Lexicalized Probabilistic Context-Free Grammars

Michael Collins

1 Introduction

In the previous lecture notes we introduced probabilistic context-free grammars (PCFGs) as a model for statistical parsing. We introduced the basic PCFG formalism; described how the parameters of a PCFG can be estimated from a set of training examples (a "treebank"); and derived a dynamic programming algorithm for parsing with a PCFG.

Unfortunately, the basic PCFGs we have described turn out to be a rather poor model for statistical parsing. This note introduces *lexicalized PCFGs*, which build directly on ideas from regular PCFGs, but give much higher parsing accuracy. The remainder of this note is structured as follows:

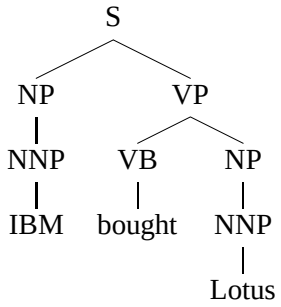
- In section 2 we describe some weaknesses of basic PCFGs, in particular focusing on their lack of sensitity to lexical information.
- In section 3 we describe the first step in deriving lexicalized PCFGs: the process of adding lexical items to non-terminals in treebank parses.
- In section 4 we give a formal definition of lexicalized PCFGs.
- In section 5 we describe how the parameters of lexicalized PCFGs can be estimated from a treebank.
- In section 6 we describe a dynamic-programming algorithm for parsing with lexicalized PCFGs.

2 Weaknesses of PCFGs as Parsing Models

We focus on two crucial weaknesses of PCFGs: 1) lack of sensitivity to lexical information; and 2), lack of sensitivity to structural preferences. Problem (1) is the underlying motivation for a move to lexicalized PCFGs. In a later lecture we will describe extensions to lexical PCFGs that address problem (2).

2.1 Lack of Sensitivity to Lexical Information

First, consider the following parse tree:



Under the PCFG model, this tree will have probability

$$q(\text{S} \rightarrow \text{NP VP}) \times q(\text{VP} \rightarrow \text{V NP}) \times q(\text{NP} \rightarrow \text{NNP}) \times q(\text{NP} \rightarrow \text{NNP})$$
$$\times q(\text{NNP} \rightarrow IBM) \times q(\text{Vt} \rightarrow bought) \times q(\text{NNP} \rightarrow Lotus)$$

Recall that for any rule $\alpha \rightarrow \beta$, $q(\alpha \rightarrow \beta)$ is an associated parameter, which can be interpreted as the conditional probability of seeing β on the right-hand-side of the rule, given that α is on the left-hand-side of the rule.

If we consider the lexical items in this parse tree (i.e., *IBM*, *bought*, and *Lotus*), we can see that the PCFG makes a very strong independence assumption. Intuitively the identity of each lexical item depends only on the part-of-speech (POS) above that lexical item: for example, the choice of the word *IBM* depends on its tag *NNP*, but does not depend directly on other information in the tree. More formally, the choice of each word in the string is conditionally independent of the entire tree, once we have conditioned on the POS directly above the word. This is clearly a very strong assumption, and it leads to many problems in parsing. We will see that lexicalized PCFGs address this weakness of PCFGs in a very direct way.

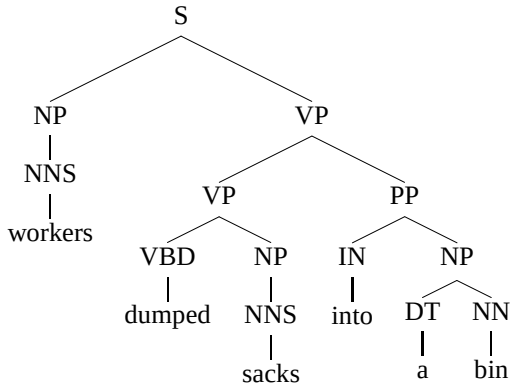
Let's now look at how PCFGs behave under a particular type of ambiguity, prepositional-phrase (PP) attachment ambiguity. Figure 1 shows two parse trees for the same sentence that includes a PP attachment ambiguity. Figure 2 lists the set of context-free rules for the two parse trees. A critical observation is the following: **the two parse trees have identical rules, with the exception of** `VP -> VP PP` **in tree (a), and** `NP -> NP PP` **in tree (b).** It follows that the probabilistic parser, when choosing between the two parse trees, will pick tree (a) if

$$q(\text{VP} \rightarrow \text{VP PP}) > q(\text{NP} \rightarrow \text{NP PP})$$

and will pick tree (b) if

$$q(\text{NP} \rightarrow \text{NP PP}) > q(\text{VP} \rightarrow \text{VP PP})$$

(a)



(b)

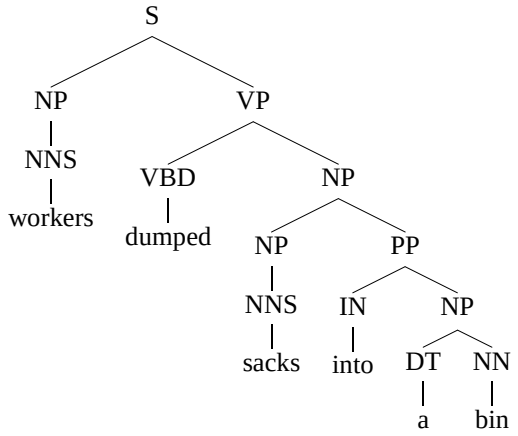


Figure 1: Two valid parses for a sentence that includes a prepositional-phrase attachment ambiguity.

(a)

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

(b)

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

Figure 2: The set of rules for parse trees (a) and (b) in figure 1.

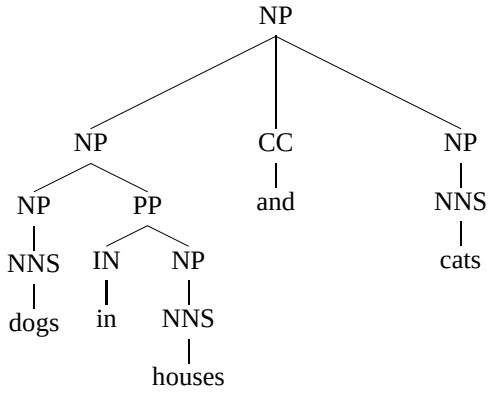
Notice that this decision is *entirely independent of any lexical information (the words) in the two input sentences*. For this particular case of ambiguity (NP vs VP attachment of a PP, with just one possible NP and one possible VP attachment) the parser will always attach PPs to VP if $q(\text{VP} \rightarrow \text{VP PP}) > q(\text{NP} \rightarrow \text{NP PP})$, and conversely will always attach PPs to NP if $q(\text{NP} \rightarrow \text{NP PP}) > q(\text{VP} \rightarrow \text{VP PP})$.

The lack of sensitivity to lexical information in this particular situation, prepositional-phrase attachment ambiguity, is known to be highly non-optimal. The lexical items involved can give very strong evidence about whether to attach to the noun or the verb. If we look at the preposition, *into*, alone, we find that PPs with *into* as the preposition are almost nine times more likely to attach to a VP rather than an NP (this statistic is taken from the Penn treebank data). As another example, PPs with the preposition *of* are about 100 times more likely to attach to an NP rather than a VP. But PCFGs ignore the preposition entirely in making the attachment decision.

As another example, consider the two parse trees shown in figure 3, which is an example of coordination ambiguity. In this case it can be verified that the two parse trees have identical sets of context-free rules (the only difference is in the order in which these rules are applied). Hence a PCFG will assign identical probabilities to these two parse trees, again completely ignoring lexical information.

In summary, the PCFGs we have described essentially generate lexical items as an afterthought, conditioned only on the POS directly above them in the tree. This is a very strong independence assumption, which leads to non-optimal decisions being made by the parser in many important cases of ambiguity.

(a)



(b)

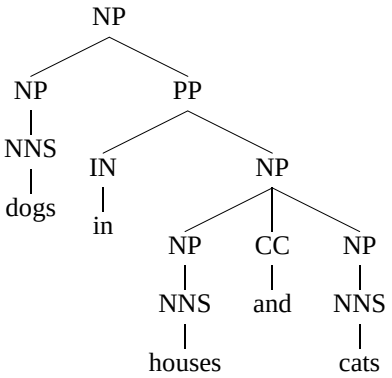


Figure 3: Two valid parses for a noun-phrase that includes an instance of coordination ambiguity.

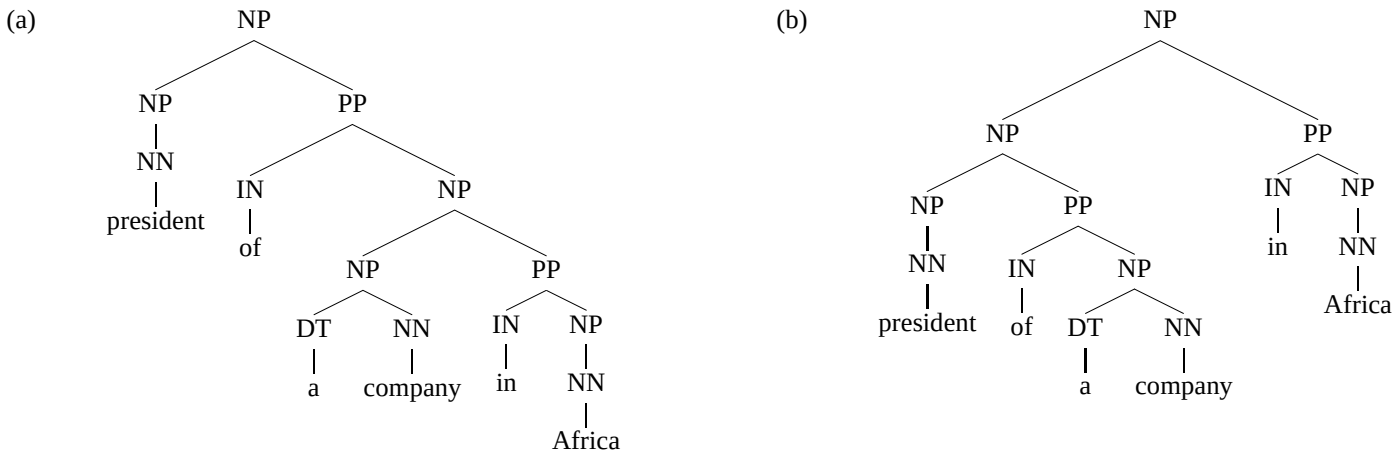


Figure 4: Two possible parses for *president of a company in Africa*.

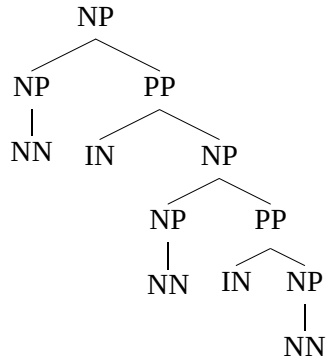
2.2 Lack of Sensitivity to Structural Preferences

A second weakness of PCFGs is their lack of sensitivity to structural preferences. We illustrate this weakness through a couple of examples.

First, consider the two potential parses for *president of a company in Africa*, shown in figure 4. This noun-phrase again involves a case of prepositional-phrase attachment ambiguity: the PP *in Africa* can either attach to *president* or *a company*. It can be verified once again that these two parse trees contain exactly the same set of context-free rules, and will therefore get identical probabilities under a PCFG.

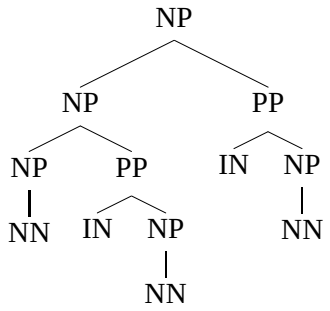
Lexical information may of course help again, in this case. However another useful source of information may be basic statistics about structural preferences (preferences that ignore lexical items). The first parse tree involves a structure of the following form, where the final PP (*in Africa* in the example) attaches to the most recent NP (*a company*):

(1)



This attachment for the final PP is often referred to as a *close attachment*, because the PP has attached to the closest possible preceding NP. The second parse structure has the form

(2)



where the final PP has attached to the further NP (*president* in the example).

We can again look at statistics from the treebank for the frequency of structure 1 versus 2: structure 1 is roughly twice as frequent as structure 2. So there is a fairly significant bias towards close attachment. Again, we stress that the PCFG assigns identical probabilities to these two trees, because they include the same set of rules: hence the PCFG fails to capture the bias towards close-attachment in this case.

There are many other examples where close attachment is a useful cue in disambiguating structures. The preferences can be even stronger when a choice is being made between attachment to two different verbs. For example, consider the sentence

John was believed to have been shot by Bill

Here the PP *by Bill* can modify either the verb *shot* (Bill was doing the shooting) or *believe* (Bill is doing the believing). However statistics from the treebank show that when a PP can attach to two potential verbs, it is about 20 times more likely

to attach to the most recent verb. Again, the basic PCFG will often give equal probability to the two structures in question, because they contain the same set of rules.

3 Lexicalization of a Treebank

We now describe how lexicalized PCFGs address the first fundamental weakness of PCFGs: their lack of sensitivity to lexical information. The first key step, described in this section, is to *lexicalize* the underlying treebank.

Figure 5 shows a parse tree before and after lexicalization. The lexicalization step has replaced non-terminals such as `S` or `NP` with new non-terminals that include lexical items, for example `S(questioned)` or `NP(lawyer)`.

The remainder of this section describes exactly how trees are lexicalized. First though, we give the underlying motivation for this step. The basic idea will be to replace rules such as

$$\text{S} \rightarrow \text{NP VP}$$

in the basic PCFG, with rules such as

$$\text{S(questioned)} \rightarrow \text{NP(lawyer) VP(questioned)}$$

in the lexicalized PCFG. The symbols S(questioned), NP(lawyer) and VP(questioned) are new non-terminals in the grammar. Each non-terminal now includes a lexical item; the resulting model has far more sensitivity to lexical information.

In one sense, nothing has changed from a formal standpoint: we will simply move from a PCFG with a relatively small number of non-terminals (`S`, `NP`, etc.) to a PCFG with a much larger set of non-terminals (`S(questioned)`, `NP(lawyer)` etc.) This will, however, lead to a radical increase in the number of rules and non-terminals in the grammar: for this reason we will have to take some care in estimating the parameters of the underlying PCFG. We describe how this is done in the next section.

First, however, we describe how the lexicalization process is carried out. The key idea will be to identify for each context-free rule of the form

$$X \rightarrow Y_1 \; Y_2 \; \ldots \; Y_n$$

an index $h \in \{1 \ldots n\}$ that specifies the *head* of the rule. The head of a context-free rule intuitively corresponds to the "center" or the most important child of the rule.[1] For example, for the rule

$$\text{S} \rightarrow \text{NP VP}$$

[1] The idea of heads has a long history in linguistics, which is beyond the scope of this note.

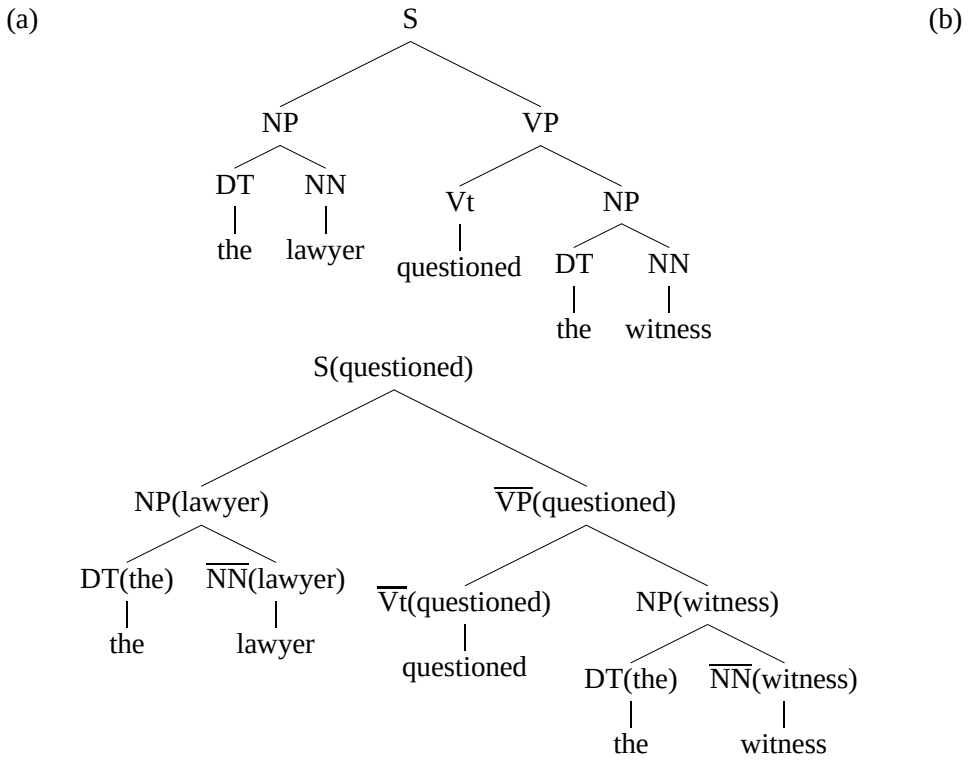


Figure 5: (a) A conventional parse tree as found for example in the Penn treebank. (b) A lexicalized parse tree for the same sentence. Note that each non-terminal in the tree now includes a single lexical item. For clarity we mark the head of each rule with an overline: for example for the rule `NP` $\rightarrow$ `DT NN` the child `NN` is the head, and hence the `NN` symbol is marked as $\overline{\texttt{NN}}$.

the head would be $h = 2$ (corresponding to the VP). For the rule

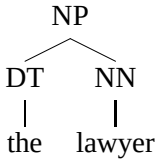
$$\text{NP} \rightarrow \text{NP PP PP PP}$$

the head would be $h = 1$ (corresponding to the NP). For the rule

$$\text{PP} \rightarrow \text{IN NP}$$

the head would be $h = 1$ (corresponding to the IN), and so on.

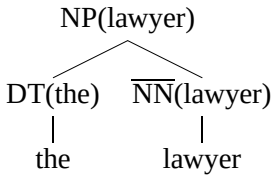
Once the head of each context-free rule has been identified, lexical information can be propagated bottom-up through parse trees in the treebank. For example, if we consider the sub-tree



and assuming that the head of the rule

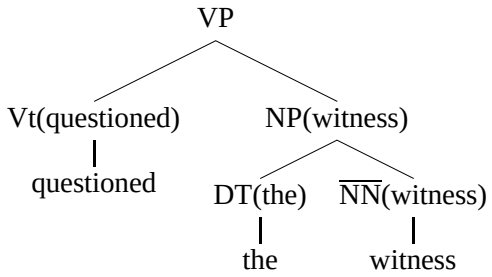
$$\text{NP} \rightarrow \text{DT NN}$$

is $h = 2$ (the NN), the lexicalized sub-tree is

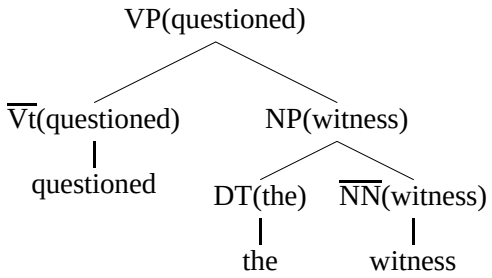


Parts of speech such as DT or NN receive the lexical item below them as their head word. Non-terminals higher in the tree receive the lexical item from their head child: for example, the NP in this example receives the lexical item lawyer, because this is the lexical item associated with the NN which is the head child of the NP. For clarity, we mark the head of each rule in a lexicalized parse tree with an overline (in this case we have $\overline{\text{NN}}$). See figure 5 for an example of a complete lexicalized tree.

As another example consider the VP in the parse tree in figure 5. Before lexicalizing the VP, the parse structure is as follows (we have filled in lexical items lower in the tree, using the steps described before):



We then identify `Vt` as the head of the rule VP $\rightarrow$ Vt NP, and lexicalize the tree as follows:



In summary, once the head of each context-free rule has been identified, lexical items can be propagated bottom-up through parse trees, to give lexicalized trees such as the one shown in figure 5(b).

The remaining question is how to identify heads. Ideally, the head of each rule would be annotated in the treebank in question: in practice however, these annotations are often not present. Instead, researchers have generally used a simple set of rules to automatically identify the head of each context-free rule.

As one example, figure 6 gives an example set of rules that identifies the head of rules whose left-hand-side is `NP`. Figure 7 shows a set of rules used for `VPs`. In both cases we see that the rules look for particular children (e.g., `NN` for the `NP` case, `Vi` for the `VP` case). The rules are fairly heuristic, but rely on some linguistic guidance on what the head of a rule should be: in spite of their simplicity they work quite well in practice.

4 Lexicalized PCFGs

The basic idea in lexicalized PCFGs will be to replace rules such as

$$S \rightarrow NP\ VP$$

with lexicalized rules such as

$$S(examined) \rightarrow NP(lawyer)\ VP(examined)$$

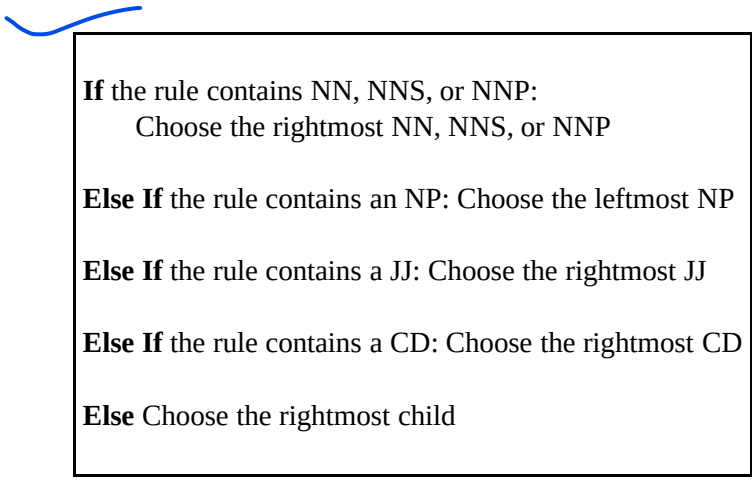


Figure 6: Example of a set of rules that identifies the head of any rule whose left-hand-side is an NP.

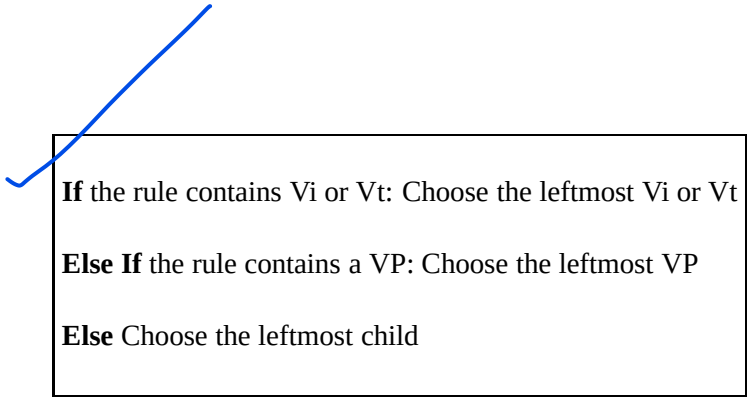


Figure 7: Example of a set of rules that identifies the head of any rule whose left-hand-side is a VP.

Thus we have replaced simple non-terminals such as `S` or `NP` with lexicalized non-terminals such as `S(examined)` or `NP(lawyer)`.

From a formal standpoint, nothing has changed: we can treat the new, lexicalized grammar exactly as we would a regular PCFG. We have just expanded the number of non-terminals in the grammar from a fairly small number (say 20, or 50) to a much larger number (because each non-terminal now has a lexical item, we could easily have thousands or tens of thousands of non-terminals).

Each rule in the lexicalized PCFG will have an associated parameter, for example the above rule would have the parameter

$$q(\text{S(examined)} \rightarrow \text{NP(lawyer) VP(examined)})$$

There are a very large number of parameters in the model, and we will have to take some care in estimating them: the next section describes parameter estimation methods.

We will next give a formal definition of lexicalized PCFGs, in Chomsky normal form. First, though, we need to take care of one detail. Each rule in the lexicalized PCFG has a non-terminal with a head word on the left hand side of the rule: for example the rule

$$\text{S(examined)} \rightarrow \text{NP(lawyer) VP(examined)}$$

has `S(examined)` on the left hand side. In addition, the rule has two children. One of the two children must have the same lexical item as the left hand side: in this example `VP(examined)` is the child with this property. To be explicit about which child shares the lexical item with the left hand side, we will add an annotation to the rule, using $\rightarrow_1$ to specify that the left child shares the lexical item with the parent, and $\rightarrow_2$ to specify that the right child shares the lexical item with the parent. So the above rule would now be written as

$$\text{S(examined)} \rightarrow_2 \text{NP(lawyer) VP(examined)}$$

The extra notation might seem unneccessary in this case, because it is clear that the second child is the head of the rule—it is the only child to have the same lexical item, *examined*, as the left hand side of the rule. However this information will be important for rules where both children have the same lexical item: take for example the rules

$$\text{PP(in)} \rightarrow_1 \text{PP(in) PP(in)}$$

and

$$\text{PP(in)} \rightarrow_2 \text{PP(in) PP(in)}$$

where we need to be careful about specifying which of the two children is the head of the rule.

We now give the following definition:

Definition 1 (Lexicalized PCFGs in Chomsky Normal Form) *A lexicalized PCFG in Chomsky normal form is a 6-tuple $G = (N, \Sigma, R, S, q, \gamma)$ where:*

- *N is a finite set of non-terminals in the grammar.*
- *Σ is a finite set of lexical items in the grammar.*
- *R is a set of rules. Each rule takes one of the following three forms:*
  1. *$X(h) \rightarrow_1 Y_1(h)\ Y_2(m)$ where $X, Y_1, Y_2 \in N$, $h, m \in \Sigma$.*
  2. *$X(h) \rightarrow_2 Y_1(m)\ Y_2(h)$ where $X, Y_1, Y_2 \in N$, $h, m \in \Sigma$.*
  3. *$X(h) \rightarrow h$ where $X \in N$, $h \in \Sigma$.*
- *For each rule $r \in R$ there is an associated parameter*

  $$q(r)$$

  The parameters satisfy $q(r) \geq 0$, and for any $X \in N, h \in \Sigma$,

  $$\sum_{r \in R : LHS(r) = X(h)} q(r) = 1$$

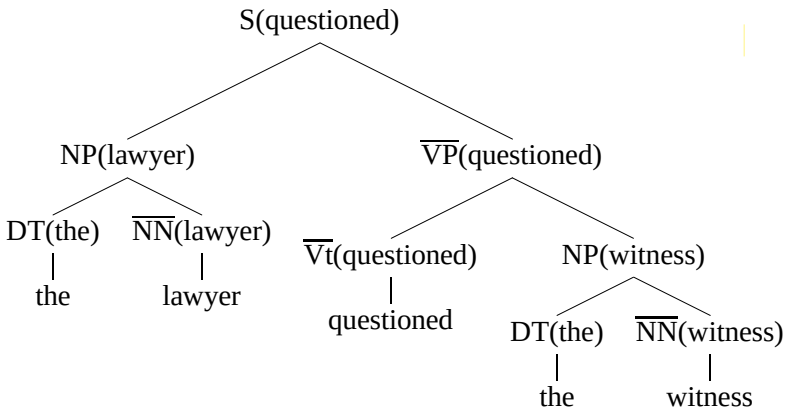
  where we use $LHS(r)$ to refer to the left hand side of any rule r.
- *For each $X \in N$, $h \in \Sigma$, there is a parameter $\gamma(X, h)$. We have $\gamma(X, h) \geq 0$, and $\sum_{X \in N, h \in \Sigma} \gamma(X, h) = 1$.*

Given a left-most derivation $r_1, r_2, \ldots r_N$ under the grammar, where each r_i is a member of R, the probability of the derivation is

$$\gamma(LHS(r_1)) \times \prod_{i=1}^{N} q(r_i)$$

□

As an example, consider the parse tree in figure 5, repeated here:



In this case the parse tree consists of the following sequence of rules:

S(questioned) $\rightarrow_2$ NP(lawyer) VP(questioned)
NP(lawyer) $\rightarrow_2$ DT(the) NN(lawyer)
DT(the) $\rightarrow$ the
NN(lawyer) $\rightarrow$ lawyer
VP(questioned) $\rightarrow_1$ Vt(questioned) NP(witness)
NP(witness) $\rightarrow_2$ DT(the) NN(witness)
DT(the) $\rightarrow$ the
NN(witness) $\rightarrow$ witness

The probability of the tree is calculated as

$\gamma(\text{S, questioned})$
$\times q(\text{S(questioned)} \rightarrow_2 \text{NP(lawyer) VP(questioned)})$
$\times q(\text{NP(lawyer)} \rightarrow_2 \text{DT(the) NN(lawyer)})$
$\times q(\text{DT(the)} \rightarrow \text{the})$
$\times q(\text{NN(lawyer)} \rightarrow \text{lawyer})$
$\times q(\text{VP(questioned)} \rightarrow_1 \text{Vt(questioned) NP(witness)})$
$\times q(\text{NP(witness)} \rightarrow_2 \text{DT(the) NN(witness)})$
$\times q(\text{DT(the)} \rightarrow \text{the})$
$\times q(\text{NN(witness)} \rightarrow \text{witness})$

Thus the model looks very similar to regular PCFGs, where the probability of a tree is calculated as a product of terms, one for each rule in the tree. One difference is that we have the $\gamma(\text{S, questioned})$ term for the root of the tree: this term can be interpreted as the probability of choosing the nonterminal S(questioned) at the

root of the tree. (Recall that in regular PCFGs we specified that a particular non-terminal, for example S, always appeared at the root of the tree.)

5 Parameter Estimation in Lexicalized PCFGs

We now describe a method for parameter estimation within lexicalized PCFGs. The number of rules (and therefore parameters) in the model is very large. However with appropriate smoothing—using techniques described earlier in the class, for language modeling—we can derive estimates that are robust and effective in practice.

First, for a given rule of the form

$$X(h) \rightarrow_1 Y_1(h)\ Y_2(m)$$

or

$$X(h) \rightarrow_2 Y_1(m)\ Y_2(h)$$

define the following variables: X is the non-terminal on the left-hand side of the rule; H is the head-word of that non-terminal; R is the rule used, either of the form $X \rightarrow_1 Y_1\ Y_2$ or $X \rightarrow_2 Y_1\ Y_2$; M is the modifier word.

For example, for the rule

$$\text{S(examined)} \rightarrow_2 \text{NP(lawyer) VP(examined)}$$

we have

$$\begin{aligned} X &= \text{S} \\ H &= \text{examined} \\ R &= \text{S} \rightarrow_2 \text{NP VP} \\ M &= \text{lawyer} \end{aligned}$$

With these definitions, the parameter for the rule has the following interpretation:

$$\begin{aligned} & q(\text{S(examined)} \rightarrow_2 \text{NP(lawyer) VP(examined)}) \\ = \ & P(R = \text{S} \rightarrow_2 \text{NP VP}, M = \text{lawyer} | X = \text{S}, H = \text{examined}) \end{aligned}$$

The first step in deriving an estimate of $q(\text{S(examined)} \rightarrow_2 \text{NP(lawyer) VP(examined)})$ will be to use the chain rule to decompose the above expression into two terms:

$$\begin{aligned} & P(R = \text{S} \rightarrow_2 \text{NP VP}, M = \text{lawyer} | X = \text{S}, H = \text{examined}) \\ = \ & P(R = \text{S} \rightarrow_2 \text{NP VP} | X = \text{S}, H = \text{examined}) \qquad (3) \\ & \times P(M = \text{lawyer} | R = \text{S} \rightarrow_2 \text{NP VP}, X = \text{S}, H = \text{examined}) \qquad (4) \end{aligned}$$

This step is exact, by the chain rule of probabilities.

We will now derive separate smoothed estimates of the quantities in Eqs. 3 and 4. First, for Eq. 3 define the following maximum-likelihood estimates:

$$q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S, examined}) = \frac{\text{count}(R = \text{S} \rightarrow_2 \text{NP VP}, X = \text{S}, H = \text{examined})}{\text{count}(X = \text{S}, H = \text{examined})}$$

$$q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S}) = \frac{\text{count}(R = \text{S} \rightarrow_2 \text{NP VP}, X = \text{S})}{\text{count}(X = \text{S})}$$

Here the count$(\ldots)$ expressions are counts derived directly from the training samples (the lexicalized trees in the treebank). Our estimate of

$$P(R = \text{S} \rightarrow_2 \text{NP VP}|X = \text{S}, H = \text{examined})$$

is then defined as

$$\lambda_1 \times q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S, examined}) + (1 - \lambda_1) \times q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S})$$

where λ_1 dictates the relative weights of the two estimates (we have $0 \leq \lambda_1 \leq 1$). The value for λ_1 can be estimated using the methods described in the notes on language modeling for this class.

Next, consider our estimate of the expression in Eq. 4. We can define the following two maximum-likelihood estimates:

$$q_{ML}(\text{lawyer}|\text{S} \rightarrow_2 \text{NP VP, examined}) = \frac{\text{count}(M = \text{lawyer}, R = \text{S} \rightarrow_2 \text{NP VP}, H = \text{examined})}{\text{count}(R = \text{S} \rightarrow_2 \text{NP VP}, H = \text{examined})}$$

$$q_{ML}(\text{lawyer}|\text{S} \rightarrow_2 \text{NP VP}) = \frac{\text{count}(M = \text{lawyer}, R = \text{S} \rightarrow_2 \text{NP VP})}{\text{count}(R = \text{S} \rightarrow_2 \text{NP VP})}$$

The estimate of

$$P(M = \text{lawyer}|R = \text{S} \rightarrow_2 \text{NP VP}, X = \text{S}, H = \text{examined})$$

is then

$$\lambda_2 \times q_{ML}(\text{lawyer}|\text{S} \rightarrow_2 \text{NP VP, examined}) + (1 - \lambda_2) \times q_{ML}(\text{lawyer}|\text{S} \rightarrow_2 \text{NP VP})$$

where $0 \leq \lambda_2 \leq 1$ is a parameter specifying the relative weights of the two terms.

Putting these estimates together, our final estimate of the rule parameter is as follows:

$$\begin{aligned} & q(\text{S(examined)} \rightarrow_2 \text{NP(lawyer) VP(examined)}) \\ = \; & (\lambda_1 \times q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S, examined}) + (1 - \lambda_1) \times q_{ML}(\text{S} \rightarrow_2 \text{NP VP}|\text{S})) \\ & \times (\lambda_2 \times q_{ML}(\text{lawyer}|\text{S} \rightarrow_2 \text{NP VP, examined}) + (1 - \lambda_2) \times q_{ML}(\text{lawyer}|\text{S} \rightarrow_2 \text{NP VP})) \end{aligned}$$

It can be seen that this estimate combines very lexically-specific information, for example the estimates

$$q_{ML}(\text{S} \rightarrow_2 \text{NP VP} | \text{S, examined})$$

$$q_{ML}(\text{lawyer} | \text{S} \rightarrow_2 \text{NP VP, examined})$$

with estimates that rely less on lexical information, for example

$$q_{ML}(\text{S} \rightarrow_2 \text{NP VP} | \text{S})$$

$$q_{ML}(\text{lawyer} | \text{S} \rightarrow_2 \text{NP VP})$$

The end result is a model that is sensitive to lexical information, but which is nevertheless robust, because we have used smoothed estimates of the very large number of parameters in the model.

6 Parsing with Lexicalized PCFGs

The parsing algorithm for lexicalized PCFGs is very similar to the parsing algorithm for regular PCFGs, as described in the previous lecture notes. Recall that for a regular PCFG the dynamic programming algorithm for parsing makes use of a dynamic programming table $\pi(i, j, X)$. Each entry $\pi(i, j, X)$ stores the highest probability for any parse tree rooted in non-terminal X, spanning words $i \ldots j$ inclusive in the input sentence. The π values can be completed using a recursive definition, as follows. Assume that the input sentence to the algorithm is $x_1 \ldots x_n$. The base case of the recursion is for $i = 1 \ldots n$, for all $X \in N$,

$$\pi(i, i, X) = q(X \rightarrow x_i)$$

where we define $q(X \rightarrow x_i) = 0$ if the rule $X \rightarrow x_i$ is not in the grammar.

The recursive definition is as follows: for any non-terminal X, for any i, j such that $1 \leq i < j \leq n$,

$$\pi(i, j, X) = \max_{X \rightarrow Y\ Z, s \in \{i \ldots (j-1)\}} q(X \rightarrow Y\ Z) \times \pi(i, s, Y) \times \pi(s + 1, j, Z)$$

Thus we have a $\max$ over all rules $X \rightarrow Y\ Z$, and all split-points $s \in \{i \ldots (j - 1)\}$. This recursion is justified because any parse tree rooted in X, spanning words $i \ldots j$, must be composed of the following choices:

- A rule $X \rightarrow Y\ Z$ at the root of the tree.
- A split point $s \in \{i \ldots (j - 1)\}$.

- A sub-tree rooted in Y, spanning words $\{i \dots s\}$.
- A sub-tree rooted in Z, spanning words $\{(s+1) \dots j\}$.

Now consider the case of lexicalized PCFGs. A key difference is that each non-terminal in the grammar includes a lexical item. A key observation is that for a given input sentence $x_1 \dots x_n$, parse trees for that sentence can only include non-terminals with lexical items that are one of $x_1 \dots x_n$.

Following this observation, we will define a dynamic programming table with entries $\pi(i, j, h, X)$ for $1 \leq i \leq j \leq n$, $h \in \{i \dots j\}$, $X \in N$, where N is the set of unlexicalized non-terminals in the grammar. We give the following definition:

Definition 2 (Dynamic programming table for lexicalized PCFGs.) *$\pi(i, j, h, X)$ is the highest probability for any parse tree with non-terminal X and lexical item h at its root, spanning words $i \dots j$ in the input.*

As an example, consider the following sentence from earlier in this note:

workers dumped the sacks into a bin

In this case we have $n = 7$ (there are seven words in the sentence). As one example entry,

$$\pi(2, 7, 2, VP)$$

will be the highest probability for any subtree rooted in $VP(dumped)$, spanning words $2 \dots 7$ in the sentence.

The π values can again be completed using a recursive definition. The base case is as follows:

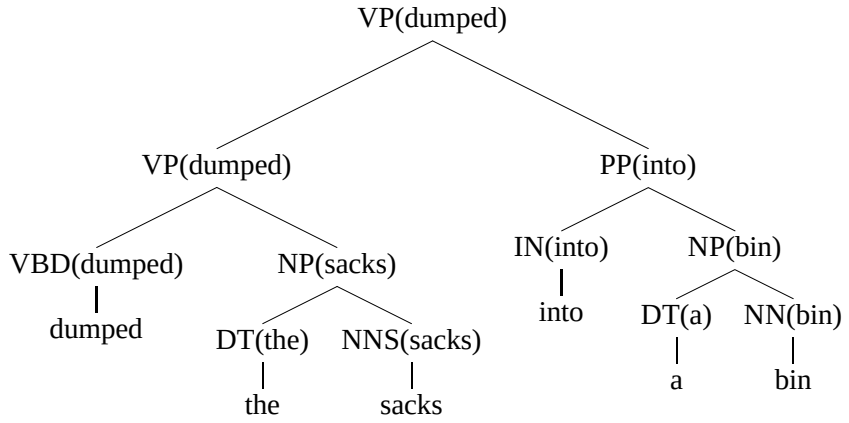
$$\pi(i, i, i, X) = q(X(x_i) \rightarrow x_i)$$

where we define $q(X(x_i) \rightarrow x_i) = 0$ if the rule $q(X(x_i) \rightarrow x_i)$ is not in the lexicalized PCFG. Note that this is very similar to the base case for regular PCFGs. As one example, we would set

$$\pi(1, 1, 1, \text{NNS}) = q(\text{NNS(workers)} \rightarrow \text{workers})$$

for the above example sentence.

We now consider the recursive definition. As an example, consider completing the value of $\pi(2, 7, 2, VP)$ for our example sentence. Consider the following sub-tree that spans words $2 \dots 7$, has lexical item $x_2 =$ dumped and label VP at its root:



We can see that this subtree has the following sub-parts:

- A choice of split-point $s \in \{1 \ldots 6\}$. In this case we choose $s = 4$ (the split between the two subtrees under the rule VP(dumped) $\rightarrow_1$ VBD(dumped) PP(into) is after $x_4 =$ sacks).
- A choice of modifier word m. In this case we choose $m = 5$, corresponding to $x_5 =$ into, because x_5 is the head word of the second child of the rule VP(dumped) $\rightarrow_1$ VBD(dumped) PP(into).
- A choice of rule at the root of the tree: in this case the rule is VP(dumped) $\rightarrow_1$ VBD(dumped) PP(into).

More generally, to find the value for any $\pi(i, j, h, X)$, we need to search over all possible choices for s, m, and all rules of the form $X(x_h) \rightarrow_1 Y_1(x_h)\ Y_2(x_m)$ or $X(x_h) \rightarrow_2 Y_1(x_m)\ Y_2(x_h)$. Figure 8 shows pseudo-code for this step. Note that some care is needed when enumerating the possible values for s and m. If s is in the range $h \ldots (j-1)$ then the head word h must come from the left sub-tree; it follows that m must come from the right sub-tree, and hence must be in the range $(s+1) \ldots j$. Conversely, if s is in the range $i \ldots (h-1)$ then m must be in the left sub-tree, i.e., in the range $i \ldots s$. The pseudo-code in figure 8 treats these two cases separately.

Figure 9 gives the full algorithm for parsing with lexicalized PCFGs. The algorithm first completes the base case of the recursive definition for the π values. It then fills in the rest of the π values, starting with the case where $j = i + 1$, then the case $j = i + 2$, and so on. Finally, the step

$$(X^*, h^*) = \arg \max_{X \in N, h \in \{1 \ldots n\}} \gamma(X, h) \times \pi(1, n, h, X)$$

finds the pair $X^*(h^*)$ which is at the root of the most probable tree for the input sentence: note that the γ term is taken into account at this step. The highest probability tree can then be recovered by following backpointers starting at $bp(1, n, h^*, X^*)$.

1. $\pi(i, j, h, X) = 0$
2. For $s = h \ldots (j-1)$, for $m = (s+1) \ldots j$, for $X(x_h) \rightarrow_1 Y(x_h)Z(x_m) \in R$,
   (a) $p = q(X(x_h) \rightarrow_1 Y(x_h)Z(x_m)) \times \pi(i, s, h, Y) \times \pi(s+1, j, m, Z)$
   (b) If $p > \pi(i, j, h, X)$,
   $$\pi(i, j, h, X) = p$$
   $$bp(i, j, h, X) = \langle s, m, Y, Z \rangle$$
3. For $s = i \ldots (h-1)$, for $m = i \ldots s$, for $X(x_h) \rightarrow_2 Y(x_m)Z(x_h) \in R$,
   (a) $p = q(X(x_h) \rightarrow_2 Y(x_m)Z(x_h)) \times \pi(i, s, m, Y) \times \pi(s+1, j, h, Z)$
   (b) If $p > \pi(i, j, h, X)$,
   $$\pi(i, j, h, X) = p$$
   $$bp(i, j, h, X) = \langle s, m, Y, Z \rangle$$

Figure 8: The method for calculating an entry $\pi(i, j, h, X)$ in the dynamic programming table. The pseudo-code searches over all split-points s, over all modifier positions m, and over all rules of the form $X(x_h) \rightarrow_1 Y(x_h)\ Z(x_m)$ or $X(x_h) \rightarrow_2 Y(x_m)\ Z(x_h)$. The algorithm stores backpointer values $bp(i, j, h, X)$.

Input: a sentence $s = x_1 \dots x_n$, a lexicalized PCFG $G = (N, \Sigma, S, R, q, \gamma)$.
Initialization:
For all $i \in \{1 \dots n\}$, for all $X \in N$,

$$\pi(i, i, i, X) = \begin{cases} q(X(x_i) \rightarrow x_i) & \text{if } X(x_i) \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- For $l = 1 \dots (n-1)$
  - For $i = 1 \dots (n-l)$
    - Set $j = i + l$
    - For all $X \in N$, $h \in \{i \dots j\}$, calculate $\pi(i, j, h, X)$ using the algorithm in figure 8.

Output:

$$(X^*, h^*) = \arg \max_{S \in N, h \in \{1 \dots n\}} \gamma(X, h) \times \pi(1, n, h, X)$$

Use backpointers starting at $bp(1, n, h^*, X^*)$ to obtain the highest probability tree.

Figure 9: The CKY parsing algorithm for lexicalized PCFGs.