Chapter 14
Statistical Parsing

Two roads diverged in a wood, and I –
I took the one less traveled by...
Robert Frost, *The Road Not Taken*

The characters in Damon Runyon's short stories are willing to bet "on any proposition whatever", as Runyon says about Sky Masterson in *The Idyll of Miss Sarah Brown*; from the probability of getting aces back-to-back to the odds against a man being able to throw a peanut from second base to home plate. There is a moral here for language processing: with enough knowledge we can figure the probability of just about anything. The last two chapters have introduced sophisticated models of syntactic structure and its parsing. In this chapter we show that it is possible to build probabilistic models of syntactic knowledge and use some of this probabilistic knowledge in efficient probabilistic parsers.

One crucial use of probabilistic parsing is to solve the problem of **disambiguation**. Recall from Ch. 13 that sentences on average tend to be very syntactically ambiguous, due to problems like **coordination ambiguity** and **attachment ambiguity**. The CKY and Earley parsing algorithms could represent these ambiguities in an efficient way, but were not equipped to resolve them. A probabilistic parser offers a solution to the problem: compute the probability of each interpretation, and choose the most-probable interpretation. Thus, due to the prevalence of ambiguity, most modern parsers used for natural language understanding tasks (thematic role labeling, summarization, question-answering, machine translation) are of necessity probabilistic.

Another important use of probabilistic grammars and parsers is in **language modeling** for speech recognition. We saw that *N*-gram grammars are used in speech recognizers to predict upcoming words, helping constrain the acoustic model search for words. Probabilistic versions of more sophisticated grammars can provide additional predictive power to a speech recognizer. Of course humans have to deal with the same problems of ambiguity as do speech recognizers, and it is interesting that psychological experiments suggest that people use something like these probabilistic grammars in human language-processing tasks (e.g., human reading or speech understanding).

The most commonly used probabilistic grammar is the **probabilistic context-free grammar** (PCFG), a probabilistic augmentation of context-free grammars in which each rule is associated with a probability. We introduce PCFGs in the next section, showing how they can be trained on a hand-labeled Treebank grammar, and how they can be parsed. We present the most basic parsing algorithm for PCFGs, which is the probabilistic version of the **CKY algorithm** that we saw in Ch. 13.

We then show a number of ways that we can improve on this basic probability model (PCFGs trained on Treebank grammars). One method of improving a trained

Treebank grammar is to change the names of the non-terminals. By making the non-terminals sometimes more specific and sometimes more general, we can come up with a grammar with a better probability model that leads to improved parsing scores. Another augmentation of the PCFG works by adding more sophisticated conditioning factors, extending PCFGs to handle probabilistic **subcategorization** information and probabilistic **lexical dependencies**.

Finally, we describe the standard PARSEVAL metrics for evaluating parsers, and discuss some psychological results on human parsing.

14.1 Probabilistic Context-Free Grammars

The simplest augmentation of the context-free grammar is the **Probabilistic Context-Free Grammar** (**PCFG**), also known as the **Stochastic Context-Free Grammar** (**SCFG**), first proposed by Booth (1969). Recall that a context-free grammar G is defined by four parameters (N, Σ, R, S); a probabilistic context-free grammar is also defined by four parameters, with a slight augmentation to each of the rules in R:

PCFG

SCFG

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of **rules** or productions, each of the form $A \rightarrow \beta$ $[p]$,
	where A is a non-terminal,
	β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$,
	and p is a number between 0 and 1 expressing $P(\beta\|A)$
S	a designated **start symbol**

That is, a PCFG differs from a standard CFG by augmenting each rule in R with a conditional probability:

$$A \rightarrow \beta \quad [p] \tag{14.1}$$

Here p expresses the probability that the given non-terminal A will be expanded to the sequence β. That is, p is the conditional probability of a given expansion β given the left-hand-side (LHS) non-terminal A. We can represent this probability as

$$P(A \rightarrow \beta)$$

or as

$$P(A \rightarrow \beta|A)$$

or as

$$P(RHS|LHS)$$

Thus if we consider all the possible expansions of a non-terminal, the sum of their probabilities must be 1:

$$\sum_{\beta} P(A \rightarrow \beta) = 1$$

Grammar		Lexicon
S → *NP VP*	[.80]	*Det* → *that* [.10] | *a* [.30] | *the* [.60]
S → *Aux NP VP*	[.15]	*Noun* → *book* [.10] | *flight* [.30]
S → *VP*	[.05]	| *meal* [.15] | *money* [.05]
NP → *Pronoun*	[.35]	| *flights* [.40] | *dinner* [.10]
NP → *Proper-Noun*	[.30]	*Verb* → *book* [.30] | *include* [.30]
NP → *Det Nominal*	[.20]	| *prefer;* [.40]
NP → *Nominal*	[.15]	*Pronoun* → *I* [.40] | *she* [.05]
Nominal → *Noun*	[.75]	| *me* [.15] | *you* [.40]
Nominal → *Nominal Noun*	[.20]	*Proper-Noun* → *Houston* [.60]
Nominal → *Nominal PP*	[.05]	| *NWA* [.40]
VP → *Verb*	[.35]	*Aux* → *does* [.60] | *can* [40]
VP → *Verb NP*	[.20]	*Preposition* → *from* [.30] | *to* [.30]
VP → *Verb NP PP*	[.10]	| *on* [.20] | *near* [.15]
VP → *Verb PP*	[.15]	| *through* [.05]
VP → *Verb NP NP*	[.05]	
VP → *VP PP*	[.15]	
PP → *Preposition NP*	[1.0]	

Figure 14.1 A PCFG which is a probabilistic augmentation of the $\mathscr{L}_1$ miniature English CFG grammar and lexicon of Fig. 13.1 in Ch. 13. These probabilities were made up for pedagogical purposes and are not based on a corpus (since any real corpus would have many more rules, and so the true probabilities of each rule would be much smaller).

Fig. 14.1 shows a PCFG: a probabilistic augmentation of the $\mathscr{L}_1$ miniature English CFG grammar and lexicon . Note that the probabilities of all of the expansions of each non-terminal sum to 1. Also note that these probabilities were made up for pedagogical purposes. In any real grammar there are a great many more rules for each non-terminal and hence the probabilities of any particular rule would tend to be much smaller.

Consistent

A PCFG is said to be **consistent** if the sum of the probabilities of all sentences in the language equals 1. Certain kinds of recursive rules cause a grammar to be inconsistent by causing infinitely looping derivations for some sentences. For example a rule $S \rightarrow S$ with probability 1 would lead to lost probability mass due to derivations that never terminate. See Booth and Thompson (1973) for more details on consistent and inconsistent grammars.

How are PCFGs used? A PCFG can be used to estimate a number of useful probabilities concerning a sentence and its parse tree(s), including the probability of a particular parse tree (useful in disambiguation) and the probability of a sentence or a piece of a sentence (useful in language modeling). Let's see how this works.

14.1.1 PCFGs for Disambiguation

A PCFG assigns a probability to each parse tree T (i.e., each **derivation**) of a sentence S. This attribute is useful in **disambiguation**. For example, consider the two parses of the sentence "Book the dinner flights" shown in Fig. 14.2. The sensible parse on the left means "Book flights that serve dinner". The nonsensical parse on the right, however, would have to mean something like "Book flights on behalf of 'the dinner'?", the way that a structurally similar sentence like “Can you book John flights?” means something like “Can you book flights on behalf of John?”.

The probability of a particular parse T is defined as the product of the probabilities of all the n rules used to expand each of the n non-terminal nodes in the parse tree T, (where each rule i can be expressed as $LHS_i \rightarrow RHS_i$):

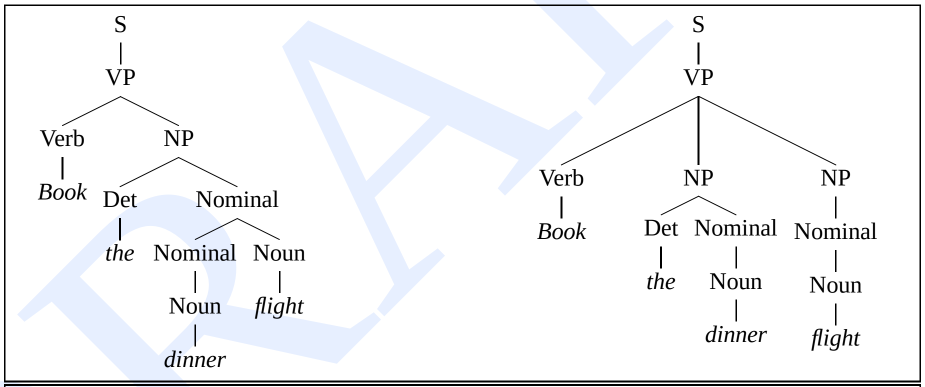
$$P(T,S) = \prod_{i=1}^{n} P(RHS_i|LHS_i) \tag{14.2}$$

The resulting probability $P(T,S)$ is both the joint probability of the parse and the sentence, and also the probability of the parse $P(T)$. How can this be true? First, by the definition of joint probability:

$$P(T,S) = P(T)P(S|T) \tag{14.3}$$

But since a parse tree includes all the words of the sentence, $P(S|T)$ is 1. Thus:

$$P(T,S) = P(T)P(S|T) = P(T) \tag{14.4}$$



Rules			P	Rules			P
S	→	VP	.05	S	→	VP	.05
VP	→	Verb NP	.20	VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20	NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20	NP	→	Nominal	.15
Nominal	→	Noun	.75	Nominal	→	Noun	.75
				Nominal	→	Noun	.75
Verb	→	book	.30	Verb	→	book	.30
Det	→	the	.60	Det	→	the	.60
Noun	→	dinner	.10	Noun	→	dinner	.10
Noun	→	flights	.40	Noun	→	flights	.40

Figure 14.2 Two parse trees for an ambiguous sentence, The transitive parse (a) corresponds to the sensible meaning “Book flights that serve dinner”, while the ditransitive parse (b) to the nonsensical meaning “Book flights on behalf of ‘the dinner”’.

The probability of each of the trees in Fig. 14.2 can be computed by multiplying together the probabilities of each of the rules used in the derivation. For example, the probability of the left tree in Fig. 14.2a (call it T_{left}) and the right tree (Fig. 14.2b or T_{right}) can be computed as follows:

$$\begin{aligned} P(T_{left}) &= .05*.20*.20*.20*.75*.30*.60*.10*.40 = \mathbf{2.2\times 10^{-6}} \\ P(T_{right}) &= .05*.10*.20*.15*.75*.75*.30*.60*.10*.40 = \mathbf{6.1\times 10^{-7}} \end{aligned}$$

We can see that the left (transitive) tree in Fig. 14.2(a) has a much higher probability than the ditransitive tree on the right. Thus this parse would correctly be chosen by a disambiguation algorithm which selects the parse with the highest PCFG probability.

Let's formalize this intuition that picking the parse with the highest probability is the correct way to do disambiguation. Consider all the possible parse trees for a given sentence S. The string of words S is called the **yield** of any parse tree over S. Thus out of all parse trees with a yield of S, the disambiguation algorithm picks the parse tree which is most probable given S:

Yield

(14.5)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T|S)$$

By definition, the probability $P(T|S)$ can be rewritten as $P(T,S)/P(S)$, thus leading to:

(14.6)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} \frac{P(T,S)}{P(S)}$$

Since we are maximizing over all parse trees for the same sentence, $P(S)$ will be a constant for each tree, so we can eliminate it:

(14.7)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T,S)$$

Furthermore, since we showed above that $P(T,S) = P(T)$, the final equation for choosing the most likely parse neatly simplifies to choosing the parse with the highest probability:

(14.8)
$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T)$$

14.1.2 PCFGs for Language Modeling

A second attribute of a PCFG is that it assigns a probability to the string of words constituting a sentence. This is important in **language modeling**, whether for use in speech recognition, machine translation, spell-correction, augmentative communication, or other applications. The probability of an unambiguous sentence is $P(T,S) = P(T)$ or just the probability of the single parse tree for that sentence. The probability of an ambiguous sentence is the sum of the probabilities of all the parse trees for the sentence:

$$P(S) = \sum_{T s.t. S=\text{yield}(T)} P(T,S) \tag{14.9}$$

$$= \sum_{T s.t. S=\text{yield}(T)} P(T) \tag{14.10}$$

An additional feature of PCFGs that is useful for language modeling is their ability to assign a probability to substrings of a sentence. For example, suppose we want to know the probability of the next word w_i in a sentence given all the words we've seen so far $w_1,\ldots,w_{i-1}$. The general formula for this is:

$$P(w_i|w_1,w_2,\ldots,w_{i-1}) = \frac{P(w_1,w_2,\ldots,w_{i-1},w_i,\ldots)}{P(w_1,w_2,\ldots,w_{i-1},\ldots)} \tag{14.11}$$

We saw in Ch. 4 a simple approximation of this probability using N-grams, conditioning on only the last word or two instead of the entire context; thus the **bigram approximation** would give us:

$$P(w_i|w_1,w_2,\ldots,w_{i-1}) \approx \frac{P(w_{i-1},w_i)}{P(w_{i-1})} \tag{14.12}$$

But the fact that the N-gram model can only make use of a couple words of context means it is ignoring potentially useful prediction cues. Consider predicting the word *after* in the following sentence from Chelba and Jelinek (2000):

(14.13) the contract ended with a loss of 7 cents after trading as low as 9 cents

A trigram grammar must predict *after* from the words *7 cents*, while it seems clear that the verb *ended* and the subject *contract* would be useful predictors that a PCFG-based parser could help us make use of. Indeed, it turns out that a PCFGs allow us to condition on the entire previous context $w_1,w_2,\ldots,w_{i-1}$ shown in Eq. 14.11. We'll see the details of ways to use PCFGs and augmentations of PCFGs as language models in Sec. 14.9.

In summary, this section and the previous one have shown that PCFGs can be applied both to disambiguation in syntactic parsing and to word prediction in language modeling. Both of these applications require that we be able to compute the probability of parse tree T for a given sentence S. The next few sections introduce some algorithms for computing this probability.

14.2 Probabilistic CKY Parsing of PCFGs

The parsing problem for PCFGs is to produce the most-likely parse $\hat{T}$ for a given sentence S, i.e.,

$$\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T) \tag{14.14}$$

The algorithms for computing the most-likely parse are simple extensions of the standard algorithms for parsing; there are probabilistic versions of both the CKY and Earley algorithms of Ch. 13. Most modern probabilistic parsers are based on the **probabilistic CKY** algorithm, first described by Ney (1991).

Probabilistic CKY

As with the CKY algorithm, we will assume for the probabilistic CKY algorithm that the PCFG is in Chomsky normal form. Recall from page 416 that grammars in CNF are restricted to rules of the form $A \rightarrow B\ C$, or $A \rightarrow w$. That is, the right-hand side of each rule must expand to either two non-terminals or to a single terminal.

For the CKY algorithm, we represented each sentence as having indices between the words. Thus an example sentence like

(14.15) Book the flight through Houston.

would assume the following indices between each word:

(14.16) ⓪ Book ① the ② flight ③ through ④ Houston ⑤

Using these indices, each constituent in the CKY parse tree is encoded in a two-dimensional matrix. Specifically, for a sentence of length n and a grammar that contains V non-terminals, we use the upper-triangular portion of an $(n+1)\times(n+1)$ matrix. For CKY, each cell $table[i,j]$ contained a list of constituents that could span the sequence of words from i to j. For probabilistic CKY, it's slightly simpler to think of the constituents in each cell as constituting a third dimension of maximum length V. This third dimension corresponds to each nonterminal that can be placed in this cell, and the value of the cell is then a probability for that nonterminal/constituent rather than a list of constituents. In summary, each cell $[i,j,A]$ in this $(n+1)\times(n+1)\times V$ matrix is the probability of a constituent A that spans positions i through j of the input.

Fig. 14.3 gives pseudocode for this probabilistic CKY algorithm, extending the basic CKY algorithm from Fig. 13.10.

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                                   and its probability
  for $j \leftarrow$ from 1 to LENGTH(words) do
     for all $\{ A \mid A \rightarrow words[j] \in grammar \}$
        $table[j-1,j,A] \leftarrow P(A \rightarrow words[j])$
     for $i \leftarrow$ from $j-2$ downto 0 do
        for $k \leftarrow i+1$ to $j-1$ do
              for all $\{ A \mid A \rightarrow BC \in grammar,$
                          and $table[i,k,B] > 0$ and $table[k,j,C] > 0 \}$
                 if $(table[i,j,A] < P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C])$ then
                    $table[i,j,A] \leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$
                    $back[i,j,A] \leftarrow \{k,B,C\}$
     return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Figure 14.3 The probabilistic CKY algorithm for finding the maximum probability parse of a string of *num_words* words given a PCFG grammar with *num_rules* rules in Chomsky Normal Form. *back* is an array of back-pointers used to recover the best parse. The *build_tree* function is left as an exercise to the reader.

Like the CKY algorithm, the probabilistic CKY algorithm as shown in Fig. 14.3 requires a grammar in Chomsky Normal Form. Converting a probabilistic grammar to CNF requires that we also modify the probabilities so that the probability of each parse remains the same under the new CNF grammar. Exercise 2 asks you to modify the algorithm for conversion to CNF in Ch. 13 so that it correctly handles rule probabilities.

In practice, we more often use a generalized CKY algorithm which handles unit productions directly rather than converting them to CNF. Recall that Exercise 3 asked you to make this change in CKY; Exercise 3 asks you to extend this change to probabilistic CKY.

Let's see an example of the probabilistic CKY chart, using the following mini-grammar which is already in CNF:

S	$\rightarrow$	$NP\ VP$	.80	Det	$\rightarrow$	*the*	.50
NP	$\rightarrow$	$Det\ N$	.30	Det	$\rightarrow$	*a*	.40
VP	$\rightarrow$	$V\ NP$	.20	N	$\rightarrow$	*meal*	.01
V	$\rightarrow$	*includes*	.05	N	$\rightarrow$	*flight*	.02

Given this grammar, Fig. 14.4 shows the first steps in the probabilistic CKY parse of this sentence:

(14.17) The flight includes a meal

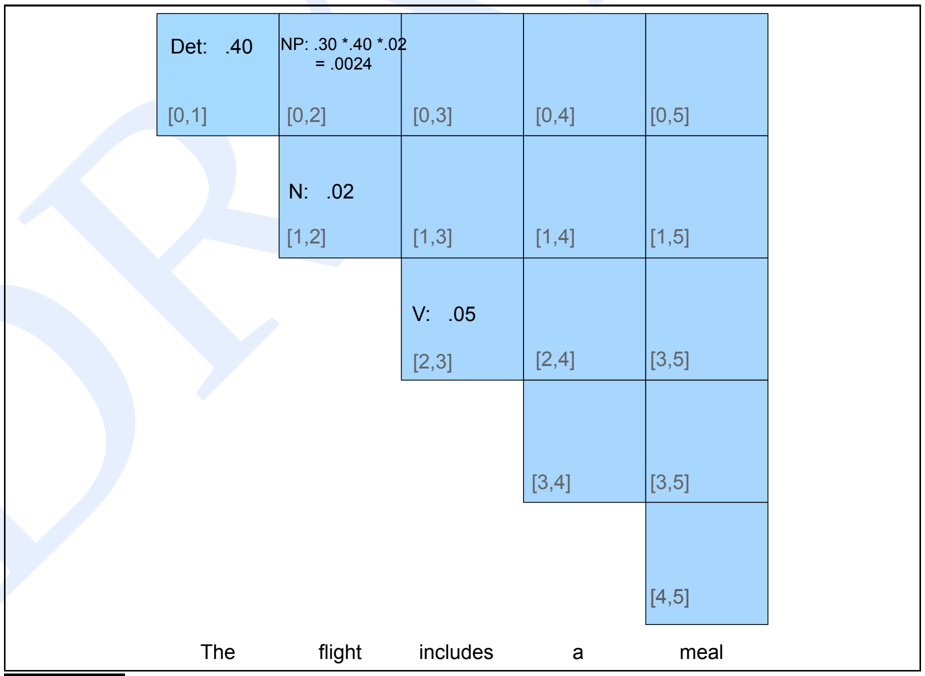


Figure 14.4 The beginning of the probabilistic CKY matrix. Filling out the rest of the chart is left as Exercise 4 for the reader.

14.3 Learning PCFG Rule Probabilities

Where do PCFG rule probabilities come from? There are two ways to learn probabilities for the rules of a grammar. The simplest way is to use a treebank, a corpus of already-parsed sentences. Recall that we introduced in Ch. 12 the idea of treebanks and the commonly-used **Penn Treebank** (Marcus et al., 1993), a collection of parse trees in English, Chinese, and other languages distributed by the Linguistic Data Consortium. Given a treebank, the probability of each expansion of a non-terminal can be computed by counting the number of times that expansion occurs and then normalizing.

$$P(\alpha \to \beta|\alpha) = \frac{\text{Count}(\alpha \to \beta)}{\sum_\gamma \text{Count}(\alpha \to \gamma)} = \frac{\text{Count}(\alpha \to \beta)}{\text{Count}(\alpha)} \tag{14.18}$$

If we don't have a treebank, but we do have a (non-probabilistic) parser, we can generate the counts we need for computing PCFG rule probabilities by first parsing a corpus of sentences with the parser. If sentences were unambiguous, it would be as simple as this: parse the corpus, increment a counter for every rule in the parse, and then normalize to get probabilities.

But wait! Since most sentences are ambiguous, i.e. have multiple parses, we don't know which parse to count the rules in. Instead, we need to keep a separate count for each parse of a sentence and weight each of these partial counts by the probability of the parse it appears in. But to get these parse probabilities to weight the rules we need to already have a probabilistic parser.

The intuition for solving this chicken-and-egg problem is to incrementally improve our estimates by beginning with a parser with equal rule probabilities, parsing the sentence, compute a probability for each parse, use these probabilities to weight the counts, then reestimate the rule probabilities, and so on, until our probabilities converge. The standard algorithm for computing this is called the **inside-outside** algorithm, and was proposed by Baker (1979) as a generalization of the forward-backward algorithm of Ch. 6. Like forward-backward, inside-outside is a special case of the EM (expectation-maximization) algorithm, and hence has two steps: the **expectation step**, and the **maximization step**. See Lari and Young (1990) or Manning and Schütze (1999) for a complete description of the algorithm.

Inside-outside

Expectation step

Maximization step

This use of the inside-outside algorithm to estimate the rule probabilities for a grammar is actually a kind of limited use of inside-outside. The inside-outside algorithm can actually be used not only to set the rule probabilities, but even to induce the grammar rules themselves. It turns out, however, that grammar induction is so difficult that inside-outside by itself is not a very successful grammar inducer; see the end notes for pointers to other grammar induction algorithms.

14.4 Problems with PCFGs

While probabilistic context-free grammars are a natural extension to context-free grammars, they have two main problems as probability estimators:

poor independence assumptions: CFG rules impose an independence assumption on probabilities, resulting in poor modeling of structural dependencies across the parse tree.

lack of lexical conditioning: CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities.

Because of these problems, most current probabilistic parsing models use some augmented version of PCFGs, or modify the Treebank-based grammar in some way. In the next few sections after discussing the problems in more detail we will introduce some of these augmentations.

14.4.1 Independence assumptions miss structural dependencies between rules

Let's look at these problems in more detail. Recall that in a CFG the expansion of a non-terminal is independent of the context, i.e., of the other nearby non-terminals in the parse tree. Similarly, in a PCFG, the probability of a particular rule like $NP \rightarrow Det\ N$ is also independent of the rest of the tree. By definition, the probability of a group of independent events is the product of their probabilities. These two facts explain why in a PCFG we compute the probability of a tree by just multiplying the probabilities of each non-terminal expansion.

Unfortunately this CFG independence assumption results in poor probability estimates. This is because in English the choice of how a node expands can after all be dependent on the location of the node in the parse tree. For example, in English it turns out that NPs that are syntactic **subjects** are far more likely to be pronouns, while NPs that are syntactic **objects** are far more likely to be non-pronominal (e.g., a proper noun or a determiner noun sequence), as shown by these statistics for NPs in the Switchboard corpus (Francis et al., 1999): [1]

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Unfortunately there is no way to represent this contextual difference in the probabilities in a PCFG. Consider two expansions of the non-terminal *NP* as a pronoun or as a determiner+noun. How shall we set the probabilities of these two rules? If we set their probabilities to their overall probability in the Switchboard corpus, the two rules have about equal probability.

$$\begin{aligned} NP &\rightarrow DT\ NN \quad .28 \\ NP &\rightarrow PRP \quad .25 \end{aligned}$$

[1] Distribution of subjects from 31,021 declarative sentences; distribution of objects from 7,489 sentences. This tendency is caused by the use of subject position to realize the **topic** or old information in a sentence (Givón, 1990). Pronouns are a way to talk about old information, while non-pronominal ("lexical") noun-phrases are often used to introduce new referents. We'll talk more about new and old information in Ch. 21.

Because PCFGs don't allow a rule probability to be conditioned on surrounding context, this equal probability is all we get; there is no way to capture the fact that in subject position, the probability for $NP \rightarrow PRP$ should go up to .91, while in object position, the probability for $NP \rightarrow DT\ NN$ should go up to .66.

These dependencies could be captured if the probability of expanding an NP as a pronoun (e.g., $NP \rightarrow PRP$) versus a lexical NP (e.g., $NP \rightarrow DT\ NN$) were *conditioned* on whether the NP was a subject or an object. Sec. 14.5 will introduce the technique of **parent annotation** for adding this kind of conditioning.

14.4.2 Lack of sensitivity to lexical dependencies

A second class of problems with PCFGs is their lack of sensitivity to the words in the parse tree. Words do play a role in PCFGs, since the parse probability includes the probability of a word given a part-of-speech (i.e., from rules like $V \rightarrow sleep$, $NN \rightarrow book$, etc).

But it turns out that lexical information is useful in other places in the grammar, such as in resolving prepositional phrase attachment (**PP**) ambiguities. Since prepositional phrases in English can modify a noun phrase or a verb phrase, when a parser finds a prepositional phrase, it must decide where to attach it into the tree. Consider the following example:

(14.19) Workers dumped sacks into a bin.

Fig. 14.5 shows two possible parse trees for this sentence; the one on the left is the correct parse; Fig. 14.6 shows another perspective on the preposition attachment problem, demonstrating that resolving the ambiguity in Fig. 14.5 is equivalent to deciding whether to attach the prepositional phrase into the rest of the tree at the NP or VP nodes; we say that the correct parse requires **VP attachment** while the incorrect parse implies **NP attachment**.

VP attachment
NP attachment

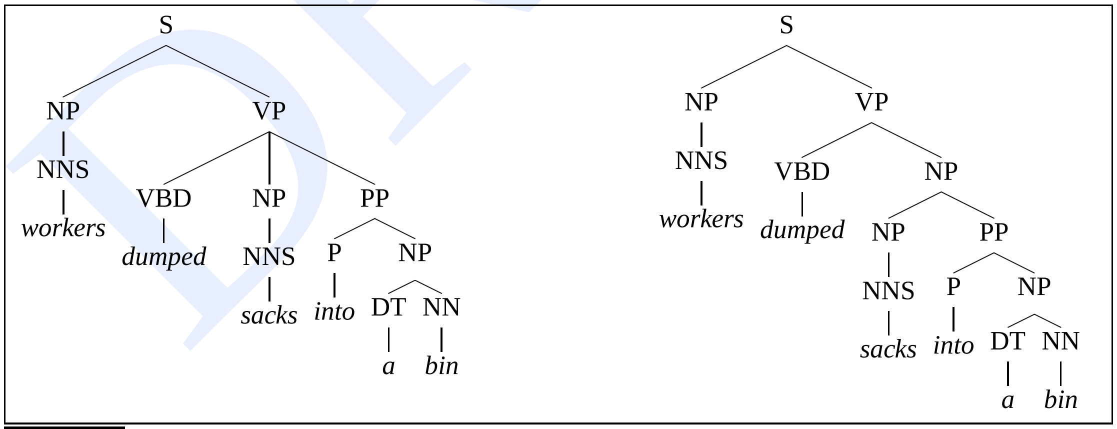


Figure 14.5 Two possible parse trees for a **prepositional phrase attachment ambiguity**. The left parse is the sensible one, in which 'into a bin' describes the resulting location of the sacks. In the right incorrect parse, the sacks to be dumped are the ones which are already 'into a bin', whatever that could mean.

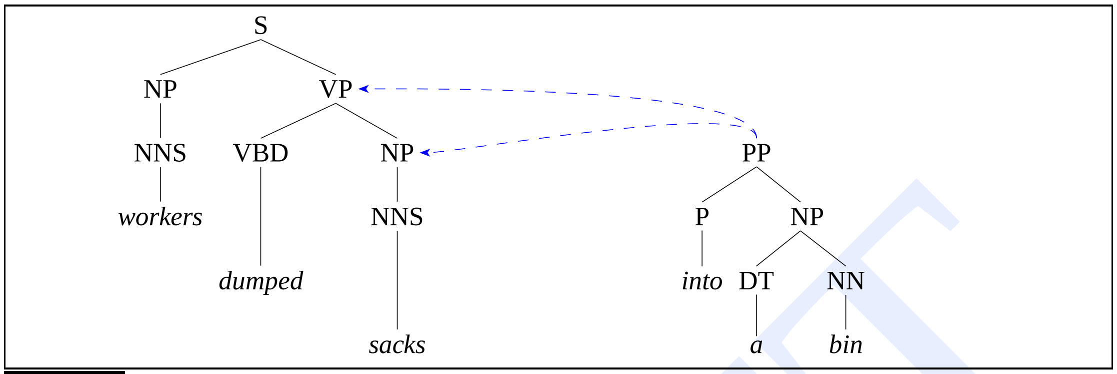


Figure 14.6 Another view of the preposition attachment problem; should the PP on the right attach to the VP or NP nodes of the partial parse tree on the left?

Why doesn't a PCFG already deal with PP attachment ambiguities? Note that the two parse trees in Fig. 14.5 have almost the exact same rules; they differ only in that the left-hand parse has this rule:

$$VP \rightarrow VBD\ NP\ PP$$

while the right-hand parse has these:

$$\begin{aligned} VP &\rightarrow VBD\ NP \\ NP &\rightarrow NP\ PP \end{aligned}$$

Depending on how these probabilities are set, a PCFG will **always** either prefer NP attachment or VP attachment. As it happens, NP attachment is slightly more common in English, and so if we trained these rule probabilities on a corpus, we might always prefer NP attachment, causing us to misparse this sentence.

But suppose we set the probabilities to prefer the VP attachment for this sentence. Now we would misparse the following sentence which requires NP attachment:

(14.20) fishermen caught tons of herring

What is the information in the input sentence which lets us know that (14.20) requires NP attachment while (14.19) requires VP attachment?

It should be clear that these preferences come from the identities of the verbs, nouns and prepositions. It seems that the affinity between the verb *dumped* and the preposition *into* is greater than the affinity between the noun *sacks* and the preposition *into*, thus leading to VP attachment. On the other hand in (14.20) , the affinity between *tons* and *of* is greater than that between *caught* and *of*, leading to NP attachment.

Thus in order to get the correct parse for these kinds of examples, we need a model which somehow augments the PCFG probabilities to deal with these **lexical dependency** statistics for different verbs and prepositions.

Lexical dependency

Coordination ambiguities are another case where lexical dependencies are the key to choosing the proper parse. Fig. 14.7 shows an example from Collins (1999), with two parses for the phrase *dogs in houses and cats*. Because *dogs* is semantically a better conjunct for *cats* than *houses* (and because dogs can't fit inside cats) the parse *[dogs in [$_{NP}$ houses and cats]]* is intuitively unnatural and should be dispreferred. The two parses in Fig. 14.7, however, have exactly the same PCFG rules and thus a PCFG will assign them the same probability.

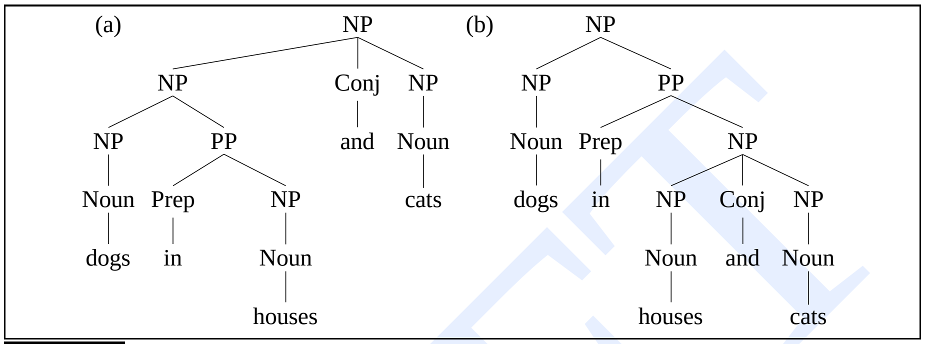


Figure 14.7 An instance of coordination ambiguity. Although the left structure is intuitively the correct one, a PCFG will assign them identical probabilities since both structure use the exact same rules. After Collins (1999).

In summary, we have shown in this section and the previous one that probabilistic context-free grammars are incapable of modeling important **structural** and **lexical** dependencies. In the next two sections we sketch current methods for augmenting PCFGs to deal with both these issues.

14.5 Improving PCFGs by Splitting and Merging Nonterminals

Let's start with the first of the two problems with PCFGs mentioned above: their inability to model structural dependencies, like the fact that NPs in subject position tend to be pronouns, where NPs in object position tend to have full lexical (non-pronominal) form. How could we augment a PCFG to correctly model this fact? One idea would be to **split** the NP non-terminal into two versions: one for subjects, one for objects. Having two nodes (e.g., $NP_{subject}$ and NP_{object}) would allow us to correctly model their different distributional properties, since we would have different probabilities for the rule $NP_{subject} \rightarrow PRP$ and the rule $NP_{object} \rightarrow PRP$.

Split

Parent annotation

One way to implement this intuition of splits is to do **parent annotation** (Johnson, 1998b), in which we annotate each node with its parent in the parse tree. Thus a node NP which is the subject of the sentence, and hence has parent S, would be annotated NPˆS, while a direct object NP, whose parent is VP, would be annotated NPˆVP. Fig. 14.8 shows an example of a tree produced by a grammar that parent annotates the phrasal non-terminals (like NP and VP).

In addition to splitting these phrasal nodes, we can also improve a PCFG by splitting the preterminal part-of-speech nodes (Klein and Manning, 2003b). For example, different kinds of adverbs (RB) tend to occur in different syntactic positions: the most

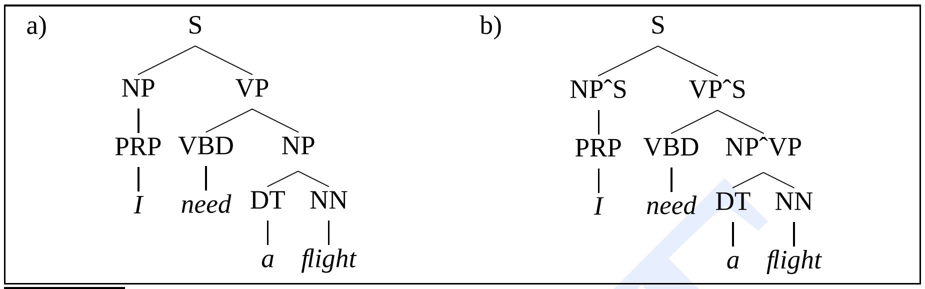


Figure 14.8 A standard PCFG parse tree (a) and one which has **parent annotation** on the nodes which aren't preterminal (b). All the non-terminal nodes (except the preterminal part-of-speech nodes) in parse (b) have been annotated with the identity of their parent.

common adverbs with ADVP parents are *also* and *now*, with VP parents are *n't* and *not*, and with NP parents *only* and *just*. Thus adding tags like RBˆADVP, RBˆVP, and RBˆNP can be useful in improving PCFG modeling.

Similarly, the Penn Treebank tag IN is used to mark a wide variety of parts-of-speech, including subordinating conjunctions (*while*, *as*, *if*), complementizers (*that*, *for*), and prepositions (*of*, *in*, *from*). Some of these differences can be captured by parent annotation (subordinating conjunctions occur under S, prepositions under PP), while others require specifically splitting the pre-terminal nodes. Fig. 14.9 shows an example from Klein and Manning (2003b), where even a parent annotated grammar incorrectly parses *works* as a noun in *to see if advertising works*. Splitting preterminals to allow *if* to prefer a sentential complement results in the correct verbal parse.

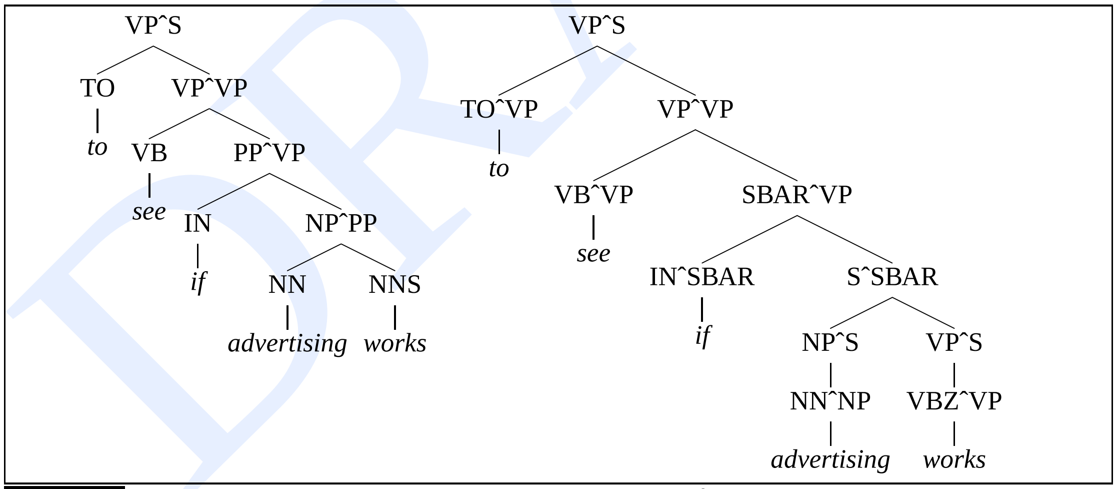


Figure 14.9 An incorrect parse even with a parent annotated parse (left). The correct parse (right), was produced by a grammar in which the pre-terminal nodes have been split, allowing the probabilistic grammar to capture the fact that *if* prefers sentential complements; adapted from Klein and Manning (2003b).

In order to deal with cases where parent annotation is insufficient, we can also hand-write rules that specify a particular node split based on other features of the tree. For example to distinguish between complementizer IN and subordinating conjunction IN, both of which can have the same parent, we could write rules conditioned on other aspects of the tree such as the lexical identity (the lexeme *that* is likely to be a complementizer, *as* a subordinating conjunction).

Node-splitting is not without problems; it increases the size of the grammar, and hence reduces the amount of training data available for each grammar rule, leading to overfitting. Thus it is important to split to just the correct level of granularity for a particular training set. While early models involved hand-written rules to try to find an optimal number of rules (Klein and Manning, 2003b), modern models automatically search for the optimal splits. The **split and merge** algorithm of Petrov et al. (2006), for example starts with a simple X-bar grammar, and then alternately splits the non-terminals, and merges together non-terminals, finding the set of annotated nodes which maximizes the likelihood of the training set treebank. As of the time of this writing, the performance of the Petrov et al. (2006) algorithm as the best of any known parsing algorithm on the Penn Treebank.

Split and merge

14.6 Probabilistic Lexicalized CFGs

The previous section showed that a simple probabilistic CKY algorithm for parsing raw PCFGs can achieve extremely high parsing accuracy if the grammar rule symbols are redesigned via automatic splits and merges.

In this section, we discuss an alternative family of models in which instead of modifying the grammar rules, we modify the probabilistic model of the parser to allow for **lexicalized** rules. The resulting family of lexicalized parsers includes the well-known **Collins parser** (Collins, 1999) and **Charniak parser** (Charniak, 1997), both of which are publicly available and widely used throughout natural language processing.

Collins parser

Charniak parser

We saw in Sec. 12.4.4 in Ch. 12 that syntactic constituents could be associated with a lexical **head**, and we defined a **lexicalized grammar** in which each non-terminal in the tree is annotated with its lexical head, where a rule like $VP \rightarrow VBD\ NP\ PP$ would be extended as:

Lexicalized grammar

(14.21) $$VP(dumped) \rightarrow VBD(dumped)\ NP(sacks)\ PP(into)$$

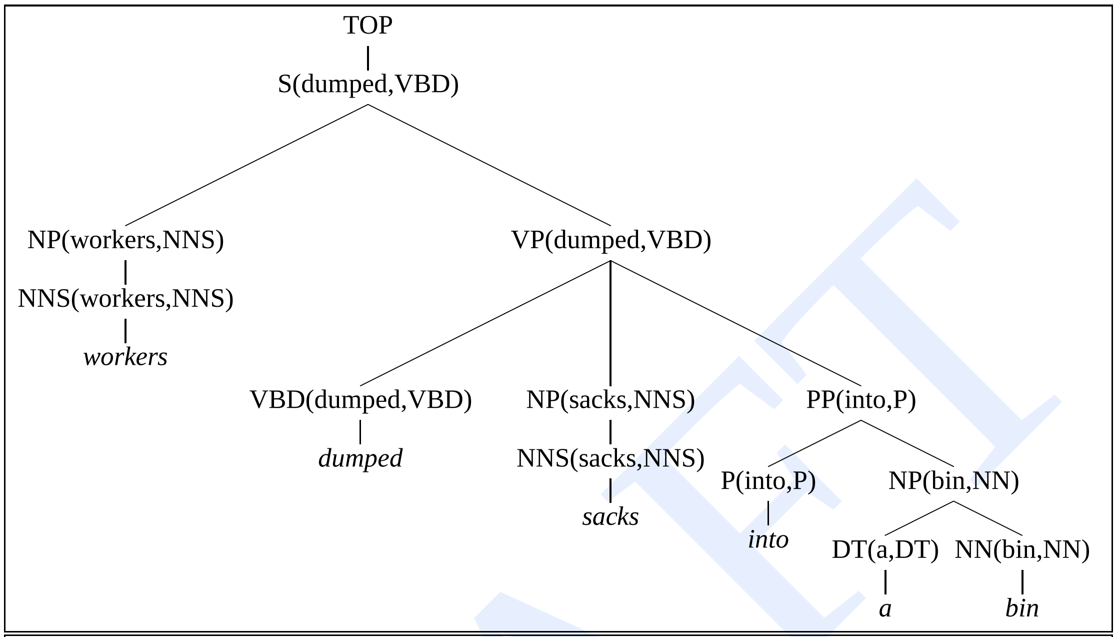
In the standard type of lexicalized grammar we actually make a further extension, which is to associate the **head tag**, the part-of-speech tags of the headwords, with the nonterminal symbols as well. Each rule is thus lexicalized by both the headword and the head tag of each constituent resulting in a format for lexicalized rules like:

Head tag

(14.22) $$VP(dumped,VBD) \rightarrow VBD(dumped,VBD)\ NP(sacks,NNS)\ PP(into,IN)$$

We show a lexicalized parse tree with head tags in Fig. 14.10, extended from Fig. 12.12.

In order to generate such a lexicalized tree, each PCFG rule must be augmented to identify one right-hand side constituent to be the head daughter. The headword for a node is then set to the headword of its head daughter, and the head tag to the part-of-speech tag of the headword. Recall that we gave in Fig. 12.13 a set of hand-written rules for identifying the heads of particular constituents.



Internal Rules					**Lexical Rules**		
TOP	→	S(dumped,VBD)			NNS(workers,NNS)	→	workers
S(dumped,VBD)	→	NP(workers,NNS)	VP(dumped,VBD)		VBD(dumped,VBD)	→	dumped
NP(workers,NNS)	→	NNS(workers,NNS)			NNS(sacks,NNS)	→	sacks
VP(dumped,VBD)	→	VBD(dumped, VBD)	NP(sacks,NNS)	PP(into,P)	P(into,P)	→	into
PP(into,P)	→	P(into,P)	NP(bin,NN)		DT(a,DT)	→	a
NP(bin,NN)	→	DT(a,DT)	NN(bin,NN)		NN(bin,NN)	→	bin

Figure 14.10 A lexicalized tree, including head tags, for a WSJ sentence, adapted from Collins (1999). Below we show the PCFG rules that would be needed for this parse tree, internal rules on the left, and lexical rules on the right.

A natural way to think of a lexicalized grammar is like parent annotation, i.e. as a simple context-free grammar with many copies of each rule, one copy for each possible headword/head tag for each constituent. Thinking of a probabilistic lexicalized CFG in this way would lead to the set of simple PCFG rules shown below the tree in Fig. 14.10.

Lexical rules
Internal rule

Note that Fig. 14.10 shows two kinds of rules: **lexical rules**, which express the expansion of a preterminal to a word, and **internal rules**, which express the other rule expansions. We need to distinguish these kinds of rules in a lexicalized grammar because they are associated with very different kinds of probabilities. The lexical rules are deterministic, i.e., have probability 1.0, since a lexicalized preterminal like $NN(bin,NN)$ can only expand to the word *bin*. But for the internal rules we will need to estimate probabilities.

Suppose we were to treat a probabilistic lexicalized CFG like a really big CFG that just happened to have lots of very complex non-terminals and estimate the probabilities for each rule from maximum likelihood estimates. Thus, using Eq. 14.18, the MLE estimate for the probability for the rule *P(VP(dumped,VBD) → VBD(dumped, VBD) NP(sacks,NNS) PP(into,P))* would be:

(14.23)
$$\frac{Count(VP(dumped,VBD) \rightarrow VBD(dumped,VBD)NP(sacks,NNS)PP(into,P))}{Count(VP(dumped,VBD))}$$

But there's no way we can get good estimates of counts like those in (14.23), because they are so specific: we're very unlikely to see many (or even any) instances of a sentence with a verb phrase headed by *dumped* that has one NP argument headed by *sacks* and a PP argument headed by *into*. In other words, counts of fully lexicalized PCFG rules like this will be far too sparse and most rule probabilities will come out zero.

The idea of lexicalized parsing is to make some further independence assumptions to break down each rule, so that we would estimate the probability

$$P(VP(dumped,VBD) \rightarrow VBD(dumped,VBD)\, NP(sacks,NNS)\, PP(into,P))$$
(14.24)

as the product of smaller independent probability estimates for which we could acquire reasonable counts. The next section summarizes one such method, the Collins parsing method.

14.6.1 The Collins Parser

Modern statistical parsers differ in exactly which independence assumptions they make. In this section we describe a simplified version of Collins's (1999) Model 1, but there are a number of other parsers that are worth knowing about; see the summary at the end of the chapter.

The first intuition of the Collins parser is to think of the right-hand side of every (internal) CFG rule as consisting of a head non-terminal, together with the non-terminals to the left of the head, and the non-terminals to the right of the head. In the abstract, we think about these rules as follows:

(14.25)
$$LHS \rightarrow L_n\, L_{n-1} \ldots L_1\, H\, R_1 \ldots R_{n-1}\, R_n$$

Since this is a lexicalized grammar, each of the symbols like L_1 or R_3 or H or LHS is actually a complex symbol representing the category and its head and head tag, like *VP(dumped,VP)* or *NP(sacks,NNS)*.

Now instead of computing a single MLE probability for this rule, we are going to break down this rule via a neat generative story, a slight simplification of what is called Collins Model 1. This new generative story is that given the left-hand side, we first generate the head of the rule, and then generate the dependents of the head, one by one, from the inside out. Each of these generation steps will have its own probability.

We are also going to add a special STOP non-terminal at the left and right edges of the rule; this non-terminal will allow the model to know when to stop generating dependents on a given side. We'll generate dependents on the left side of the head until we've generated STOP on the left side of the head, at which point we move to the right side of the head and start generating dependents there until we generate STOP. So it's as if we are generating a rule augmented as follows:

(14.26) $$P(VP(dumped,VBD) \rightarrow \text{STOP } VBD(dumped,VBD)\ NP(sacks,NNS)\ PP(into,P) \text{ STOP}$$

Let's see the generative story for this augmented rule. We're going to make use of three kinds of probabilities: P_H for generating heads, P_L for generating dependents on the left, and P_R for generating dependents on the right.

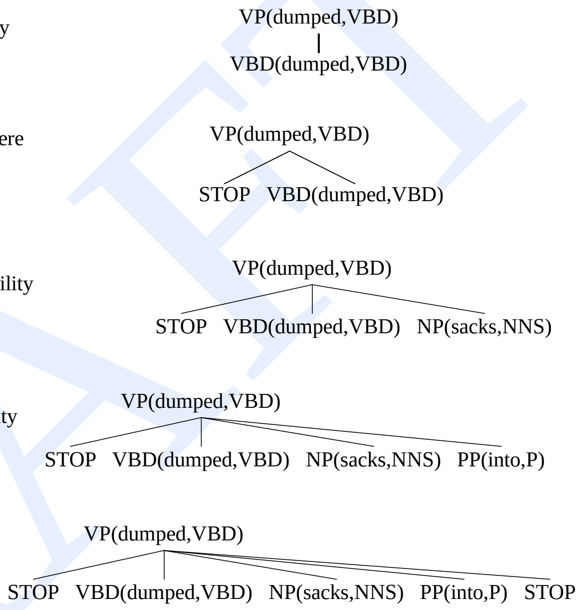
1) First generate the head VBD(dumped,VBD) with probability P(H|LHS) = P(VBD(dumped,VBD) | VP(dumped,VBD))

2) Then generate the left dependent (which is STOP, since there isn't one) with probability P(STOP| VP(dumped,VBD) VBD(dumped,VBD)

3) Then generate right dependent NP(sacks,NNS) with probability P_r(NP(sacks,NNS| VP(dumped,VBD), VBD(dumped,VBD))

4) Then generate the right dependent PP(into,P) with probability P_r(PP(into,P) | VP(dumped,VBD), VBD(dumped,VBD))

5) Finally generate the right dependent STOP with probability P_r(STOP | VP(dumped,VBD), VBD(dumped,VBD))



In summary, the probability of this rule:

(14.27) $$P\,(\,VP(dumped,VBD) \rightarrow VBD(dumped,VBD)\ \ NP(sacks,NNS) PP(into,P)\,)$$

is estimated as:

(14.28) $$\begin{aligned} P_H(VBD|VP,dumped) &\times P_L(STOP|VP,VBD,dumped) \\ &\times P_R(NP(sacks,NNS)|VP,VBD,dumped) \\ &\times P_R(PP(into,P)|VP,VBD,dumped) \\ &\times P_R(STOP|VP,VBD,dumped) \end{aligned}$$

Each of these probabilities can be estimated from much smaller amounts of data than the full probability in (14.28). For example, the maximum likelihood estimate for the component probability $P_R(NP(sacks,NNS)|VP,VBD,dumped)$ is:

(14.29) $$\frac{\text{Count}(\ VP(dumped,VBD)\ \text{ with }\ NNS(sacks) \text{as a daughter somewhere on the right}\ )}{\text{Count}(\ VP(dumped,VBD)\ )}$$

These counts are much less subject to sparsity problems than complex counts like those in (14.28).

More generally, if we use h to mean a headword together with its tag, l to mean a word+tag on the left and r to mean a word+tag on the right, the probability of an entire rule can be expressed as:

1. Generate the head of the phrase $H(hw,ht)$ with probability $P_H(H(hw,ht)|P,hw,ht)$
2. Generate modifiers to the left of the head with total probability:

   $$\prod_{i=1}^{n+1} P_L(L_i(lw_i,lt_i)|P,H,hw,ht)$$

   such that $L_{n+1}(lw_{n+1},lt_{n+1})=$STOP, and we stop generating once we've generated a STOP token.
3. Generate modifiers to the right of the head with total probability:

   $$\prod_{i=1}^{n+1} P_P(R_i(rw_i,rt_i)|P,H,hw,ht)$$

   such that $R_{n+1}(rw_{n+1},rt_{n+1}) = STOP$, and we stop generating once we've generated a STOP token.

14.6.2 Advanced: Further Details of the Collins Parser

The actual Collins parser models are more complex (in a couple of ways) than the simple model presented in the previous section. Collins Model 1 includes a **distance** feature. Thus instead of computing P_L and P_R as follows:

Distance

$$P_L(L_i(lw_i,lt_i)|P,H,hw,ht) \tag{14.30}$$

$$P_R(R_i(rw_i,rt_i)|P,H,hw,ht) \tag{14.31}$$

Collins Model 1 conditions also on a distance feature:

$$P_L(L_i(lw_i,lt_i)|P,H,hw,ht,distance_L(i-1)) \tag{14.32}$$

$$P_R(R_i(rw_i,rt_i)|P,H,hw,ht,distance_R(i-1)) \tag{14.33}$$

The distance measure is a function of the sequence of words *below* the previous modifiers (i.e. the words which are the yield of each modifier non-terminal we have already generated on the left). Fig. 14.11, adapted from Collins (2003) shows the computation of the probability $P(R_2(rh_2,rt_2)|P,H,hw,ht,distance_R(1))$:

The simplest version of this distance measure is just a tuple of two binary features based on the surface string below these previous dependencies: (1) is the string of length zero? (i.e. were no previous words generated?) (2) does the string contain a verb?

Collins Model 2 adds more sophisticated features, conditioning on subcategorization frames for each verb, and distinguishing arguments from adjuncts.

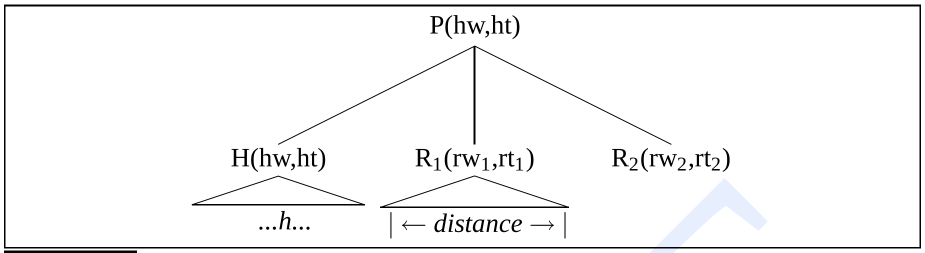


Figure 14.11 The next child R_2 is generated with probability $P(R_2(rh_2,rt_2)|P,H,hw,ht,distance_R(1))$. The distance is the yield of the previous dependent nonterminal R_1. Had there been another intervening dependent, its yield would have been included as well. Adapted from Collins (2003).

Finally, smoothing is as important for statistical parsers as it was for N-gram models. This is particularly true for lexicalized parsers, since (even using the Collins or other methods of independence assumptions) the lexicalized rules will otherwise condition on many lexical items that may never occur in training.

Consider the probability $P_R(R_i(rw_i,rt_i)|P,hw,ht)$. What do we do if a particular right-hand side constituent never occurs with this head? The Collins model addresses this problem by interpolating three backed-off models: fully lexicalized (conditioning on the headword), backing off to just the head tag, and altogether unlexicalized:

Backoff Level	$P_R(R_i(rw_i,rt_i\|...)$	**Example**
1	$P_R(R_i(rw_i,rt_i)\|P,hw,ht)$	P_R(NP(sacks,NNS)|VP, VBD, dumped)
2	$P_R(R_i(rw_i,rt_i)\|P,ht)$	$P_R(NP(sacks,NNS)\|VP,VBD)$
3	$P_R(R_i(rw_i,rt_i)\|P)$	$P_R(NP(sacks,NNS)\|VP)$

Similar backoff models are built also for P_L and P_H. Although we've used the word 'backoff', in fact these are not backoff models but interpolated models. The three models above are linearly interpolated, where e_1, e_2, and e_3 are the maximum likelihood estimates of the three backoff models above:

$$P_R(...) = \lambda_1 e_1 + (1-\lambda_1)(\lambda_2 e_2 + (1-\lambda_2)e_3) \quad (14.34)$$

The values of $\lambda_1 and \lambda_2$ are set to implement Witten-Bell discounting (Witten and Bell, 1991) following Bikel et al. (1997).

Unknown words are dealt with in the Collins model by replacing any unknown word in the test set, and any word occurring less than 6 times in the training set, with a special UNKNOWN word token. Unknown words in the test set are assigned a part-of-speech tag in a preprocessing step by the Ratnaparkhi (1996) tagger; all other words are tagged as part of the parsing process.

The parsing algorithm for the Collins model is an extension of probabilistic CKY; see Collins (2003). Extending the CKY algorithm to handle basic lexicalized probabilities is left as an exercise for the reader.

14.7 Evaluating Parsers

The standard techniques for evaluating parsers and grammars are called the PARSEVAL measures, and were proposed by Black et al. (1991) based on the same ideas from signal-detection theory that we saw in earlier chapters. The intuition of the PARSEVAL metric is to measure how much the **constituents** in the hypothesis parse tree look like the constituents in a hand-labeled gold reference parse. PARSEVAL thus assumes we have a human-labeled "gold standard" parse tree for each sentence in the test set; we generally draw these gold standard parses from a treebank like the Penn Treebank.

Given these gold standard reference parses for a test set, a given constituent in a hypothesis parse C_h of a sentence s is labeled "correct" if there is a constituent in the reference parse C_r with the same starting point, ending point, and non-terminal symbol.

We can then measure the precision and recall just as we did for chunking in the previous chapter.

$$\textbf{labeled recall:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of correct constituents in reference parse of } s}$$

$$\textbf{labeled precision:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of total constituents in hypothesis parse of } s}$$

As with other uses of precision and recall, instead of reporting them separately, we often report a single number, the **F-measure** (van Rijsbergen, 1975): The F-measure is defined as:

F-measure

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

The β parameter is used to differentially weight the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is sometimes called $F_{\beta=1}$ or just F_1:

(14.35) $$F_1 = \frac{2PR}{P+R}$$

The F-measure derives from a weighted harmonic mean of precision and recall. Recall that the harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of the reciprocals:

(14.36) $$\text{HarmonicMean}(a_1, a_2, a_3, a_4, ..., a_n) = \frac{n}{\frac{1}{a_1}\frac{1}{a_2}\frac{1}{a_3}\cdots\frac{1}{a_n}}$$

and hence F-measure is

(14.37) $$F = \frac{1}{\frac{1}{\alpha P} \times \frac{1}{(1-\alpha)R}} \quad \text{or} \left(\text{with } \beta^2 = \frac{1-\alpha}{\alpha}\right) \quad F = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

We additionally use a new metric, crossing brackets, for each sentence s:

cross-brackets: the number of constituents for which the reference parse has a bracketing such as ((A B) C) but the hypothesis parse has a bracketing such as (A (B C)).

As of the time of this writing, the performance of modern parsers that are trained and tested on the Wall Street Journal treebank is somewhat higher than 90% recall, 90% precision, and about 1% cross-bracketed constituents per sentence.

For comparing parsers which use different grammars, the PARSEVAL metric includes a canonicalization algorithm for removing information likely to be grammar-specific (auxiliaries, pre-infinitival "to", etc.) and for computing a simplified score. The interested reader should see Black et al. (1991). The canonical publicly-available implementation of the PARSEVAL metrics is called `evalb` (Sekine and Collins, 1997).

evalb

You might wonder why we don't evaluate parsers by measuring how many *sentences* are parsed correctly, instead of measuring *constituent* accuracy. The reason we use constituents is that measuring constituents gives us a more fine-grained metric. This is especially true for long sentences, where most parsers don't get a perfect parse. If we just measured sentence accuracy, we wouldn't be able to distinguish between a parse that got most of the constituents wrong, and one that just got one constituent wrong.

Nonetheless, constituents are not always an optimal domain for parser evaluation. For example, using the PARSEVAL metrics requires that our parser produce trees in the exact same format as the gold standard. That means that if we want to evaluate a parser which produces different styles of parses (dependency parses, or LFG feature-structures, etc.) against say the Penn Treebank (or against another parser which produces Treebank format), we need to map the output parses into Treebank format. A related problem is that constituency may not be the level we care the most about. We might be more interested in how well the parser does at recovering grammatical dependencies (subject, object, etc), which could give us a better metric for how useful the parses would be to semantic understanding. For these purposes we can use alternative evaluation metrics based on measuring the precision and recall of labeled dependencies, where the labels indicate the grammatical relations (Lin, 1995; Carroll et al., 1998; Collins et al., 1999). Kaplan et al. (2004), for example, compared the Collins (1999) parser with the Xerox XLE parser (Riezler et al., 2002), which produces much richer semantic representations, by converting both parse trees to a dependency representation.

14.8 Advanced: Discriminative Reranking

The models we have seen of parsing so far, the PCFG parser and the Collins lexicalized parser, are generative parsers. By this we mean that the probabilistic model implemented in these parsers gives us the probability of generating a particular sentence by assigning a probability to each choice the parser could make in this generation procedure.

Generative models have some significant advantages; they are easy to train using maximum likelihood and they give us an explicit model of how different sources of evidence are combined. But generative parsing models also make it hard to incorporate arbitrary kinds of information into the probability model. This is because the probability is based on the generative derivation of a sentence; it is difficult to add features that are not local to a particular PCFG rule.

Consider for example how to represent global facts about tree structure. Parse trees in English tend to be right-branching; we'd therefore like our model to assign a higher probability to a tree which is more right-branching, all else being equal. It is also the case that heavy constituents (those with a large number of words) tend to appear later in the sentence. Or we might want to condition our parse probabilities on global facts like the identity of the speaker (perhaps some speakers are more likely to use complex relative clauses, or use the passive). Or we might want to condition on complex discourse factors across sentences. None of these kinds of global factors is trivial to incorporate into the generative models we have been considering. A simplistic model that for example makes each non-terminal dependent on how right-branching the tree is in the parse so far, or makes each NP non-terminal sensitive to the number of relative clauses the speaker or writer used in previous sentences, would result in counts that are far too sparse.

We discussed this problem in Ch. 6, where the need for these kinds of global features motivated the use of log-linear (MEMM) models for POS tagging instead of HMMs. For parsing, there are two broad classes of discriminative models: dynamic programming approaches and two-stage models of parsing that use discriminative reranking. We'll discuss discriminative reranking in the rest of this section; see the end of the chapter for pointers to discriminative dynamic programming approaches.

In the first stage of a discriminative reranking system, we can run a normal statistical parser of the type we've described so far. But instead of just producing the single best parse, we modify the parser to produce a ranked list of parses together with their probabilities. We call this ranked list of *N* parses the ***N*-best list** (the *N*-best list was first introduced in Ch. 9 when discussing multiple-pass decoding models for speech recognition). There are various ways to modify statistical parsers to produce an *N*-best list of parses; see the end of the chapter for pointers to the literature. For each sentence in the training set and the test set, we run this *N*-best parser and produce a set of *N* parse/probability pairs.

N-best list

The second stage of a discriminative reranking model is a classifier which takes each of these sentences with their *N* parse/probability pairs as input, extracts some large set of features and chooses the single best parse from the *N*-best list. We can use any type of classifier for the reranking, such as the log-linear classifiers introduced in Ch. 6.

A wide variety of features can be used for reranking. One important feature to include is the parse probability assigned by the first-stage statistical parser. Other features might include each of the CFG rules in the tree, the number of parallel conjuncts, how heavy each constituent is, measures of how right-branching the parse tree is, how many times various tree fragments occur, bigrams of adjacent non-terminals in the tree, and so on.

The two-stage architecture has a weakness: the accuracy rate of the complete architecture can never be better than the accuracy rate of the best parse in the first-stage N-best list. This is because the reranking approach is merely choosing one of the N-best parses; even if we picked the very best parse in the list, we can't get 100% accuracy if the correct parse isn't in the list! Therefore it is important to consider the ceiling **oracle accuracy** (often measured in F-measure) of the N-best list. The oracle accuracy (F-measure) of a particular N-best list is the accuracy (F-measure) we get if we chose the parse that had the highest accuracy. We call this an **oracle** accuracy because it relies on perfect knowledge (as if from an oracle) of which parse to pick.[2] Of course it only makes sense to implement discriminative reranking if the N-best F-measure is higher than the 1-best F-measure. Luckily this is often the case; for example the Charniak (2000) parser has an F-measure of 0.897 on section 23 of the Penn Treebank, but the Charniak and Johnson (2005) algorithm for producing the 50-best parses has a much higher oracle F-measure of 0.968.

Oracle accuracy

14.9 Advanced: Parser-Based Language Modeling

We said earlier that statistical parsers can take advantage of longer-distance information than N-grams, which suggests that they might do a better job at language modeling/word prediction. It turns out that if we have a very large amount of training data, a 4-gram or 5-gram grammar is nonetheless still the best way to do language modeling. But in situations where there is not enough data for such huge models, parser-based language models are beginning to be developed which have higher accuracy N-gram models.

Two common applications for language modeling are speech recognition and machine translation. The simplest way to use a statistical parser for language modeling for either of these applications is via a two-stage algorithm of the type discussed in the previous section and in Sec. 10.1. In the first stage, we run a normal speech recognition decoder, or machine translation decoder, using a normal N-gram grammar. But instead of just producing the single best transcription or translation sentence, we modify the decoder to produce a ranked N-best list of transcriptions/translations sentences, each one together with its probability (or, alternatively, a lattice).

Then in the second stage, we run our statistical parser and assign a parse probability to each sentence in the N-best list or lattice. We then rerank the sentences based on this parse probability and choose the single best sentence. This algorithm can work better than using a simple trigram grammar. For example, on the task of recognizing spoken sentences from the Wall Street Journal using this two-stage architecture, the probabilities assigned by the Charniak (2001) parser improved the word error rate by about 2 percent absolute, over a simple trigram grammar computed on 40 million words (Hall and Johnson, 2003). We can either use the parse probabilities assigned by the parser as-is, or we can linearly combine it with the original N-gram probability.

An alternative to the two-pass architecture, at least for speech recognition, is to modify the parser to run strictly left-to-right, so that it can incrementally give the proba-

[2] We introduced this same oracle idea in Ch. 9 when we talked about the **lattice error rate**.

bility of the next word in the sentence. This would allow the parser to be fit directly into the first-pass decoding pass and obviate the second-pass altogether. While a number of such left-to-right parser-based language modeling algorithms exist (Stolcke, 1995; Jurafsky et al., 1995; Roark, 2001; Xu et al., 2002), it is fair to say that it is still early days for the field of parser-based statistical language models.

14.10 Human Parsing

Human sentence processing

Are the kinds of probabilistic parsing models we have been discussing also used by humans when they are parsing? This question lies in a field called **human sentence processing**? Recent studies suggest that there are at least two ways in which humans apply probabilistic parsing algorithms, although there is still disagreement on the details.

Reading time

One family of studies has shown that when humans read, the predictability of a word seems to influence the **reading time**; more predictable words are read more quickly. One way of defining predictability is from simple bigram measures. For example, Scott and Shillcock (2003) had participants read sentences while monitoring their gaze with an **eye-tracker**. They constructed the sentences so that some would have a verb-noun pair with a high bigram probability (such as (14.38a)) and others a verb-noun pair with a low bigram probability (such as (14.38b)).

(14.38) a) **HIGH PROB:** One way to **avoid confusion** is to make the changes during vacation;

b) **LOW PROB:** One way to **avoid discovery** is to make the changes during vacation

They found that the higher the bigram predictability of a word, the shorter the time that participants looked at the word (the **initial-fixation duration**).

While this result only provides evidence for N-gram probabilities, more recent experiments have suggested that the probability of an upcoming word given the syntactic parse of the preceding sentence prefix also predicts word reading time Hale (2006), Levy (2007).

Interestingly, this effect of probability on reading time has also been shown for morphological structure; the time to recognize a word is influenced by entropy of the word and the entropy of the word's morphological paradigm Moscoso del Prado Martín et al. (2004b).

Garden-path

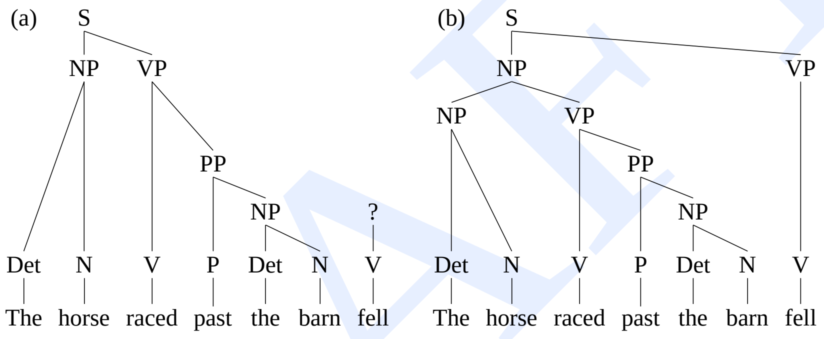
The second family of studies has examined how humans disambiguate sentences which have multiple possible parses, suggesting that humans prefer whichever parse is more probable. These studies often rely on a specific class of temporarily ambiguous sentences called **garden-path** sentences. These sentences, first described by Bever (1970), are sentences which are cleverly constructed to have three properties that combine to make them very difficult for people to parse:

1. They are **temporarily ambiguous**: The sentence is unambiguous, but its initial portion is ambiguous.

2. One of the two or more parses in the initial portion is somehow preferable to the human parsing mechanism.
3. But the dispreferred parse is the correct one for the sentence.

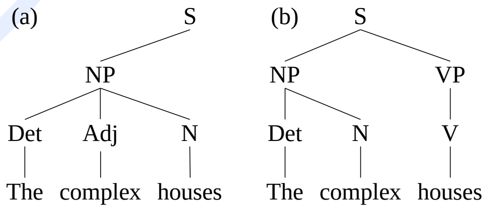
The result of these three properties is that people are "led down the garden path" toward the incorrect parse, and then are confused when they realize it's the wrong one. Sometimes this confusion is quite conscious, as in Bever's example (14.39); in fact this sentence is so hard to parse that readers often need to be shown the correct structure. In the correct structure *raced* is part of a reduced relative clause modifying *The horse*, and means "The horse [which was raced past the barn] fell"; this structure is also present in the sentence "Students taught by the Berlitz method do worse when they get to France".

(14.39) The horse raced past the barn fell.

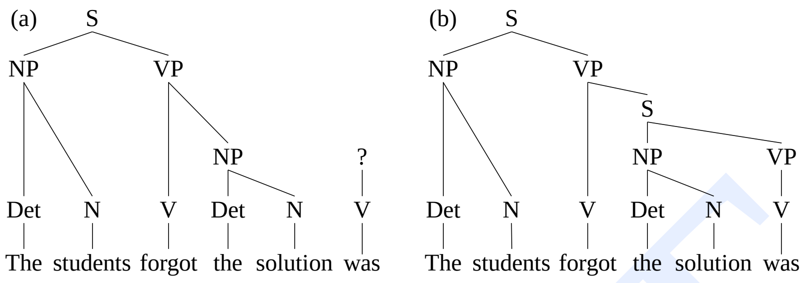

(a)
S
NP
VP
PP
NP
?
Det N V P Det N V
The horse raced past the barn fell
(b)
S
NP
VP
NP
VP
PP
NP
Det N V P Det N V
The horse raced past the barn fell

In Marti Hearst's example (14.40), subjects often misparse the verb *houses* as a noun (analyzing *the complex houses* as a noun phrase, rather than a noun phrase and a verb). Other times the confusion caused by a garden-path sentence is so subtle that it can only be measured by a slight increase in reading time. Thus in example (14.41) readers often mis-parse *the solution* as the direct object of *forgot* rather than as the subject of an embedded sentence. This mis-parse is subtle, and is only noticeable because experimental participants take longer to read the word *was* than in control sentences. This "mini-garden-path" effect at the word *was* suggests that subjects had chosen the direct object parse and had to re-analyze or rearrange their parse now that they realize they are in a sentential complement.

(14.40) The complex houses married and single students and their families.



(14.41) The student forgot the solution was in the back of the book.


(a)
S
NP
VP
NP
?
Det
N
V
Det
N
V
The students forgot the solution was
(b)
S
NP
VP
S
NP
VP
Det
N
V
Det
N
V
The students forgot the solution was

While many factors seem to play a role in these preferences for a particular (incorrect) parse, at least one factor seems to be syntactic probabilities, especially lexicalized (subcategorization) probabilities. For example, the probability of the verb *forgot* taking a direct object ($VP \rightarrow V\ NP$) is higher than the probability of it taking a sentential complement ($VP \rightarrow V\ S$); this difference causes readers to expect a direct object after *forget* and be surprised (longer reading times) when they encounter a sentential complement. By contrast, a verb which prefers a sentential complement (like *hope*) didn't cause extra reading time at *was*.

Similarly, the garden path in (14.40) may be caused by the fact that $P(houses|Noun) > P(houses|Verb)$ and $P(complex|Adjective) > P(complex|Noun)$, and the garden path in (14.39) at least partially by the low probability of the reduced relative clause construction.

Besides grammatical knowledge, human parsing is affected by many other factors which we will describe later, including resource constraints (such as memory limitations, to be discussed in Ch. 15), thematic structure (such as whether a verb expects semantic *agents* or *patients*, to be discussed in Ch. 19) and discourse constraints (Ch. 21).

14.11 Summary

This chapter has sketched the basics of **probabilistic** parsing, concentrating on **probabilistic context-free grammars** and **probabilistic lexicalized context-free grammars**.

- Probabilistic grammars assign a probability to a sentence or string of words, while attempting to capture more sophisticated syntactic information than the N-gram grammars of Ch. 4.
- A **probabilistic context-free grammar** (**PCFG**) is a context-free grammar in which every rule is annotated with the probability of choosing that rule. Each PCFG rule is treated as if it were **conditionally independent**; thus the probability of a sentence is computed by **multiplying** the probabilities of each rule in the parse of the sentence.
- The probabilistic CKY (**Cocke-Kasami-Younger**) algorithm is a probabilistic version of the CKY parsing algorithm. There are also probabilistic versions of other parsers like the Earley algorithm.

- PCFG probabilities can be learning by counting in a **parsed corpus**, or by parsing a corpus. The **inside-outside** algorithm is a way of dealing with the fact that the sentences being parsed are ambiguous.
- Raw PCFGs suffer from poor independence assumptions between rules and lack of sensitivity to lexical dependencies.
- One way to deal with this problem is to split and merge non-terminals (automatically or by hand).
- **Probabilistic lexicalized CFG**s are another solution to this problem in which the basic PCFG model is augmented with a **lexical head** for each rule. The probability of a rule can then be conditioned on the lexical head or nearby heads.
- Parsers for lexicalized PCFGs (like the Charniak and Collins parsers) are based on extensions to probabilistic CKY parsing.
- Parsers are evaluated using three metrics: **labeled recall**, **labeled precision**, and **cross-brackets**.
- There is evidence based on **garden-path sentences** and other on-line sentence-processing experiments that the human parser uses some kinds of probabilistic information about grammar.

Bibliographical and Historical Notes

Many of the formal properties of probabilistic context-free grammars were first worked out by Booth (1969) and Salomaa (1969). Baker (1979) proposed the inside-outside algorithm for unsupervised training of PCFG probabilities, and used a CKY-style parsing algorithm to compute inside probabilities. Jelinek and Lafferty (1991) extended the CKY algorithm to compute probabilities for prefixes. Stolcke (1995) drew on both of these algorithms in adapting the Earley algorithm to use with PCFGs.

A number of researchers starting in the early 1990s worked on adding lexical dependencies to PCFGs, and on making PCFG rule probabilities more sensitive to surrounding syntactic structure. For example Schabes et al. (1988) and Schabes (1990) presented early work on the use of heads. Many papers on the use of lexical dependencies were first presented at the DARPA Speech and Natural Language Workshop in June, 1990. A paper by Hindle and Rooth (1990) applied lexical dependencies to the problem of attaching prepositional phrases; in the question session to a later paper Ken Church suggested applying this method to full parsing (Marcus, 1990). Early work on such probabilistic CFG parsing augmented with probabilistic dependency information includes Magerman and Marcus (1991), Black et al. (1992), Bod (1993), and Jelinek et al. (1994), in addition to Collins (1996), Charniak (1997), and Collins (1999) discussed above. Other recent PCFG parsing models include Klein and Manning (2003a) and Petrov et al. (2006). .

This early lexical probabilistic work led initially to work focused on solving specific parsing problems like preposition-phrase attachment, using methods including Transformation Based Learning (TBL) (Brill and Resnik, 1994), Maximum Entropy

(Ratnaparkhi et al., 1994), Memory-Based Learning (Zavrel and Daelemans, 1997), log-linear models (Franz, 1997), decision trees using semantic distance between heads (computed from WordNet) (Stetina and Nagao, 1997), and Boosting (Abney et al., 1999b).

Another direction extended the lexical probabilistic parsing work to build probabilistic formulations of grammar other than PCFGs, such as probabilistic TAG grammar (Resnik, 1992; Schabes, 1992), based on the TAG grammars discussed in Ch. 12, probabilistic LR parsing (Briscoe and Carroll, 1993), and probabilistic link grammar (Lafferty et al., 1992). An approach to probabilistic parsing called **[**Supertagging**]**supertagging extends the part-of-speech tagging metaphor to parsing by using very complex tags that are in fact fragments of lexicalized parse trees (Bangalore and Joshi, 1999; Joshi and Srinivas, 1994), based on the lexicalized TAG grammars of Schabes et al. (1988). For example the noun *purchase* would have a different tag as the first noun in a noun compound (where it might be on the left of a small tree dominated by Nominal) than as the second noun (where it might be on the right). Supertagging has also been applied to CCG parsing and HPSG parsing (Clark and Curran, 2004a; Matsuzaki et al., 2007; Blunsom and Baldwin, 2006). Non-supertagging statistical parsers for CCG include Hockenmaier and Steedman (2002).

[

Goodman (1997), Abney (1997), and Johnson et al. (1999) gave early discussions of probabilistic treatments of feature-based grammars. Other recent work on building statistical models of feature-based grammar formalisms like HPSG and LFG includes Riezler et al. (2002), Kaplan et al. (2004), and Toutanova et al. (2005).

We mentioned earlier that discriminative approaches to parsing fall into the two broad categories of dynamic programming methods and discriminative reranking methods. Recall that discriminative reranking approaches require N-best parses. Parsers based on A* search can easily be modified to generate N-best lists just by continuing the search past the first-best parse (Roark, 2001). Dynamic programming algorithms like the ones described in this chapter can be modified by eliminating the dynamic programming and using heavy pruning (Collins, 2000; Collins and Koo, 2005; Bikel, 2004), or via new algorithms (Jiménez and Marzal, 2000; Gildea and Jurafsky, 2002; Charniak and Johnson, 2005; Huang and Chiang, 2005), some adapted from speech recognition algorithms such as Schwartz and Chow (1990) (see Sec. 10.1).

By contrast, in dynamic programming methods, instead of outputting and then reranking an N-best list, the parses are represented compactly in a chart, and log-linear and other methods are applied for decoding directly from the chart. Such modern methods include Johnson (2001), Clark and Curran (2004b), and Taskar et al. (2004). Other reranking developments include changing the optimization criterion (Titov and Henderson, 2006).

Another important recent area of research is dependency parsing; algorithms include Eisner's bilexical algorithm (Eisner, 1996b, 1996a, 2000a), maximum spanning tree approaches (using on-line learning) (McDonald et al., 2005, 2005), and approaches based on building classifiers for parser actions (Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre et al., 2006; Titov and Henderson, 2007). A distinction is usually made between projective and **non-projective dependencies**. Non-projective dependencies are those in which the dependency lines cross; this is not very common in English, but is very common in many languages with more free word order. Non-

Non-projective dependencies

projective dependency algorithms include McDonald et al. (2005) and Nivre (2007). The Klein-Manning parser combines dependency and constituency information (Klein and Manning, 2003c).

Manning and Schütze (1999) has an extensive coverage of probabilistic parsing. Collins' (1999) dissertation includes a very readable survey of the field and introduction to his parser.

The field of grammar induction is closely related to statistical parsing, and a parser is often used as part of a grammar induction algorithm. One of the earliest statistical works in grammar induction was Horning (1969), who showed that PCFGs could be induced without negative evidence. Early modern probabilistic grammar work showed that simply using EM was insufficient (Lari and Young, 1990; Carroll and Charniak, 1992). Recent probabilistic work such as Yuret (1998), Clark (2001), Klein and Manning (2002), and Klein and Manning (2004), are summarized in Klein (2005) and Adriaans and van Zaanen (2004). Work since that summary includes Smith and Eisner (2005), Haghighi and Klein (2006), and Smith and Eisner (2007).

Exercises

14.1 Implement the CKY algorithm.

14.2 Modify the algorithm for conversion to CNF from Ch. 13 to correctly handle rule probabilities. Make sure that the resulting CNF assigns the same total probability to each parse tree.

14.3 Recall that Exercise 3 asked you to update the CKY algorithm to handles unit productions directly rather than converting them to CNF. Extend this change to probabilistic CKY.

14.4 Fill out the rest of the probabilistic CKY chart in Fig. 14.4.

14.5 Sketch out how the CKY algorithm would have to be augmented to handle lexicalized probabilities.

14.6 Implement your lexicalized extension of the CKY algorithm.

14.7 Implement the PARSEVAL metrics described in Sec. 14.7. Next either use a treebank or create your own hand-checked parsed testset. Now use your CFG (or other) parser and grammar and parse the testset and compute labeled recall, labeled precision, and cross-brackets.