CHAPTER

14 Statistical Constituency Parsing

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 models of syntactic constituency structure and its parsing. Here, we show that it is possible to build probabilistic models of syntactic knowledge and efficient probabilistic parsers.

One crucial use of probabilistic parsing is to solve the problem of **disambiguation**. Recall from Chapter 13 that sentences on average tend to be syntactically ambiguous because of phenomena like **coordination ambiguity** and **attachment ambiguity**. The CKY parsing algorithm can represent these ambiguities in an efficient way but is 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. The most commonly used probabilistic constituency grammar formalism 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 Treebank grammars and how they can be parsed with a probabilistic version of the **CKY algorithm** of Chapter 13.

We then show a number of ways that we can improve on this basic probability model (PCFGs trained on Treebank grammars), such as by modifying the set of non-terminals (making them either more specific or more general), or adding more sophisticated conditioning factors like subcategorization or dependencies. Heavily lexicalized grammar formalisms such as Lexical-Functional Grammar (LFG) (Bresnan, 1982), Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994), Tree-Adjoining Grammar (TAG) (Joshi, 1985), and Combinatory Categorial Grammar (CCG) pose additional problems for probabilistic parsers. Section 14.7 introduces the task of **supertagging** and the use of heuristic search methods based on the **A* algorithm** in the context of CCG parsing.

Finally, we describe the standard techniques and metrics for evaluating parsers.

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:

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

Figure 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. A real grammar has a great many more rules for each non-terminal; 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 flight" shown in Fig. 14.2. The sensible

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* [.05] ∣ *money* [.05]
NP → *Pronoun*	[.35]	∣ *flight* [.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 that is a probabilistic augmentation of the $\mathscr{L}_1$ miniature English CFG grammar and lexicon of Fig. **??**. These probabilities were made up for pedagogical purposes and are not based on a corpus (since any real corpus would have many more rules, so the true probabilities of each rule would be much smaller).

parse on the left means "Book a flight that serves dinner". The nonsensical parse on the right, however, would have to mean something like "Book a flight on behalf of 'the dinner'" just as a structurally similar sentence like "Can you book John a flight?" means something like "Can you book a flight 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:

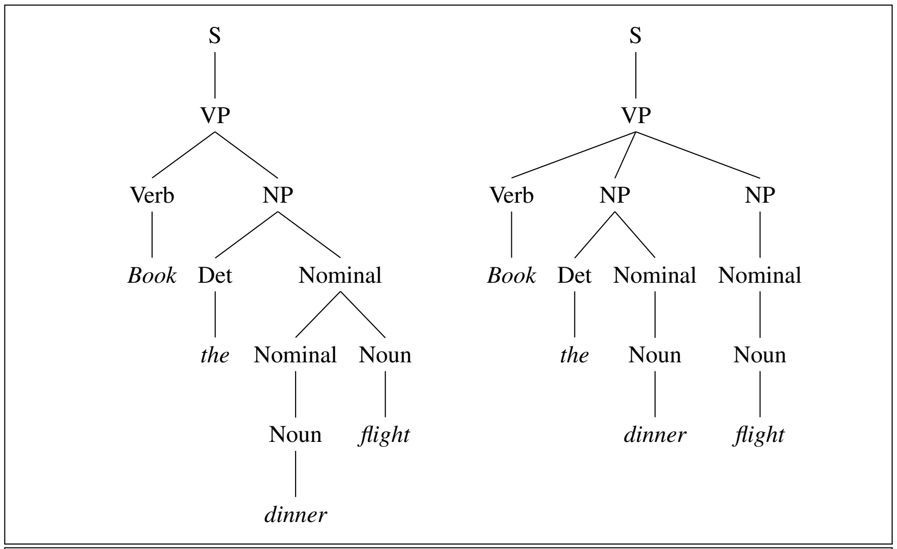
$$P(T,S) = P(T)P(S|T) \tag{14.3}$$

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

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

We can compute the probability of each of the trees in Fig. 14.2 by multiplying 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:

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



Rules			P	Rules			P
S	$\rightarrow$	VP	.05	S	$\rightarrow$	VP	.05
VP	$\rightarrow$	Verb NP	.20	VP	$\rightarrow$	Verb NP NP	.10
NP	$\rightarrow$	Det Nominal	.20	NP	$\rightarrow$	Det Nominal	.20
Nominal	$\rightarrow$	Nominal Noun	.20	NP	$\rightarrow$	Nominal	.15
Nominal	$\rightarrow$	Noun	.75	Nominal	$\rightarrow$	Noun	.75
				Nominal	$\rightarrow$	Noun	.75
Verb	$\rightarrow$	book	.30	Verb	$\rightarrow$	book	.30
Det	$\rightarrow$	the	.60	Det	$\rightarrow$	the	.60
Noun	$\rightarrow$	dinner	.10	Noun	$\rightarrow$	dinner	.10
Noun	$\rightarrow$	flight	.40	Noun	$\rightarrow$	flight	.40

Figure 14.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the sensible meaning "Book a flight that serves dinner", while the parse on the right corresponds to the nonsensical meaning "Book a flight on behalf of 'the dinner' ".

We can see that the left tree in Fig. 14.2 has a much higher probability than the tree on the right. Thus, this parse would correctly be chosen by a disambiguation algorithm that 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 that is most probable given S:

yield

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

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

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

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:

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

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:

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

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, spelling 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:

$$\begin{aligned} P(S) &= \sum_{T s.t. S=\text{yield}(T)} P(T,S) & (14.9) \\ &= \sum_{T s.t. S=\text{yield}(T)} P(T) & (14.10) \end{aligned}$$

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 $w_1, ..., w_{i-1}$. The general formula for this is

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

We saw in Chapter 3 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

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

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 PCFGs allow us to condition on the entire previous context $w_1, w_2, ..., w_{i-1}$ shown in Eq. 14.11.

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, that is,

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

The algorithms for computing the most likely parse are simple extensions of the standard algorithms for parsing; most modern probabilistic parsers are based on the **probabilistic CKY** algorithm, first described by Ney (1991). The probabilistic CKY algorithm assumes the PCFG is in Chomsky normal form. Recall from page **??** that in CNF, the right-hand side of each rule must expand to either two non-terminals or to a single terminal, i.e., rules have the form $A \rightarrow B\ C$, or $A \rightarrow w$.

probabilistic CKY

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 non-terminal that can be placed in this cell, and the value of the cell is then a probability for that non-terminal/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 of type A that spans positions i through j of the input.

Figure 14.3 gives the probabilistic CKY algorithm.

```
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 backpointers used to recover the best parse. The *build_tree* function is left as an exercise to the reader.

Like the basic CKY algorithm in Fig. **??**, the probabilistic CKY algorithm 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 14.2 asks you to modify the algorithm for conversion to CNF in Chapter 13 so that it correctly handles rule probabilities.

In practice, a generalized CKY algorithm that handles unit productions directly is typically used. Recall that Exercise 13.3 asked you to make this change in CKY; Exercise 14.3 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	.40
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 the sentence "The flight includes a meal".

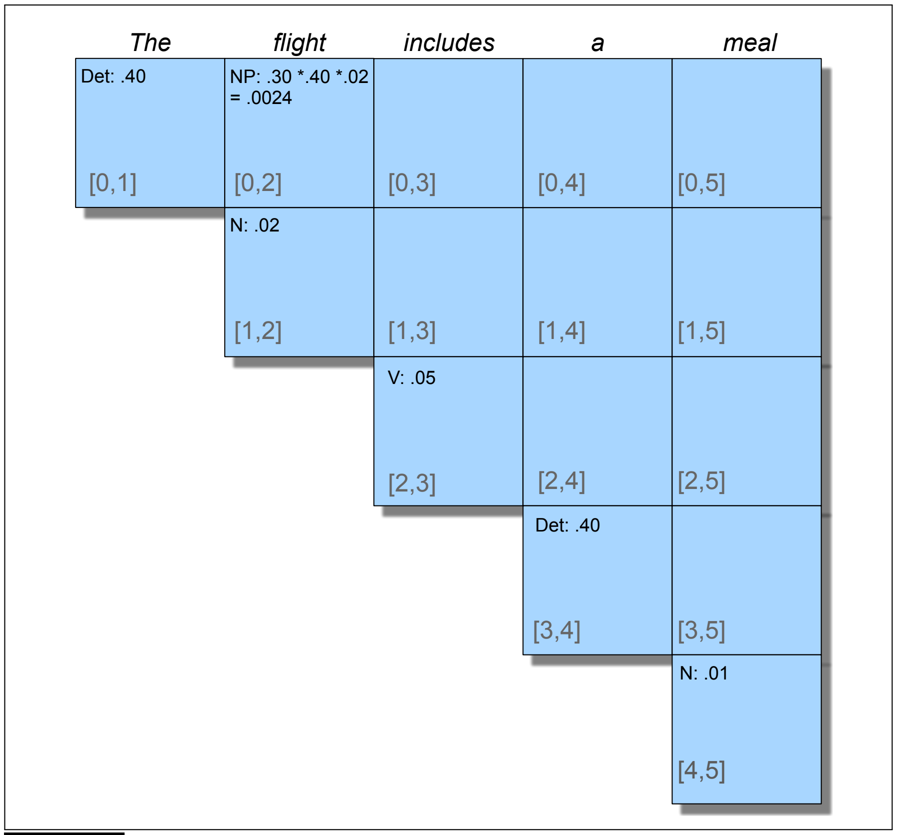


Figure 14.4 The beginning of the probabilistic CKY matrix. Filling out the rest of the chart is left as Exercise 14.4 for the reader.

14.3 Ways to Learn 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 Chapter 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 that is distributed by the Linguistic Data Consortium. Given a treebank, we can compute the probability of each expansion of a non-terminal by counting the number of times that expansion occurs and then normalizing.

$$P(\alpha \to \beta|\alpha) = \frac{\text{Count}(\alpha \to \beta)}{\sum_{\gamma}\text{Count}(\alpha \to \gamma)} = \frac{\text{Count}(\alpha \to \beta)}{\text{Count}(\alpha)} \tag{14.17}$$

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, that is, 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, then parse the sentence, compute a probability for each parse, use these probabilities to weight the counts, re-estimate the rule probabilities, and so on, until our probabilities converge. The standard algorithm for computing this solution is called the **inside-outside** algorithm; it was proposed by Baker (1979) as a generalization of the forward-backward algorithm for HMMs. Like forward-backward, inside-outside is a special case of the Expectation Maximization (EM) algorithm, and hence has two steps: the **expectation step**, and the **maximization step**. See Lari and Young (1990) or Manning and Schütze (1999) for more on the algorithm.

inside-outside

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 that leads to 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, probabilistic constituent 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 introduce some of these augmentations.

14.4.1 Independence Assumptions Miss Rule Dependencies

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, that is, 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 depend on the location of the node in the parse tree. For example, in English it turns out that *NP*s that are syntactic **subjects** are far more likely to be pronouns, and *NP*s 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 *NP*s 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.

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

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. Section 14.5 introduces 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 (e.g., from rules like $V \rightarrow sleep$, $NN \rightarrow book$, etc.).

[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 (Chapter 22).

But it turns out that lexical information is useful in other places in the grammar, such as in resolving prepositional phrase (*PP*) attachment 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:

(14.18) Workers dumped sacks into a bin.

Figure 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**, and the incorrect parse implies **NP attachment**.

VP attachment
NP attachment

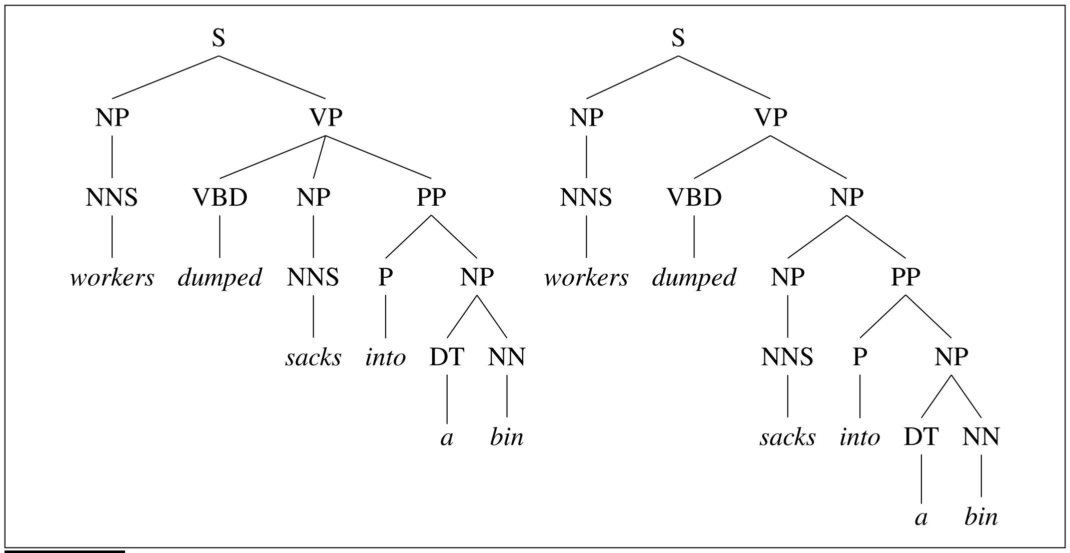


Figure 14.5 Two possible parse trees for a **prepositional phrase attachment ambiguity**. The left parse is the sensible one, in which "into a bin" describes the resulting location of the sacks. In the right incorrect parse, the sacks to be dumped are the ones which are already "into a bin", whatever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

Why doesn't a PCFG already deal with *PP* attachment ambiguities? Note that the two parse trees in Fig. 14.5 have almost exactly the 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, 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, which requires *NP* attachment:

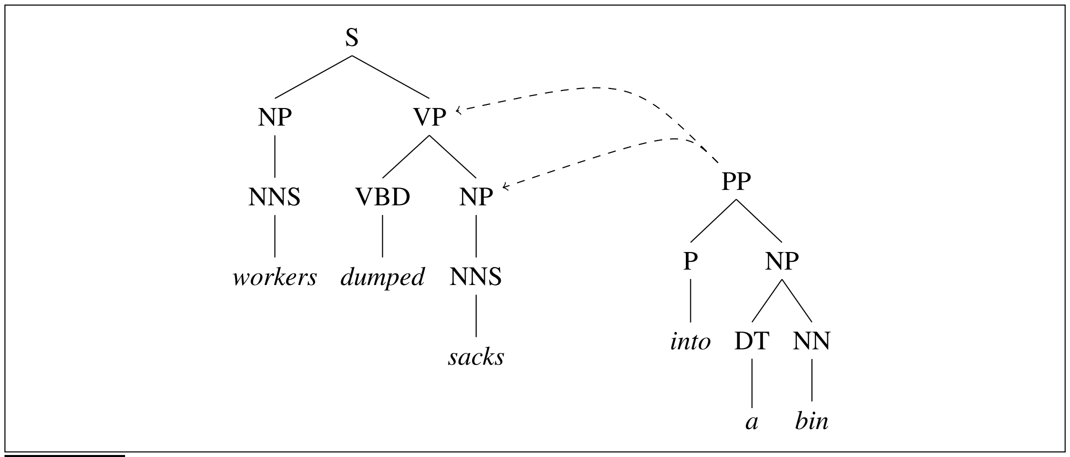


Figure 14.6 Another view of the preposition attachment problem. Should the *PP* on the right attach to the *VP* or *NP* nodes of the partial parse tree on the left?

(14.19) fishermen caught tons of herring

What information in the input sentence lets us know that (14.19) requires *NP* attachment while (14.18) requires *VP* attachment? These preferences come from the identities of the verbs, nouns, and prepositions. 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 (14.19) the affinity between *tons* and *of* is greater than that between *caught* and *of*, leading to *NP* attachment. Thus, to get the correct parse for these kinds of examples, we need a model that somehow augments the PCFG probabilities to deal with these **lexical dependency** statistics for different verbs and prepositions.

lexical dependency

Coordination ambiguities are another case in which lexical dependencies are the key to choosing the proper parse. Figure 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 most 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.

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 Non-Terminals

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, whereas *NP*s 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

split

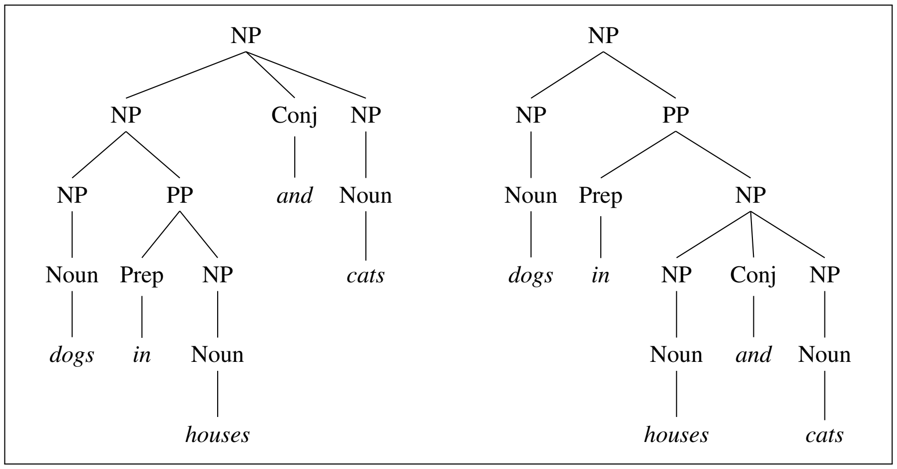


Figure 14.7 An instance of coordination ambiguity. Although the left structure is intuitively the correct one, a PCFG will assign them identical probabilities since both structures use exactly the same set of rules. After Collins (1999).

different probabilities for the rule $NP_{subject} \rightarrow PRP$ and the rule $NP_{object} \rightarrow PRP$.

parent annotation

One way to implement this intuition of splits is to do **parent annotation** (Johnson, 1998), in which we annotate each node with its parent in the parse tree. Thus, an *NP* node that 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*. Figure 14.8 shows an example of a tree produced by a grammar that parent-annotates the phrasal non-terminals (like *NP* and *VP*).

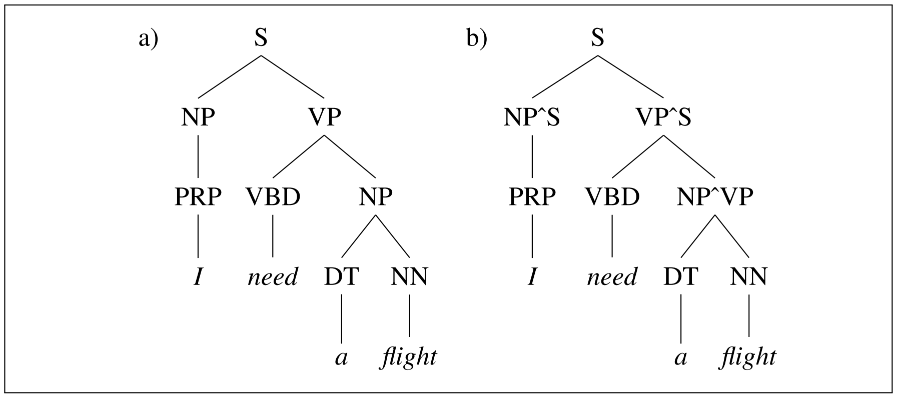


Figure 14.8 A standard PCFG parse tree (a) and one which has **parent annotation** on the nodes which aren't pre-terminal (b). All the non-terminal nodes (except the pre-terminal part-of-speech nodes) in parse (b) have been annotated with the identity of their parent.

In addition to splitting these phrasal nodes, we can also improve a PCFG by splitting the pre-terminal part-of-speech nodes (Klein and Manning, 2003b). For example, different kinds of adverbs (RB) tend to occur in different syntactic positions: the most common adverbs with ADVP parents are *also* and *now*, with *VP* parents *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 can 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 splitting the pre-terminal nodes. Figure 14.9 shows an example from Klein and Manning (2003b) in which even a parent-annotated grammar incorrectly parses *works* as a noun in *to see if advertising works*. Splitting pre-terminals to allow *if* to prefer a sentential complement results in the correct verbal parse.

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 employed handwritten rules to try to find an optimal number of non-terminals (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, alternately splits the non-terminals, and merges non-terminals, finding the set of annotated nodes that maximizes the likelihood of the training set 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 by 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 **Collins parser** (Collins, 1999) and the **Charniak parser** (Charniak, 1997).

lexicalized grammar

We saw in Section **??** 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

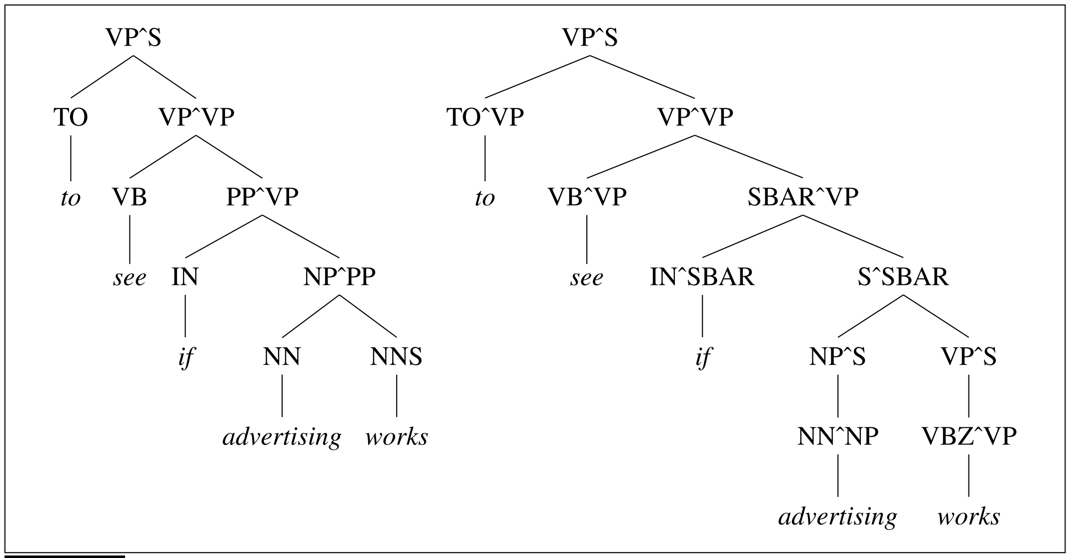


Figure 14.9 An incorrect parse even with a parent-annotated parse (left). The correct parse (right), was produced by a grammar in which the pre-terminal nodes have been split, allowing the probabilistic grammar to capture the fact that *if* prefers sentential complements. Adapted from Klein and Manning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

be extended as

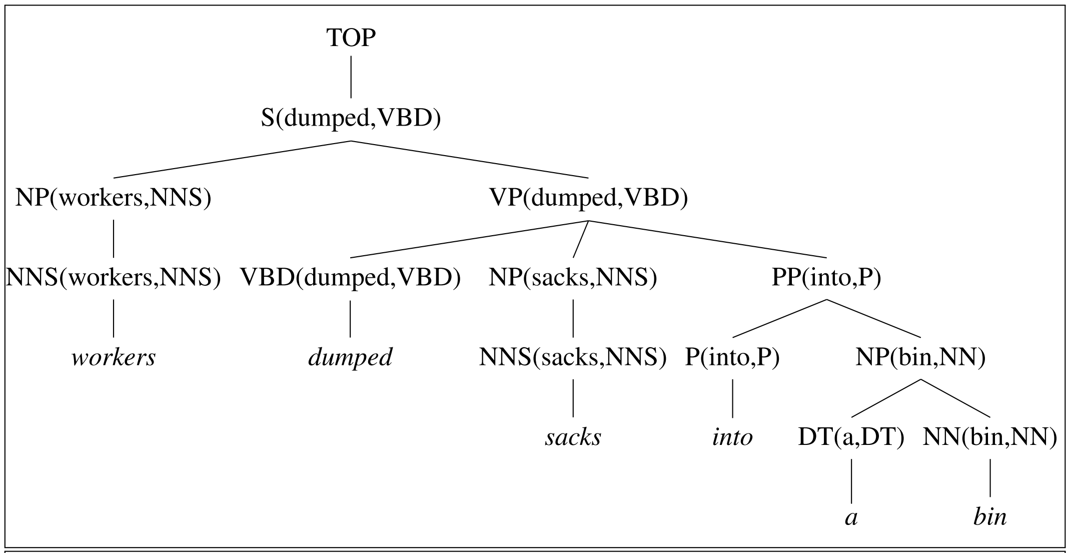
$$VP(dumped) \rightarrow VBD(dumped)\ NP(sacks)\ PP(into) \quad (14.20)$$

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 non-terminal 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

$$VP(dumped,VBD) \rightarrow VBD(dumped,VBD)\ NP(sacks,NNS)\ PP(into,P) \quad (14.21)$$

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



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 needed for this parse tree, internal rules on the left, and lexical rules on the right.

To generate such a lexicalized tree, each PCFG rule must be augmented to identify one right-hand 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. **??** a set of handwritten rules for identifying the heads of particular constituents.

A natural way to think of a lexicalized grammar is as a parent annotation, that is, 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 rules

Note that Fig. 14.10 shows two kinds of rules: **lexical rules**, which express the expansion of a pre-terminal 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, that is, they have probability 1.0 since a lexicalized pre-terminal like $NN(bin, NN)$ can only expand to the word *bin*. But for the internal rules, we 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, according to Eq. 14.17, the MLE estimate for the probability for the rule *P(VP(dumped,VBD)* $\rightarrow$ *VBD(dumped, VBD) NP(sacks,NNS) PP(into,P))* would be

$$\frac{Count(VP(dumped,VBD) \rightarrow VBD(dumped, VBD)\ NP(sacks,NNS)\ PP(into,P))}{Count(VP(dumped,VBD))} \tag{14.22}$$

But there's no way we can get good estimates of counts like those in (14.22) because they are so specific: we're 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 0.

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

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

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

Statistical parsers differ in exactly which independence assumptions they make. Let's look at the assumptions in a simplified version of the Collins parser. 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:

$$LHS \rightarrow L_n\, L_{n-1} \ldots L_1\, H\, R_1 \ldots R_{n-1}\, R_n \tag{14.23}$$

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)*.

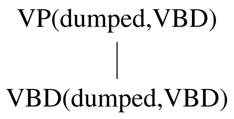
Now, instead of computing a single MLE probability for this rule, we are going to break down this rule via a neat generative story, a slight simplification of what is called Collins Model 1. This new generative story is that given the left-hand side, we first generate the head of the rule and then generate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

We also add a special STOP non-terminal at the left and right edges of the rule; this non-terminal allows the model to know when to stop generating dependents on a given side. We 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:

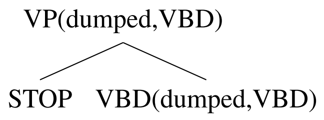
$$P(\textit{VP(dumped,VBD)} \rightarrow \text{STOP } \textit{VBD(dumped, VBD) NP(sacks,NNS) PP(into,P)} \text{ STOP}) \tag{14.24}$$

Let's see the generative story for this augmented rule. We 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.

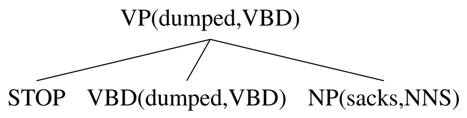
1. Generate the head VBD(dumped,VBD) with probability
P(H|LHS) = P(VBD(dumped,VBD) | VP(dumped,VBD))



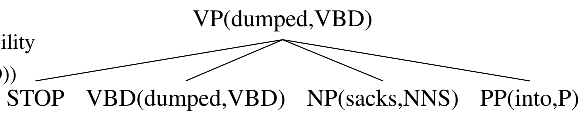
2. Generate the left dependent (which is STOP, since there isn't one) with probability
P(STOP| VP(dumped,VBD) VBD(dumped,VBD))



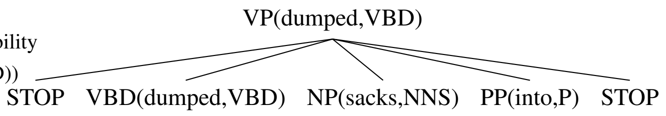
3. Generate right dependent NP(sacks,NNS) with probability
P_r(NP(sacks,NNS| VP(dumped,VBD), VBD(dumped,VBD))



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



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



In summary, the probability of this rule

$$P(\textit{VP(dumped,VBD)} \rightarrow \textit{VBD(dumped, VBD) NP(sacks,NNS) PP(into,P)}) \tag{14.25}$$

is estimated as

$$\begin{aligned} P_H(\textit{VBD}|\textit{VP, dumped}) \; &\times \; P_L(\textit{STOP}|\textit{VP},\textit{VBD},\textit{dumped}) \\ &\times \; P_R(\textit{NP(sacks,NNS)}|\textit{VP},\textit{VBD},\textit{dumped}) \\ &\times \; P_R(\textit{PP(into,P)}|\textit{VP},\textit{VBD},\textit{dumped}) \\ &\times \; P_R(\textit{STOP}|\textit{VP},\textit{VBD},\textit{dumped}) \end{aligned} \tag{14.26}$$

Each of these probabilities can be estimated from much smaller amounts of data than the full probability in (14.25). For example, the maximum likelihood estimate for the component probability $P_R(\textit{NP(sacks,NNS)}|\textit{VP},\textit{VBD},\textit{dumped})$ is

$$\frac{\text{Count}(\textit{VP(dumped,VBD)} \text{ with } \textit{NNS(sacks)} \text{ as a daughter somewhere on the right})}{\text{Count}(\textit{ VP(dumped,VBD) })} \tag{14.27}$$

These counts are much less subject to sparsity problems than are complex counts like those in (14.25).

More generally, if H is a head with head word hw and head tag ht, lw/lt and rw/rt are the word/tag on the left and right respectively, and P is the parent, then the probability of an entire rule can be expressed as follows:

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 $L_{n+1}(lw_{n+1},lt_{n+1}) = \text{STOP}$, and we stop generating once we've generated a STOP token.
3. Generate modifiers to the right of the head with total probability:
$$\prod_{i=1}^{n+1} P_R(R_i(rw_i,rt_i)|P,H,hw,ht)$$
such that $R_{n+1}(rw_{n+1},rt_{n+1}) = \text{STOP}$, and we stop generating once we've generated a STOP token.

The parsing algorithm for the Collins model is an extension of probabilistic CKY. Extending the CKY algorithm to handle basic lexicalized probabilities is left as Exercises 14.5 and 14.6 for the reader.

14.7 Probabilistic CCG Parsing

Lexicalized grammar frameworks such as CCG pose problems for which the phrase-based methods we've been discussing are not particularly well-suited. To quickly review, CCG consists of three major parts: a set of categories, a lexicon that associates words with categories, and a set of rules that govern how categories combine in context. Categories can be either atomic elements, such as S and NP, or functions such as $(S\backslash NP)/NP$ which specifies the transitive verb category. Rules specify how functions, their arguments, and other functions combine. For example, the following rule templates, **forward** and **backward function application**, specify the way that functions apply to their arguments.

$$\begin{aligned} X/Y\ \ Y &\Rightarrow X \\ Y\ \ X\backslash Y &\Rightarrow X \end{aligned}$$

The first rule applies a function to its argument on the right, while the second looks to the left for its argument. The result of applying either of these rules is the category specified as the value of the function being applied. For the purposes of this discussion, we'll rely on these two rules along with the **forward** and **backward composition** rules and **type-raising**, as described in Chapter 12.

14.7.1 Ambiguity in CCG

As is always the case in parsing, managing ambiguity is the key to successful CCG parsing. The difficulties with CCG parsing arise from the ambiguity caused by the large number of complex lexical categories combined with the very general nature of the grammatical rules. To see some of the ways that ambiguity arises in a categorial framework, consider the following example.

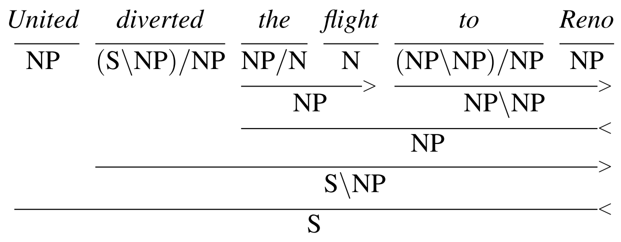
(14.28) United diverted the flight to Reno.

Our grasp of the role of *the flight* in this example depends on whether the prepositional phrase *to Reno* is taken as a modifier of *the flight*, as a modifier of the entire verb phrase, or as a potential second argument to the verb *divert*. In a context-free grammar approach, this ambiguity would manifest itself as a choice among the following rules in the grammar.

$$\begin{aligned} \textit{Nominal} &\rightarrow \textit{Nominal PP} \\ \textit{VP} &\rightarrow \textit{VP PP} \\ \textit{VP} &\rightarrow \textit{Verb NP PP} \end{aligned}$$

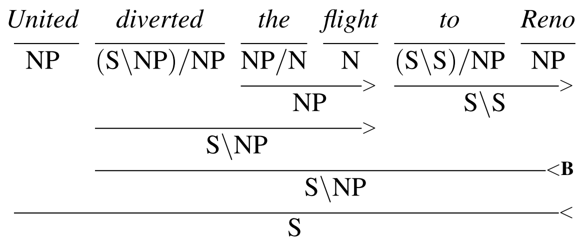
In a phrase-structure approach we would simply assign the word *to* to the category *P* allowing it to combine with *Reno* to form a prepositional phrase. The subsequent choice of grammar rules would then dictate the ultimate derivation. In the categorial approach, we can associate *to* with distinct categories to reflect the ways in which it might interact with other elements in a sentence. The fairly abstract combinatoric rules would then sort out which derivations are possible. Therefore, the source of ambiguity arises not from the grammar but rather from the lexicon.

Let's see how this works by considering several possible derivations for this example. To capture the case where the prepositional phrase *to Reno* modifies *the flight*, we assign the preposition *to* the category $(NP\backslash NP)/NP$, which gives rise to the following derivation.

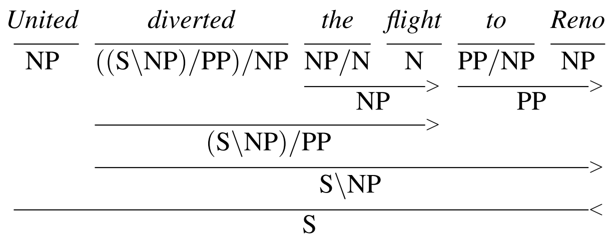

United diverted the flight to Reno
NP (S\NP)/NP NP/N N (NP\NP)/NP NP
NP >
NP\NP >
NP <
S\NP >
S <

Here, the category assigned to *to* expects to find two arguments: one to the right as with a traditional preposition, and one to the left that corresponds to the *NP* to be modified.

Alternatively, we could assign *to* to the category $(S\backslash S)/NP$, which permits the following derivation where to Reno modifies the preceding verb phrase.


United diverted the flight to Reno
NP (S\NP)/NP NP/N N (S\S)/NP NP
NP >
S\S >
S\NP >
S\NP <B
S <

A third possibility is to view *divert* as a ditransitive verb by assigning it to the category $((S\backslash NP)/PP)/NP$, while treating *to Reno* as a simple prepositional phrase.



While CCG parsers are still subject to ambiguity arising from the choice of grammar rules, including the kind of spurious ambiguity discussed in Chapter 12, it should be clear that the choice of lexical categories is the primary problem to be addressed in CCG parsing.

14.7.2 CCG Parsing Frameworks

Since the rules in combinatory grammars are either binary or unary, a bottom-up, tabular approach based on the CKY algorithm should be directly applicable to CCG parsing. Recall from Fig. 14.3 that PCKY employs a table that records the location, category and probability of all valid constituents discovered in the input. Given an appropriate probability model for CCG derivations, the same kind of approach can work for CCG parsing.

Unfortunately, the large number of lexical categories available for each word, combined with the promiscuity of CCG's combinatoric rules, leads to an explosion in the number of (mostly useless) constituents added to the parsing table. The key to managing this explosion of zombie constituents is to accurately assess and exploit the most likely lexical categories possible for each word — a process called supertagging.

The following sections describe two approaches to CCG parsing that make use of supertags. Section 14.7.4, presents an approach that structures the parsing process as a heuristic search through the use of the A* algorithm. The following section then briefly describes a more traditional maximum entropy approach that manages the search space complexity through the use of **adaptive supertagging** — a process that iteratively considers more and more tags until a parse is found.

14.7.3 Supertagging

supertagging

Chapter 8 introduced the task of part-of-speech tagging, the process of assigning the correct lexical category to each word in a sentence. **Supertagging** is the corresponding task for highly lexicalized grammar frameworks, where the assigned tags often dictate much of the derivation for a sentence.

CCG supertaggers rely on treebanks such as CCGbank to provide both the overall set of lexical categories as well as the allowable category assignments for each word in the lexicon. CCGbank includes over 1000 lexical categories, however, in practice, most supertaggers limit their tagsets to those tags that occur at least 10 times in the training corpus. This results in an overall total of around 425 lexical categories available for use in the lexicon. Note that even this smaller number is large in contrast to the 45 POS types used by the Penn Treebank tagset.

As with traditional part-of-speech tagging, the standard approach to building a CCG supertagger is to use supervised machine learning to build a sequence classifier using labeled training data. A common approach is to use the maximum entropy Markov model (MEMM), as described in Chapter 8, to find the most likely sequence of tags given a sentence. The features in such a model consist of the current word w_i, its surrounding words within l words w_{i-l}^{i+l}, as well as the k previously assigned supertags t_{i-k}^{i-1}. This type of model is summarized in the following equation from Chapter 8. Training by maximizing log-likelihood of the training corpus and decoding via the Viterbi algorithm are the same as described in Chapter 8.

$$\begin{aligned}\hat{T} &= \underset{T}{\operatorname{argmax}}\, P(T|W)\\ &= \underset{T}{\operatorname{argmax}} \prod_i P(t_i|w_{i-l}^{i+l}, t_{i-k}^{i-1})\\ &= \underset{T}{\operatorname{argmax}} \prod_i \frac{\exp\left(\sum_i w_i f_i(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}{\sum_{t' \in \text{tagset}} \exp\left(\sum_i w_i f_i(t', w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}\end{aligned} \tag{14.29}$$

Word and tag-based features with k and l both set to 2 provides reasonable results given sufficient training data. Additional features such as POS tags and short character suffixes are also commonly used to improve performance.

Unfortunately, even with additional features the large number of possible supertags combined with high per-word ambiguity leads to error rates that are too high for practical use in a parser. More specifically, the single best tag sequence $\hat{T}$ will typically contain too many incorrect tags for effective parsing to take place. To overcome this, we can instead return a probability distribution over the possible supertags for each word in the input. The following table illustrates an example distribution for a simple example sentence. In this table, each column represents the probability of each supertag for a given word *in the context of the input sentence*. The "..." represent all the remaining supertags possible for each word.

United	serves	Denver
N/N: 0.4	$(S\backslash NP)/NP$: 0.8	NP: 0.9
NP: 0.3	N: 0.1	N/N: 0.05
S/S: 0.1	...	...
$S\backslash S$: .05		
...		

In a MEMM framework, the probability of the optimal tag sequence defined in Eq. 14.29 is efficiently computed with a suitably modified version of the Viterbi algorithm. However, since Viterbi only finds the single best tag sequence it doesn't provide exactly what we need here; we need to know the probability of each possible word/tag pair. The probability of any given tag for a word is the sum of the probabilities of all the supertag sequences that contain that tag at that location. A table representing these values can be computed efficiently by using a version of the forward-backward algorithm used for HMMs.

The same result can also be achieved through recurrent neural network (RNN) sequence models, which have the advantage of embeddings to represent inputs and allow representations that span the entire sentence, as opposed to size-limited sliding windows. RNN approaches also avoid the use of high-level features, such as part of speech tags, which is helpful since errors in tag assignment can propagate to errors in supertags. As with the forward-backward algorithm, RNN-based methods can provide a probability distribution over the lexical categories for each word in the input.

14.7.4 CCG Parsing using the A* Algorithm

The A* algorithm is a heuristic search method that employs an agenda to find an optimal solution. Search states representing partial solutions are added to an agenda based on a cost function, with the least-cost option being selected for further exploration at each iteration. When a state representing a complete solution is first selected from the agenda, it is guaranteed to be optimal and the search terminates.

The A* cost function, $f(n)$, is used to efficiently guide the search to a solution. The f-cost has two components: $g(n)$, the exact cost of the partial solution represented by the state n, and $h(n)$ a heuristic approximation of the cost of a solution that makes use of n. When $h(n)$ satisfies the criteria of not overestimating the actual cost, A* will find an optimal solution. Not surprisingly, the closer the heuristic can get to the actual cost, the more effective A* is at finding a solution without having to explore a significant portion of the solution space.

When applied to parsing, search states correspond to edges representing completed constituents. As with the PCKY algorithm, edges specify a constituent's start and end positions, its grammatical category, and its f-cost. Here, the g component represents the current cost of an edge and the h component represents an estimate of the cost to complete a derivation that makes use of that edge. The use of A* for phrase structure parsing originated with (Klein and Manning, 2003a), while the CCG approach presented here is based on (Lewis and Steedman, 2014).

Using information from a supertagger, an agenda and a parse table are initialized with states representing all the possible lexical categories for each word in the input, along with their f-costs. The main loop removes the lowest cost edge from the agenda and tests to see if it is a complete derivation. If it reflects a complete derivation it is selected as the best solution and the loop terminates. Otherwise, new states based on the applicable CCG rules are generated, assigned costs, and entered into the agenda to await further processing. The loop continues until a complete derivation is discovered, or the agenda is exhausted, indicating a failed parse. The algorithm is given in Fig. 14.11.

Heuristic Functions

Before we can define a heuristic function for our A* search, we need to decide how to assess the quality of CCG derivations. For the generic PCFG model, we defined the probability of a tree as the product of the probability of the rules that made up the tree. Given CCG's lexical nature, we'll make the simplifying assumption that the probability of a CCG derivation is just the product of the probability of the supertags assigned to the words in the derivation, ignoring the rules used in the derivation. More formally, given a sentence S and derivation D that contains supertag sequence T, we have:

$$\begin{aligned} P(D,S) &= P(T,S) && (14.30) \\ &= \prod_{i=1}^{n} P(t_i|s_i) && (14.31) \end{aligned}$$

```
function CCG-ASTAR-PARSE(words) returns table or failure

  supertags ← SUPERTAGGER(words)
  for i ← from 1 to LENGTH(words) do
      for all {A | (words[i], A, score) ∈ supertags}
          edge ← MAKEEDGE(i − 1, i, A, score)
          table ← INSERTEDGE(table, edge)
          agenda ← INSERTEDGE(agenda, edge)
  loop do
     if EMPTY?(agenda) return failure
     current ← POP(agenda)
     if COMPLETEDPARSE?(current) return table
     table ← INSERTEDGE(chart, edge)
     for each rule in APPLICABLERULES(edge) do
        successor ← APPLY(rule, edge)
        if successor not ∈ in agenda or chart
            agenda ← INSERTEDGE(agenda, successor)
        else if successor ∈ agenda with higher cost
            agenda ← REPLACEEDGE(agenda, successor)
```

Figure 14.11 A*-based CCG parsing.

To better fit with the traditional A* approach, we'd prefer to have states scored by a cost function where lower is better (i.e., we're trying to minimize the cost of a derivation). To achieve this, we'll use negative log probabilities to score derivations; this results in the following equation, which we'll use to score completed CCG derivations.

$$P(D,S) = P(T,S) \tag{14.32}$$

$$= \sum_{i=1}^{n} -\log P(t_i|s_i) \tag{14.33}$$

Given this model, we can define our f-cost as follows. The f-cost of an edge is the sum of two components: $g(n)$, the cost of the span represented by the edge, and $h(n)$, the estimate of the cost to complete a derivation containing that edge (these are often referred to as the **inside** and **outside costs**). We'll define $g(n)$ for an edge using Equation 14.33. That is, it is just the sum of the costs of the supertags that comprise the span.

For $h(n)$, we need a score that approximates but *never overestimates* the actual cost of the final derivation. A simple heuristic that meets this requirement assumes that each of the words in the outside span will be assigned its *most probable supertag*. If these are the tags used in the final derivation, then its score will equal the heuristic. If any other tags are used in the final derivation the f-cost will be higher since the new tags must have higher costs, thus guaranteeing that we will not overestimate.

Putting this all together, we arrive at the following definition of a suitable f-cost for an edge.

$$\begin{aligned} f(w_{i,j},t_{i,j}) &= g(w_{i,j})+h(w_{i,j}) \\ &= \sum_{k=i}^{j} -\log P(t_k|w_k) + \\ &\quad \sum_{k=1}^{i-1} \min_{t \in tags}(-\log P(t|w_k)) + \sum_{k=j+1}^{N} \min_{t \in tags}(-\log P(t|w_k)) \end{aligned} \tag{14.34}$$

As an example, consider an edge representing the word *serves* with the supertag N in the following example.

(14.35) United serves Denver.

The g-cost for this edge is just the negative log probability of this tag, $-log_{10}(0.1)$, or 1. The outside h-cost consists of the most optimistic supertag assignments for *United* and *Denver*, which are N/N and NP respectively. The resulting f-cost for this edge is therefore 1.443.

An Example

Fig. 14.12 shows the initial agenda and the progress of a complete parse for this example. After initializing the agenda and the parse table with information from the supertagger, it selects the best edge from the agenda — the entry for *United* with the tag N/N and f-cost 0.591. This edge does not constitute a complete parse and is therefore used to generate new states by applying all the relevant grammar rules. In this case, applying forward application to *United: N/N* and *serves: N* results in the creation of the edge *United serves: N[0,2], 1.795* to the agenda.

Skipping ahead, at the third iteration an edge representing the complete derivation *United serves Denver, S[0,3], .716* is added to the agenda. However, the algorithm does not terminate at this point since the cost of this edge (.716) does not place it at the top of the agenda. Instead, the edge representing *Denver* with the category *NP* is popped. This leads to the addition of another edge to the agenda (type-raising *Denver*). Only after this edge is popped and dealt with does the earlier state representing a complete derivation rise to the top of the agenda where it is popped, goal tested, and returned as a solution.

The effectiveness of the A* approach is reflected in the coloring of the states in Fig. 14.12 as well as the final parsing table. The edges shown in blue (including all the initial lexical category assignments not explicitly shown) reflect states in the search space that never made it to the top of the agenda and, therefore, never contributed any edges to the final table. This is in contrast to the PCKY approach where the parser systematically fills the parse table with all possible constituents for all possible spans in the input, filling the table with myriad constituents that do not contribute to the final analysis.

14.8 Evaluating Parsers

The standard techniques for evaluating parsers and grammars are called the PARSEVAL measures; they were proposed by Black et al. (1991) and were based on the same ideas from signal-detection theory that we saw in earlier chapters. The

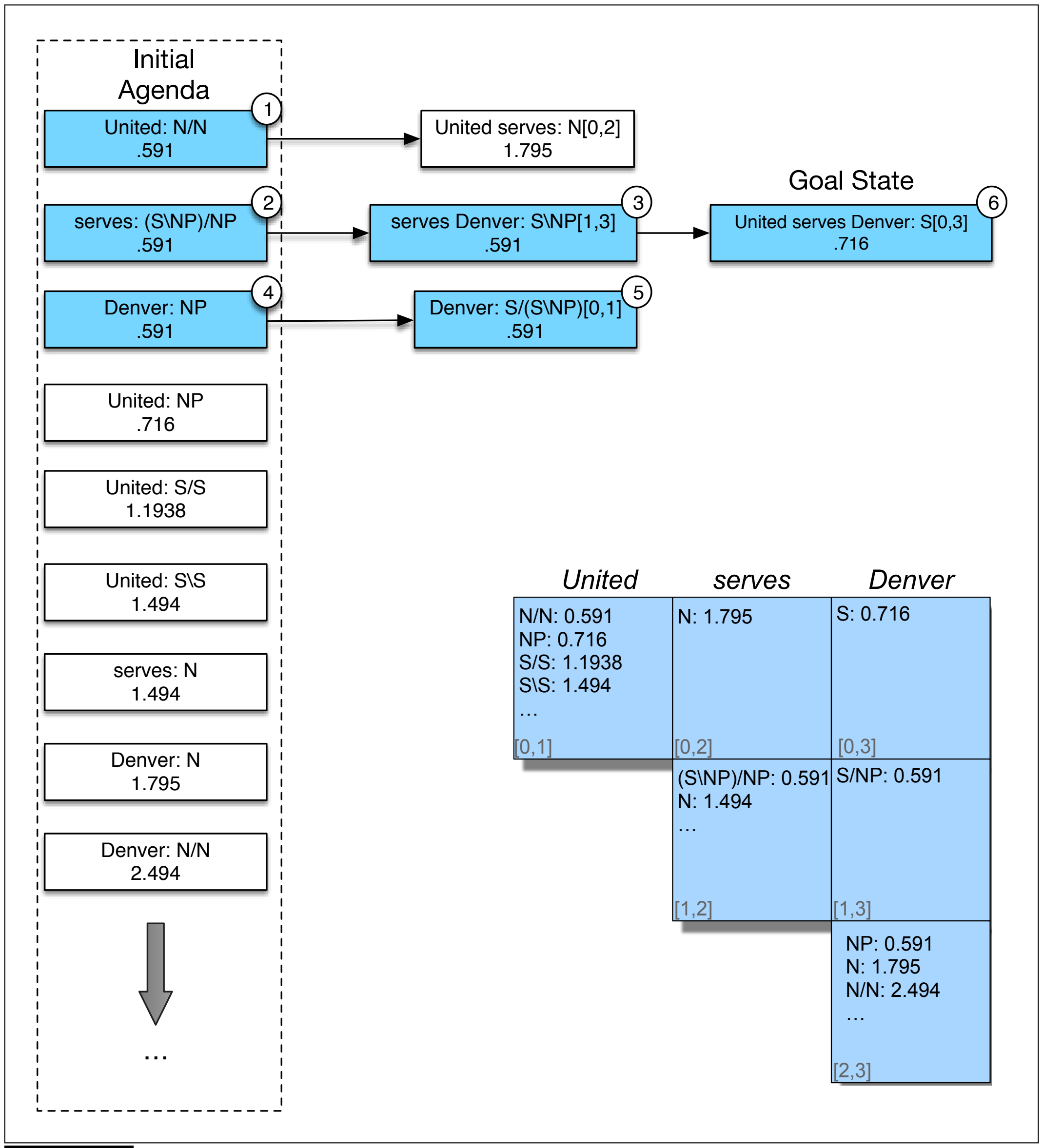


Figure 14.12 Example of an A* search for the example "United serves Denver". The circled numbers on the blue boxes indicate the order in which the states are popped from the agenda. The costs in each state are based on f-costs using negative log_{10} probabilities.

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.

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

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

F-measure

As with other uses of precision and recall, we often report a combination of the two, the **F-measure** (van Rijsbergen, 1975), which, as we saw in Chapter 4, is defined as:

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

Values of $\beta > 1$ favor recall and values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is called $\mathrm{F}_{\beta=1}$ or just F_1:

$$F_1 = \frac{2PR}{P+R} \tag{14.36}$$

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)).

For comparing parsers that 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 (Black et al., 1991). The canonical implementation of the PARSEVAL metrics is called **evalb** (Sekine and Collins, 1997).

evalb

Nonetheless, phrasal constituents are not always an appropriate unit for parser evaluation. In lexically-oriented grammars, such as CCG and LFG, the ultimate goal is to extract the appropriate predicate-argument relations or grammatical dependencies, rather than a specific derivation. Such relations are also more directly relevant to further semantic processing. For these purposes, we can use alternative evaluation metrics based on the precision and recall of labeled dependencies whose labels indicate the grammatical relations (Lin 1995, Carroll et al. 1998, Collins et al. 1999).

Finally, you might wonder why we don't evaluate parsers by measuring how many *sentences* are parsed correctly instead of measuring *component* accuracy in the form of constituents or dependencies. The reason we use components is that it 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 parts wrong and one that just got one part wrong.

14.9 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 Chapter 3.

- A **probabilistic context-free grammar** (**PCFG**) is a context-free grammar in which every rule is annotated with the probability of that rule being chosen. 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 learned 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 among 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 with three metrics: **labeled recall**, **labeled precision**, and **cross-brackets**.

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) adapted 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 by 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 that used semantic distance between heads (computed from WordNet) (Stetina and Nagao, 1997), and boosting (Abney et al., 1999). Another direction extended the lexical probabilistic parsing work to build probabilistic formulations of grammars other than PCFGs, such as probabilistic TAG grammar (Resnik 1992, Schabes 1992), based on the TAG grammars discussed in Chapter 12, probabilistic LR parsing (Briscoe and Carroll, 1993), and probabilistic link grammar (Lafferty et al., 1992). The supertagging approach we saw for CCG was developed for TAG grammars (Bangalore and Joshi 1999, Joshi and Srinivas 1994), based on the lexicalized TAG grammars of Schabes et al. (1988).

Exercises

14.1 Implement the CKY algorithm.

14.2 Modify the algorithm for conversion to CNF from Chapter 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 13.3 asked you to update the CKY algorithm to handle 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 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 Section 14.8. Next, either use a treebank or create your own hand-checked parsed test set. Now use your CFG (or other) parser and grammar, parse the test set and compute labeled recall, labeled precision, and cross-brackets.

Abney, S. P., Schapire, R. E., and Singer, Y. (1999). Boosting applied to tagging and PP attachment. In *EMNLP/VLC-99*, 38–45.

Baker, J. K. (1979). Trainable grammars for speech recognition. In Klatt, D. H. and Wolf, J. J. (Eds.), *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, 547–550.

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

Black, E., Abney, S. P., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J. L., Liberman, M. Y., Marcus, M. P., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings DARPA Speech and Natural Language Workshop*, 306–311.

Black, E., Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., and Roukos, S. (1992). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings DARPA Speech and Natural Language Workshop*, 134–139.

Bod, R. (1993). Using an annotated corpus as a stochastic grammar. In *EACL-93*, 37–44.

Booth, T. L. (1969). Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, 74–81.

Booth, T. L. and Thompson, R. A. (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers*, *C-22*(5), 442–450.

Bresnan, J. (Ed.). (1982). *The Mental Representation of Grammatical Relations*. MIT Press.

Brill, E. and Resnik, P. (1994). A rule-based approach to prepositional phrase attachment disambiguation. In *COLING-94*, 1198–1204.

Briscoe, T. and Carroll, J. (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, *19*(1), 25–59.

Carroll, J., Briscoe, T., and Sanfilippo, A. (1998). Parser evaluation: A survey and a new proposal. In *LREC-98*, 447–454.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *AAAI-97*, 598–603.

Chelba, C. and Jelinek, F. (2000). Structured language modeling. *Computer Speech and Language*, *14*, 283–332.

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *ACL-96*, 184–191.

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

Collins, M., Hajič, J., Ramshaw, L. A., and Tillmann, C. (1999). A statistical parser for Czech. In *ACL-99*, 505–512.

Francis, H. S., Gregory, M. L., and Michaelis, L. A. (1999). Are lexical subjects deviant?. In *CLS-99*. University of Chicago.

Franz, A. (1997). Independence assumptions considered harmful. In *ACL/EACL-97*, 182–189.

Givón, T. (1990). *Syntax: A Functional Typological Introduction*. John Benjamins.

Hindle, D. and Rooth, M. (1990). Structural ambiguity and lexical relations. In *Proceedings DARPA Speech and Natural Language Workshop*, 257–262.

Hindle, D. and Rooth, M. (1991). Structural ambiguity and lexical relations. In *ACL-91*, 229–236.

Jelinek, F. and Lafferty, J. D. (1991). Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, *17*(3), 315–323.

Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., Ratnaparkhi, A., and Roukos, S. (1994). Decision tree parsing using a hidden derivation model. In *ARPA Human Language Technologies Workshop*, 272–277.

Johnson, M. (1998). PCFG models of linguistic tree representations. *Computational Linguistics*, *24*(4), 613–632.

Joshi, A. K. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?. In Dowty, D. R., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing*, 206–250. Cambridge University Press.

Joshi, A. K. and Srinivas, B. (1994). Disambiguation of super parts of speech (or supertags): Almost parsing. In *COLING-94*, 154–160.

Klein, D. and Manning, C. D. (2001). Parsing and hypergraphs. In *IWPT-01*, 123–134.

Klein, D. and Manning, C. D. (2003a). A* parsing: Fast exact Viterbi parse selection. In *HLT-NAACL-03*.

Klein, D. and Manning, C. D. (2003b). Accurate unlexicalized parsing. In *HLT-NAACL-03*.

Lafferty, J. D., Sleator, D., and Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, *4*, 35–56.

Lewis, M. and Steedman, M. (2014). A* ccg parsing with a supertag-factored model.. In *EMNLP*, 990–1000.

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

Magerman, D. M. and Marcus, M. P. (1991). Pearl: A probabilistic chart parser. In *EACL-91*.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Marcus, M. P. (1990). Summary of session 9: Automatic acquisition of linguistic structure. In *Proceedings DARPA Speech and Natural Language Workshop*, 249–250.

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

Ney, H. (1991). Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, *39*(2), 336–340.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *COLING/ACL 2006*, 433–440.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Ratnaparkhi, A., Reynar, J. C., and Roukos, S. (1994). A maximum entropy model for prepositional phrase attachment. In *ARPA Human Language Technologies Workshop*, 250–255.

Resnik, P. (1992). Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *COLING-92*, 418–424.

Salomaa, A. (1969). Probabilistic and weighted grammars. *Information and Control*, *15*, 529–544.

Schabes, Y. (1990). *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Schabes, Y. (1992). Stochastic lexicalized tree-adjoining grammars. In *COLING-92*, 426–433.

Schabes, Y., Abeillé, A., and Joshi, A. K. (1988). Parsing strategies with 'lexicalized' grammars: Applications to Tree Adjoining Grammars. In *COLING-88*, 578–583.

Sekine, S. and Collins, M. (1997). The evalb software. `http://cs.nyu.edu/cs/projects/proteus/evalb`.

Stetina, J. and Nagao, M. (1997). Corpus based PP attachment ambiguity resolution with a semantic dictionary. In Zhou, J. and Church, K. W. (Eds.), *Proceedings of the Fifth Workshop on Very Large Corpora*, 66–80.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, *21*(2), 165–202.

van Rijsbergen, C. J. (1975). *Information Retrieval*. Butterworths.

Zavrel, J. and Daelemans, W. (1997). Memory-based learning: Using similarity for smoothing. In *ACL/EACL-97*, 436–443.