Roadmap

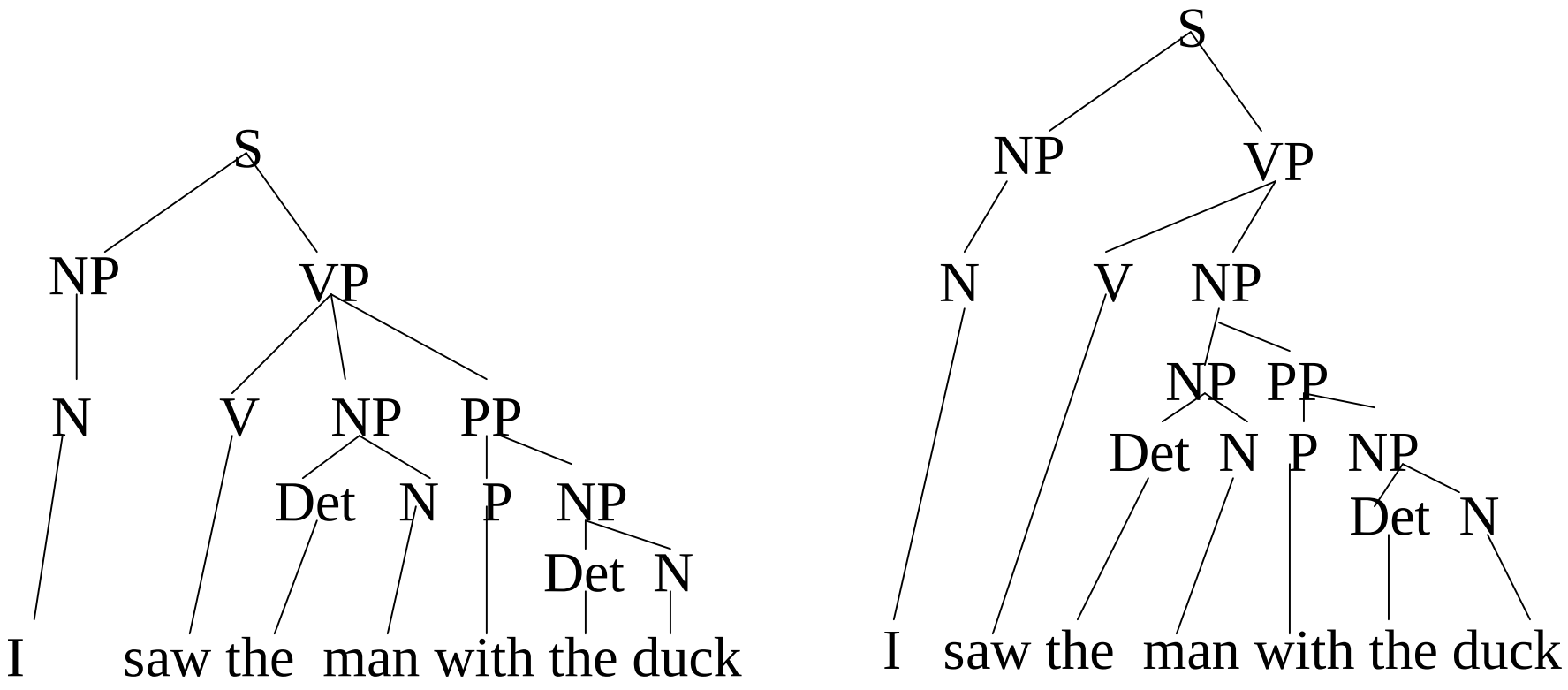
- Probabilistic CFGs
  - Handling ambiguity – more likely analyses
  - Adding probabilities
    - Grammar
    - Parsing: probabilistic CYK
    - Learning probabilities: Treebanks & Inside-Outside
    - Issues with probabilities
  - Resolving issues
    - Lexicalized grammars
      - Independence assumptions
  - Alternative grammar formalisms
    - Dependency Grammar

Representation: Probabilistic Context-free Grammars

- PCFGs: 5-tuple
  - A set of terminal symbols: Σ
  - A set of non-terminal symbols: N
  - A set of productions P: of the form A -> α
    - Where A is a non-terminal and α in (Σ U N)*
  - A designated start symbol S
  - A function assigning probabilities to rules: D
- L = W|w in Σ* and S=>*w
  - Where S=>*w means S derives w by some seq

Parse Ambiguity

- Two parse trees



Small Probabilistic Grammar

PP $\overset{1.0}{\rightarrow}$ P NP

NP -> N
NP $\overset{0.65}{\rightarrow}$ Det N
NP $\overset{0.10}{\rightarrow}$ Det Adj N
NP $\overset{0.05}{\rightarrow}$ NP PP

VP $\overset{0.45}{\rightarrow}$ V
VP $\overset{0.45}{\rightarrow}$ V NP
VP $\overset{0.10}{\rightarrow}$ V NP PP

S $\overset{0.85}{\rightarrow}$ NP VP
S $\overset{0.15}{\rightarrow}$ S conj S

Parse Probabilities

$$P(T,S) = \prod_{n \in T} p(r(n))$$

- T(ree),S(entence),n(ode),R(ule)
- T1 = 0.85*0.2*0.1*0.65*1*0.65 = 0.007
- T2 = 0.85*0.2*0.45*0.05*0.65*1*0.65 = 0.003

- Select T1
- Best systems achieve 92-93% accuracy

Probabilistic CYK Parsing

- Augmentation of Cocke-Younger-Kasami
  - Bottom-up parsing
- Inputs
  - PCFG in CNF G={N,Σ,P,S,D}, N have indices
  - N words w1…wn
- DS:Dynamic programming array: π[i,j,a]
    - Holding max prob index a spanning i,j
- Output: Parse π[1,n,1] with S and w1..wn

Probabilistic CYK Parsing

- Base case: Input strings of length 1
  - In CNF, prob must be from A=>wi
- Recursive case: For strings > 1, A=>*wij iff there is rule A->BC and some k, 1<=k<j st B derives the first k symbols and C the last j-k. Since len < |wij|, probability in table. Multiply subparts; compute max over all subparts.

Inside-Outside Algorithm

- EM approach
  - Similar to Forward-Backward training of HMM
- Estimate number of times production used
  - Base on sentence parses
  - Issue: Ambiguity
    - Distribute across rule possibilities
  - Iterate to convergence

Issues with PCFGs

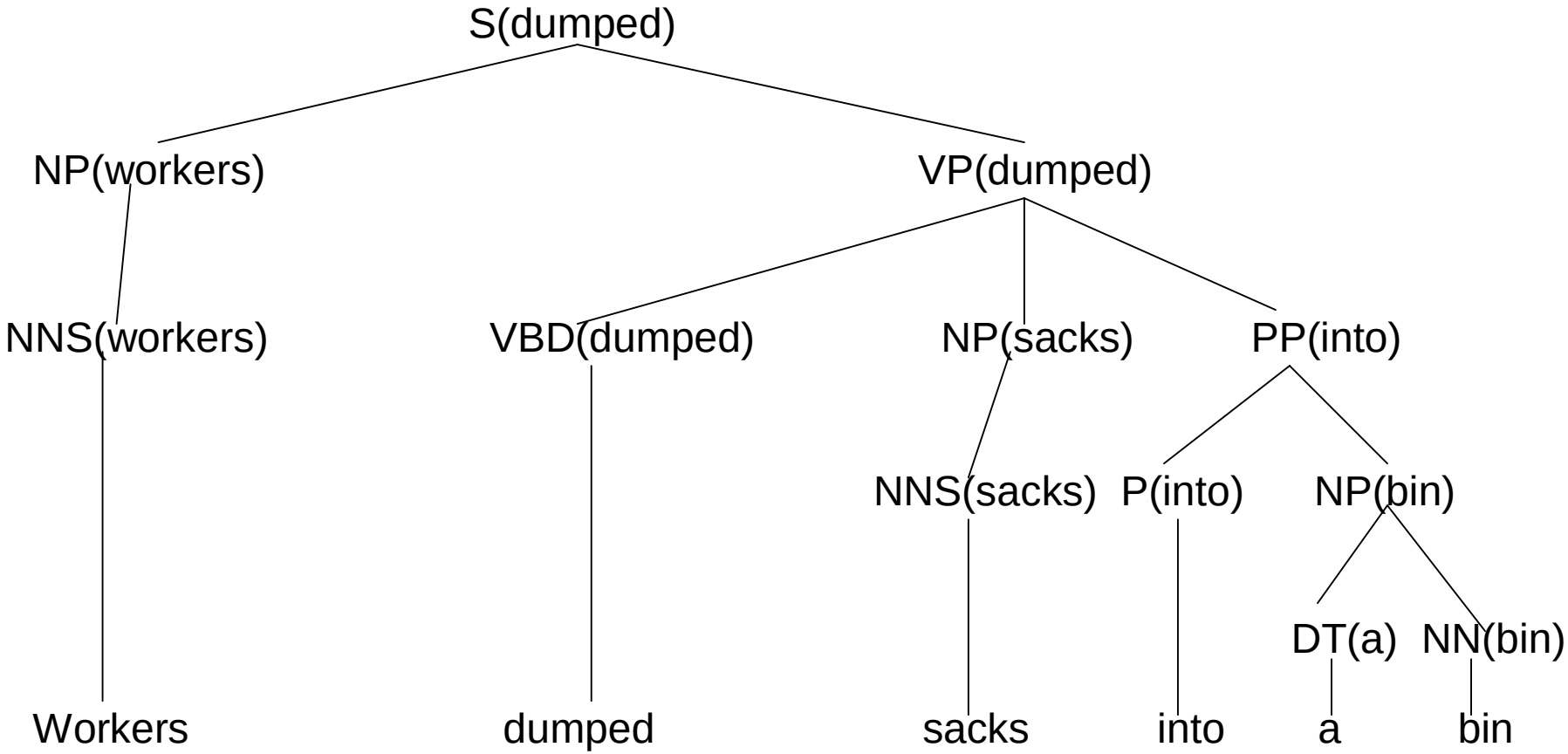
- Non-local dependencies
  - Rules are context-free; language isn't
- Example:
  - Subject vs non-subject NPs
    - Subject: 90% pronouns (SWB)
    - NP-> Pron vs NP-> Det Nom: doesn't know if subj
- Lexical context:
  - Verb subcategorization:
    - Send NP PP vs Saw NP PP
  - One approach: lexicalization

Probabilistic Lexicalized CFGs

- Key notion: “head”
  - Each non-terminal assoc w/lexical head
    - E.g. verb with verb phrase, noun with noun phrase
  - Each rule must identify RHS element as head
    - Heads propagate up tree
  - Conceptually like adding 1 rule per head value

    - VP(dumped) -> VBD(dumped)NP(sacks)PP(into)
    - VP(dumped) -> VBD(dumped)NP(cats)PP(into)

PLCFG with head features

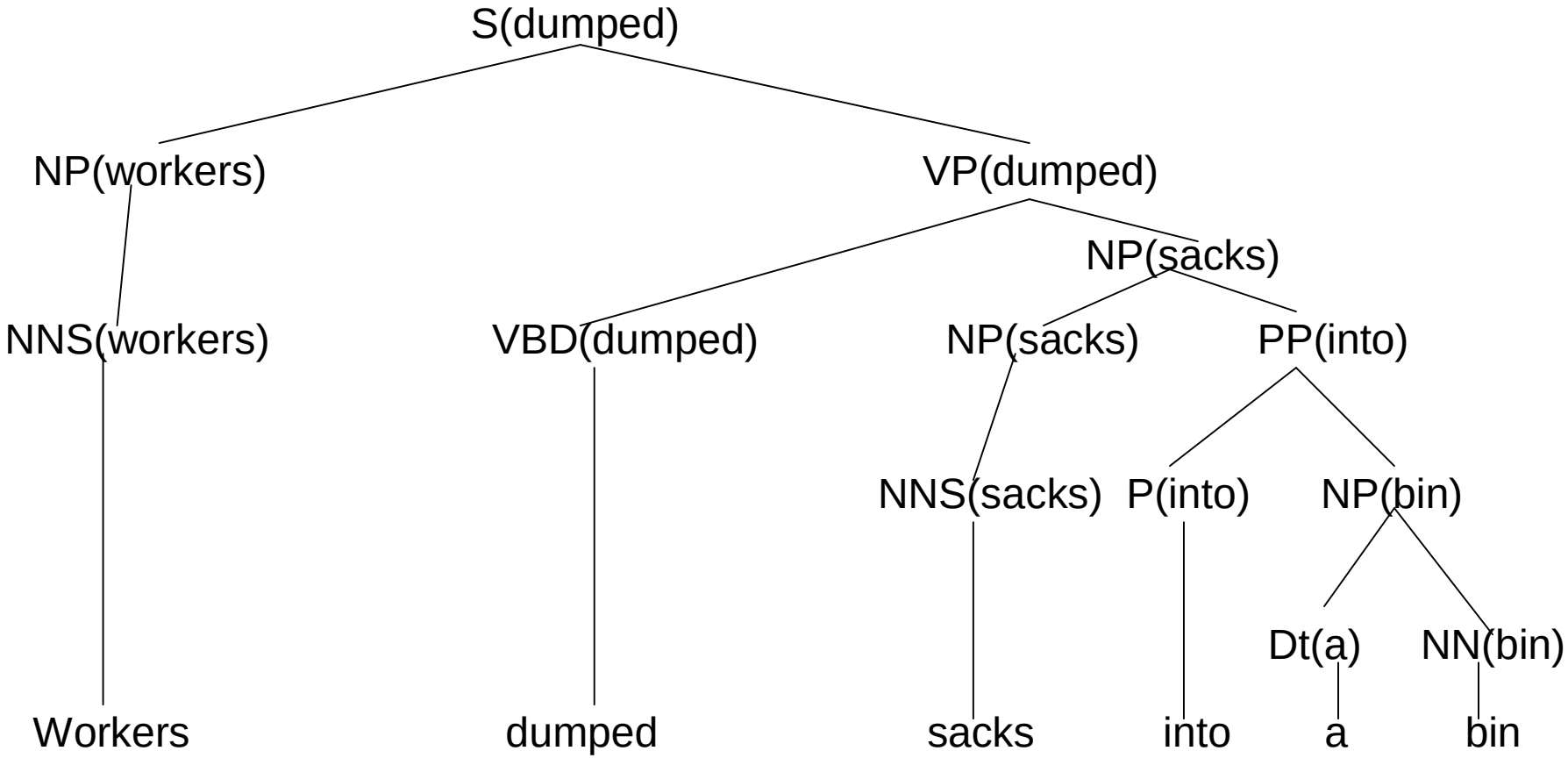


PLCFGs

- Issue: Too many rules
  - No way to find corpus with enough examples
- (Partial) Solution: Independence assumed
  - Condition rule on
    - Category of LHS, head
  - Condition head on
    - Category of LHS and parent's head

$$P(T,S) = \prod_{n \in T} p(r(n) \mid n.h(n)) * p(h(n) \mid n, h(m(n)))$$

Disambiguation Example



Disambiguation Example II

$$P(VP \rightarrow VBDNPPP \mid VP, dumped)$$
$$= \frac{C(VP(dumped) \rightarrow VBDNPP)}{\sum_{\beta} C(VP(dumped) \rightarrow \beta)}$$
$$= 6/9 = 0.67$$

$$p(VP \rightarrow VBDNP \mid VP, dumped)$$
$$= \frac{C(VP(dumped) \rightarrow VBDNP)}{\sum_{\beta} C(VP(dumped) \rightarrow \beta)}$$
$$= 0/9 = 0$$

$$p(in \mid PP, dumped)$$
$$= \frac{C(X(dumped) \rightarrow ...PP(in)..)}{\sum_{\beta} C(X(dumped) \rightarrow ...PP...)}$$
$$= 2/9 = 0.22$$

$$p(in \mid PP, sacks)$$
$$= \frac{C(X(sacks) \rightarrow ...PP(in)...)}{\sum_{\beta} C(X(sacks) \rightarrow ...PP...)}$$
$$= 0/0$$

Evaluation

- Compare to “Gold Standard” manual parse
  - Correct if constituent has same start, end, label
- Three measures:
  - Labeled Recall:
    - # correct constituents in candidate parse of s

      # correct constituents in treebank parse of c
  - Labeled Precision:
    - # correct constituents in candidate parse of s

      # total constituents in candidate parse of c

  - Cross-brackets: (A (B C)) vs ((A B) C)
- Standard: 90%,90%,1%

Dependency Grammars

- Pure lexical dependency
  - Relate words based on binary syntactic relns
  - No constituency, phrase structure rules
- Why?
  - Better for languages with flexible word orders
  - Abstracts away from surface form/order
- Root node + word nodes
  - Link with dependency relations – fixed set
    - E.g. subj, obj, dat, loc, attr, mode, comp,…

Dependency Grammar Example