Faculty for Informatics

Technical University of Munich

Natural Language Processing
IN2361

Prof. Dr. Georg Groh

Social Computing
Research Group

Chapter 14
Statistical
Constituency Parsing

- content is based on [1]
- certain elements (e.g. equations or tables) were taken over or taken over in a modified form from [1]
- citations of [1] or from [1] are omitted for legibility
- errors are fully in the responsibility of Georg Groh
- BIG thanks to Dan and James for a great book!

Probabilistic Context Free Grammars

- Parsing a sentence can lead to ambiguities (e.g. coordination or attachment ambiguities) → use the most probable interpretation → probabilistic parsing

- Probabilistic CF grammars (PCFGs):

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 $P(\beta \mid A)$
S	a designated **start symbol**

possible notations:

$P(A \rightarrow \beta)$

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

$P(RHS \mid LHS)$

- For each A, we must have $\sum_{\beta} P(A \rightarrow \beta) = 1$

- PCFG "consistent" if the probability of all possible sentences sums to 1: $\sum_i P(s_i) = 1$ where $\forall i \; S \overset{*}{\underset{G}{\Rightarrow}} s_i$

PCFGs for Disambiguation

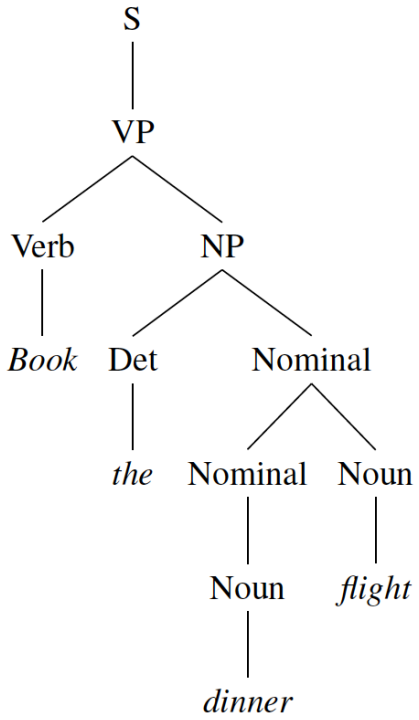
"Book the dinner flight"

For a parse tree T of a sentence S we have

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

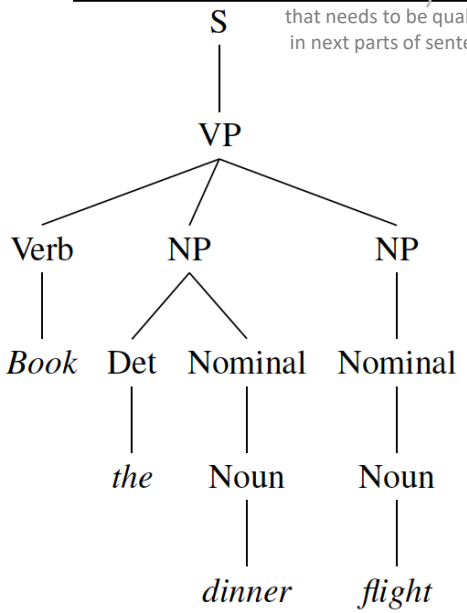
$$= P(T)\underbrace{P(S|T)}_{=1} = P(T)$$

"Book a flight that serves dinner"



"Book a flight on behalf of 'the dinner'"

that needs to be qualified further in next parts of sentence



$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$

$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$

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

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

PCFGs for Disambiguation

- denote by $S = \text{yield}(T)$ as the sentence S produced by a tree T →

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

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

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

P(S) is constant

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

from previous slide

PCFGs for Language Modeling

- probability of a sentence $S = w_1 w_2 \ldots w_n$:

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

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

- predicting a word from previous words using bigram assumption

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

discards a lot of useful context, that PCFGs allow to (iteratively) maintain.

Example:

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

Bigram model can only use *cents* to predict *after*, however *ended* and *contract* are certainly useful here.

Probabilistic CKY Parsing of PCFGs

probabilistic CKY algorithm to yield $\hat{T}(S) = \underset{T s.t. S=\text{yield}(T)}{\text{argmax}} P(T)$

Modify CKY:

- each cell [i,j] exists in versions [i,j,A] for each possible non-terminal. Cell [i,j,A] stores a probability
- when working up+right update cell [i,j,A] if necessary (if rule A → B C has higher probability where B ∈ cell [i,k,B] and C ∈ cell [k,j,C] ; then also store B, C, and k in [i,j,A] for reconstructing parse later)

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]

Probabilistic CKY Parsing of PCFGs

The	*flight*	*includes*	*a*	*meal*
Det: .40 [0,1]	NP: .30 *.40 *.02 = .0024 [0,2]	[0,3]	[0,4]	[0,5]
	N: .02 [1,2]	[1,3]	[1,4]	
		V: .05 [2,3]	[2,4]	[2,5]
			Det: .40 [3,4]	[3,5]
				N: .01 [4,5]

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

Learn PCFG Rule Probabilities

- If we have a Treebank → Probabilities by counting

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

- If we have a non-probabilistic parser:
  - typically multiple parse trees T_i for a sentence exist
  - → weight each $\text{Count}(\alpha \rightarrow \beta)$ with $P(T_i)$.
  - But we need $P(\alpha \rightarrow \beta|\alpha)$ to compute $P(T_i)$
  - → use EM algorithm variant (“Inside-Out-algorithm”)

Problems with PCFGs: Independence Assumptions

- poor independence assumptions: CFG rules ignore context A→ BC
  + independence assumption on probabilities $P(T,S) = \prod P(RHS_i|LHS_i)$
  → poor modelling of structural dependencies across parse tree.

- example: NPs that are **subjects** are more likely to be pronoun;
  NPs that are **objects** more likely to be non-pronoun (e.g. Det + NN)

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

statistics on Switchboard corpus

however, for probabilities we must use non-context-aware "prior" "overall" probabilities

$$NP \rightarrow DT\ NN \quad .28$$
$$NP \rightarrow PRP \quad .25$$

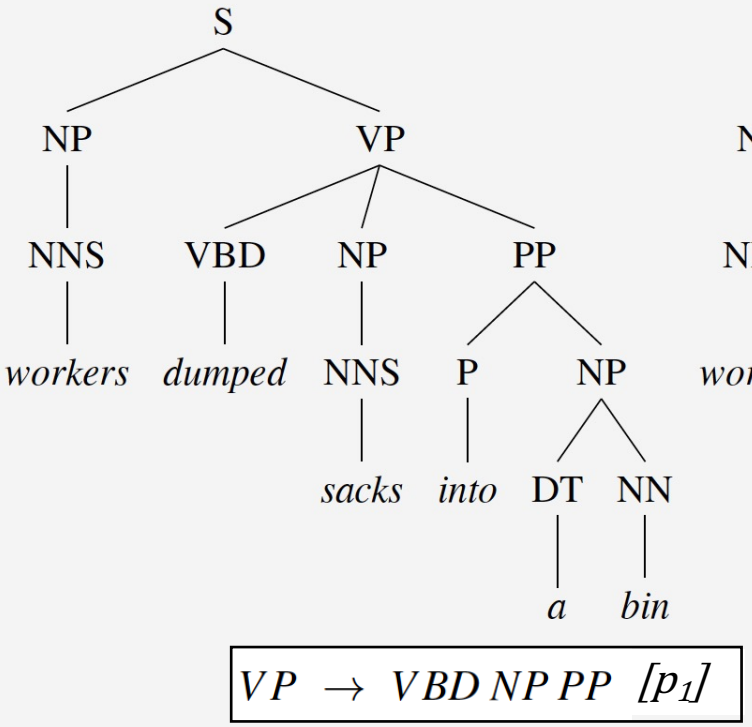
possible solution: parent annotation (see later slides)

Problems with PCFGs: Lack of Lexical Conditioning

- lack of lexical conditioning: CFG rules don't model syntactic facts about specific words → problems with
  - preposition attachment
  - coordinate structure ambiguities
  - etc

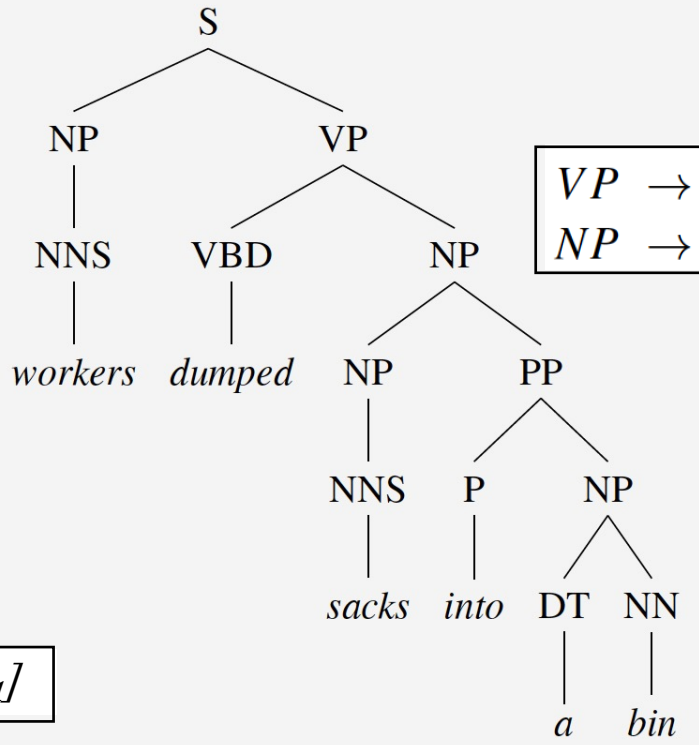
Example: Preposition Attachment

VP attachment (correct): "dump the sacks where? into the bin"



$VP \rightarrow VBD\ NP\ PP\ [p_1]$

NP attachment "the sacks that have the 'into the bin' quality"



$VP \rightarrow VBD\ NP\ [p_2]$

$NP \rightarrow NP\ PP\ [p_3]$

depending on p_1 p_2 p_3 PCFG will always prefer VP attachment or NP attachment. Since NP attachment is slightly more common → always prefer NP attachment

Example: Preposition Attachment

- *workers dumped sacks into a bin* → VP attachment required
  fishermen caught tons of herring → NP attachment required

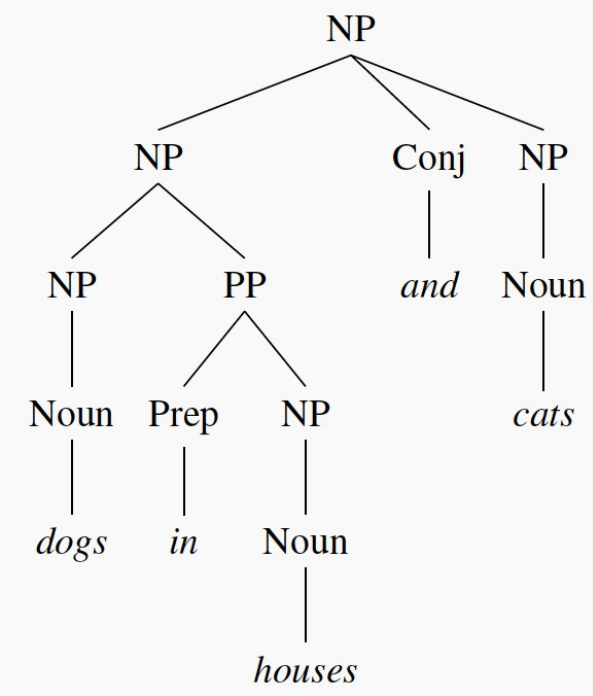
- where is the required information encoded?
  In the words themselves:

  association *dumped* ↔ *into* stronger than *sacks* ↔ *into*
  association *tons* ↔ *of* stronger than *caught* ↔ *of*

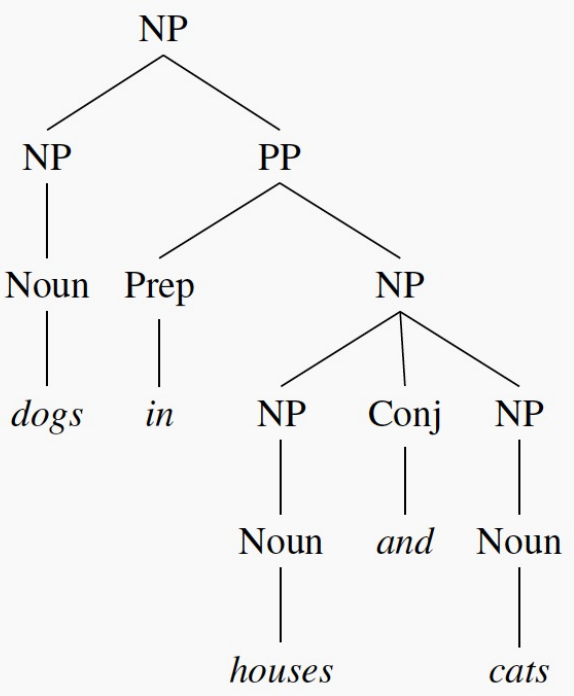
  → lexical dependency of probabilities required

Example: Coordination Ambiguity

"cats (wherever) & dogs in houses"



"dogs in cats & dogs in houses"

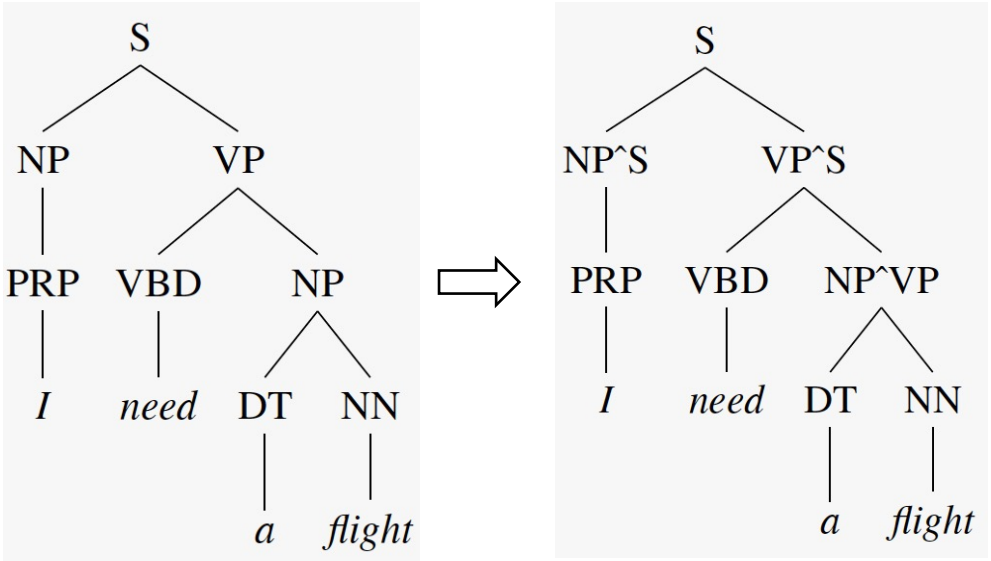


rules are identical for both trees → $P(T_1)=P(T_2)$

but *dogs* is a more natural coordination partner for *cats* than *houses*

Improve Preposition Attachment: Split Non-Terminals

- idea: split NP into $NP_{subject}$ and NP_{object}; then $\begin{cases} NP_{subject} \rightarrow PRP & [p=0.91] \\ NP_{object} \rightarrow PRP & [p=0.34] \end{cases}$
- implemented by parent annotation:



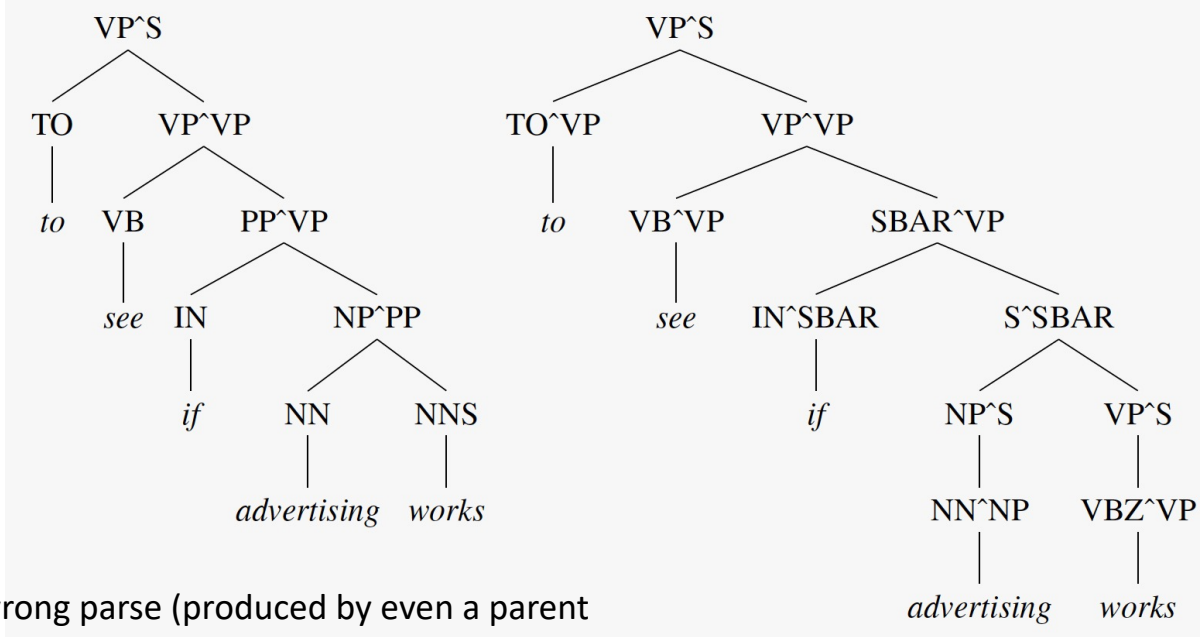
- example: Penn Treebank tag IN: can mark wide variety of words: subordinating conjunctions (*while, as, if*), complementizers (*that, for*), prepositions (*of, in, from*)

  → use parent annotation: IN^S: subordinating conjunctions occur under S, IN^PP prepositions under prepositional phrase etc.

Parent Annotation

- possible: also split (parent-annotate) the pre-terminal ("POS") nodes (PRP, VBD, DT, NN in previous example). example: most common adverbs (RB) are:
  - *also* and *now* when RB has ADVP parent (RB^ADVP)
  - *n't* and *not* when RB has VP parent (RB^VP)
  - *only* and *just* when RB has NP parent (RB^NP)

- however, even parent annotation is sometimes not enough: example:



wrong parse (produced by even a parent annotated parser): *advertising* associated as a noun with other noun *works*

correct parse: *advertising works* as a sentence

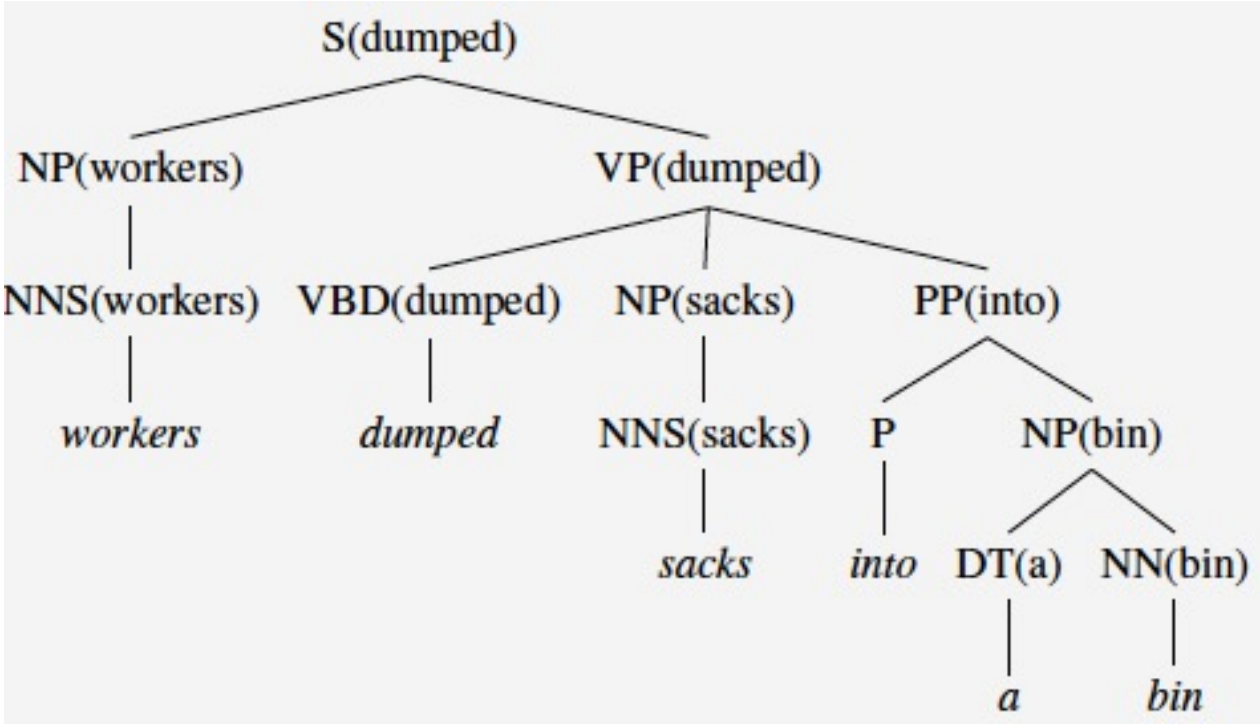
Problems of Node Splitting

- node splitting → number of grammar rules increases →
  if not enough data: overfitting

- why not let an algorithm decide on if and where to split a node or merge split nodes (if necessary): Split and Merge algorithm (Petrov 2006): still state of the art parser on Penn Treebank

Probabilistic Lexicalized CFGs

- lexicalized grammars: annotate non-terminal with corresponding head word:

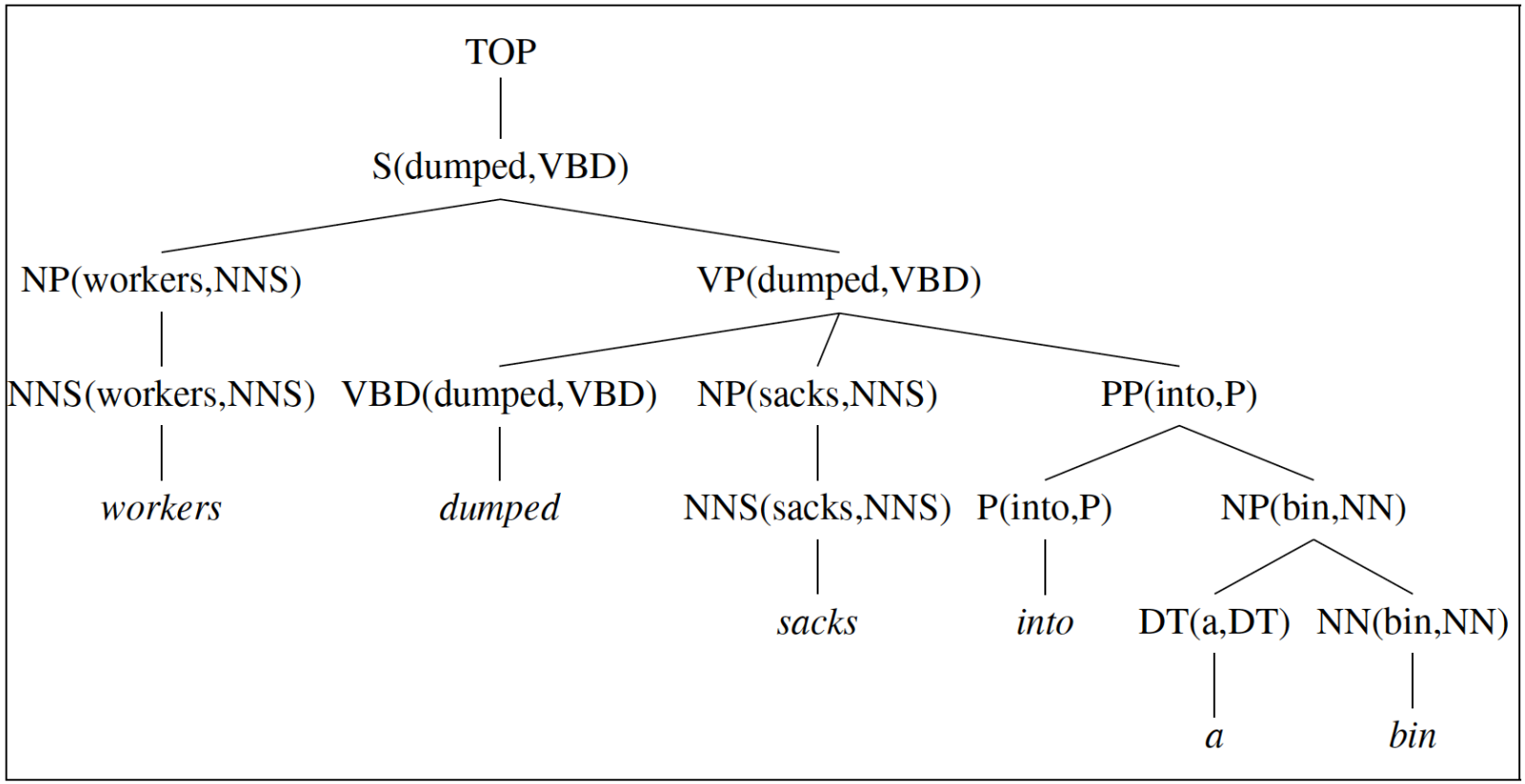


- rules then look like e.g.

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

Probabilistic Lexicalized CFGs

- standard way: augment not only with head word but also with POS tag (pre-terminal non-terminal) of head word



Internal Rules

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

Lexical Rules

NNS(workers,NNS)	→	workers
VBD(dumped,VBD)	→	dumped
NNS(sacks,NNS)	→	sacks
P(into,P)	→	into
DT(a,DT)	→	a
NN(bin,NN)	→	bin

Probabilistic Lexicalized CFGs

- lexical rules like $\mathrm{NN(bin,NN)} \rightarrow \mathrm{bin}$ have probability = 1

- Problem: internal rules like

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

are way to specific to apply MLE estimation for corresponding probabilities

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

- → modify probabilistic independence assumptions → Collins Parser, Charniak Parser

Collins Parser

- Perceive rules such as

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

in the following way:

$$LHS \rightarrow L_n L_{n-1} \dots L_1 H R_1 \dots R_{n-1} R_n$$

where L_n and R_n are STOP symbols

- and "generate" probabilities for rule from/conditioning on LHS: starting with head word then left symbols then right symbols, involving "sorts" of probabilities: P_H, P_L, P_R:

$$P(VP(dumped,VBD) \rightarrow \underbrace{}_{L_1 = STOP}\ \underbrace{VBD(dumped, VBD)}_{H}\ \underbrace{NP(sacks,NNS)}_{R_1}\ \underbrace{PP(into,P))}_{R_2}\ \underbrace{}_{R_3 = STOP} =$$

$P_H(VBD(dumped,VBD) \mid VP,VBD,dumped) := P_H(VBD \mid VP,dumped)$

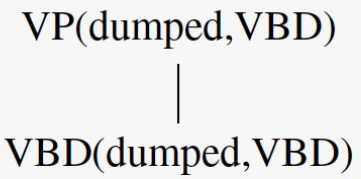
$$P_H(VBD|VP, dumped) \times P_L(STOP|VP,VBD,dumped)$$
$$\times P_R(NP(sacks,NNS)|VP,VBD,dumped) \times P_R(PP(into,P)|VP,VBD,dumped)$$
$$\times P_R(STOP|VP,VBD,dumped)$$

example: generate $P(VP(dumped,VBD) \rightarrow$

STOP *VBD(dumped, VBD) NP(sacks,NNS) PP(into,P)* STOP$)$

1. Generate the head VBD(dumped,VBD) with probability

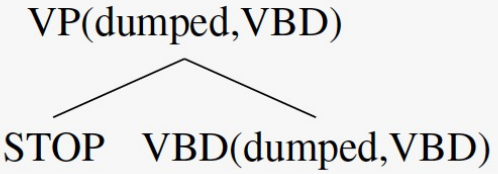
P_H(H|LHS) = P(VBD(dumped,VBD) | VP(dumped,VBD))



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

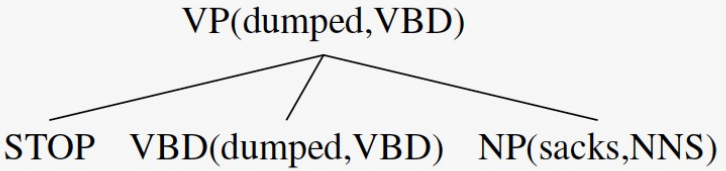
P_L(STOP| VP(dumped,VBD) VBD(dumped,VBD))

abbreviated on previous slide as: "$VP, dumped, VBD$"



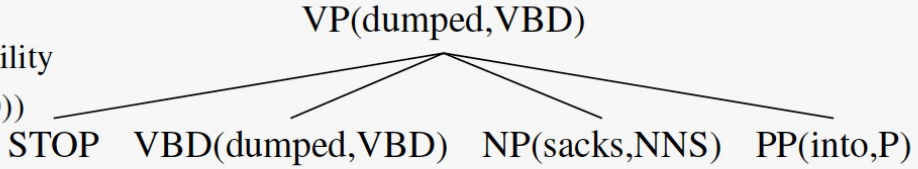
3. Generate right dependent NP(sacks,NNS) with probability

P_r(NP(sacks,NNS)| VP(dumped,VBD), VBD(dumped,VBD))



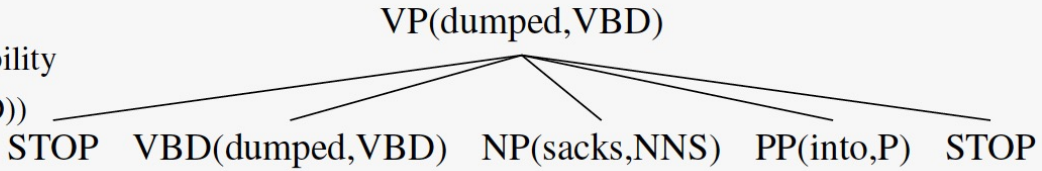
4. Generate the right dependent PP(into,P) with probability

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))



example: generate $P(VP(dumped,VBD) \rightarrow$

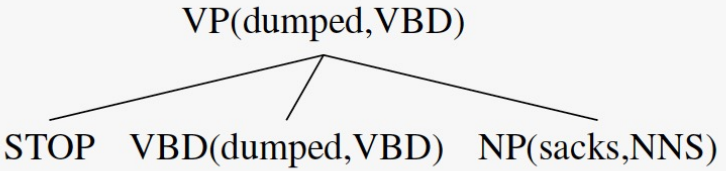
STOP $VBD(dumped, VBD)$ $NP(sacks,NNS)$ $PP(into,P)$ STOP$)$

advantage: involved counts are much less sparse:

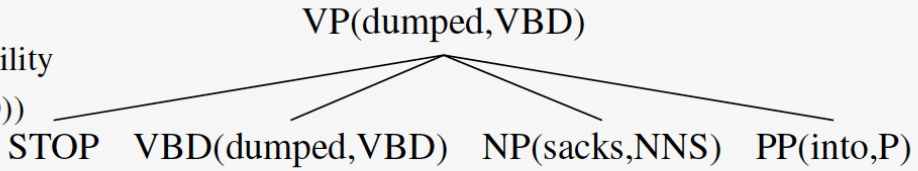
$$P_R(NP(sacks,NNS)|VP, VBD, dumped) \approx$$

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

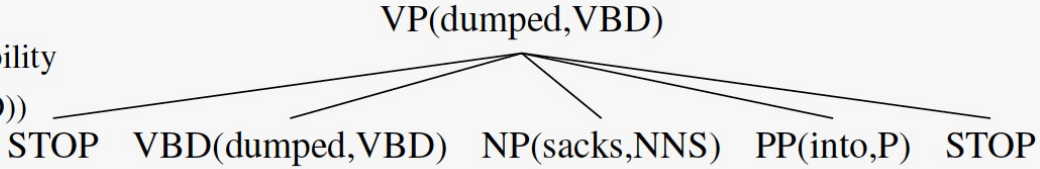
3. Generate right dependent NP(sacks,NNS) with probability
P_r(NP(sacks,NNS)| VP(dumped,VBD), VBD(dumped,VBD))



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))



Calculate probability for rule $LHS \rightarrow L_n L_{n-1} \dots L_1 H R_1 \dots R_{n-1} R_n =$

$P(hw, ht) \rightarrow STOP\ L_{n-1}(lw_{n-1}, lt_{n-1}) \dots L_1\ (lw_1, lt_1) H(hw, ht) R_1(rw_1, rt_1) \dots R_{n-1}\ (rw_{n-1}, rt_{n-1}) STOP$

1. Generate the head of the phrase $H(hw, ht)$ with probability:

$$P_H(H(hw, ht)|P, hw, ht)$$

H: non-terminal POS symbol of head

P="parent" (LHS-non-terminal symbol)

2. Generate modifiers to the left of the head with total probability

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

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

3. Generate modifiers to the right of the head with total probability:

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

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

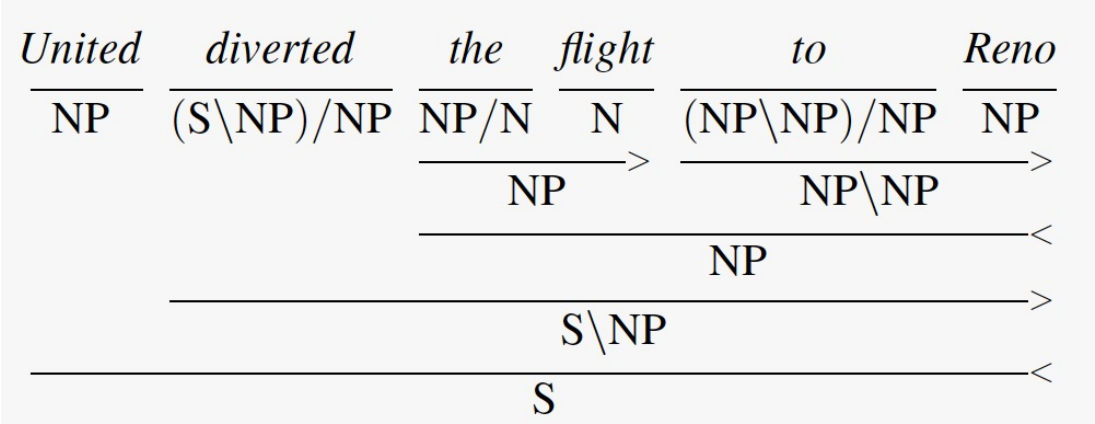
Probabilistic Combinatorial Categorial Grammars (CCGs)

- ambiguity example: *United diverted the flight to Reno* :

traditional CFG: choice of internal rule determines parse

$Nominal \rightarrow Nominal\ PP$	United diverted [the flight to Reno]
$VP \rightarrow VP\ PP$	United [diverted to Reno] the flight
$VP \rightarrow Verb\ NP\ PP$	United [diverted the flight to Reno]

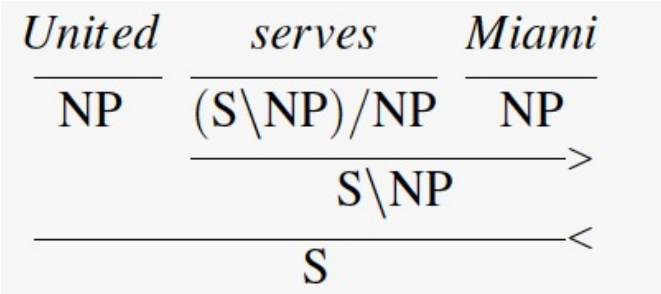
traditional CCG (lexicalized grammar): choice of lexical rule determines parse



Repetition: Combinatory Categorial Grammars

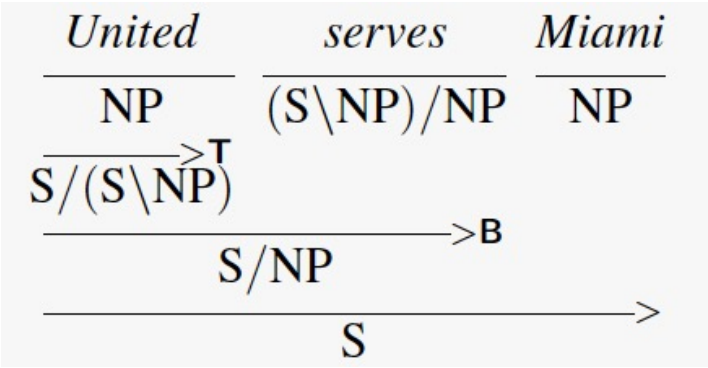
- $X/Y \;\; Y \;\Rightarrow\; X$ $\quad$ X/Y : function that seeks its argument to the right
  $Y \;\; X\backslash Y \;\Rightarrow\; X$ $\quad$ $X\backslash Y$: function that seeks its argument to the left

- Lexicon: $\quad$ *flight* : N
  Miami : NP
  cancel : $(S\backslash NP)/NP$
  (transitive verb (one object))



- function composition B: $\quad$ $X/Y \;\; Y/Z \;\Rightarrow\; X/Z$
  $Y\backslash Z \;\; X\backslash Y \;\Rightarrow\; X\backslash Z$

- type raising T: $\quad$ $X \;\Rightarrow\; T/(T\backslash X)$
  $X \;\Rightarrow\; T\backslash(T/X)$
  with T any category

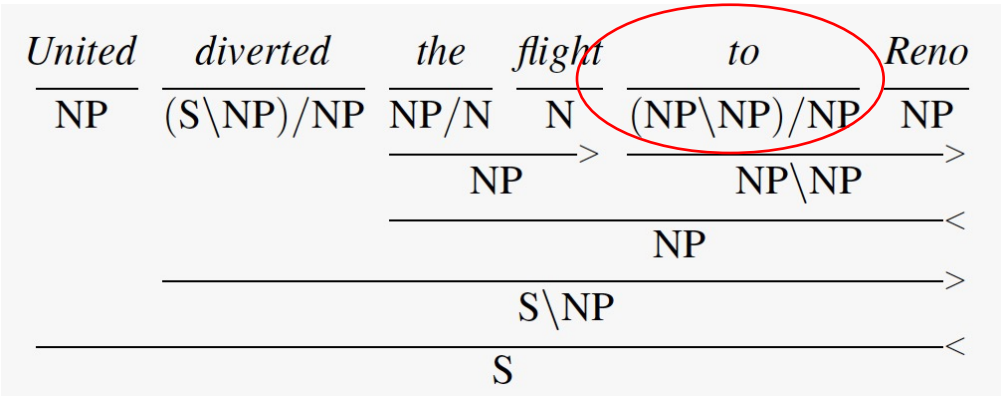


Probabilistic Combinatorial Categorial Grammars (CCGs)

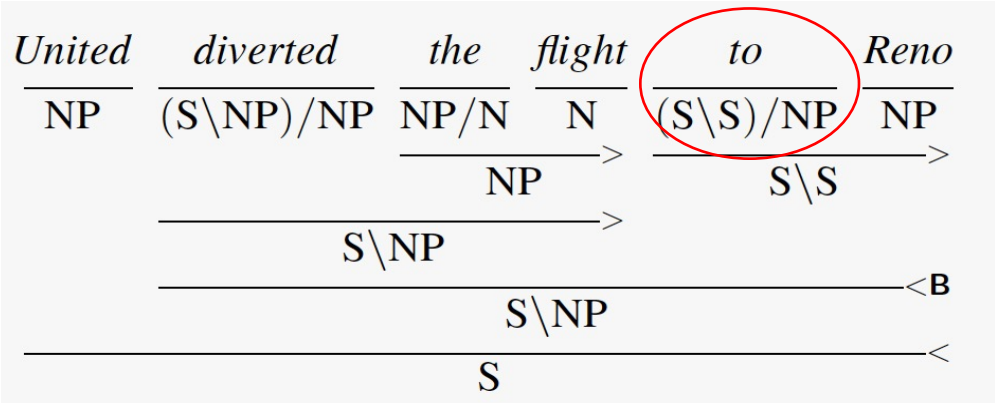
- ambiguity example (ctd.): *United diverted the flight to Reno* :

  traditional CCG (lexicalized grammar): choice of lexical rule determines parse (ctd.)

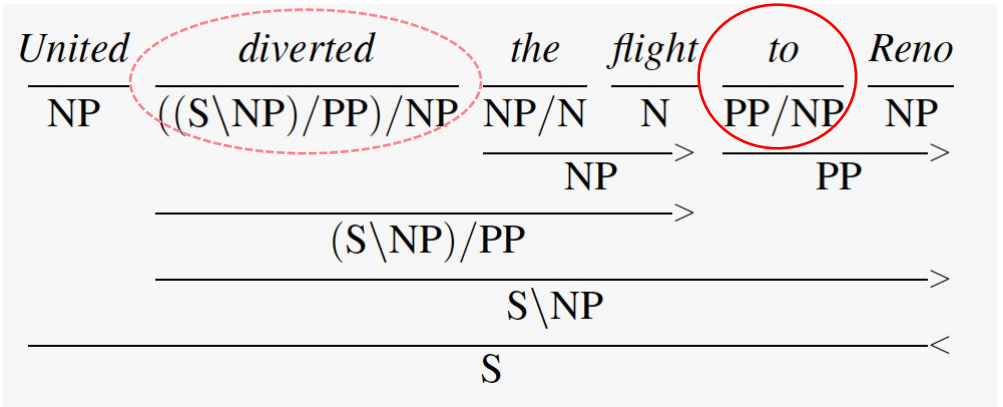
to Reno modifies *flight*



to Reno modifies the verb-phrase: *to* starting a subordinate sentence



divert as di-transitive verb and *to Reno* as simple prepositional phrase argument



PCYK-Style Parsing for CCGs

- PCYK parsing idea: in each meta-cell [i,j,A] keep track of most probable derivation for A

- for CCGs: a lot of possible lexical categories (lexical rules) for each word AND very few but very generally applicable internal rules → combinatorial explosion: each cell [i,j] will exist in a lot of variations $[i,j,A^{(n)}]$ (most of them useless (zombie constitutents))

- → possible remedy: accurately assess and exploit only the most likely lexical categories for each word: Supertagging as a previous step before parsing (→less combinatorial explosion) and / or applying heuristic search algorithms (e.g. A*)

Supertagging

- in CCG bank of CCG parses: over 1000 lexical categories: keep only those that occur more than 10 times → 425 supertags (lexical categories) (compared to 45 POS tags in the Penn Treebank)

- Supertagging (analogous to POS-tagging): sequence labeling problem: approach as before (cp. chapter 8): Maximum Entropy Markov Model (MEMM): train model with MLE; for argmax: Viterbi

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

e.g. use symmetric window of l=k=2

Supertagging

- However: even with additional features f_i : large number of supertags and high per-word ambiguity → too many errors by Viterbi

- Instead of most likely supertag-sequence (by Viterbi) we actually need a probability distribution over possible supertags 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		
...		

→ compute it with a modified version of forward backward algorithm (see chapter Appendix A) or with RNNs (see later)

RNNs for Supertagging

- Recurrent Neural Networks (RNNs) also do well for this task:
  - using vector-valued features for words (word-embeddings)
  - use entire sequence for assessing a word instead of sliding window
  - avoid high-level features (such as POS tags) because of error propagation

CCG Parsing using the A* Algorithm

- A*-algorithm: complete (finds a solution if ex.), optimal (finds best solution), optimally complex (no less complex alg. with same heuristic), informed (using heuristic) search algorithm
  - iteratively choose next state n with optimal estimated cost f(n)= g(n)+h(n), where g(n) actual current actual cost to n from start node and h(n) non-overestimating estimate to target state from n
  - each state: either in closed list (treated completely, g correct), or open list (known but not yet fully treated, g may shrink), or unknown

CCG Parsing using the A* Algorithm

```
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)
```

edges represent spans of symbols (==completed constituents)
$w_{i,j}$ (words from i to j)

agenda ←→ open-list

table: (current) parse table

chart: (current) upper triangle of n+1 x n+1 PCYK matrix

CCG Parsing using the A* Algorithm

how to compute heuristics f(n)?

- simplifying assumption: ignoring usage of other rules, we assume that probability of a derivation D of a sentence $S = (w_1, w_2, \dots)$ = product of probabilities of supertags $T = (t_1, t_2, \dots)$ for words chosen in D

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

taking negative log → convert probability to a cost:

$$\begin{aligned} P(D,S) &= P(T,S) \\ &= \sum_{i=1}^{n} -\log P(t_i|w_i) \end{aligned}$$

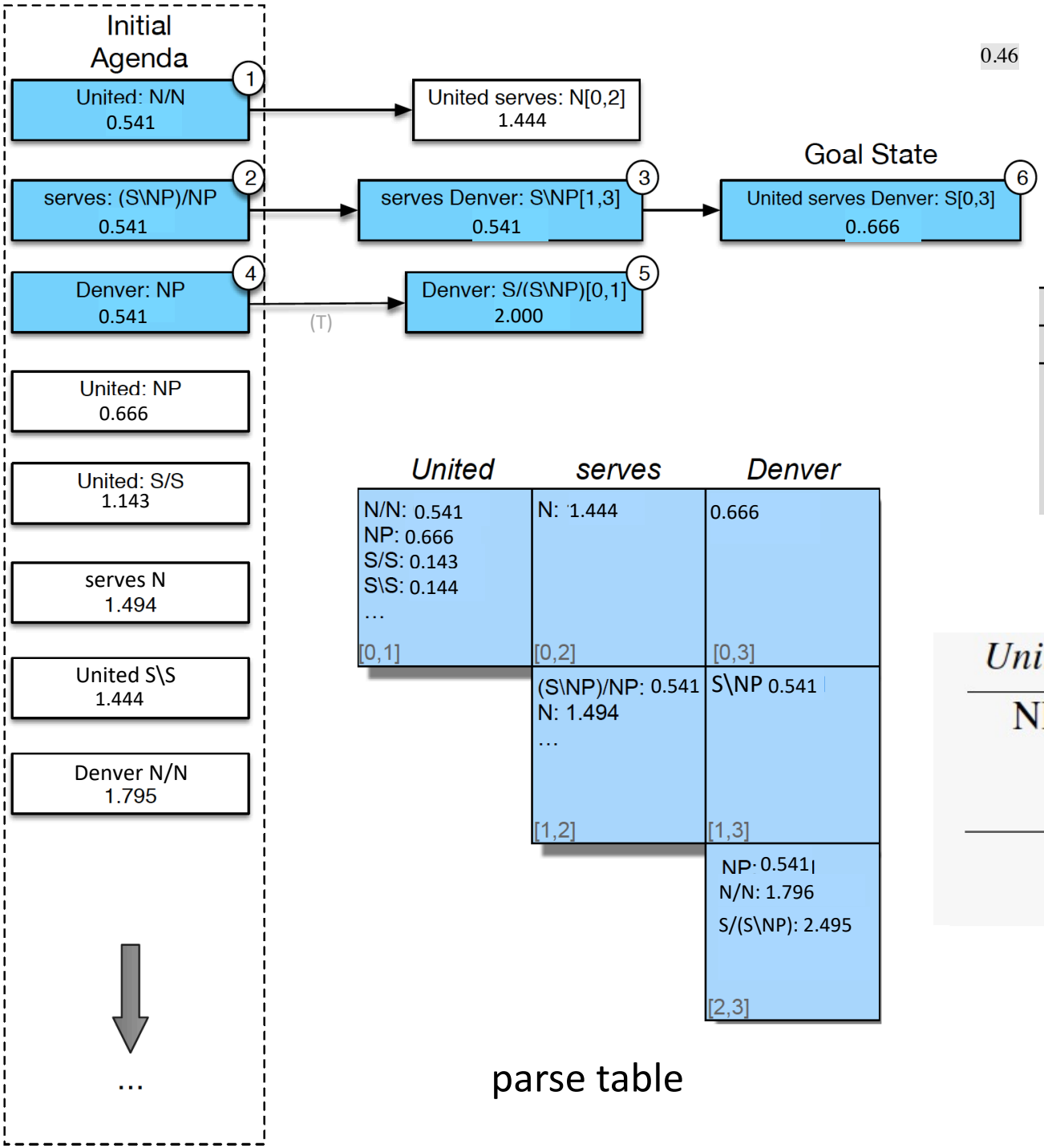
CCG Parsing using the A* Algorithm

- → $f(n) = g(n) + h(n)$ = inside cost + outside cost:
  - g(n) cost of current span (sub-sequence ("sub-string") of words currently considered)
  - h(n) estimated cost for words outside of span: assume that each of these outside words is assigned most probable supertag (→ never overestimate costs)
- →

$$f(w_{i,j}, t_{i,j}) = g(w_{i,j}) + h(w_{i,j})$$

$$= \sum_{k=i}^{j} -\log P(t_k|w_k) +$$

$$\sum_{k=1}^{i-1} \min_{t \in tags}(-\log P(t|w_k)) + \sum_{k=j+1}^{N} \min_{t \in tags}(-\log P(t|w_k))$$



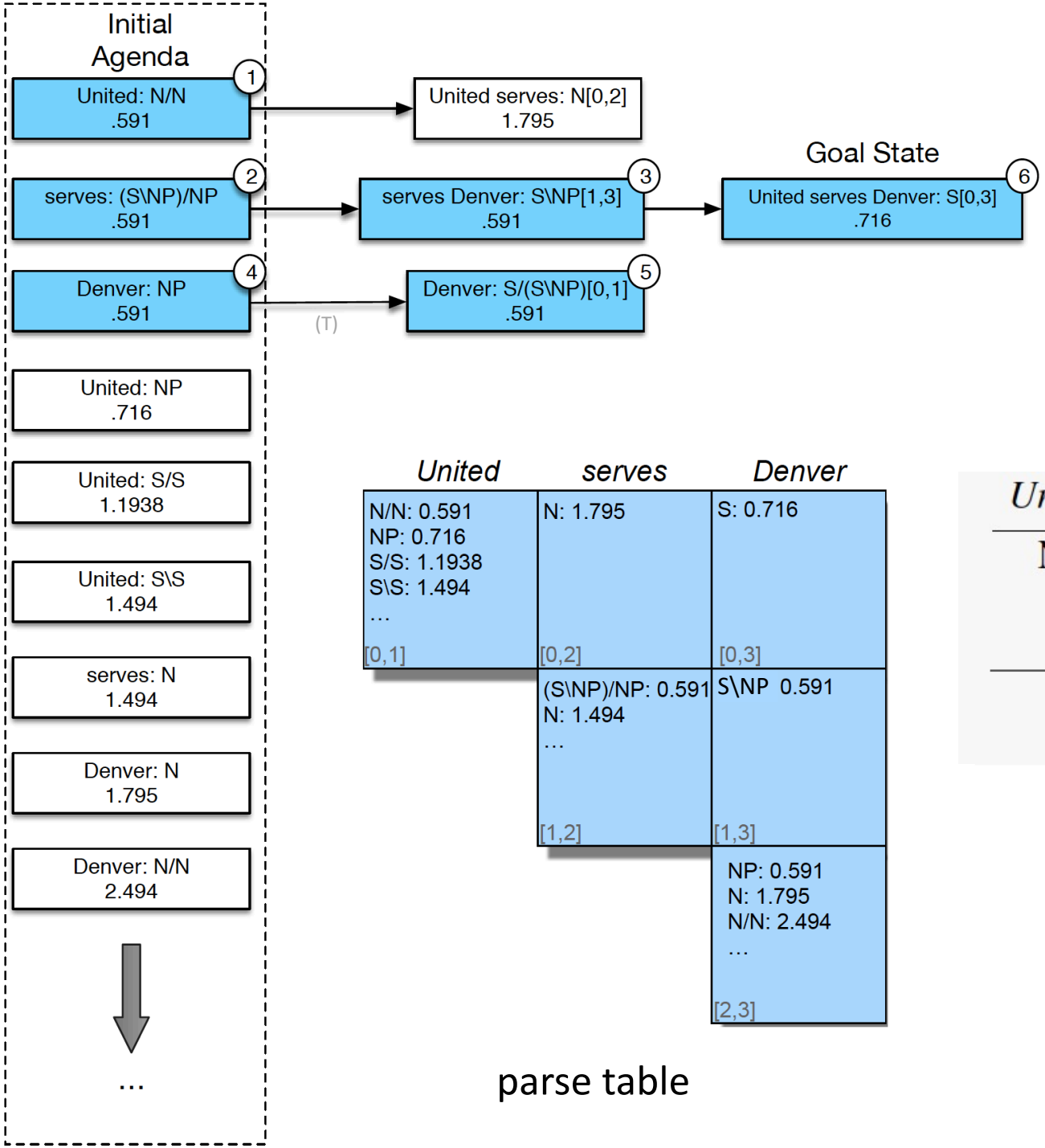
United	serves	Denver
N/N: 0.4	$(S\backslash NP)/NP$: 0.8	NP: 0.9
NP: 0.3	N: 0.1	N/N: 0.05
S/S: 0.1	...	...
$S\backslash S$: .05		S/(S\NP) 0.01
...		

$-\log_{10} P$		
United	serves	Denver
N/N: 0.398	(S\NP)/NP 0.097	NP: 0.046
NP: 0.523	N: 1.000	N/N: 1.301
S/S: 1.000		S/(S\NP) 2.000
S\S: 1.301		

United	*serves*	*Denver*
N/N: 0.541 NP: 0.666 S/S: 0.143 S\S: 0.144 … [0,1]	N: 1.444 [0,2]	0.666 [0,3]
	(S\NP)/NP: 0.541 N: 1.494 … [1,2]	S\NP 0.541 [1,3]
		NP 0.541 N/N: 1.796 S/(S\NP): 2.495 [2,3]

parse table

United	*serves*	*Denver*
NP	(S\NP)/NP	NP
	S\NP (>)	
S (<)		



United	*serves*	*Denver*
N/N: 0.591 NP: 0.716 S/S: 1.1938 S\S: 1.494 … [0,1]	N: 1.795 [0,2]	S: 0.716 [0,3]
	(S\NP)/NP: 0.591 N: 1.494 … [1,2]	S\NP 0.591 [1,3]
		NP: 0.591 N: 1.795 N/N: 2.494 … [2,3]

parse table

$$\dfrac{\dfrac{\textit{United}}{\text{NP}} \quad \dfrac{\dfrac{\textit{serves}}{(\text{S}\backslash\text{NP})/\text{NP}} \quad \dfrac{\textit{Denver}}{\text{NP}}}{\text{S}\backslash\text{NP}}\,>}{\text{S}}\,<$$

This is the original version from the book.
The actual costs are not in 100 % correspondence to the table on slide 30 (assuming $\log_{10}$ as stated in caption of figure 14.12)

Performance Evaluation of Classifiers

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

$$\text{or} \quad \frac{\sum_s \#\text{ of correct constituents in hypothesis parse of } s}{\sum_s \#\text{ of correct constituents in reference parse of } s}$$

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

$$\text{or} \quad \frac{\sum_s \#\text{ of correct constituents in hypothesis parse of } s}{\sum_s \#\text{ of total constituents in hypothesis parse of } s}$$

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

cross bracketing errors: e.g. predicted: (A(BC)) but correct ((AB)C)

- modern parsers: F1 > 0.9 & < 0.01 cross bracketing errors
- For some applications (and e.g. for lexical parsers such as CCG): more interested in extracting appropriate predicate-argument relations or grammatical dependencies, rather than a specific derivation → better metric of parser usefulness for further semantic processing → use alternative evaluation metrics

Bibliography

(1) Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft, version Oct 2019); Online: https://web.stanford.edu/~jurafsky/slp3/ (URL, Oct 2019); this slide-set is especially based on chapter 14.

(2) S. Russell, P. Norvig: Artificial Intelligence – A Modern Approach, 3rd edition, Pearson 2010, section 3.5

Recommendations for Studying

- minimal approach:

  work with the slides and understand their contents! Think beyond instead of merely memorizing the contents

- standard approach:

  minimal approach + read the corresponding pages in Jurafsky [1]

- interested students

  == standard approach