Sentences in the treebank occur frequently with PRO subjects which may or may not be controlled: as the treebank annotation currently stands the non-terminal is S whether or not a sentence has an overt subject. This is a problem for the subcategorization probabilities in models 2 and 3 — the probability of having zero subjects, $\mathcal{P}_{lc}(\{\} \mid \texttt{S, VP, } verb)$ will be fairly high because of this. In addition, sentences with and without subjects appear in quite different syntactic environments. For these reasons we modify the non-terminal for sentences without subjects to be SG. See figure 7.11. The resulting model has a cleaner division of subcategorization: $\mathcal{P}_{lc}(\{\texttt{NP-C}\} \mid \texttt{S, VP, } verb) \approx 1$ and $\mathcal{P}_{lc}(\{\texttt{NP-C}\} \mid \texttt{SG, VP, } verb) = 0$. The model will learn probabilistically the environments in which S and SG are likely to appear.

### 7.5.5 The Punctuation Rule

The hard constraint concerning punctuation, originally described in section 6.2.7, is also used in the models of this chapter. It would be preferable to develop a probabilistic analogue of this rule, but we leave this to future research.

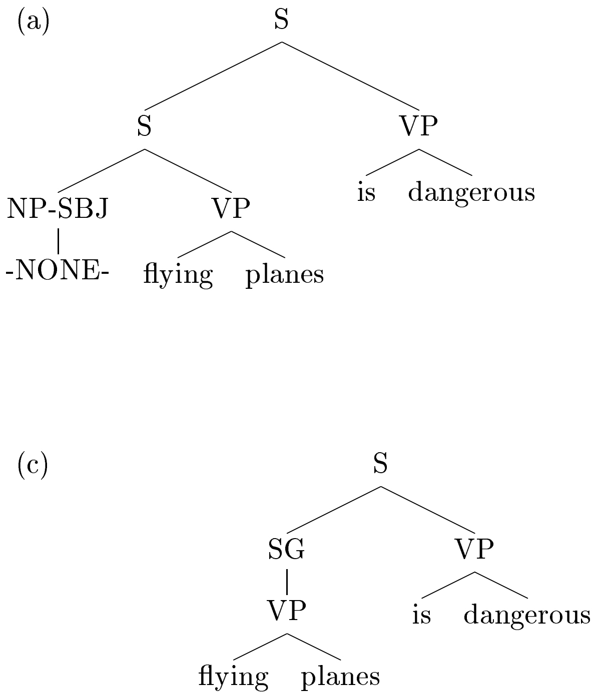



Figure 7.11: (a) the treebank annotates sentences with empty subjects with an empty *-NONE-* element under subject position; (b) in training (and for evaluation), this null element is removed; (c) in models 2 and 3 sentences without subjects are changed to have a non-terminal `SG`.

| Back-off Level | $\mathcal{P}_H(H \mid ...)$ | $\mathcal{P}_G(G \mid ...)$ $\mathcal{P}_{LC}(LC \mid ...)$ $\mathcal{P}_{RC}(RC \mid ...)$ | $\mathcal{P}_{L1}(L_i(lt_i), c, p \mid ...)$ $\mathcal{P}_{R1}(R_i(rt_i), c, p \mid ...)$ | $\mathcal{P}_{L2}(lw_i \mid ...)$ $\mathcal{P}_{R2}(rw_i \mid ...)$ |
|---|---|---|---|---|
| 1 | P, w, t | P, H, w, t | P, H, w, t, $\Delta$, LC | $L_i$, $lt_i$, c, p, P, H, w, t, $\Delta$, LC |
| 2 | P, t | P, H, t | P, H, t, $\Delta$, LC | $L_i$, $lt_i$, c, p, P, H, t, $\Delta$, LC |
| 3 | P | P, H | P, H, $\Delta$, LC | $lt_i$ |

Table 7.1: The conditioning variables for each level of back-off. For example, $\mathcal{P}_H$ estimation interpolates $e_1 = \mathcal{P}_H(H \mid P, w, t)$, $e_2 = \mathcal{P}_H(H \mid P, t)$, and $e_3 = \mathcal{P}_H(H \mid P)$. $\Delta$ is the distance measure.

## 7.6 Practical Issues

### 7.6.1 Parameter Estimation

Table 7.1 shows the various levels of back-off for each type of parameter in the model. Note that we decompose $\mathcal{P}_L(L_i(lw_i, lt_i), c, p \mid P, H, w, t, \Delta, LC)$ (where $lw_i$ and $lt_i$ are the word and POS tag generated with non-terminal $L_i$, $c$ and $p$ are the *coord* and *punc* flags associated with the non-terminal, $\Delta$ is the distance measure) into the product

$$\mathcal{P}_{L1}(L_i(lt_i), c, p \mid P, H, w, t, \Delta, LC) \times \mathcal{P}_{L2}(lw_i \mid L_i, lt_i, c, p, P, H, w, t, \Delta, LC)$$

These two probabilities are then smoothed separately. ([Eisner 96b] originally used POS tags to smooth a generative model in this way.) In each case the final estimate is

$$e = \lambda_1 e_1 + (1 - \lambda_1)(\lambda_2 e_2 + (1 - \lambda_2) e_3)$$

where $e_1$, $e_2$ and $e_3$ are maximum likelihood estimates with the context at levels 1, 2 and 3 in the table, and $\lambda_1$, $\lambda_2$ and $\lambda_3$ are smoothing parameters where $0 \leq \lambda_i \leq 1$. We use the smoothing method described in section 2.9.4: if the more specific estimate is $\frac{n_i}{f_i}$ — that is, $f_i$ is the value of the denominator count — and the number of unique outcomes in the distribution is $u_i$, then

$$\lambda_i = \frac{f_i}{f_i + 5u_i} \tag{7.18}$$

The constant 5 was optimized on the development set, section 0 of the treebank (in practice it was found that any value in the range 2–5 gave a very similar level of performance).

### 7.6.2 Dealing with Unknown Words

All words occurring less than 5 times in training data, and words in test data which have never been seen in training, are replaced with the "UNKNOWN" token. This allows the model to robustly handle the statistics for rare or new words.

### 7.6.3 Part of Speech Tagging

Part of speech tags are generated along with the words in the models, so tagging is fully integrated. In effect, all possible tag sequences are considered. When parsing, the POS tags allowed for each word are limited to those which have been seen in training data for that word (any word/tag pairs not seen in training would give an estimate of zero in the $\mathcal{P}_{L2}$ and $\mathcal{P}_{R2}$ distributions). For unknown words, the output from the tagger described in [Ratnaparkhi 96] is used as the single possible tag for that word. (A method such as the unknown-word model of [Weischedel et al. 93] — which allows multiple possible tags for unknown words, probabilistically generating the word-features of unknown words — would almost certainly be preferable, in that it would fully integrate POS tagging for unknown words into the parsing model.)

## 7.7 The Parsing Algorithm

A chart parser is used to find the maximum probability tree for each sentence. Figure 7.12 shows four basic operations that can be used to create new edges from existing edges in the chart; figure 7.13 sketches the full algorithm, with calls to these four operations. The algorithm is described in detail in Appendix B (Model 1) and Appendix C (Model 2).

### 7.7.1 An Analysis of Parsing Complexity

Appendix D derives upper bounds for the parsing complexity of the algorithm in figure 7.13. Note, however, that the beam search method means that the parsing algorithm is almost certainly more efficient in practice. (In fact, our feeling is that the running time of the algorithm depends much more on the effectiveness of the pruning method, rather than the asymptotic complexity of the algorithm. [Caraballo and Charniak 98, Goodman 97b] both discuss strategies for pruning the search space of a probabilistic parser.) The complexity analysis assumes the following definitions:

- $n$ is the number of words in a sentence.
- $N$ is the number of non-terminals in the grammar, excluding POS tags.
- $T$ is the maximum number of POS tags for any word in the vocabulary.
- $D$ is the number of values for the left and right distance variables.
- $\bar{D}$ is the number of values regarding distance that need to be stored for edges that have their `STOP` probabilities (for the models of this chapter, $\bar{D} = 2$; a flag specifying whether or not an edge contains a verb is all that is needed).
- $L$ is the number of distinct left-subcategorization states seen in conditioning contexts in training data.
- $R$ is the number of distinct right-subcategorization states seen in conditioning contexts in training data.

With these definitions, the complexity of the algorithm is $O(n^5 T^2 N^3 D^2 \bar{D} L R)$.

Appendix D also shows that a tighter bound can be derived if we assume the following definition of the set $\mathcal{X}$:

$$\begin{aligned}\mathcal{X} = \{\langle\text{X,Y,L,R}\rangle \quad | \quad & \langle\text{Parent} = \text{X, Head-label} = \text{Y, left-subcat} = \text{L}\rangle \text{ and} \\ & \langle\text{Parent} = \text{X, Head-label} = \text{Y, right-subcat} = \text{R}\rangle \\ & \text{are both seen as conditioning contexts in training data}\}\end{aligned} \tag{7.19}$$

In the worst case, $\mathcal{X} = N^2 LR$, but in practice $\mathcal{X}$ may be much smaller than $N^2 LR$. An $O(N^2 LR)$ factor within $O(n^5 T^2 N^3 D^2 \bar{D} L R)$ is then reduced to $O(|\mathcal{X}|)$; the overall parsing complexity is $O(n^5 |\mathcal{X}| T^2 N D^2 \bar{D})$.

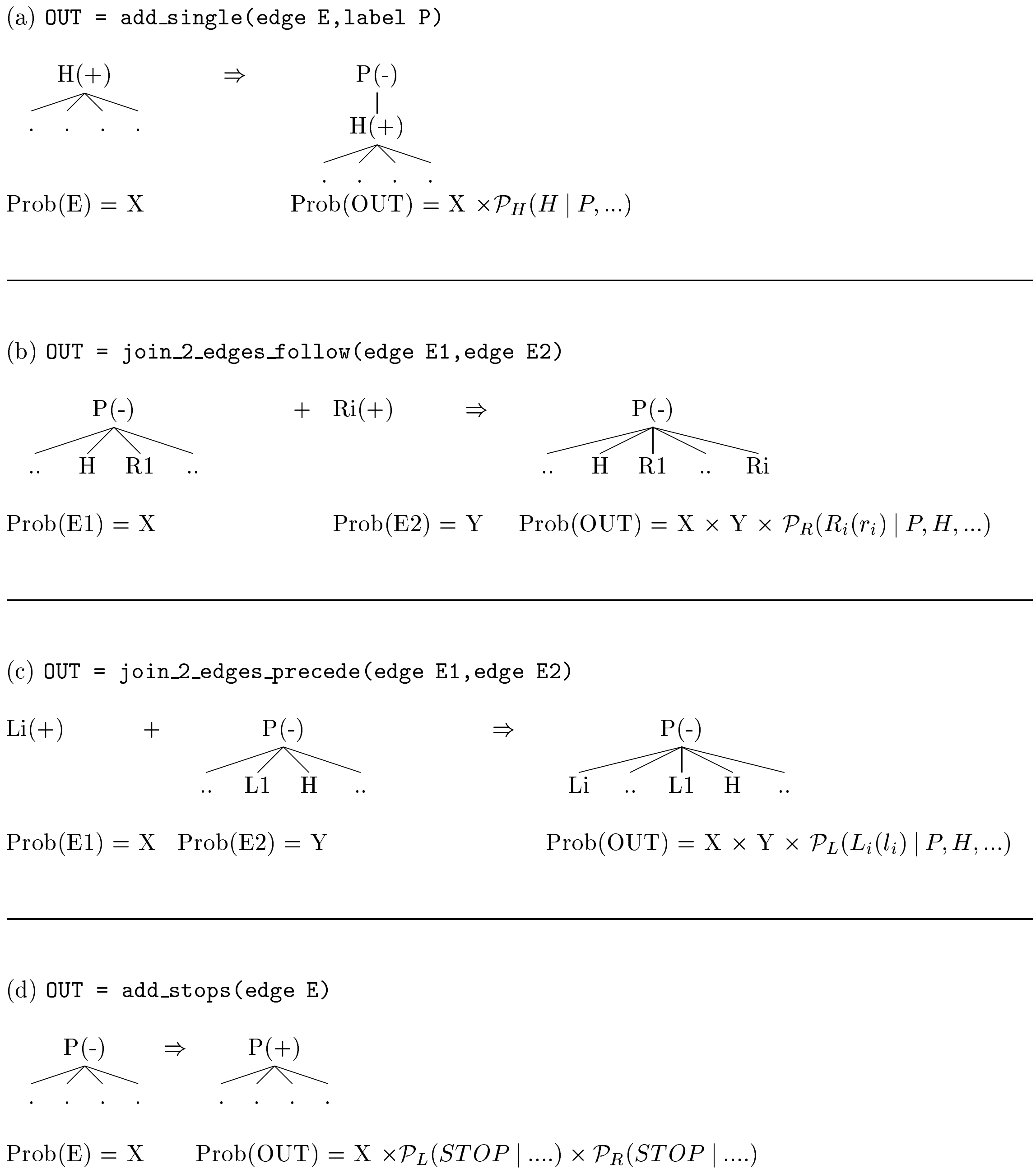



Figure 7.12: Four operations where a new constituent, `OUT`, is formed from either two existing edges, `E1` and `E2`, or a single edge, `E`. Figure 7.13 gives pseudo-code that makes calls to these four operations. (+) means a constituent is complete (i.e. it includes the stop probabilities), (−) means a constituent is incomplete. In (a), a new constituent is started by projecting a complete rule upwards; in (b) and (c), a constituent takes either a right or left modifier; in (d), `STOP` probabilities are added to complete the constituent.

```
edge parse()
{
   //initialize adds edges for all word/tag pairs in the sentence, and
   //adds their projections through add_singles and add_stops,
   //giving the 'base' of the bottom-up parse
   initialize();

   //assume n is the number of words in the sentence
   for span = 2 to n
      for start = 1 to n-span+1
      {
         end = start + span -1;

         //this step combines pairs of edges
         for split = start to end-1
         {
           foreach edge e1 in chart[start,split] such that e1.stop == FALSE
           foreach edge e2 in chart[split+1,end] such that e2.stop == TRUE
              join_2_edges_follow(e1,e2);

           foreach edge e1 in chart[start,split] such that e1.stop == TRUE
           foreach edge e2 in chart[split+1,end] such that e2.stop == FALSE
              join_2_edges_precede(e1,e2);
         }
         //Allow at most MAXU consecutive unary rules
         for i = 1 to MAXU
         {
           //this step adds stop probabilities
           foreach edge e in chart[start,end] such that e.stop == FALSE
              add_stops(e);

           //this step adds unary projections upwards
           foreach edge e in chart[start,end] such that e.stop == TRUE
           foreach non-terminal P in the grammar
              add_single(e,P);
         }
      }

   //assume TOP is the start symbol (must be at the top of the tree)
   X = edge with highest probability spanning words 1...n, with label TOP;
   return X;
}
```

Figure 7.13: A sketch of the parsing algorithm. The functions `join_2_edges_follow`, `join_2_edges_precede`, `add_single` and `add_stops` are illustrated in figure 7.12.

| MODEL | ≤ 40 Words (2245 sentences) | | | | |
|---|---|---|---|---|---|
| | LR | LP | CBs | 0 CBs | ≤ 2 CBs |
| [Magerman 95] | 84.6% | 84.9% | 1.26 | 56.6% | 81.4% |
| Chapter 6 | 85.8% | 86.3% | 1.14 | 59.9% | 83.6% |
| [Goodman 97] | 84.8% | 85.3% | 1.21 | 57.6% | 81.4% |
| [Charniak 97] | 87.5% | 87.4% | 1.00 | 62.1% | 86.1% |
| Model 1 | 87.9% | 88.2% | 0.95 | 65.8% | 86.3% |
| Model 2 | 88.5% | 88.7% | 0.92 | 66.7% | 87.1% |
| Model 3 | 88.6% | 88.7% | 0.90 | 67.1% | 87.4% |

| MODEL | ≤ 100 Words (2416 sentences) | | | | |
|---|---|---|---|---|---|
| | LR | LP | CBs | 0 CBs | ≤ 2 CBs |
| [Magerman 95] | 84.0% | 84.3% | 1.46 | 54.0% | 78.8% |
| Chapter 6 | 85.3% | 85.7% | 1.32 | 57.2% | 80.8% |
| [Charniak 97] | 86.7% | 86.6% | 1.20 | 59.5% | 83.2% |
| [Ratnaparkhi 97] | 86.3% | 87.5% | 1.21 | 60.2% | — |
| Model 1 | 87.5% | 87.7% | 1.09 | 63.4% | 84.1% |
| Model 2 | 88.1% | 88.3% | 1.06 | 64.0% | 85.1% |
| Model 3 | 88.0% | 88.3% | 1.05 | 64.3% | 85.4% |

Table 7.2: Results on Section 23 of the WSJ Treebank. **LR/LP** = labeled recall/precision. **CBs** is the average number of crossing brackets per sentence. **0 CBs,** $\leq$ **2 CBs** are the percentage of sentences with 0 or $\leq 2$ crossing brackets respectively. All the results in this table are for models trained and tested on the same data, using the same evaluation metric. (Note that these results show a slight improvement over those in [Collins 97]; the main model changes were the improved treatment of punctuation (section 7.5.3) together with the addition of the $P_p$ and $P_{cc}$ parameters.)

## 7.8 Results

The parser was trained on sections 02 - 21 of the Wall Street Journal portion of the Penn Treebank [Marcus et al. 93] (approximately 40,000 sentences), and tested on section 23 (2,416 sentences). We use the PARSEVAL measures [Black et al. 91] to compare performance:

**Labeled Precision** $= \frac{number\ of\ correct\ constituents\ in\ proposed\ parse}{number\ of\ constituents\ in\ proposed\ parse}$

**Labeled Recall** $= \frac{number\ of\ correct\ constituents\ in\ proposed\ parse}{number\ of\ constituents\ in\ treebank\ parse}$

**Crossing Brackets** = number of constituents which violate constituent boundaries with a constituent in the treebank parse.

For a constituent to be 'correct' it must span the same set of words (ignoring punctuation, i.e. all tokens tagged as commas, colons or quotes) and have the same label[10] as a constituent in the treebank parse. Table 7.2 shows the results for Models 1, 2 and 3.

The precision/recall of the traces found by Model 3 was 93.8%/90.1% (out of 437 cases in section 23 of the treebank), where three criteria must be met for a trace to be "correct": (1) it must be an argument to the correct head-word; (2) it must be in the correct position in relation to that head word (preceding or following); (3) it must be dominated by the correct non-terminal label. For example, in figure 7.7 the trace is an argument to **bought**, which it **follows**, and it is dominated by a **VP**. Of the 437 cases, 341 were string-vacuous extraction from subject position, recovered with 96.3%/98.8% precision/recall; and 96 were longer distance cases, recovered with 81.4%/59.4% precision/recall [11].

### 7.8.1 A Closer look at the Results

In this section we look more closely at the parser, by evaluating its performance on specific constituents or constructions. The intention is to get a better idea of the parser's strengths and weaknesses. First, table 7.3 has a breakdown of precision and recall by constituent type.

A breakdown of accuracy by constituent type isn't all that informative though, as it fails to capture the idea of *attachment* accuracy. For this reason we also evaluate the parser's precision and recall in recovering dependencies between words: accuracy on different kinds of attachments can then be investigated. A dependency is defined as a triple with the following elements (see figure 7.14 for an example tree and its associated dependencies):

**1) Relation** A ⟨`Parent`, `Head`, `Modifier`, `Direction`⟩ 4-tuple, where the four elements are the parent, head and modifier non-terminals involved in the dependency, and the direction of the dependency (L for left, R for right). For example, ⟨`S`, `VP`, `NP-C`, `L`⟩ would indicate a subject-verb dependency. In coordination cases there is a fifth element

[10] [Magerman 95] collapses `ADVP` and `PRT` to the same label, for comparison we also removed this distinction when calculating scores.

[11] We exclude infinitival relative clauses from these figures, for example "I called a plumber `TRACE` to fix the sink" where 'plumber' is co-indexed with the trace subject of the infinitival. The algorithm scored 41%/18% precision/recall on the 60 cases in section 23 — but infinitival relatives are extremely difficult even for human annotators to distinguish from purpose clauses (in this case, the infinitival could be a purpose clause modifying 'called') (Ann Taylor, p.c.)

| Proportion | Count | Label | Recall | Precision |
|---|---|---|---|---|
| 42.21 | 15146 | NP | 91.15 | 90.26 |
| 19.78 | 7096 | VP | 91.02 | 91.11 |
| 13.00 | 4665 | S | 91.21 | 90.96 |
| 12.83 | 4603 | PP | 86.18 | 85.51 |
| 3.95 | 1419 | SBAR | 87.81 | 88.87 |
| 2.59 | 928 | ADVP | 82.97 | 86.52 |
| 1.63 | 584 | ADJP | 65.41 | 68.95 |
| 1.00 | 360 | WHNP | 95.00 | 98.84 |
| 0.92 | 331 | QP | 84.29 | 78.37 |
| 0.48 | 172 | PRN | 32.56 | 61.54 |
| 0.35 | 126 | PRT | 86.51 | 85.16 |
| 0.31 | 110 | SINV | 83.64 | 88.46 |
| 0.27 | 98 | NX | 12.24 | 66.67 |
| 0.25 | 88 | WHADVP | 95.45 | 97.67 |
| 0.08 | 29 | NAC | 48.28 | 63.64 |
| 0.08 | 28 | FRAG | 21.43 | 46.15 |
| 0.05 | 19 | WHPP | 100.00 | 100.00 |
| 0.04 | 16 | UCP | 25.00 | 28.57 |
| 0.04 | 16 | CONJP | 56.25 | 69.23 |
| 0.04 | 15 | SQ | 53.33 | 66.67 |
| 0.03 | 12 | SBARQ | 66.67 | 88.89 |
| 0.03 | 9 | RRC | 11.11 | 33.33 |
| 0.02 | 7 | LST | 57.14 | 100.00 |
| 0.01 | 3 | X | 0.00 | — |
| 0.01 | 2 | INTJ | 0.00 | — |

Table 7.3: Recall and precision for different constituent types, for section 0 of the treebank with model 2. Label is the non-terminal label; Proportion is the percentage of constituents in the treebank section 0 that have this label; Count is the number of constituents that have this label.

of the tuple, CC: ⟨NP, NP, NP, R CC⟩ would be an instance of NP coordination.

In addition, the relation is "normalized" to some extent. First, all POS tags are replaced with the token TAG: this is so that POS tagging errors do not lead to errors in dependencies[12]. Second, any complement markings on the parent or head non-terminal are removed. For example, ⟨NP-C, NPB, PP, R⟩ is replaced by ⟨NP, NPB, PP, R⟩. This prevents parsing errors where a complement has been mistaken to be an adjunct (or vice versa) leading to more than one dependency error. (In figure 7.14, if the NP *the man* was mistakenly identified as an adjunct then without normalisation this would lead to two dependency errors: both the PP dependency and the verb-object relation would be incorrect. With normalization, only the verb-object relation is incorrect.)

**2) Modifier** The index of the modifier word in the sentence.

**3) Head** The index of the head word in the sentence.

Under this definition, gold-standard and parser-output trees can be converted to sets of dependencies, and precision/recall can be calculated on these dependencies. Dependency accuracies are given for section 0 of the treebank in figure 7.15. Tables 7.4 and 7.5 give a breakdown of the accuracies by dependency type.

Tables 7.6 and 7.7 show the dependency accuracy for 8 sub-types of dependency, which together account for 94% of all dependencies. These sub-types are:

**Complement to a verb: 93.76/92.96 recall/precision.** This type includes any relations of the form ⟨ S VP ** ⟩ where ** is any complement, or ⟨ VP TAG ** ⟩ where ** is any complement except VP-C (i.e., auxiliary-verb—verb dependencies are excluded). The most frequent verb complements, Subject-verb and Object-verb, are recovered with over 95% and 92% precision/recall respectively.

**Other complements: 94.47/94.12 recall/precision.** This type includes any dependencies where the modifier is a complement, and the dependency does not fall into the *complement to a verb* type.

**PP Modification: 82.29/81.51 recall/precision.** Any dependency where the modifier is a PP.

[12] The justification for this is that there is an estimated 3% error rate in the hand-assigned POS tags in the treebank [Ratnaparkhi 96], and we didn't want this noise to contribute to dependency errors.

**Coordination: 61.47/62.20 recall/precision.**

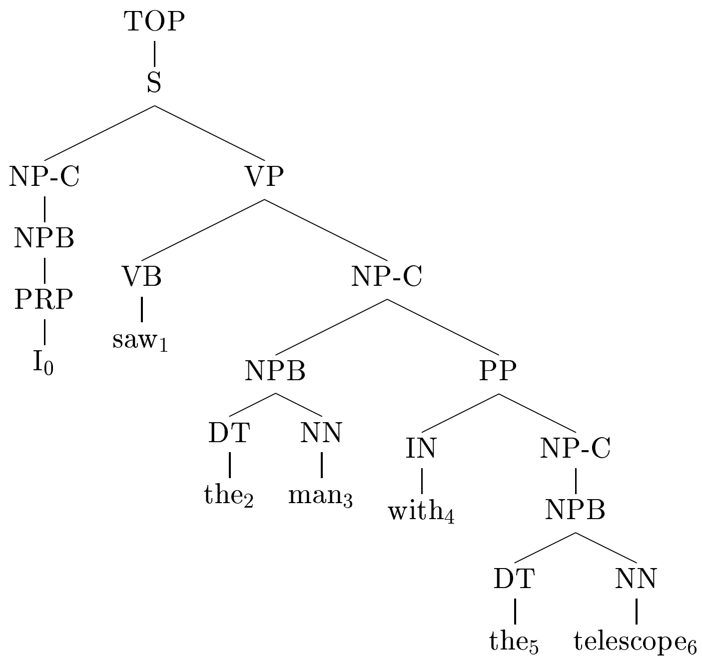
**Modification within baseNPs: 93.20/92.59 recall/precision.** Any dependency where the parent is NPB.

**Modification to NPs: 73.20/75.49 recall/precision.** Any dependency where the parent is NP, the head is NPB, and the modifier is not a PP.

**Sentential Head: 94.99/94.99 recall/precision.** Dependency involving the head-word of the entire sentence.

**Adjunct to a verb: 75.11/78.44 recall/precision.** Any dependency where the parent is VP, the head is TAG, and the modifier is not a PP; or where the parent is S, the head is VP, and the modifier is not a PP.

A conclusion to draw from these accuracies is that the parser is doing very well at recovering the core structure of sentences: complements, sentential heads, and baseNP relationships (NP chunks) are all recovered with over 90% accuracy. The main sources of error are adjuncts. Coordination is especially difficult, most likely because it often involves a dependency between two content-words, leading to very sparse statistics.




| "Raw" Dependencies | | | Normalized Dependencies | | |
|---|---|---|---|---|---|
| Relation | Modifier | Head | Relation | Modifier | Head |
| `S VP NP-C L` | 0 | 1 | `S VP NP-C L` | 0 | 1 |
| `TOP TOP S R` | 1 | -1 | `TOP TOP S R` | 1 | -1 |
| `NPB NN DT L` | 2 | 3 | `NPB TAG TAG L` | 2 | 3 |
| `VP VB NP-C R` | 3 | 1 | `VP TAG NP-C R` | 3 | 1 |
| `NP-C NPB PP R` | 4 | 3 | `NP NPB PP R` | 4 | 3 |
| `NPB NN DT L` | 5 | 6 | `NPB TAG TAG L` | 5 | 6 |
| `PP IN NP-C R` | 6 | 4 | `PP TAG NP-C R` | 6 | 4 |

Figure 7.14: A tree and its associated dependencies. Note that in "normalizing" dependencies, all POS tags are replaced with "TAG", and the NP-C parent in the third relation is replaced with NP.

| Evaluation | Precision | Recall |
|---|---|---|
| No Labels | 91.0% | 90.9% |
| No Complements | 88.5% | 88.5% |
| All | 88.3% | 88.3% |

Figure 7.15: Dependency accuracy on Section 0 of the treebank with Model 2. *No Labels* means that only the dependency needs to be correct, the relation may be wrong; *No Complements* means all complement (-C) markings are stripped before comparing relations; *All* means complement markings are kept on the modifying non-terminal.

| R | CP | P | Count | Relation | Rec | Prec |
|---|---|---|---|---|---|---|
| 1 | 29.65 | 29.65 | 11786 | `NPB TAG TAG L` | 94.60 | 93.46 |
| 2 | 40.55 | 10.90 | 4335 | `PP TAG NP-C R` | 94.72 | 94.04 |
| 3 | 48.72 | 8.17 | 3248 | `S VP NP-C L` | 95.75 | 95.11 |
| 4 | 54.03 | 5.31 | 2112 | `NP NPB PP R` | 84.99 | 84.35 |
| 5 | 59.30 | 5.27 | 2095 | `VP TAG NP-C R` | 92.41 | 92.15 |
| 6 | 64.18 | 4.88 | 1941 | `VP TAG VP-C R` | 97.42 | 97.98 |
| 7 | 68.71 | 4.53 | 1801 | `VP TAG PP R` | 83.62 | 81.14 |
| 8 | 73.13 | 4.42 | 1757 | `TOP TOP S R` | 96.36 | 96.85 |
| 9 | 74.53 | 1.40 | 558 | `VP TAG SBAR-C R` | 94.27 | 93.93 |
| 10 | 75.83 | 1.30 | 518 | `QP TAG TAG R` | 86.49 | 86.65 |
| 11 | 77.08 | 1.25 | 495 | `NP NPB NP R` | 74.34 | 75.72 |
| 12 | 78.28 | 1.20 | 477 | `SBAR TAG S-C R` | 94.55 | 92.04 |
| 13 | 79.48 | 1.20 | 476 | `NP NPB SBAR R` | 79.20 | 79.54 |
| 14 | 80.40 | 0.92 | 367 | `VP TAG ADVP R` | 74.93 | 78.57 |
| 15 | 81.30 | 0.90 | 358 | `NPB TAG NPB L` | 97.49 | 92.82 |
| 16 | 82.18 | 0.88 | 349 | `VP TAG TAG R` | 90.54 | 93.49 |
| 17 | 82.97 | 0.79 | 316 | `VP TAG SG-C R` | 92.41 | 88.22 |
| 18 | 83.70 | 0.73 | 289 | `NP NP NP R CC` | 55.71 | 53.31 |
| 19 | 84.42 | 0.72 | 287 | `S VP PP L` | 90.24 | 81.96 |
| 20 | 85.14 | 0.72 | 286 | `SBAR WHNP SG-C R` | 90.56 | 90.56 |
| 21 | 85.79 | 0.65 | 259 | `VP TAG ADJP R` | 83.78 | 80.37 |
| 22 | 86.43 | 0.64 | 255 | `S VP ADVP L` | 90.98 | 84.67 |
| 23 | 86.95 | 0.52 | 205 | `NP NPB VP R` | 77.56 | 72.60 |
| 24 | 87.45 | 0.50 | 198 | `ADJP TAG TAG L` | 75.76 | 70.09 |
| 25 | 87.93 | 0.48 | 189 | `NPB TAG TAG R` | 74.07 | 75.68 |

Table 7.4: Accuracy of the 25 most frequent dependency types in section 0 of the treebank, as recovered by model 2. R = rank; CP = cumulative percentage; P = percentage; Rec = Recall; Prec = precision.

| R | CP | P | Count | Relation | Rec | Prec |
|---|---|---|---|---|---|---|
| 26 | 88.40 | 0.47 | 187 | VP TAG NP R | 66.31 | 74.70 |
| 27 | 88.85 | 0.45 | 180 | VP TAG SBAR R | 74.44 | 72.43 |
| 28 | 89.29 | 0.44 | 174 | VP VP VP R CC | 74.14 | 72.47 |
| 29 | 89.71 | 0.42 | 167 | NPB TAG ADJP L | 65.27 | 71.24 |
| 30 | 90.11 | 0.40 | 159 | VP TAG SG R | 60.38 | 68.57 |
| 31 | 90.49 | 0.38 | 150 | VP TAG S-C R | 74.67 | 78.32 |
| 32 | 90.81 | 0.32 | 129 | S S S R CC | 72.09 | 69.92 |
| 33 | 91.12 | 0.31 | 125 | PP TAG SG-C R | 94.40 | 89.39 |
| 34 | 91.43 | 0.31 | 124 | QP TAG TAG L | 77.42 | 83.48 |
| 35 | 91.72 | 0.29 | 115 | S VP TAG L | 86.96 | 90.91 |
| 36 | 92.00 | 0.28 | 110 | NPB TAG QP L | 80.91 | 81.65 |
| 37 | 92.27 | 0.27 | 106 | SINV VP NP R | 88.68 | 95.92 |
| 38 | 92.53 | 0.26 | 104 | S VP S-C L | 93.27 | 78.86 |
| 39 | 92.79 | 0.26 | 102 | NP NP NP R | 30.39 | 25.41 |
| 40 | 93.02 | 0.23 | 90 | ADJP TAG PP R | 75.56 | 78.16 |
| 41 | 93.24 | 0.22 | 89 | TOP TOP SINV R | 96.63 | 94.51 |
| 42 | 93.45 | 0.21 | 85 | ADVP TAG TAG L | 74.12 | 73.26 |
| 43 | 93.66 | 0.21 | 83 | SBAR WHADVP S-C R | 97.59 | 98.78 |
| 44 | 93.86 | 0.20 | 81 | S VP SBAR L | 88.89 | 85.71 |
| 45 | 94.06 | 0.20 | 79 | VP TAG ADVP L | 51.90 | 49.40 |
| 46 | 94.24 | 0.18 | 73 | SINV VP S L | 95.89 | 92.11 |
| 47 | 94.40 | 0.16 | 63 | NP NPB SG R | 88.89 | 81.16 |
| 48 | 94.55 | 0.15 | 58 | S VP PRN L | 25.86 | 48.39 |
| 49 | 94.70 | 0.15 | 58 | NX TAG TAG R | 10.34 | 75.00 |
| 50 | 94.83 | 0.13 | 53 | NP NPB PRN R | 45.28 | 60.00 |

Table 7.5: Accuracy of the 26-50'th most frequent dependency types in section 0 of the treebank, as recovered by model 2. R = rank; CP = cumulative percentage; P = percentage; Rec = Recall; Prec = precision

| Type | Sub-type | Description | Count | Recall | Precision |
|---|---|---|---|---|---|
| Complement to a verb | S VP NP-C L | Subject | 3248 | 95.75 | 95.11 |
| | VP TAG NP-C R | Object | 2095 | 92.41 | 92.15 |
| 6495 = 16.3% of all cases | VP TAG SBAR-C R | | 558 | 94.27 | 93.93 |
| | VP TAG SG-C R | | 316 | 92.41 | 88.22 |
| | VP TAG S-C R | | 150 | 74.67 | 78.32 |
| | S VP S-C L | | 104 | 93.27 | 78.86 |
| | S VP SG-C L | | 14 | 78.57 | 68.75 |
| | ... | | | | |
| | TOTAL | | 6495 | 93.76 | 92.96 |
| Other complements | PP TAG NP-C R | | 4335 | 94.72 | 94.04 |
| | VP TAG VP-C R | | 1941 | 97.42 | 97.98 |
| 7473 = 18.8% of all cases | SBAR TAG S-C R | | 477 | 94.55 | 92.04 |
| | SBAR WHNP SG-C R | | 286 | 90.56 | 90.56 |
| | PP TAG SG-C R | | 125 | 94.40 | 89.39 |
| | SBAR WHADVP S-C R | | 83 | 97.59 | 98.78 |
| | PP TAG PP-C R | | 51 | 84.31 | 70.49 |
| | SBAR WHNP S-C R | | 42 | 66.67 | 84.85 |
| | SBAR TAG SG-C R | | 23 | 69.57 | 69.57 |
| | PP TAG S-C R | | 18 | 38.89 | 63.64 |
| | SBAR WHPP S-C R | | 16 | 100.00 | 100.00 |
| | S ADJP NP-C L | | 15 | 46.67 | 46.67 |
| | PP TAG SBAR-C R | | 15 | 100.00 | 88.24 |
| | ... | | | | |
| | TOTAL | | 7473 | 94.47 | 94.12 |
| PP modification | NP NPB PP R | | 2112 | 84.99 | 84.35 |
| | VP TAG PP R | | 1801 | 83.62 | 81.14 |
| 4473 = 11.2% of all cases | S VP PP L | | 287 | 90.24 | 81.96 |
| | ADJP TAG PP R | | 90 | 75.56 | 78.16 |
| | ADVP TAG PP R | | 35 | 68.57 | 52.17 |
| | NP NP PP R | | 23 | 0.00 | 0.00 |
| | PP PP PP L | | 19 | 21.05 | 26.67 |
| | NAC TAG PP R | | 12 | 50.00 | 100.00 |
| | ... | | | | |
| | TOTAL | | 4473 | 82.29 | 81.51 |
| Coordination | NP NP NP R | | 289 | 55.71 | 53.31 |
| | VP VP VP R | | 174 | 74.14 | 72.47 |
| 763 = 1.9% of all cases | S S S R | | 129 | 72.09 | 69.92 |
| | ADJP TAG TAG R | | 28 | 71.43 | 66.67 |
| | VP TAG TAG R | | 25 | 60.00 | 71.43 |
| | NX NX NX R | | 25 | 12.00 | 75.00 |
| | SBAR SBAR SBAR R | | 19 | 78.95 | 83.33 |
| | PP PP PP R | | 14 | 85.71 | 63.16 |
| | ... | | | | |
| | TOTAL | | 763 | 61.47 | 62.20 |

Table 7.6: Accuracy for various types/sub-types of dependency (part 1). Only sub-types occurring more than 10 times are shown.

| Type | Sub-type | Description | Count | Recall | Precision |
|---|---|---|---|---|---|
| Mod'n within BaseNPs | NPB TAG TAG L | | 11786 | 94.60 | 93.46 |
| | NPB TAG NPB L | | 358 | 97.49 | 92.82 |
| 12742 = 29.6% of all cases | NPB TAG TAG R | | 189 | 74.07 | 75.68 |
| | NPB TAG ADJP L | | 167 | 65.27 | 71.24 |
| | NPB TAG QP L | | 110 | 80.91 | 81.65 |
| | NPB TAG NAC L | | 29 | 51.72 | 71.43 |
| | NPB NX TAG L | | 27 | 14.81 | 66.67 |
| | NPB QP TAG L | | 15 | 66.67 | 76.92 |
| | ... | | | | |
| | TOTAL | | 12742 | 93.20 | 92.59 |
| Mod'n to NPs | NP NPB NP R | Appositive | 495 | 74.34 | 75.72 |
| | NP NPB SBAR R | Relative clause | 476 | 79.20 | 79.54 |
| 1418 = 3.6% of all cases | NP NPB VP R | Reduced relative | 205 | 77.56 | 72.60 |
| | NP NPB SG R | | 63 | 88.89 | 81.16 |
| | NP NPB PRN R | | 53 | 45.28 | 60.00 |
| | NP NPB ADVP R | | 48 | 35.42 | 54.84 |
| | NP NPB ADJP R | | 48 | 62.50 | 69.77 |
| | ... | | | | |
| | TOTAL | | 1418 | 73.20 | 75.49 |
| Sentential head | TOP TOP S R | | 1757 | 96.36 | 96.85 |
| | TOP TOP SINV R | | 89 | 96.63 | 94.51 |
| 1917 = 4.8% of all cases | TOP TOP NP R | | 32 | 78.12 | 60.98 |
| | TOP TOP SG R | | 15 | 40.00 | 33.33 |
| | ... | | | | |
| | TOTAL | | 1917 | 94.99 | 94.99 |
| Adjunct to a verb | VP TAG ADVP R | | 367 | 74.93 | 78.57 |
| | VP TAG TAG R | | 349 | 90.54 | 93.49 |
| 2242 = 5.6% of all cases | VP TAG ADJP R | | 259 | 83.78 | 80.37 |
| | S VP ADVP L | | 255 | 90.98 | 84.67 |
| | VP TAG NP R | | 187 | 66.31 | 74.70 |
| | VP TAG SBAR R | | 180 | 74.44 | 72.43 |
| | VP TAG SG R | | 159 | 60.38 | 68.57 |
| | S VP TAG L | | 115 | 86.96 | 90.91 |
| | S VP SBAR L | | 81 | 88.89 | 85.71 |
| | VP TAG ADVP L | | 79 | 51.90 | 49.40 |
| | S VP PRN L | | 58 | 25.86 | 48.39 |
| | S VP NP L | | 45 | 66.67 | 63.83 |
| | S VP SG L | | 28 | 75.00 | 52.50 |
| | VP TAG PRN R | | 27 | 3.70 | 12.50 |
| | VP TAG S R | | 11 | 9.09 | 100.00 |
| | ... | | | | |
| | TOTAL | | 2242 | 75.11 | 78.44 |

Table 7.7: Accuracy for various types/sub-types of dependency (part 2). Only sub-types occurring more than 10 times are shown.

# Chapter 8

# Discussion

This chapter discusses and motivates the models in chapter 7 in more detail. We first consider the distance measure: its effect on accuracy, and cases where it helps to discriminate between rival analyses. We then look at the underlying assumptions that the models make about the tree annotation style. Next, we look more closely at why it is important to break rules down, rather than to simply read a context-free grammar from the treebank. Finally, we consider related work.

## 8.1 More about the Distance Measure

The distance measure, whose implementation was described in section 7.2.3, deserves more discussion and motivation. In this section we consider it from three perspectives: its influence on parsing accuracy; an analysis of distributions in training data that are sensitive to the distance variables; and some examples of sentences where it is useful in discriminating between competing analyses.

### 8.1.1 The Impact of the Distance Measure on Accuracy

Table 8.1 shows the results for models 1 and 2 with and without the adjacency and verb distance measures. It's clear that the distance measure improves accuracy.

| MODEL | A | V | LR | LP | CBs | 0 CBs | ≤ 2 CBs |
|---|---|---|---|---|---|---|---|
| Model 1 | NO | NO | 75.0% | 76.5% | 2.18 | 38.5% | 66.4 |
| Model 1 | YES | NO | 86.6% | 86.7% | 1.22 | 60.9% | 81.8 |
| Model 1 | YES | YES | 87.8% | 88.2% | 1.03 | 63.7% | 84.4 |
| Model 2 | NO | NO | 85.1% | 86.8% | 1.28 | 58.8% | 80.3 |
| Model 2 | YES | NO | 87.7% | 87.8% | 1.10 | 63.8% | 83.2 |
| Model 2 | YES | YES | 88.7% | 89.0% | 0.95 | 65.7% | 85.6 |

Table 8.1: Results on Section 0 of the WSJ Treebank. A = YES, V = YES mean that the adjacency/verb conditions respectively were used in the distance measure. **LR/LP** = labeled recall/precision. **CBs** is the average number of crossing brackets per sentence. **0 CBs,** $\leq$ 2 **CBs** are the percentage of sentences with 0 or $\leq$ 2 crossing brackets respectively.

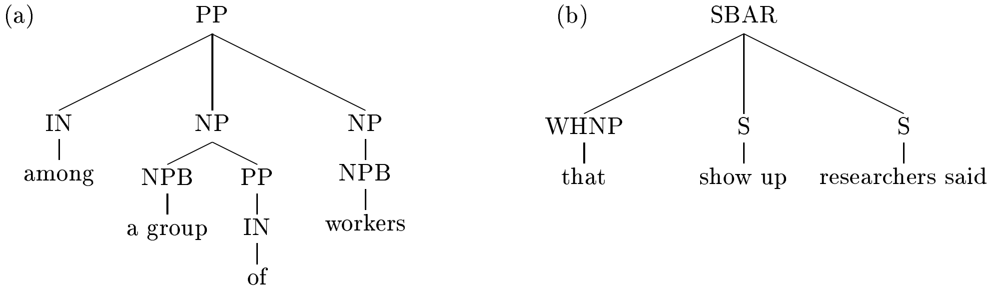



Figure 8.1: Two examples of bad parses produced by model 1 with no distance or subcategorization conditions (Model1(No,No) in table 8.1). In (a) one PP has two complements, the other has none; in (b) the SBAR has two complements. In both examples either the adjacency condition or the subcategorization parameters will correct the errors, so these are examples where the adjacency and subcategorization variables overlap in their utility.

What is most striking is just how *badly* model 1 performs without the distance measure. Looking at the parser's output, the reason for this is that the adjacency condition in the distance measure is approximating subcategorization information. In particular, in phrases such as PPs and SBARs (and, to a lesser extent, in VPs) which almost always take exactly one complement to the right of their head, the adjacency feature encodes this mono-valency through parameters $\mathcal{P}(\text{STOP}|\text{PP/SBAR, adjacent}) = 0$ and $\mathcal{P}(\text{STOP}|\text{PP/SBAR, not adjacent}) = 1$. Figure 8.1.1 shows some particularly bad structures returned by model 1 with no distance variables. (See section 3.3.7 for more discussion of how the distance variable approximates subcategorization.)

The other surprise is that subcategorization *can* be very useful, but that the distance measure has masked this utility. One interpretation in moving from the least parameterized model Model1(No,No) to the fully parameterized model Model2(Yes,Yes) is that the adjacency condition adds around 11% in accuracy; the verb condition adds another 1.5%; and subcategorization finally adds a mere 0.8%. Under this interpretation subcategorization information isn't all that useful (and this was my original assumption, as historically this was the order in which I had added features to the model).

But under another interpretation subcategorization is very useful: in moving from Model1(No,No) to Model2(No,No) we see a 10% improvement due to subcategorization parameters; adjacency then adds a 1.5% improvement; and the verb condition adds a final 1% improvement.

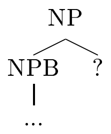
From an engineering point of view, given a choice of whether to add just distance or subcategorization to the model, distance is preferable. But linguistically it is clear that adjacency can only approximate subcategorization, and that subcategorization is more "correct" in some sense. In free word order languages distance may not approximate subcategorization at all well — a complement may appear to either the right or left of the head, confusing the adjacency condition.

### 8.1.2 Frequencies in Training Data

Tables 8.2 and 8.3 show the effect of distance on the distribution of modifiers in two of the most frequent syntactic environments: NP and verb modification. The distribution varies a great deal with distance. Most striking is the way that the probability of STOP increases with increasing distance: from 71% to 89% to 98% in the NP case, from 8% to 60% to 96% in the verb case. Each modifier probability generally decreases with distance. For example, the probability of seeing a PP modifier to an NP decreases from 17.7% to 5.57% to 0.93%.

### 8.1.3 The Adjacency Condition and Right-Branching Structures

Both the adjacency and verb components of the distance measure allow the model to learn a preference for right-branching structures. First, consider the adjacency condition. Table 8.4 shows some examples where right-branching structures are more frequent. Using




| A=TRUE,V=FALSE | | A=FALSE,V=FALSE | | A=FALSE,V=TRUE | |
|---|---|---|---|---|---|
| %age | ? | %age | ? | %age | ? |
| 70.78 | STOP | 88.53 | STOP | 97.65 | STOP |
| 17.7 | PP | 5.57 | PP | 0.93 | PP |
| 3.54 | SBAR | 2.28 | SBAR | 0.55 | SBAR |
| 3.43 | NP | 1.55 | NP | 0.35 | NP |
| 2.22 | VP | 0.92 | VP | 0.22 | VP |
| 0.61 | SG | 0.38 | SG | 0.09 | SG |
| 0.56 | ADJP | 0.26 | PRN | 0.07 | PRN |
| 0.54 | PRN | 0.22 | ADVP | 0.04 | ADJP |
| 0.36 | ADVP | 0.15 | ADJP | 0.03 | ADVP |
| 0.08 | TO | 0.09 | -RRB- | 0.02 | S |
| 0.08 | CONJP | 0.02 | UCP | 0.02 | -RRB- |
| 0.03 | UCP | 0.01 | X | 0.01 | X |
| 0.02 | JJ | 0.01 | RRC | 0.01 | VBG |
| 0.01 | VBN | 0.01 | RB | 0.01 | RB |
| 0.01 | RRC | | | | |
| 0.01 | FRAG | | | | |
| 0.01 | CD | | | | |
| 0.01 | -LRB- | | | | |

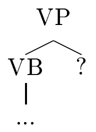
Table 8.2: Distribution of non-terminals generated as post-modifiers to an NP (see tree to the left), at various distances from the head. A=TRUE means the modifier is adjacent to the head, V=TRUE means there is a verb between the head and the modifier. The distributions were calculated from the first 10000 events for each of the distributions in sections 2-21 of the treebank.

the statistics from tables 8.2 and 8.3, the probability of the alternative structures can be calculated — the results are given below. The right-branching structures get higher probability (although this is before the lexical dependency probabilities are multiplied in, so this "prior" preference for right-branching structures can be over-ruled by lexical preferences). If the distance variables were not conditioned on, the product of terms for the two alternatives would be identical, and the model would have no preference for one structure over another.

Probabilities for the two alternative PP structures in table 8.4 (excluding probability terms that are constant across the two structures. A=1 means distance is adjacent, A=0 means not adjacent):

**Right-branching**

$$\mathcal{P}(\text{PP}|\text{NP, NPB, A=1})\mathcal{P}(\text{STOP}|\text{NP, NPB, A=0})$$
$$\mathcal{P}(\text{PP}|\text{NP, NPB, A=1})\mathcal{P}(\text{STOP}|\text{NP, NPB, A=0})$$
$$= 0.177 * 0.8853 * 0.177 * 0.8853 = 0.02455 \qquad (8.1)$$




| A=TRUE,V=FALSE | | A=FALSE,V=FALSE | | A=FALSE,V=TRUE | |
|---|---|---|---|---|---|
| %age | ? | %age | ? | %age | ? |
| 39 | NP-C | 59.87 | STOP | 95.92 | STOP |
| 15.8 | PP | 22.7 | PP | 1.73 | PP |
| 8.43 | SBAR-C | 3.3 | NP-C | 0.92 | SBAR |
| 8.27 | STOP | 3.16 | SG | 0.5 | NP |
| 5.35 | SG-C | 2.71 | ADVP | 0.43 | SG |
| 5.19 | ADVP | 2.65 | SBAR | 0.16 | ADVP |
| 5.1 | ADJP | 1.5 | SBAR-C | 0.14 | SBAR-C |
| 3.24 | S-C | 1.47 | NP | 0.05 | NP-C |
| 2.82 | RB | 1.11 | SG-C | 0.04 | PRN |
| 2.76 | NP | 0.82 | ADJP | 0.02 | S-C |
| 2.28 | PRT | 0.2 | PRN | 0.01 | VBN |
| 0.63 | SBAR | 0.19 | PRT | 0.01 | VB |
| 0.41 | SG | 0.09 | S | 0.01 | UCP |
| 0.16 | VB | 0.06 | S-C | 0.01 | SQ |
| 0.1 | S | 0.06 | -RRB- | 0.01 | S |
| 0.1 | PRN | 0.03 | FRAG | 0.01 | FRAG |
| 0.08 | UCP | 0.02 | -LRB- | 0.01 | ADJP |
| 0.04 | VBZ | 0.01 | X | 0.01 | -RRB- |
| 0.03 | VBN | 0.01 | VBP | 0.01 | -LRB- |
| 0.03 | VBD | 0.01 | VB | | |
| 0.03 | FRAG | 0.01 | UCP | | |
| 0.03 | -LRB- | 0.01 | RB | | |
| 0.02 | VBG | 0.01 | INTJ | | |
| 0.02 | SBARQ | | | | |
| 0.02 | CONJP | | | | |
| 0.01 | X | | | | |
| 0.01 | VBP | | | | |
| 0.01 | RBR | | | | |
| 0.01 | INTJ | | | | |
| 0.01 | DT | | | | |
| 0.01 | -RRB- | | | | |

Table 8.3: Distribution of non-terminals generated as post-modifiers to a verb within a VP (see tree to the left), at various distances from the head. A=TRUE means the modifier is adjacent to the head, V=TRUE means there is a verb between the head and the modifier. The distributions were calculated from the first 10000 events for each of the distributions in sections 2-21. Auxiliary verbs (verbs taking a VP complement to their right) were excluded from these statistics.

**Non Right-branching**

$$\begin{aligned}
&\mathcal{P}(\text{PP}|\text{NP, NPB, A=1})\mathcal{P}(\text{PP}|\text{NP, NPB, A=0}) \\
&\mathcal{P}(\text{STOP}|\text{NP, NPB, A=0})\mathcal{P}(\text{STOP}|\text{NP, NPB, A=1}) \\
&= 0.177 * 0.0557 * 0.8853 * 0.7078 = 0.006178 \qquad (8.2)
\end{aligned}$$

Probabilities for the SBAR case in table 8.4, assuming the SBAR contains a verb (V=0 means modification does not cross a verb, V=1 means it does):

**Right-branching**

$$\begin{aligned}
&\mathcal{P}(\text{PP}|\text{NP,NPB,A=1,V=0})\mathcal{P}(\text{SBAR}|\text{NP,NPB,A=1,V=0}) \\
&\mathcal{P}(\text{STOP}|\text{NP,NPB,A=0,V=1})\mathcal{P}(\text{STOP}|\text{NP,NPB,A=0,V=1}) \\
&= 0.177 * 0.0354 * 0.9765 * 0.9765 = 0.005975 \qquad (8.3)
\end{aligned}$$

**Non Right-branching**

$$\begin{aligned}
&\mathcal{P}(\text{PP}|\text{NP,NPB,A=1})\mathcal{P}(\text{STOP}|\text{NP,NPB,A=1}) \\
&\mathcal{P}(\text{SBAR}|\text{NP,NPB,A=0})\mathcal{P}(\text{STOP}|\text{NP,NPB,A=0,V=1}) \\
&= 0.177 * 0.7078 * 0.0228 * 0.9765 = 0.002789 \qquad (8.4)
\end{aligned}$$

### 8.1.4 The Verb Condition and Right-Branching Structures

Table 8.5 shows some examples where the verb condition is important in differentiating the probability of two structures. In both cases an adjunct can attach either high or low, but the high attachment results in a dependency crossing a verb, and has lower probability.

An alternative to the surface string feature would be a predicate such as "were any of the previous modifiers in $X$", where $X$ is a set of non-terminals that are likely to contain a verb, such as VP, SBAR, S or SG. This would allow the model to handle cases like the first example in table 8.5 correctly. The second example shows why it is preferable to condition on the surface string. In this case the verb is "invisible" to the top level, as it is generated recursively below the NP object.

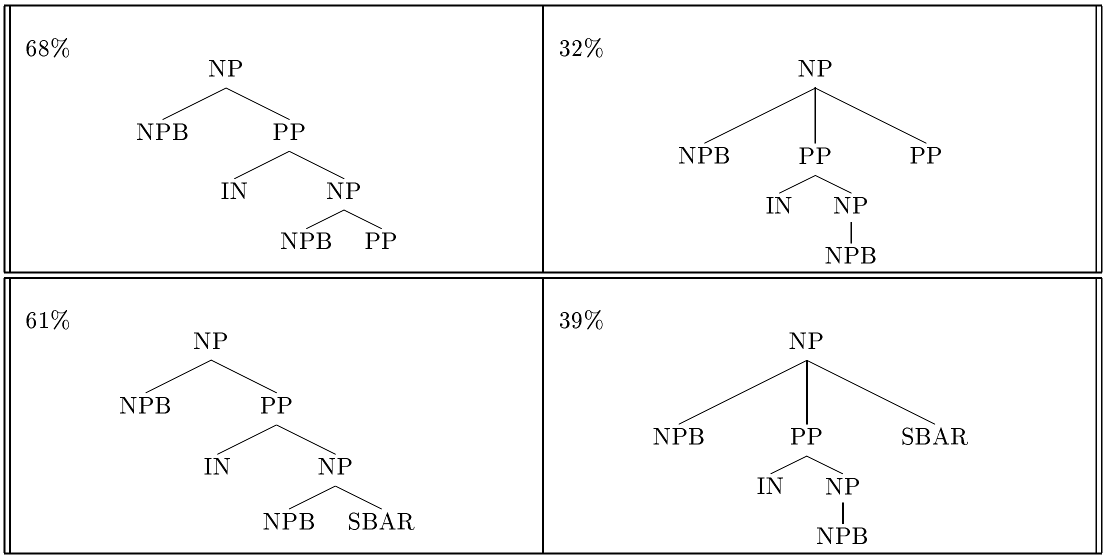



Table 8.4: Some alternative structures for the same surface sequence of chunks (`NPB PP PP` in the first case, `NPB PP SBAR` in the second case), where the adjacency condition distinguishes between the two structures. The percentages are taken from sections 2-21 of the treebank. In both cases right-branching structures are more frequent.

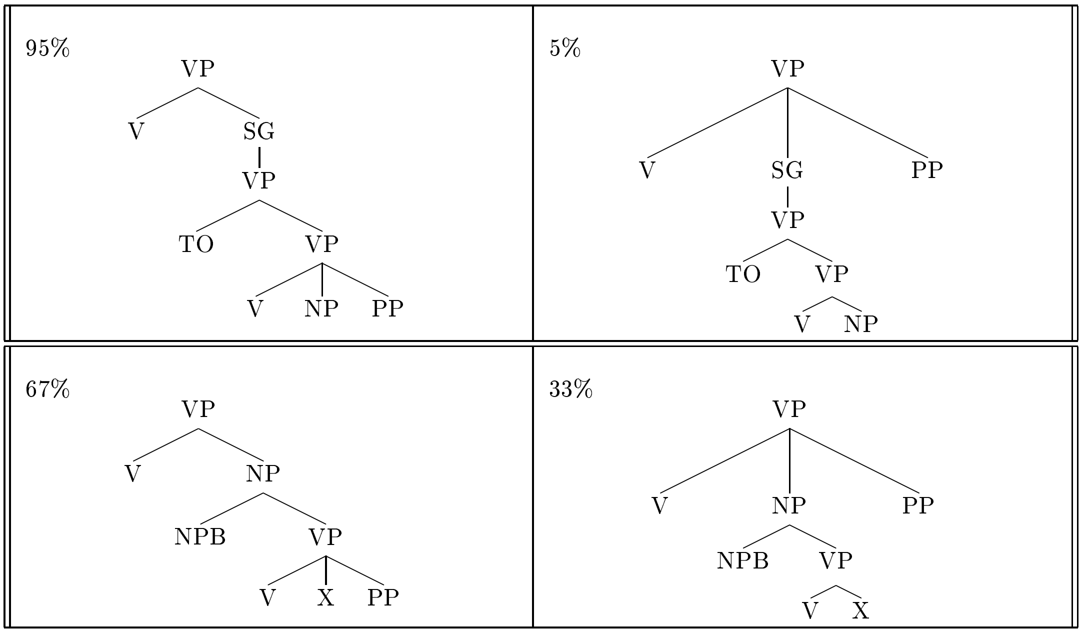



Table 8.5: Some alternative structures for the same surface sequence of chunks, where the verb condition in the distance measure distinguishes between the two structures. In both cases the low-attachment analyses will get higher probability under the model, due to the low probability of generating a PP modifier involving a dependency that crosses a verb. (X stands for any non-terminal.)

### 8.1.5 Structural vs. Semantic Preferences

One hypothesis would be that lexical statistics are *really* what is important in parsing: that arriving at a correct interpretation for a sentence is simply a matter of finding the most semantically plausible analysis, and that the statistics related to lexical dependencies approximate this notion of plausibility. Implicitly, we'd be just as well off (maybe even better off) if statistics were calculated between items at the predicate-argument level, with no reference to structure. The distance preferences under this interpretation are just a way of mitigating sparse data problems: when the lexical statistics are too sparse, then falling back on some structural preference is not ideal, but is at least better than chance. This hypothesis is suggested by the results on PP attachment, which showed that models will perform better given lexical statistics, and that a straight structural preference is merely a fall-back.

But some examples suggest this is not the case: that, in fact, many sentences have several equally semantically plausible analyses, but that structural preferences distinguish strongly between them. Take the following example (from [Pereira and Warren 80]):

**Example 1.** *John was believed to have been shot by Bill*

Surprisingly, this sentence has two analyses — Bill can be the deep subject of either "believed" or "shot". Yet people have a very strong preference for Bill to be doing the shooting, so much so that they may even miss the second analysis. (To see that the dispreferred analysis is semantically quite plausible, consider *Bill believed John to have been shot.*)

As evidence that structural preferences can even over-ride semantic plausibility, take the following example (from [Pinker 94]):

**Example 2.** *Flip said that Squeaky will do the work yesterday*

This sentence is a garden path: the structural preference for "yesterday" to modify the most recent verb is so strong that it is easy to miss the (only) semantically plausible interpretation, paraphrased below as *Flip said yesterday that Squeaky will do the work.*

The model makes the correct predictions in these cases. In example 1, the statistics in table 8.3 show that a PP is 9 times as likely to attach low than high when two verbs are candidate attachment points (the chances of seeing a PP modifier are 15.8% and 1.73% in columns 1 and 3 of the table respectively). In example 2, the probability of seeing an NP (adjunct) modifier to *do* in a non-adjacent but non-verb-crossing environment is 2.11% in sections 2-21 of the treebank (8 out of 379 cases); in contrast the chance of seeing an NP adjunct modifying *said* across a verb is 0.026% (1 out of 3778 cases). The difference is a factor of almost 80.

## 8.2 The Importance of the Choice of Tree Representation

Figures 8.2 and 8.3 show some alternative styles of syntactic annotation. The Penn treebank annotation style tends to leave trees quite flat, typically with one level of structure for each X-bar level; at the other extreme are completely binary-branching representations. The two annotation styles are in some sense equivalent, in that it is easy to define a one-to-one mapping between them. But crucially, two different annotation styles may lead to quite different parsing accuracies for a given model, even if the two representations are equivalent under some one-to-one mapping.

A parsing model does not need to be tied to the annotation style of the treebank on which it is trained. The following procedure can be used to transform trees in both training and test data to a new representation:

1. Transform training data trees to the new representation and train the model.
2. Recover parse trees in the new representation when running the model over test data sentences.
3. Convert the test output back to the treebank representation for scoring purposes.

As long as there is a one-to-one mapping between the treebank and the new representation, nothing is lost in doing this. [Goodman 97] and [Johnson 97] both suggest this strategy: [Goodman 97] converts the treebank to binary branching trees; [Johnson 97] considers conversion to a number of different representations, and discusses how this influences accuracy for non-lexicalized PCFGs.

The models developed in chapter 7 have tacitly assumed the Penn-treebank style of annotation, and will perform badly given other representations (such as binary branching

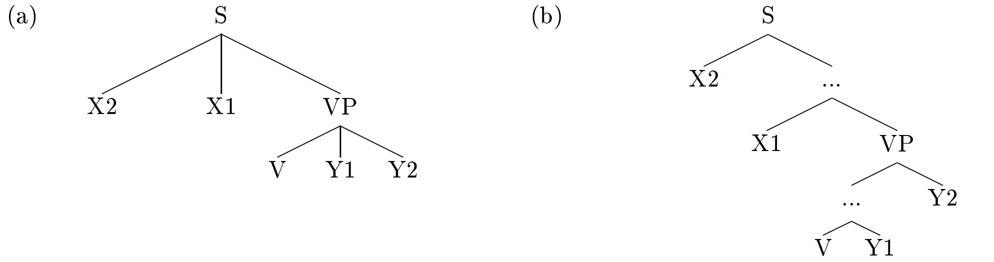



Figure 8.2: Alternative annotation styles for a sentence S with a verb head V, left modifiers X1...X2, and right modifiers Y1...Y2. (a) is the Penn treebank style of analysis: one level of structure for each bar level. (b) is an alternative but equivalent binary branching representation.

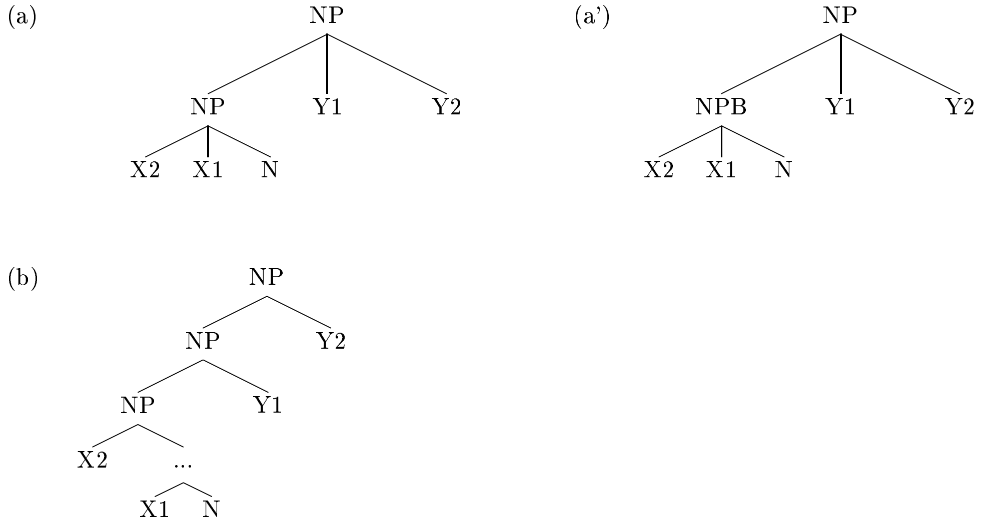



Figure 8.3: Alternative annotation styles for a noun phrase with a noun head N, left modifiers X1...X2, and right modifiers Y1...Y2. (a) is the Penn treebank style of analysis: one level of structure for each bar level, although notice that the non-recursive as well as recursive noun phrases are labeled NP. (b) is an alternative but equivalent binary branching representation. (a') is our modification of the Penn treebank style to differentiate recursive and non-recursive NPs (in some sense NPB is a bar 1 structure, NP is a bar 2 structure).

trees). This section makes this point more explicit: describing exactly what annotation style is suitable for the models of chapter 7, and showing how other annotation styles will cause problems. This dependence on Penn–treebank style annotations does *not* imply that the models are inappropriate for a treebank annotated in a different style — in this case we simply recommend transforming the trees to flat, one-level per X-bar level trees before training the model, as in the 3-step procedure outlined above.

Other models in the literature are also very likely to be sensitive to annotation style. [Charniak 97]'s models will most likely perform quite differently with binary branching trees (for example, his current models will learn that rules such as `VP -> V SG PP` are very rare, but with binary branching structures this context-sensitivity will be lost). The models of [Magerman 95, Ratnaparkhi 97] use contextual predicates which would most likely need to be modified given a different annotation style. [Goodman 97]'s models are the exception, as he already specifies that the treebank should be transformed to his chosen representation, binary branching trees.

### 8.2.1 Representation Affects Structural, not Lexical, Preferences

The alternative representations in figures 8.2 and 8.3 have the same lexical dependencies (providing that the binary-branching structures are centered about the head of the phrase, as in the examples). The difference between the representations involves structural preferences such as the right-branching preferences encoded by the distance measure. A binary branching tree representation makes the distance measure as described in chapter 7 useless as a preference for right branching structures.

To see this, consider the examples in figure 8.4. In each binary branching example the generation of the final modifying PP is "blind" to the distance between it and the head that it modifies. At the top level of the tree it is apparently adjacent to the head; crucially the closer modifier (SG in (a), the other PP in (b)) is hidden lower in the tree structure. So the model will be unable to differentiate generation of the PP in adjacent vs. non-adjacent or non-verb-crossing vs. verb-crossing environments, and the structures in figure 8.4 will get unreasonably high probability.

This does not mean that distance preferences cannot be encoded in a binary branching

PCFG. [Goodman 97] achieves this by adding distance features to the non-terminals. The spirit of this implementation is that the top level rules `VP -> VP PP` and `NP -> NP PP` would be modified to `VP -> VP(+rverb) PP` and `NP -> NP(+rmod) PP`, where (+rverb) means a phrase where the head has a verb in its right-modifiers, and (+rmod) means a phrase that has at least one right-modifier to the head. The model will learn from training data that $\mathcal{P}(\text{VP} \rightarrow \text{VP(+rverb) PP}|\text{VP}) \ll \mathcal{P}(\text{VP} \rightarrow \text{VP(-rverb) PP}|\text{VP})$, i.e., that a prepositional phrase modification is much more likely when it does not cross a verb.

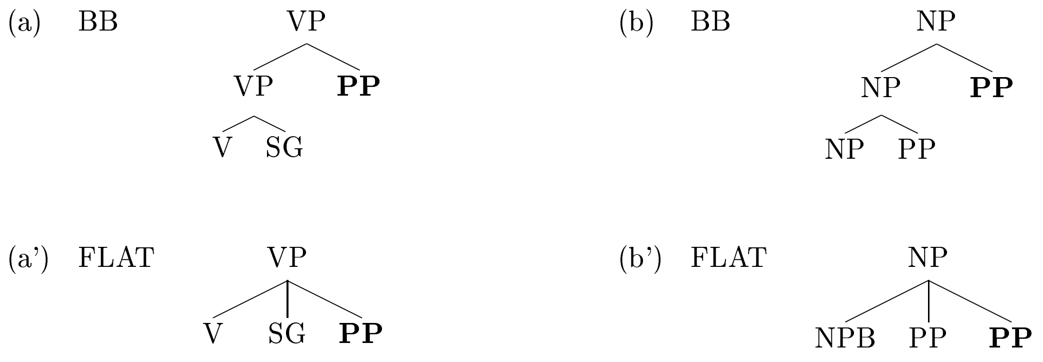



Figure 8.4: BB = binary branching structures; FLAT = Penn treebank style annotations. In each case the binary branching annotation style prevents the model from learning that these structures should receive low probability due to the long distance dependency associated with the final PP (in bold).

### 8.2.2 The Importance of Differentiating Non-recursive vs. Recursive NPs

Figure 8.5 shows the modification to the Penn treebank annotation to relabel baseNPs as NPB. It also illustrates a problem that arises if this distinction is not made: structures such as that in figure 8.5(b) receive high probability even if they are *never* seen in training data. ([Johnson 97] notes that this structure has higher probability than the correct, flat structure, given counts taken from the treebank for a standard PCFG.) The model is fooled by the binary branching style into modeling both PPs as being adjacent to the head of the noun-phrase, so 8.5(b) will get very high probability.

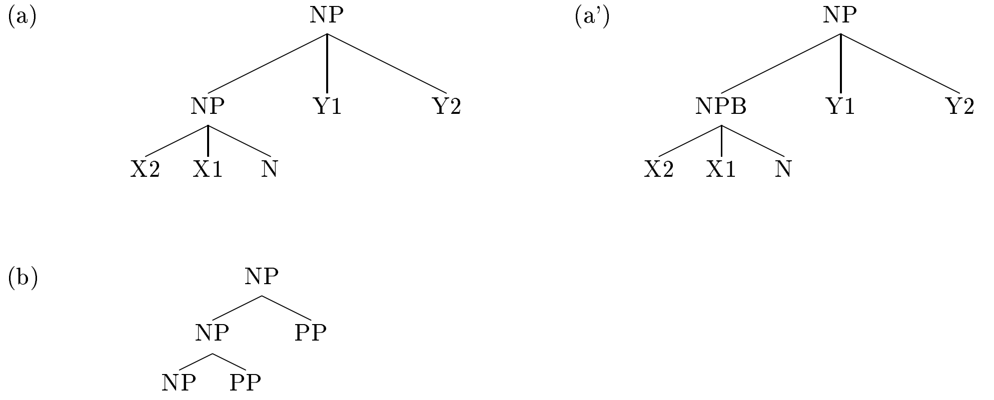



Figure 8.5: (a) The way the Penn treebank annotates NPs. (a') Our modification to the annotation, to differentiate recursive (NP) vs. non-recursive (NPB) noun phrases. (b) a structure that is never seen in training data, but will receive much too high probability from a model trained on trees of style (a).

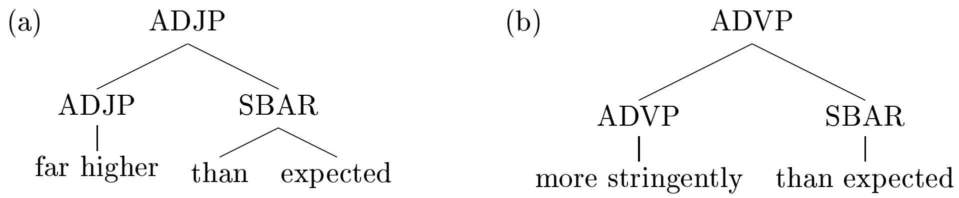



Figure 8.6: Examples of other phrases in the Penn treebank where non-recursive and recursive phrases are not differentiated.

This problem does not only apply to NPs — other phrases such as adjectival phrases (ADJPs) or adverbial phrases (ADVPs) also have non-recursive (bar 1) and recursive (bar 2) levels, which are not differentiated in the Penn treebank. See figure 8.6 for examples. Ideally these cases should be differentiated too: we did not implement this change because it is unlikely to make much difference to accuracy given the relative infrequency of these cases (excluding coordination cases, and looking at the 80,254 instances in sections 2-21 of the Penn treebank where a parent and head non-terminal were the same: 94.5% were the NP case; 2.6% were cases of coordination where a punctuation mark was the coordinator[1]; only 2.9% were similar to those in figure 8.6).

### 8.2.3 Summary

To summarise, the models in this chapter assume:

[1]for example, (S (S John eats apples) ; (S Mary eats bananas))

1. Tree representations are "flat": i.e., one level per X-bar level.
2. Different X-bar levels have different labels (in particular, non-recursive vs. recursive levels are differentiated, at least for the most frequent case of NPs).

## 8.3 The Need to Break Down Rules

The parsing approaches described in the previous two chapters have both concentrated on breaking down context-free rules in the treebank into smaller components. In the previous chapter, rules were initially broken down to bare-bones Markov processes: context dependency was built back up through the distance measure and subcategorization. Even with this additional context, the models are still able to recover rules in test data that have never been seen in training data.

An alternative, proposed in [Charniak 97], is to limit parsing to those context free rules seen in training data. A lexicalized rule is predicted in two steps. First, the whole context-free rule is generated. Second, the lexical items are filled in. The probability of a rule is estimated as[2]:

$$\begin{aligned}
\mathcal{P}(L_n(l_n)...L_1(l_1)H(h)R_1(r_1)...R_m(r_m) \mid P(h)) = \\
\mathcal{P}(L_n...L_1HR_1...R_m) \mid P(h)) \times \\
\prod_{i=1...n} \mathcal{P}_l(l_i \mid L_i, P, h) \times \\
\prod_{j=1...m} \mathcal{P}_r(r_j \mid R_j, P, h) \times
\end{aligned} \tag{8.5}$$

The estimation technique used in [Charniak 97] for the CF rule probabilities interpolates several estimates, the lowest being $\mathcal{P}(L_n...L_1HR_1...R_m) \mid P)$. Any rules not seen in training data will get zero probability under this model. Parse trees in test data will be limited to include rules seen in training.

A problem with this approach is coverage. As shown in this section, many test data sentences will require rules that have not been seen in training. Hence the motivation for

[2]Charniak's model also conditions on the parent of the non-terminal being expanded, we omit this here for brevity.

breaking down rules into smaller components. This section motivates the need to break down rules from four perspectives: first, we discuss how the Penn treebank annotation style leads to a very large number of grammar rules; second, we assess the extent of the coverage problem by looking at rule frequencies in training data; third, we run experiments to assess the impact of the coverage problem on accuracy; fourth, we discuss how breaking rules down may improve estimation as well as coverage.

### 8.3.1 The Penn Treebank Annotation Style Leads to Many Rules

The "flatness" of the Penn treebank annotation style has already been discussed in section 8.2. The flatness of the trees leads to a very large (and constantly growing) number of rules. The prime reason for this is that the number of adjuncts to a head is potentially unlimited, for example there can be any number of PP adjuncts to a head verb. A binary-branching (Chomsky adjunction) grammar can generate an unlimited number of adjuncts with very few rules. For example, the following grammar generates any sequence `VP -> V NP PP*`:

```
VP -> V NP
VP -> VP PP
```

In contrast, the Penn treebank style would create a new rule for each number of PPs seen in training data. The grammar would be

```
VP -> V NP
VP -> V NP PP
VP -> V NP PP PP
VP -> V NP PP PP PP
```
.... and so on

Other adverbial adjuncts, such as adverbial phrases or adverbial SBARs, can also modify a verb several times; and all of these different types of adjuncts can be seen together in the same rule. The result is a combinatorial explosion in the number of rules. To give a flavour of this, here is a random sample of rules that occurred only once in sections 2-21 of the Penn treebank, and were of the format `VP -> VB modifier*`:

```
VP -> VB NP NP NP PRN
VP -> VB NP SBAR PP SG ADVP
VP -> VB NP ADVP ADVP PP PP
VP -> VB RB
VP -> VB NP PP NP SBAR
VP -> VB NP PP SBAR PP
```

It is not only verb phrases that cause this kind of combinatorial explosion: other phrases, in particular non-recursive noun phrases, also contribute a huge number of rules. The next section considers the distributional properties of the rules in more detail.

Note that there is good motivation for the Penn treebank's decision to represent rules in this way, rather than with rules expressing Chomsky adjunction (i.e., a schema where complements and adjuncts are separated, through rule types $\langle \texttt{VP} \rightarrow \texttt{VB}\ \{\texttt{complement}\}* \rangle$ and $\langle \texttt{VP} \rightarrow \texttt{VP}\ \{\texttt{adjunct}\} \rangle$). First, it allowed the argument/adjunct distinction for PP modifiers to verbs to be left undefined: this decision was found to be very difficult for annotators. Second, in the *surface* ordering (as opposed to deep structure), adjuncts are often found closer to the head than complements, thereby yielding structures that fall outside the Chomsky adjunction schema. (For example, a rule such as $\langle \texttt{VP} \rightarrow \texttt{VB NP-C PP SBAR-C} \rangle$ is found very frequently in the Penn treebank; `SBAR` complements nearly always extrapose over adjuncts.)

### 8.3.2 Quantifying the Coverage Problem

In order to quantify the coverage problem, rules were collected from sections 2-21 of the Penn treebank. Punctuation was raised as high as possible in the tree, and the rules did not have complement markings or the distinction between baseNPs and recursive NPs. 939,382 rule tokens were collected in this way; there were 12,409 distinct rule types. We also collected the count for each rule. Table 8.6 shows some statistics for these rules.

A majority of rules in the grammar — 6,765, or 54.5% — occurred only once. These rules account for 0.72% of rules by *token*. That is, if a rule was drawn at random from

| Rule Count | No. of Rules by Type | Percentage by Type | No. of Rules by token | Percentage by token |
|---|---|---|---|---|
| 1 | 6765 | 54.52 | 6765 | 0.72 |
| 2 | 1688 | 13.60 | 3376 | 0.36 |
| 3 | 695 | 5.60 | 2085 | 0.22 |
| 4 | 457 | 3.68 | 1828 | 0.19 |
| 5 | 329 | 2.65 | 1645 | 0.18 |
| 6 ... 10 | 835 | 6.73 | 6430 | 0.68 |
| 11 ... 20 | 496 | 4.00 | 7219 | 0.77 |
| 21 ... 50 | 501 | 4.04 | 15931 | 1.70 |
| 51 ... 100 | 204 | 1.64 | 14507 | 1.54 |
| $> 100$ | 439 | 3.54 | 879596 | 93.64 |

Table 8.6: Statistics for rules taken from sections 2-21 of the treebank, where complement markings were not included on non-terminals.

the 939,382 rule tokens in sections 2-21 of the treebank, there would be a 0.72% chance of that being the only instance of that rule. On the other hand, when drawing a rule at random from the 12,409 rules in the *grammar* induced from those sections, there would be a 54.5% chance of that rule having occurred only once.

The percentage by token of the 1-count rules is an indication of the coverage problem. From this estimate, 0.72% of all rules (or 1 in 139 rules) required in test data would never have been seen in training. It was also found that 15.0% (1 in 6.67) of all sentences had at least one rule that occurred just once. This gives an estimate that roughly 1 in 6.67 sentences in test data will not be covered by a grammar induced from 40,000 sentences in the treebank.

If the complement markings are added to the non-terminals, and the baseNP/non-recursive NP distinction is made, then the coverage problem is made worse. Table 8.7 gives the statistics in this case. 17.1% of all sentences (1 in 5.8 sentences) contained at least one 1-count rule.

### 8.3.3 The Impact of Coverage on Accuracy

Parsing experiments were used to assess the impact of the coverage problem on parsing accuracy. Section 0 of the treebank was parsed with models 1 and 2 as before, but the parse trees were restricted to include rules already seen in training data. Table 8.8 shows

| Rule Count | No. of Rules by Type | Percentage by Type | No. of Rules by token | Percentage by token |
|---|---|---|---|---|
| 1 | 7865 | 55.00 | 7865 | 0.84 |
| 2 | 1918 | 13.41 | 3836 | 0.41 |
| 3 | 815 | 5.70 | 2445 | 0.26 |
| 4 | 528 | 3.69 | 2112 | 0.22 |
| 5 | 377 | 2.64 | 1885 | 0.20 |
| 6 ... 10 | 928 | 6.49 | 7112 | 0.76 |
| 11 ... 20 | 595 | 4.16 | 8748 | 0.93 |
| 21 ... 50 | 552 | 3.86 | 17688 | 1.88 |
| 51 ... 100 | 240 | 1.68 | 16963 | 1.81 |
| $> 100$ | 483 | 3.38 | 870728 | 92.69 |

Table 8.7: Statistics for rules taken from sections 2-21 of the treebank, where complement markings were included on non-terminals.

| MODEL | Accuracy | | | | |
|---|---|---|---|---|---|
| | LR | LP | CBs | 0 CBs | $\leq 2$ CBs |
| Model 1 | 87.9 | 88.3 | 1.02 | 63.9 | 84.4 |
| Model 1 (restricted) | 87.4 | 86.7 | 1.19 | 61.7 | 81.8 |
| Model 2 | 88.8 | 89.0 | 0.94 | 65.9 | 85.6 |
| Model 2 (restricted) | 87.9 | 87.0 | 1.19 | 62.5 | 82.4 |

Table 8.8: Results on Section 0 of the WSJ Treebank. "restricted" means the model is restricted to recovering rules that have been seen in training data. **LR/LP** = labeled recall/precision. **CBs** is the average number of crossing brackets per sentence. **0 CBs,** $\leq 2$ **CBs** are the percentage of sentences with 0 or $\leq 2$ crossing brackets respectively.

the results. Restricting the rules leads to a 0.5/1.6% decrease in recall/precision for model 1, and a 0.9/2.0% decrease for model 2.

### 8.3.4 Breaking Down Rules Improves Estimation

Coverage problems are not the only motivation for breaking down rules. The method may also improve estimation. To illustrate this, consider the rules headed by told, whose counts are shown in table 8.9. Estimating the probability $\mathcal{P}(\text{Rule} \mid \text{VP, told})$ using [Charniak 97]'s method would interpolate two maximum-likelihood estimates:

$$\lambda \mathcal{P}_{ml}(\text{Rule} \mid \text{VP, told}) + (1-\lambda)\mathcal{P}_{ml}(\text{Rule} \mid \text{VP}) \tag{8.6}$$

Estimation interpolates between the specific, lexically sensitive distribution in table 8.9, and the non-lexical estimate based on just the parent non-terminal, `VP`. There are many different rules in the more specific distribution (26 different rule types, out of 147 tokens where told was a VP head); and there are several 1-count rules (11 cases). From these statistics $\lambda$ would have to be relatively low. There's a high chance of a new rule for "told" being required in test data, therefore a reasonable amount of probability mass must be left to the backed-off estimate $\mathcal{P}_{ml}(\text{Rule} \mid \text{VP})$.

This estimation method is missing a crucial generalisation: in spite of there being many different *rules*, the distribution over *subcategorization frames* is much sharper. "told" is seen with only 5 subcategorization frames in training data — the large number of rules is almost entirely due to adjuncts or punctuation appearing after or between complements. The estimation method in model 2 effectively estimates the probability of a rule as

$$\mathcal{P}_{lc}(\text{LC} \mid \text{VP, told}) * \mathcal{P}_{rc}(\text{RC} \mid \text{VP, told}) * \mathcal{P}(\text{Rule} \mid \text{VP, told, LC, RC}) \tag{8.7}$$

The left and right subcategorization frames, `LC` and `RC`, are chosen first. The entire rule is then generated by Markov processes.

Once armed with the $\mathcal{P}_{lc}$ and $\mathcal{P}_{rc}$ parameters, the model has the ability to learn the generalisation that "told" appears with a quite limited, sharp distribution over subcategorization frames. Say these parameters are again estimated through interpolation, for example

$$\lambda \mathcal{P}_{ml}(\text{LC} \mid \text{VP, told}) + (1-\lambda)\mathcal{P}_{ml}(\text{LC} \mid \text{VP}) \tag{8.8}$$

In this case $\lambda$ can be quite high. Only 5 subcategorization frames (as opposed to 26 rule types) have been seen in the 147 cases. The lexically specific distribution $\mathcal{P}_{ml}(\text{LC} \mid \text{VP, told})$ can therefore be quite highly trusted. Relatively little probability mass is left to the backed-off estimate.

In summary, from the distributions in table 8.9, the model should be quite uncertain about what *rules* "told" can appear with. However, it should be relatively certain about the *subcategorization frame*. Introducing subcategorization parameters allows the model to generalise in an important way about rules. We have carefully isolated the "core" of rules — the subcategorization frame — that the model should be certain about.

(a)

| Count | Rule |
|---|---|
| 70 | `VP told -> VBD NP-A SBAR-A` |
| 23 | `VP told -> VBD NP-A` |
| 6 | `VP told -> VBD NP-A SG-A` |
| 5 | `VP told -> VBD NP-A NP SBAR-A` |
| 5 | `VP told -> VBD NP-A : S-A` |
| 4 | `VP told -> VBD NP-A PP SBAR-A` |
| 4 | `VP told -> VBD NP-A PP` |
| 4 | `VP told -> VBD NP-A NP` |
| 3 | `VP told -> VBD NP-A PP NP SBAR-A` |
| 2 | `VP told -> VBD NP-A PP PP` |
| 2 | `VP told -> VBD NP-A NP PP` |
| 2 | `VP told -> VBD NP-A , SBAR-A` |
| 2 | `VP told -> VBD NP-A , S-A` |
| 2 | `VP told -> VBD` |
| 2 | `VP told -> ADVP VBD NP-A SBAR-A` |
| 1 | `VP told -> VBD NP-A SG-A SBAR` |
| 1 | `VP told -> VBD NP-A SBAR-A PP` |
| 1 | `VP told -> VBD NP-A SBAR , PP` |
| 1 | `VP told -> VBD NP-A PP SG-A` |
| 1 | `VP told -> VBD NP-A PP NP` |
| 1 | `VP told -> VBD NP-A PP : S-A` |
| 1 | `VP told -> VBD NP-A NP : S-A` |
| 1 | `VP told -> VBD NP-A ADVP SBAR-A` |
| 1 | `VP told -> VBD NP-A ADVP PP NP` |
| 1 | `VP told -> VBD NP-A ADVP` |
| 1 | `VP told -> VBD NP-A , PRN , SBAR-A` |
| 147 | TOTAL |

(b)

| Count | Subcat Frame |
|---|---|
| 89 | {`NP-A, SBAR-A`} |
| 39 | {`NP-A`} |
| 9 | {`NP-A, S-A`} |
| 8 | {`NP-A, SG-A`} |
| 2 | {} |
| 147 | TOTAL |

Table 8.9: (a) Distribution over rules with "told" as the head (from sections 2-21 of the treebank); (b) Distribution over subcategorization frames with "told" as the head.

We should note that Charniak's method will certainly have some advantages in estimation: it will capture some statistical properties of rules that our independence assumptions will lose (e.g., the distribution over the number of PP adjuncts seen for a particular head).

## 8.4 Comparison to Related Work on Parsing the Penn WSJ Treebank

### 8.4.1 [Charniak 97]

The model described in [Charniak 97] has two types of parameters:

**Lexical Dependency Parameters** Charniak's dependency parameters are similar to the $L2$ parameters of section 7.6.1. Whereas our parameters are

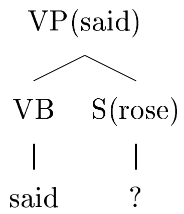
$$\mathcal{P}_{L2}(lw_i \mid L_i, lt_i, c, p, P, H, w, t, \Delta, LC)$$

Charniak's parameters in our notation would be

$$\mathcal{P}_{L2}(lw_i \mid L_i, P, w)$$

For example, the dependency parameter for an NP headed by *profits* being the subject of the verb *rose* would be $P(profits|NP, S, rose)$.

**Rule Parameters** The second type of parameters are associated with context free rules in the tree. As an example, take the S node in the following tree:




This non-terminal could expand with any of the rules $S \rightarrow \beta$ in the grammar. The rule probability is defined as $P(S \rightarrow \beta|rose, S, VP)$. So the rule probability depends on the non-terminal being expanded, its headword, and also its parent.

The next few sections give further explanation of the differences between Charniak's models and the models of this thesis.

#### Additional Features of Charniak's Model

There are some notable additional features of Charniak's model. First, the rule probabilities are conditioned on the parent of the non-terminal being expanded. Our models do not include this information, although distinguishing recursive from non-recursive NPs can be considered a reduced form of this information. (See section 8.2.2 for a discussion of this distinction; the arguments in that section are also motivation for Charniak's choice of conditioning on the parent.)

Second, Charniak uses word-class information to smooth probabilities, and reports a 0.35% improvement from this feature.

Finally, Charniak uses 30 million words of text for unsupervised training. A parser is trained from the treebank, and used to parse this text; statistics are then collected from this machine-parsed text and merged with the treebank statistics to train a second model. This gave a 0.5% improvement in performance.

### The Dependency Parameters of Charniak's Model

While similar to ours, Charniak's dependency parameters are conditioned on less information. Whereas our parameters are $\mathcal{P}_{L2}(lw_i \mid L_i, lt_i, c, p, P, H, w, t, \Delta, LC)$, Charniak's parameters in our notation would be $\mathcal{P}_{L2}(lw_i \mid L_i, P, w)$. The additional information is as follows:

**H** The head non-terminal label (`VP` in the profits/rose example). At first glance this might seem redundant — for example an `S` will usually take a `VP` as its head. However, in some cases the head label can vary, for example an `S` can take another `S` as its head in coordination cases.

$lt_i, t$ The POS tags for the head and modifier words. This allows our model to use POS tags as word-class information. Charniak's model may be missing an important generalization in this respect.

$c$ The coordination flag. This distinguishes, for example, coordination cases from appositives: Charniak's model will have the same parameter — $P(modifier|head, NP, NP)$ — in both of these cases.

$p$, $\Delta$, **LC/RC** The punctuation, distance and subcategorization variables. It is difficult to tell without empirical tests whether these features are important.

#### The Rule Parameters of Charniak's Model

The rule parameters in Charniak's model are effectively decomposed into our $L1$ parameters (section 7.6.1), the head parameters, and — in models 2 and 3 — the subcategorization and gap parameters. This decomposition allows our model to assign probability to rules not seen in training data: see section 8.3 for extensive discussion.

#### Right-Branching Structures in Charniak's Model

Our models have used distance features to encode preferences for right-branching structures. Charniak's model does not represent this information explicitly, but instead learns it implicitly through rule probabilities. For example, for an `NP PP PP` sequence the preference for a right-branching structure is encoded through a much higher probability for the rule `NP` $\rightarrow$ `NP PP` rather than `NP` $\rightarrow$ `NP PP PP`. (Notice that conditioning on the rule's parent is needed to disallow the structure `[NP [NP PP] PP]`; see [Johnson 97] for further discussion.)

This strategy does not encode all of the information in the distance measure. The distance measure effectively penalises rules `NP` $\rightarrow$ `NPB NP PP` where the middle NP contains a verb: in this case the PP modification results in a dependency that crosses a verb. Charniak's model is unable to distinguish cases where the middle NP does/doesn't contain a verb (i.e., the PP modification does/doesn't cross a verb).

### 8.4.2 [Jelinek et al. 94, Magerman 95, Ratnaparkhi 96]

We now make a detailed comparison to the history-based models of [Ratnaparkhi 97] and [Jelinek et al. 94, Magerman 95]. A strength of these models is undoubtedly the powerful estimation techniques that they use: maximum entropy modeling (in [Ratnaparkhi 97]), or decision trees (in [Jelinek et al. 94, Magerman 95]). A weakness, we will argue in this section, is the method of associating parameters with parser moves. We give examples where this leads to the parameters unnecessarily fragmenting the training data in some cases, or ignoring important context in other cases.

We first analyze the model of [Magerman 95] through three common examples of ambiguity: PP attachment, coordination and appositives. In each case a word sequence $S$ has two competing structures — $T_1$ and $T_2$ — with associated decision sequences $\langle d_1, ..., d_n \rangle$ and $\langle e_1, ..., e_m \rangle$ respectively. Thus the probability of the two structures can be written as

$$P(T_1|S) = \prod_{i=1...n} P(d_i|d_1...d_{i-1}, S) \tag{8.9}$$

$$P(T_2|S) = \prod_{i=1...m} P(e_i|e_1...e_{i-1}, S) \tag{8.10}$$

It will be useful to isolate the decision between the two structures to a single probability term. Let the value $j$ be the minimum value of $i$ such that $d_i \neq e_i$. Then we can rewrite the two probabilities:

$$P(T_1|S) = \prod_{i=1...j-1} P(d_i|d_1...d_{i-1}, S) \times P(d_j|d_1...d_{j-1}, S) \times \prod_{i=j+1...n} P(d_i|d_1...d_{i-1}, S) \tag{8.11}$$

$$P(T_2|S) = \prod_{i=1...j-1} P(e_i|e_1...e_{i-1}, S) \times P(e_j|e_1...e_{j-1}, S) \times \prod_{i=j+1...m} P(e_i|e_1...e_{i-1}, S) \tag{8.12}$$

The first thing to note is that $\prod_{i=1...j-1} P(d_i|d_1...d_{i-1}, S) = \prod_{i=1...j-1} P(e_i|e_1...e_{i-1}, S)$, so that these probability terms are irrelevant to the decision between the two structures. We make one additional assumption, that

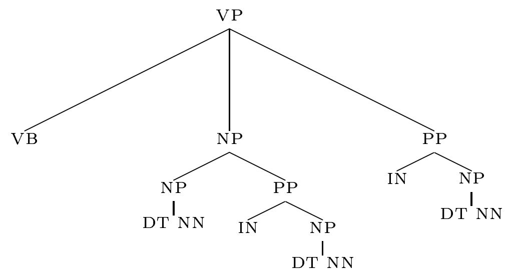
$$\prod_{i=j+1...n} P(d_i|d_1...d_{i-1}, S) \approx \prod_{i=j+1...m} P(e_i|e_1...e_{i-1}, S) \approx 1$$

This is justified for the examples in this section, because once the $j$th decision is made, the following decisions are practically deterministic. (Equivalently, we are assuming that $P(T_1|S) + P(T_2|S) \approx 1$, i.e., that very little probability mass is lost to trees other than $T_1$ or $T_2$.) Given these two equalities, we have isolated the decision between the two structures to the parameters $P(d_j|d_1...d_{j-1}, S)$ and $P(e_j|e_1...e_{j-1}, S)$.

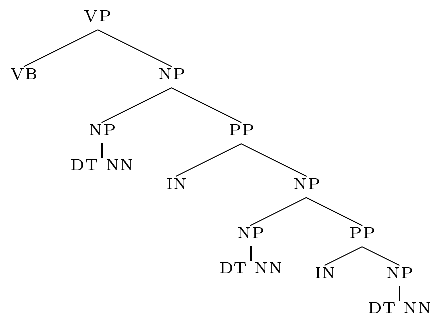
Figure 8.7 shows a case of PP attachment. The first thing to notice is that the PP attachment decision is made before the PP is even built. The decision is linked to the NP preceding the preposition: whether the arc above the NP should go left or right.

The next thing to notice is that at least one important feature, the verb, falls outside of the conditioning context. (The model only considers information up to two constituents preceding or following the location of the decision.) This could be fixed by considering

(a)




(b)




(c)

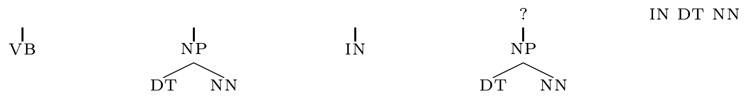



Figure 8.7: (a) and (b) are two candidate structures for the same sequence of words. (c) shows the first decision (labeled “?”) where the two structures differ. The arc above the NP can go either left (for verb attachment of the PP) or right (for noun attachment of the PP).

(a)

(b)

(c)

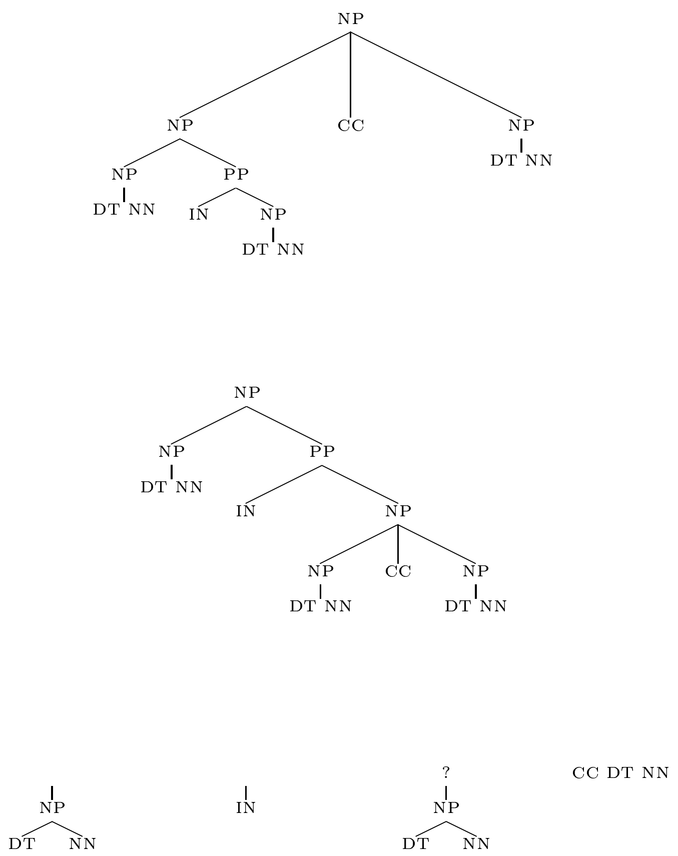


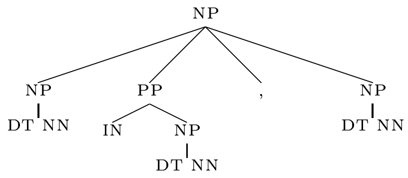

Figure 8.8: (a) and (b) are two candidate structures for the same sequence of words. (c) shows the first decision (labeled "?") where the two structures differ. The arc above the NP can go either left (for high attachment (a) of the coordinated phrase) or right (for low attachment (b) of the coordinated phrase).

additional context: but there is no fixed bound on how far the verb can be from the decision point. Note also that in other cases the method fragments the data in unnecessary ways. Cases where the verb directly precedes the NP, or is one place further to the left, are treated separately.

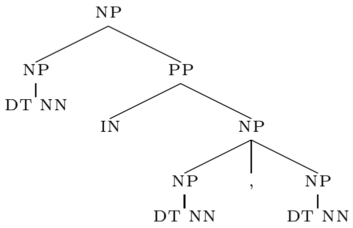
Figure 8.8 shows a similar example, NP coordination ambiguity. Again, the pivotal decision is made in a somewhat counter-intuitive location: at the NP preceding the coordinator. At this point the NP following the coordinator has not been built, and its head noun is not in the contextual window. Figure 8.9 shows an appositive example where the head noun of the appositive NP is not in the contextual window when the decision is made.

These last two examples can be extended to illustrate another problem. The NP after

(a)




(b)




(c)


Figure 8.9: (a) and (b) are two candidate structures for the same sequence of words. (c) shows the first decision (labeled "?") where the two structures differ. The arc above the NP can go either left (for high attachment (a) of the appositive phrase) or right (for noun attachment (b) of the appositive phrase).

the conjunct or comma could be the subject of a following clause. For example, in *John likes Mary and Bill loves Jill* the decision not to coordinate *Mary* and *Bill* is made just after the NP *Mary* is built. At this point, the verb *loves* is outside the contextual window, and the model has no way of telling that *Bill* is the subject of the following clause. The model is assigning probability mass to *globally* implausible structures due to points of *local* ambiguity in the parsing process.

Some of these problems can be fixed by changing the derivation order or the conditioning context. [Ratnaparkhi 97] has an additional chunking stage which means that the head noun does fall within the contextual window for the coordination and appositive cases.

### 8.4.3 [Eisner 96, Eisner 96b]

[Eisner 96] describes a number of probability models with dependency parameters; in addition [Eisner 96b] describes further experiments and newer models. The second paper describes models A, B, C, and D. Model C is quite similar to model 1 of chapter 7 (and pre-dated model 1's original publication in [Collins 97]). This model represents a node in the dependency tree as a tag/word pair. Each word can have 0 or more right or left dependents; probability distributions over different sequences are modeled using 1st order Markov processes. The major differences between model C and our model 1 are as follows:

- There are no non-terminal labels beyond the POS tags at each node (see section 3.3.5 of this thesis for discussion of the importance of non-terminal information.)
- The dependency trees are flat, with each head taking all of its modifies at the same level. Because of this the head parameters $P_H$ of model 1 are unnecessary in model C.
- The Markov process is 1st order, conditioning on the previously generated tag-word pair as well as the head. This may well be additional useful context, and will capture some features of the distance variable: for example, the adjacency condition, or that the `STOP` symbol is likely to follow a heavy phrase such as an `SBAR`.
- [Eisner 96b] describes the addition of distance variables to the model. The distance of a modifier from its head is generated along with the modifier. This improves performance, but leads to a probability model that does not sum to 1.

[Eisner 96b] reports results for dependency accuracy on a test corpus. The best results were 92.6% accuracy, for model D. The model of chapter 6 was found to have identical accuracy when evaluated with the same training and test data (with the modest caveat that the chapter 6 parser was penalised by using machine generated POS tags for this test, [Eisner 96b] used hand-labeled tags).

Model C performs at 90.4% accuracy; it is somewhat surprising that model C performs worse than model D, given that the models have similar features, and that model C is less deficient (i.e., it comes closer to a model that sums to 1). A likely explanation is that the smoothing method's used — holding a count of 3 out for backed-off estimates, and not smoothing at all when the denominator count was greater than 8 — severely impacted performance in model C. (Our experience has been that generative models, which typically have many more possible outcomes for a distribution, require more back-off than conditional models. The count of 3 is very low, and the threshold of 8 is particularly extreme.)

### 8.4.4 [Goodman 97, Goodman 98]

[Goodman 97, Goodman 98] describes *Probabilistic feature grammars* (PFGs). Each non-terminal in the grammar is represented as a set of feature-value pairs; the probability $P(X \rightarrow Y \quad Z|X)$ of a rule $X \rightarrow Y \quad Z$ is decomposed as incremental prediction of the feature values of $Y$ and $Z$. The formalism assumes binary branching rules (without loss of generality: a one-to-one mapping from n-ary rules to binary-branching rules is given).

As [Goodman 98] argues, the formalism has a number of computational advantages. All conditioning variables are encoded through features, allowing a unified account of lexical dependencies, distance features and so on. Parsing can be achieved through standard dynamic programming algorithms, and the formalism allows efficient computation of the inside and outside probabilities for unsupervised training, using the method in [Baker 79].

Having chosen the features on non-terminals, the parameters of the model are specified by first choosing an order for the features being predicted, then making independence assumptions and choosing a back-off order for smoothing. Thus these choices are likely to have a critical effect on the parameterization of the model (as in the Belief Networks example in section 1.5.1) — although [Goodman 98] places little emphasis on these choices.

In particular, the use of binary branching trees may be computationally advantageous, but it may make a linguistically plausible parameterization difficult. [Goodman 98] (page 231), in referring to the work in [Collins 97], claims that "a PFG that is extremely similar could be created". Our feeling is that while it would easy enough to add the same *features* to Goodman's model, it would take considerably more ingenuity to encode the same *parameters* within a PFG. As an example, the models of chapter 7 generate modifiers to a head from inside to outside, while a binary-branching formalism effectively generates modifiers outside to inside; once subcategorization is added this leads to quite different parameters for the two models. Our guess is that these differences, combined with the choice of decomposition in predicting the features on child non-terminals, are likely to be quite important for parsing accuracy.

# Chapter 9

# Future Work

## 9.1 Improving Parsing Accuracy

**Increased Context and Improved Estimation**. The models in chapter 7 have parameters that are conditioned on a limited amount of context, making n-gram estimation methods feasible. The history–based framework, as described in section 7.2.2, allows the model to condition on any structure that has been previously generated. For example, it might make sense to condition upon the previously generated child (as in [Eisner 96]), or to condition upon features of the surface string under previous modifiers (in addition to the two features encoded by the distance measure). Adding many more features of this kind would quickly make n-gram estimation methods infeasible; a natural choice would be to move to more sophisticated estimation methods such as decision trees (as used in [Magerman 95, Jelinek et al. 94]) or maximum-entropy models (as in [Ratnaparkhi 97]). Thus we could combine the insights of the work in this thesis (i.e., an emphasis on using a head-centered with the strengths of the work in [Magerman 95, Jelinek et al. 94, Ratnaparkhi 97] (i.e., estimation techniques that can use very rich feature sets).

**Unsupervised Learning**. Many parsing errors are no doubt caused by poor parameter estimates resulting from sparse data problems. Chapter 5 gave results for PP attachment showing that over 90% accuracy could be achieved on test cases where three or more of the head-words involved had been seen in training data (only around 30% of test data examples met this criterion); but that accuracy dropped to below 72% in cases where the preposition was the sole source of conditioning information. The breakdown of results by dependency type in section 7.8.1 suggested that many parsing errors are due to adjunct placement (e.g., the model recovers PP attachments with 82% precision and recall); our guess is that many of these errors are due to sparse data problems.

Results in [Hindle and Rooth 91], and more recently in [Ratnaparkhi 98], have suggested that unsupervised training for the PP attachment problem can yield results that are competitive with those for supervised models. Two natural questions are then:

1. Given a *very* large amount of training data, could an unsupervised method give improved accuracy over a supervised approach? (e.g., could it approach over 90% accuracy on *all* cases in test data, rather than just 30% of the cases?)
2. Can the methods in [Hindle and Rooth 91, Ratnaparkhi 98] be generalized to other, or all, cases of parsing ambiguity?

## 9.2 Recovering Additional Information

**Predicate Argument Structure**. Chapter 7 stressed the need for additional parse information (namely, complement markings and wh-movement annotations) to facilitate the recovery of predicate–argument structure. There are several other phenomena that have not been treated, for example: wh-movement of phrases other than complement NPs; PRO-control (e.g., recovery of the subject of *leave* in cases like *she promised him to leave*, *she persuaded him to leave*, *she kept him from leaving*); and non-constituent coordination (e.g., *Sam likes and Bill hates peanuts* or *Sam eats peanuts and Bill grapes*). Some of these phenomena (such as PRO control) may be easily recovered through post-processing of the parser's output; others, such as the wh-movement or coordination cases, may be complex enough to require full integration into the statistical model.

Lexicalized grammatical formalisms such as LFG, TAG, CCG and HPSG have all considered these phenomena in some detail; the parameters of this thesis could no doubt be carried over to these theories, with the result that a stochastic treatment of the various phenomena would in some sense come for free. (We actually see this as the main motivation for a move to stochastic versions of these formalisms; our feeling is that moving to these formalisms is unlikely to help parsing accuracy much, given that most of the parsing errors have to do with ambiguity rather than syntactic constraints.)

**"Deep" Syntactic Roles**. A related point is the extraction of "deep" syntactic roles. There is still some remaining ambiguity in the mapping from syntactic trees to semantic roles. Take the following examples. Uncontrolled PRO is usually co-indexed with the previous subject, as in example (1) below, but given sufficient semantic evidence it may be co-indexed with other NPs such as the previous object, as in (2):

(1) Jan was named as president, PRO succeeding Bill.

(2) The company named Jan president, PRO succeeding Bill.

Syntactic roles within nominalizations are quite flexible in their mapping to semantic roles. A possessive NP can be a deep-subject of the noun that it modifies, or a deep-object, or an adjunct such as a temporal modifier:

| | |
|---|---|
| **His** appointment of Clinton | ⇒ deep-subject |
| **His** appointment by Clinton | ⇒ deep-object |
| **today** 's appointment | ⇒ temporal modifier |

Similarly, pre-modifying nouns can take a number of semantic roles

the **Clinton** appointment ⇒ deep subject or object?

the **January** appointment⇒ temporal modifier

The examples suggest that a combination of syntactic biases and semantic plausibility is needed to resolve these kinds of ambiguities. The ideal would be an integrated statistical model that simultaneously recovered syntactic structure and "deep" syntactic roles.

**Information Extraction**. [Miller et al. 98] describe the BBN system for the 7th Message Understanding Conference (MUC-7). One of the tasks was to extract pairs of entities in relationships such as the employer-employee relationship, as in the following example:

John Smith, an IBM spokesman, said ...

⇒ {Employee = *John Smith*, Employer = *IBM*}

[Miller et al. 98] describe an integrated statistical model which simultaneously recovers the syntactic tree along with semantic roles. The syntactic part of the model is quite similar to the models in chapter 7. [Collins and Miller 98] describe a model for extraction of events in the MUC-6 management successions domain (IN is a person coming into a new post, POST is the title of the position), as in the following example:

> Last week Hensley West, 59 years old, was named as president,
> a surprising development.
> $\Rightarrow$ {IN = Hensley West, POST = president}

In [Collins and Miller 98] the model is semantically driven, but syntactically naive; a parsing model that recovered both syntactic and semantic information would again be preferable.

## 9.3 Parsing Languages other than English

This thesis has described work on parsing English; other languages may raise new problems for the approach. Model 1 of chapter 7 was applied to parsing Czech [Hajic et al. 98] in the 1998 Johns Hopkins Summer Workshop on Language Engineering. Czech has some characteristics that make it very different from English, including a very high degree of inflection, and much freer word order. The model recovered dependencies with 80% accuracy (with around 82% accuracy for newswire articles), when trained and tested on the Prague Dependency Treebank (PDT) [Hajic 98]. A major topic for future research is how to deal with the high degree of inflection. This raised two problems:

- The part-of-speech tags encoded multiple fields of information — case, person, number, gender and so on. This meant that the POS tags were potentially a very rich source of disambiguating information, but also that there were a very large number of tags (potentially over 3,000). The question then becomes how to leverage the information in the tags, without running into sparse data problems. A key problem may be how to parameterize probabilities of the form $P(\text{modifier POS}|\text{head POS})$.
- There are a huge number of possible word forms in Czech, leading to a large number of words in test data being unseen in training data: again, sparse data problems are

prevalent. A natural area for future research concerns how to use the (much less sparse) word *stems*; a key problem may be how to parameterize probabilities of the form $P(\text{word-form}|\text{word-stem, POS tag})$.

# Chapter 10

# Conclusions

This thesis has considered how to *parameterize* the statistical parsing problem; in other words, we have considered the following question:

- What linguistic objects (e.g., context-free rules, parse moves etc.) should the model's parameters be associated with? I.e., how should trees be decomposed into smaller events?

Our hypothesis has been that probabilities should be conditioned on lexical heads, and that they should reflect a head's local domain of influence within a parse tree.

**Chapter 3** motivated a series of lexically conditioned parameter types through example trees where the parameters provided useful discriminative information. **Chapter 5** described the first empirical indication of the utility of lexical information: a method that considered the four head-words involved in prepositional phrase attachment decisions correctly classified test examples with over 84% accuracy. **Chapter 6** then described a first attempt at generalizing these results to full parsing. The resulting parser made extensive use of probabilities tied to pairs of words in dependency relationships, and recovered constituent labelings with over 85% precision and recall.

**Chapter 7** described the final parsing models of this thesis. The models use a history-based approach, where a parse tree is represented by the series of decisions made in a top-down, head-centered derivation of the tree. Independence assumptions then follow naturally. The resulting models have parameters that encode the X-bar schema, subcategorization, ordering of complements, placement of adjuncts, lexical dependencies, wh-movement, and preferences for close attachment; all of these preferences are expressed by probabilities conditioned on lexical heads.

Evaluation on Wall Street Journal text (all sentences $\leq$ 100 words in length) showed that the model recovered labeled constituents with 88.3/88.0% precision/recall. The later models have an additional advantage over previous approaches, in that they make the complement/adjunct distinction, and they recover wh-movement annotations. This information should facilitate the mapping to predicate-argument structure.

While this thesis has largely concentrated on representational issues, the learning component should not be underestimated. Once the model structure has been defined, the learning process is remarkable in a couple of respects. First, the model incorporates a vast amount of information. There are almost 780,000 dependency events in training data[1]; if backed-off counts are also considered, the model incorporates information from over 9 million events or sub-events associated with dependencies. The head-projection, subcategorization and other parameters further inflate this number. Second, the model has a tremendous ability to balance the interaction between diverse sources of disambiguating information (e.g., lexical dependencies vs. close-attachment preferences), and to balance fine-grained lexical statistics against coarser statistics based on part-of-speech tags, or more structural information. The mathematical foundations, as outlined in **Chapter 2**, are essential in solving the delicate task of blending these different information sources.

This brings us to our final point. Our guess is that it would be extremely difficult to hand-craft a parser with accuracy that competes with statistical approaches: the volume of information required, together with the complex interactions between the different types of information, quickly becomes overwhelming for a human. On the other hand, we should not expect statistical methods to provide the whole solution. In choosing the model structure — in particular, in making a choice of decomposition in history-based approaches — we have introduced a substantial bias. It is better to ackowledge this bias, and to work with it, rather than to pretend that the learning component will learn all there is to learn.

[1]This count is by token — there are 390,000 distinct dependency *types*.

Most importantly, it is best to embed our prior knowledge of linguistic structure in these modeling choices. Linguistic knowledge, then, should be brought to bear in designing the model structure; learning becomes a problem of estimating parameter values within this structure. Neither part of the problem should be underestimated in its importance.

# Appendix A

# A Description of The Head Rules

This appendix describes the rules used to find heads of constituents in the treebank; i.e., for a context-free rule $\langle \texttt{X} \rightarrow \texttt{Y}_1 \ \ldots \ \texttt{Y}_n \rangle$ these rules decide which of $\langle \texttt{Y}_1 \ \ldots \ \texttt{Y}_n \rangle$ is the head of the rule.

Table A.1 shows the rules used for most constituents in the treebank (there are a couple of exceptions to this table — `NP`s and coordinated phrases — which we will describe soon). The rules are a modified version of those used in the SPATTER parser [Magerman 95][1]. As an example of how the table is used, for rules $\langle \texttt{X} \rightarrow \texttt{Y}_1 \ \ldots \ \texttt{Y}_n \rangle$ where `X` is a `VP`, the algorithm would first search from the left of the sequence $\langle \texttt{Y}_1 \ \ldots \ \texttt{Y}_n \rangle$ for the first $\texttt{Y}_i$ of type `TO`; if no `TO`s were found it would then search for the first $\texttt{Y}_i$ of type `VBD`; if no `VBD`s were found it would search for a `VBP`; and so on. If none of the items on the list were found, the left-most child of the rule ($\texttt{Y}_1$) would be chosen.

### Rules for `NP`s

The rules for `NP`s are slightly different, and are as follows:

- If the last word is tagged POS, return (last-word);
- Else search from right to left for the first child which is an `NN, NNP, NNPS, NNS, NX, POS`, or `JJR`.
- Else search from left to right for the first child which is an `NP`

[1] Thanks to David Magerman for allowing us to use and distribute them.

- Else search from right to left for the first child which is a $\mathtt{\$,}$ ADJP or PRN.
- Else search from right to left for the first child which is a CD.
- Else search from right to left for the first child which is a JJ, JJS, RB or QP.
- Else return the last word.

### Rules for Coordinated Phrases

In coordinated phrases such as $\langle \mathtt{NP} \rightarrow \mathtt{NP\ CC\ NP} \rangle$, the first coordinated child is taken as the head of the phrase. The second coordinated child is then taken to modify this head in a special coordination relationship (e.g., see section 7.5.2 for how the model of chapter 7 deals with coordination). Other modifiers are also taken to modify the first head, in the usual way (e.g., in a rule $\langle \mathtt{NP} \rightarrow \mathtt{NP}_1\ \mathtt{CC}\ \mathtt{NP}_2\ \mathtt{ADJP} \rangle$, the ADJP is taken to modify $\mathtt{NP}_1$).

The head rules are then required to identify rules containing sub-sequences $\langle \mathtt{Y}_i\ \mathtt{Y}_{i+1}\ \mathtt{Y}_{i+2} \rangle$ where: (1) $\mathtt{Y}_i$ is the first head; (2) $\mathtt{Y}_{i+1}$ is a coordinator; and (3) $\mathtt{Y}_{i+2}$ is a modifier to the first head, in a special coordination relationship. This is accomplished in the following steps:

- First run the rule $\langle \mathtt{X} \rightarrow \mathtt{Y}_1\ \ldots\ \mathtt{Y}_n \rangle$ through the usual head-rules, thereby identifying the head of the rule, $\mathtt{Y}_h$.
- If $h < n - 2$, and $\mathtt{Y}_{h+1}$ is the non-terminal CC, then the triple $\langle \mathtt{Y}_h\ \mathtt{Y}_{h+1}\ \mathtt{Y}_{h+2} \rangle$ forms a triple of non-terminals in a coordinating relationship (i.e., they fulfill the three criteria described above).
- Else If $h > 2$, and $\mathtt{Y}_{h-1}$ is the non-terminal CC, then the triple $\langle \mathtt{Y}_{h-2}\ \mathtt{Y}_{h-1}\ \mathtt{Y}_h \rangle$ forms a triple of non-terminals in a coordinating relationship. In this case, the head is modified to be $\mathtt{Y}_{h-2}$.

| Parent Non-terminal | Direction | Priority List |
|---|---|---|
| ADJP | Left | NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB |
| ADVP | Right | RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN |
| CONJP | Right | CC RB IN |
| FRAG | Right | |
| INTJ | Left | |
| LST | Right | LS : |
| NAC | Left | NN NNS NNP NNPS NP NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW |
| PP | Right | IN TO VBG VBN RP FW |
| PRN | Left | |
| PRT | Right | RP |
| QP | Left | $ IN NNS NN JJ RB DT CD NCD QP JJR JJS |
| RRC | Right | VP NP ADVP ADJP PP |
| S | Left | TO IN VP S SBAR ADJP UCP NP |
| SBAR | Left | WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG |
| SBARQ | Left | SQ S SINV SBARQ FRAG |
| SINV | Left | VBZ VBD VBP VB MD VP S SINV ADJP NP |
| SQ | Left | VBZ VBD VBP VB MD VP SQ |
| UCP | Right | |
| VP | Left | TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP |
| WHADJP | Left | CC WRB JJ ADJP |
| WHADVP | Right | CC WRB |
| WHNP | Left | WDT WP WP$ WHADJP WHPP WHNP |
| WHPP | Right | IN TO FW |

Table A.1: The head-rules used by the parser. *Parent* is the non-terminal on the left-hand-side of a rule. *Direction* specifies whether search starts from the left or right end of the rule. *Priority* gives a priority ranking, with priority decreasing when moving down the list.

# Appendix B

# The Parsing Algorithm for Model 1 of Chapter 7

This appendix describes the parsing algorithm for Model 1 in chapter 7. Specifically:

- Section B.1 describes the *edge* data-type. This is central to the parsing algorithm, in that constituents in the chart are represented using the *edge* data-type.
- Section B.2 describes how new edges in the chart are created from existing edges.
- Section B.3 describes how the entire chart is filled: subroutines used for initialization, completion of entire spans of the chart, and finally full parsing.

All of the pseudo-code for the parser is given in figures B.3 to B.11. Figure B.11 gives the highest level routine, **parse()**, which will parse an entire sentence.

Appendix C describes the modifications to the algorithm required for model 2.

## B.1 The *edge* data-type

The central data-type in the chart parser is the *edge* data-type, which holds all the information about a particular constituent in the chart. Table B.1 lists the elements in the *edge* data-type, figure B.1 shows an example constituent and its *edge* representation. Figure B.2 shows how a leaf-node (POS tag/word pair) is represented.

The chart itself is a set of edges. We will assume that edges are indexed by their start

| Variable | Type | Description |
|---|---|---|
| type | int | 0 for a leaf (POS tag with no children), 1 for a non-terminal |
| label | string | the non-terminal label |
| headlabel | string | the non-terminal label of the head-child of the edge |
| headword | string | the head word |
| headtag | string | the part-of-speech tag of the head-word |
| start | int | index of first word in the edge's span |
| end | int | index of last word in the edge's span |
| lc | context | distance features to the left of the head (see below for a description of the **context** data-type). |
| rc | context | distance features to the right of the head |
| stop | boolean | TRUE if the edge has received its stop probabilities |
| prob | double | log probability of the edge |
| children | linked list | list of the children of the edge (in left to right order) |

Table B.1: Variables in the *edge* data-type

| Variable | Type | Description |
|---|---|---|
| adj | boolean | (adjacency) TRUE if the head has taken no modifiers |
| verb | boolean | TRUE if one of the modifiers to the head dominates a verb |

Table B.2: Variables in the *context* data-type

point (first word in their span), their end point (last word in their span), and their non-terminal label. So $chart[start, end, label]$ is a set of all edges in the chart spanning words $start$ to $end$ inclusive, with non-terminal $label$.

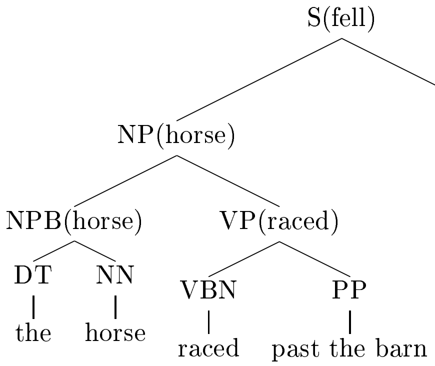
There are two important functions associated with *edge*, pseudo-code is shown in figure B.3. $edges\_equal(edge\ \ e1, edge\ \ e2)$ returns TRUE if edges $e1$ and $e2$ are the same for the purposes of the dynamic programming algorithm. $add\_edge(edge\ \ e, int\ \ start, int\ \ end)$ adds $e$ to the set $chart[start, end, e.label]$ if it passes the dynamic programming conditions.

## B.2 Subroutines that Create New Edges

There are four operations that create new edges:

**join_2_edges_follow(e1,e2)** Figure B.4. Takes as input two edges, $e1$ and $e2$, and forms a new edge, $e3$. $e1$ and $e2$ are adjacent in the chart, with $e2$ following $e1$[1]. $e2$ is a modifier to $e1$, so $e3$ gets its headword from $e1$.

[1] i.e., if $e1$ spans words $i...j$, $e2$ spans words $k...l$, then $k = j + 1$.




| Variable | Value |
|---|---|
| type | 1 |
| label | **S** |
| headlabel | **VP** |
| headword | *fell* |
| headtag | **VB** |
| lc.adj | FALSE |
| lc.verb | TRUE |
| rc.adj | TRUE |
| rc.verb | FALSE |

Figure B.1: An example constituent, and the values for its *edge* representation


| Variable | Value |
|---|---|
| type | 0 |
| label | **VBN** |
| headlabel | — |
| headword | *raced* |
| headtag | **VBN** |
| stop | TRUE |
| lc.adj | TRUE |
| lc.verb | FALSE |
| rc.adj | TRUE |
| rc.verb | FALSE |

Figure B.2: An example leaf-node constituent, and the values for its *edge* representation.

```
boolean edges_equal(edge e1,edge e2)
{
   if(e1.type      != e2.type      OR
      e1.label     != e2.label     OR
      e1.headlabel != e2.headlabel OR
      e1.headword  != e2.headword  OR
      e1.headtag   != e2.headtag   OR
      e1.lc        != e2.lc        OR
      e1.rc        != e2.rc        OR
      e1.stop      != e2.stop )

          return FALSE;

   return TRUE;
}

void add_edge(edge e, int start, int end)
{
   foreach edge x in chart[start,end,e.label]
      if(equal_edge(e,x))
      {
         if(e.prob > x.prob)
             replace x with e;

         return;
      }

   add e to the set chart[start,end,e.label]
}
```

Figure B.3: Two functions associated with the *edge* data-type. `edges_equal` compares two edges for the purposes of the dynamic programming algorithm. `add_edge` adds an edge to the chart if it passes the dynamic programming condition.

**join_2_edges_precede(e1,e2)** Figure B.5. Same as **join_2_edges_follow**, except $e1$ is a modifier to $e2$, so $e3$ gets its headword from $e2$.

**add_singles(e)** Figure B.6. Takes as input a single edge, $e$, and adds all possible edges containing a unary rule with $e$ as the head-child.

**add_stops(e)** Figure B.7. Takes as input a single edge, $e$, which has not yet included stop probabilities. Creates a new edge in the chart with the stop probabilities added.

## B.3 Subroutines that Complete Entire Spans of the Chart

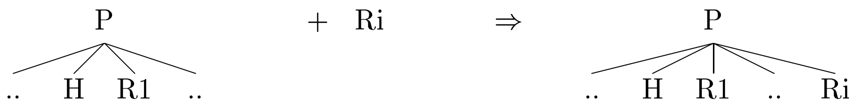
**add_singles_stops(start,end)** Figure B.8. This creates all new edges for span start to end in the chart which are created through **add_singles** or **add_stops**. It is a subroutine that is used by both **initialize** and **complete**.

**initialize()** Figure B.9. This initializes the chart, adding an edge for each possible (word,tag) pair.

**complete(int start,int end)** Figure B.10. This adds all edges to the chart that span words start to end inclusive. (Assumes that $end - start > 0$, as **initialize** has already created all the single word constituents.)

**parse()** Figure B.11. Parses the entire sentence.

(a)



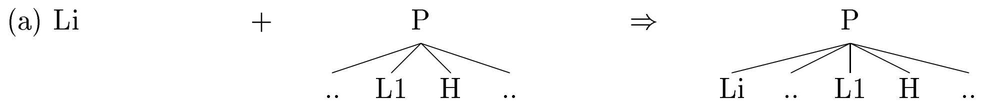

Prob = X      Prob = Y      Prob = X $\times$ Y $\times$ $\mathcal{P}_R(R_i(r_i) \mid P, H, ...)$

edge e1      edge e2      edge e3

(b)

```
void join_2_edges_follow(edge e1,edge e2)
{
   edge e3;

   e3.type      = 1;
   e3.label     = e1.label;
   e3.headlabel = e1.headlabel;
   e3.headword  = e1.headword;
   e3.headtag   = e1.headtag;
   e3.stop      = FALSE;
   e3.lc        = e1.lc;
   e3.rc.adj    = FALSE;
   e3.rc.verb   = e1.rc.verb OR
                  e2.lc.verb OR
                  e2.rc.verb OR
                  e2.headtag is a verb;
   e3.start     = e1.start;
   e3.end       = e2.end;
   e3.children  = e1.children + e2;

   e3.prob = e1.prob + e2.prob +
             log P_r(e2.label,e2.headtag,e2.headword |
                      parent-label       == e1.label,
                      headchild-label    == e1.headlabel,
                      headword           == e1.headword,
                      headtag            == e1.headtag,
                      distance.adjacency == e1.rc.adj,
                      distance.verb      == e1.rc.verb);

   add_edge(e3,e1.start,e2.end);
}
```

Figure B.4: **join_2_edges_follow(edge e1,edge e2)** joins two edges $e1$ and $e2$ to form a new edge $e3$. (a) illustrates the process. (b) gives pseudocode.




Prob = X   Prob = Y   Prob = X × Y × $\mathcal{P}_L(L_i(l_i) \mid P, H, ...)$

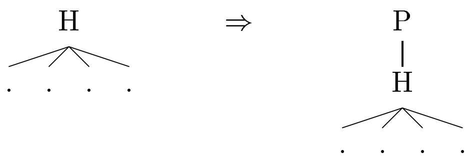
edge e1   edge e2   edge e3

(b)

```
void join_2_edges_precede(edge e1,edge e2)
{
   edge e3;

   e3.type      = 1;
   e3.label     = e2.label;
   e3.headlabel = e2.headlabel;
   e3.headword  = e2.headword;
   e3.headtag   = e2.headtag;
   e3.stop      = FALSE;
   e3.rc        = e2.rc;
   e3.lc.adj    = FALSE;
   e3.lc.verb   = e2.lc.verb OR
                  e1.lc.verb OR
                  e1.rc.verb OR
                  e1.headtag is a verb;
   e3.start     = e1.start;
   e3.end       = e2.end;
   e3.children  = e1 + e2.children;

   e3.prob = e1.prob + e2.prob +
             log P_l(e1.label,e1.headtag,e1.headword |
                     parent-label       == e2.label,
                     headchild-label    == e2.headlabel,
                     headword           == e2.headword,
                     headtag            == e2.headtag,
                     distance.adjacency == e2.lc.adj,
                     distance.verb      == e2.lc.verb);

   add_edge(e3,e1.start,e2.end);
}
```

Figure B.5: **join_2_edges_precede(edge e1,edge e2)** joins two edges $e1$ and $e2$ to form a new edge $e3$. (a) illustrates the process. (b) gives pseudocode.

(a)



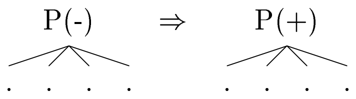

Prob = X $\qquad$ Prob = X $\times \mathcal{P}_H(H \mid P, ...)$

(b)

```
void add_singles(edge e)
{
   edge e3;

   e3.type      = 1;
   e3.headlabel = e.label;
   e3.headword  = e.headword;
   e3.headtag   = e.headtag;
   e3.stop      = FALSE;
   e3.lc.adj    = TRUE;
   e3.lc.verb   = FALSE;
   e3.rc.adj    = TRUE;
   e3.rc.verb   = FALSE;
   e3.start     = e.start;
   e3.end       = e.end;

   e3.children  = e;

   foreach X in the set of non-terminals
   {
      e3.label     = X;

      e3.prob = e.prob +
                log P_h(e.label |
                        parent-label == X,
                        headword     == e.headword,
                        headtag      == e.headtag);

      add_edge(e3,e.start,e.end);
   }
}
```

Figure B.6: **add_singles(edge e)** adds edges with a unary rule re-writing to edge $e$. (a) illustrates the process. (b) gives pseudocode.

(a)




Prob = X $\qquad$ Prob = X $\times \mathcal{P}_L(STOP \mid ....) \times \mathcal{P}_R(STOP \mid ....)$

(b)

```
void add_stops(edge e)
{
   edge e3;

   e3.type      = 1;
   e3.label     = e.label;
   e3.headlabel = e.headlabel;
   e3.headword  = e.headword;
   e3.headtag   = e.headtag;
   e3.stop      = TRUE;
   e3.lc        = e.lc;
   e3.rc        = e.rc;
   e3.start     = e.start;
   e3.end       = e.end;
   e3.children  = e.children;
   e3.prob = e.prob +
             log P_r( STOP |
                     parent-label       == e.label,
                     headchild-label    == e.headlabel,
                     headword           == e.headword,
                     headtag            == e.headtag,
                     distance.adjacency == e.rc.adj,
                     distance.verb      == e.rc.verb) +
             log P_l( STOP |
                     parent-label       == e.label,
                     headchild-label    == e.headlabel,
                     headword           == e.headword,
                     headtag            == e.headtag,
                     distance.adjacency == e.lc.adj,
                     distance.verb      == e.lc.verb);

   add_edge(e3,e.start,e.end);
}
```

Figure B.7: **add_stops(edge e)** forms a new edge by adding stop probabilities to edge $e$. (a) illustrates the process. P(-) has stop == FALSE, P(+) has stop == TRUE. (b) gives pseudocode.

```
void add_singles_stops(int start,int end)
{
   //MAXUNARY is the maximum number of unary productions allowed
   //in a row
   #define MAXUNARY 5

   foreach edge X in chart[start,end] such that X.stop == FALSE
      add_stops(X)

   for i = 1 to MAXUNARY
   {
      foreach edge Y created by last set of calls to add_stops
         add_singles(Y)

      foreach edge Y created by last set of calls to add_singles
         add_stops(Y)
   }

}
```

Figure B.8: **add_singles_stops(int start, int end)** adds all stop probabilities, and edges which are created by unary rules, for the chart entries spanning words start-end. The code makes the approximation that there can never be more than MAXUNARY unary rules building directly on top of each other.

```
void initialize()
{
   edge e;

   //n is the number of words in the input sentence
   for i = 1 to n
   {
      if(word_i is an ``unknown'' word)
         set X = {POS tag from tagger for word_i}
      else
         set X = {set of all tags seen for word_i in training data}

      foreach POS-tag T in X
      {
         e.type      = 0;
         e.label     = T;
         e.headword  = word_i;
         e.headtag   = T;
         e.stop      = TRUE;
         e.lc.adj    = TRUE;
         e.lc.verb   = FALSE;
         e.rc.adj    = TRUE;
         e.rc.verb   = FALSE;
         e.start     = i;
         e.end       = i;
         e.prob      = 0;

         add_edge(e,i,i);
         add_singles_stops(i,i);
      }
   }
}
```

Figure B.9: **initialize()** initializes the chart.

```
void complete(int start,int end)
{
   //split is the split point
   for split = start to end-1
   {
      foreach edge e1 in chart[start,split] such that e1.stop == FALSE
         foreach edge e2 in chart[split+1,end] such that e2.stop == TRUE
            join_2_edges_follow(e1,e2);

      foreach edge e1 in chart[start,split] such that e1.stop == TRUE
         foreach edge e2 in chart[split+1,end] such that e2.stop == FALSE
            join_2_edges_precede(e1,e2);
   }

   add_singles_stops(start,end);
}
```

Figure B.10: **complete(int start,int end)** completes all edges in the chart spanning words start to end.

```
edge parse()
{
   initialize();

   //assume n is the number of words in the sentence
   for span = 2 to n
      for start = 1 to n-span+1
      {
         end = start + span -1;
         complete(start,end);
      }

   //assume TOP is the start symbol (must be at the top of the tree)
   X = edge in chart[1,n,TOP] with highest probability;

   return X;
}
```

Figure B.11: **parse()** parses a sentence, returning the edge pointing to the top of the highest probability tree.

# Appendix C

# The Parsing Algorithm for Model 2 of Chapter 7

Model 2 introduces subcategorization features, and requires some (relatively minor) modifications to the parsing algorithm for model 1 described in appendix B. The changes are:

- The **context** data-type is extended to include a subcat frame, i.e. a bag specifying the complements that an edge still requires. Hence the **lc** and **rc** variables in the edge data-type now contain the left and right subcategorization frames for the edge, as well as the distance features. See table C.1.
- **add_singles** is modified to add subcategorization frames and probabilities. See figure C.1.
- **join_2_edges_follow, join_2_edges_precede,** and **add_stops** are modified to keep track of the subcategorization frame, and to condition the probabilities on the subcat frame. See figures C.2, C.3 and C.4.
- **initialize()** is modified to start leaf-nodes with empty subcat frames. See figure C.5.

```
void add_singles(edge e)
{
   edge e3;

   e3.type      = 1;
   e3.headlabel = e.label;
   e3.headword  = e.headword;
   e3.headtag   = e.headtag;
   e3.stop      = FALSE;
   e3.lc.adj    = TRUE;
   e3.lc.verb   = FALSE;
   e3.rc.adj    = TRUE;
   e3.rc.verb   = FALSE;
   e3.start     = e.start;
   e3.end       = e.end;

   e3.children  = e;

   foreach X in the set of non-terminals
   foreach Y in the set of possible left subcat frames
   foreach Z in the set of possible right subcat frames
   {
      e3.label     = X;
      e3.lc.subcat = Y;
      e3.rc.subcat = Z;

      e3.prob = e.prob +
                log P_h(e.label |
                        parent-label == X,
                        headword     == e.headword,
                        headtag      == e.headtag) +
                log P_lc(Y |
                         parent-label == X,
                         head-label   == e.label,
                         headword     == e.headword,
                         headtag      == e.headtag) +
                log P_rc(Z |
                         parent-label == X,
                         head-label   == e.label,
                         headword     == e.headword,
                         headtag      == e.headtag);

      add_edge(e3,e.start,e.end);
   }
}
```



Figure C.1: **add_singles(edge e)** adds edges with a unary rule re-writing to edge $e$. Model 2 additionally has loops over left and right subcat frames.

```
void join_2_edges_follow(edge e1,edge e2)
{
   edge e3;

   e3.type      = 1;
   e3.label     = e1.label;
   e3.headlabel = e1.headlabel;
   e3.headword  = e1.headword;
   e3.headtag   = e1.headtag;
   e3.stop      = FALSE;
   e3.lc        = e1.lc;
   e3.rc.adj    = FALSE;
   e3.rc.verb   = e1.rc.verb OR
                  e2.lc.verb OR
                  e2.rc.verb OR
                  e2.headtag is a verb;
   e3.start     = e1.start;
   e3.end       = e2.end;
   e3.children  = e1.children + e2;

   e3.lc.subcat = e1.lc.subcat;
   e3.rc.subcat = e2.lc.subcat;

   if e2.label is a complement
      e3.rc.subcat = e3.rc.subcat - e2.label;

   e3.prob = e1.prob + e2.prob +
             log P_r(e2.label,e2.headtag,e2.headword |
                     parent-label       == e1.label,
                     headchild-label    == e1.headlabel,
                     headword           == e1.headword,
                     headtag            == e1.headtag,
                     distance.adjacency == e1.rc.adj,
                     distance.verb      == e1.rc.verb,
                     subcat             == e1.rc.subcat);

   add_edge(e3,e1.start,e2.end);
}
```

Figure C.2: **join_2_edges_follow(edge e1,edge e2)** joins two edges $e1$ and $e2$ to form a new edge $e3$. Model 2 has modifications to calculate the subcat frames for the new edge, and add the subcat frame to the conditioning context.

```
void join_2_edges_precede(edge e1,edge e2)
{
   edge e3;

   e3.type      = 1;
   e3.label     = e2.label;
   e3.headlabel = e2.headlabel;
   e3.headword  = e2.headword;
   e3.headtag   = e2.headtag;
   e3.stop      = FALSE;
   e3.rc        = e2.rc;
   e3.lc.adj    = FALSE;
   e3.lc.verb   = e2.lc.verb OR
                  e1.lc.verb OR
                  e1.rc.verb OR
                  e1.headtag is a verb;
   e3.start     = e1.start;
   e3.end       = e2.end;
   e3.children  = e1 + e2.children;

   e3.lc.subcat = e2.lc.subcat;
   e3.rc.subcat = e2.rc.subcat;

   if e1.label is a complement
      e3.lc.subcat = e3.lc.subcat - e1.label;

   e3.prob = e1.prob + e2.prob +
             log P_l(e1.label,e1.headtag,e1.headword |
                     parent-label       == e2.label,
                     headchild-label    == e2.headlabel,
                     headword           == e2.headword,
                     headtag            == e2.headtag,
                     distance.adjacency == e2.lc.adj,
                     distance.verb      == e2.lc.verb,
                     subcat             == e2.lc.subcat);

   add_edge(e3,e1.start,e2.end);
}
```

Figure C.3: **join_2_edges_precede(edge e1,edge e2)** joins two edges $e1$ and $e2$ to form a new edge $e3$. Model 2 has modifications to calculate the subcat frames for the new edge, and add the subcat frame to the conditioning context.

```
void add_stops(edge e)
{
   edge e3;

   e3.type      = 1;
   e3.label     = e.label;
   e3.headlabel = e.headlabel;
   e3.headword  = e.headword;
   e3.headtag   = e.headtag;
   e3.stop      = TRUE;
   e3.lc        = e.lc;
   e3.rc        = e.rc;
   e3.start     = e.start;
   e3.end       = e.end;
   e3.children  = e.children;
   e3.prob = e.prob +
             log P_r( STOP |
                     parent-label       == e.label,
                     headchild-label    == e.headlabel,
                     headword           == e.headword,
                     headtag            == e.headtag,
                     distance.adjacency == e.rc.adj,
                     distance.verb      == e.rc.verb,
                     subcat             == e.rc.subcat) +
             log P_l( STOP |
                     parent-label       == e.label,
                     headchild-label    == e.headlabel,
                     headword           == e.headword,
                     headtag            == e.headtag,
                     distance.adjacency == e.lc.adj,
                     distance.verb      == e.lc.verb,
                     subcat             == e.lc.subcat);

   add_edge(e3,e.start,e.end);
}
```

Figure C.4: **add_stops(edge e)** forms a new edge by adding stop probabilities to edge *e*. Model 2 additionally conditions on the subcat frame variable (the probability of stopping will in fact be 0 if this frame is not empty).

| Variable | Type | Description |
|---|---|---|
| adj | boolean | (adjacency) TRUE if the head has taken no modifiers |
| verb | boolean | TRUE if one of the modifiers to the head dominates a verb |
| subcat | bag | specifies the complements still required by the edge |

Table C.1: Variables in the *context* data-type

```
void initialize()
{
   edge e;

   //n is the number of words in the input sentence
   for i = 1 to n
   {
      if(word_i is an ‘‘unknown’’ word)
         set X = {POS tag from tagger for word_i}
      else
         set X = {set of all tags seen for word_i in training data}

      foreach POS-tag T in X
      {
         e.type      = 0;
         e.label     = T;
         e.headword  = word_i;
         e.headtag   = T;
         e.stop      = TRUE;
         e.lc.adj    = TRUE;
         e.lc.verb   = FALSE;
         e.lc.subcat = empty;
         e.rc.adj    = TRUE;
         e.rc.verb   = FALSE;
         e.rc.subcat = empty;
         e.start     = i;
         e.end       = i;
         e.prob      = 0;

         add_edge(e,i,i);
         add_singles_stops(i,i);
      }
   }
}
```

Figure C.5: **initialize()** initializes the chart. Model 2 additionally sets the subcat frames to be empty.

# Appendix D

# An Analysis of Parsing Complexity for the Models of Chapter 7

In this section we derive an upper bound for the parsing complexity of the algorithms in Appendix B and C, sketched in figure 7.13. Note, however, that the beam search method means that the parsing algorithm is almost certainly more efficient in practice. (In fact, our feeling is that the running time of the algorithm depends much more on the effectiveness of the pruning method, rather than the asymptotic complexity of the algorithm.)

Calls to the functions `join_2_edges_follow`, `join_2_edges_precede`, `add_single` and `add_stops` take $O(1)$ time. The calls to `join_2_edges_follow` and `join_2_edges_precede` dominate the complexity of the algorithm, as they are most deeply nested (within 5 for/foreach loops). We can now analyse the number of calls to `join_2_edges_follow`. (The analysis for `join_2_edges_precede` is similar, and gives the same complexity.) `join_2_edges_follow` is buried within 5 loops, shown in the table below:

| Complexity | Loop |
|---|---|
| $O(n)$ | for span = 2 to n |
| $O(n)$ | for start = 1 to n-span+1 |
| $O(n)$ | for split = start to end-1 |
| $O(D_1)$ | foreach edge e1 in chart[start,split] such that e1.stop == FALSE |
| $O(D_2)$ | foreach edge e2 in chart[split+1,end] such that e2.stop == TRUE |

$D_1$ is an upper bound on the number of edges which do not have their `STOP` probabilities, for a given cell in the chart. $D_2$ is a similar bound for the number of edges *with* `STOP` probabilities. The running time is $O(n^3 D_1 D_2)$, where $D_1$ and $D_2$ are to be determined.

## D.1 A First Analysis of $D_1$ and $D_2$

$D_1$ and $D_2$ are related to the edge representation, and in particular the features taken into account for dynamic programming: i.e., the features compared when deciding whether two edges are equivalent, in that the lower probability edge can be safely discarded. Table B.1 describes the representation of edges; figure B.3 describes the comparison function for the two edges.

Assuming that an edge is a non-terminal, rather than a leaf-node (word/POS-tag pair), the following factors are taken into account:

- The head-word of the constituent. This has $O(n)$ possibilities.
- The head-tag of the constituent. This has $O(T)$ possibilities, where $T$ is the maximum number of different tags seen with any word in the vocabulary.
- The label of the constituent. This has $O(N)$ possibilities, where $N$ is the number of non-terminals in the grammar, excluding POS tags.
- The head-label of the constituent. This is the label of the non-terminal that is the head of the constituent. It has $O(N)$ possibilities (to be exact, $N+1$ possibilities; it can be any one of the non-terminals in the grammar, or it can be the same as the head-tag of the constituent).
- The left-distance variable. We define this to have $O(D)$ possibilities (D=3 for the models of chapter 7, as there are flags for adjacency and the presence of a verb).
- The right-distance variable. This also has $O(D)$ possibilities.
- The left-subcategorization state. We define this to have $O(L)$ possibilities. ($L$ is the number of distinct left-subcategorization states seen in conditioning contexts in training data.)
- The right-subcategorization state. We define this to have $O(R)$ possibilities. ($R$ is the number of distinct right-subcategorization states seen in conditioning contexts in training data.)

If an edge is a leaf (a POS-tag/word pair), their are $O(nT)$ possible values (only the word and its part-of-speech need to be specified). For other edges, there are $O(nTN^2D^2LR)$ possible values. If we assume $O(D_1) = O(D_2) = O(nTN^2D^2LR)$, then the overall parsing complexity is $O(n^3D_1D_2) = O(n^5T^2N^4D^4L^2R^2)$.

## D.2 A Second Analysis of $D_1$ and $D_2$

The complexity of the algorithm can be reduced by noting that once a constituent has its stop probabilities, some features become irrelevant for the dynamic programming comparison. Specifically,

- The head-label is not relevant.
- The distance variables can be collapsed from $O(D^2)$ to some lower bound $O(\bar{D})$. For example, in models 1, 2 and 3, $\bar{D} = 2$, as a single flag — whether or not an edge contains a verb — is the only distance feature required for a stopped edge.
- The left and right subcategorization frames are irrelevant (and will always be empty).

With these assumptions, $O(D_1)$ (the number of unstopped edges) remains as $O(nTN^2D^2LR)$, but $O(D_2) = O(nNT\bar{D})$. The running time of the entire algorithm is $O(n^3D_1D_2) = O(n^5T^2N^3D^2\bar{D}LR)$.

## D.3 A Third Analysis of $D_1$ and $D_2$

The next thing to note is that the analysis of $O(D_1)$ has been rather pessimistic. It assumes that all triples ⟨*parent non-terminal, head non-terminal, left-subcat state*⟩ or ⟨*parent non-terminal, head non-terminal, right-subcat state*⟩ are possible. In reality, many of these combinations will receive 0 probability under the model and will never be observed when decoding. This leads to an $O(N^2LR)$ factor in the complexity which can be reduced. Assume the following definition of the set $\mathcal{X}$:

$$\begin{aligned}\mathcal{X} = \{\langle \text{X,Y,L,R} \rangle \quad | \quad & \langle \text{Parent = X, Head-label = Y, left-subcat = L} \rangle \text{ and} \\ & \langle \text{Parent = X, Head-label = Y, right-subcat = R} \rangle \\ & \text{are both seen as conditioning contexts in training data}\}\end{aligned}$$

(D.1)

In the worst case, $\mathcal{X} = N^2LR$, but in practice $\mathcal{X}$ may be much smaller than $N^2LR$. The $O(N^2LR)$ factor is then reduced to $O(|\mathcal{X}|)$; $O(D_1)$ becomes $O(nT|\mathcal{X}|D^2)$; and the overall parsing complexity is $O(n^5|\mathcal{X}|T^2ND^2\bar{D})$.

# Appendix E

# Efficiency Considerations when Parsing

## E.1 Beam Search

To improve efficiency it is important to "prune" constituents in the chart that are relatively low in probability, and are therefore unlikely to be part of the highest probability parse for a sentence. [Caraballo and Charniak 98, Goodman 97b] have discussed pruning strategies quite extensively; this appendix describes the method used in the parsers in chapter 7.

### E.1.1 The Figure of Merit

The first thing to consider is what "score" or probability should be used to rank edges in the chart. An obvious choice is the probability stored with each edge in the chart: the "inside" probability, or $P(\text{subtree} \mid \text{label, head-tag, head-word})$, the conditional probability of the edge's subtree, given its non-terminal label, head-word, and head-tag.

However, as has been argued in [Caraballo and Charniak 98, Goodman 97b], the inside probability alone is a poor measure of how likely an edge is to be part of the highest probability tree. The problem is that the measure takes no account of the prior probability of seeing a constituent with the particular (label, head-tag, head-word) triple.

For example, an extremely unlikely constituent, such as a **VP** headed by the preposition *of*, might easily get a high inside probability because the *conditional* probability $P(\text{subtree} \mid \text{VP, Preposition, of})$ of seeing the subtree once the (VP,preposition,of) triple was generated could be high.

For this reason an additional "prior" probability of seeing the (label, head-tag, head-word) triple is taken into account ([Goodman 97b] also describes the use of a prior; [Caraballo and Charniak 98] describe rather more sophisticated ways of calculating the prior). So the measure used to rank edges (or "figure of merit", as named in [Caraballo and Charniak 98]) is

$$P_{inside}(\text{subtree} \mid \text{label, head-tag, head-word}) \times P_{prior}(\text{label, head-tag, head-word}) \tag{E.1}$$

where we decompose $P_{prior}$ as

$$P_{prior}(\text{label, head-tag, head-word}) = P(\text{head-tag, head-word}) \times P(\text{label} \mid \text{head-tag, head-word}) \tag{E.2}$$

and the second probability term is smoothed through linear interpolation. The counts are collected from all events where a (label, head-tag, head-word) triple is generated: either as part of a dependency or unary event.

### E.1.2 The Beam

Having defined the figure of merit for each edge, the beam strategy is relatively simple. Given that the highest score for any constituent in span *start...end* of the chart is $bestprob[start, end]$, then any other constituent in this span of the chart must have probability $> \alpha\ bestprob[start, end]$. $\alpha$ is the beam width. For the experiments in chapter 7, $\alpha = \frac{1}{10000}$ was used.

## E.2 Temporary Caching of Probabilities

The calls to the probability functions $P_l$, $P_r$, $P_h$, $P_{lc}$, and $P_{rc}$ are expensive. Each probability calculation typically requires look up of several hashed counts, and several floating point additions/multiplications.

It turns out that for a particular sentence being parsed, there are often many repeated calls to these functions calculating exactly the same parameter values. To reduce the amount of repeated computation, a temporary cache of complete probability values is stored for each sentence being parsed. When making a call to calculate a probability this temporary hash table is first consulted to see if that probability has been calculated before — if so the value is recovered immediately and returned. Otherwise, the probability is calculated using the full set of hash look ups and floating point operations, and is stored in the temporary cache before being returned.

# Bibliography


[Abney 97] S. Abney. 1997. Stochastic Attribute-Value Grammars. *Computational Linguistics,* 23(4):597-618.

[Allen 87] J. Allen. 1987. *Natural Language Understanding.* Benjamin/Cummings Publishing.

[Alshawi 96] H. Alshawi. 1996. Head Automata and Bilingual Tiling: Translation with Minimal Representations. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 167-176.

[Alshawi and Carter 94] H. Alshawi and D. Carter. Training and Scaling Preference Functions for Disambiguation. *Computational Linguistics,* 20(4):635-648.

[Appelt et al. 93] D. Appelt, J. Hobbs, J. Bear, D. J. Israel, and M. Tyson. 1993. FASTUS: a finite-state processor for information extraction from real-world text. In *Proceedings of IJCAI-93*, (Chambery, France), September 1993.

[Baker 79] J. K. Baker. 1979. Trainable Grammars for Speech Recognition. In Jared J. Wolf and Dennis H. Klatt, editors, *Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America,* MIT, Cambridge, MA.

[Baum 71] Baum, L.E. (1971). An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. In *Inequalities, III: Proceedings of a Symposium.* (Shish, Qved ed.). New York: Academic Press.

[BD 77] Bickel and Docksum (1977). *Mathematical Statistics: Basic Ideas and Selected Topics.* Prentice Hall, Englewood Cliffs, New Jersey.

[Bikel et al. 97] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. 1997. Nymble: a High-Performance Learning Name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 194-201.

[Black et al. 91] E. Black et al. 1991. A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the February 1991 DARPA Speech and Natural Language Workshop.*

[Black et al. 92a] E. Black, J. Lafferty and S. Roukos. 1992. Development and Evaluation of a Broad-Coverage Probabilistic Grammar of English-Language Computer Manuals. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 185-192.

[Black et al. 92b] E. Black, F. Jelinek, J. Lafferty, D. Magerman, R. Mercer and S. Roukos. 1992. Towards History-Based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, Harriman, NY.

[Black et al. 93] E. Black, R. Garside and G. Leech. 1993. *Statistically-Driven Computer Grammars of English: The IBM/Lancaster Approach.* Rodopi B.V., Amsterdam–Atlanta, GA.

[Bod 93] R. Bod. 1993. Using an Annotated Corpus as a Stochastic Grammar. In *Proceedings of the Sixth Conference of the European Chapter of the ACL*, pages 37-44.

[Booth and Thompson 73] T. L. Booth and R. A. Thompson. 1973. *Applying Probability Measures to Abstract Languages.* IEEE Transactions on Computers, C-22(5), pages 442-450.

[Brew 95] C. Brew. (1995). Stochastic HPSG. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 83-89, Dublin, Ireland. University College.

[Brill 93] E. Brill. 1993. Automatic Grammar Induction and Parsing Free Text: A Transformation-Based Approach. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics.*

[Brill 95] E. Brill. 1995. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*, 21(4):543-565.

[Brill and Resnik 94] E. Brill and P. Resnik. A Rule-Based Approach to Prepositional Phrase Attachment Disambiguation. In *Proceedings of the fifteenth international conference on computational linguistics (COLING-1994)*, 1994.

[Briscoe and Carroll 93] T. Briscoe and J. Carroll. 1993. Generalized LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. *Computational Linguistics*, 19(1):25-60.

[Brown et al. 1992] P. F. Brown, V. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer. 1992. "Class-based $n$-gram models of natural language." *Computational Linguistics,* **18**(4), pages 467–479.

[Carroll and Briscoe 95] J. Carroll and T. Briscoe. 1995. Apportioning Development Effort in a Probabilistic LR Parsing System through Evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, University of Pennsylvania, May 1996.

[Charniak 93] E. Charniak. 1993. *Statistical language learning.* Cambridge, Mass.: MIT Press.

[Charniak et al. 93] E. Charniak, C. Hendrickson, N. Jacobson and M. Perkowitz. 1993. Equations for Part-of-Speech Tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93).*

[Charniak and Carroll 94] E. Charniak and G. Carroll. 1994. Context-Sensitive Statistics For Improved Grammatical Language Models. In *Proceedings of the 12th National Conference on Artificial Intelligence*, AAAI Press, Seattle, WA. pages 742–747.

[Charniak 96] E. Charniak. 1996. Tree-Bank Grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference*, AAAI 96, IAAI 96, August 4-8, 1996, Portland, Oregon. pages 1031–1036.

[Charniak 97] E. Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park (1997).

[Caraballo and Charniak 98] S. Caraballo and E. Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics,* **24**(2), pages 275–298.

[Chelba and Jelinek 98] C. Chelba and F. Jelinek. 1998. Exploiting Syntactic Structure for Language Modeling. In *Proceedings of COLING-ACL 1998*, Montreal.

[Chen and Goodman 96] S. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318.

[Chitrao and Grishman 90] M. Chitrao and R. Grishman. 1990. Statistical Parsing of Messages. In *Proceedings Speech and Natural Language Workshop*, Hidden Valley, PA, pages 263–266, Morgan Kaufman Publishers.

[Chomsky 57] N. Chomsky. 1957. *Syntactic Structures*, Mouton, The Hague.

[Chomsky 95] N. Chomsky. 1995. *The Minimalist Program.* Cambridge, Mass.: The MIT Press.

[Church and Patil 82] K. Church and R. Patil. 1982. Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table. *American Journal of Computational Linguistics*, 8(3-4):139–149.

[Church 88] K. Church. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Second Conference on Applied Natural Language Processing, ACL.*

[Collins and Brooks 95] M. Collins and J. Brooks. 1995. Prepositional Phrase Attachment through a Backed-off Model. *Proceedings of the Third Workshop on Very Large Corpora*, pages 27-38.

[Collins 96] M. Collins. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184-191.

[Collins 97] M. Collins. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 16-23.

[Collins and Miller 98] M. Collins and S. Miller. 1998. Semantic Tagging using a Probabilistic Context Free Grammar. In *Proceedings of the Sixth Workshop on Very Large Corpora.*

[Dempster, Laird and Rubin 77] Dempster, A.P., Laird, N.M., and Rubin, D.B. (1977). Maximum Likelihood from Incomplete Data Via the EM Algorithm, *Journal of the Royal Statistical Society*, Ser B, 39, 1-38.

[Dowding et al. 93] J. Dowding, J. M. Gawron, D. Appelt, J. Bear, L. Cherny, R. Moore, D. Moran. 1993. GEMINI: A Natural Language System for Spoken-Language Understanding. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, Columbus, Ohio, pp. 54–61.

[Eisner 96] J. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. *Proceedings of COLING-96*, pages 340-345.

[Eisner 96b] J. Eisner. 1996. An Empirical Comparison of Probability Models for Dependency Grammar. Technical report IRCS-96-11, Institute for Research in Cognitive Science, University of Pennsylvania.

[Frank 92] R. Frank. 1992. Syntactic Locality and Tree Adjoining Grammar: Grammatical, Acquisition and Processing Perspectives. Ph.D. Thesis, University of Pennsylvania.

[Gale and Church 90] W. Gale and K. Church. Poor Estimates of Context are Worse than None. In *Proceedings of the June 1990 DARPA Speech and Natural Language Workshop*, Hidden Valley, Pennsylvania.

[Gazdar et al. 95] G. Gazdar, E.H. Klein, G.K. Pullum, I.A. Sag. 1985. *Generalized Phrase Structure Grammar.* Harvard University Press.

[Goodman 96] J. Goodman. 1996. Efficient Algorithms for Parsing the DOP Model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 143-152, May 1996.

[Goodman 96b] J. Goodman. 1996. Parsing Algorithms and Metrics. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 177-183, Santa Cruz, CA, June 1996.

[Goodman 97] J. Goodman. 1997. Probabilistic Feature Grammars. In *Proceedings of the Fourth International Workshop on Parsing Technologies.*

[Goodman 97b] J. Goodman. 1997. Global thresholding and multiple-pass parsing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing.*

[Goodman 98] J. Goodman. 1998. *Parsing Inside-Out.* Ph.D. Thesis, Harvard University.

[Grishman 95] R. Grishman. 1995. The NYU System for MUC-6 or Where's the Syntax? In *Proceedings of the Sixth Message Understanding Conference*, Morgan Kaufmann.

[Hajic 98] J. Hajic. 1998. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In *Issues of Valency and Meaning*, pages 106-132, Karolinum, Charles University Press, Prague.

[Hajic et al. 98] J. Hajic, E. Brill, M. Collins, B. Hladka, D. Jones, C. Kuo, L. Ramshaw, O. Schwartz, C. Tillmann, and D. Zeman. Core Natural Language Processing Technology Applicable to Multiple Languages. In *1998 Johns Hopkins Summer Workshop on Language Engineering, Final Report.*

[Hermjakob and Mooney 97] U. Hermjakob and R. J. Mooney. Learning Parse and Translation Decisions from Examples with Rich Context. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 482-489.

[Hindle and Rooth 91] D. Hindle and M. Rooth. 1991. Structural Ambiguity and Lexical Relations. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics.*

[Hindle and Rooth 93] D. Hindle and M. Rooth. Structural Ambiguity and Lexical Relations. *Computational Linguistics,* 19(1):103-120, 1993.

[Hopcroft and Ullman 79] J. E. Hopcroft and J. D. Ullman. 1979. *Introduction to automata theory, languages, and computation.* Reading, Mass.: Addison-Wesley.

[Jelinek 90] F. Jelinek. 1990. Self-organized Language Modeling for Speech Recognition. In *Readings in Speech Recognition*. Edited by Waibel and Lee. Morgan Kaufmann Publishers.

[Jelinek et al. 94] F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, S. Roukos. 1994. Decision Tree Parsing using a Hidden Derivation Model. *Proceedings of the 1994 Human Language Technology Workshop*, pages 272-277.

[Johnson 97] M. Johnson. 1997. The Effect of Alternative Tree Representations on Tree Bank Grammars. In *Proceedings of NeMLAP 3*.

[Jones and Eisner 92a] M. A. Jones and J. M. Eisner. 1992. A probabilistic parser applied to software testing documents. In *Proceedings of National Conference on Artificial Intelligence (AAAI-92)*, San Jose, pages 322-328.

[Jones and Eisner 92b] M. A. Jones and J. M. Eisner. 1992. A probabilistic parser and its application. In *Proceedings of the AAAI-92 Workshop on Statistically-Based Natural Language Processing Techniques*, San Jose.

[Joshi 87] A. Joshi. 1987. An Introduction to tree adjoining grammars, in A. Manaster-Ramis, editor, *Mathematics of Language*. John Benjamins, Amsterdam, 1987.

[Joshi and Srinivas 94] A. Joshi and B. Srinivas. 1994. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *International Conference on Computational Linguistics (COLING 94)*, Kyoto University, Japan, August 1994.

[Karp et al. 94] Daniel Karp, Yves Schabes, Martin Zaidel and Dania Egedi. A Freely Available Wide Coverage Morphological Analyzer for English. In *Proceedings of the 15th International Conference on Computational Linguistics*, 1994.

[Kaplan and Bresnan 82] R. Kaplan and J. Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations.* The MIT Press, Cambridge, MA, pages 173–281. Reprinted in Mary Dalrymple, Ronald M. Kaplan, John Maxwell, and Annie Zaenen, eds., *Formal Issues in Lexical-Functional Grammar*, 29–130. Stanford: Center for the Study of Language and Information. 1995.

[Katz 87] S. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. ASSP-35, No. 3, 1987.

[Koller, McAllester and Pfeffer 97] D. Koller, D. McAllester, and A. Pfeffer. 1997. Effective Bayesian Inference for Stochastic Programs. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI).* Providence, Rhode Island.

[Lafferty et al. 92] J. Lafferty, D. Sleator and, D. Temperley. 1992. Grammatical Trigrams: A Probabilistic Model of Link Grammar. *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language.*

[Lauer 95] M. Lauer. 1995. Corpus Statistics Meet the Noun Compound: Some Empirical Results. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, Boston, MA., pages 47-54.

[McCawley 68] J. McCawley. 1968. The Role of Semantics in Grammar. In Emmon Bach and Robert Harms, editors, *Universals in Linguistic Theory*, pages 124–169. Holt, Rinehart and Winston.

[Magerman and Marcus 91] D. Magerman and M. Marcus. 1991. Pearl: A Probabilistic Chart Parser. *Proceedings of the 1991 European ACL Conference*, Berlin, Germany.

[Magerman and Weir 92] D. Magerman and D. Weir. 1992. Efficiency, Robustness, and Accuracy in Picky Chart Parsing. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 40–47.

[Magerman 95] D. Magerman. 1994. *Natural Language Parsing as Statistical Pattern Recognition.* Ph.D. thesis, Stanford University.

[Magerman 95] D. Magerman. 1995. Statistical Decision-Tree Models for Parsing. *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276-283.

[Marcus 90] M. Marcus. 1990. Session Summary (Session 9: Automatic Acquisition of Linguistic Structure (Special session)). In *Proceedings of the June 1990 DARPA Speech and Natural Language Workshop*, Hidden Valley, Pennsylvania, pages 249–250.

[Marcus et al. 93] M. Marcus, B. Santorini and M. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313-330.

[Marcus et al. 94] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, B. Schasberger. 1994. The Penn Treebank: Annotating Predicate Argument Structure. *Proceedings of the 1994 Human Language Technology Workshop*, pages 110-115.

[de Marcken 95] C. de Marcken. 1995. On the Unsupervised Induction of Phrase-Structure Grammars. In *Proceedings of the Third Workshop on Very Large Corpora.*

[Miller et al. 98] S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone, R. Weischedel and the Annotation Group. 1998. *Algorithms that Learn to Extract Information. BBN: Description of the SIFT System as used for MUC-7.* In Proceedings of the Seventh Message Understanding Conference.

[Nederhof et al 98] M-J. Nederhof, A. Sarkar and G. Satta. 1998. Prefix Probabilities from Probabilistic Tree Adjoining Grammars. In *Proceedings of COLING-ACL 1998*, Montreal.

[Nederhof et al 1998b] M-J. Nederhof, A. Sarkar and G. Satta. 1998. Prefix Probabilities from Linear Indexed Grammars. In *Proceedings of the Fourth Workshop on Tree Adjoining Grammars, TAG+ 4*, Philadelphia, August 1998.

[MUC-6, 1995] *Proceedings of the Sixth Message Understanding Conference (MUC-6).* Morgan Kaufmann, San Mateo, CA.

[Pereira and Warren 80] F. Pereira and D. 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 and Schabes 92] F. Pereira and Y. Schabes. 1992. Inside-Outside Reestimation from Partially Bracketed Corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 128-135.

[Pinker 94] S. Pinker. 1994. *The Language Instinct.* Penguin Books.

[Pollard and Sag 94] C. Pollard and I. Sag. 1994. Head–Driven Phrase Structure Grammar. Chicago: University of Chicago Press and Stanford: CSLI Publications.

[Ramshaw and Marcus 95] L. Ramshaw and M. Marcus. 1995. Text Chunking using Transformation-Based Learning. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 82-94.

[Ratnaparkhi 98] A. Ratnaparkhi. 1998. Unsupervised Statistical Models for Prepositional Phrase Attachment. In *Proceedings of the Seventeenth International Conference on Computational Linguistics*, Aug. 10-14, 1998. Montreal, Canada.

[Ratnaparkhi 97] A. Ratnaparkhi. 1997. A Linear Observed Time Statistical Parser Based on Maximum Entropy Models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, Brown University, Providence, Rhode Island.

[Ratnaparkhi 96] A. Ratnaparkhi. 1996. A Maximum Entropy Model for Part-Of-Speech Tagging. *Conference on Empirical Methods in Natural Language Processing*, May 1996.

[Ratnaparkhi et al. 94] A. Ratnaparkhi, J. Reynar and S. Roukos. A Maximum Entropy Model for Prepositional Phrase Attachment. In *Proceedings of the ARPA Workshop on Human Language Technology,* Plainsboro, NJ, March 1994.

[Resnik 92] P. Resnik. 1992. Probabilistic Tree-Adjoining Grammar as a Framework for Statistical Natural Language Processing. In *Proceedings of COLING 92*, Volume II, pages 418–424.

[Russell and Norvig 95] S. J. Russell and P. Norvig. 1995. *Artificial intelligence: a modern approach.* Englewood Cliffs, N.J. : Prentice Hall.

[Sarkar 98] A. Sarkar. 1998. Conditions on Consistency of Probabilistic Tree Adjoining Grammars. In *Proceedings of COLING-ACL 1998*, Montreal.

[Schabes 92] Y. Schabes. 1992. Stochastic Lexicalized Tree-Adjoining Grammars. In *Proceedings of COLING 92*, Volume II, pages 426–432.

[Schabes et al 93] Y. Schabes, M. Roth and R. Osborne. 1993. Parsing the Wall Street Journal with the Inside-Outside Algorithm. In *Proceedings of the Sixth Conference of the European Chapter of the ACL*, pages 341–347.

[Schabes and Waters 93] Y. Schabes and R. Waters. 1993. Stochastic Lexicalized Context-Free Grammar. In *Proceedings of the Third International Workshop on Parsing*

*Technologies.*

[Sekine et al 92] S. Sekine, J. Carroll, S. Ananiadou, and J. Tsujii. 1992. Automatic Learning for Semantic Collocation. In *Proceedings of the Third Conference on Applied Natural Language Processing.*

[Sekine and Grishman 95] S. Sekine and R. Grishman. 1995. A Corpus-based Probabilistic Grammar with Only Two Non-terminals. In *Proceedings of the Fourth International Workshop on Parsing Technology.*

[Seneff 92] S. Seneff. 1992. TINA: A Natural Language System for Spoken Language Applications. *Computational Linguistics*, 18(1):61-86.

[Sleator and Temperley 91] D. Sleator and D. Temperley. 1991. Parsing English with a Link Grammar. Carnegie Mellon University Computer Science technical report CMU-CS-91-196, October 1991.

[Srinivas 97] B. Srinivas. 1997. Complexity of Lexical Descriptions and its Relevance to Partial Parsing. PhD Dissertation, University of Pennsylvania.

[Steedman 96] M. Steedman. 1996. *Surface Structure and Interpretation. (Linguistic Inquiry Monograph No.30)*, MIT Press.

[Thomason 86] M. G. Thomason. 1986. Syntactic Pattern Recognition: Stochastic Languages. In T.Y. Young and K-S Fu, editors, *Handbook of Pattern Recognition and Image Processing.* Academic Press.

[Weischedel et al. 93] R. Weischedel, M. Meteer, R. Schwartz, L. Ramshaw, and J. Palmucci. 1993. Coping with Ambiguity and Unknown Words through Probabilistic Models. *Computational Linguistics* 19(2): pages 359–382.

[Witten and Bell 91] I. T. Witten and T. C. Bell. 1991. The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, July 1991.

[Wood 93] M. M. Wood. 1993. *Categorial Grammars*, Routledge.

[Woods 70] W. A. Woods. 1970. Transition network grammars for natural language analysis. In Grosz, Jones, and Webber, editors, *Readings in Natural Language Processing.*