Today: Probabilistic Parsing

Goal: Find the most likely parse.

1. Parsing with PCFGs
2. Problems
3. Probabilistic lexicalized CFGs

CFG's

A context free grammar consists of:

1. a set of non-terminal symbols N
2. a set of terminal symbols Σ (disjoint from N)
3. a set of productions, P, each of the form $A \rightarrow \alpha$, where A is a non-terminal and α is a string of symbols from the infinite set of strings $(\Sigma \cup N)$
4. a designated start symbol S

Probabilistic CFGs

Augments each rule in P with a conditional probability:

$A \rightarrow \beta\ [p]$

where p is the probability that the non-terminal A will be expanded to the sequence β. Often referred to as

$P(A \rightarrow \beta)$ or

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

Example

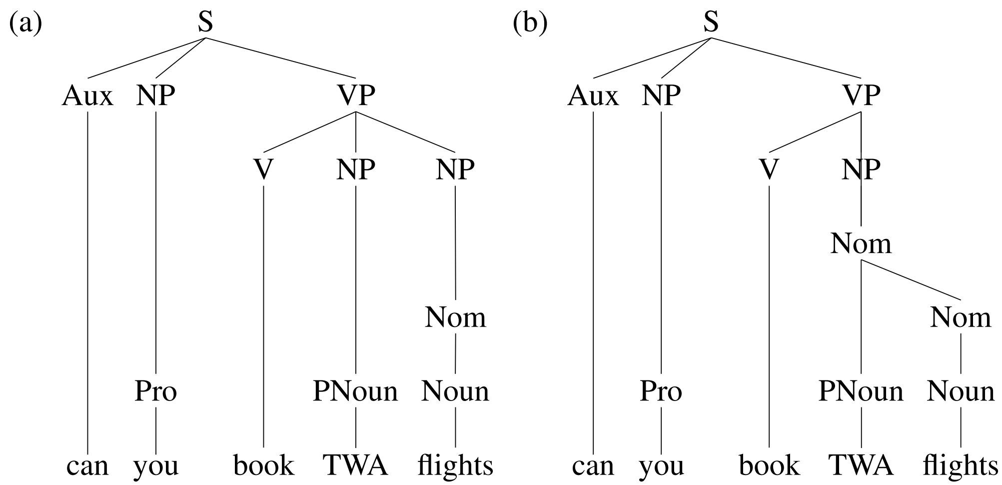
$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow that\,[.05] \mid the\,[.80] \mid a\,[.15]$	
$S \rightarrow Aux\ NP\ VP$	[.15]	$Noun \rightarrow book$	[.10]
$S \rightarrow VP$	[.05]	$Noun \rightarrow flights$	[.50]
$NP \rightarrow Det\ Nom$	[.20]	$Noun \rightarrow meal$	[.40]
$NP \rightarrow Proper\text{-}Noun$	[.35]	$Verb \rightarrow book$	[.30]
$NP \rightarrow Nom$	[.05]	$Verb \rightarrow include$	[.30]
$NP \rightarrow Pronoun$	[.40]	$Verb \rightarrow want$	[.40]
$Nom \rightarrow Noun$	[.75]	$Aux \rightarrow can$	[.40]
$Nom \rightarrow Noun\ Nom$	[.20]	$Aux \rightarrow does$	[.30]
$Nom \rightarrow Proper\text{-}Noun\ Nom$	[.05]	$Aux \rightarrow do$	[.30]
$VP \rightarrow Verb$	[.55]	$Proper\text{-}Noun \rightarrow TWA$	[.40]
$VP \rightarrow Verb\ NP$	[.40]	$Proper\text{-}Noun \rightarrow Denver$	[.40] .60
$VP \rightarrow Verb\ NP\ NP$	[.05]	$Pronoun \rightarrow you\,[.40] \mid I\,[.60]$	

Why are PCFGs useful?

- Assigns a probability to each parse tree T
- Useful in **disambiguation**
  - Choose the most likely parse
  - Computing the probability of a parse

    If we make independence assumptions, $\mathrm{P(T)} = \prod_{n \in T} \mathrm{p(r(n))}$.
- Useful in **language modeling** tasks

Example



Rules			P	Rules			P
S	→	Aux NP VP	.15	S	→	Aux NP VP	.15
NP	→	Pro	.40	NP	→	Pro	.40
VP	→	V NP NP	.05	VP	→	V NP	.40
NP	→	Nom	.05	NP	→	Nom	.05
NP	→	PNoun	.35	Nom	→	PNoun Nom	.05
Nom	→	Noun	.75	Nom	→	Noun	.75
Aux	→	Can	.40	Aux	→	Can	.40
NP	→	Pro	.40	NP	→	Pro	.40
Pro	→	you	.40	Pro	→	you	.40
Verb	→	book	.30	Verb	→	book	.30
PNoun	→	TWA	.40	Pnoun	→	TWA	.40
Noun	→	flights	.50	Noun	→	flights	.50

Where does the grammar come from?

1. developed manually
2. from a **treebank**

Treebanks

- Corpus with *sentence - parse tree* (presumably the right one) *pairs*.
- Penn TreeBank a widely used treebank.

▪Most well known is the Wall Street Journal section of the Penn TreeBank.

  ▪1 M words from the 1987-1989 Wall Street Journal.

```
( (S (`` ``)
    (S-TPC-2
      (NP-SBJ-1 (PRP We) )
      (VP (MD would)
        (VP (VB have)
          (S
            (NP-SBJ (-NONE- *-1) )
            (VP (TO to)
              (VP (VB wait)
                (SBAR-TMP (IN until)
                  (S
                    (NP-SBJ (PRP we) )
                    (VP (VBP have)
                      (VP (VBN collected)
                        (PP-CLR (IN on)
                          (NP (DT those)(NNS assets)))))))))))))
    (, ,) ('' '')
    (NP-SBJ (PRP he) )
    (VP (VBD said)
      (S (-NONE- *T*-2) ))
    (. .) ))
```

Treebanks

- How are they created?
  - Parse the collection with an automatic parser
  - Manually correct each parse as necessary.
- Requires detailed annotation guidelines that provide
  - a POS tagset
  - a grammar
  - instructions for how to deal with particular grammatical constructions.

Treebank Grammars

- Treebanks implicitly define a grammar.
- Simply take the local rules that make up the sub-trees in all the trees in the collection and you have a grammar.
- Not complete, but if you have decent size corpus, you'll have a grammar with decent coverage.

Treebank Grammars

- Tend to be very flat due to the fact that they tend to avoid recursion.
  - To ease the annotators burden
- For example, the Penn Treebank has 4500 different rules for VPs. Among them...

$$
\begin{array}{l}
\mathtt{VP} \rightarrow \mathtt{VBD\ PP} \\
\mathtt{VP} \rightarrow \mathtt{VBD\ PP\ PP} \\
\mathtt{VP} \rightarrow \mathtt{VBD\ PP\ PP\ PP} \\
\mathtt{VP} \rightarrow \mathtt{VBD\ PP\ PP\ PP\ PP}
\end{array}
$$

Where do the probabilities come from?

1. from a **treebank**:

$$P(\alpha \to \beta|\alpha) = Count(\alpha \to \beta)/Count(\alpha)$$

2. use EM (forward-backward algorithm, inside-outside algorithm)

Parsing with PCFGs

Produce the most likely parse for a given sentence:

$$\hat{T}(S) = argmax_{T \in \tau(S)} P(T)$$

where $\tau(S)$ is the set of possible parse trees for S.

- Augment the Earley algorithm to compute the probability of each of its parses.

  When adding an entry E of category C to the chart using rule i with n subconstituents, $E_1, \ldots, E_n$:

  $P(E) = P(rule\ i \mid C) * P(E_1) * \ldots * P(E_n)$

- probabilistic CKY (Cocke-Kasami-Younger) algorithm

Problems with PCFGs

Do not model *structural dependencies*.

Often the choice of how a non-terminal expands depends on the location of the node in the parse tree.

E.g. Strong tendency in English for the syntactic subject of a spoken sentence to be a pronoun.

- 91% of declarative sentences in the Switchboard corpus are pronouns (vs. lexical).
- In contrast, 34% of direct objects in Switchboard are pronouns.

Problems with PCFGs

Do not adequately model *lexical dependencies*.

Moscow sent more than 100,000 soldiers into Afghanistan...

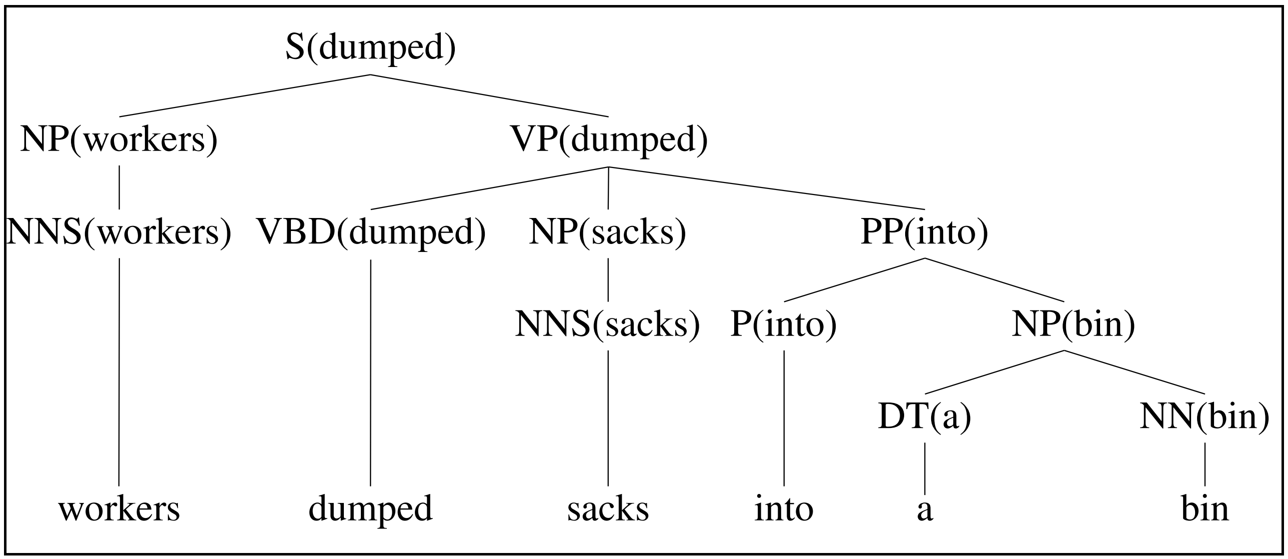
PP can attach to either the NP or the VP:
NP $\rightarrow$ NP PP or VP $\rightarrow$ V NP PP?

Attachment choice depends (in part) on the verb: *send* subcategorizes for a destination (e.g. expressed via a PP that begins with *into* or *to* or ...).

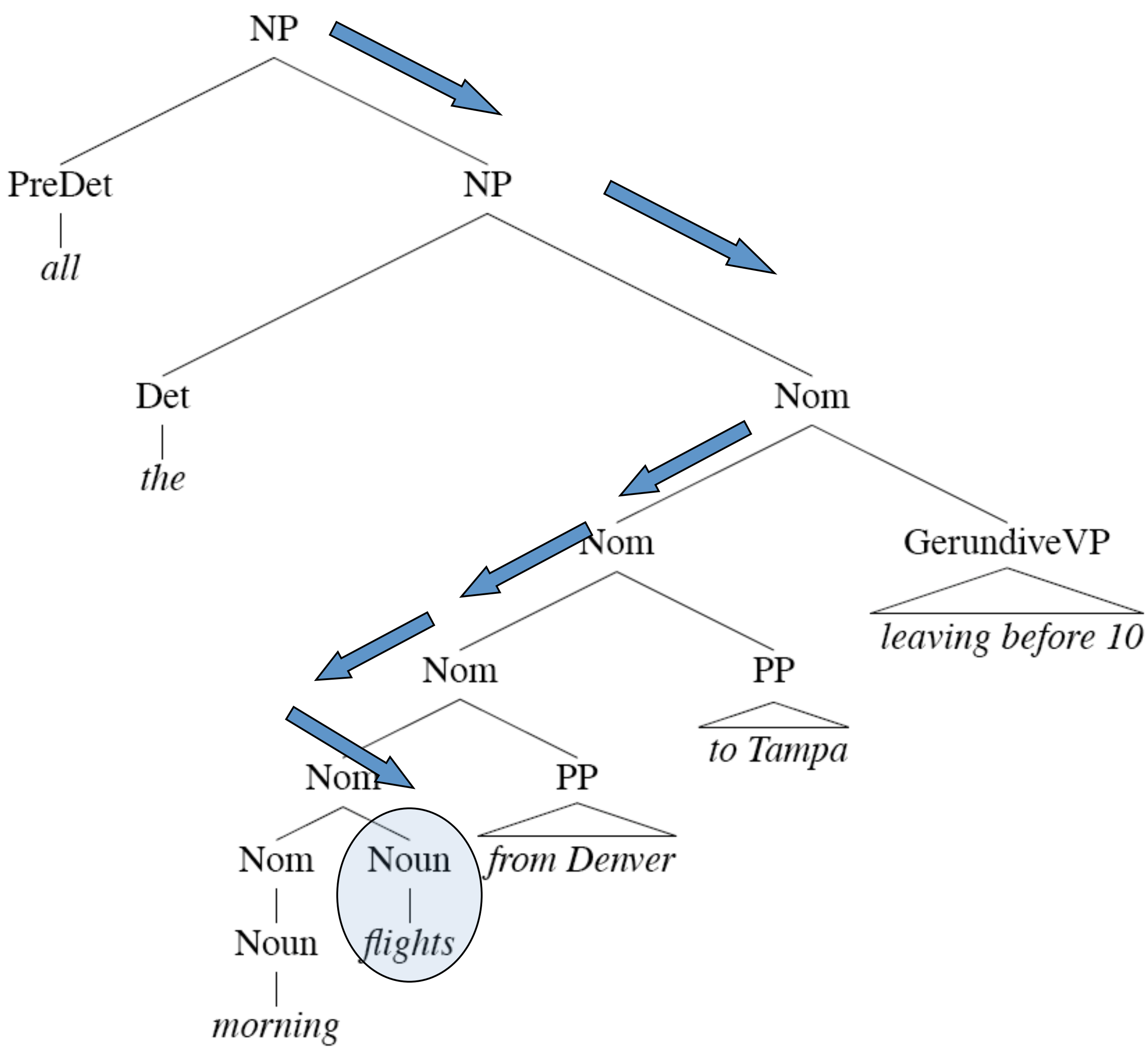
Probabilistic lexicalized CFGs

- Each non-terminal is associated with its head.
- Each PCFG rule needs to be augmented to identify one rhs constituent to be the head daughter.
- Headword for a node in the parse tree is set to the headword of its head daughter.

Example


S(dumped)
NP(workers)
VP(dumped)
NNS(workers)
VBD(dumped)
NP(sacks)
PP(into)
NNS(sacks)
P(into)
NP(bin)
DT(a)
NN(bin)
workers
dumped
sacks
into
a
bin

Noun Phrases



Probabilistic lexicalized CFGs

View a lexicalized (P)CFG as a simple (P)CFG with a lot more rules.

VP(dumped) $\rightarrow$ VBD(dumped) NP(sacks) PP(into) [$3 \text{x} 10^{-10}$]
VP(dumped) $\rightarrow$ VBD(dumped) NP(cats) PP(into) [$8 \text{x} 10^{-10}$]
VP(dumped) $\rightarrow$ VBD(dumped) NP(sacks) PP(above) [$1 \text{x} 10^{-12}$]
...

Problem?

Evaluation Measures and State of the Art

- labeled recall: # correct constituents in candidate parse of s / # correct constituents in treebank parse of s
- labeled precision: # correct constituents in candidate parse of s / total # of constituents in candidate parse of s
- crossing brackets: the number of crossed brackets

State of the art: 90% recall, 90% precision, 1% crossed bracketed constituents per sentence (WSJ treebank)