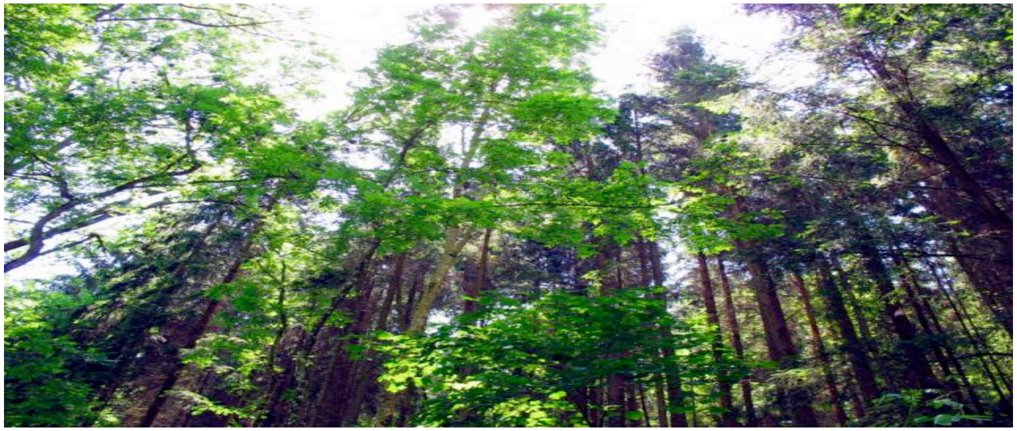
Forest Reranking

Discriminative Parsing with Non-Local Features



Liang Huang

University of Pennsylvania

ACL 2008 talk, Columbus, OH, June 2008

Is Supervised Parsing Done?

is it a done area?

Bod (2007)

Is the End of Supervised Parsing in Sight?

Is Supervised Parsing Done?

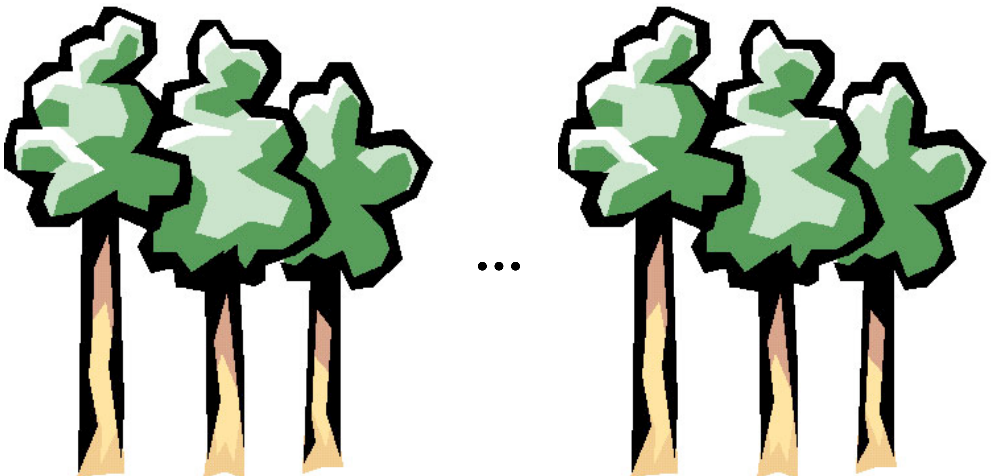
is it a done area?

Bod (2007)

Is the End of Supervised Parsing in Sight?

- motivation: use non-local features
  - linguistically-motivated features for *n*-best reranking (Charniak and Johnson, 2005; Collins, 2000)
  - but can we integrate them back into chart parsing?
  - YES: using a packed forest!
- result: best whole Treebank parsing accuracy to date

Why is *n*-best list a bad idea?



- too few variations (limited scope)
  - 41% correct parses are not in ~30-best (Collins, 2000)
  - worse for longer sentences; tiny fraction of whole space
- too many redundancies
  - 50-best usually encodes 5-6 binary decisions ($2^5 < 50 < 2^6$)

Why is *n*-best list a bad idea?

packed forest

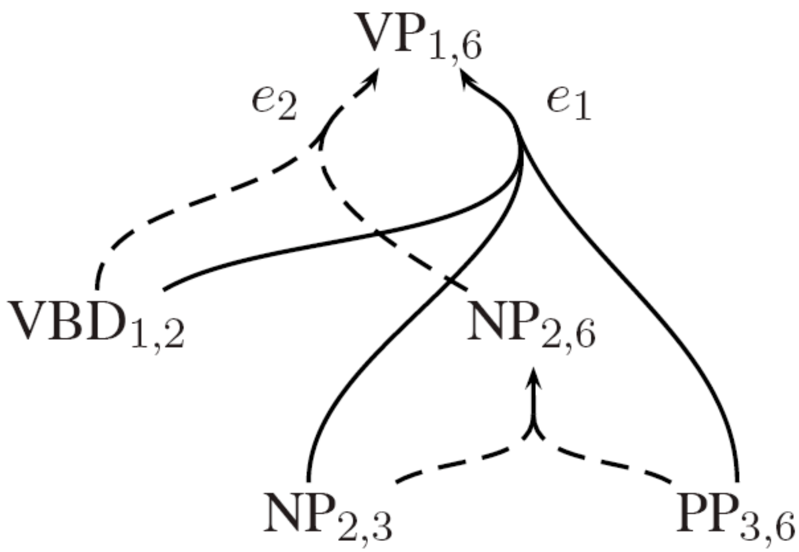
- too few variations (limited scope)
  - 41% correct parses are not in ~30-best (Collins, 2000)
  - worse for longer sentences; tiny fraction of whole space
- too many redundancies
  - 50-best usually encodes 5-6 binary decisions ($2^5<50<2^6$)

Outline

- Packed Forest and General Idea
- Forest Reranking and Non-Local Features
    - Perceptron for Generic Reranking
    - Local vs. Non-Local Features
    - Incremental Computation of Non-Local Features
- Decoding Algorithm
- Experiments

Packed Forest

- a compact representation of many parses
  - by sharing common sub-derivations
  - polynomial-space encoding of exponentially large set

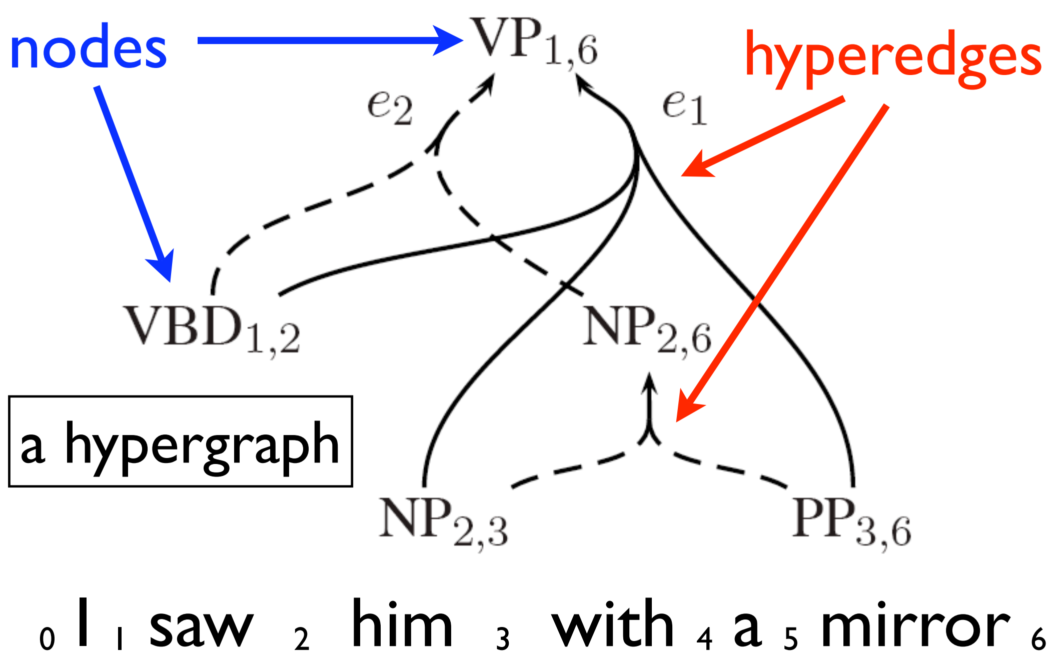


$_0$ I $_1$ saw $_2$ him $_3$ with $_4$ a $_5$ mirror $_6$

(Klein and Manning, 2001; Huang and Chiang, 2005)

Packed Forest

- a compact representation of many parses
  - by sharing common sub-derivations
  - polynomial-space encoding of exponentially large set

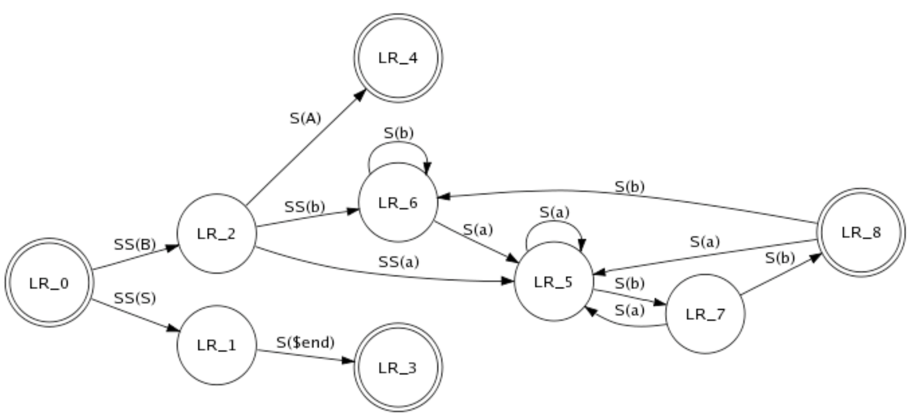


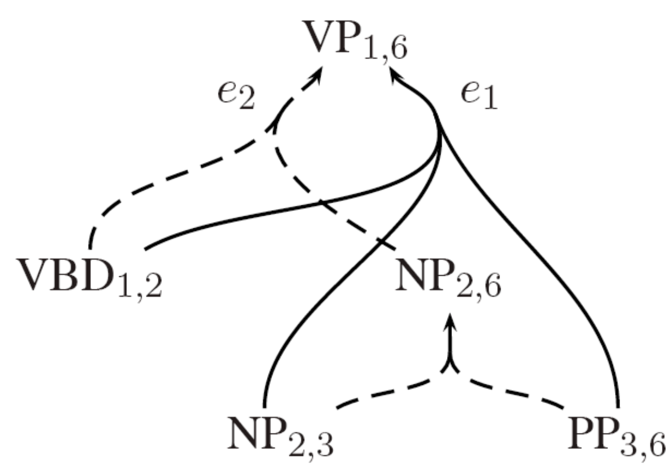
$$e_1 \quad \frac{VBD_{1,2} \quad NP_{2,3} \quad PP_{3,6}}{VP_{1,6}}$$

(Klein and Manning, 2001; Huang and Chiang, 2005)

Lattices vs. Forests

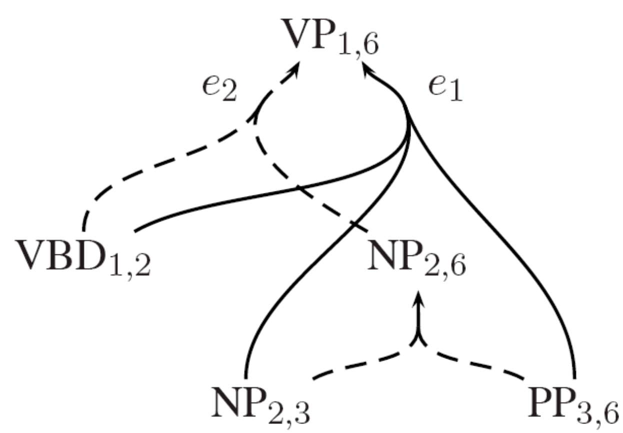
- forest generalizes "lattice" from finite-state world
  - both are compact encodings of exponentially many derivations (paths or trees)
  - graph => hypergraph; regular grammar => CFG





Reranking on a Forest?

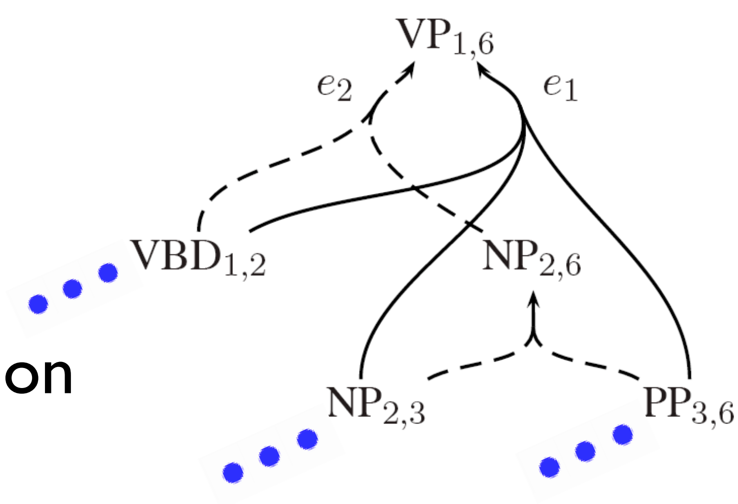
- with only local features
  - dynamic programming, tractable
    (Taskar et al. 2004; McDonald et al., 2005)
- with non-local features
  - intractable, so we do approximation
  - on-the-fly reranking at internal
  - use non-locals as early and as much as possible!



methods \ features	*local*	*non-local*
n-best reranking	only at the root node	
DP-based discrim. parsing	exact	N/A
forest reranking	exact	on-the-fly

Reranking on a Forest?

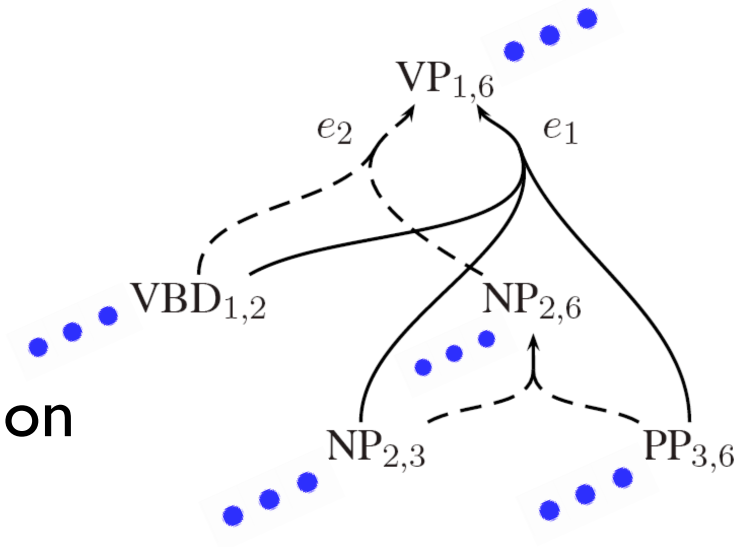
- with only local features
  - dynamic programming, tractable
    (Taskar et al. 2004; McDonald et al., 2005)
- with non-local features
  - intractable, so we do approximation
  - on-the-fly reranking at internal
  - use non-locals as early and as much as possible!



methods \ features	*local*	*non-local*
n-best reranking	only at the root node	
DP-based discrim. parsing	exact	N/A
forest reranking	exact	on-the-fly

Reranking on a Forest?

- with only local features
  - dynamic programming, tractable
    (Taskar et al. 2004; McDonald et al., 2005)
- with non-local features
  - intractable, so we do approximation
  - on-the-fly reranking at internal
  - use non-locals as early and as much as possible!



methods \ features	*local*	*non-local*
n-best reranking	only at the root node	
DP-based discrim. parsing	exact	N/A
forest reranking	exact	on-the-fly

Penn
University *of* Pennsylvania

Outline

- Packed Forest and General Idea
- Forest Reranking and Non-Local Features
  - Perceptron for Generic Reranking
  - Local vs. Non-Local Features
  - Incremental Computation of Non-Local Features
- Decoding Algorithm
- Experiments

Generic Reranking by Perceptron

- for each sentence s_i, we have a set of candidates $cand(s_i)$
  - and an oracle tree y_i^+, among the candidates
- a feature mapping from tree y to vector $\mathbf{f}(y)$

Input: Training examples $\{\boxed{cand(s_i)}, \boxed{y_i^+}\}_{i=1}^N$

$\mathbf{w} \leftarrow \mathbf{0}$ $\triangleright$ initial weights

for $t \leftarrow 1 \ldots T$ **do** $\triangleright$ T iterations

  for $i \leftarrow 1 \ldots N$ **do**

    $\hat{y} = \mathrm{argmax}_{y \in cand(s_i)}\ \mathbf{w} \cdot \mathbf{f}(y)$

    if $\hat{y} \neq y_i^+$ **then**

      $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(y_i^+) - \mathbf{f}(\hat{y})$

return $\mathbf{w}$

(Collins, 2002)

Generic Reranking by Perceptron

- for each sentence s_i, we have a set of candidates $cand(s_i)$
  - and an oracle tree y_i^+, among the candidates
- a feature mapping from tree y to vector $\mathbf{f}(y)$

1: **Input**: Training examples $\{cand(s_i), y_i^+\}_{i=1}^N$

2: $\mathbf{w} \leftarrow \mathbf{0}$ ▷ initial weights

3: **for** $t \leftarrow 1 \dots T$ **do** ▷ T iterations

4: **for** $i \leftarrow 1 \dots N$ **do**

5: $\hat{y} = \operatorname{argmax}_{y \in cand(s_i)} \mathbf{w} \cdot \mathbf{f}(y)$

6: **if** $\hat{y} \neq y_i^+$ **then**

7: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(y_i^+) - \mathbf{f}(\hat{y})$

8: **return** $\mathbf{w}$

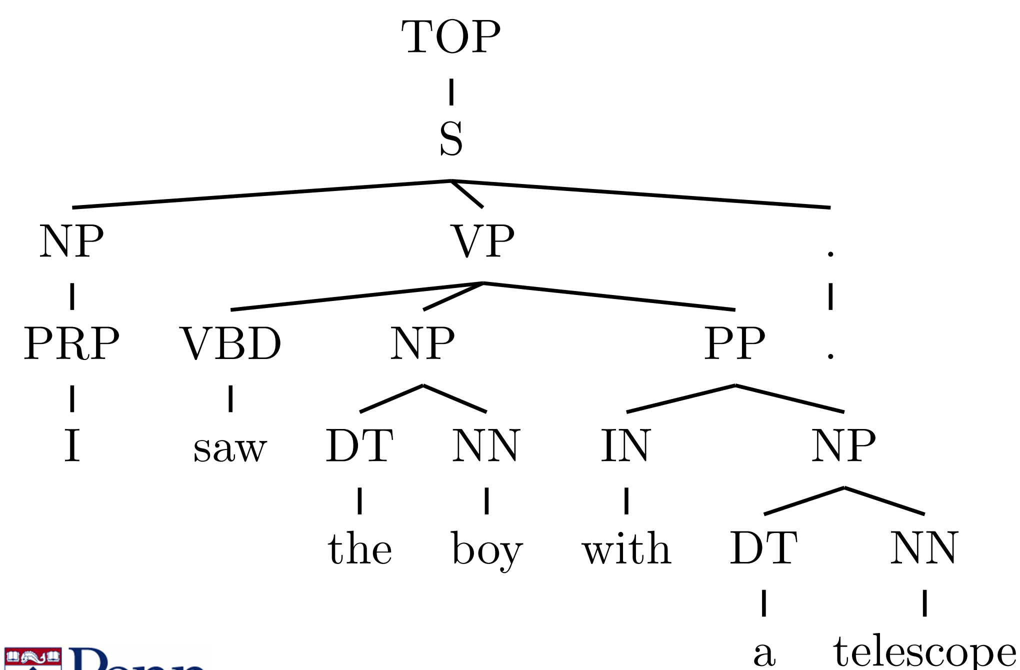
"decoder"

feature representation

(Collins, 2002)

Features

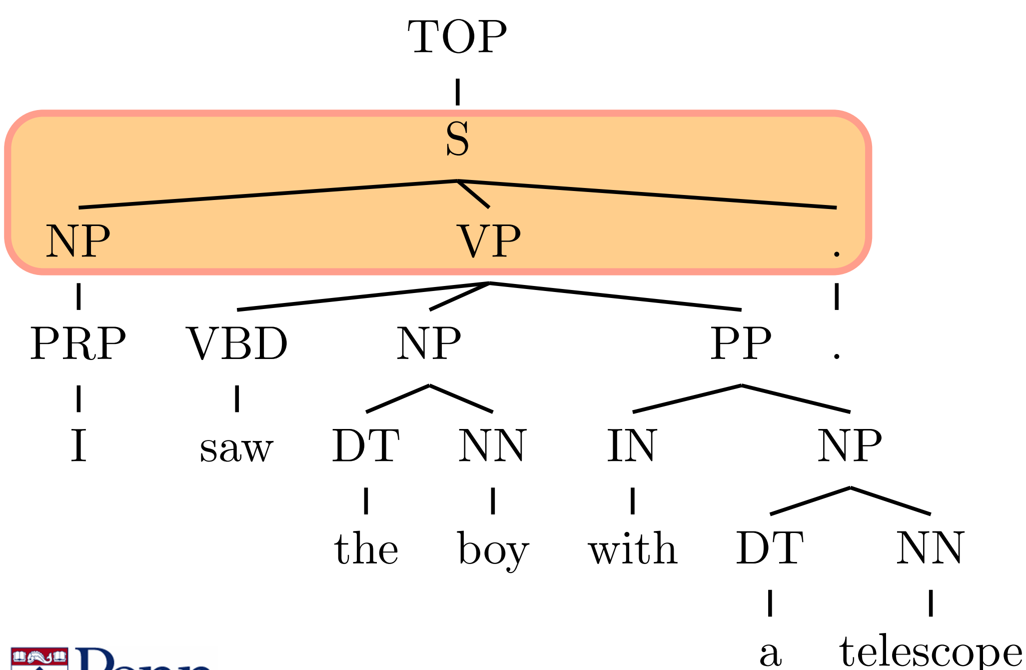
- a feature f is a function from tree y to a real number
  - $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser
  - every other feature *counts* the number of times a particular configuration occurs in y



our features are from
(Charniak & Johnson, 2005)
(Collins, 2000)

Features

- a feature f is a function from tree y to a real number
  - $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser
  - every other feature *counts* the number of times a particular configuration occurs in y



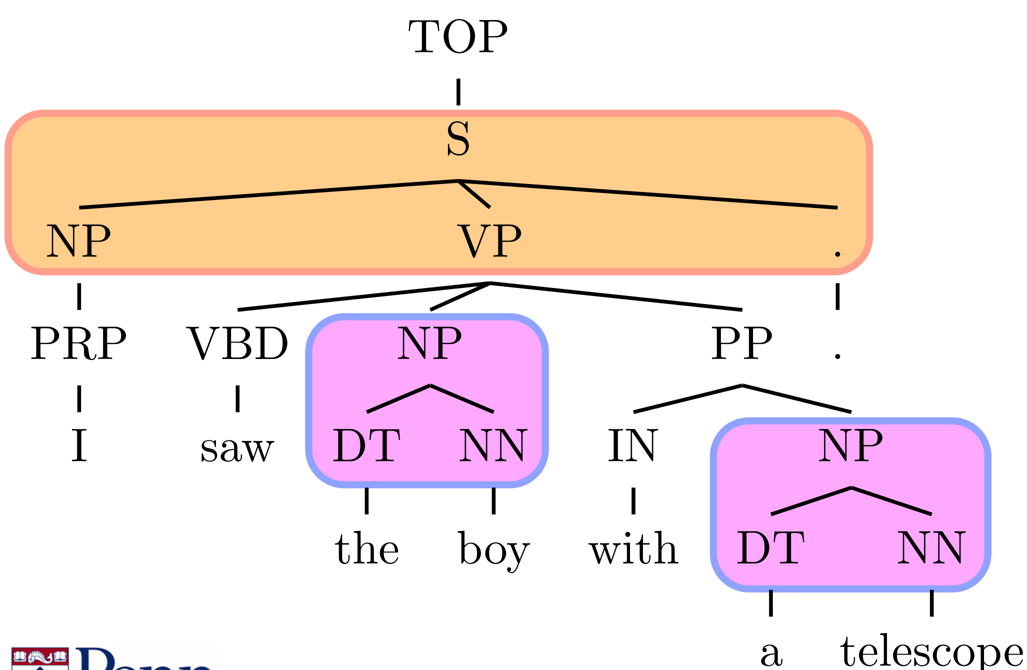
our features are from
(Charniak & Johnson, 2005)
(Collins, 2000)

instances of Rule feature

$$f_{100}(y) = f_{S \rightarrow NP\ VP\ .}(y) = 1$$
$$f_{200}(y) = f_{NP \rightarrow DT\ NN}(y) = 2$$

Features

- a feature f is a function from tree y to a real number
  - $f_1(y) = \log \Pr(y)$ is the log Prob from generative parser
  - every other feature *counts* the number of times a particular configuration occurs in y



our features are from
(Charniak & Johnson, 2005)
(Collins, 2000)

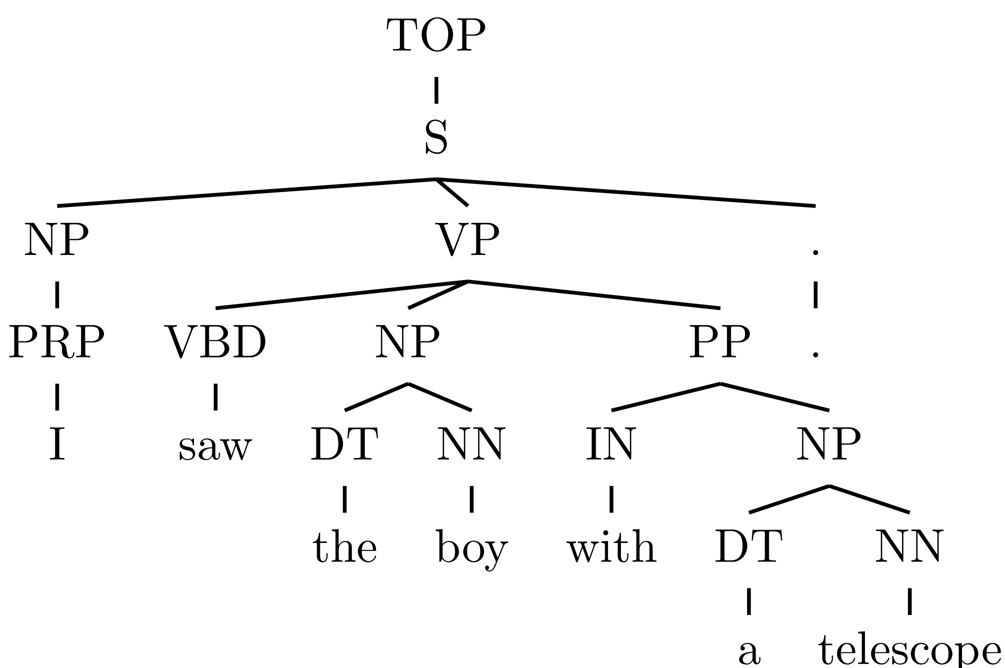
instances of Rule feature

$$f_{100}(y) = f_{S \to NP\ VP\ .}(y) = 1$$

$$f_{200}(y) = f_{NP \to DT\ NN}(y) = 2$$

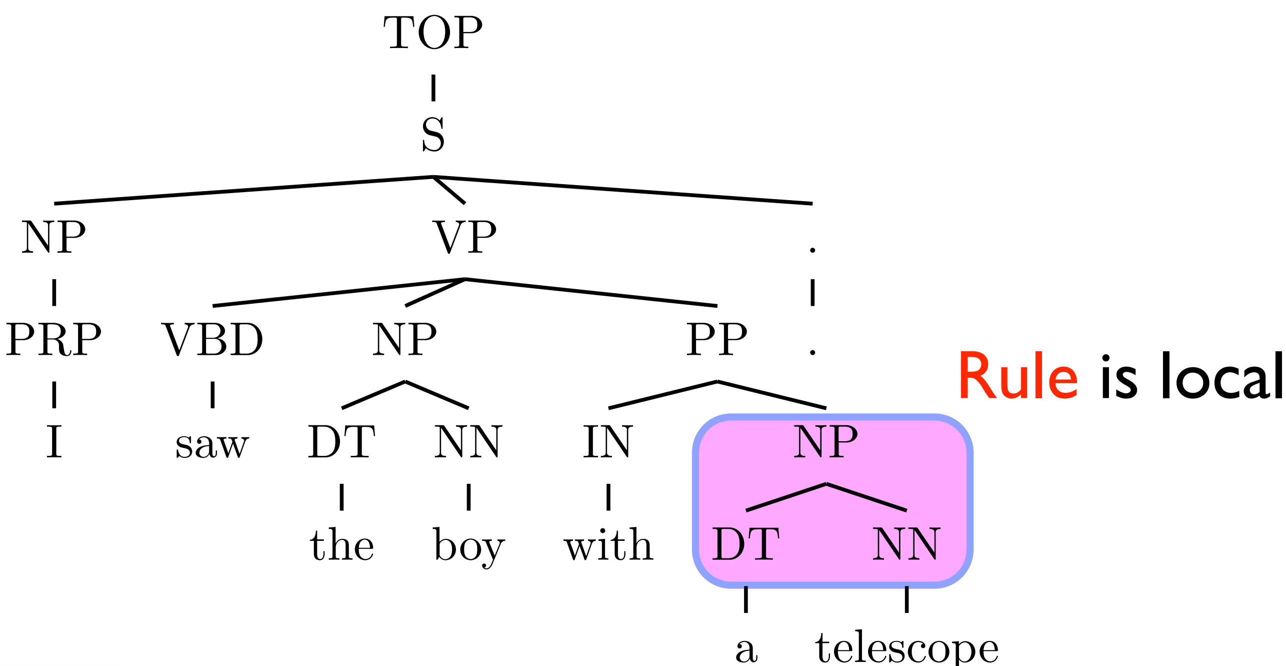
Local vs. Non-Local Features

- a feature is local iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest)
- local features can be pre-computed on each hyperedge in the forest; non-locals can not



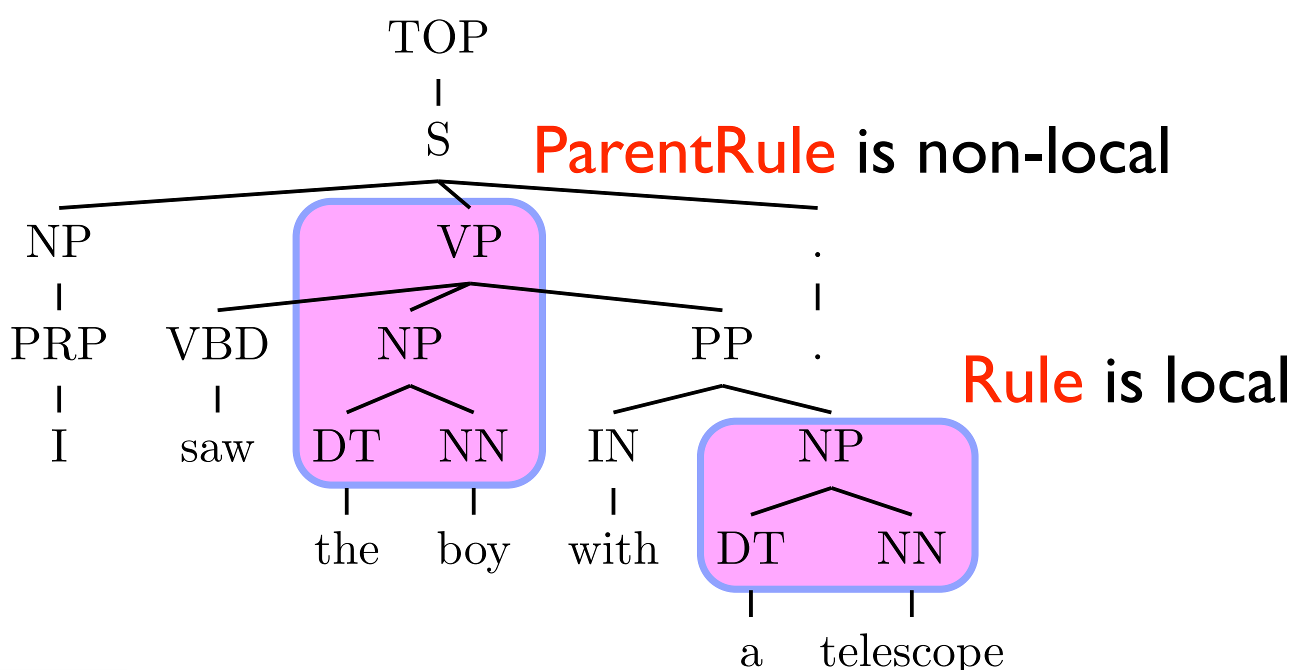
Local vs. Non-Local Features

- a feature is local iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest)
- local features can be pre-computed on each hyperedge in the forest; non-locals can not



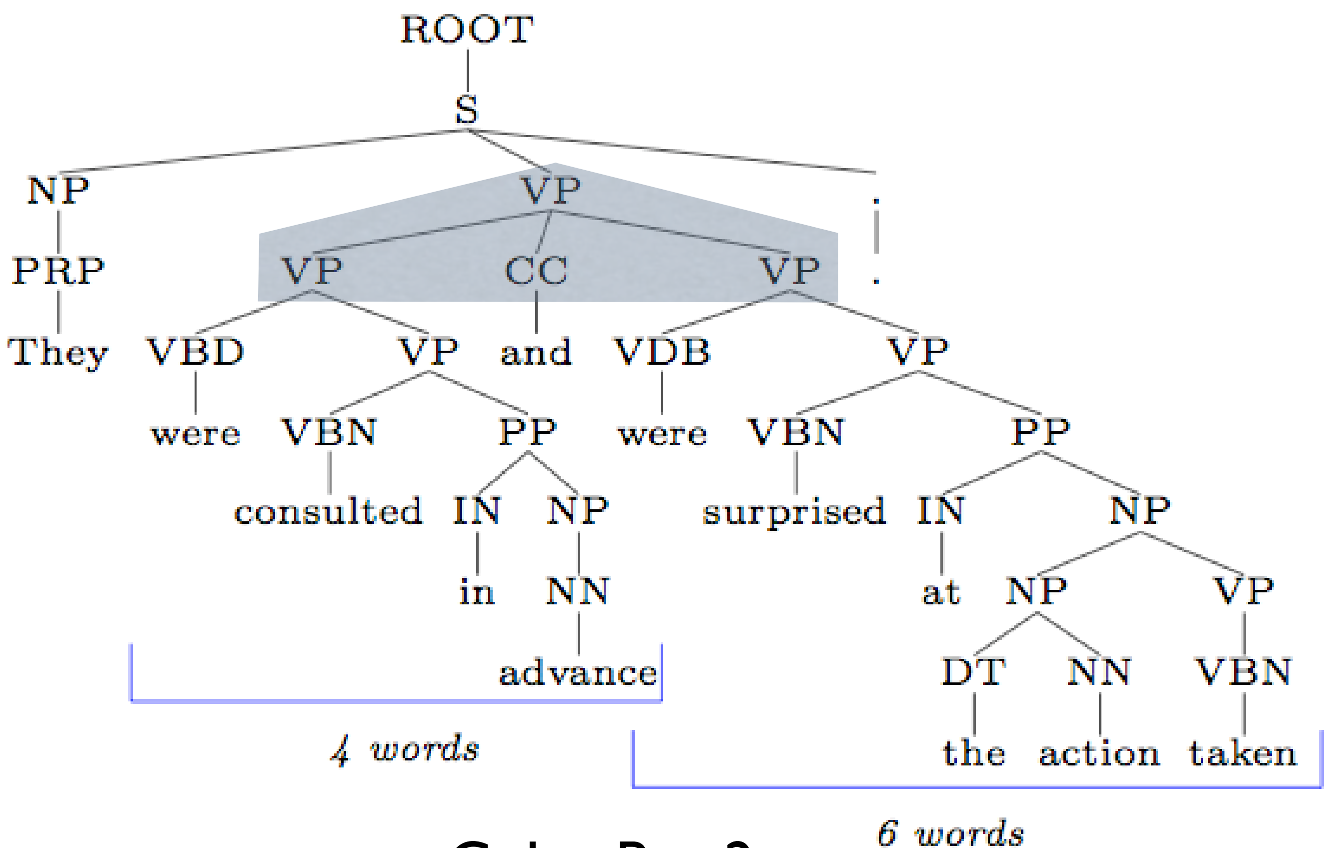
Local vs. Non-Local Features

- a feature is local iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest)
- local features can be pre-computed on each hyperedge in the forest; non-locals can not



Local vs. Non-Local: Examples

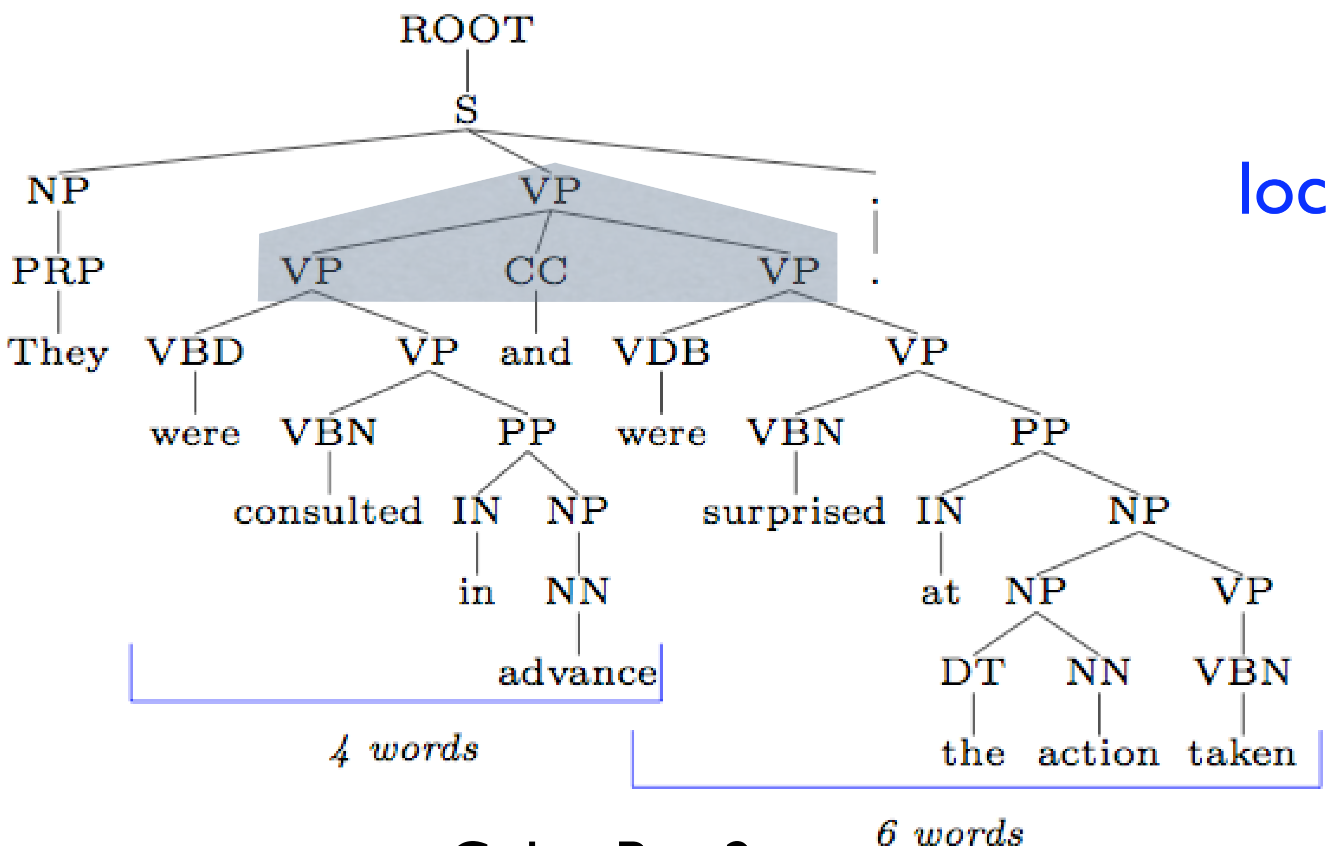
- CoLenPar feature captures the difference in lengths of adjacent conjuncts (Charniak and Johnson, 2005)



CoLenPar: 2

Local vs. Non-Local: Examples

- CoLenPar feature captures the difference in lengths of adjacent conjuncts (Charniak and Johnson, 2005)

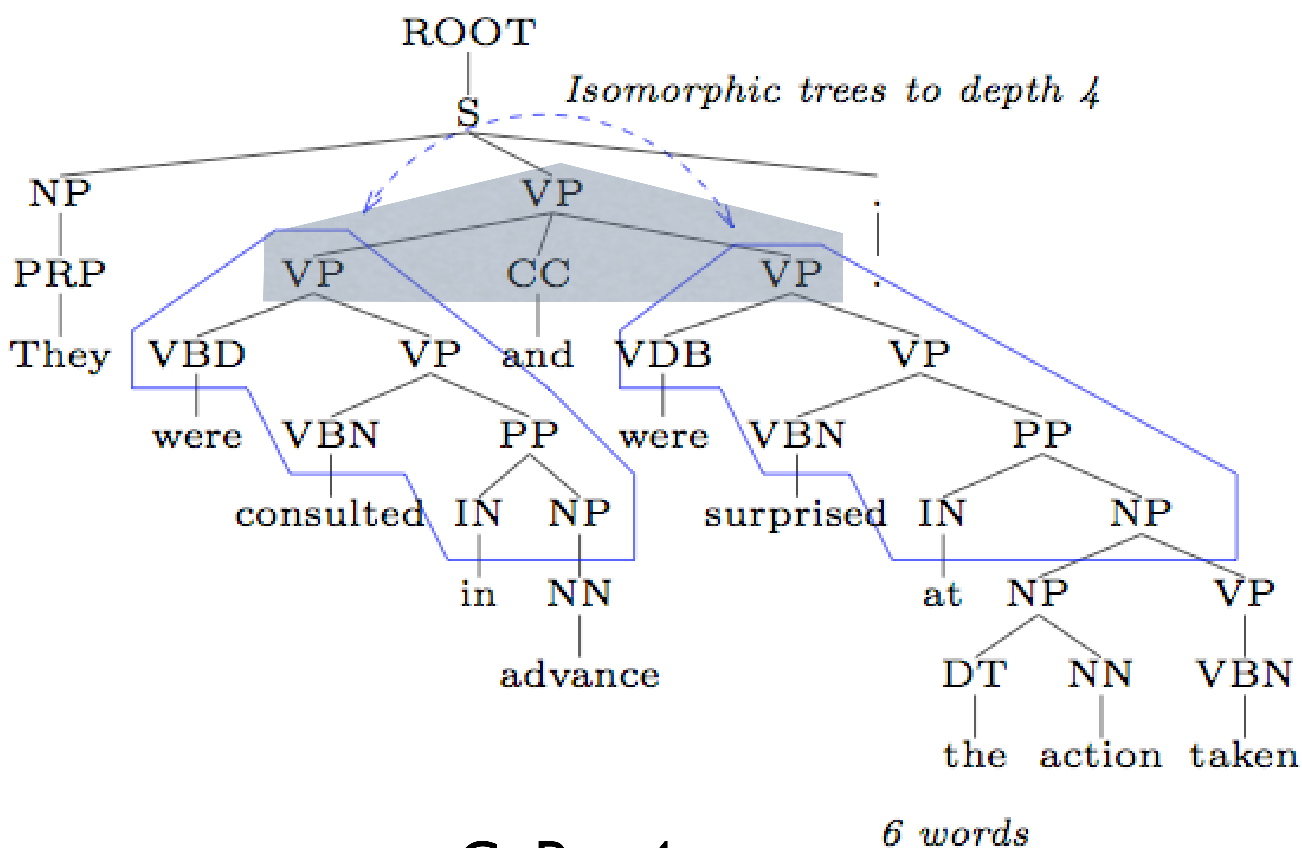


local!

CoLenPar: 2

Local vs. Non-Local: Examples

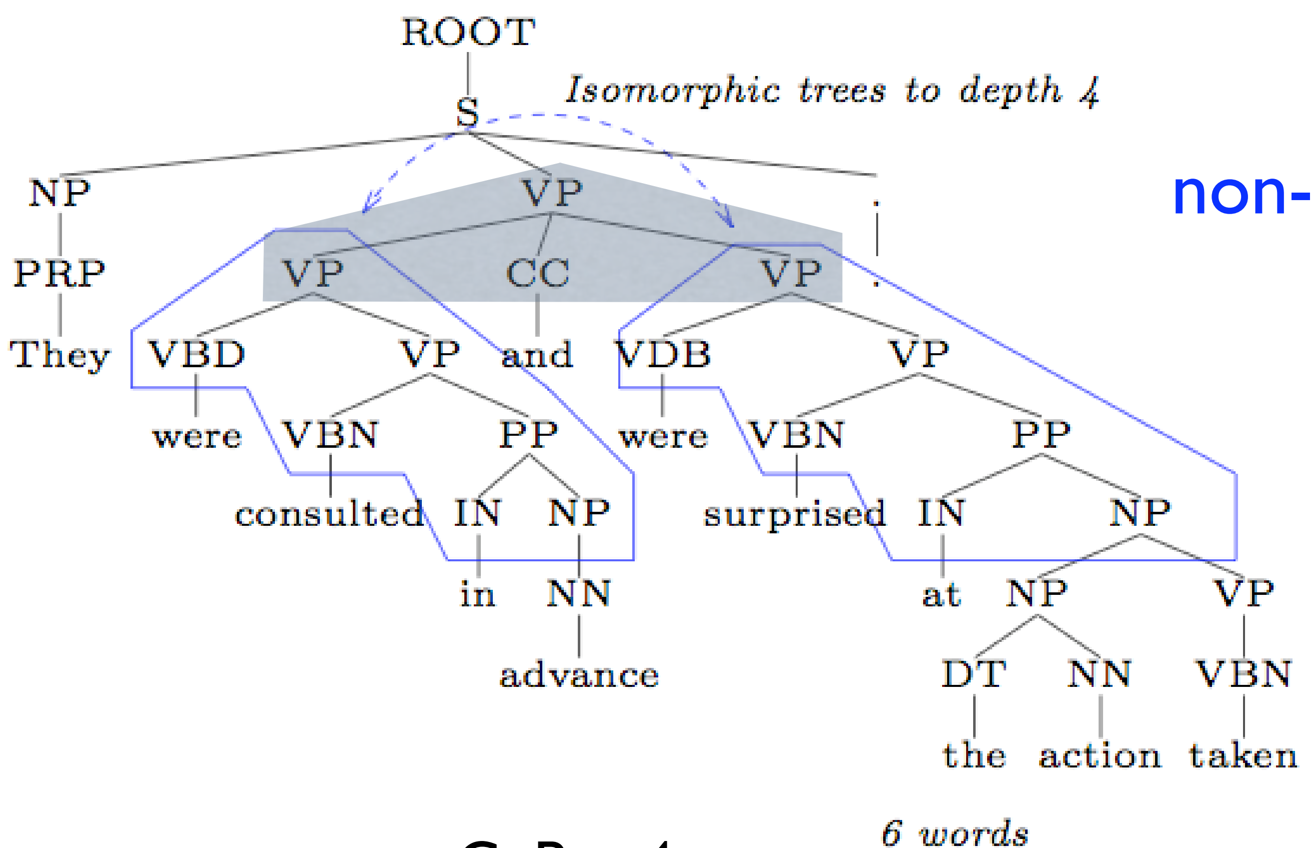
- CoPar feature captures the depth to which adjacent conjuncts are isomorphic (Charniak and Johnson, 2005)



CoPar: 4

Local vs. Non-Local: Examples

- CoPar feature captures the depth to which adjacent conjuncts are isomorphic (Charniak and Johnson, 2005)



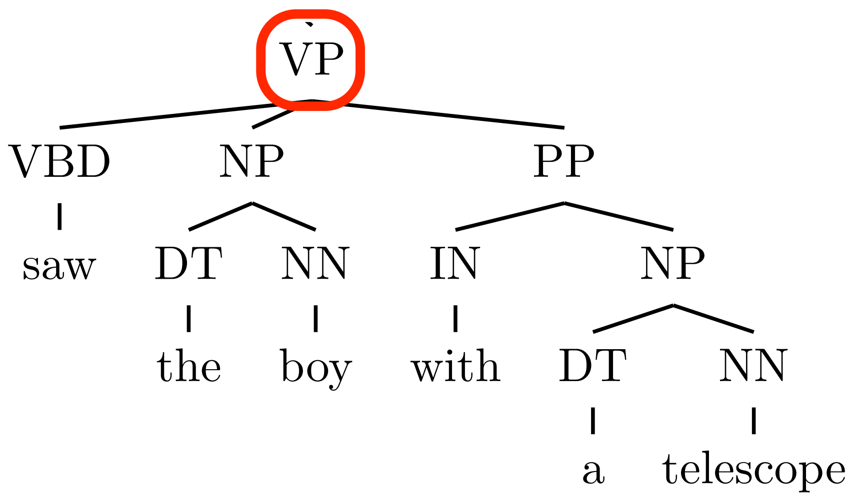
non-local!

CoPar: 4

Factorizing non-local features

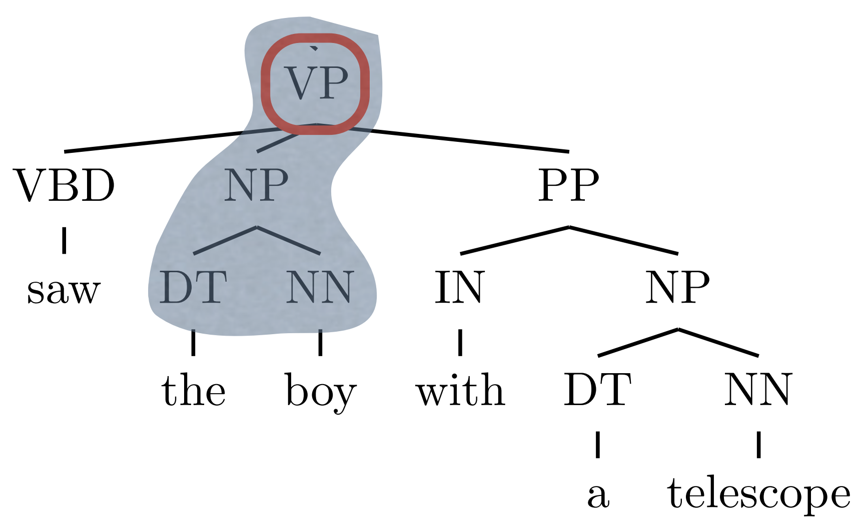
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors

unit instance of ParentRule feature at VP node



Factorizing non-local features

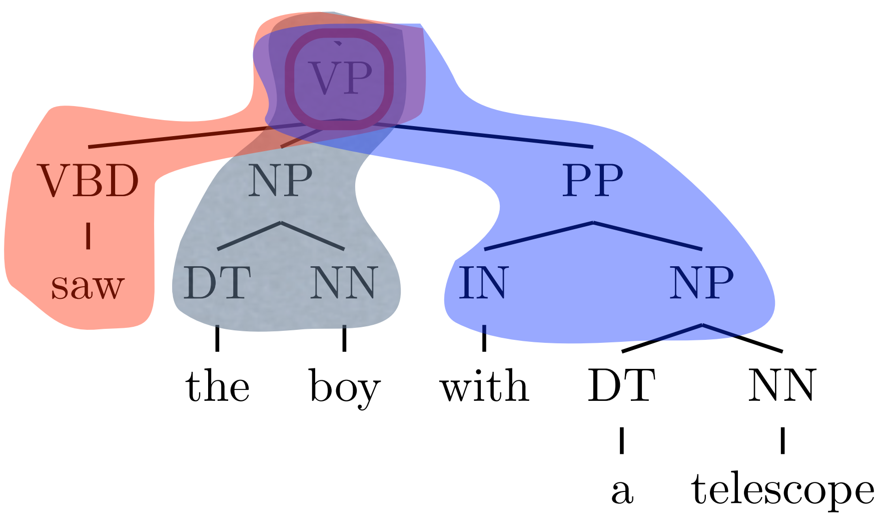
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at VP node

Factorizing non-local features

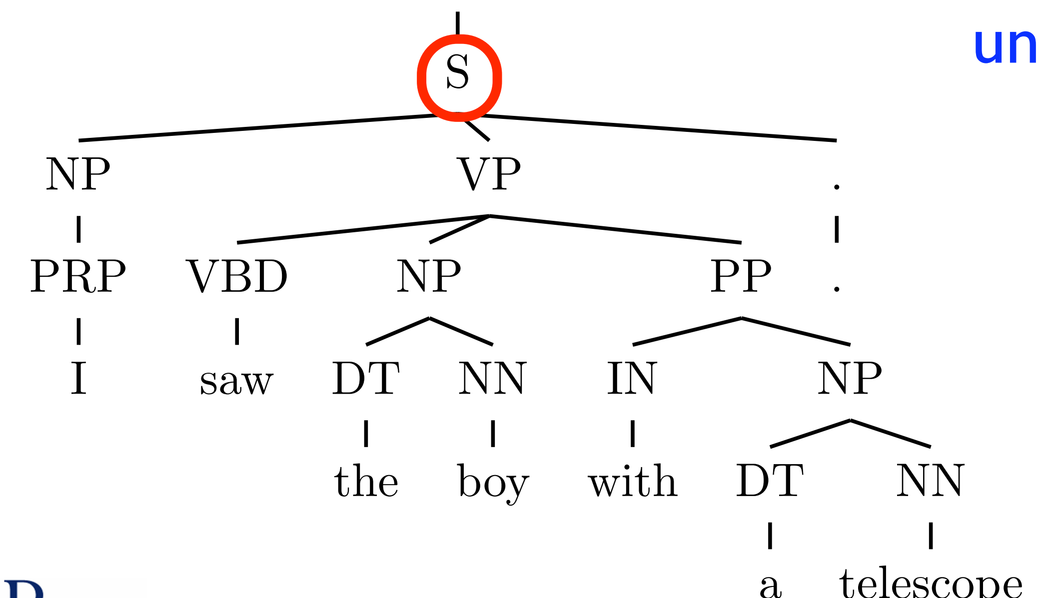
- going bottom-up, at each node
    - compute (partial values of) feature instances that become computable at this level
    - postpone those uncomputable to ancestors



unit instance of ParentRule feature at VP node

Factorizing non-local features

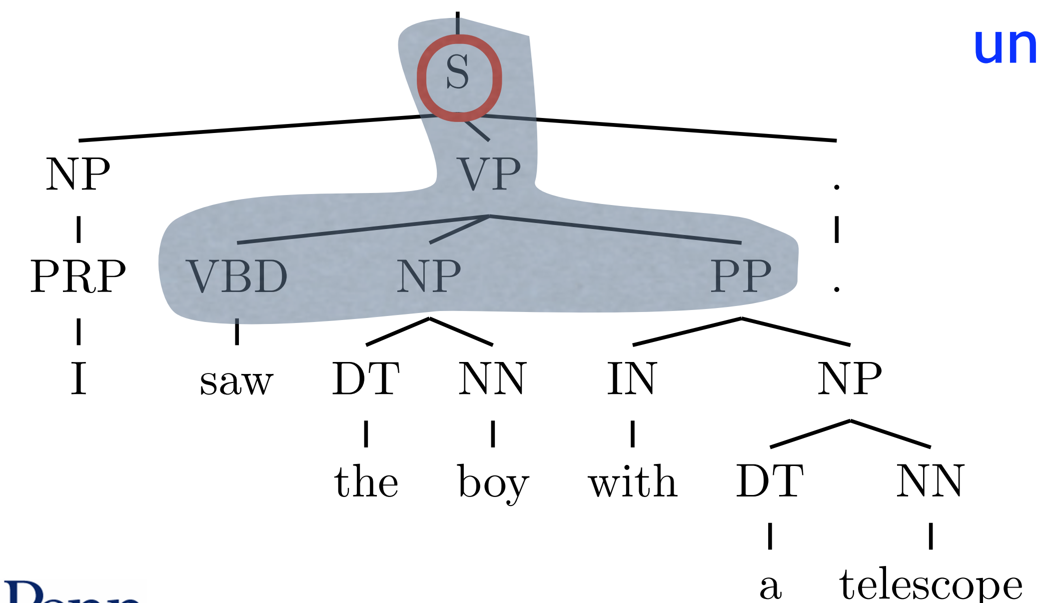
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at S node

Factorizing non-local features

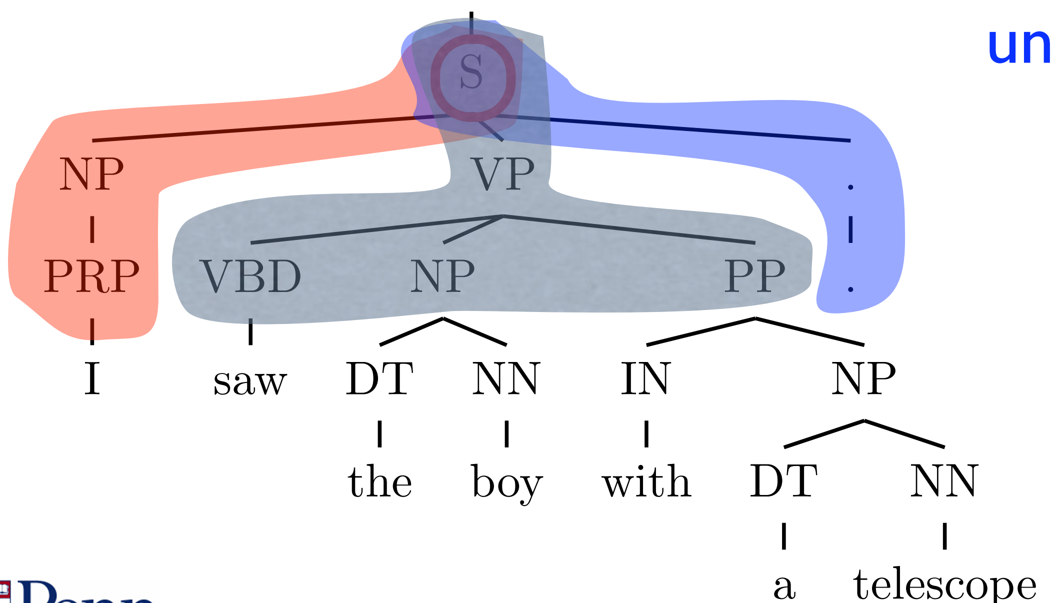
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at S node

Factorizing non-local features

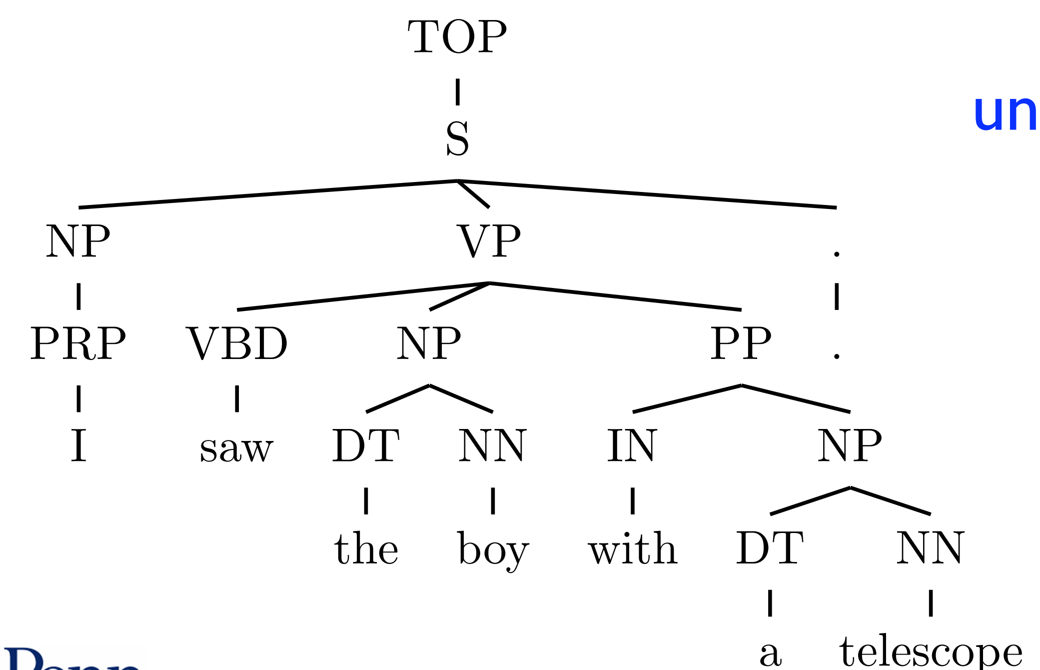
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at S node

Factorizing non-local features

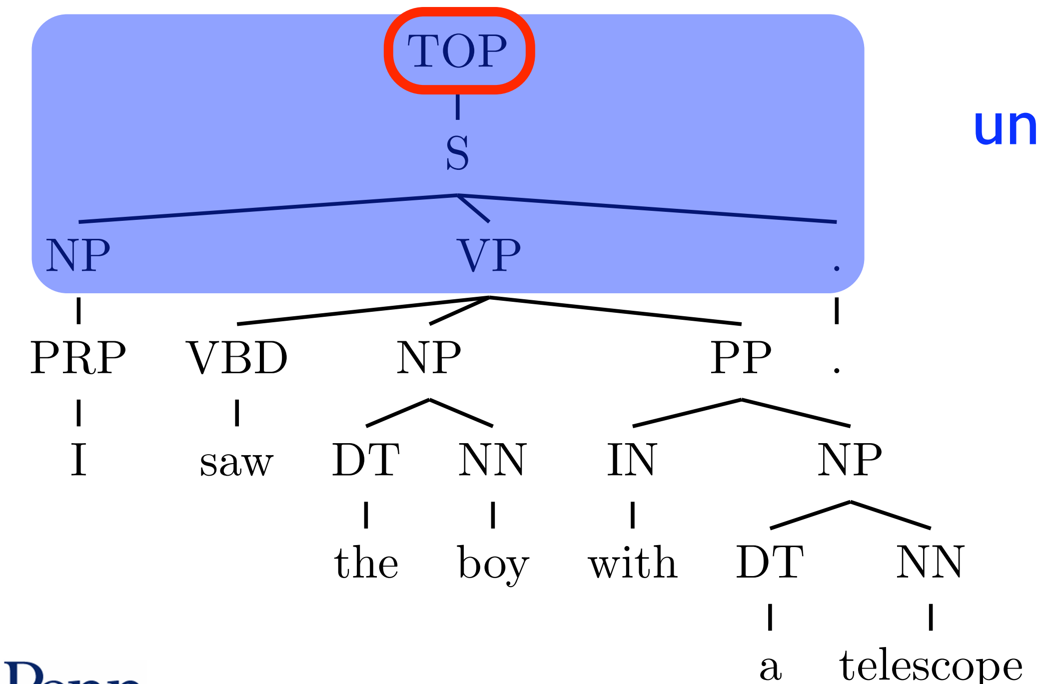
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at TOP node

Factorizing non-local features

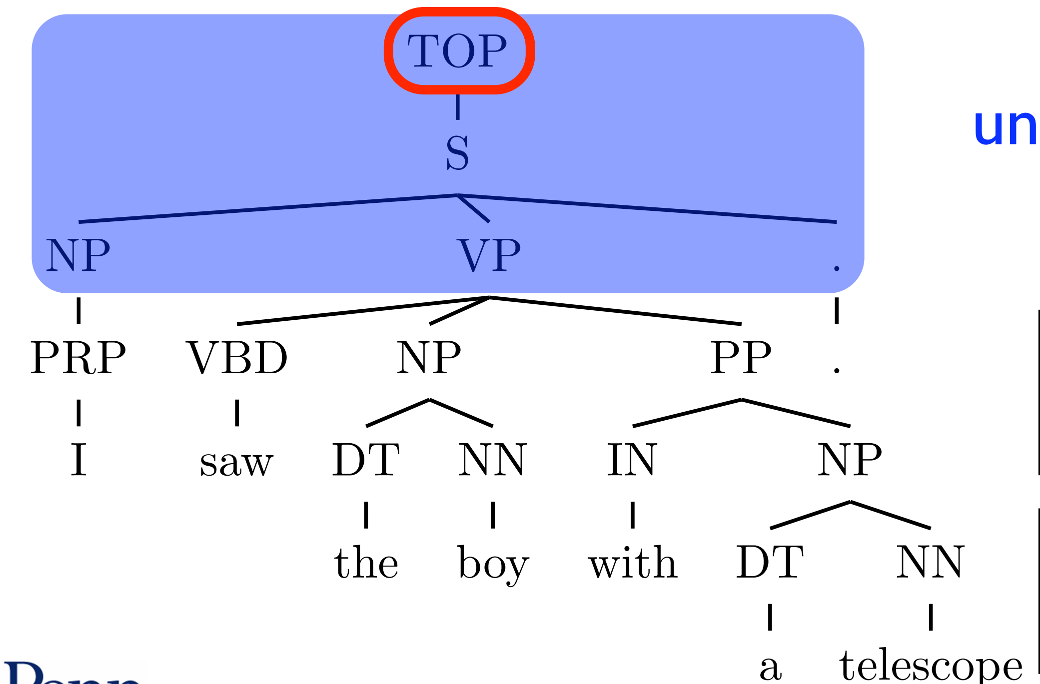
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at TOP node

Factorizing non-local features

- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at TOP node

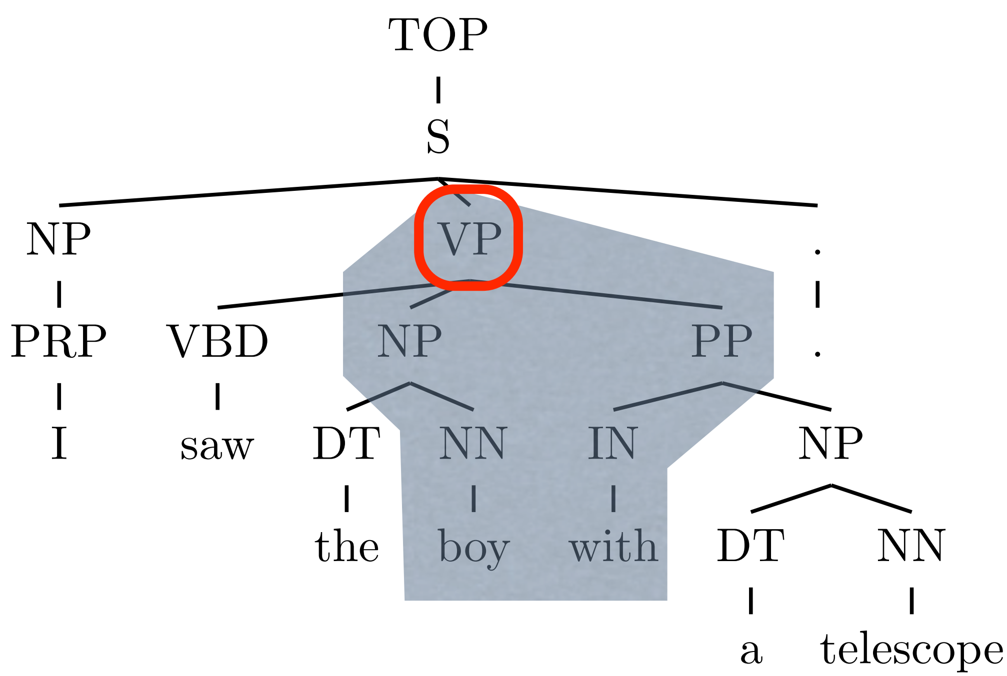
non-local features factor across nodes *dynamically*

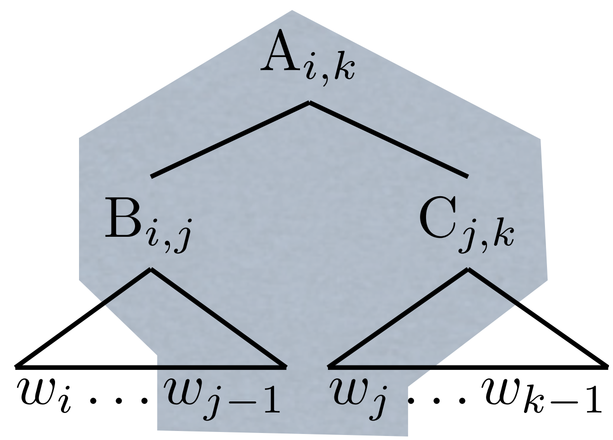
local features factor across hyperedges *statically*

Penn
University of Pennsylvania

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

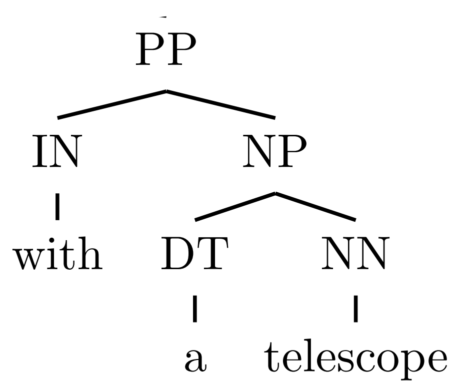


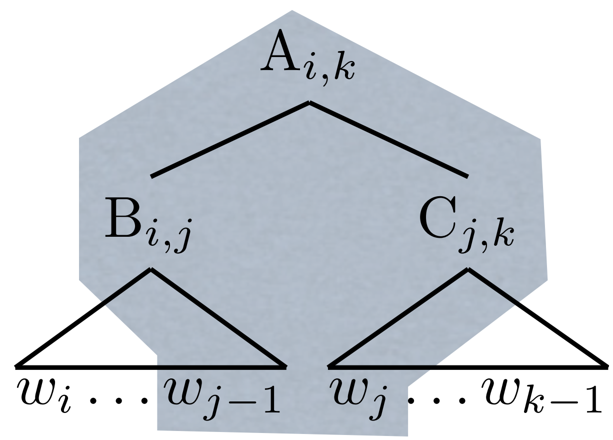


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

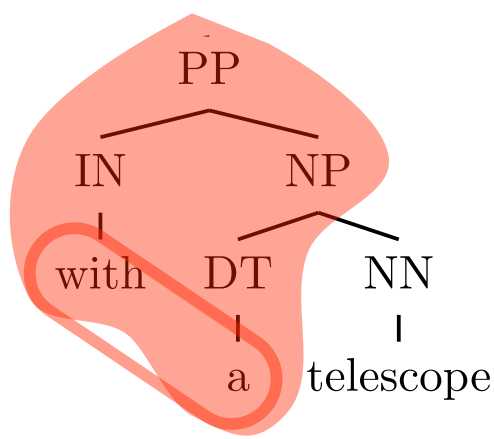


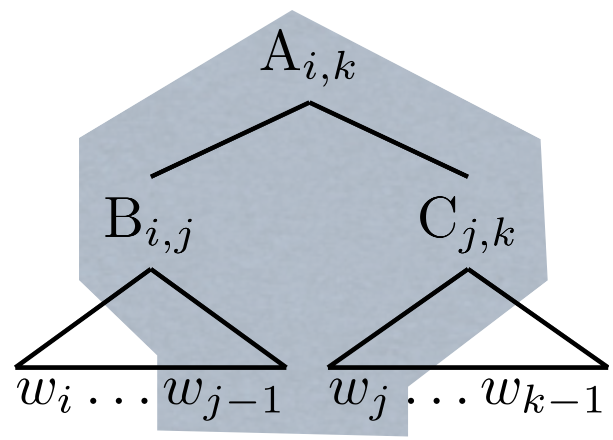


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

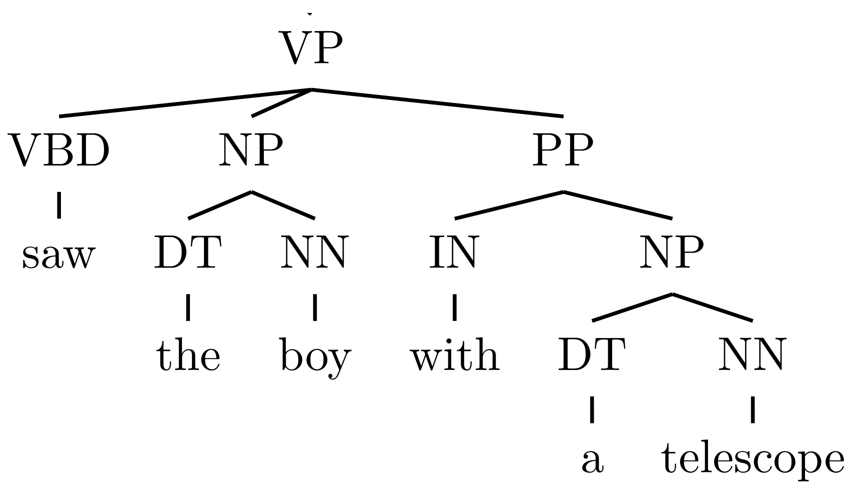


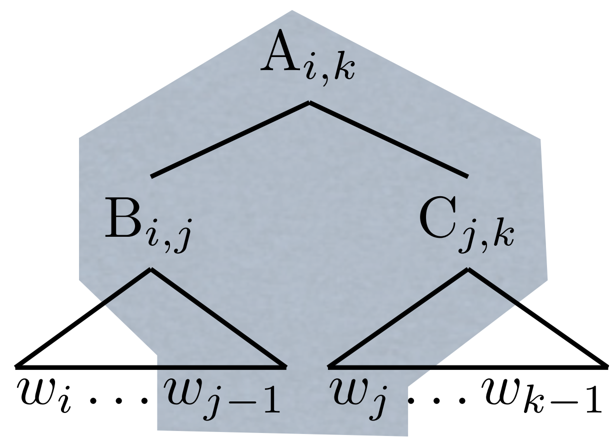


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

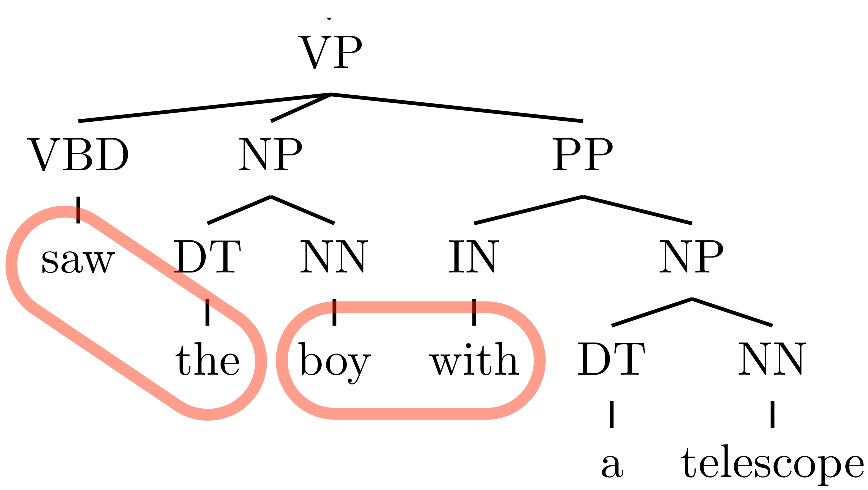


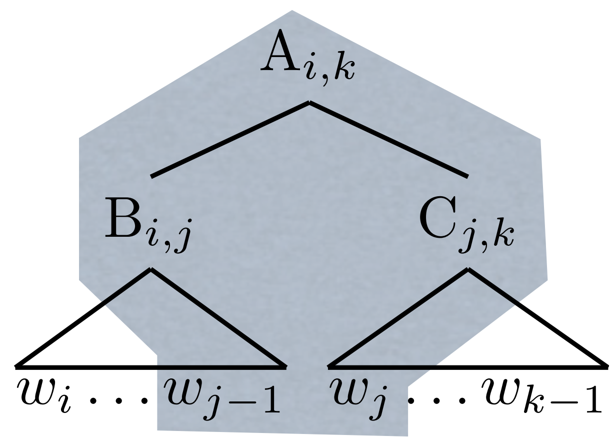


unit instance of node A

NGram Tree (C&J 05)

- an NGram Tree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

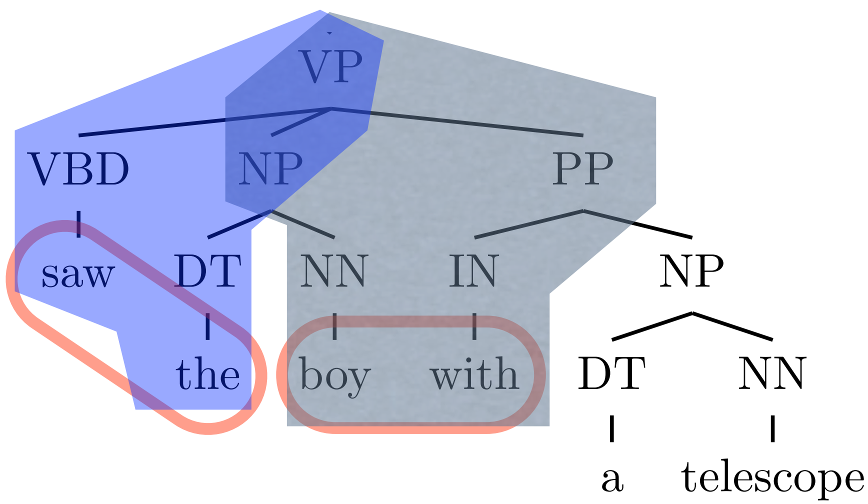


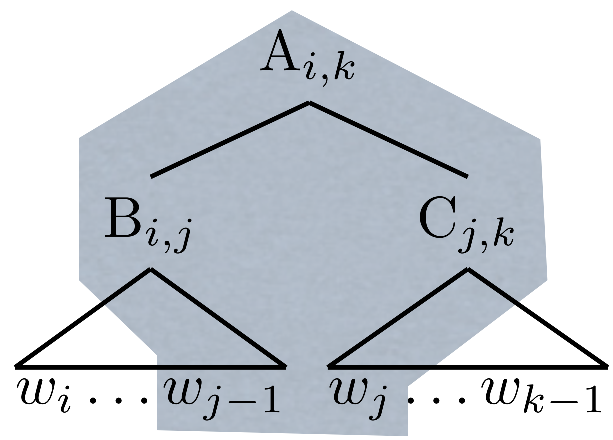


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

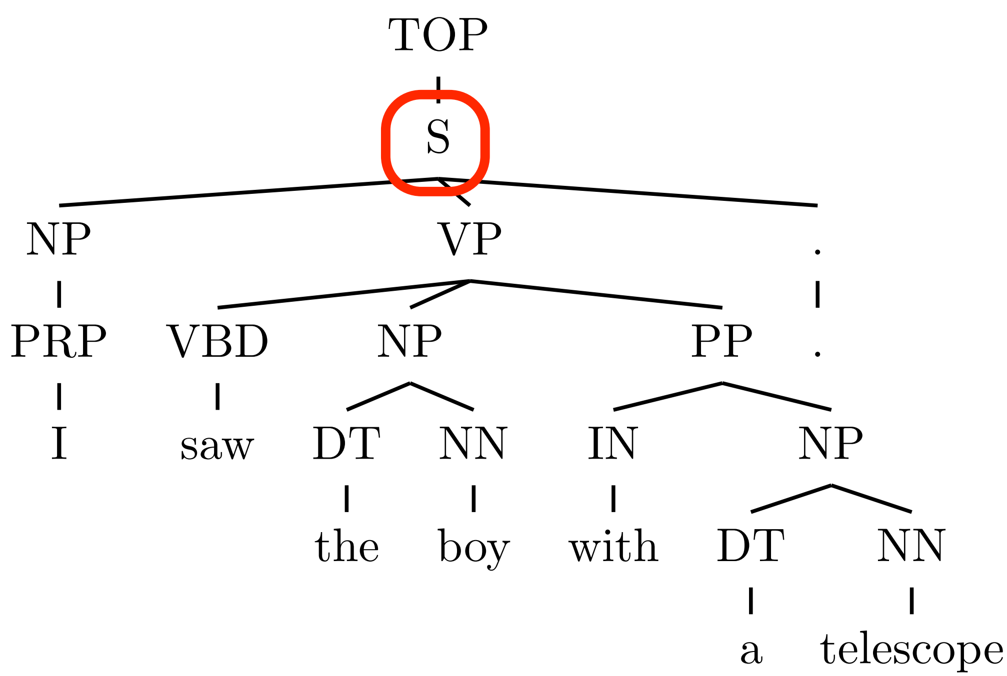


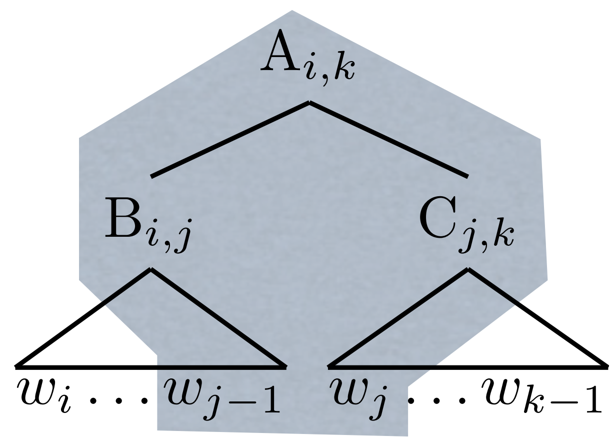


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

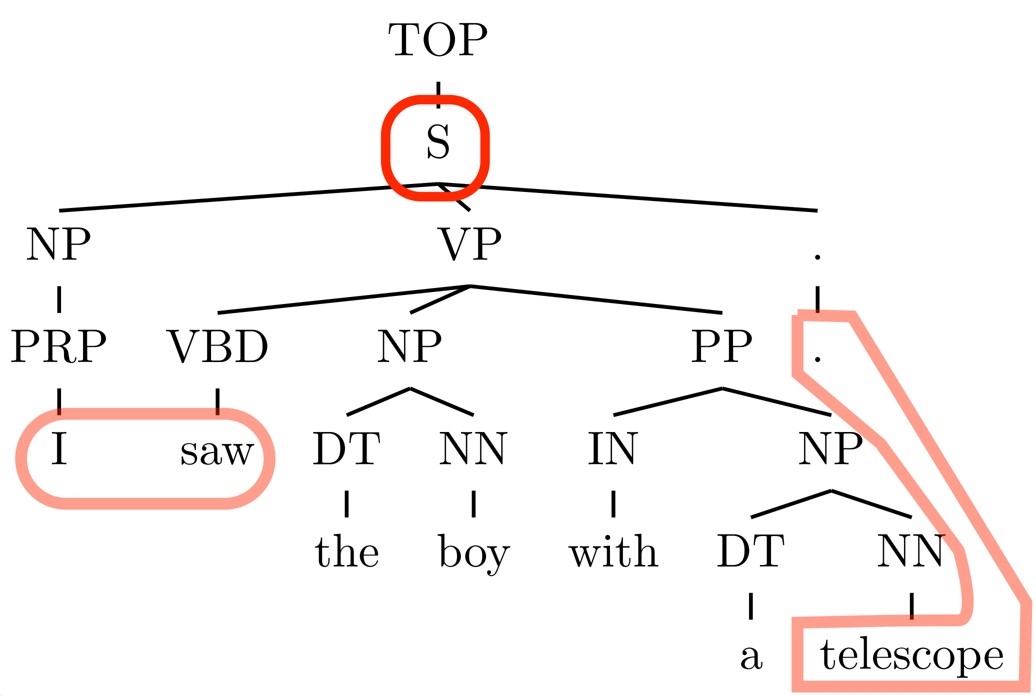


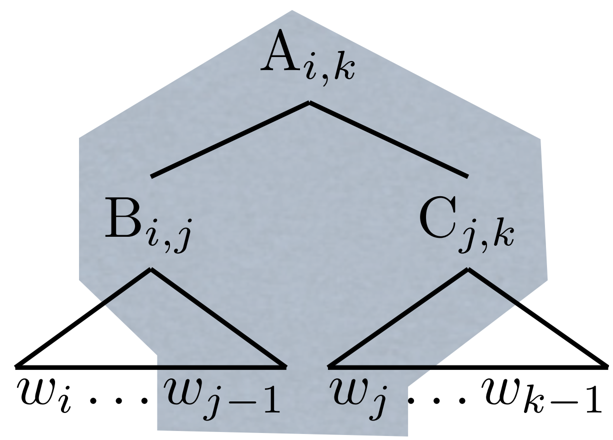


unit instance of node A

NGram Tree (C&J 05)

- an NGram Tree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees

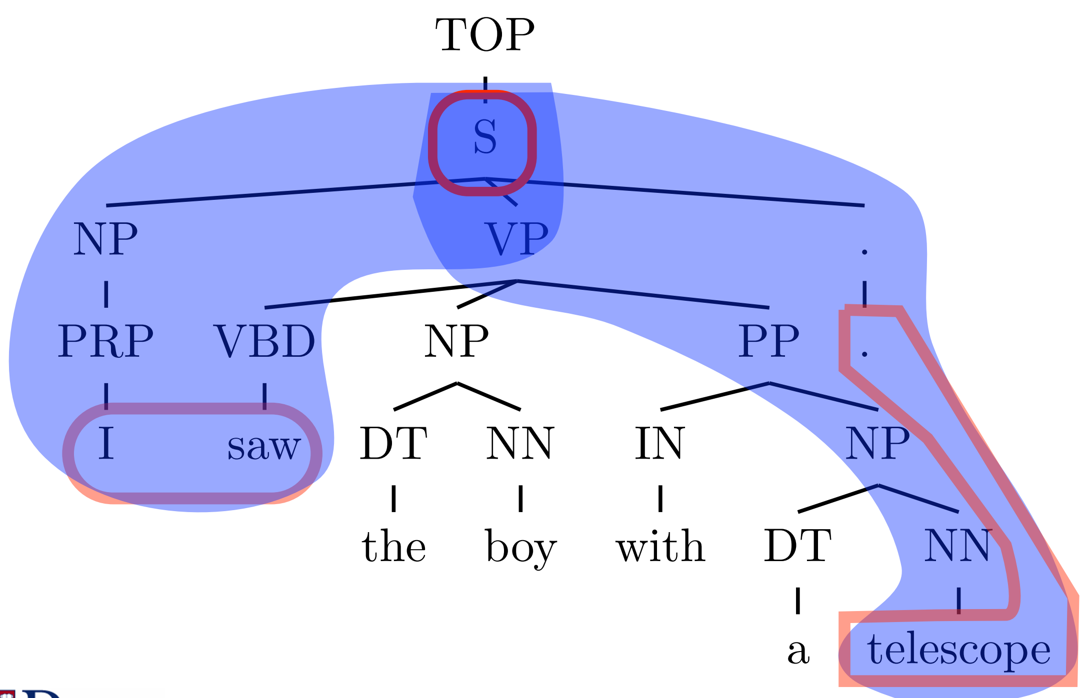


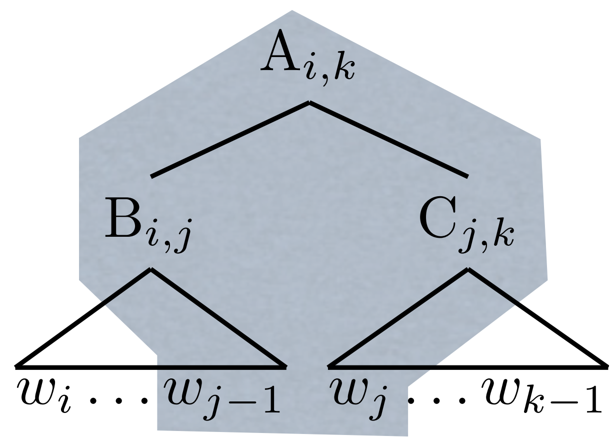


unit instance of node A

NGramTree (C&J 05)

- an NGramTree captures the smallest tree fragment that contains a bigram (two consecutive words)
- unit instances are boundary words between subtrees





unit instance of node A

Outline

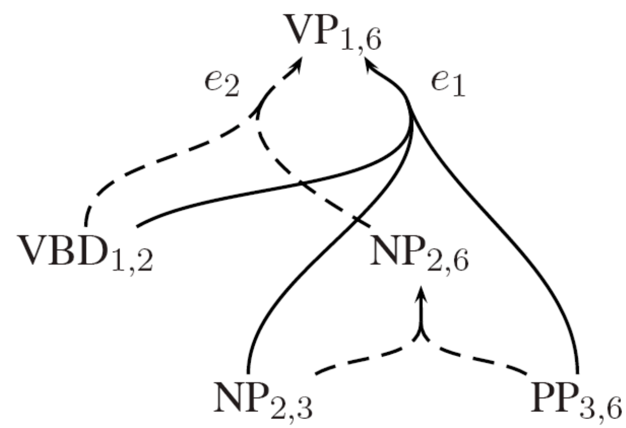
- Packed Forest and General Idea
- Forest Reranking and Non-Local Features
  - Perceptron for Generic Reranking
  - Local vs. Non-Local Features
  - Incremental Computation of Non-Local Features
- Decoding Algorithm
- Experiments

the argmax operator

$$\hat{y} = \mathrm{argmax}_{y \in cand(s_i)} \mathbf{w} \cdot \mathbf{f}(y)$$

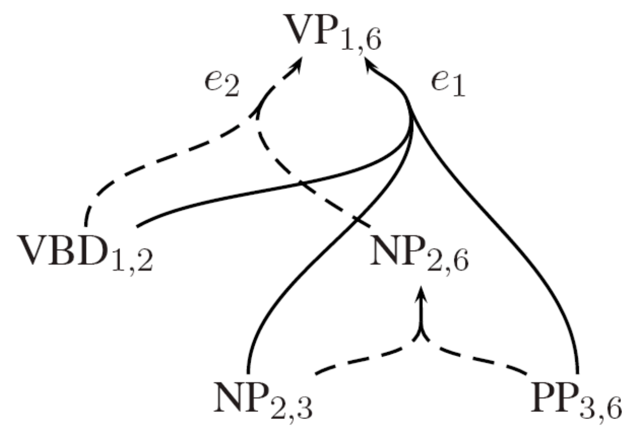
Penn
University of Pennsylvania

General Idea of Decoding



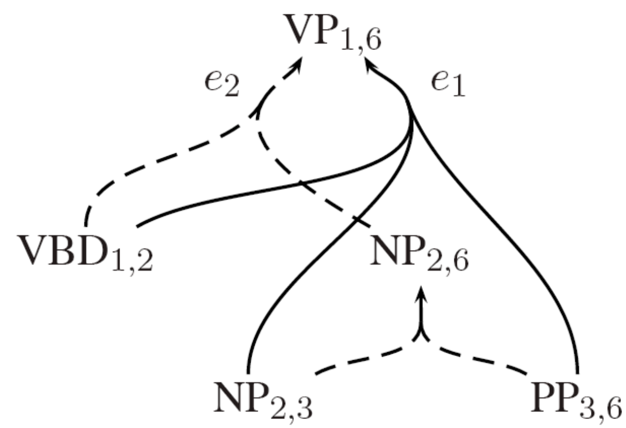
General Idea of Decoding

- bottom-up (chart parsing)



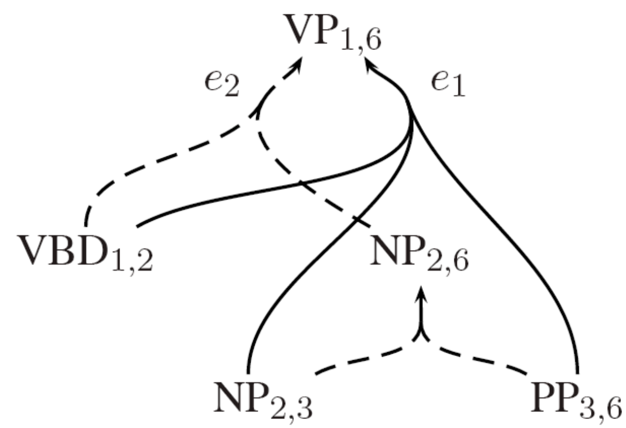
General Idea of Decoding

- bottom-up (chart parsing)
- keep top *k* trees at each node
  - combine top subtrees
  - score unit non-local features

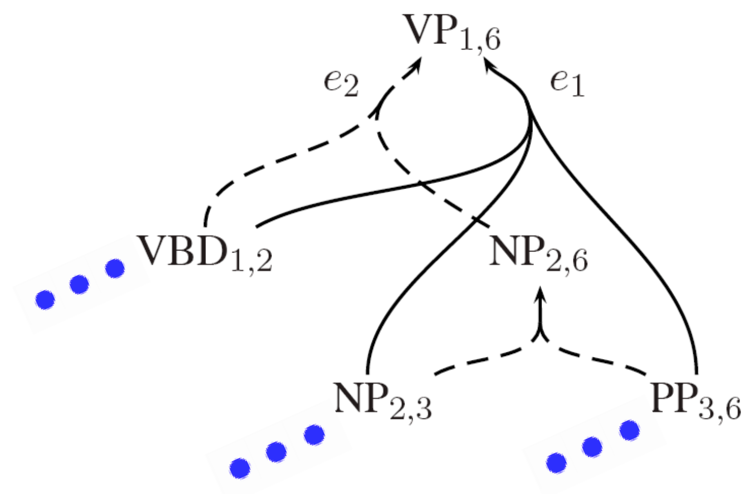


General Idea of Decoding

- bottom-up (chart parsing)
- keep top *k* trees at each node
  - combine top subtrees
  - score unit non-local features
- similar to machine translation decoding with integrated language models
  - non-local features <=> LM combo
  - so we use forest rescoring from MT (Chiang 2007; Huang and Chiang 2007) to speed up the computation



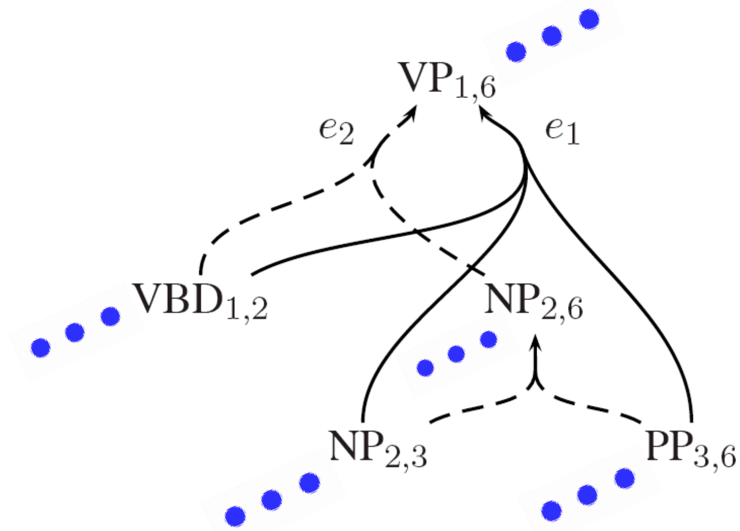
General Idea of Decoding



- bottom-up (chart parsing)
- keep top *k* trees at each node
  - combine top subtrees
  - score unit non-local features
- similar to machine translation decoding with integrated language models
  - non-local features <=> LM combo
  - so we use forest rescoring from MT (Chiang 2007; Huang and Chiang 2007) to speed up the computation

General Idea of Decoding

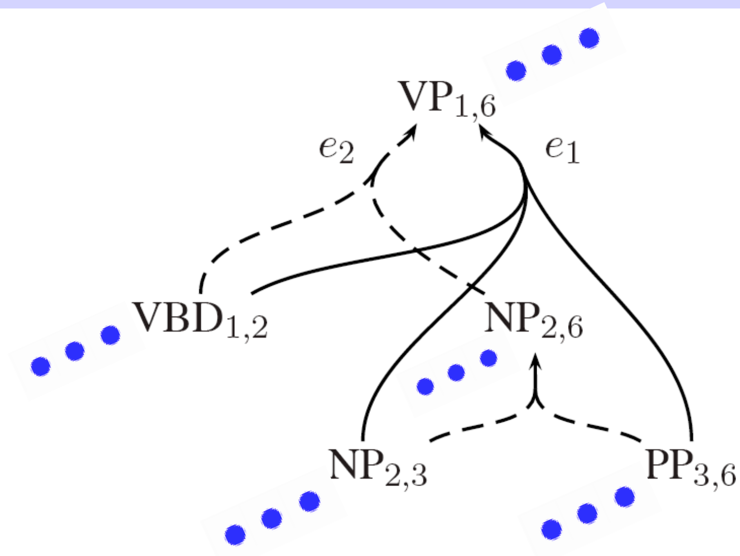
- bottom-up (chart parsing)
- keep top *k* trees at each node
  - combine top subtrees
  - score unit non-local features



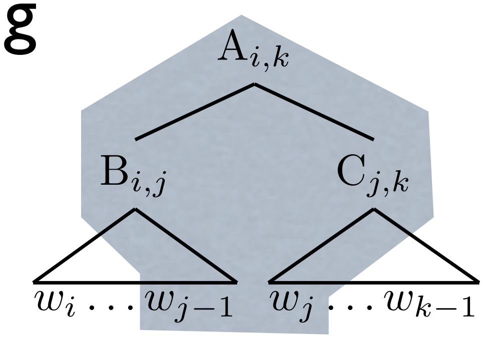
- similar to machine translation decoding with integrated language models
  - non-local features <=> LM combo
  - so we use forest rescoring from MT (Chiang 2007; Huang and Chiang 2007) to speed up the computation

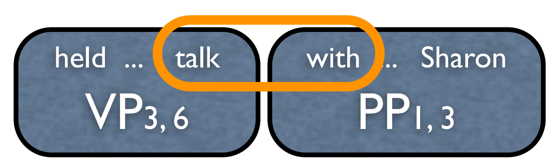
General Idea of Decoding

- bottom-up (chart parsing)
- keep top *k* trees at each node
  - combine top subtrees
  - score unit non-local features



- similar to machine translation decoding with integrated language models
  - non-local features <=> LM combo
  - so we use forest rescoring from MT (Chiang 2007; Huang and Chiang 2007) to speed up the computation

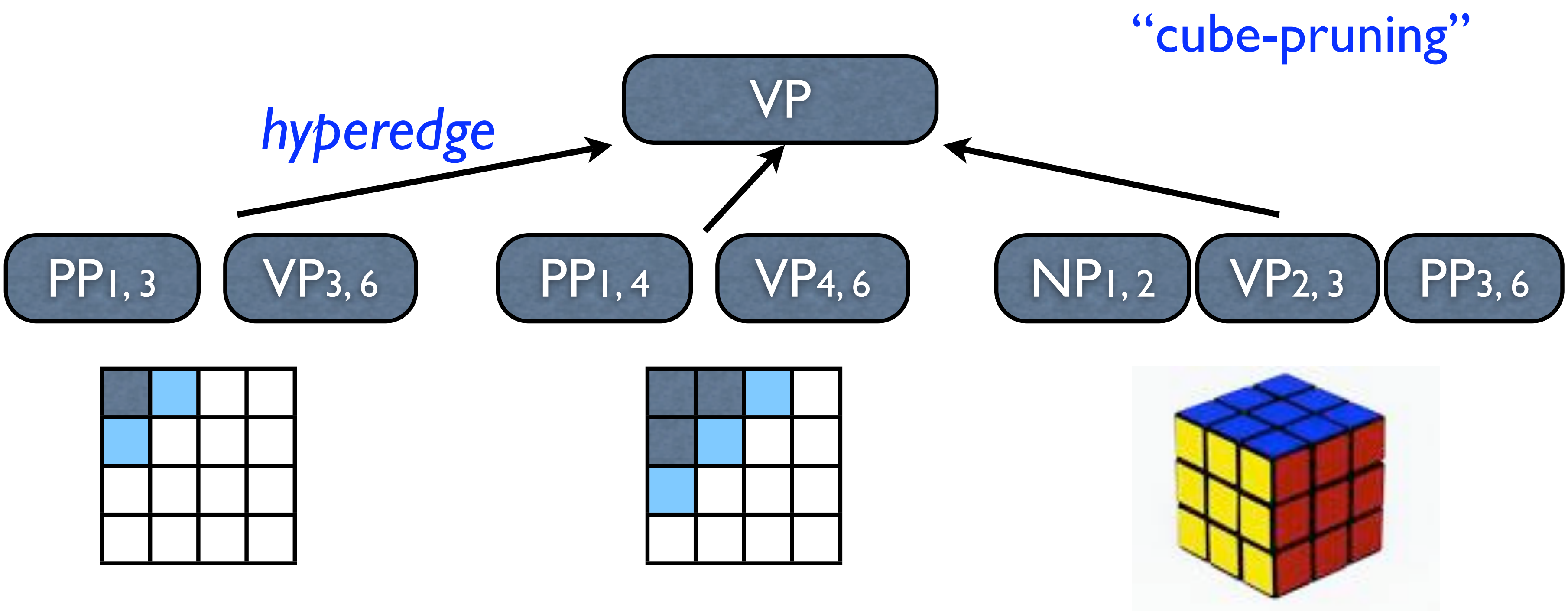




Penn
University of Pennsylvania

Faster Decoding

- best-first exploration of hyperedges simultaneously! significant savings of computation
- most of the item combinations are neglected



(Huang and Chiang, 2005, 2007; Chiang, 2007)

Experiments

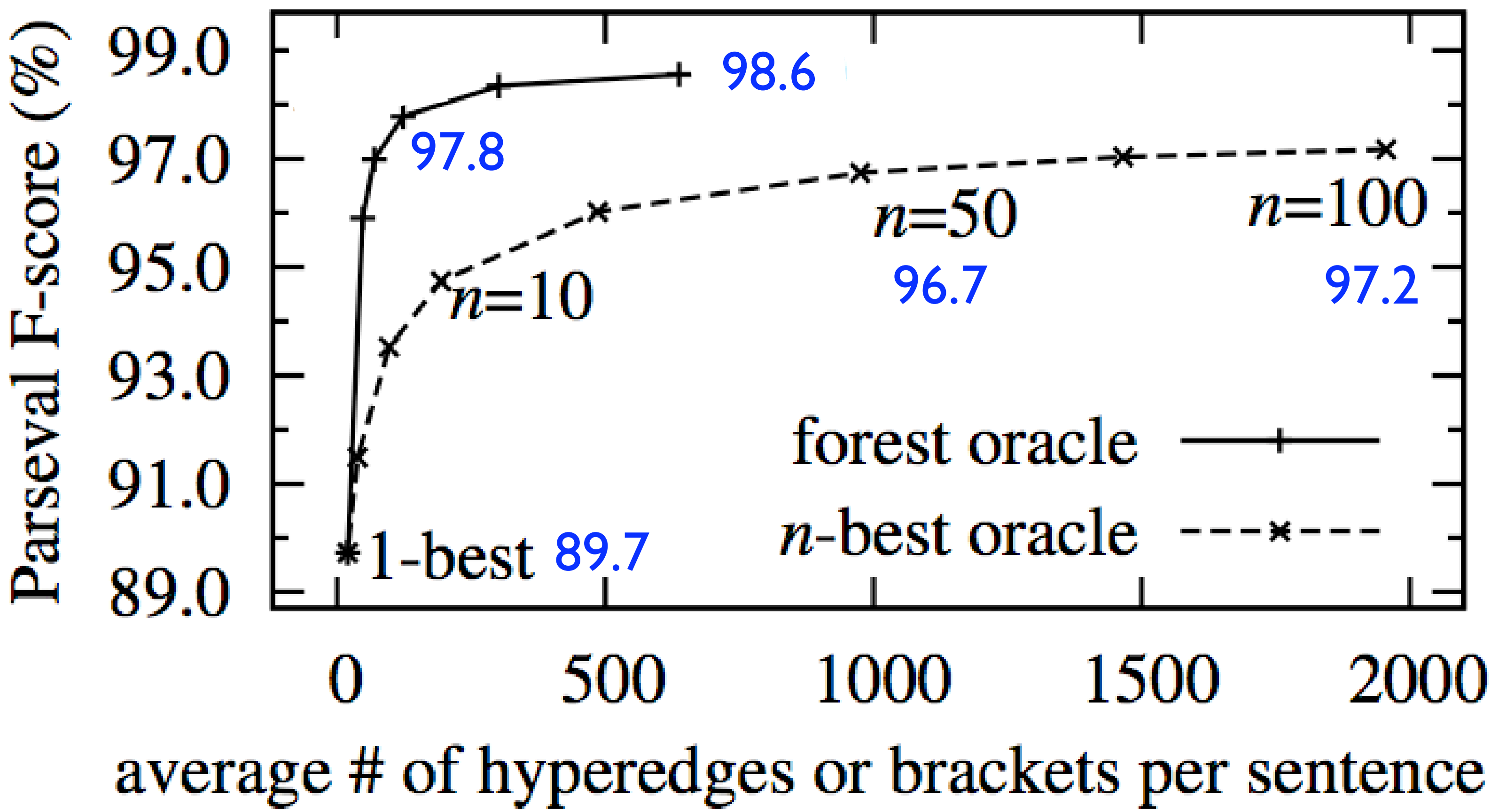
scaled to the whole Penn Treebank

Data Preparation

- use Charniak parser as baseline
- standard split: train: sec 02-21, dev: sec 22, test: sec 23
- training set split into 20 fold (cross-validation style)
- modify Charniak parser to output forests!
  - pruned by an Inside-Outside style algorithm
- use 15 features templates from (Charniak and Johnson, 2005; Collins, 2000); 800, 582 feature instances (~70% local)
- both *n*-best and forest reranking systems implemented in pure Python, on 64-bit Dual-core 3.0 GHz machines

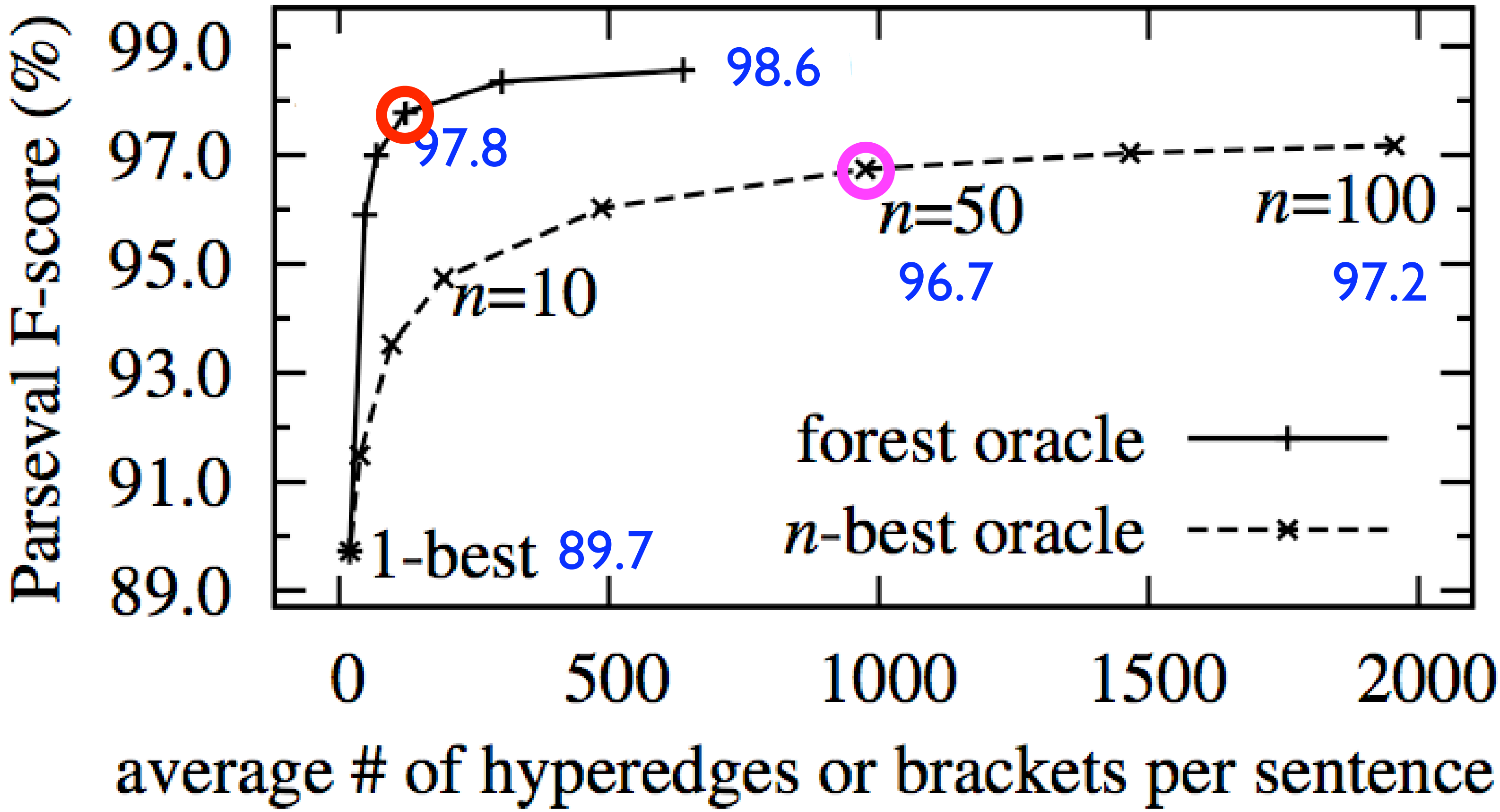
Forest vs. n-best Oracles

- forests enjoy higher oracle scores than *n*-best lists
  - a dynamic programming algorithm for forest oracle



Forest vs. n-best Oracles

- forests enjoy higher oracle scores than *n*-best lists
  - a dynamic programming algorithm for forest oracle



Main Results

- forest reranking outperforms both 50-best and 100-best reranking
- and can be trained on the whole treebank in ~1 day even with a pure Python implementation!

baseline: 1-best Charniak parser		89.72
approach	training time	F1%
50-best reranking	4 x 0.3h	91.43
100-best reranking	4 x 0.7h	91.49
forest reranking	4 x 6.1h	91.69

details in the paper.

Penn
University of Pennsylvania

Comparison with Others

approach	system	F_1%
reranking	Collins (2000)	89.7
	Charniak and Johnson (2005)	91.0
dynamic programming	Petrov and Klein (2008)	88.3
	this work	91.7
generative	Bod (2000)	90.7
	Petrov and Klein (2007)	90.1

semi-supervised	McClosky et al. (2006)	92.1

Penn
University *of* Pennsylvania

Conclusion

- A Framework for Reranking on Packed Forests
  - forests have more variations and smaller sizes
  - dynamic programming algorithm for forest oracles
- Two Key Ideas that made it work
  - incremental, recursive computation of features
  - forest rescoring for approximate decoding
- Discriminative training scaled to the whole PTB
  - better than both 50-best and 100-best reranking
  - better than any previous results trained on PTB

Penn University of Pennsylvania

Conclusion

- more akin to traditional chart parsing, not reranking!
  - multipass search (Goodman, 1997)
    - non-local features in the pruned forest
    - but without blowing up the forest
  - better search algorithms should help!
  - could in principle incorporate fancier features
- also applicable to other problems involving forest
  - sequence segmentation/labeling, dependency parsing, machine translation, generation, ...

Penn
University of Pennsylvania

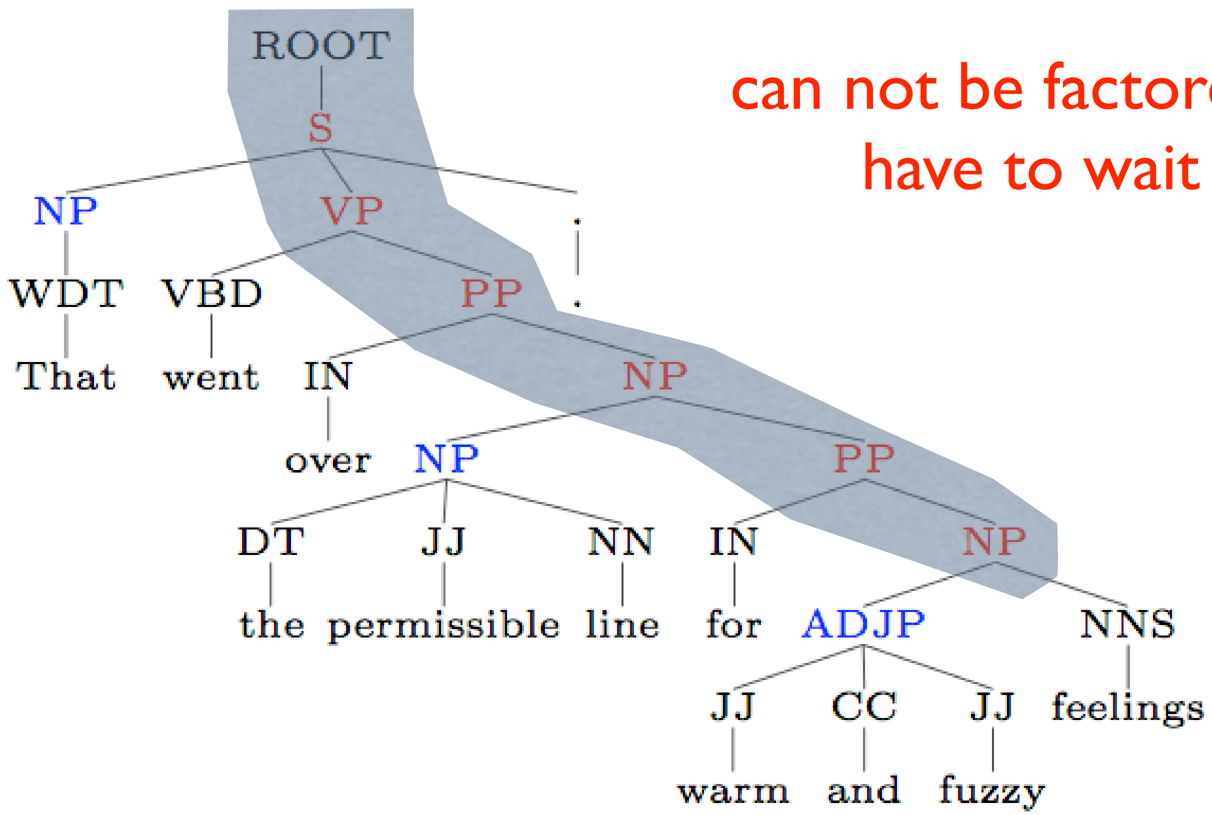
Forest is your friend.   Save the forest.

Thank you!

Forest-dumping Charniak parser
will be available online.

Global Feature - RightBranch

- length of rightmost (non-punctuation) path
  - English has a right-branching tendency

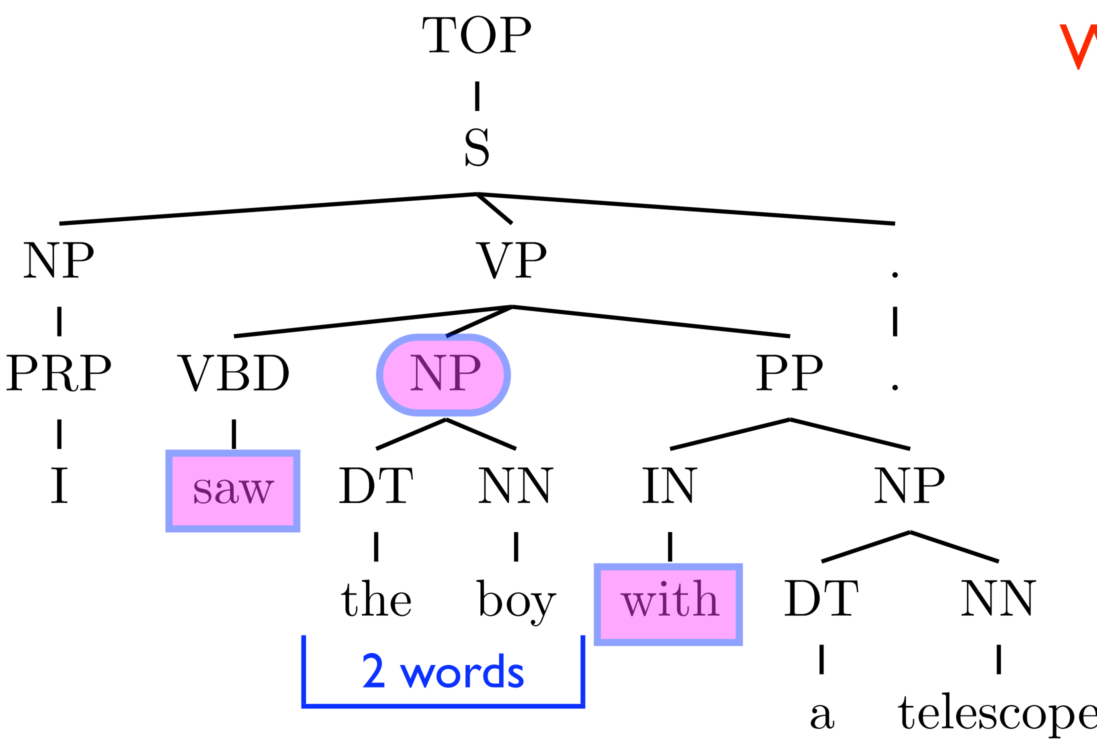


can not be factored anywhere
have to wait till root

(Charniak and Johnson, 2005)

WordEdges (C&J 05)

- a WordEdges feature classifies a node by its label, (binned) span length, and surrounding words
- a POSEdges feature uses surrounding POS tags

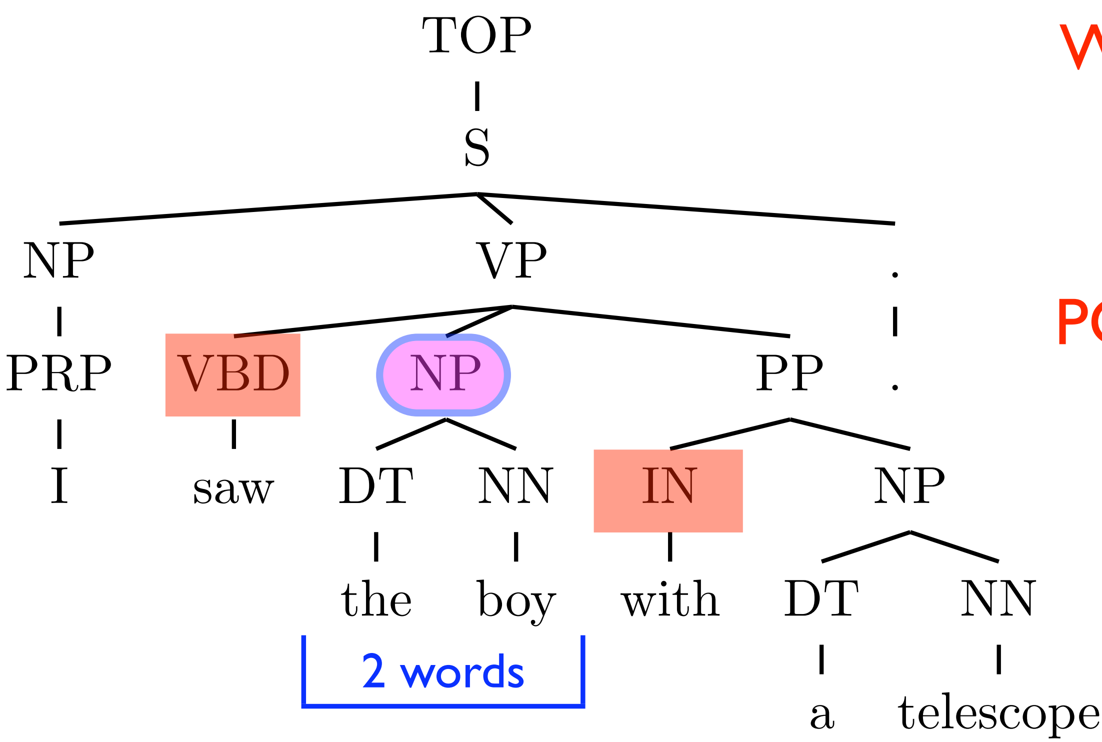


WordEdges is local

$$f_{400}(y) = f_{\text{NP 2 saw with}}(y) = 1$$

WordEdges (C&J 05)

- a WordEdges feature classifies a node by its label, (binned) span length, and surrounding words
- a POSEdges feature uses surrounding POS tags



WordEdges is local

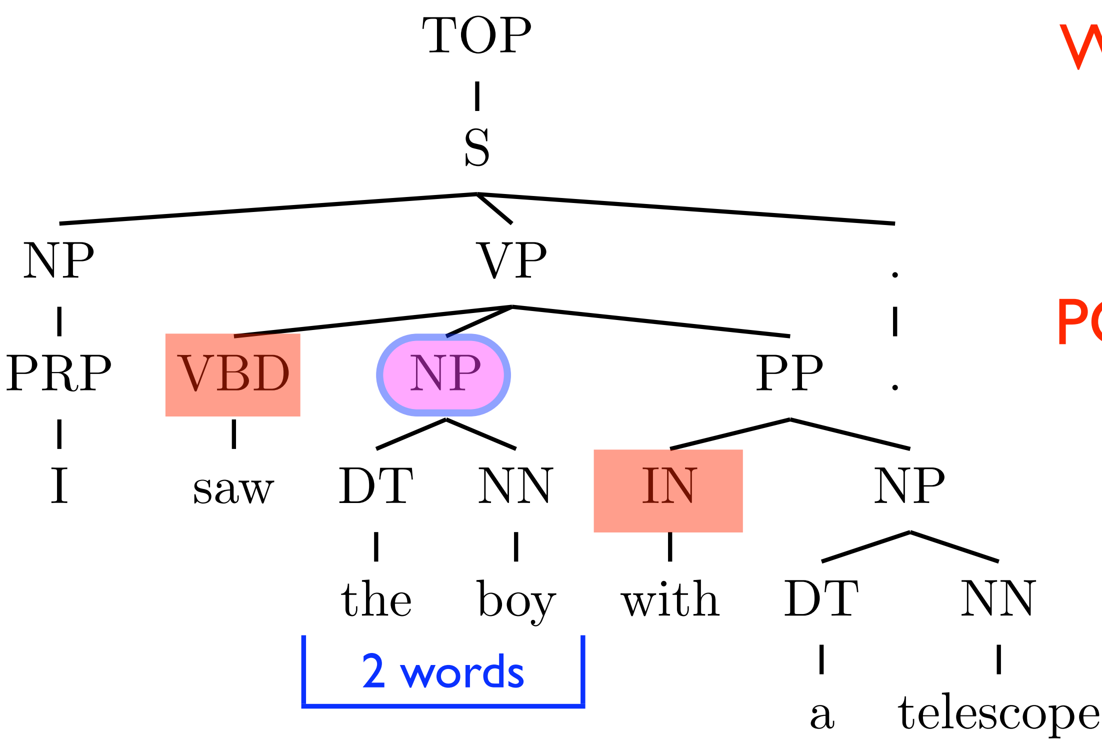
$$f_{400}(y) = f_{\text{NP 2 saw with}}(y) = 1$$

POSEdges is non-local

$$f_{800}(y) = f_{\text{NP 2 VBD IN}}(y) = 1$$

WordEdges (C&J 05)

- a WordEdges feature classifies a node by its label, (binned) span length, and surrounding words
- a POSEdges feature uses surrounding POS tags



WordEdges is local

$$f_{400}(y) = f_{\text{NP 2 saw with}}(y) = 1$$

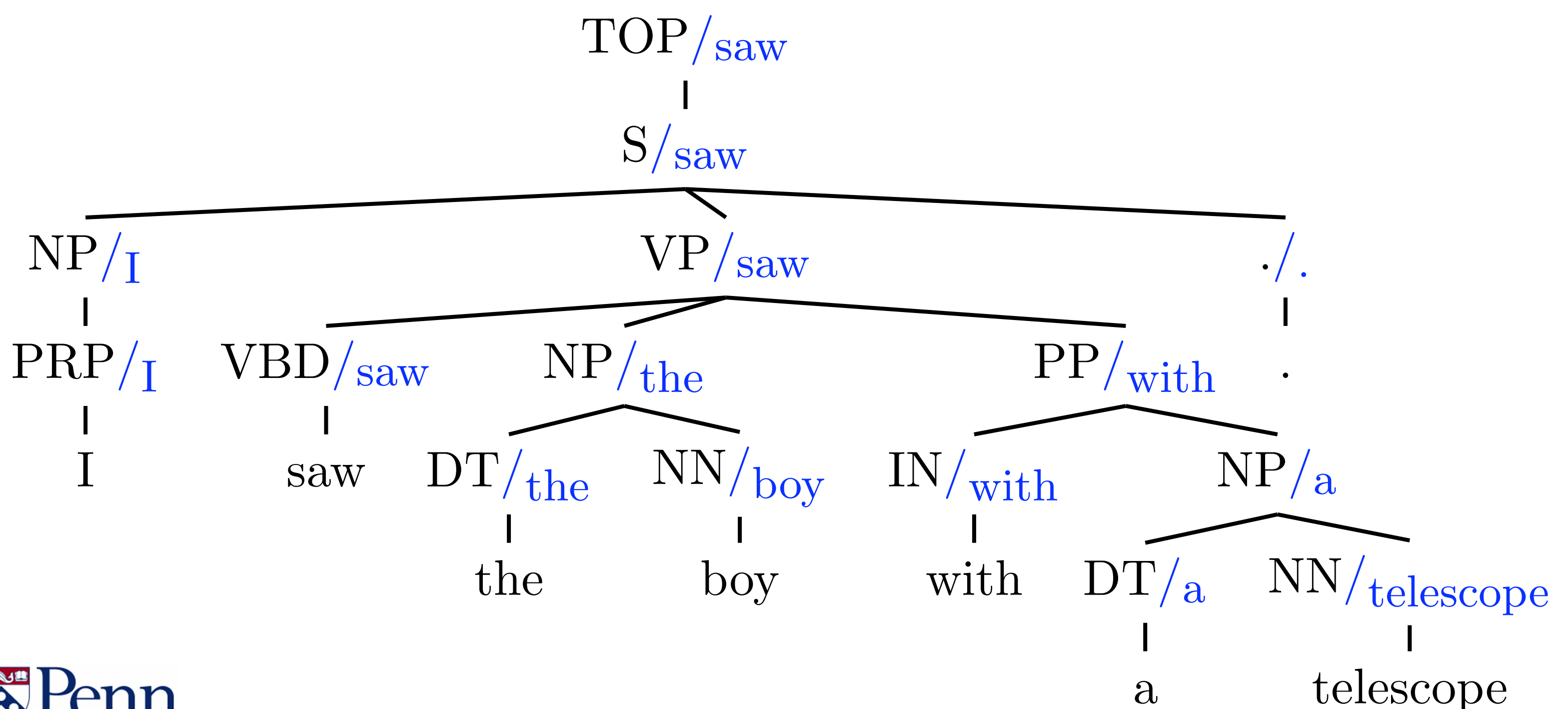
POSEdges is non-local

$$f_{800}(y) = f_{\text{NP 2 VBD IN}}(y) = 1$$

local features comprise ~70% of all instances!

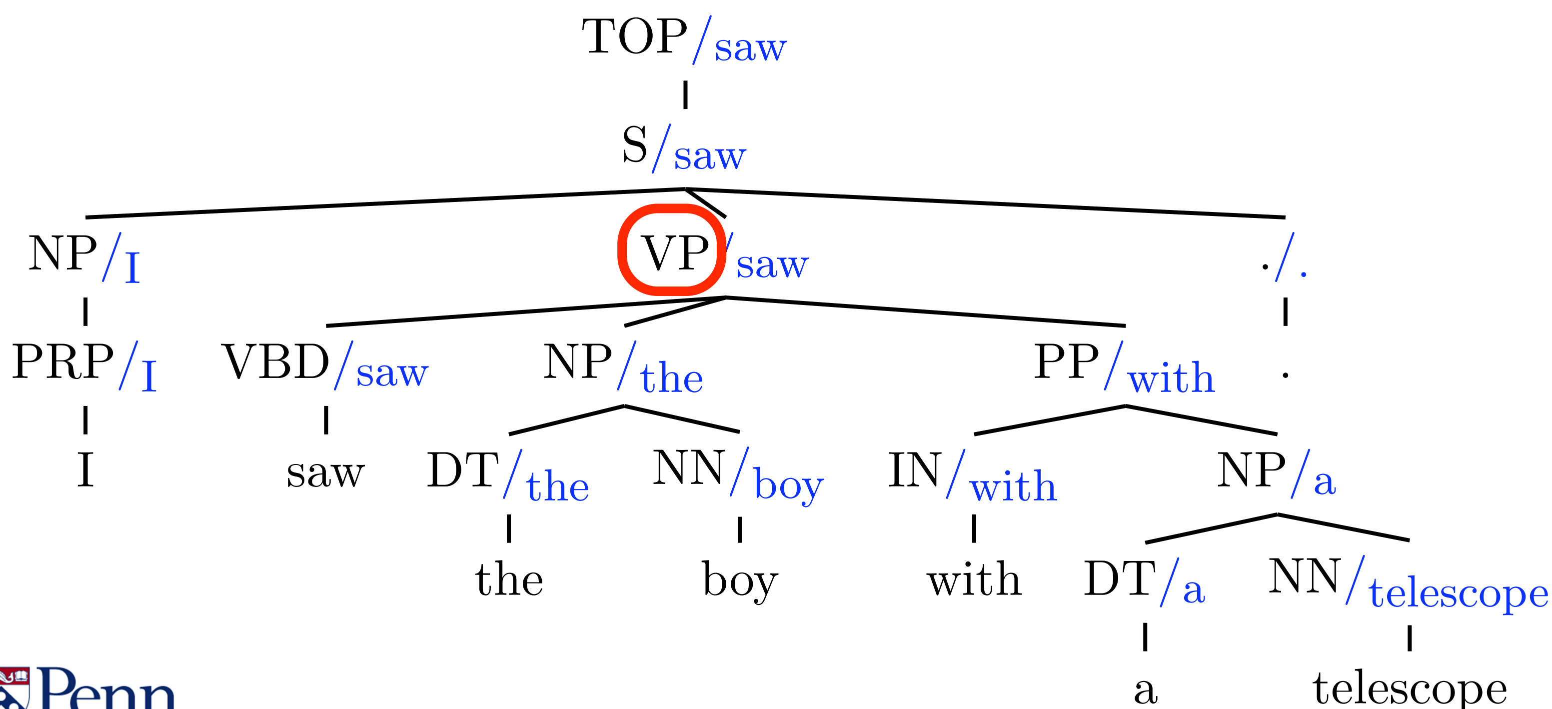
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children



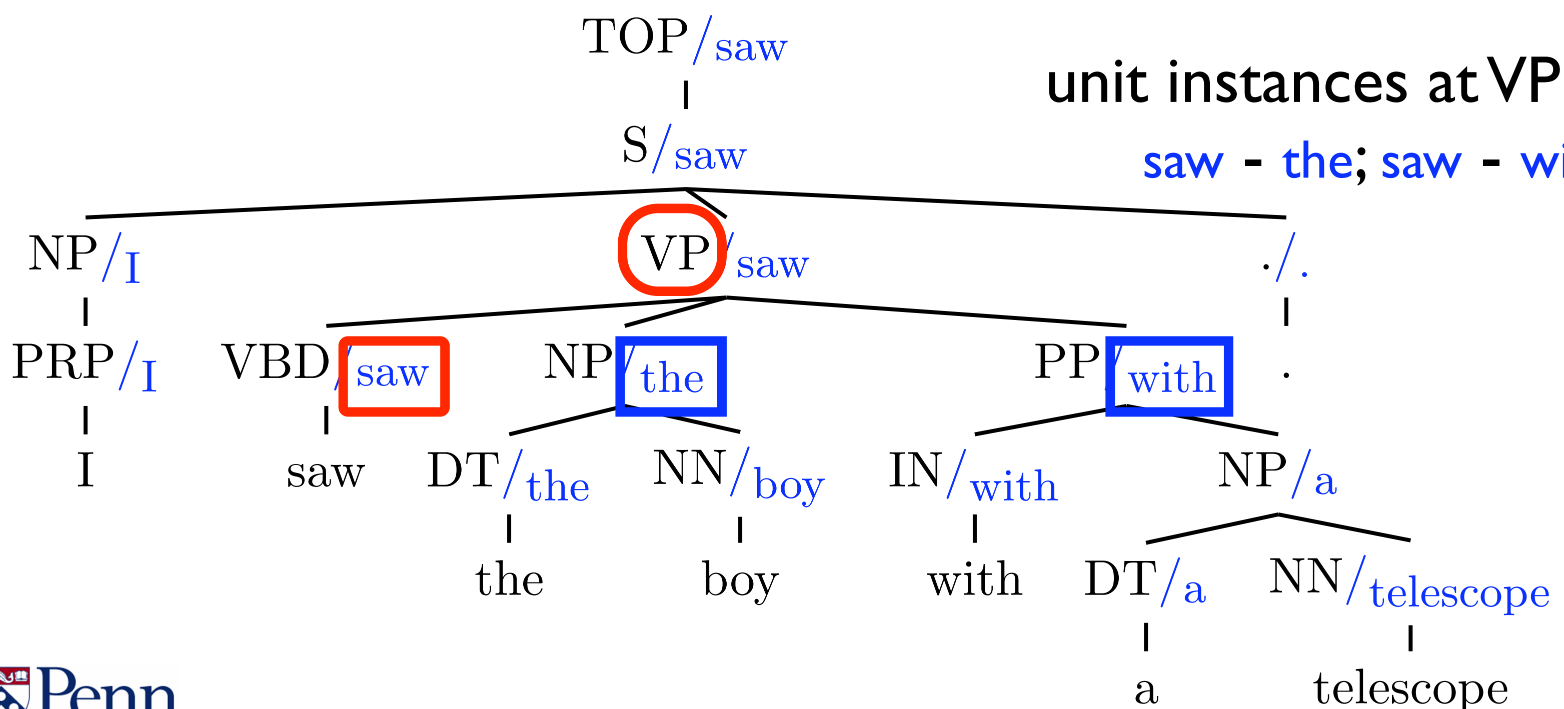
Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children



Heads (C&J 05, Collins 00)

- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children

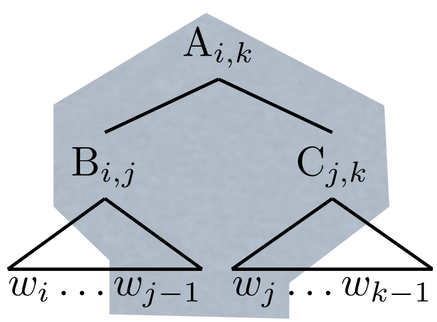


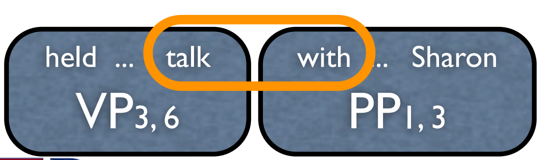
unit instances at VP node

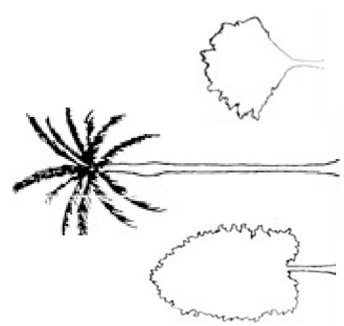
saw - the; saw - with

Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 07)
- priority queue for next-best (Huang and Chiang, 2005)
  - each iteration pops the best and pushes successors
  - unit non-local feature costs as a non-monotonic cost



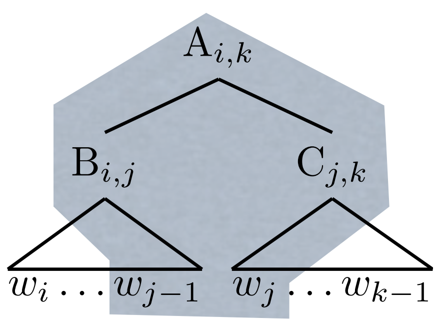


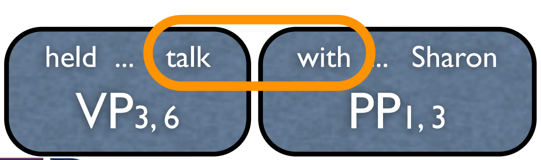


	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

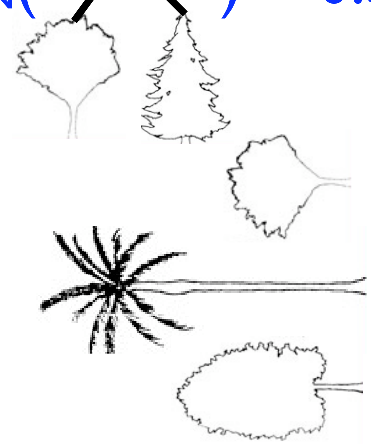
Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 07)
- priority queue for next-best (Huang and Chiang, 2005)
  - each iteration pops the best and pushes successors
  - unit non-local feature costs as a non-monotonic cost





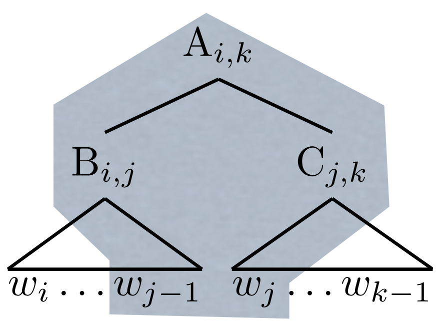
$\mathbf{w} \cdot \mathbf{f}_N(\ \ ) = 0.5$

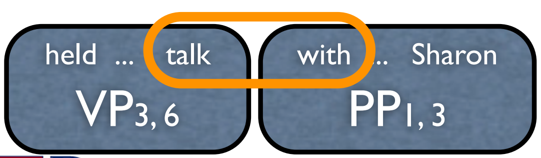


	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

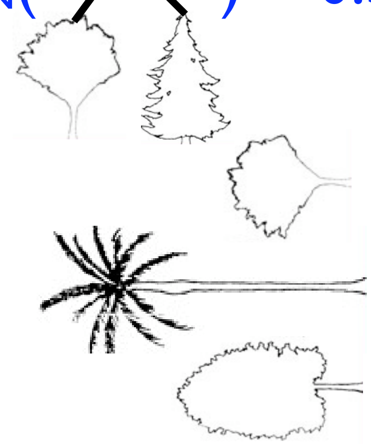
Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 2007)
- priority queue for next-best (Huang and Chiang, 2005)
  - each iteration pops the best and pushes successors
  - unit non-local feature costs as a non-monotonic cost





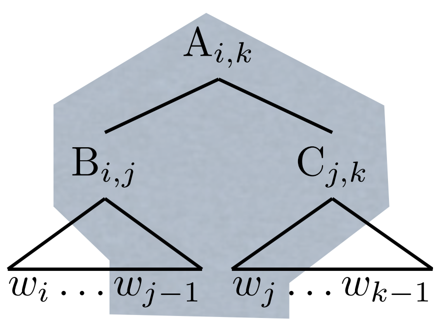
$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$

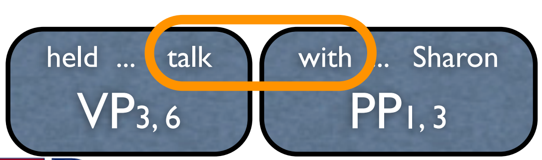


	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

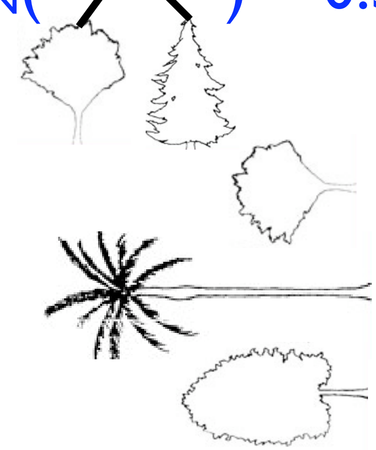
Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - forest rescoring from MT (Chiang 2007; Huang and Chiang 2007)
- priority queue for next-best (Huang and Chiang, 2005)
  - each iteration pops the best and pushes successors
  - unit non-local feature costs as a non-monotonic cost





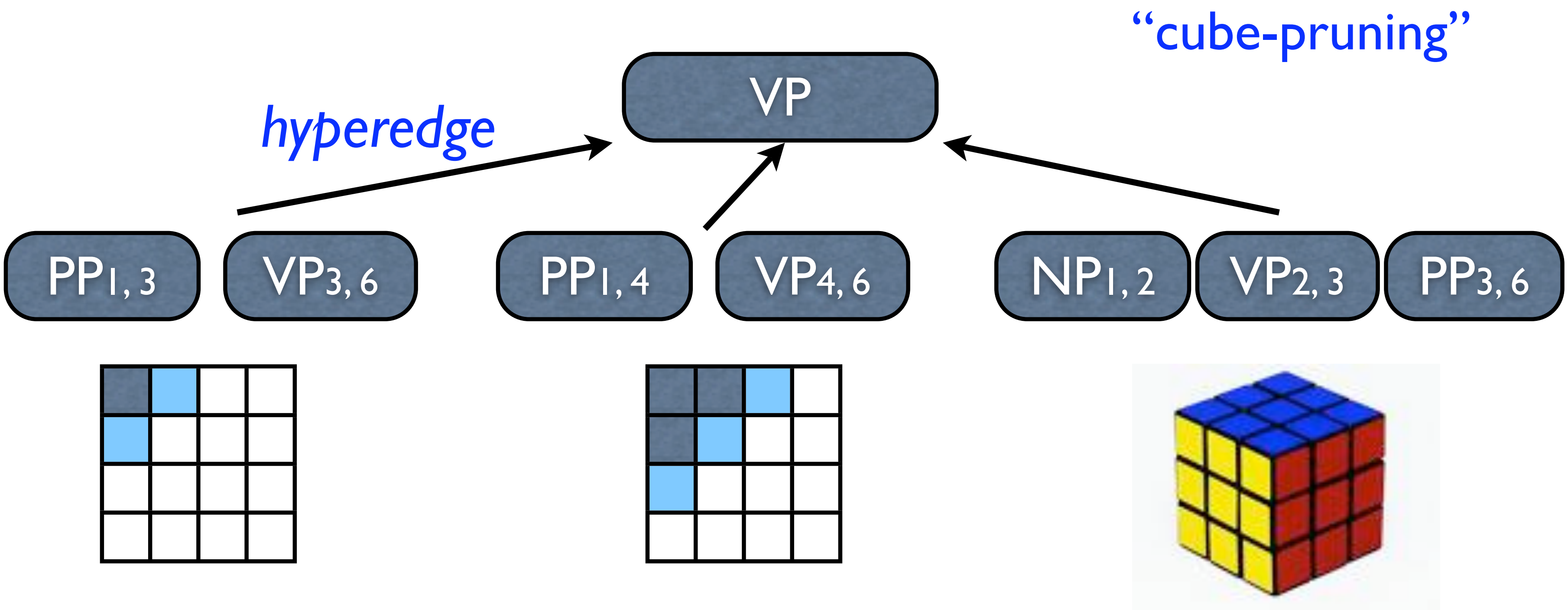
$\mathbf{w} \cdot \mathbf{f}_N($ $) = 0.5$



	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

Approximate Decoding

- process all hyperedges simultaneously!
  significant savings of computation



complexity: $O(E + V \mathbf{U} k \log k)$,

bottom-neck: the time for on-the-fly extraction

Penn University of Pennsylvania

(Huang and Chiang, 2005; 2007; Chiang, 2007)

Forest Oracle

the candidate tree that is closest to gold-standard

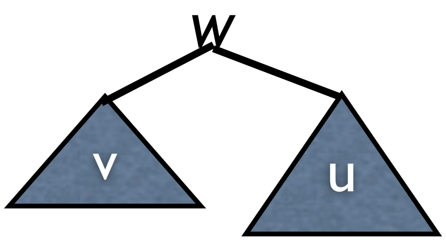
Optimal Parseval F-score

- find the tree in the forest with highest F-score
- Parseval F_1-score is the harmonic mean between labeled precision and labeled recall
  - can not optimize F-scores on sub-forests separately
  - can not optimize precision and recall simultaneously
- we instead use dynamic programming
  - optimizes the number of matched brackets per given number of test brackets
  - "when the test (sub-) parse has 5 brackets, what is the max. number of matched brackets?"

Combining Oracle Functions

- to combine two nodes along a hyperedge, we need to distribute test brackets between the two, and optimize the number of matches

$$(f \otimes g)(t) \triangleq \max_{t_1+t_2=t} f(t_1) + g(t_2)$$



t	f(t)
2	1
3	2

$\otimes$

t	g(t)
4	4
5	4

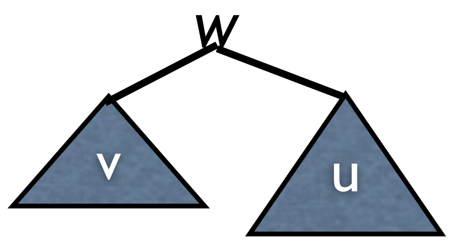
=

t	$(f\otimes g)(t)$
6	5
7	6
8	6

Combining Oracle Functions

- to combine two nodes along a hyperedge, we need to distribute test brackets between the two, and optimize the number of matches

$$(f \otimes g)(t) \triangleq \max_{t_1+t_2=t} f(t_1) + g(t_2)$$



t	f(t)
2	1
3	2

$\otimes$

t	g(t)
4	4
5	4

=

t	$(f\otimes g)(t)$
6	5
7	6
8	6

ora[*w*]

this node matched?

N

t	$(f\otimes g)\Uparrow_{(1,0)}(t)$
7	5
8	6
9	6

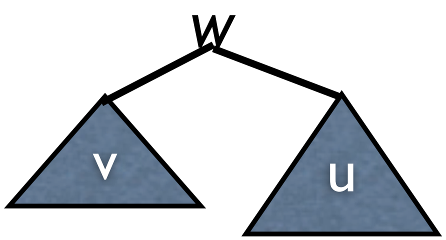
Y

t	$(f\otimes g)\Uparrow_{(1,1)}(t)$
7	6
8	7
9	7

Combining Oracle Functions

- to combine two nodes along a hyperedge, we need to distribute test brackets between the two, and optimize the number of matches

$$(f \otimes g)(t) \triangleq \max_{t_1+t_2=t} f(t_1) + g(t_2)$$



t	f(t)
2	1
3	2

$\otimes$

t	g(t)
4	4
5	4

=

t	(f⊗g)(t)
6	5
7	6
8	6

final answer:

$$F(y^+, y^*) = \max_t \frac{2 \cdot ora[\mathrm{TOP}](t)}{t + |y^*|}$$

N

t	$(f \otimes g) \Uparrow_{(1,0)} (t)$
7	5
8	6
9	6

ora[w]

this node matched?

Y

t	$(f \otimes g) \Uparrow_{(1,1)} (t)$
7	6
8	7
9	7

Forest vs. *n*-best Oracles

- forests enjoy higher oracle scores than *n*-best lists
  - a dynamic programming algorithm for forest oracle