Forest-Based Search Algorithms

for Parsing and Machine Translation

Liang Huang
University of Pennsylvania

Google Research, March 14th, 2008

Search in NLP

- is not trivial!

I saw her duck.

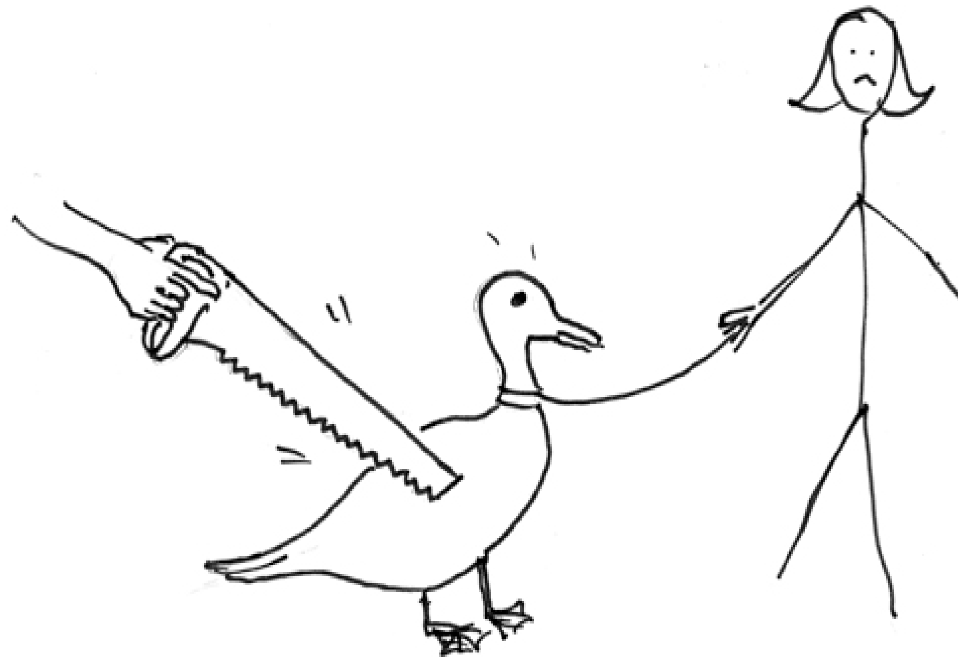
Aravind Joshi

Search in NLP

- is not trivial!

I saw her duck.

Aravind Joshi



Search in NLP

- is not trivial!

I eat sushi with tuna.

Aravind Joshi

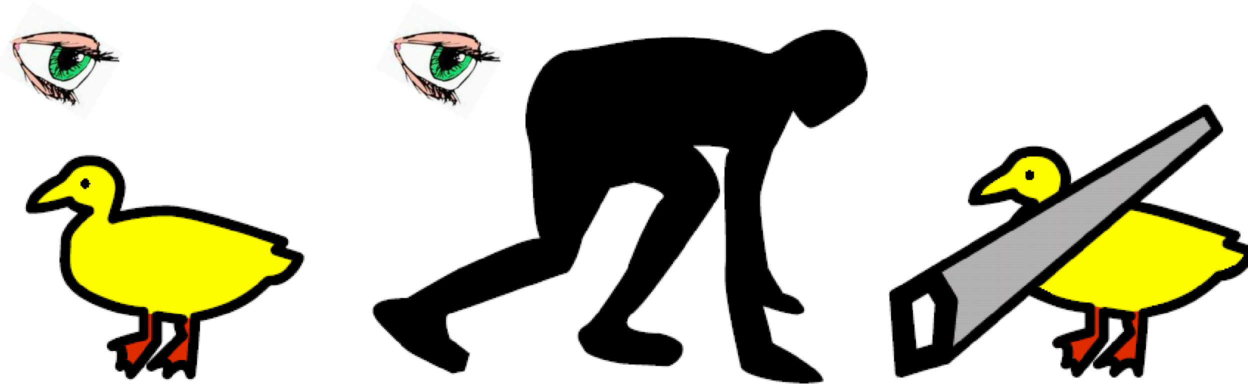
Search in NLP

- is not trivial!

I eat sushi with tuna.

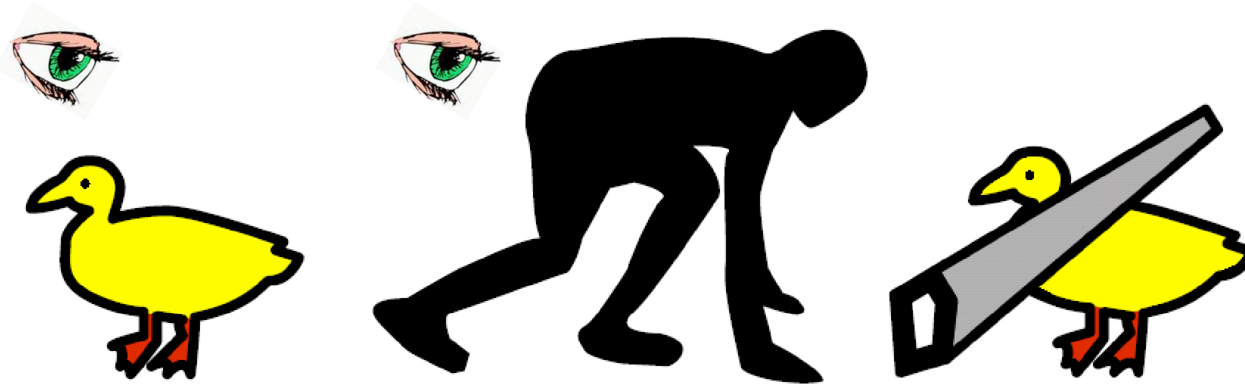
Aravind Joshi

I saw her duck.



Penn
UNIVERSITY of PENNSYLVANIA

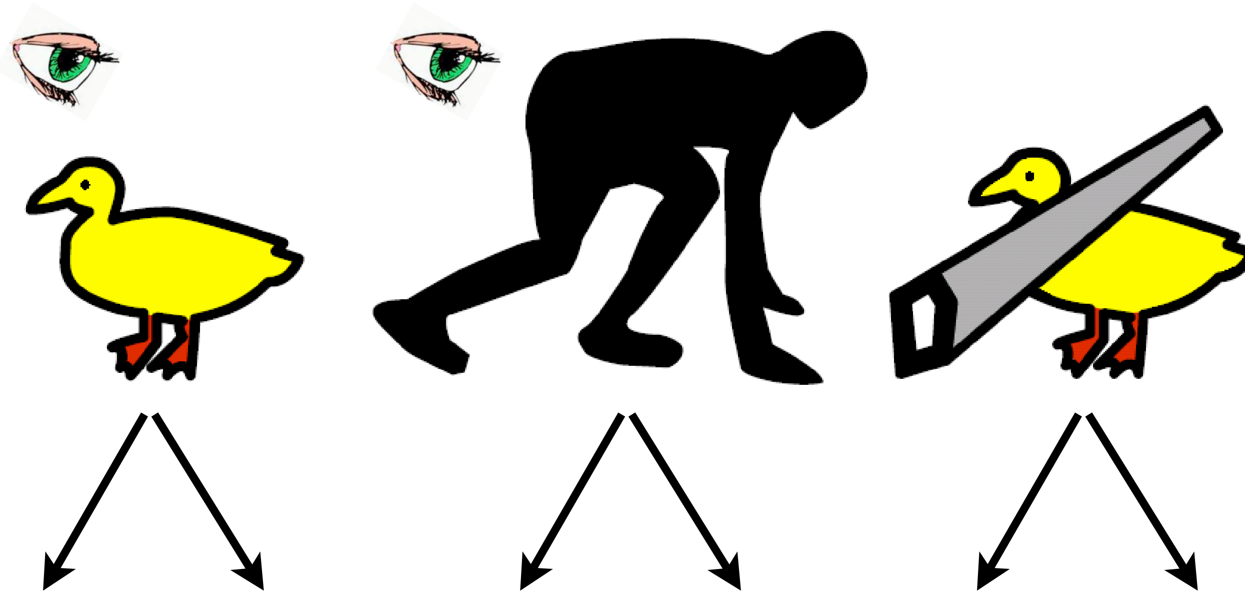
I saw her duck.



- how about...
  - I saw her duck with a telescope.

Penn
UNIVERSITY of PENNSYLVANIA

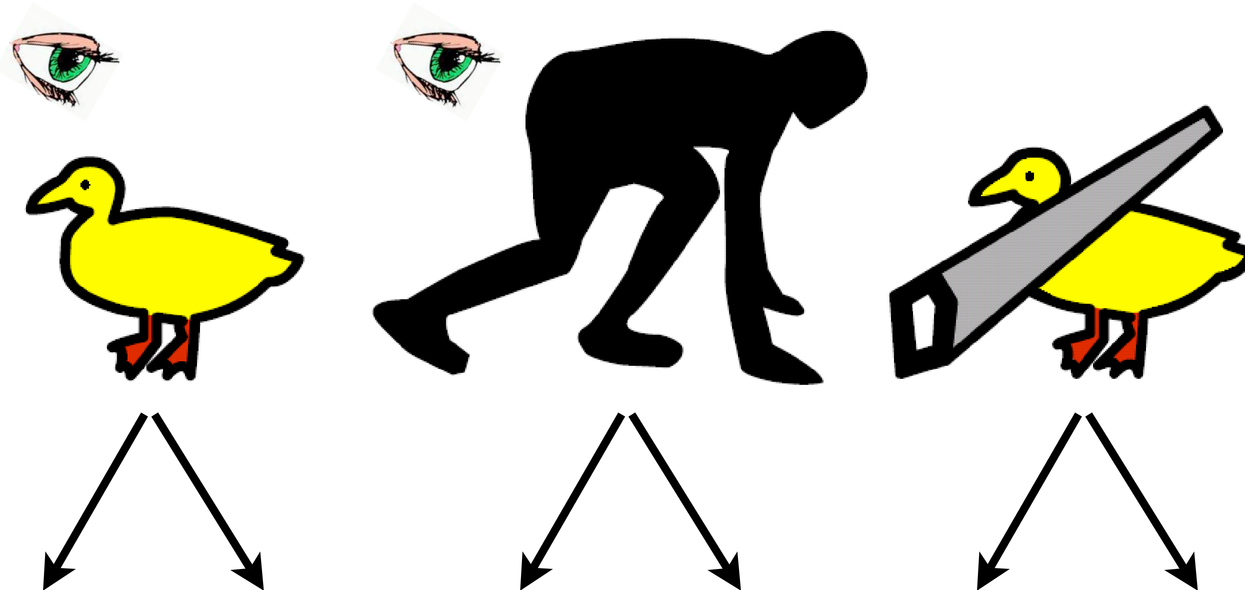
I saw her duck.



- how about...
  - I saw her duck with a telescope.

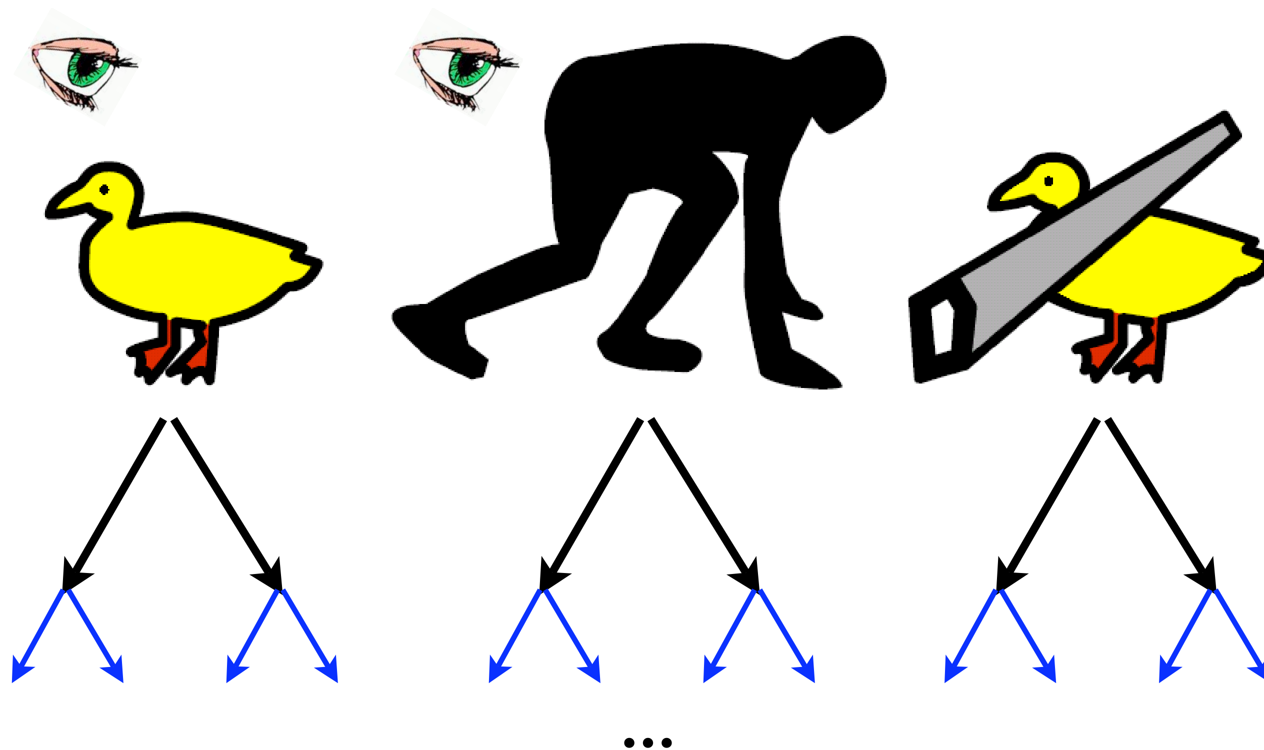
Penn University of Pennsylvania

I saw her duck.



- how about...
  - I saw her duck with a telescope.
  - I saw her duck with a telescope in the garden...

I saw her duck.



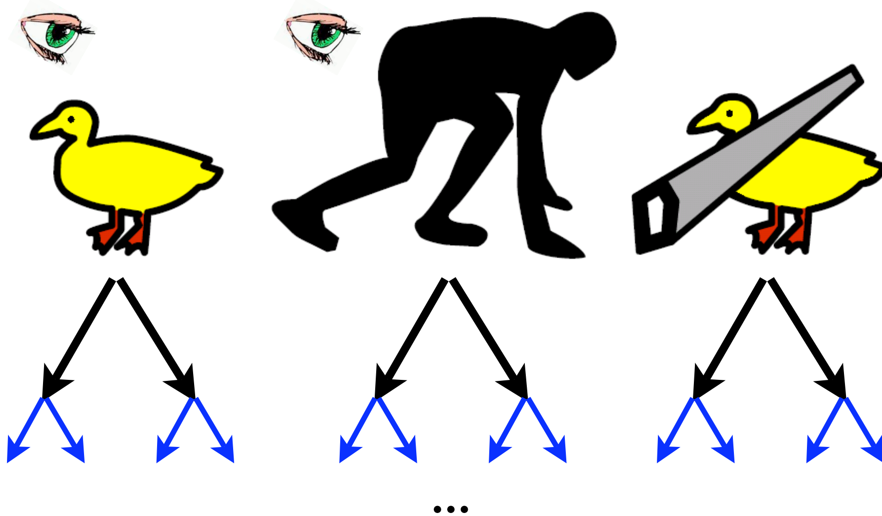
...

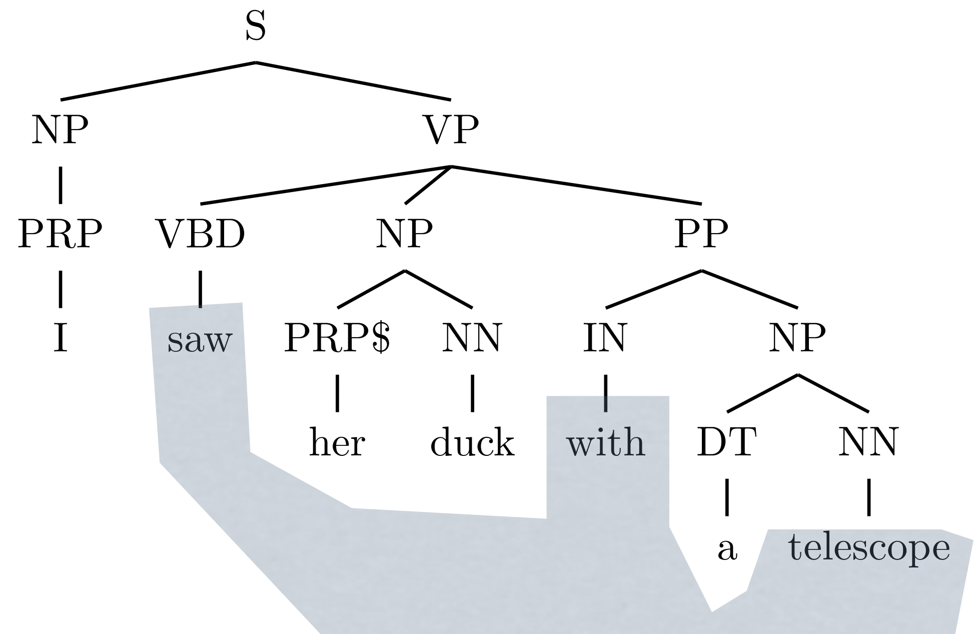
- how about...
  - I saw her duck with a telescope.
  - I saw her duck with a telescope in the garden...

Penn
University of Pennsylvania

Parsing/NLP is HARD!

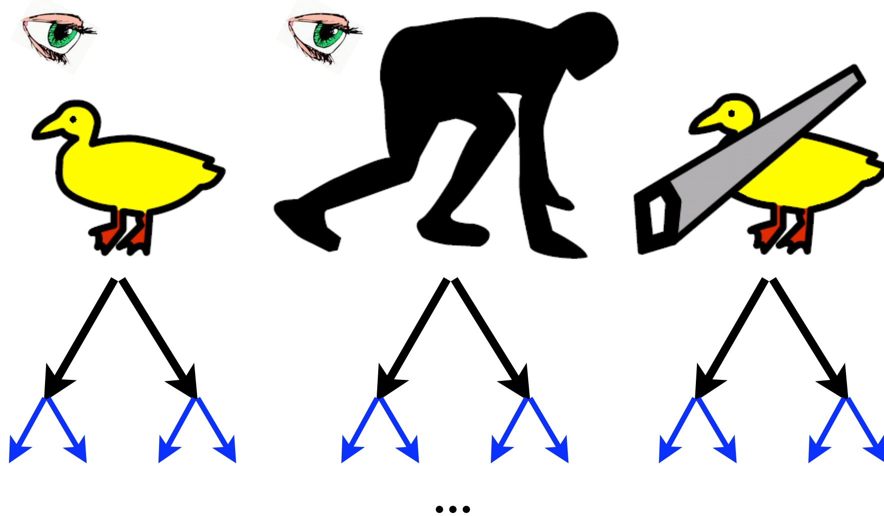
- exponential explosion of the search space
  - solution: locally factored space => packed forest
  - efficient algorithms based on dynamic programming
- non-local dependencies
  - solution: ???

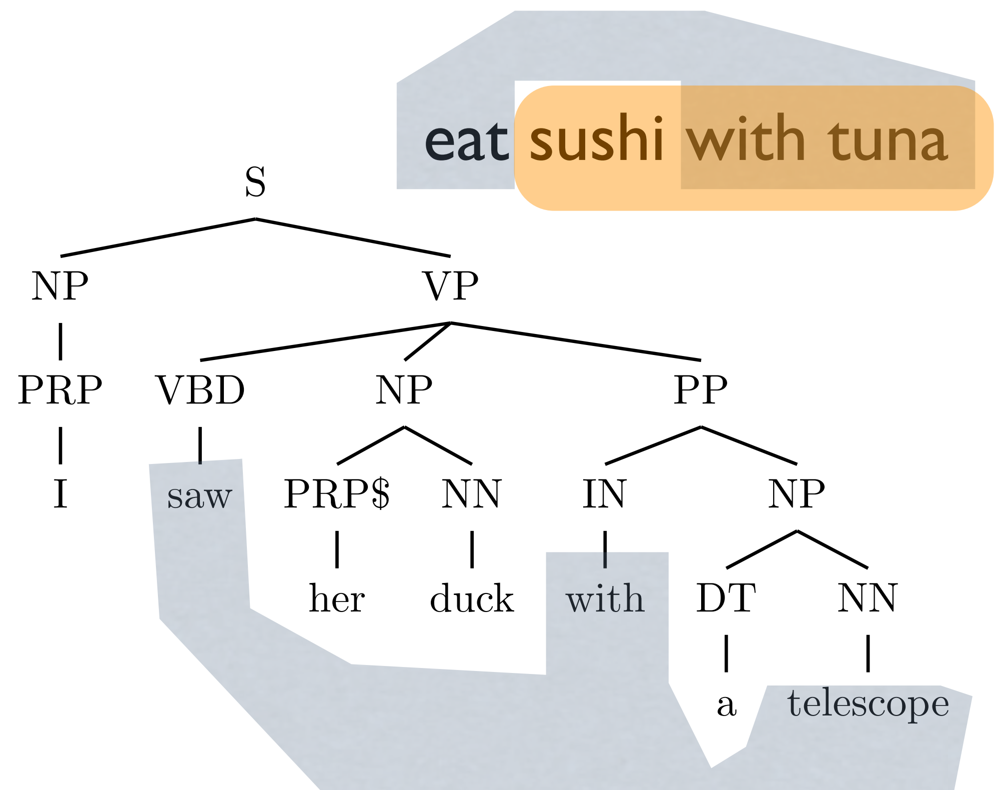




Parsing/NLP is HARD!

- exponential explosion of the search space
  - solution: locally factored space => packed forest
  - efficient algorithms based on dynamic programming
- non-local dependencies
  - solution: ???





Key Problem

Penn
UNIVERSITY of PENNSYLVANIA

Key Problem

- How to efficiently incorporate non-local information?

Key Problem

- How to efficiently incorporate non-local information?
- Solution 1: pipelined reranking / rescoring
  - postpone disambiguation by propagating *k*-best lists
  - examples: tagging => parsing => semantics
  - need very efficient algorithms for *k*-best search

Penn
University *of* Pennsylvania

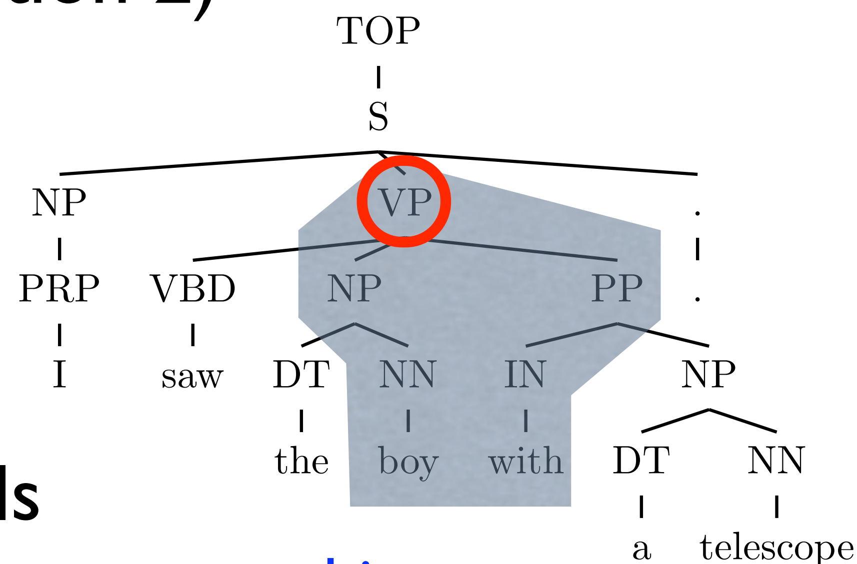
Key Problem

- How to efficiently incorporate non-local information?
- Solution 1: pipelined reranking / rescoring
  - postpone disambiguation by propagating *k*-best lists
  - examples: tagging => parsing => semantics
  - need very efficient algorithms for *k*-best search
- Solution 2: joint approximate search
  - integrate non-local information in the search
  - intractable; so only approximately
  - largely open

Penn
UNIVERSITY *of* PENNSYLVANIA

Outline

- Packed Forests and Hypergraph Framework
- Exact *k*-best Search in the Forest (for Solution 1)
- Approximate Joint Search (Solution 2) with Non-Local Features
  - Forest Reranking
- Machine Translation
  - Decoding w/ Language Models
  - Forest Rescoring
- Future Directions

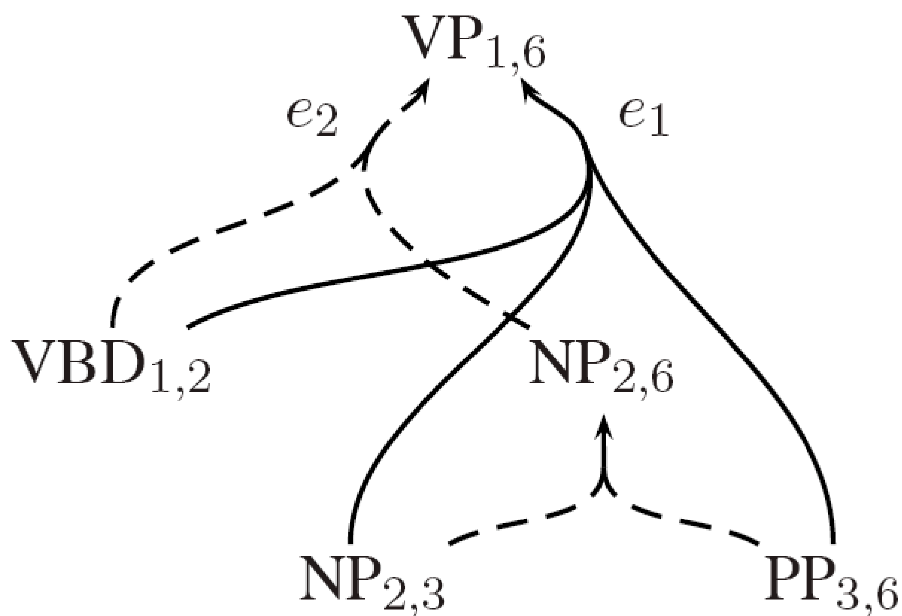


bigram

Packed Forests and Hypergraph Framework

Packed Forests

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



$$e_1 \quad \frac{\mathrm{VBD}_{1,2} \quad \mathrm{NP}_{2,3} \quad \mathrm{PP}_{3,6}}{\mathrm{VP}_{1,6}}$$

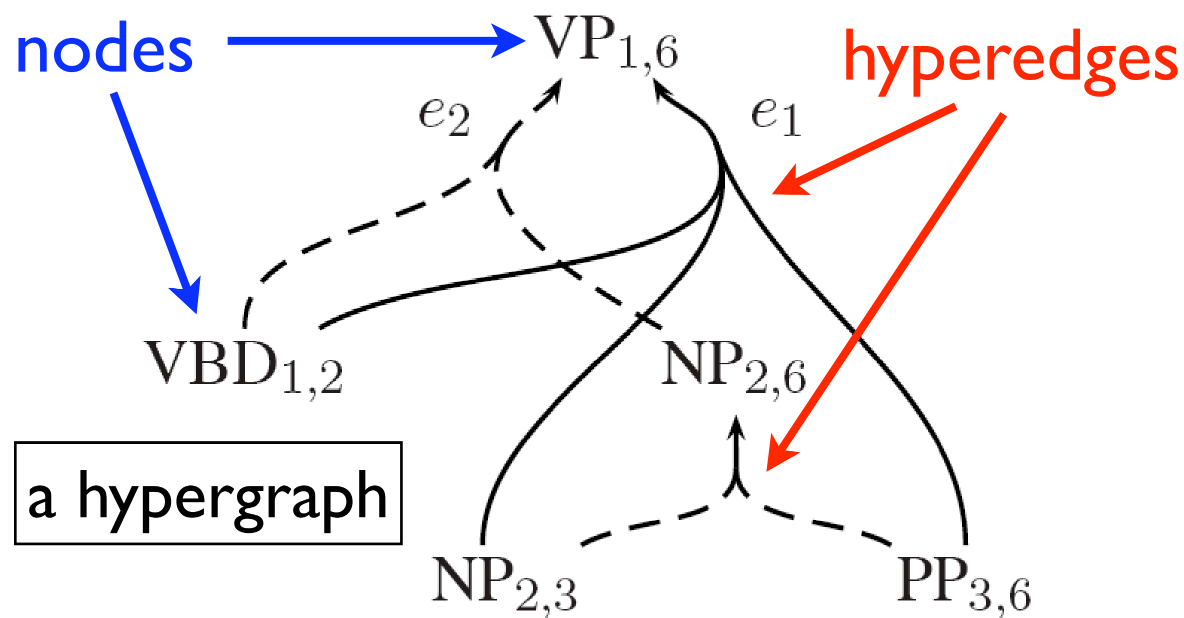
$_0$ I $_1$ saw $_2$ him $_3$ with $_4$ a $_5$ mirror $_6$

Penn University of Pennsylvania

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

Packed Forests

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



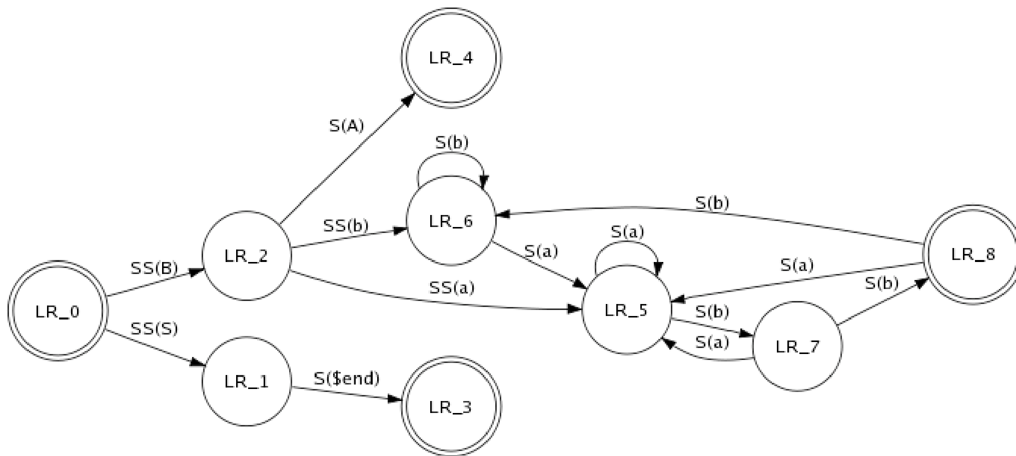
$$e_1 \quad \frac{\mathrm{VBD}_{1,2} \quad \mathrm{NP}_{2,3} \quad \mathrm{PP}_{3,6}}{\mathrm{VP}_{1,6}}$$

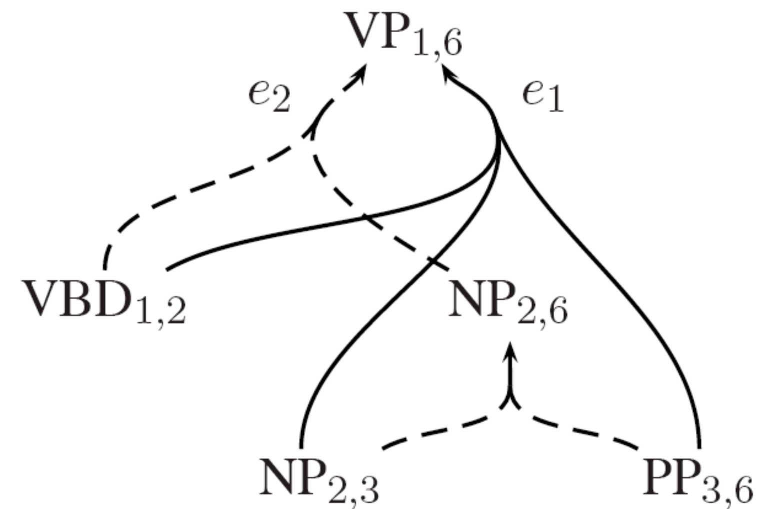
$_0$ I $_1$ saw $_2$ him $_3$ with $_4$ a $_5$ mirror $_6$

(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

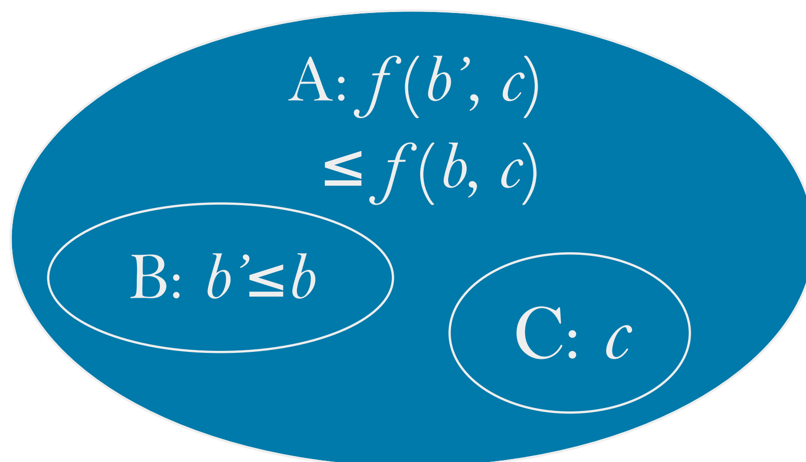


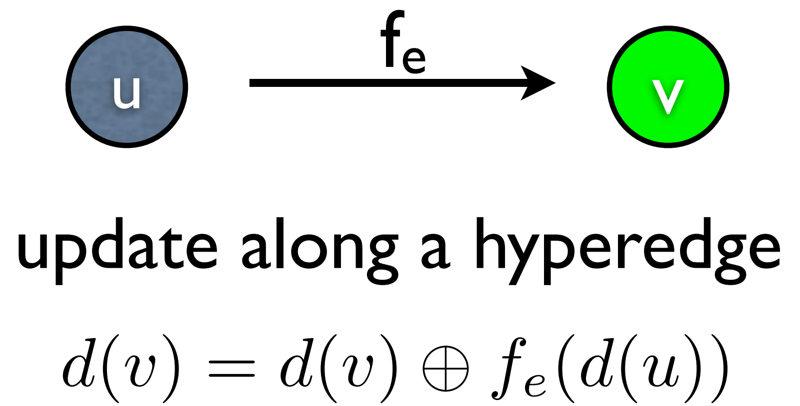


Penn
University of Pennsylvania

Weight Functions

- Each hyperedge e has a weight function f_e
  - monotonic in each argument
  - e.g. in CKY, $f_e(a, b) = a \times b \times \Pr(\text{rule})$
- optimal subproblem property in dynamic programming
  - optimal solutions include optimal sub-solutions

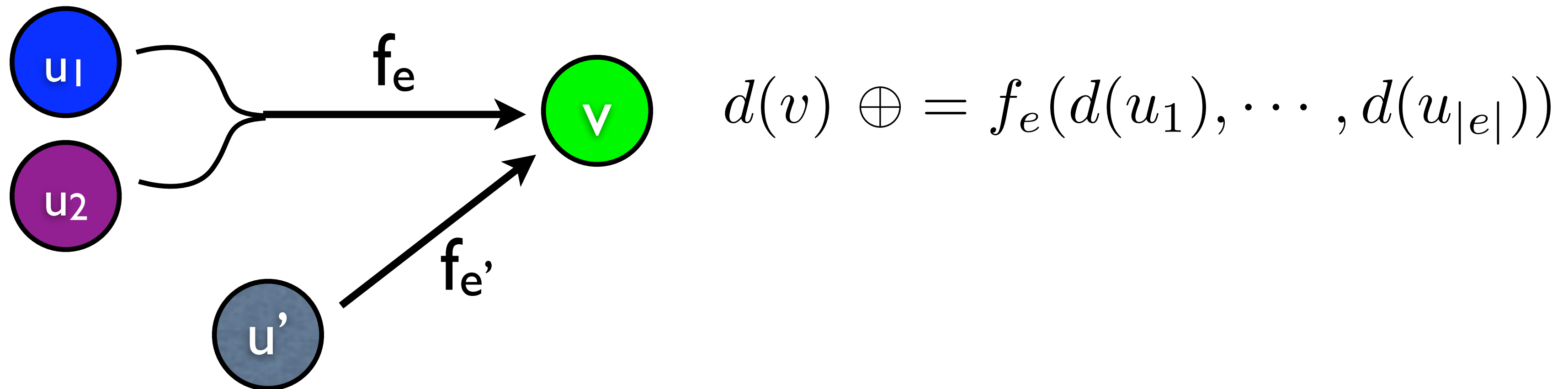




Penn
University of Pennsylvania

Generalized Viterbi Algorithm

1. topological sort (assumes acyclicity)
2. visit each node v in sorted order and do updates
   - for each incoming hyperedge $e = ((u_1, .., u_{|e|}), v, f_e)$
   - use $d(u_i)$'s to update $d(v)$
   - key observation: $d(u_i)$'s are fixed to optimal at this time

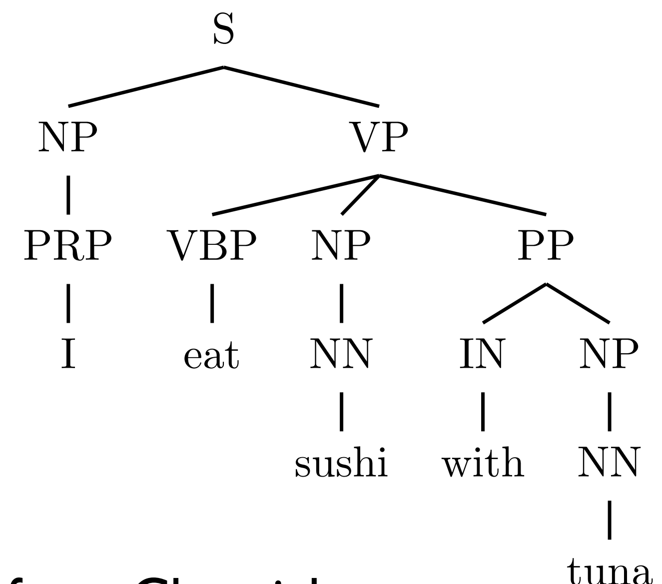


- time complexity: O(V+E) = O(E) for CKY: $O(n^3)$

1-best => *k*-best

- we need *k*-best for pipelined reranking / rescoring
  - since 1-best is not guaranteed to be correct
  - rerank *k*-best list with non-local features
  - we need fast algorithms for very big values of *k*

I eat sushi with tuna.

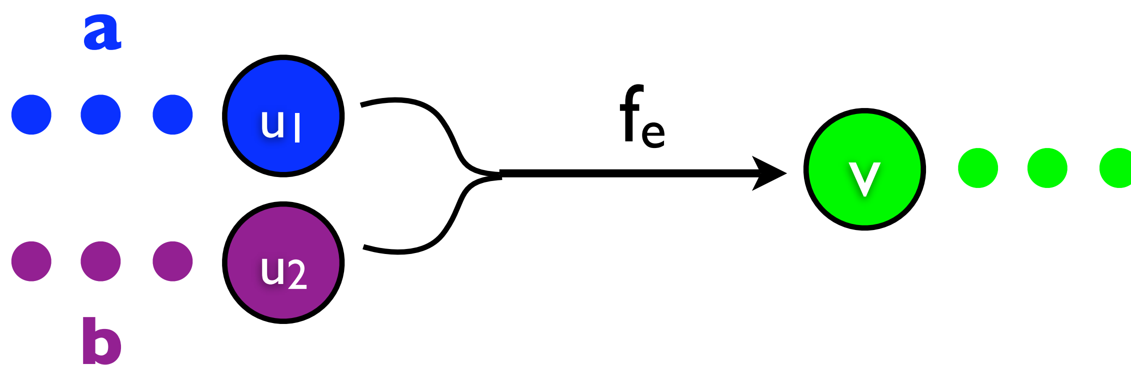


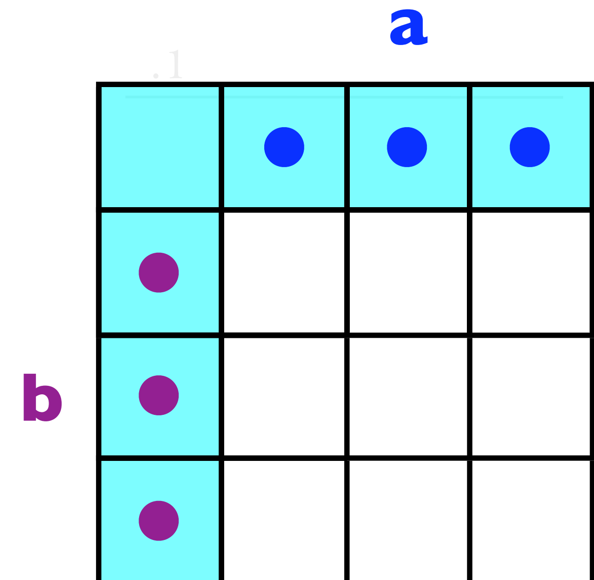
1-best from Charniak parser

Penn
University of Pennsylvania

k-best Viterbi Algorithm 0

- straightforward k-best extension
  - a vector of k (sorted) values for each node
  - now what's the result of f_e (**a**, **b**) ?
    - $k \times k = k^2$ possibilities! => then choose top k

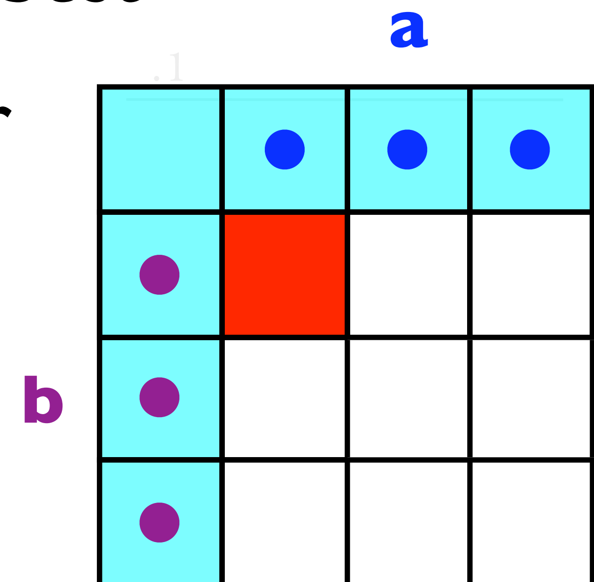




- time complexity: $O(k^2 E)$

k-best Viterbi Algorithm 1

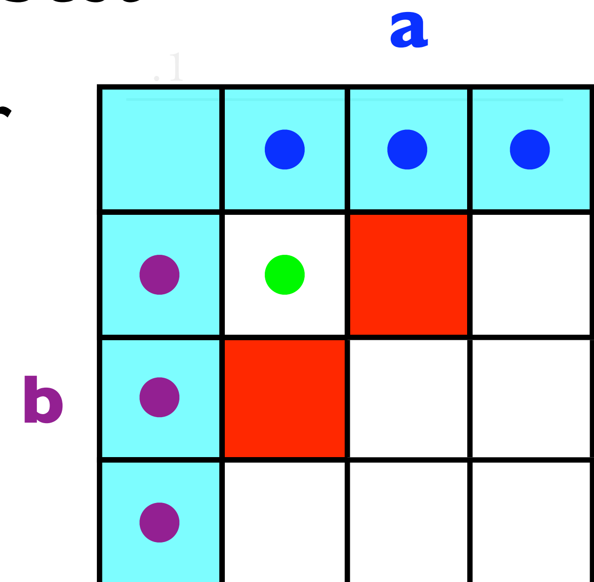
- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: O($k \log k$ E)



Penn
University of Pennsylvania

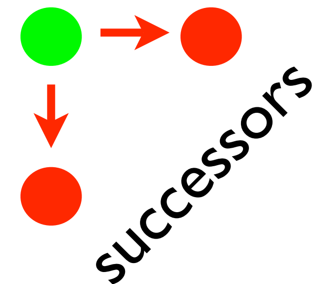
k-best Viterbi Algorithm 1

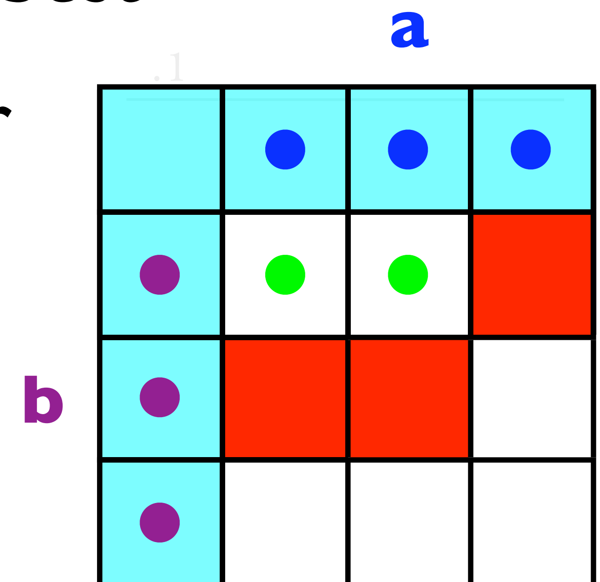
- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: $O(k \log k\ E)$



k-best Viterbi Algorithm 1

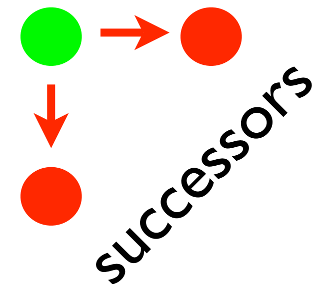
- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: O(*k* log *k* E)

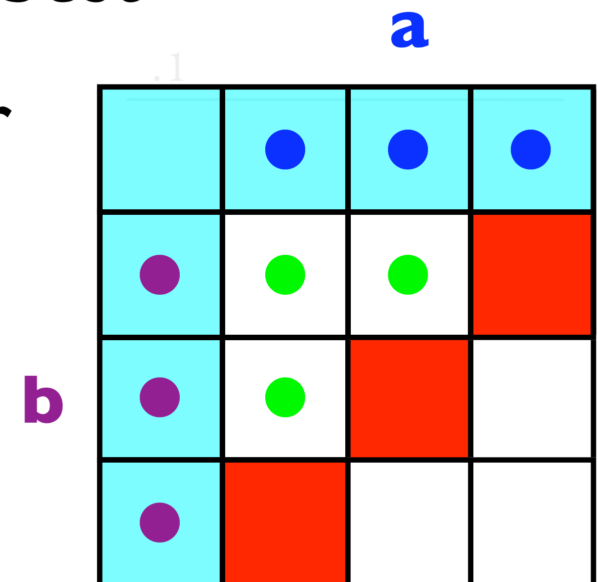




k-best Viterbi Algorithm 1

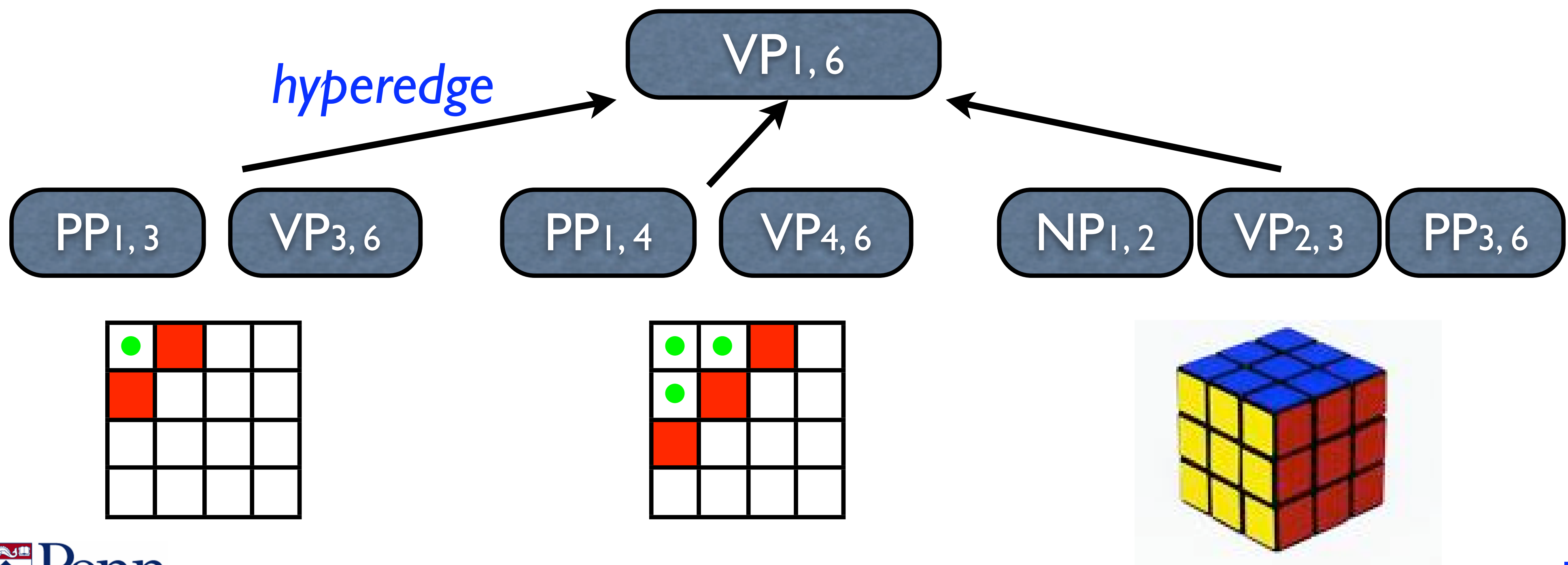
- key insight: do not need to enumerate all k^2
  - since vectors **a** and **b** are sorted
  - and the weight function f_e is monotonic
- (a_1, b_1) must be the best
  - either (a_2, b_1) or (a_1, b_2) is the 2nd-best
- use a priority queue for the frontier
  - extract best
  - push two successors
- time complexity: $O(k \log k\ E)$





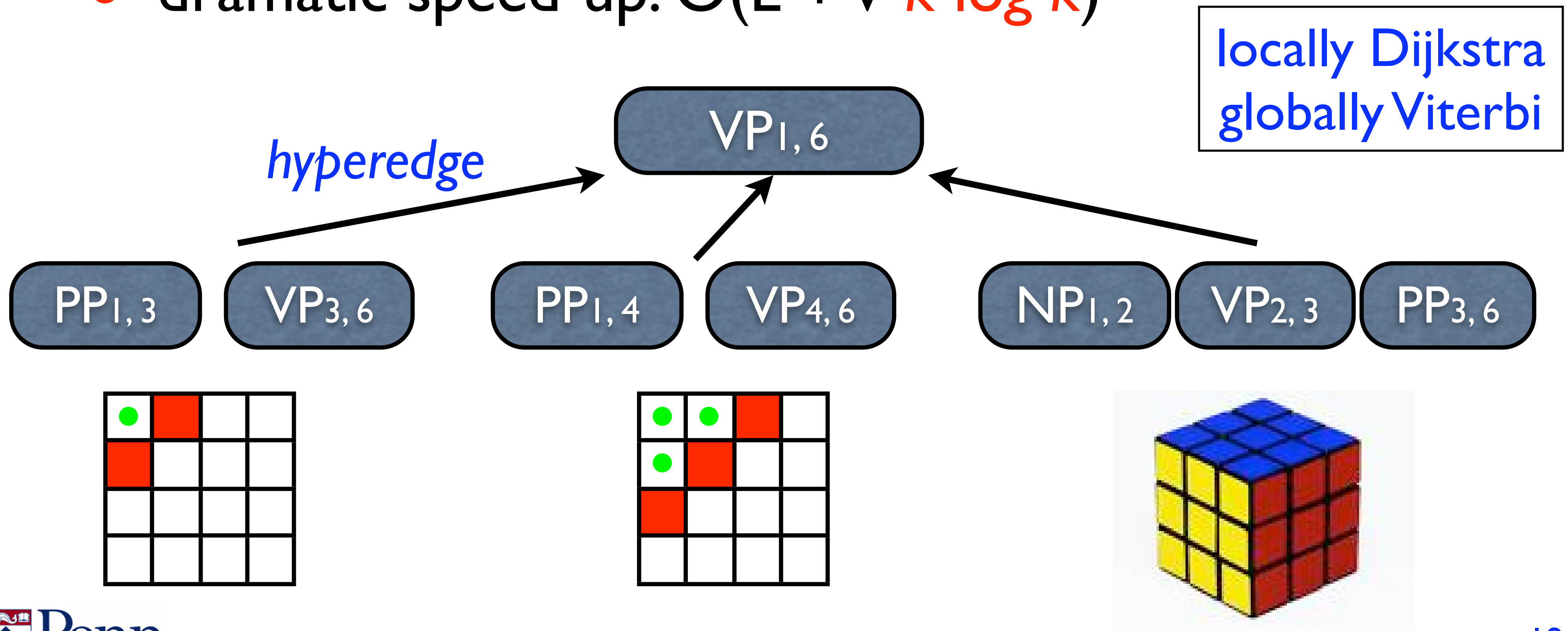
k-best Viterbi Algorithm 2

- Algorithm 1 works on each hyperedge sequentially
  - O(*k* log *k* E) is still too slow for big *k*
- Algorithm 2 processes all hyperedges in parallel
  - dramatic speed-up: O(E + V *k* log *k*)



k-best Viterbi Algorithm 2

- Algorithm 1 works on each hyperedge sequentially
  - O(*k* log *k* E) is still too slow for big *k*
- Algorithm 2 processes all hyperedges in parallel
  - dramatic speed-up: O(E + V *k* log *k*)



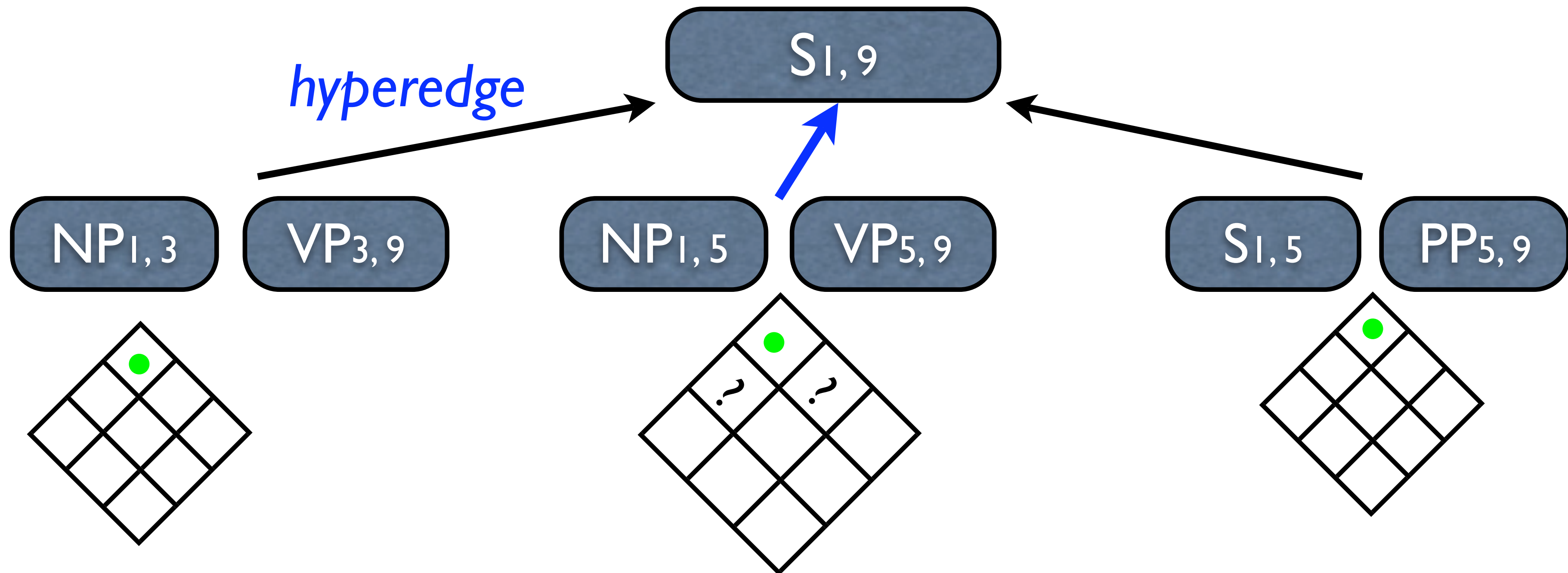
k-best Viterbi Algorithm 3

- Algorithm 2 computes k-best for each node
  - but we are only interested in k-best of the root node
- Algorithm 3 computes as many as really needed
  - forward-phase
    - same as 1-best Viterbi, but stores the forest (keeping alternative hyperedges)
  - backward-phase
    - recursively asking "what's your 2nd-best" top-down
    - asks for more when need more

Penn UNIVERSITY of PENNSYLVANIA

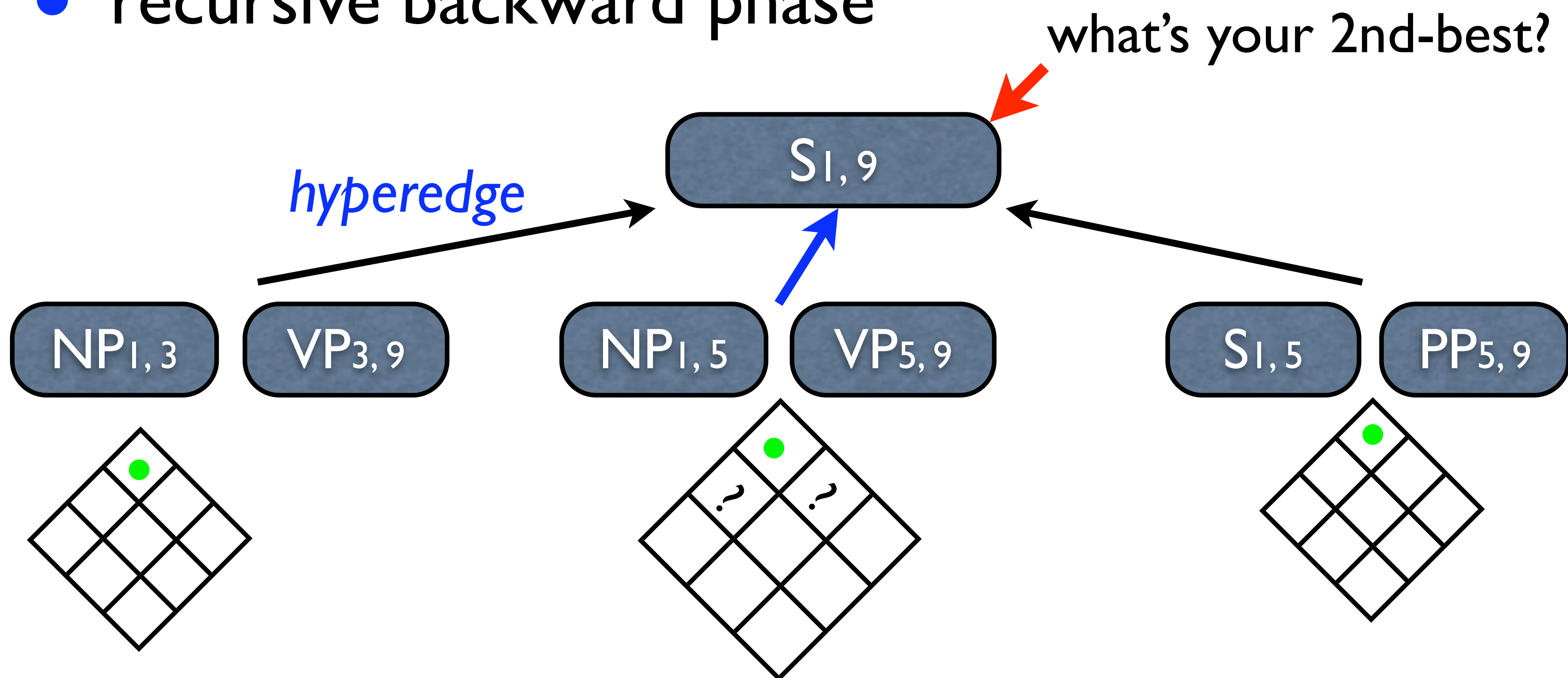
k-best Viterbi Algorithm 3

- only 1-best is known after the forward phase
- recursive backward phase



k-best Viterbi Algorithm 3

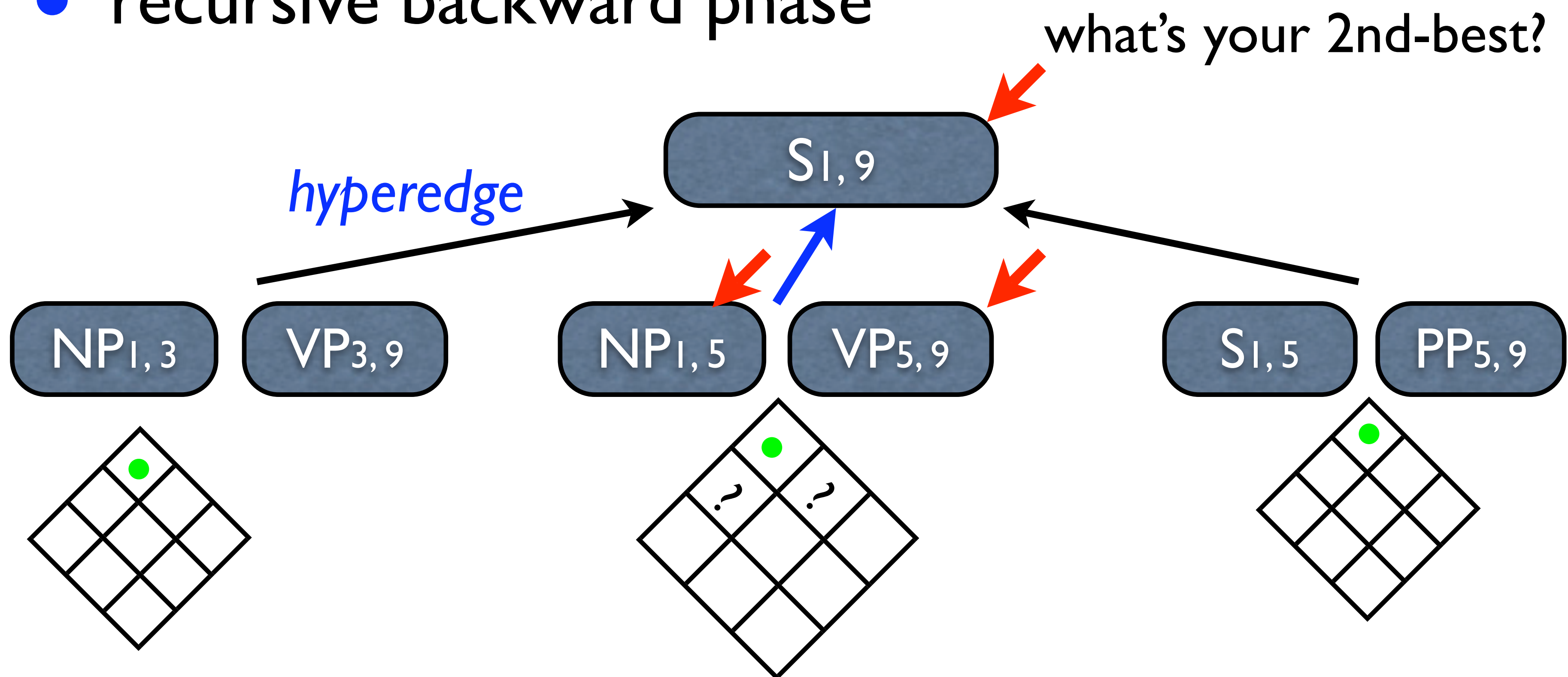
- only 1-best is known after the forward phase
- recursive backward phase



Penn
University of Pennsylvania

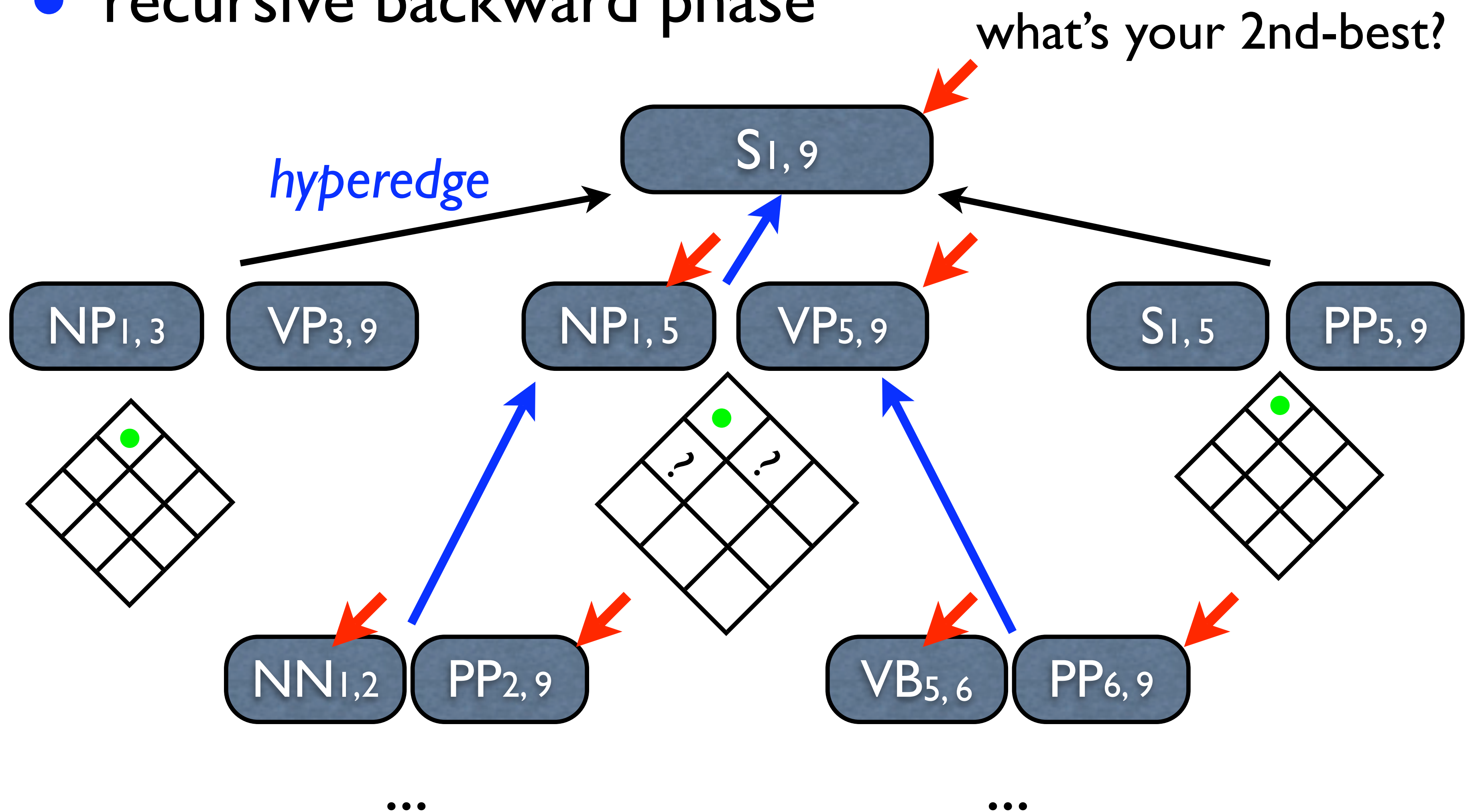
k-best Viterbi Algorithm 3

- only 1-best is known after the forward phase
- recursive backward phase



k-best Viterbi Algorithm 3

- only 1-best is known after the forward phase
- recursive backward phase



Summary of Algorithms

- Algorithms 1 => 2 => 3
  - lazier and lazier (computation on demand)
  - larger and larger locality
  - Algorithm 3 is very fast, but requires storing forest

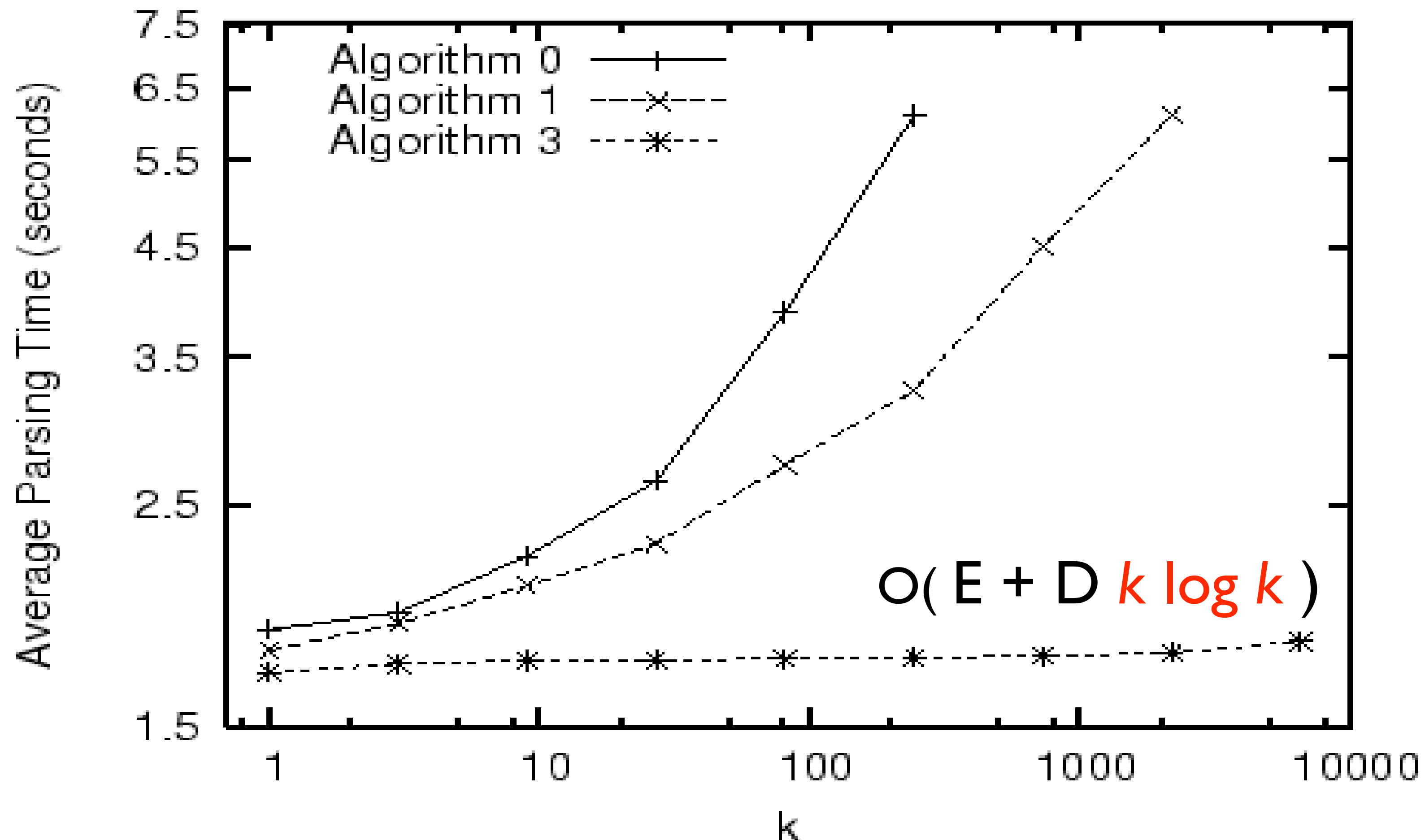
	locality	time	space
Algorithm 1	hyperedge	$O(E\, k \log k)$	$O(kV)$
Algorithm 2	node	$O(E + V\, k \log k)$	$O(kV)$
Algorithm 3	global	$O(E + D\, k \log k)$	$O(E + k D)$

E - hyperedges: $O(n^3)$; V - nodes: $O(n^2)$; D - derivation: $O(n)$

Penn
University of Pennsylvania

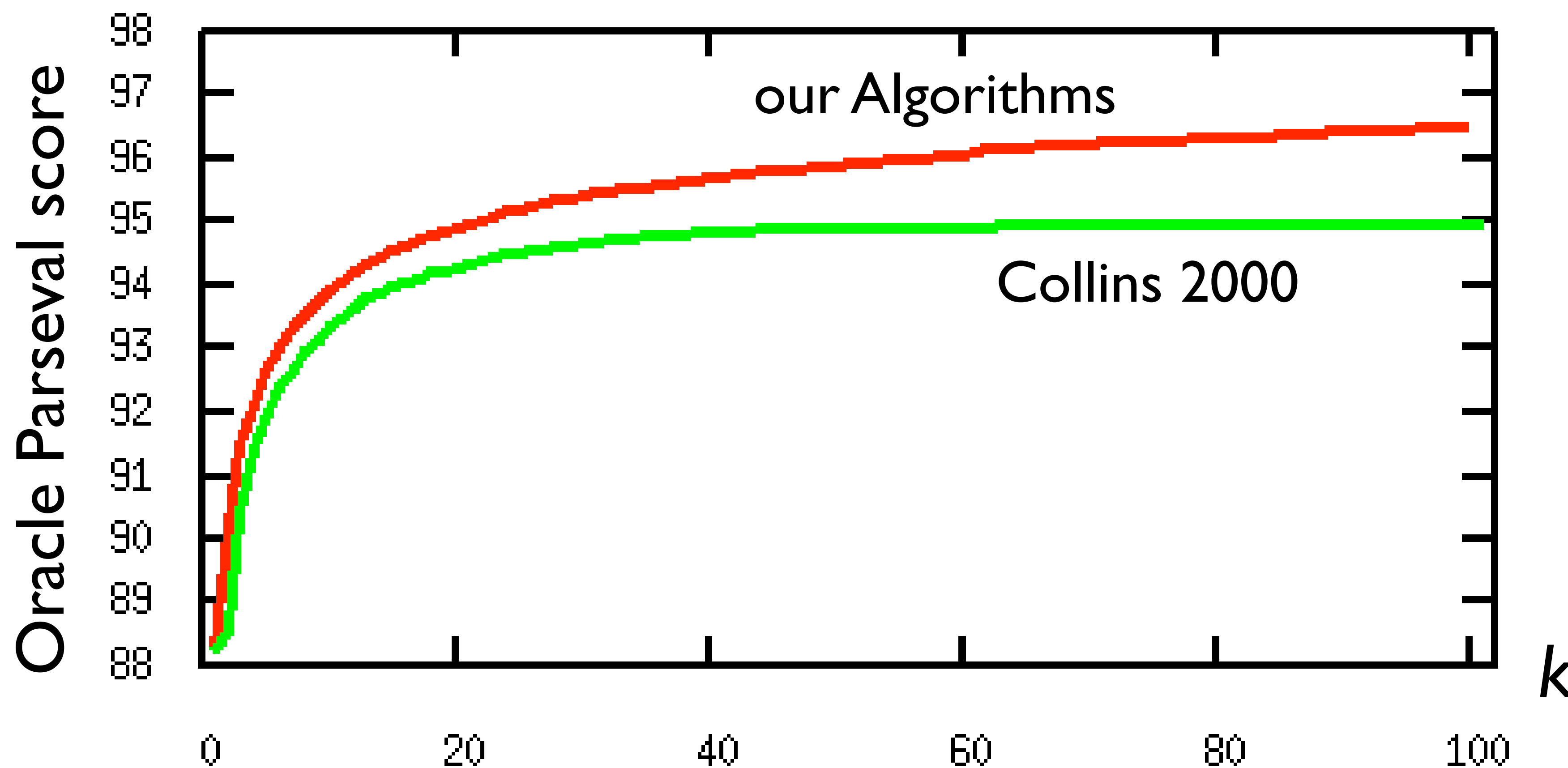
Experiments - Efficiency

- on state-of-the-art Collins/Bikel parser (Bikel, 2004)
- average parsing time per sentence using Algs. 0, 1, 3



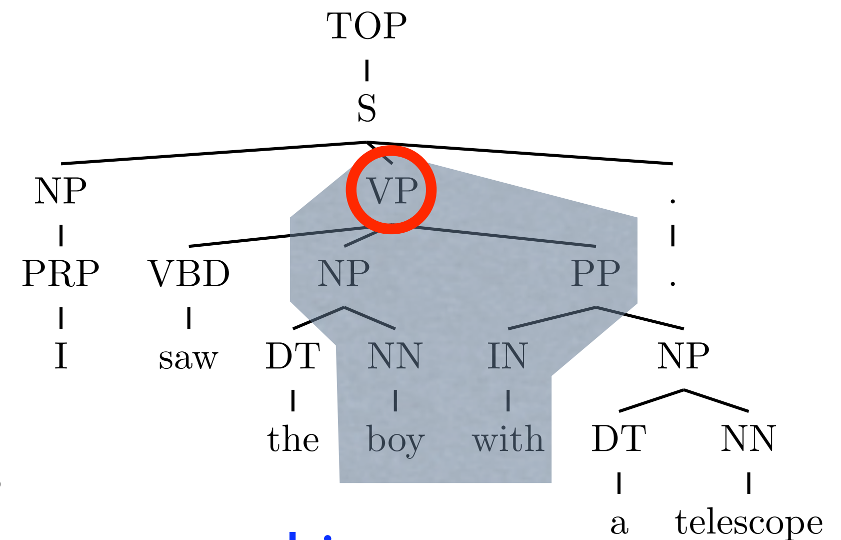
Reranking and Oracles

- oracle - the candidate closest to the correct parse among the *k*-best candidates
- measures the potential of real reranking

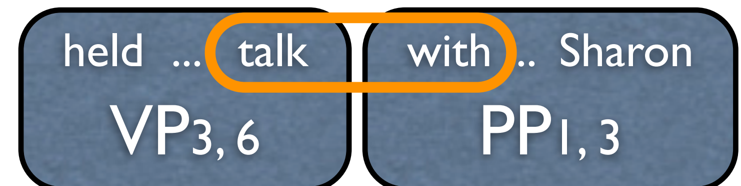


Outline

- Packed Forests and Hypergraph Framework
- Exact k-best Search in the Forest (Solution 1)
- Approximate Joint Search (Solution 2) with Non-Local Features
  - Forest Reranking
- Machine Translation
  - Decoding w/ Language Models
  - Forest Rescoring
- Future Directions



bigram



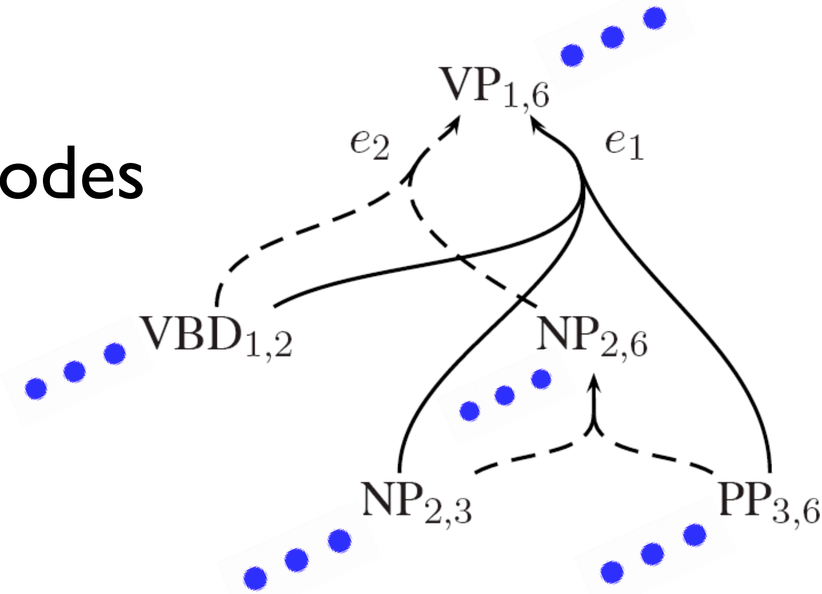
Penn
University of Pennsylvania

Why *n*-best reranking is bad?

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

Reranking on a Forest?

- with only local features
  - dynamic programming, tractable (Taskar et al. 2004; McDonald et al., 2005)
- with non-local features
  - on-the-fly reranking at internal nodes
  - top *k* derivations at each node
  - use as many non-local features as possible at each node
  - chart parsing + discriminative reranking
- we use perceptron for simplicity



Penn
University of Pennsylvania

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** $t \leftarrow 1 \ldots T$ **do** ▷ T iterations

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

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

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"

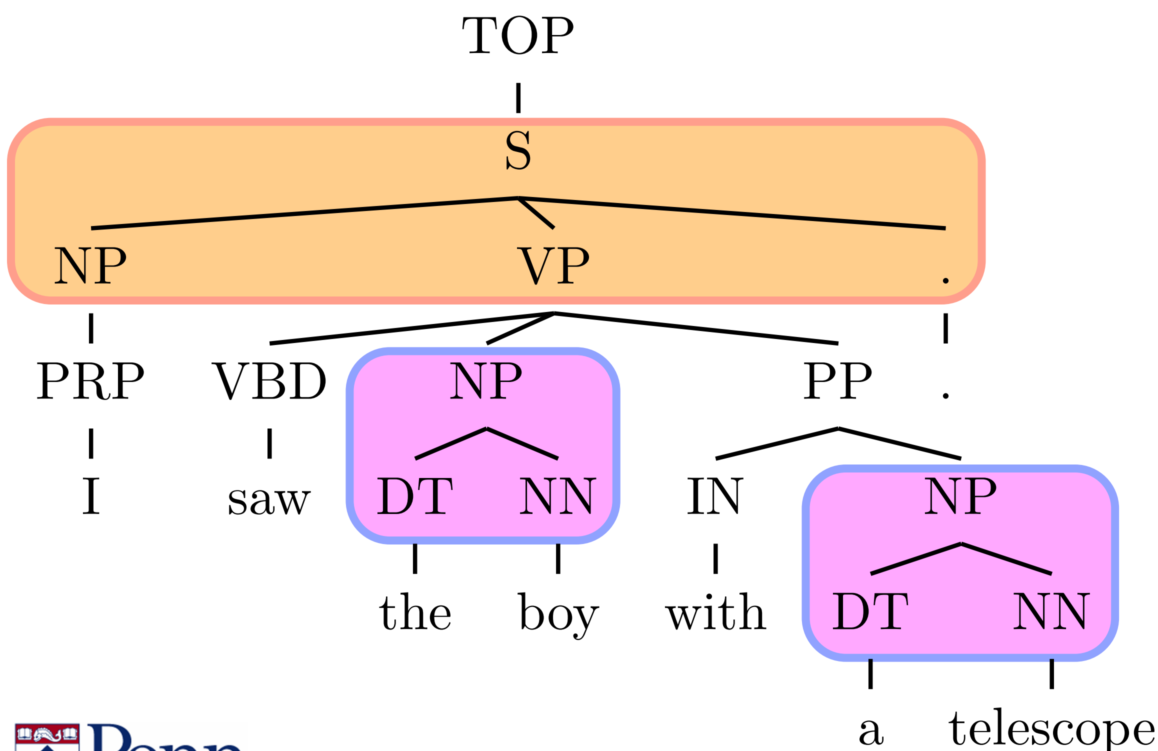
feature representation

Penn University of Pennsylvania

(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)

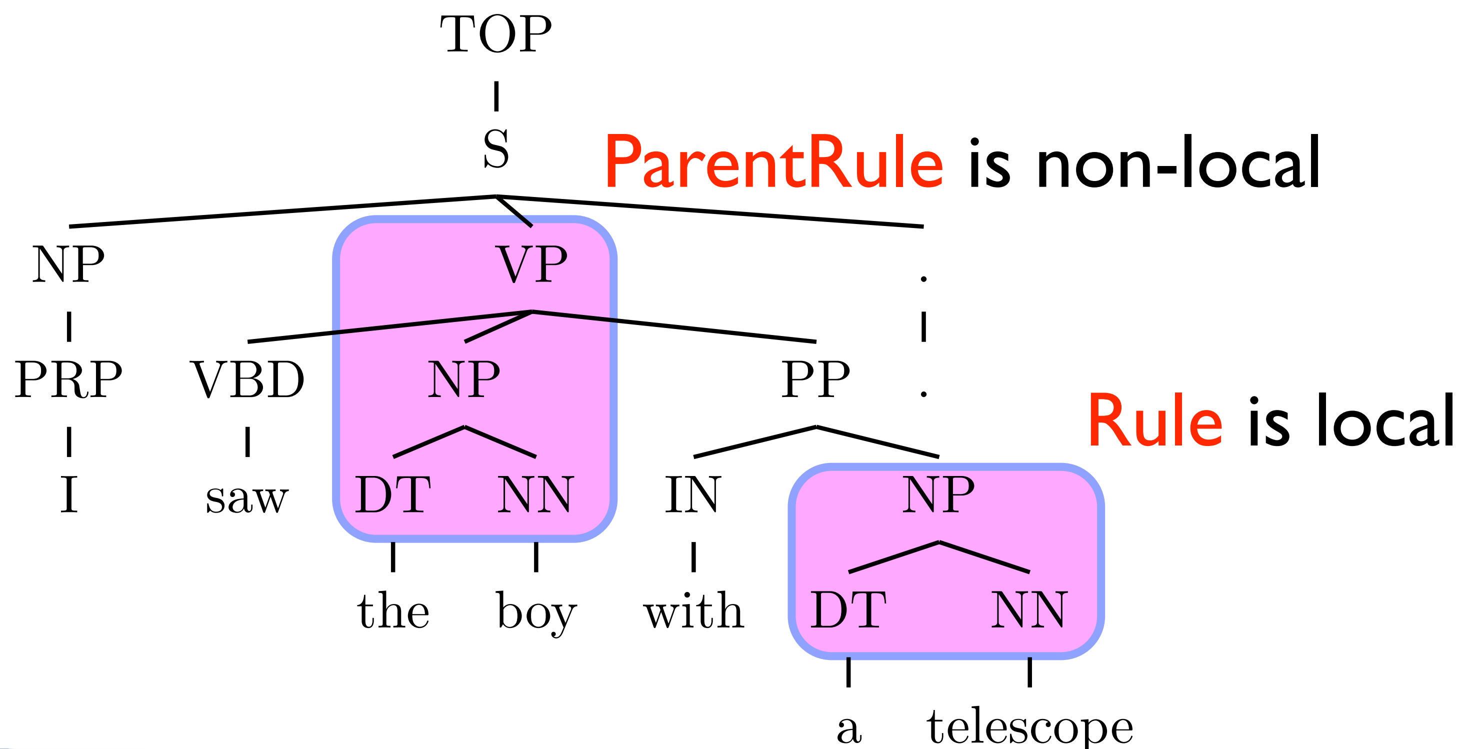
instances of Rule feature

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

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

Local vs. Non-Local Features

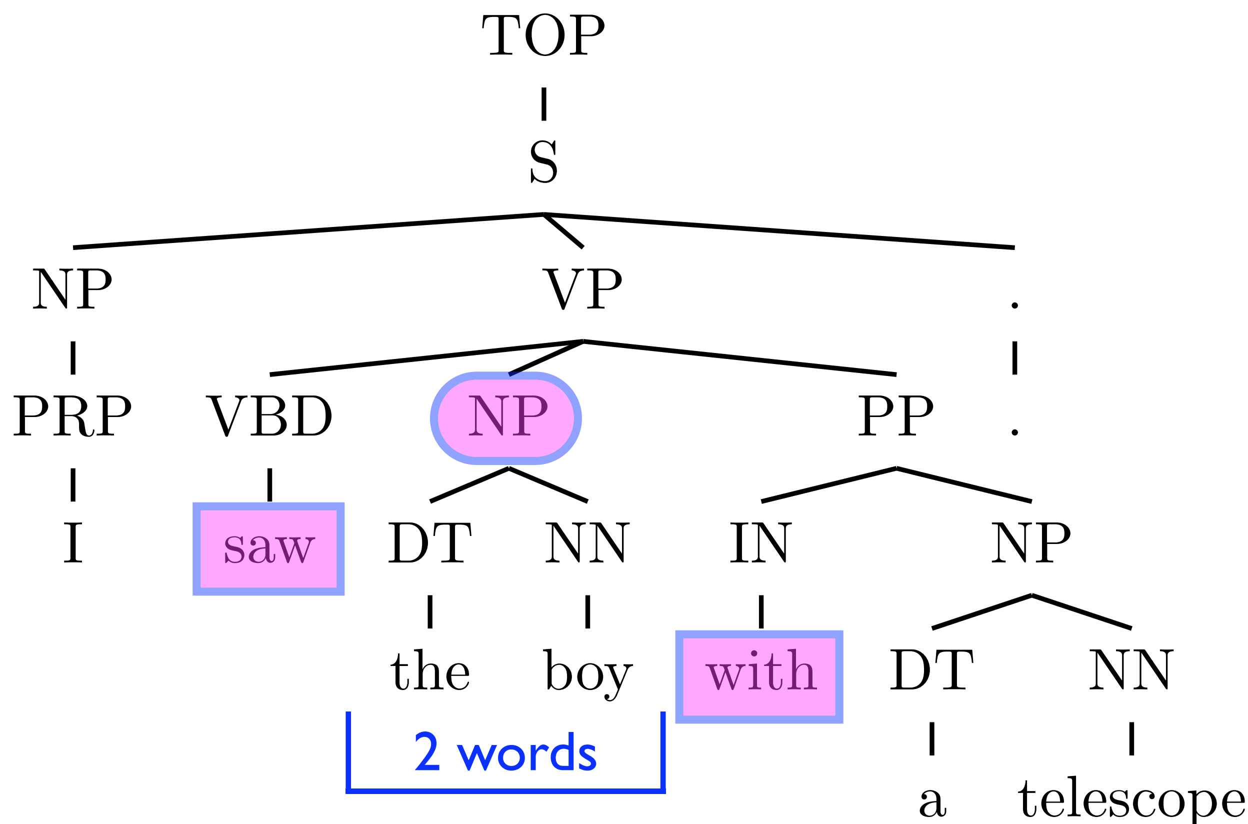
- a feature is local iff. it can be factored among local productions of a tree (i.e., hyperedges in a forest)
- local features can be pre-computed on each hyperedge in the forest; non-locals can not



Penn
University of Pennsylvania

WordEdges (C&J 05)

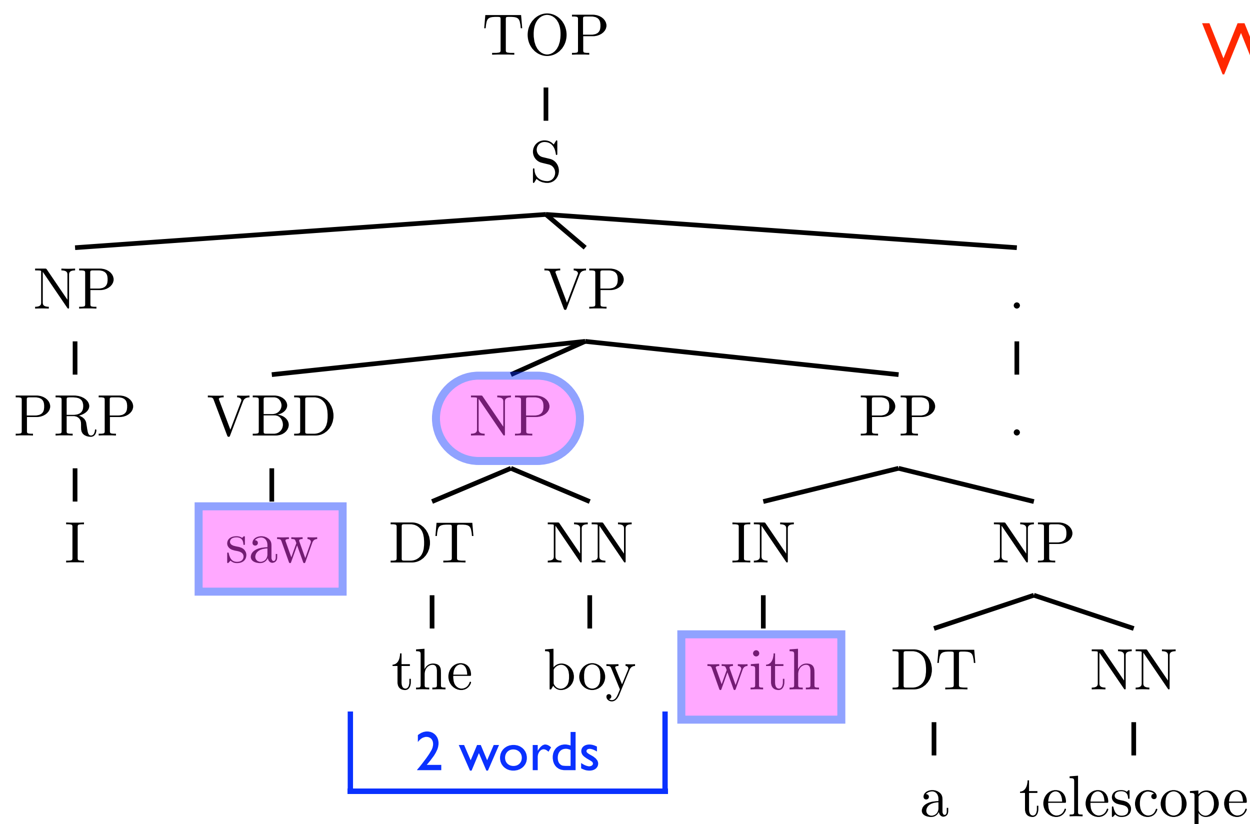
- a WordEdges feature classifies a node by its label, (binned) span length, and surrounding words
- a POSEdges feature uses surrounding POS tags



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

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

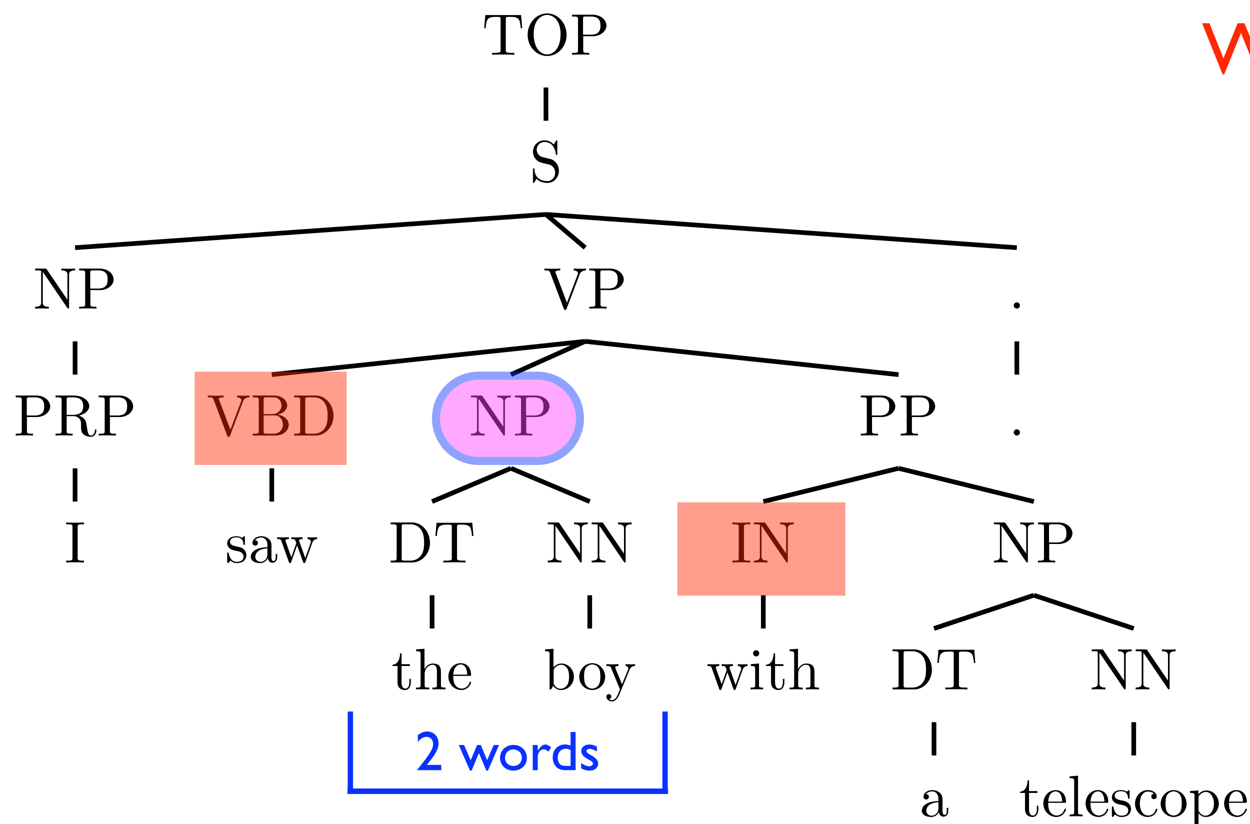


WordEdges is local

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

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



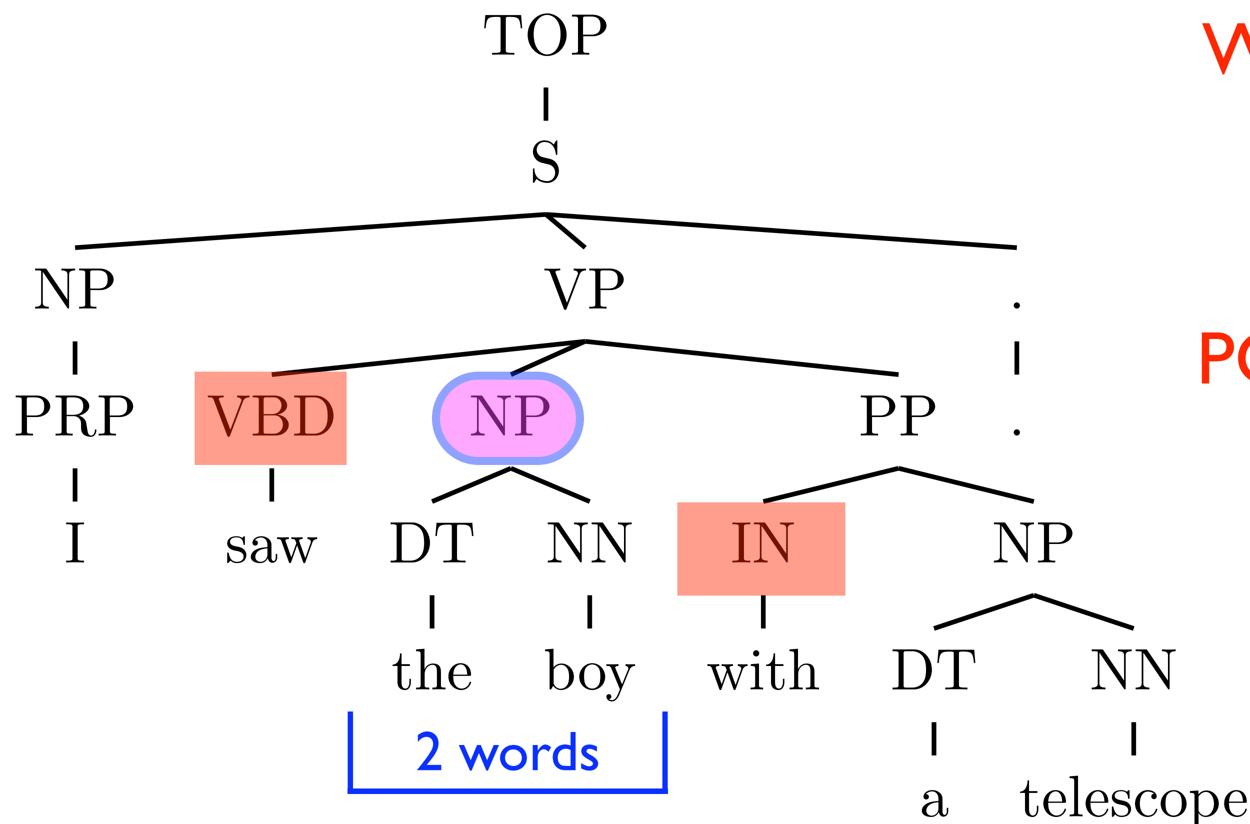
WordEdges is local

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

Penn
UNIVERSITY of PENNSYLVANIA

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

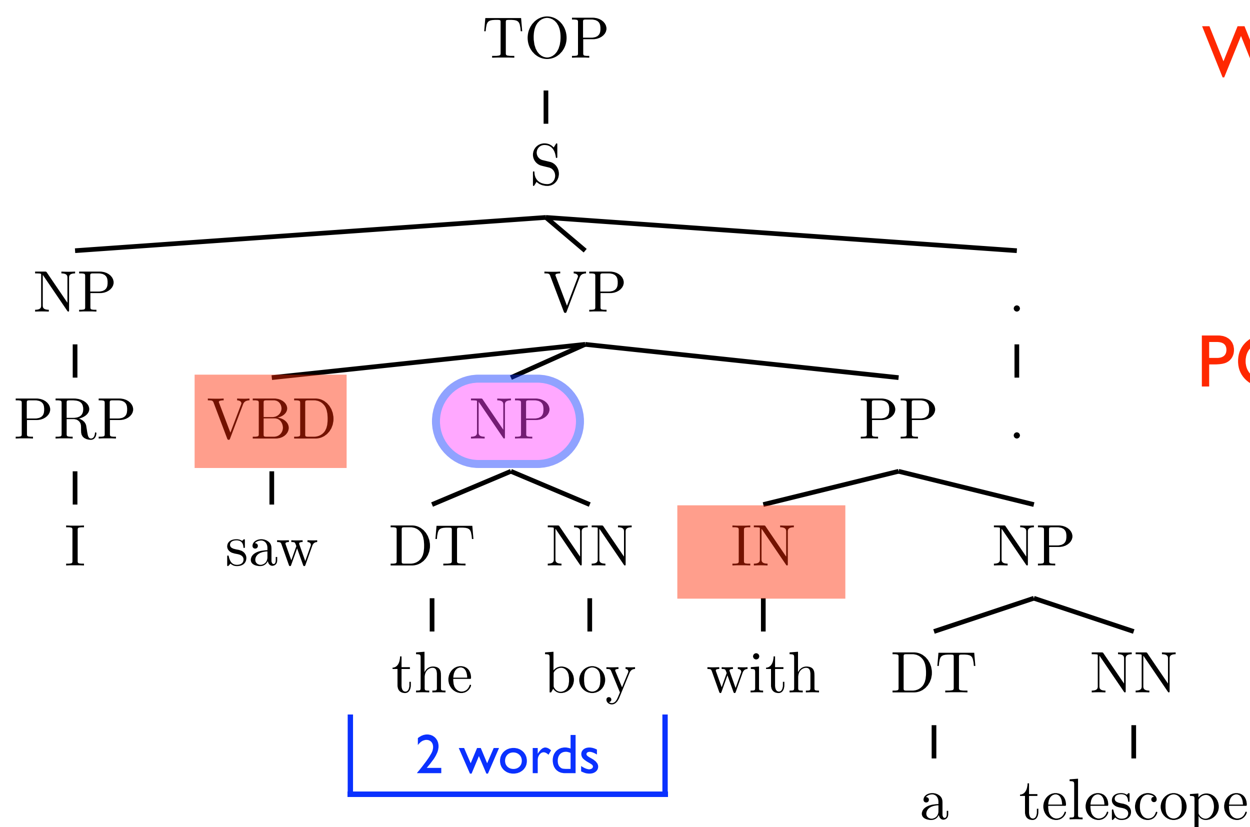
$$f_{400}(y) = f_{\text{NP 2 saw with}}(y) = 1$$

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

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

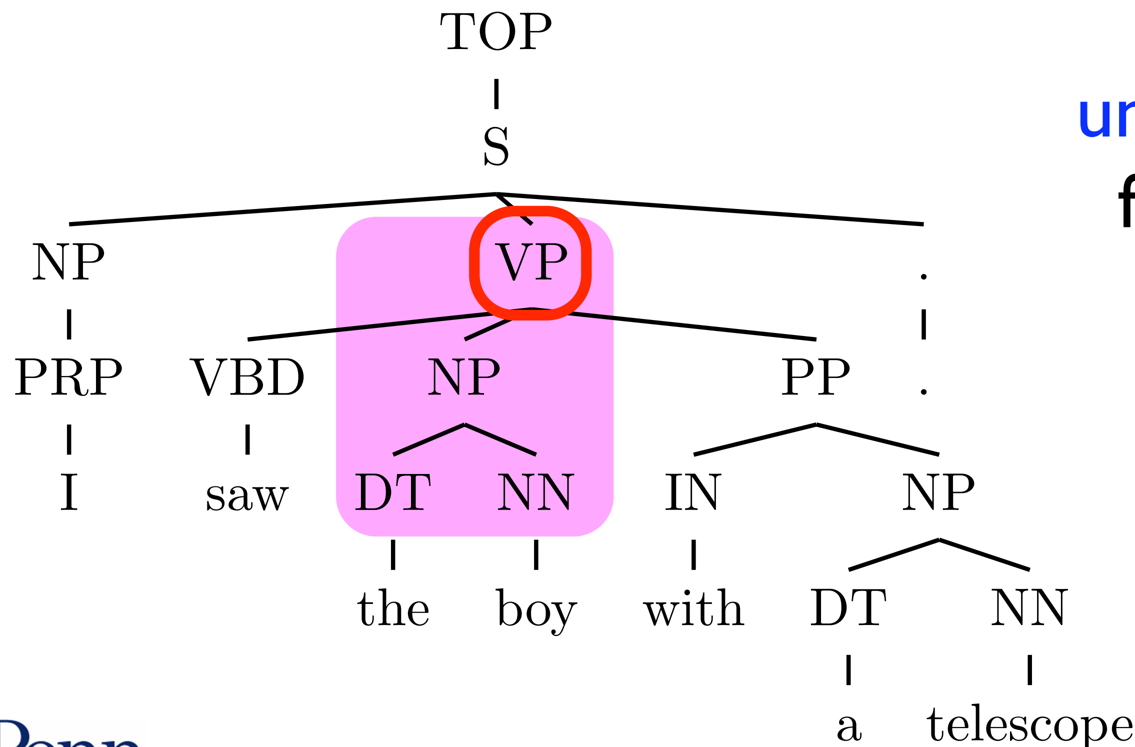
POSEdges is non-local

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

local features comprise ~70% of all instances!

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

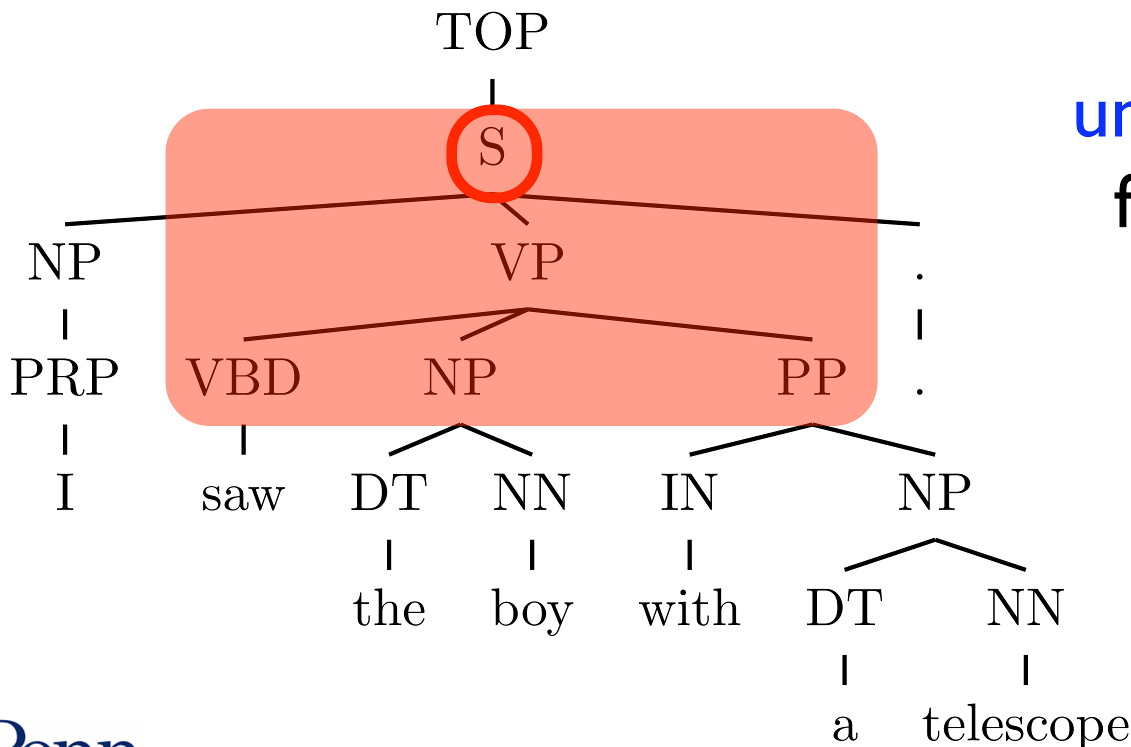


unit instance of ParentRule feature at the TOP node

Penn University of Pennsylvania

Factorizing non-local features

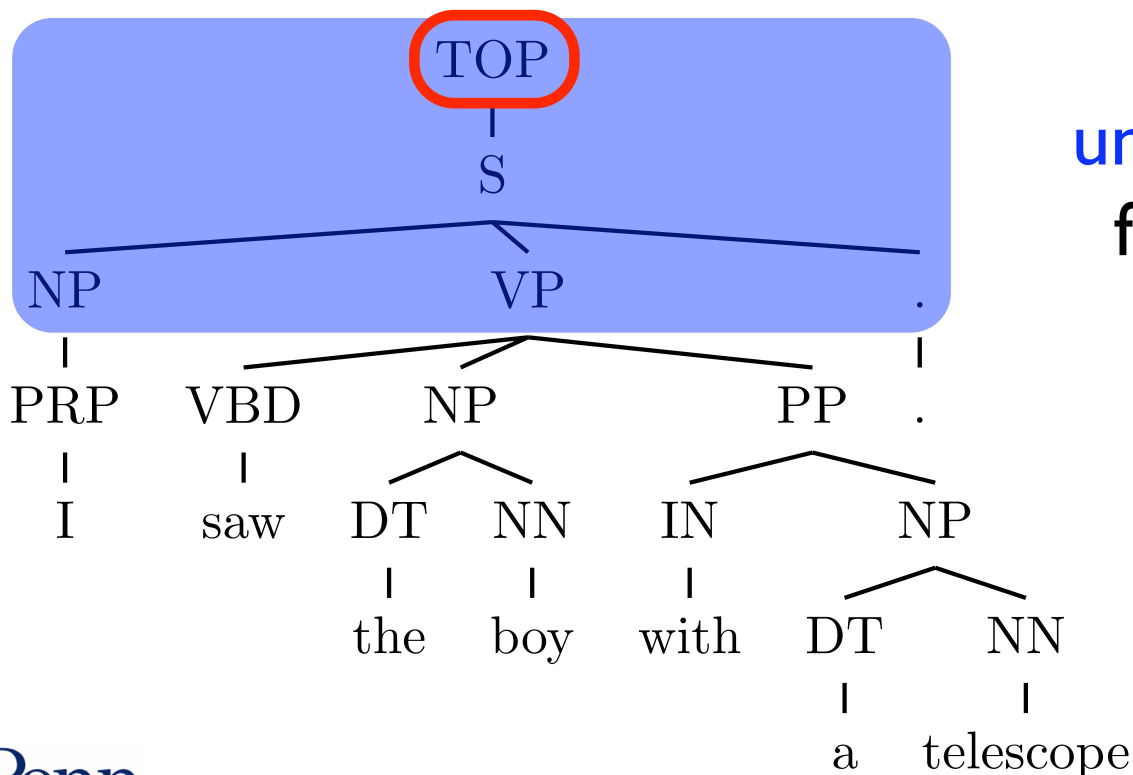
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors



unit instance of ParentRule feature at the TOP node

Factorizing non-local features

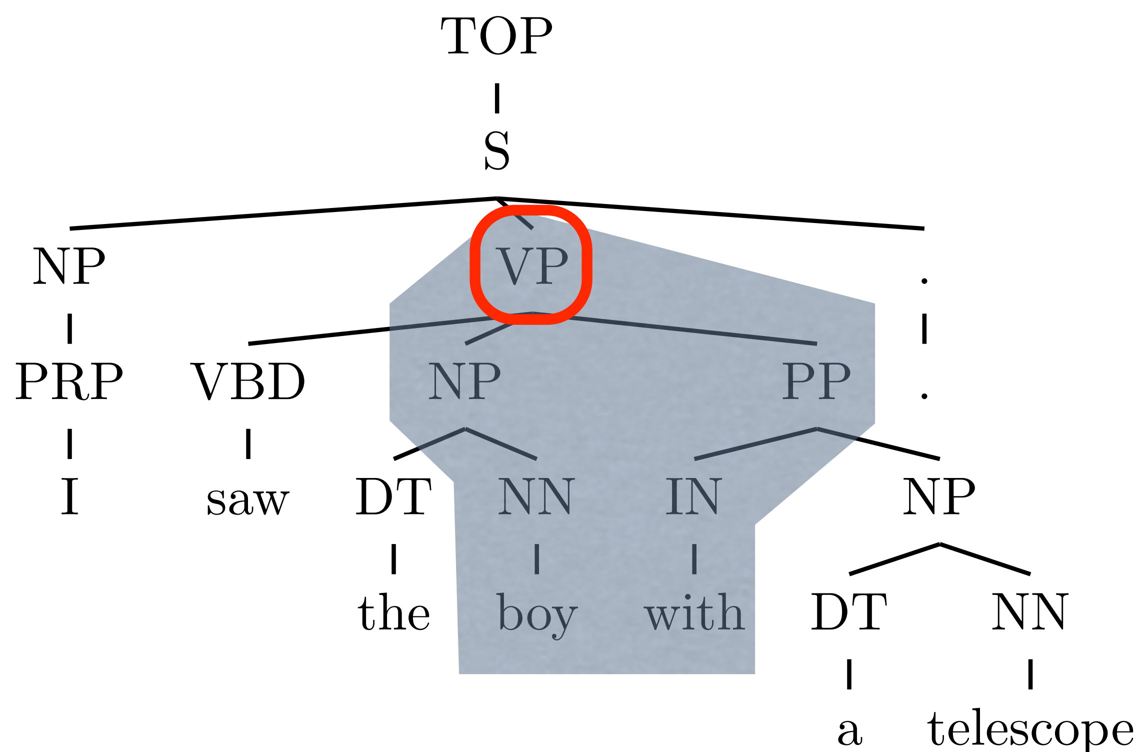
- going bottom-up, at each node
  - compute (partial values of) feature instances that become computable at this level
  - postpone those uncomputable to ancestors

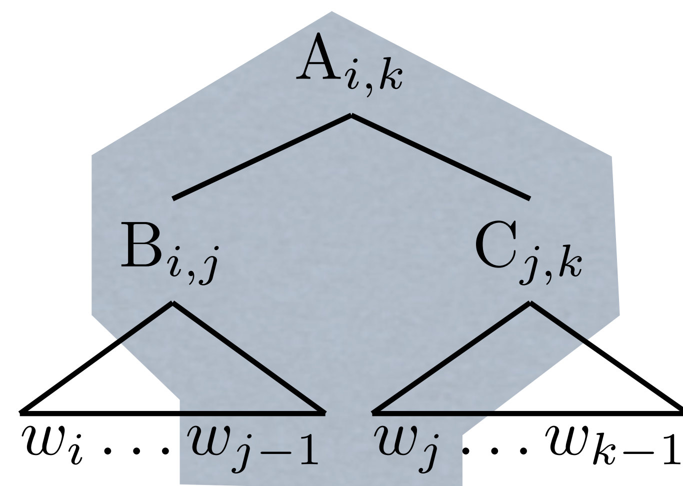


unit instance of ParentRule feature at the TOP node

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

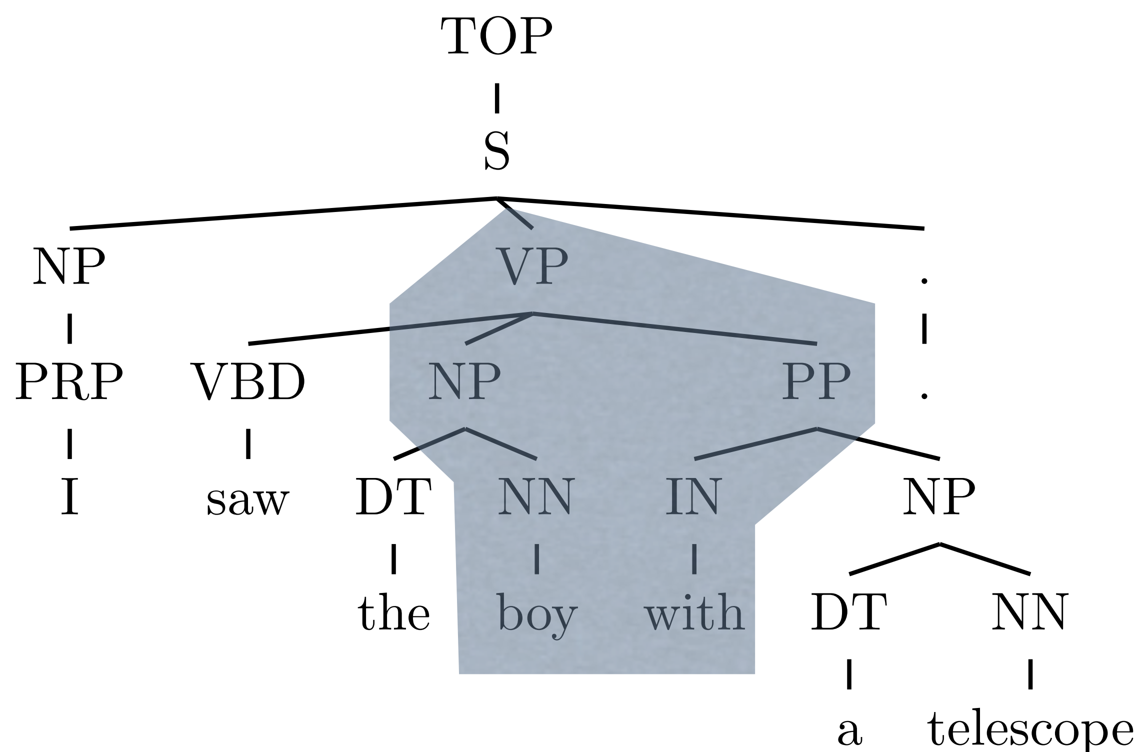


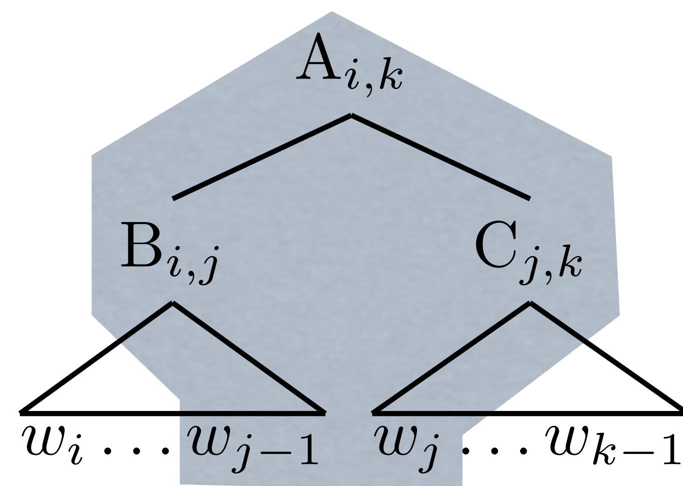


unit instance of node A

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

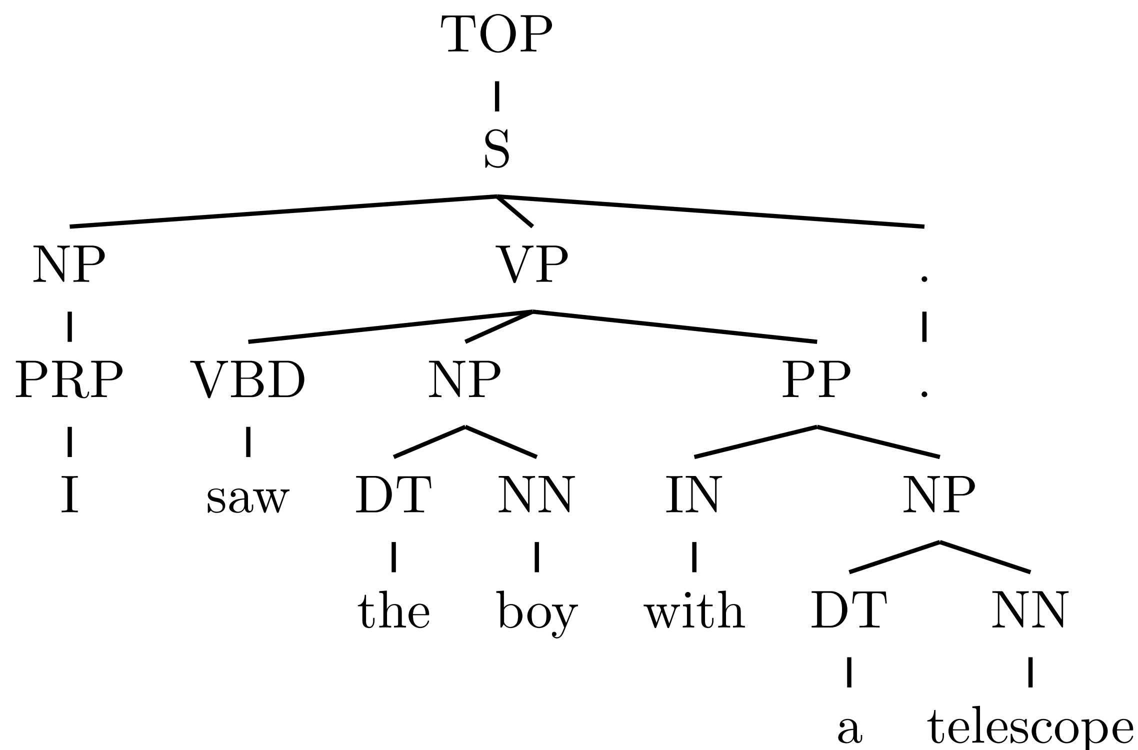


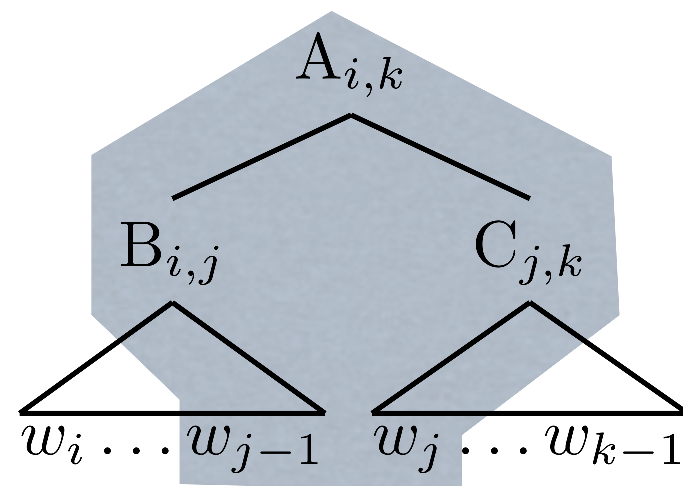


unit instance of node A

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

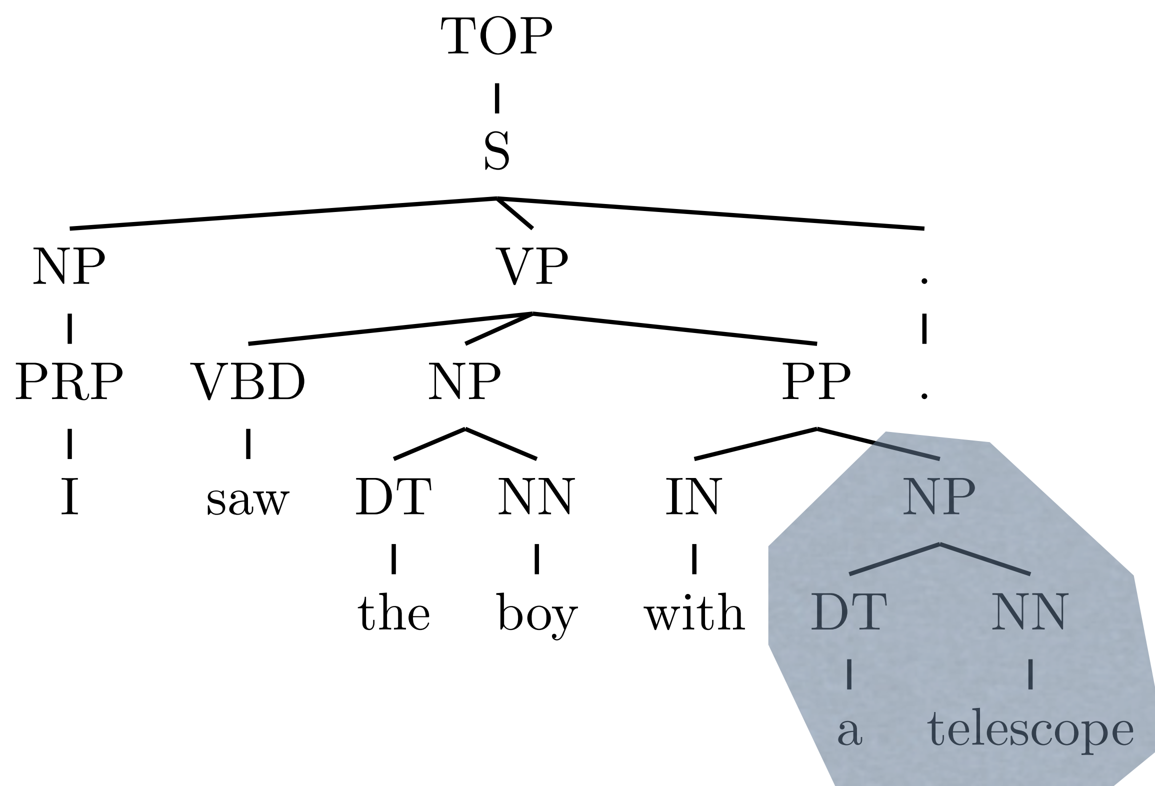


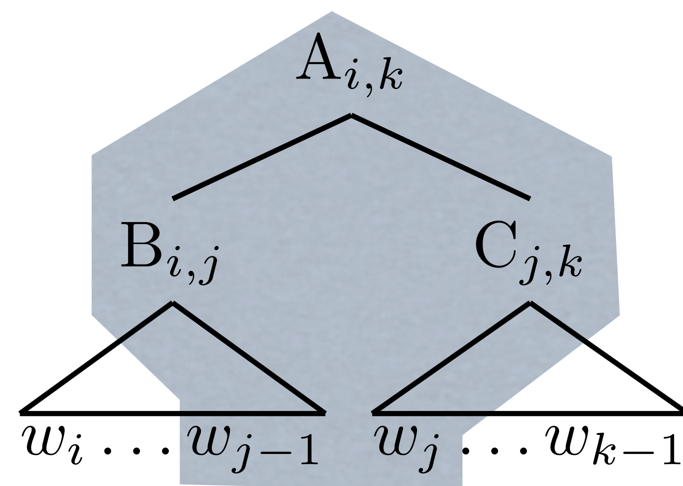


unit instance of node A

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

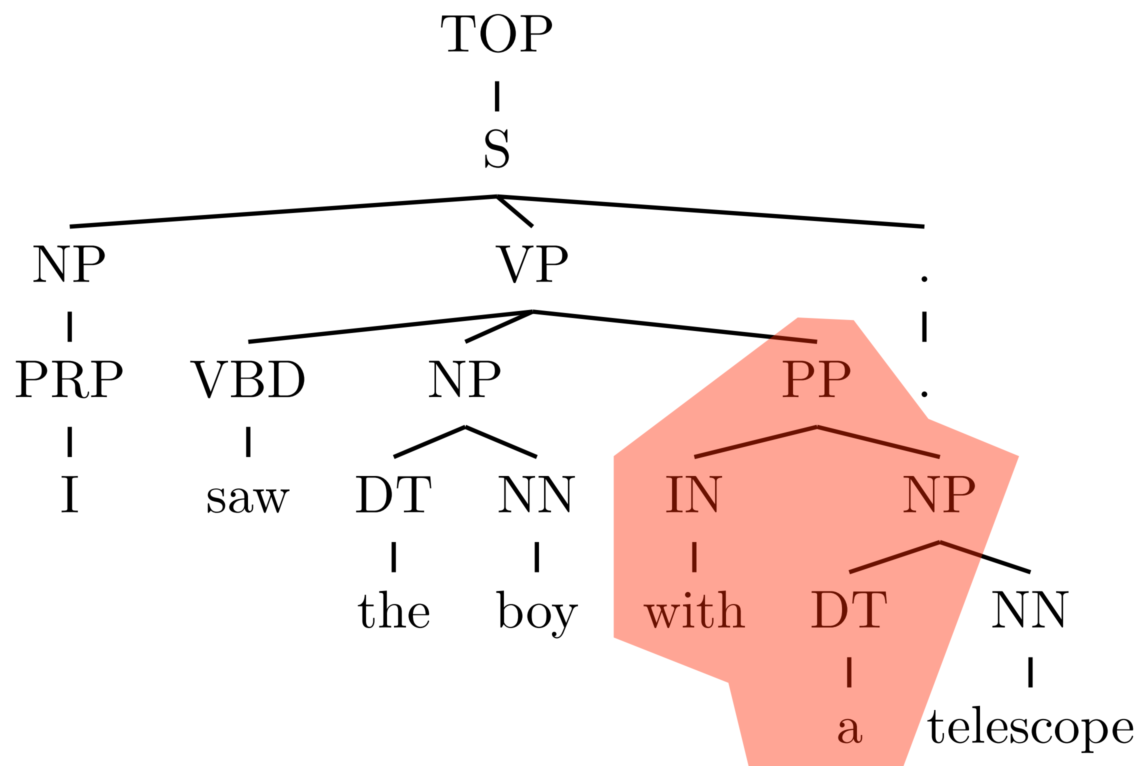


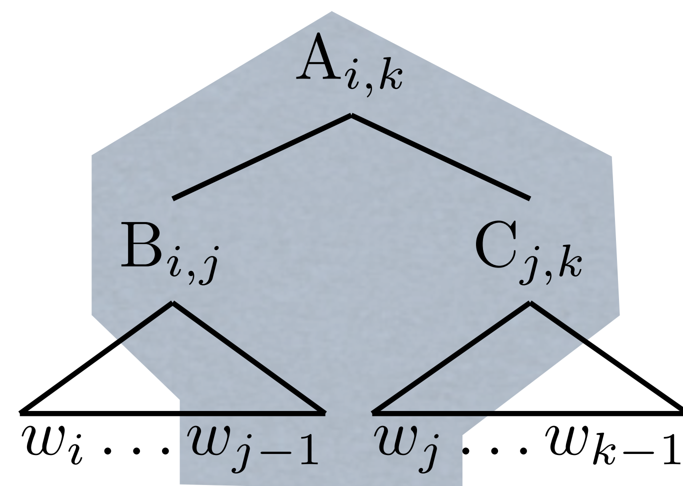


unit instance of node A

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

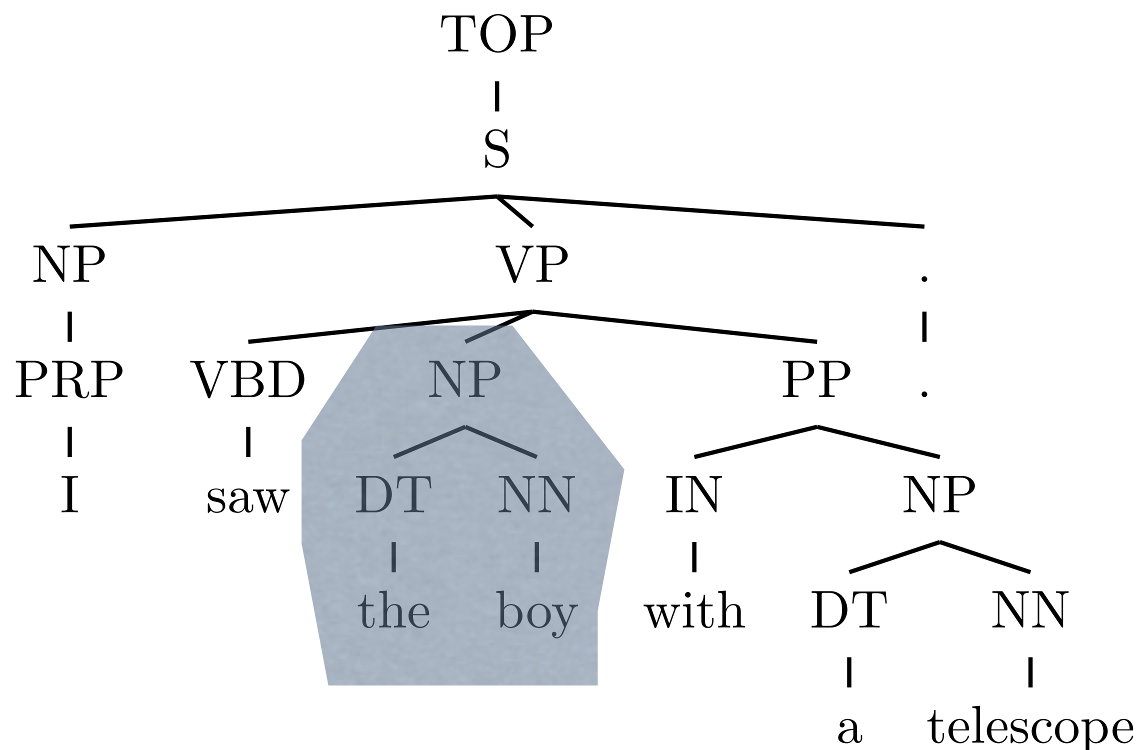


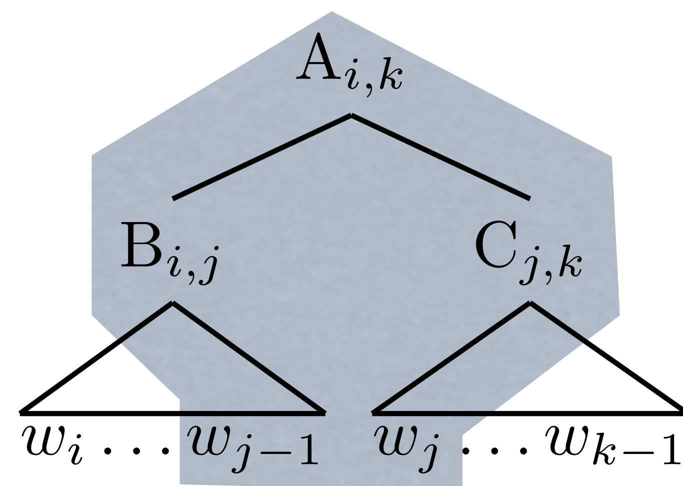


unit instance of node A

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

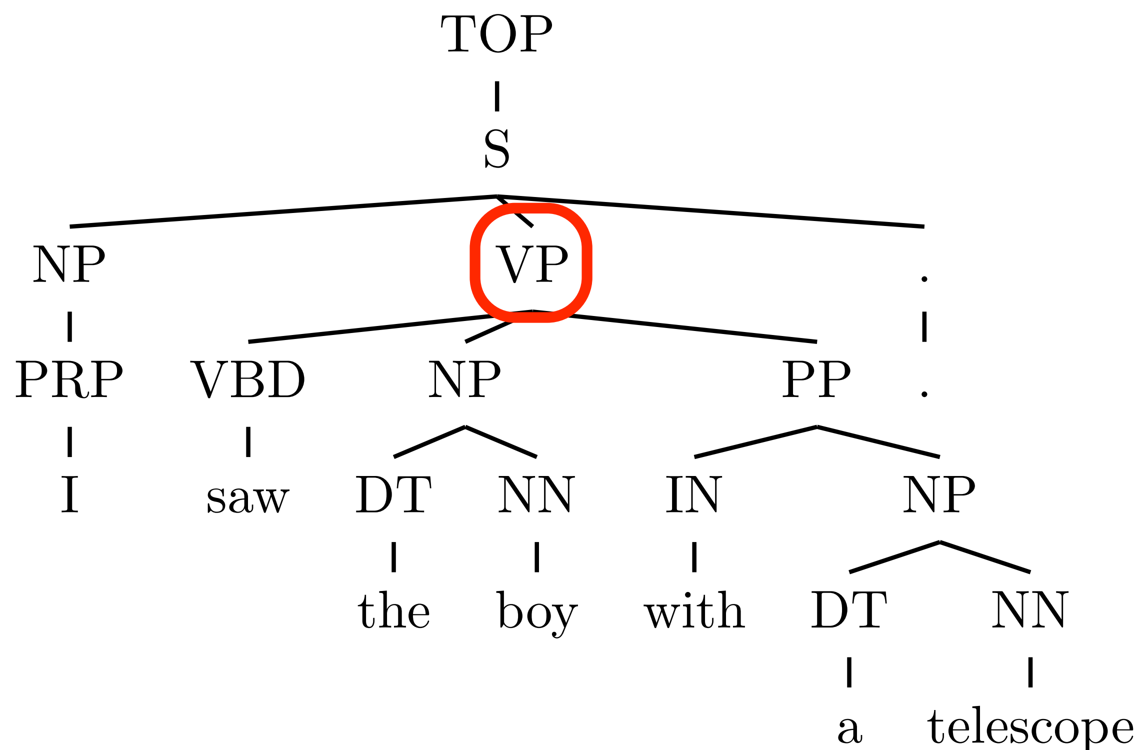


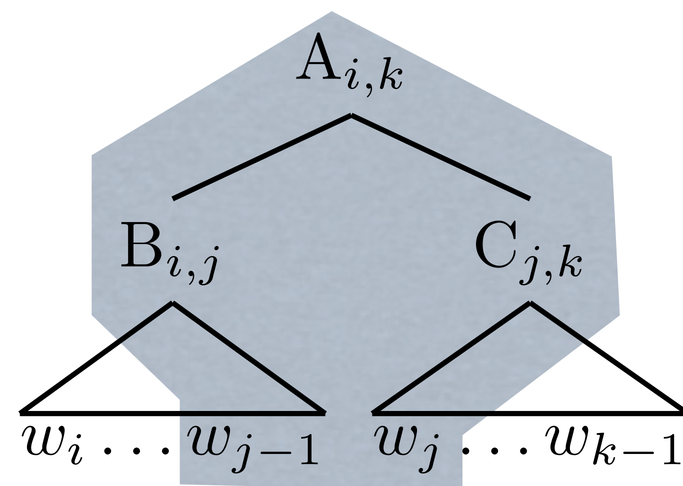


unit instance of node A

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

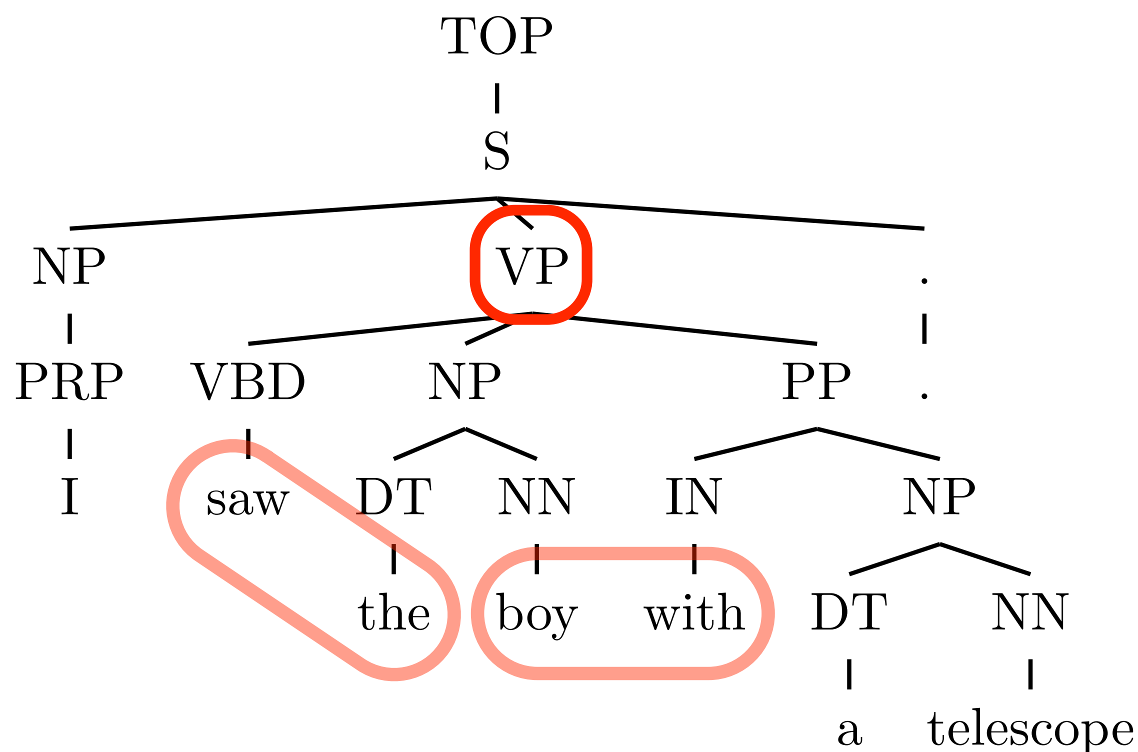


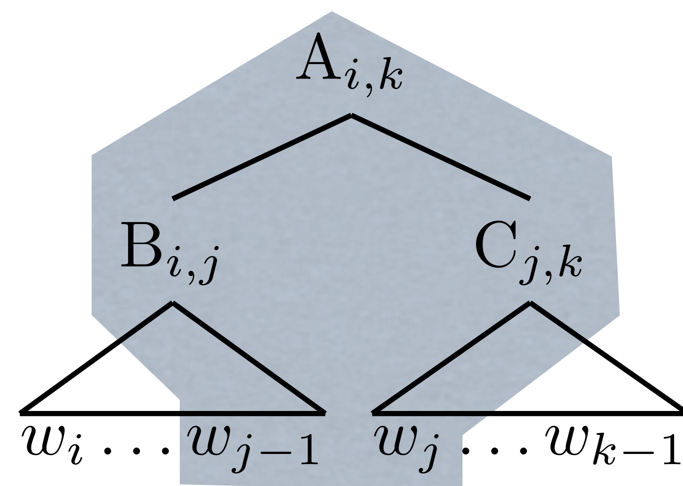


unit instance of node A

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

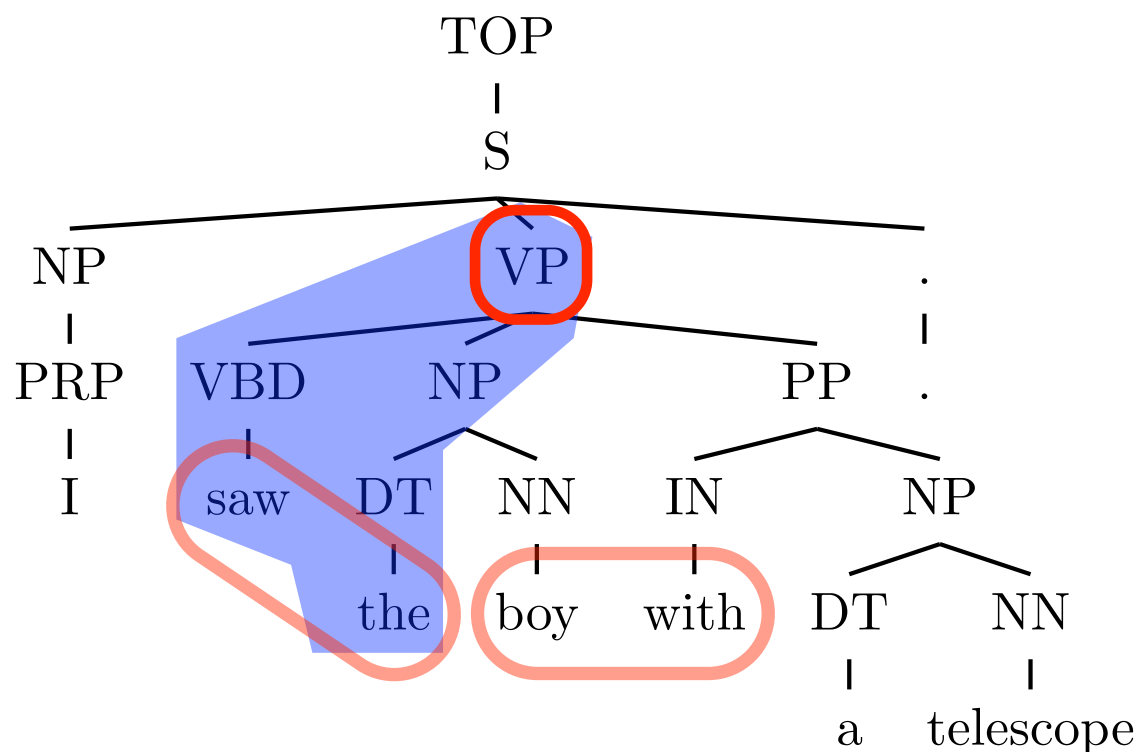


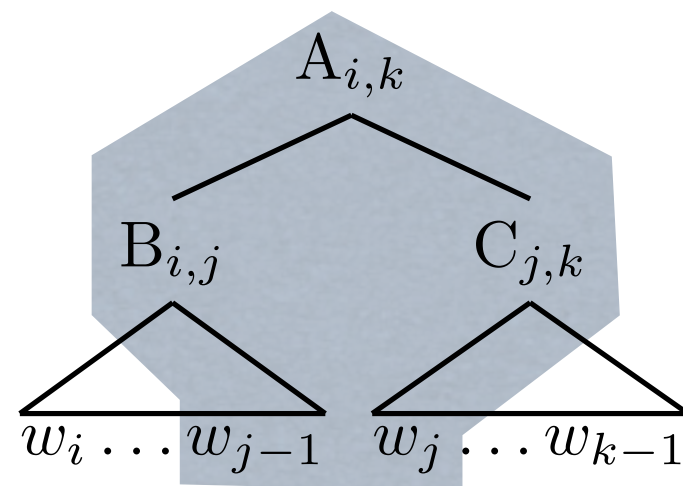


unit instance of node A

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

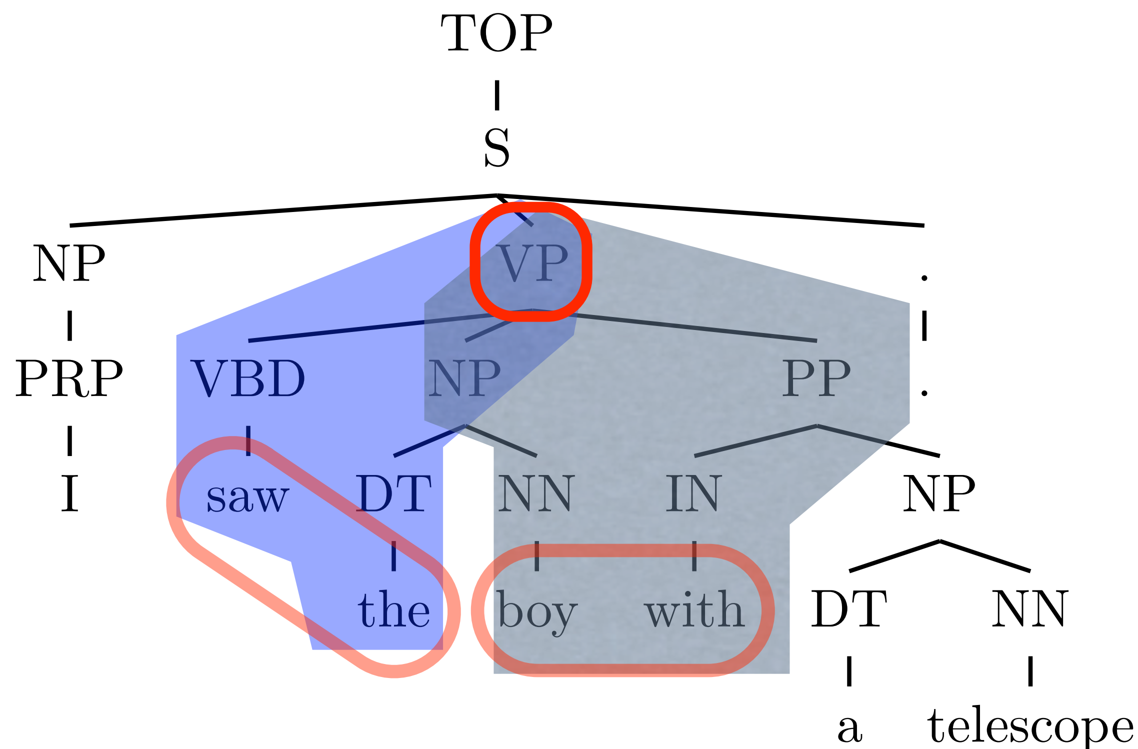


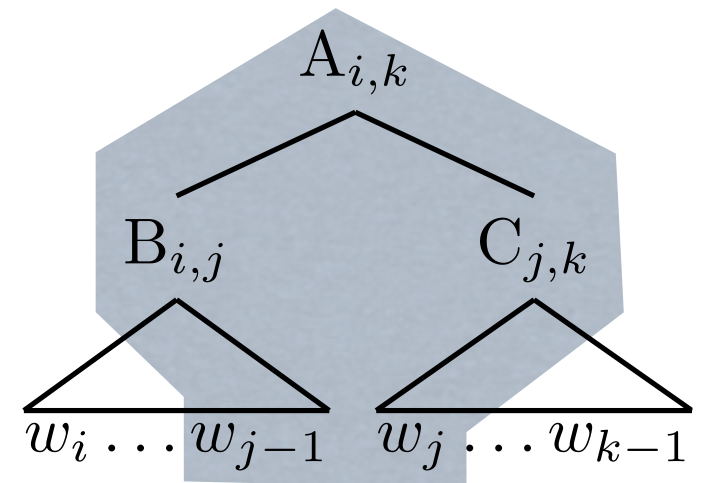


unit instance of node A

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

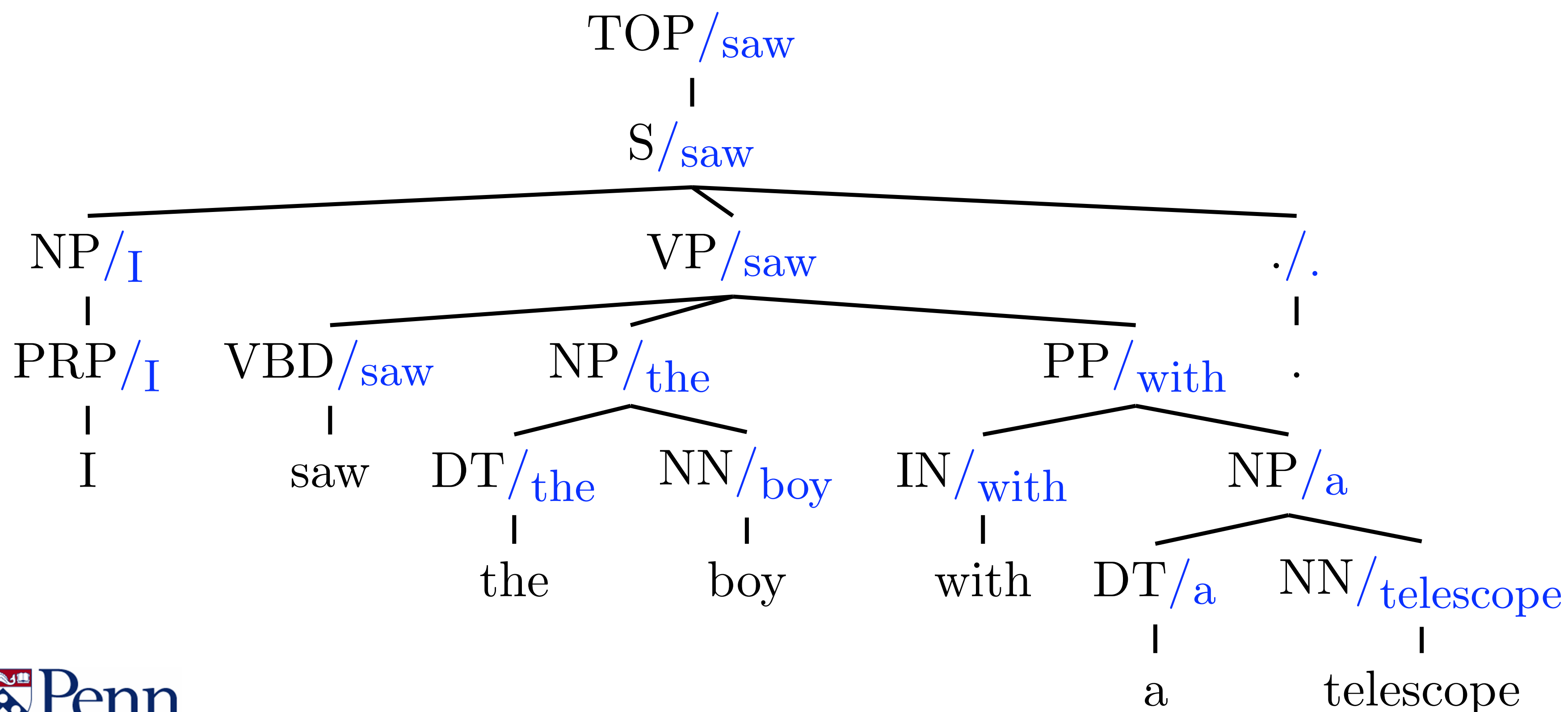




unit instance of node A

Heads (C&J 05, Collins 00)

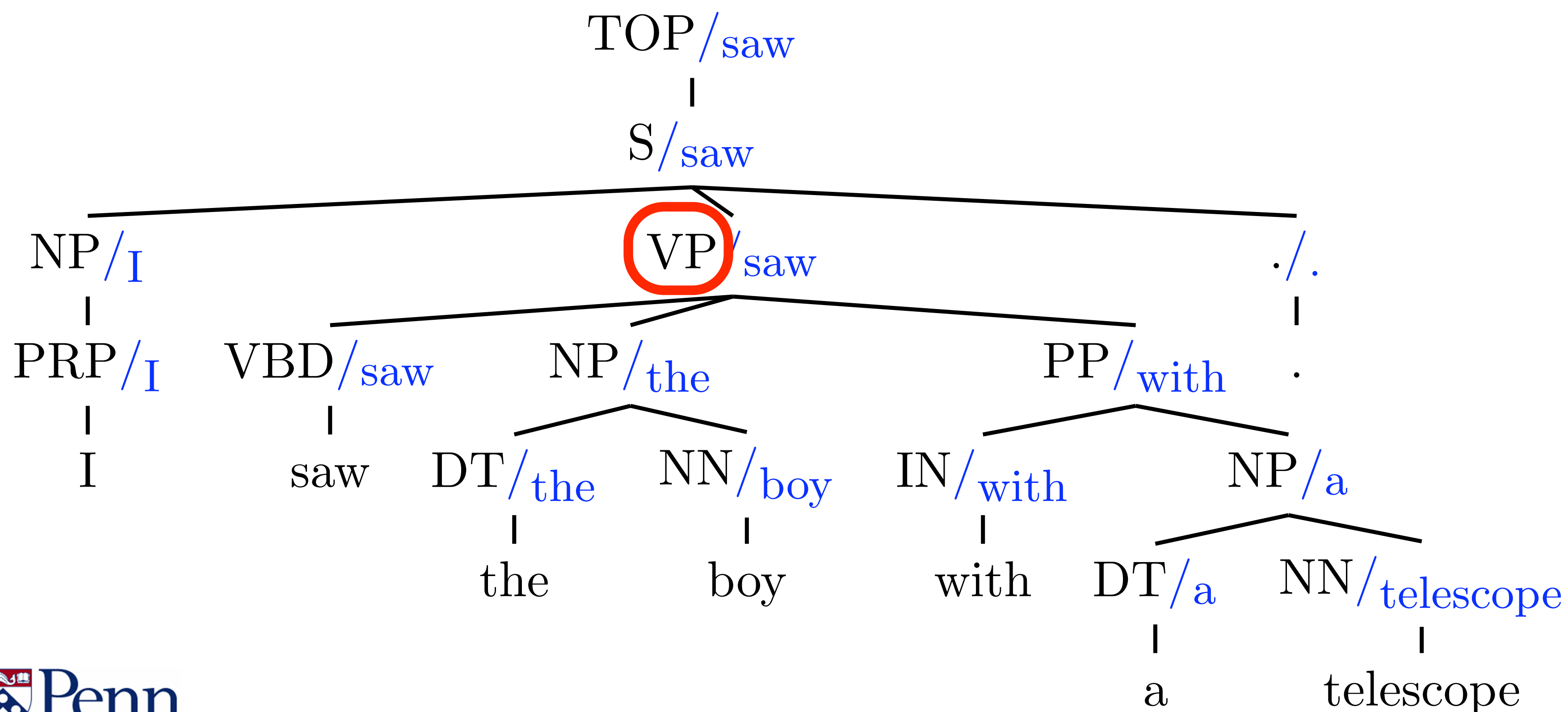
- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children



Penn University of Pennsylvania

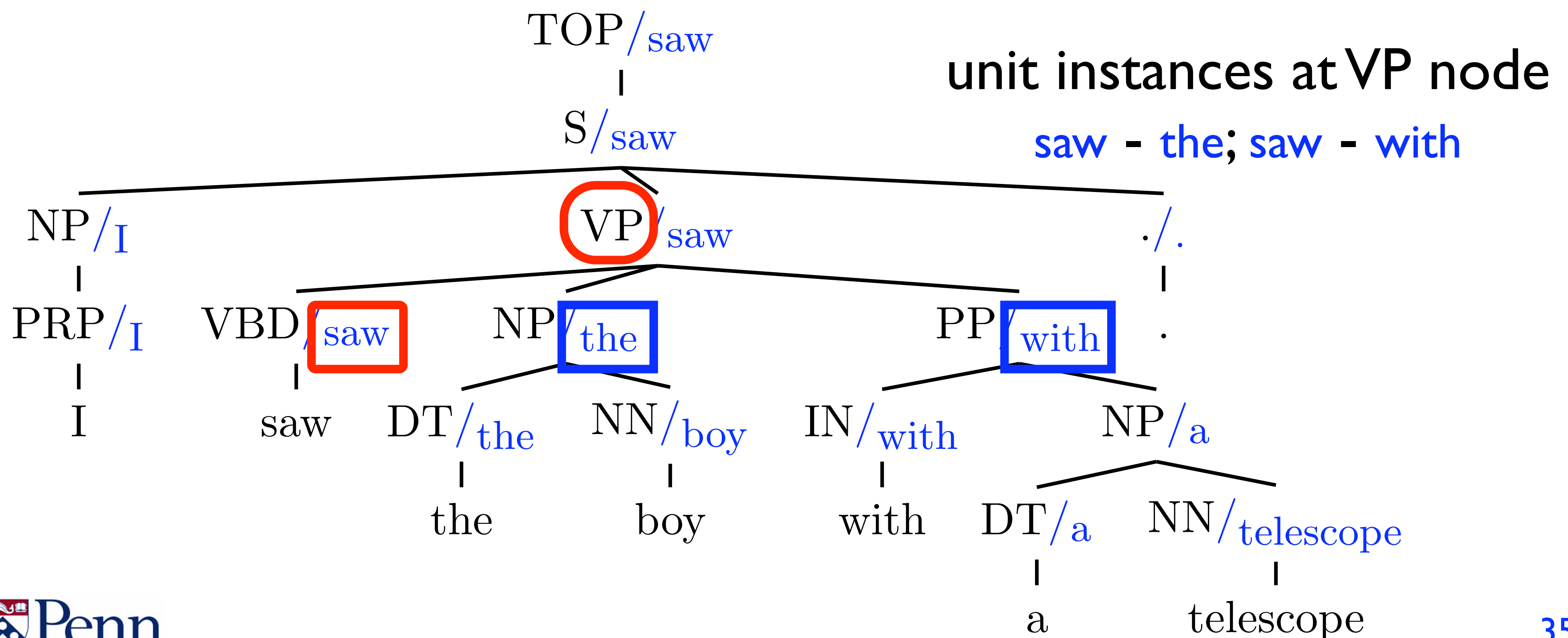
Heads (C&J 05, Collins 00)

- 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)

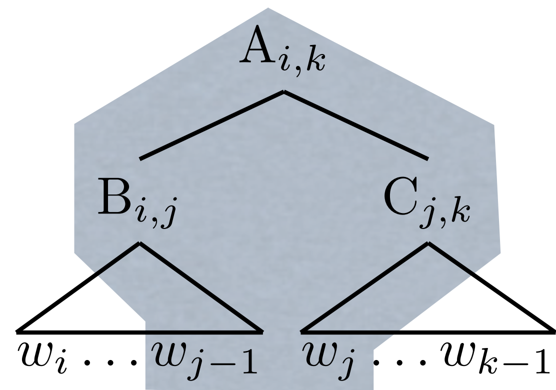
- head-to-head lexical dependencies
- we percolate heads bottom-up
- unit instances are between the head word of the head child and the head words of non-head children



Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

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

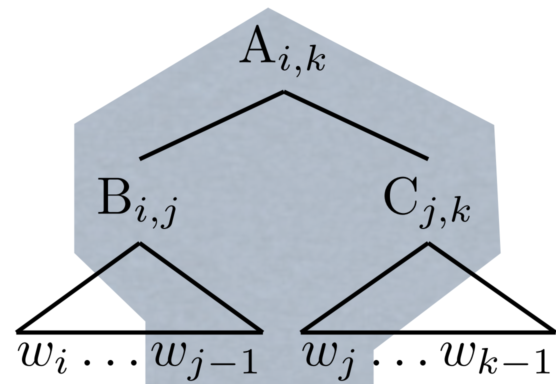


	1.0	3.0	8.0
1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
3.5	4.5 + 0.6	6.5 +10.5	11.5 + 0.6

Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

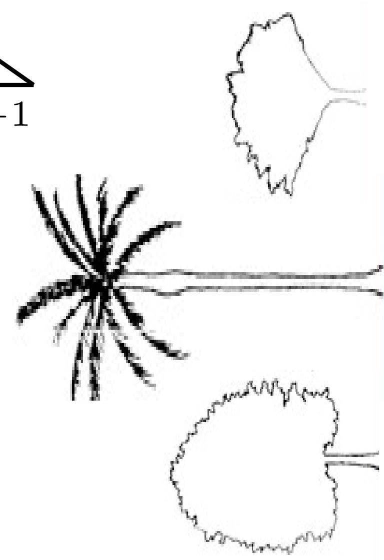
$\mathbf{w} \cdot \mathbf{f}_N($) = 0.5

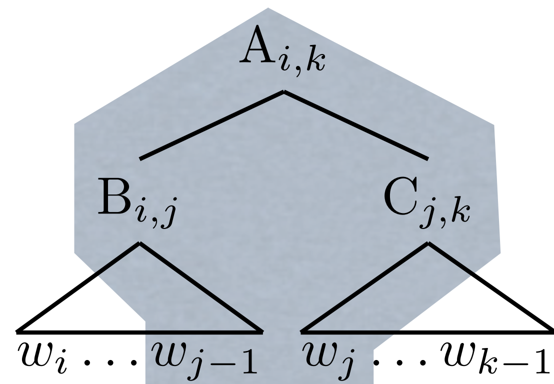


	1.0	3.0	8.0
1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
3.5	4.5 + 0.6	6.5 +10.5	11.5 + 0.6

Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features

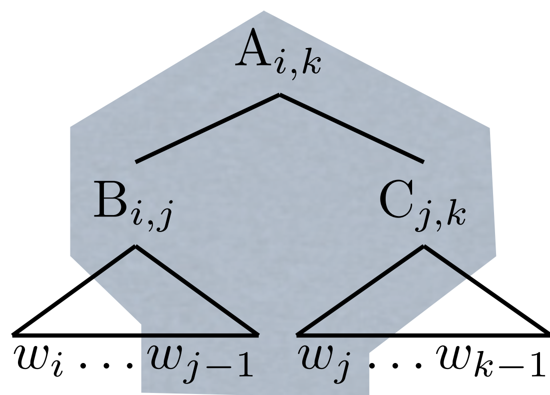
$\mathbf{w} \cdot \mathbf{f}_N($  $) = 0.5$

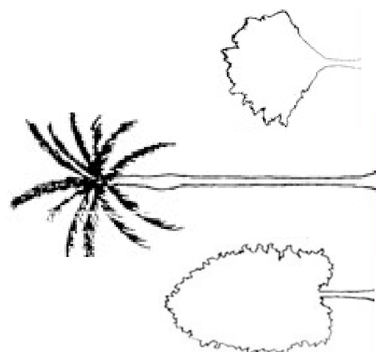


	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

Approximate Decoding

- bottom-up, keeps top *k* derivations at each node
  - non-monotonic grid due to non-local features
- priority queue for next-best
  - each iteration pops the best and pushes successors
  - extract unit non-local features on-the-fly

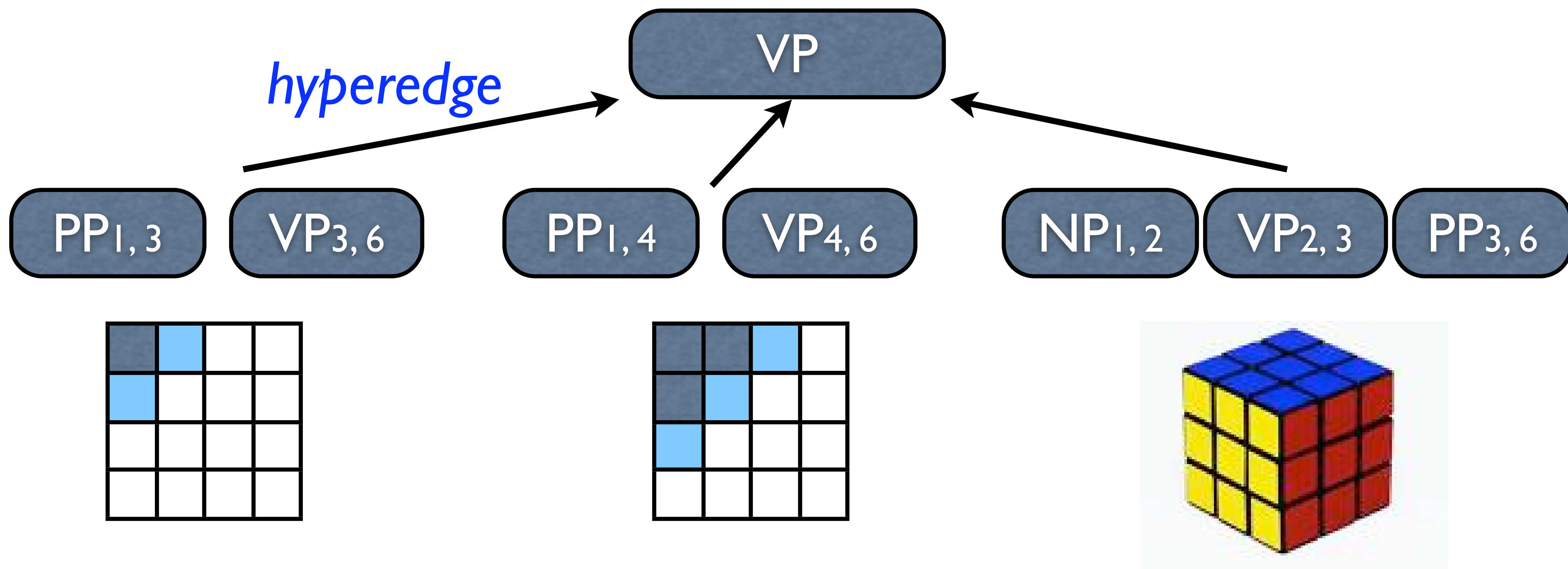




	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

Algorithm 2 => Cube Pruning

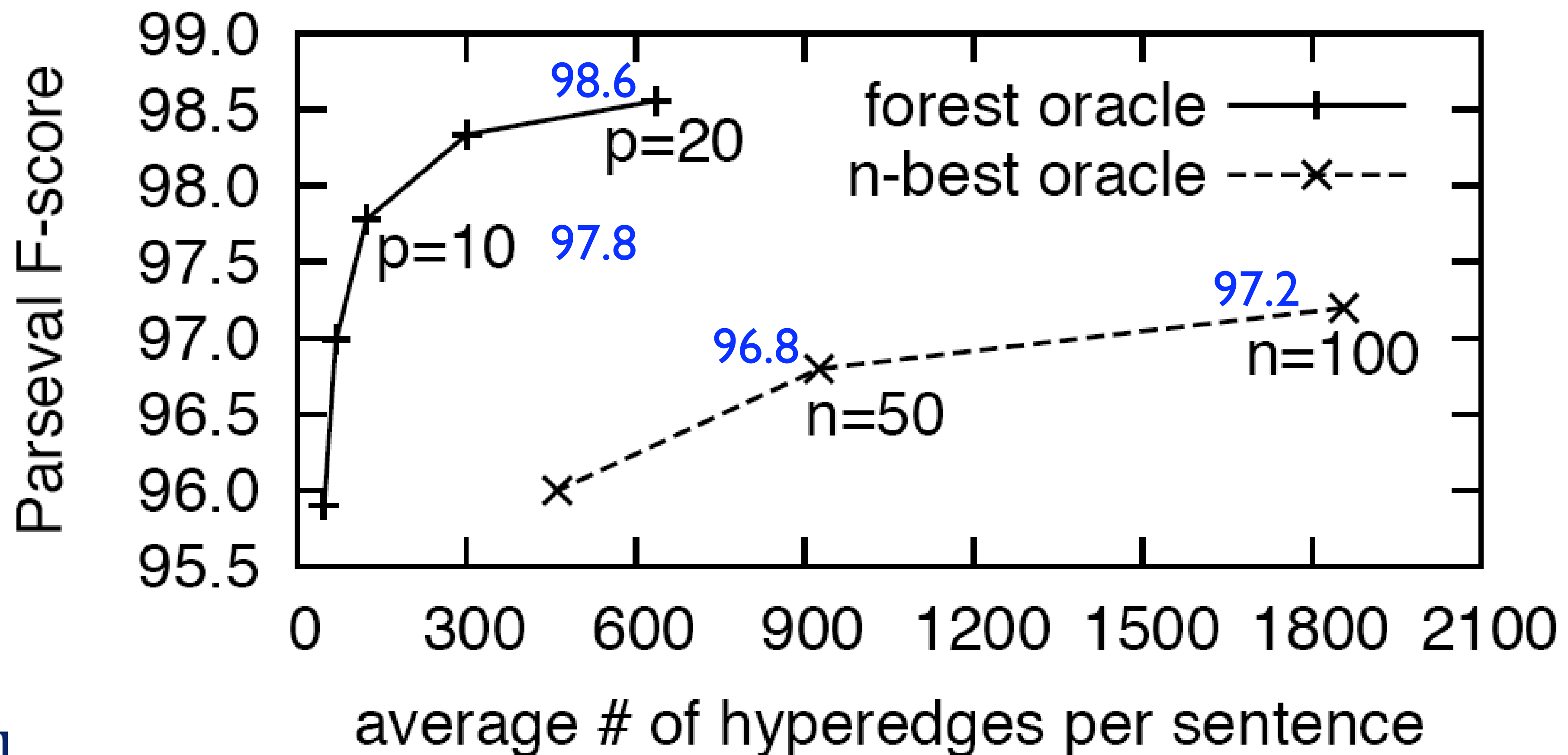
- process all hyperedges simultaneously! significant savings of computation



bottom-neck: the time for on-the-fly non-local feature extraction

Forest vs. n-best Oracles

- on top of Charniak parser (modified to dump forest)
- forests enjoy higher oracle scores than n-best lists
  - with much smaller sizes



Main Results

- pre-comp. is for feature-extraction (can be parallelized)
- # of training iterations is determined on the dev set
- forest reranking outperforms both 50- and 100-best

baseline: 1-best Charniak parser				89.72
features	*n* or *k*	pre-comp.	training	F_1%
local	50	1.4G / 25h	1 x 0.3h	91.01
all	50	2.4G / 34h	5 x 0.5h	91.43
all	100	5.3G / 77h	5 x 1.3h	91.47
local	-	1.2G / 5.1h	3 x 1.4h	91.25
all	*k*=15		4 x 11h	91.69

Penn
University *of* Pennsylvania

Comparison with Others

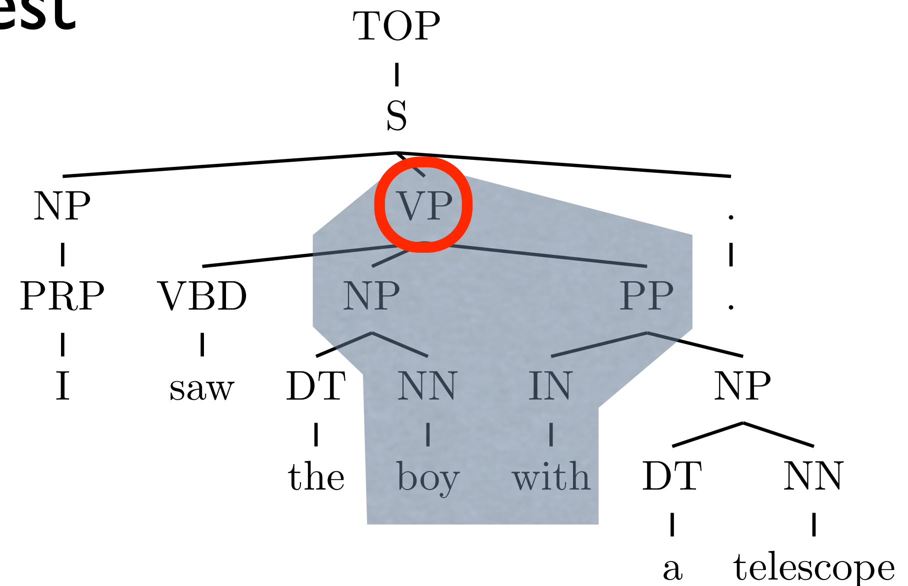
type	system	F_1%
D	Collins (2000)	89.7
	Henderson (2004)	90.1
	Charniak and Johnson (2005)	91.0
	updated (2006)	91.4
	Petrov and Klein (2008)	88.3
	this work	91.7
G	Bod (2000)	90.7
	Petrov and Klein (2007)	90.1
S	McClosky et al. (2006)	92.1

best accuracy to date on the Penn Treebank

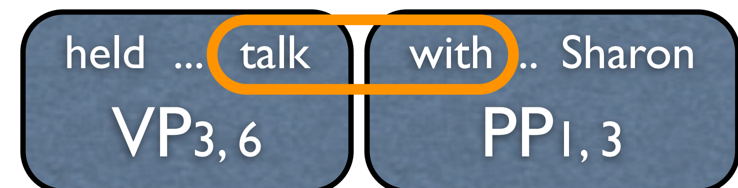
Penn
University *of* Pennsylvania

Outline

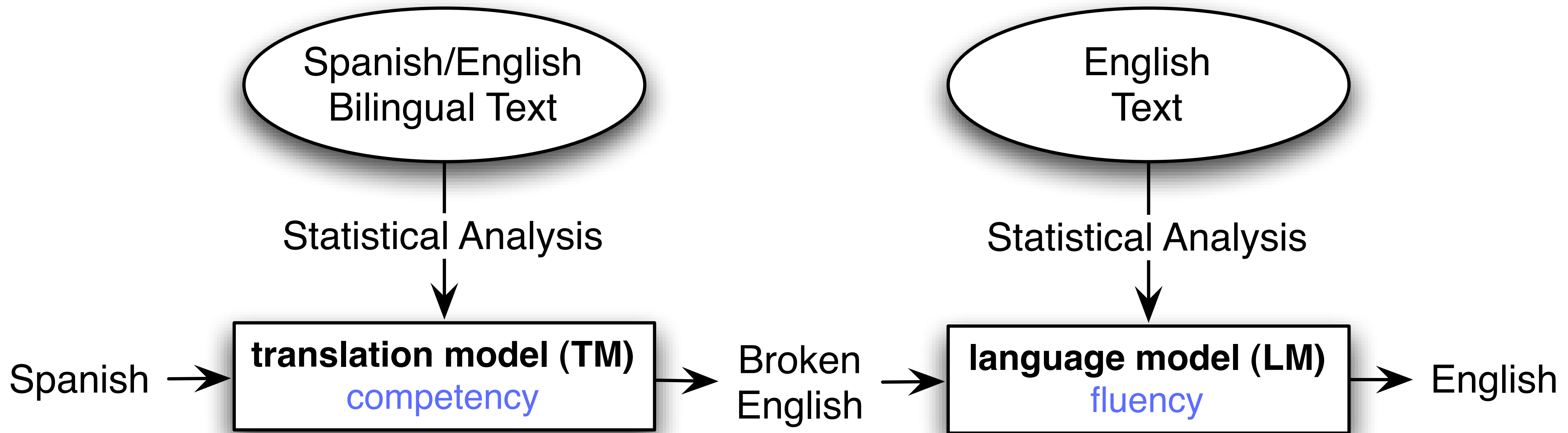
- Packed Forests and Hypergraph Framework
- Exact *k*-best Search in the Forest
- Approximate Joint Search with Non-Local Features
  - Forest Reranking
- Machine Translation
  - Decoding w/ Language Models
  - Forest Rescoring
- Future Directions



bigram



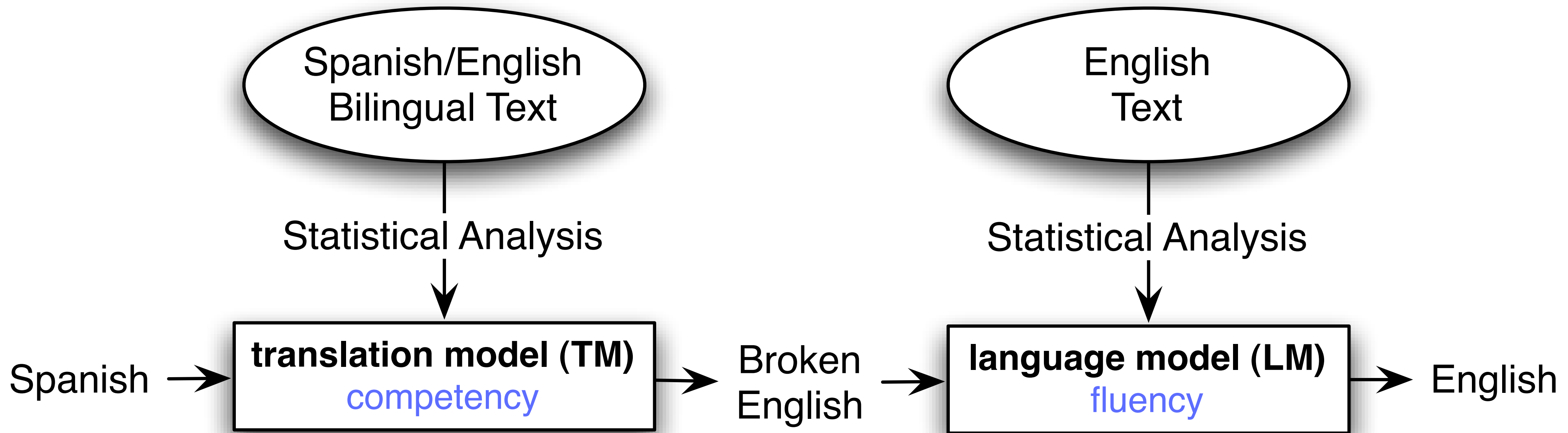
Statistical Machine Translation



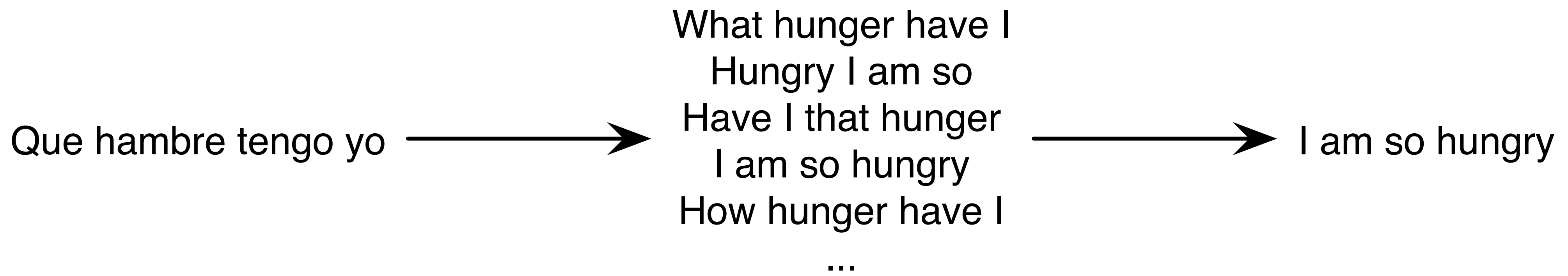
Que hambre tengo yo → What hunger have I / Hungry I am so / Have I that hunger / I am so hungry / How hunger have I / ... → I am so hungry

(Knight and Koehn, 2003)

Statistical Machine Translation

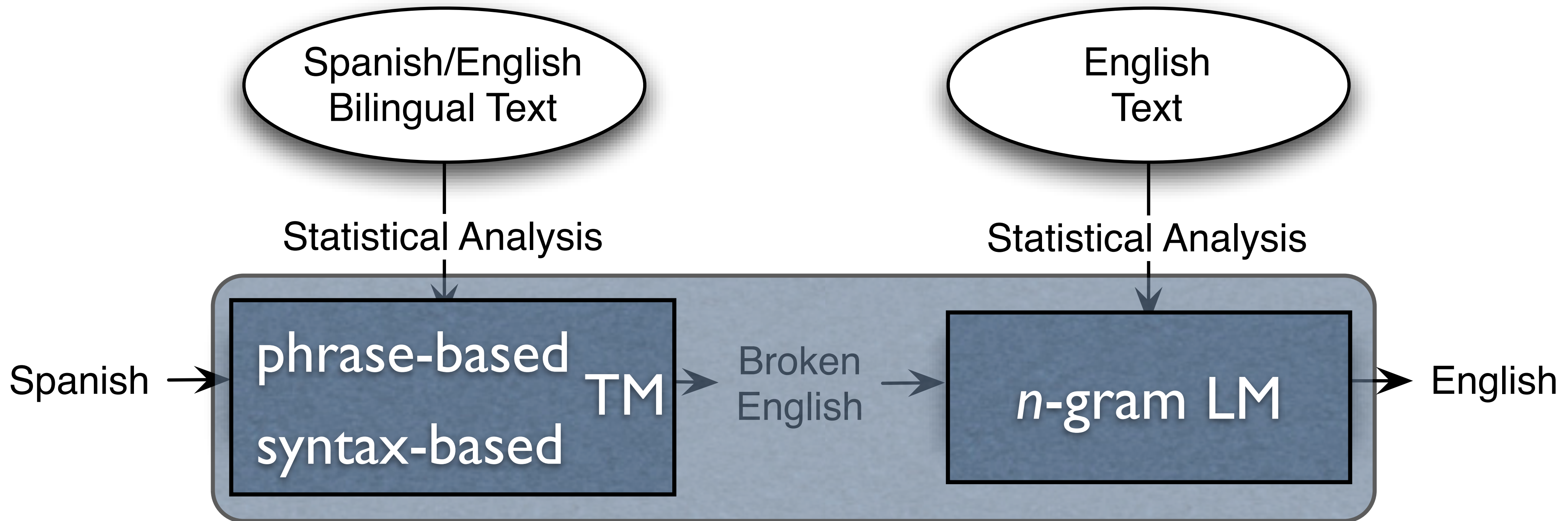


k-best rescoring (Algorithm 3)



(Knight and Koehn, 2003)

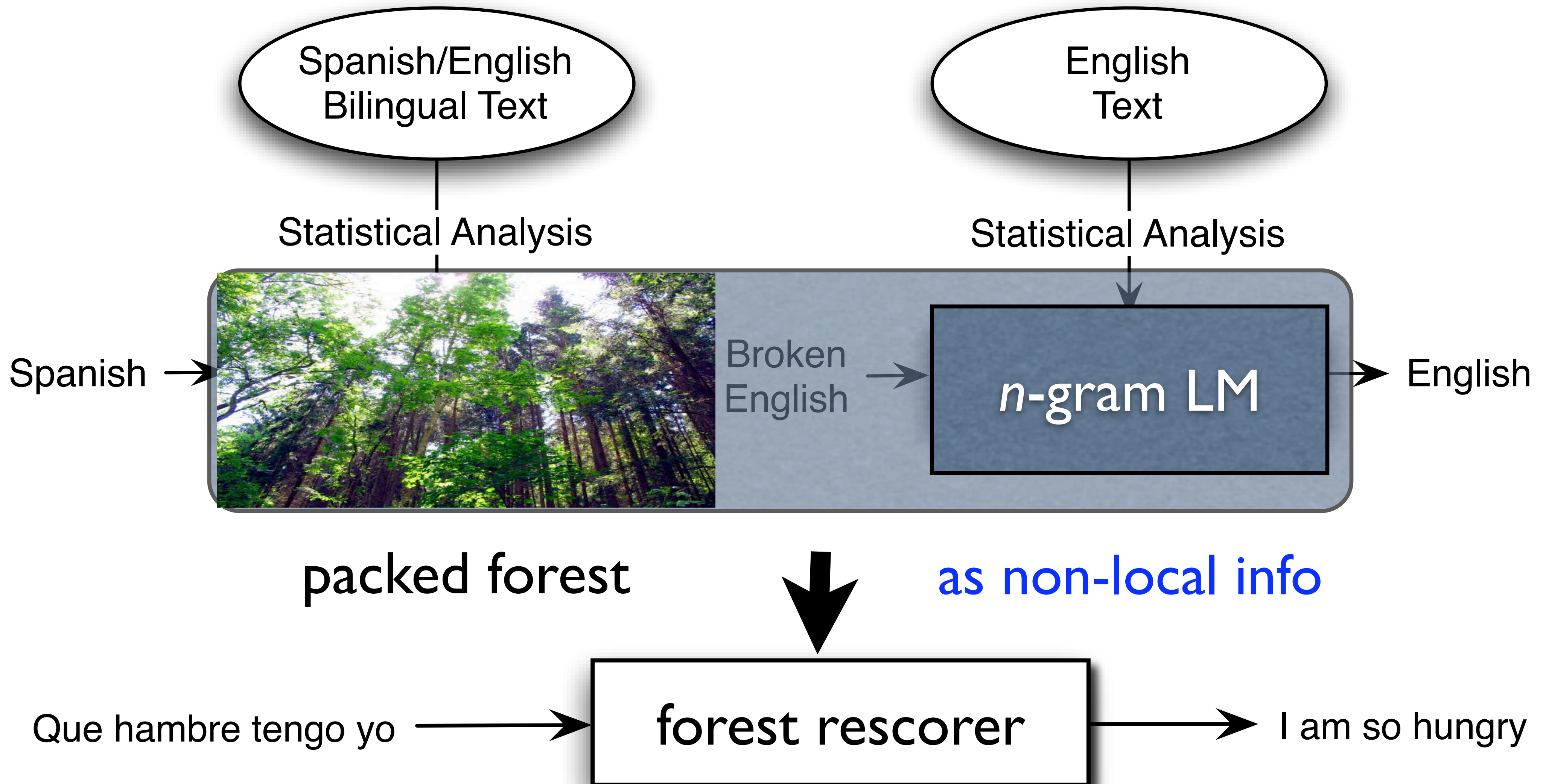
Statistical Machine Translation



Que hambre tengo yo → **integrated decoder** → I am so hungry

computationally challenging! ☹

Forest Rescoring



Syntax-based Translation

- synchronous context-free grammars (SCFGs)
  - context-free grammar in two dimensions
  - generating pairs of strings/trees simultaneously
  - co-indexed nonterminal further rewritten as a unit

$$\begin{array}{llll} \mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\ \mathbf{VP} & \rightarrow & \textit{juxing le huitan}, & \textbf{held a meeting} \\ \mathbf{PP} & \rightarrow & \textit{yu Shalong}, & \textbf{with Sharon} \end{array}$$

Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

$$\begin{array}{lcl} \mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, \\ \mathbf{VP} & \rightarrow & \textit{juxing le huitan}, \\ \mathbf{PP} & \rightarrow & \textit{yu Shalong}, \end{array}$$

$VP_{1,6}$

$PP_{1,3}$ $VP_{3,6}$

yu Shalong *juxing le huitan*

Penn
University of Pennsylvania

Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

$$\begin{array}{lcll} \mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\ \mathbf{VP} & \rightarrow & \textit{juxing le huitan}, & \mathbf{held\ a\ meeting} \\ \mathbf{PP} & \rightarrow & \textit{yu Shalong}, & \mathbf{with\ Sharon} \end{array}$$

held a talk with Sharon

VP1, 6

with Sharon | held a talk

PP1, 3 | VP3, 6

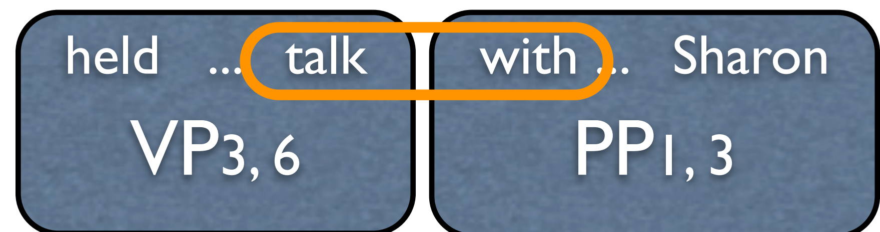
yu Shalong | *juxing le huitan*

Penn
University of Pennsylvania

Adding a Bigram Model

- exact dynamic programming
  - nodes now split into +LM items
  - with English boundary words
- search space too big for exact search
  - beam search: keep at most k +LM items each node
  - but can we do better?

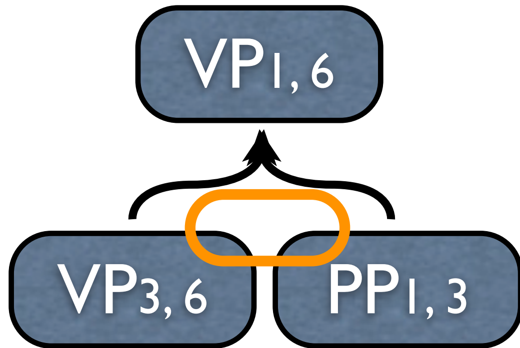
bigram



+LM items

Penn
University of Pennsylvania

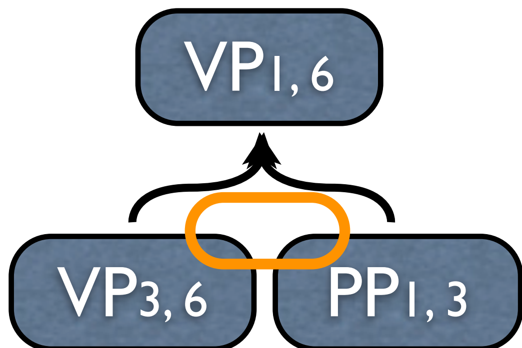
Non-Monotonic Grid



non-monotonicity
due to LM combo costs

	$(\mathrm{PP}^{\text{with} \star \text{Sharon}}_{1,3})$ 1.0	$(\mathrm{PP}^{\text{along} \star \text{Sharon}}_{1,3})$ 3.0	$(\mathrm{PP}^{\text{with} \star \text{Shalong}}_{1,3})$ 8.0
$(\mathrm{VP}^{\text{held} \star \text{meeting}}_{3,6})$ 1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
$(\mathrm{VP}^{\text{held} \star \text{talk}}_{3,6})$ 1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
$(\mathrm{VP}^{\text{hold} \star \text{conference}}_{3,6})$ 3.5	4.5 + 0.6	6.5 + 10.5	11.5 + 0.6

Non-Monotonic Grid



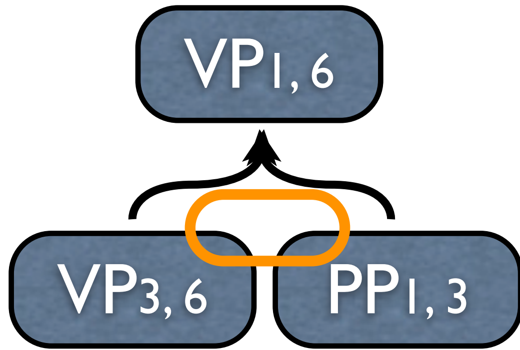
bigram (meeting, with)

non-monotonicity
due to LM combo costs

	$(PP_{1,3}^{with \star Sharon})$	$(PP_{1,3}^{along \star Sharon})$	$(PP_{1,3}^{with \star Shalong})$
	1.0	3.0	8.0
$(VP_{3,6}^{held \star meeting})$ 1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
$(VP_{3,6}^{held \star talk})$ 1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
$(VP_{3,6}^{hold \star conference})$ 3.5	4.5 + 0.6	6.5 + 10.5	11.5 + 0.6

Penn
UNIVERSITY of PENNSYLVANIA

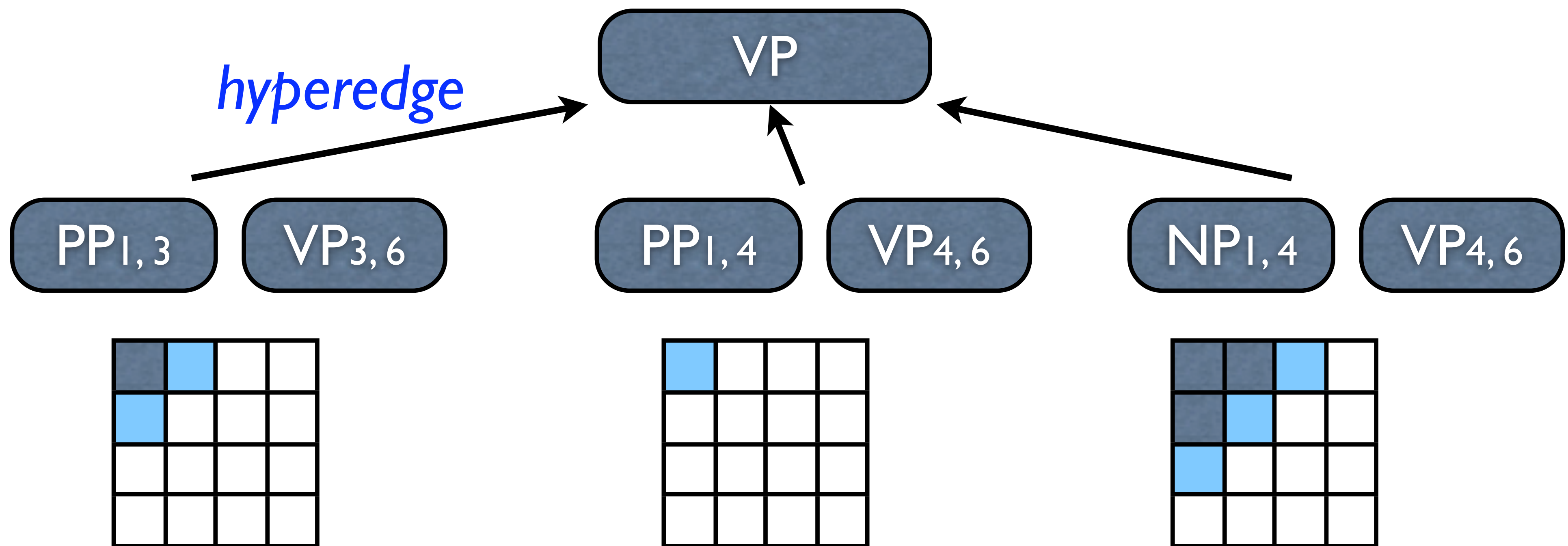
Algorithm 2 -Cube Pruning



	$(\mathrm{PP}^{\text{with} \star \text{Sharon}}_{1,3})$ 1.0	$(\mathrm{PP}^{\text{along} \star \text{Sharon}}_{1,3})$ 3.0	$(\mathrm{PP}^{\text{with} \star \text{Shalong}}_{1,3})$ 8.0
$(\mathrm{VP}^{\text{held} \star \text{meeting}}_{3,6})$ 1.0	2.5	9.0	9.5
$(\mathrm{VP}^{\text{held} \star \text{talk}}_{3,6})$ 1.1	2.4	9.5	9.4
$(\mathrm{VP}^{\text{hold} \star \text{conference}}_{3,6})$ 3.5	5.1	17.0	12.1

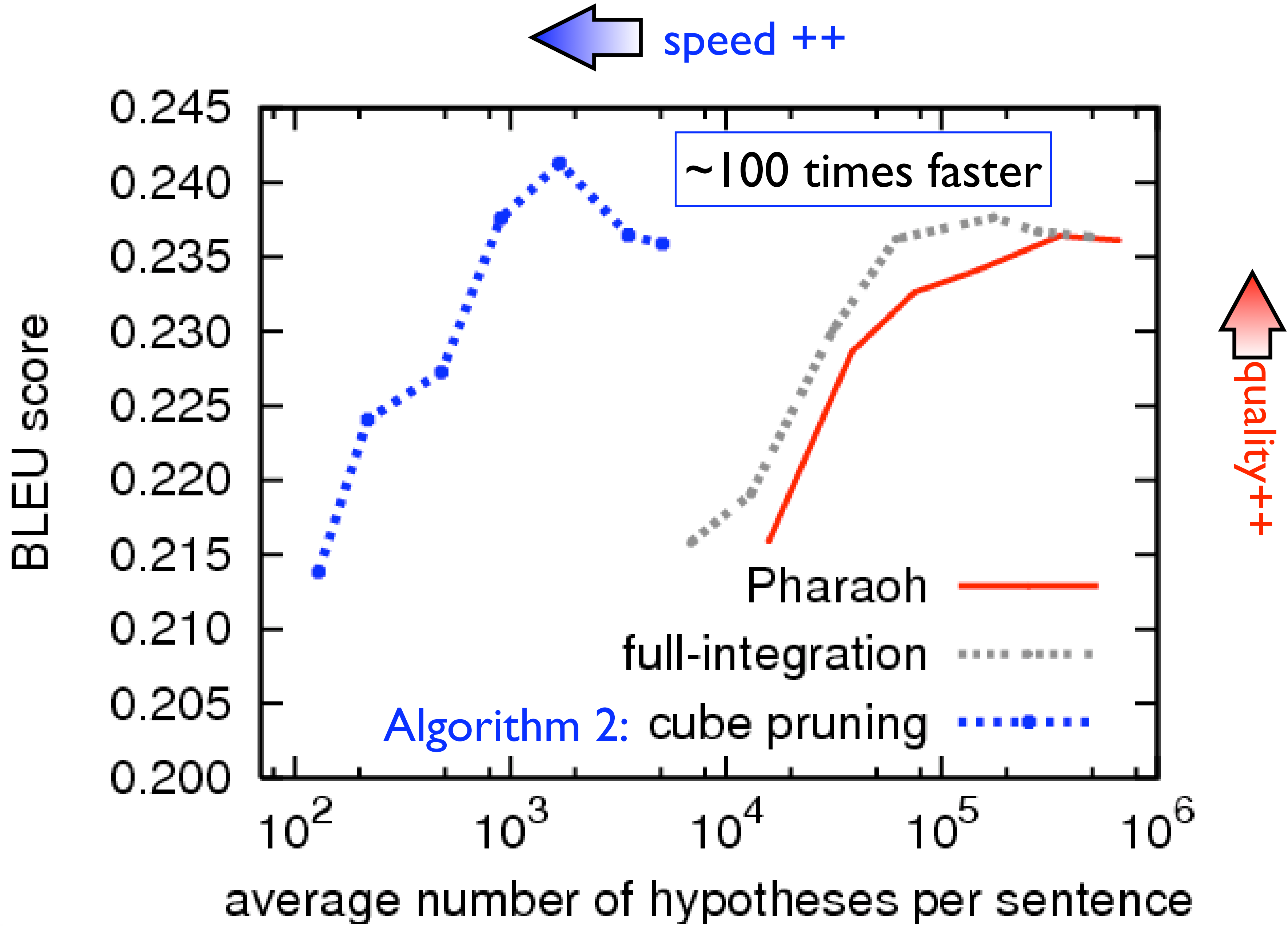
Algorithm 2 => Cube Pruning

k-best Algorithm 2,
with search errors



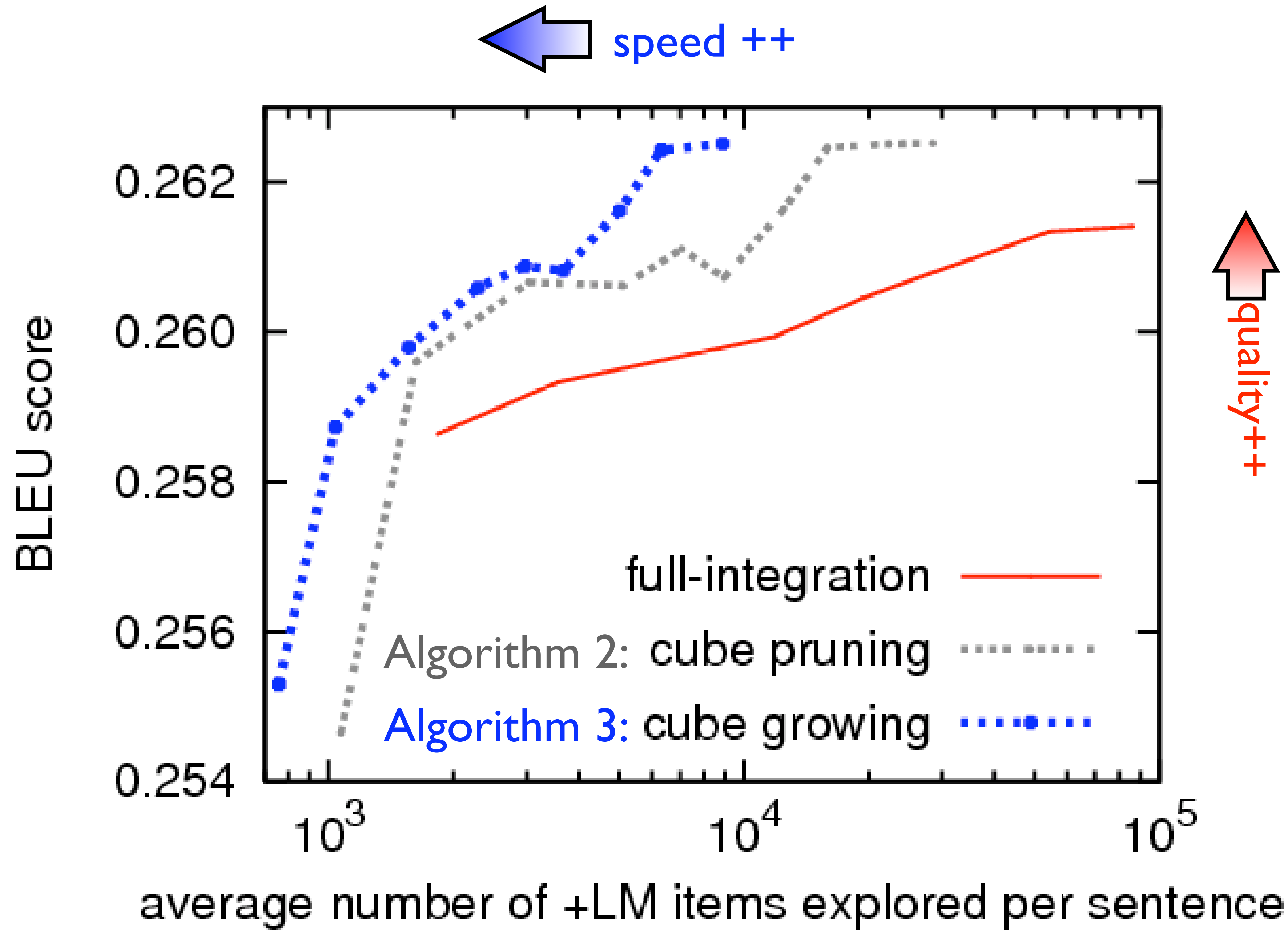
process all hyperedges simultaneously!
significant savings of computation

Phrase-based: Translation Accuracy



Penn
University of Pennsylvania

Syntax-based: Translation Accuracy



Conclusion so far

- General framework of DP on hypergraphs
  - monotonicity => exact 1-best algorithm
- Exact *k*-best algorithms
- Approximate search with non-local information
  - Forest Reranking for discriminative parsing
  - Forest Rescoring for MT decoding
- Empirical Results
  - orders of magnitudes faster than previous methods
  - best Treebank parsing accuracy to date

Penn
University *of* Pennsylvania

Impact

- These algorithms have been widely implemented in
  - state-of-the-art parsers
    - Charniak parser
    - McDonald's dependency parser
    - MIT parser (Collins/Koo), Berkeley and Stanford parsers
    - DOP parsers (Bod, 2006/7)
  - major statistical MT systems
    - Syntax-based systems from ISI, CMU, BBN, ...
    - Phrase-based system: Moses [underway]

Future Directions

Further work on Forest Reranking

- Better Decoding Algorithms
  - pre-compute most non-local features
  - use Algorithm 3 cube growing
  - intra-sentence level parallelized decoding
- Combination with Semi-supervised Learning
  - easy to apply to self-training (McClosky et al., 2006)
- Deeper and deeper Decoding (e.g., semantic roles)
- Other Machine Learning Algorithms
- Theoretical and Empirical Analysis of Search Errors

Machine Translation / Generation

- Discriminative training using non-local features
  - local-features showed modest improvement on phrase-base systems (Liang et al., 2006)
  - plan for syntax-based (tree-to-string) systems
    - fast, linear-time decoding
- Using packed parse forest for
  - tree-to-string decoding (Mi, Huang, Liu, 2008)
  - rule extraction (tree-to-tree)
- Generation / Summarization: non-local constraints

Thanks!

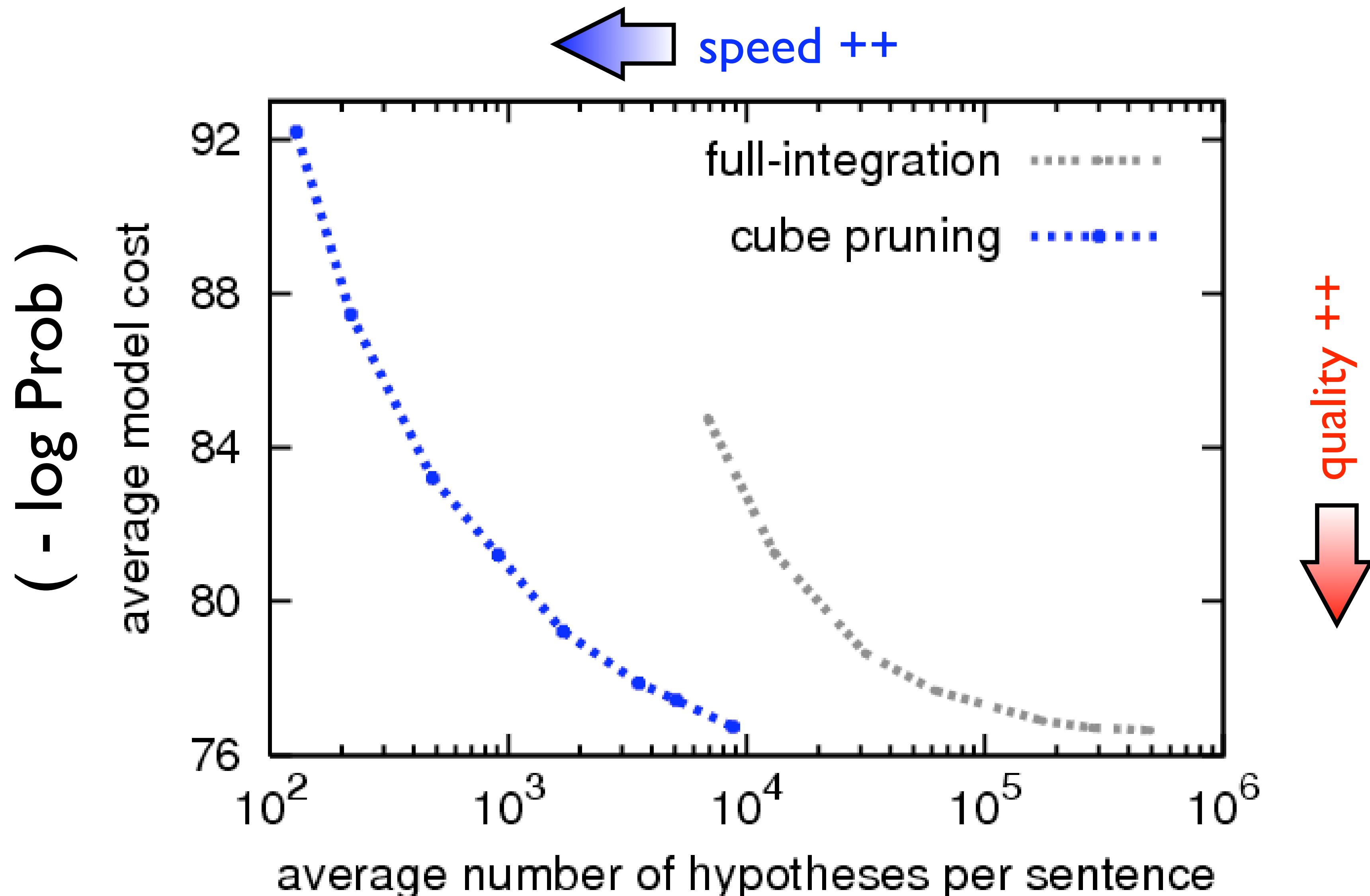
Questions?

Comments?

Penn
UNIVERSITY of PENNSYLVANIA

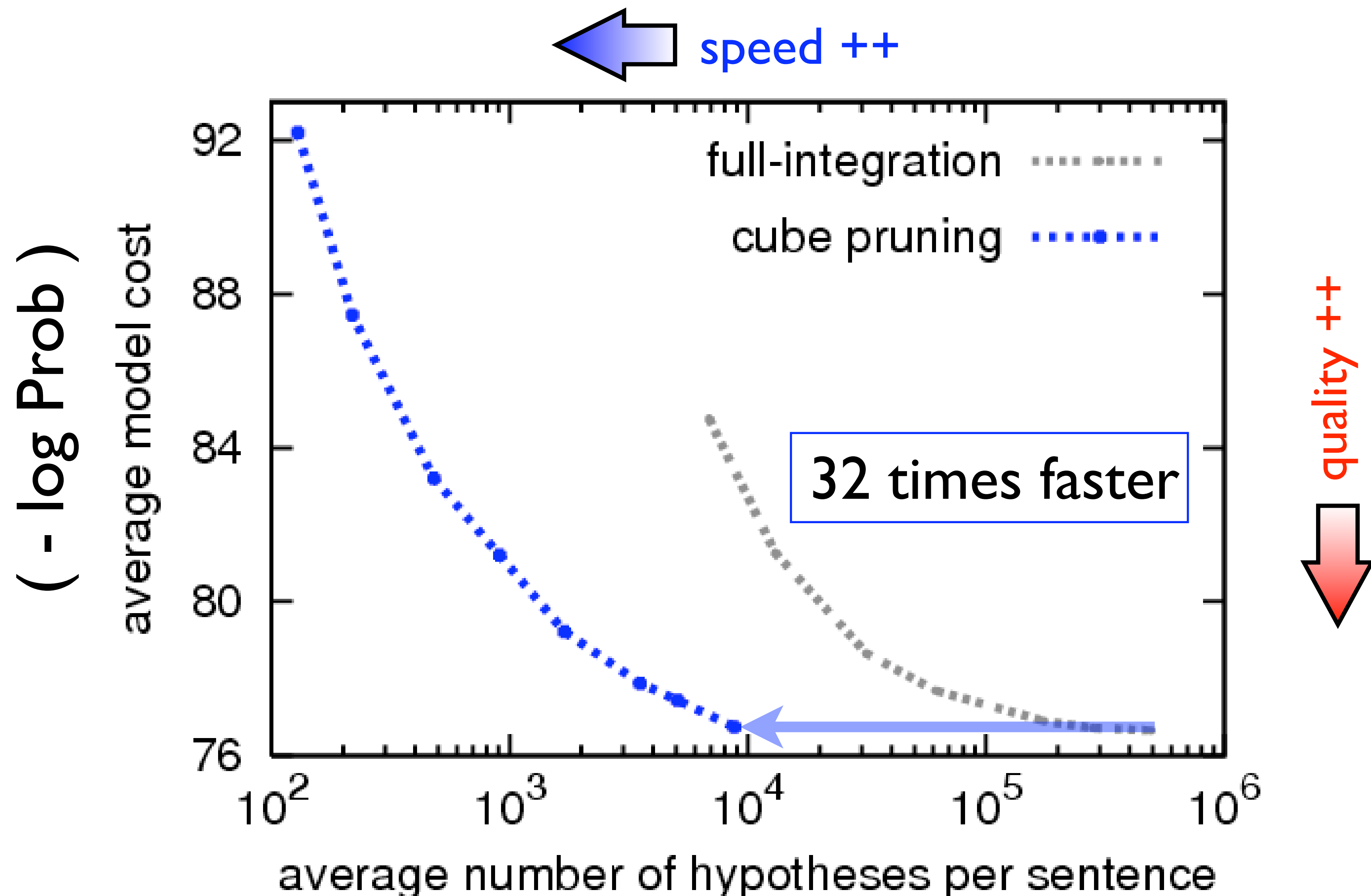
Speed vs. Search Quality

tested on our faithful clone of Pharaoh



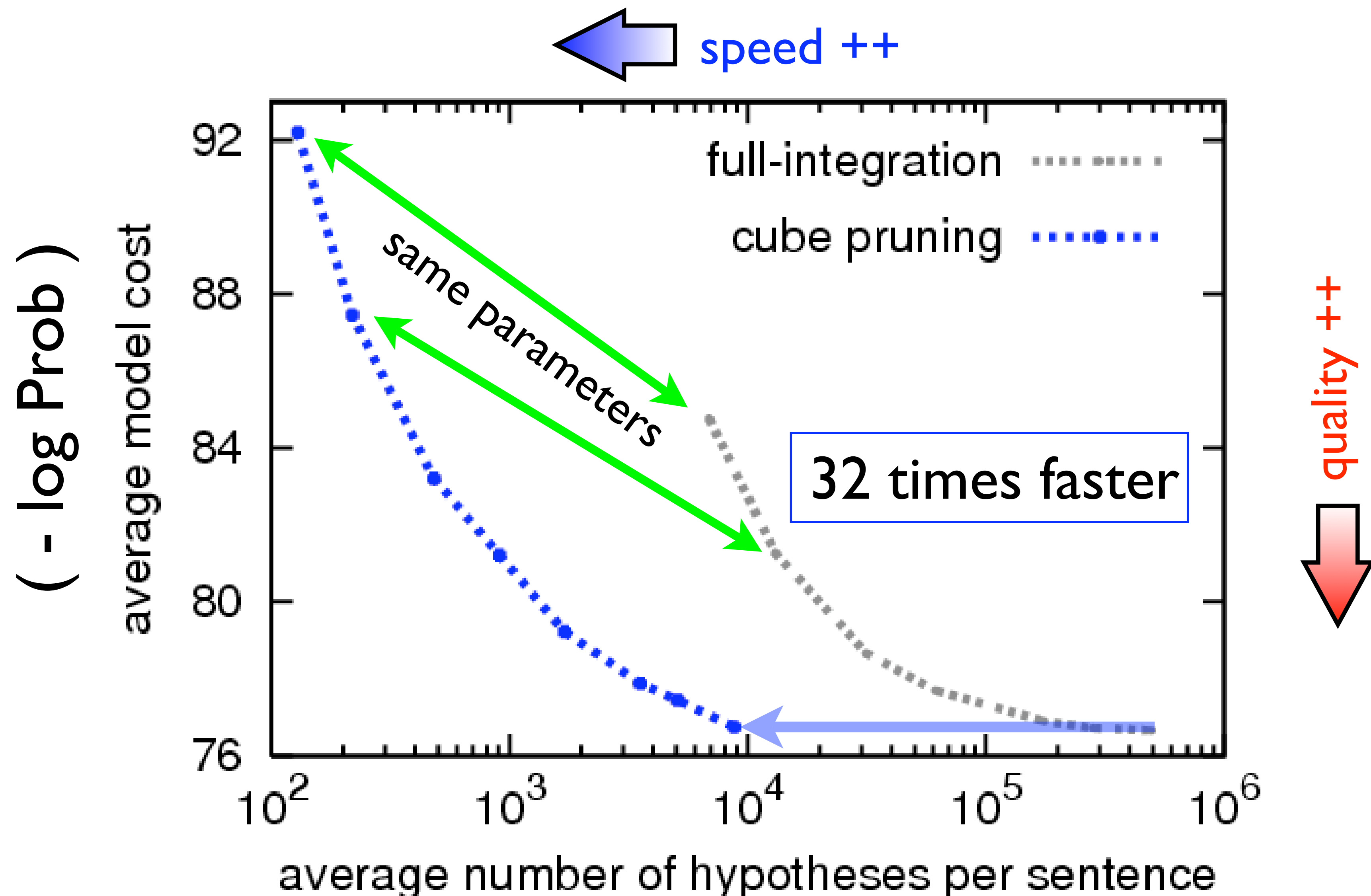
Speed vs. Search Quality

tested on our faithful clone of Pharaoh

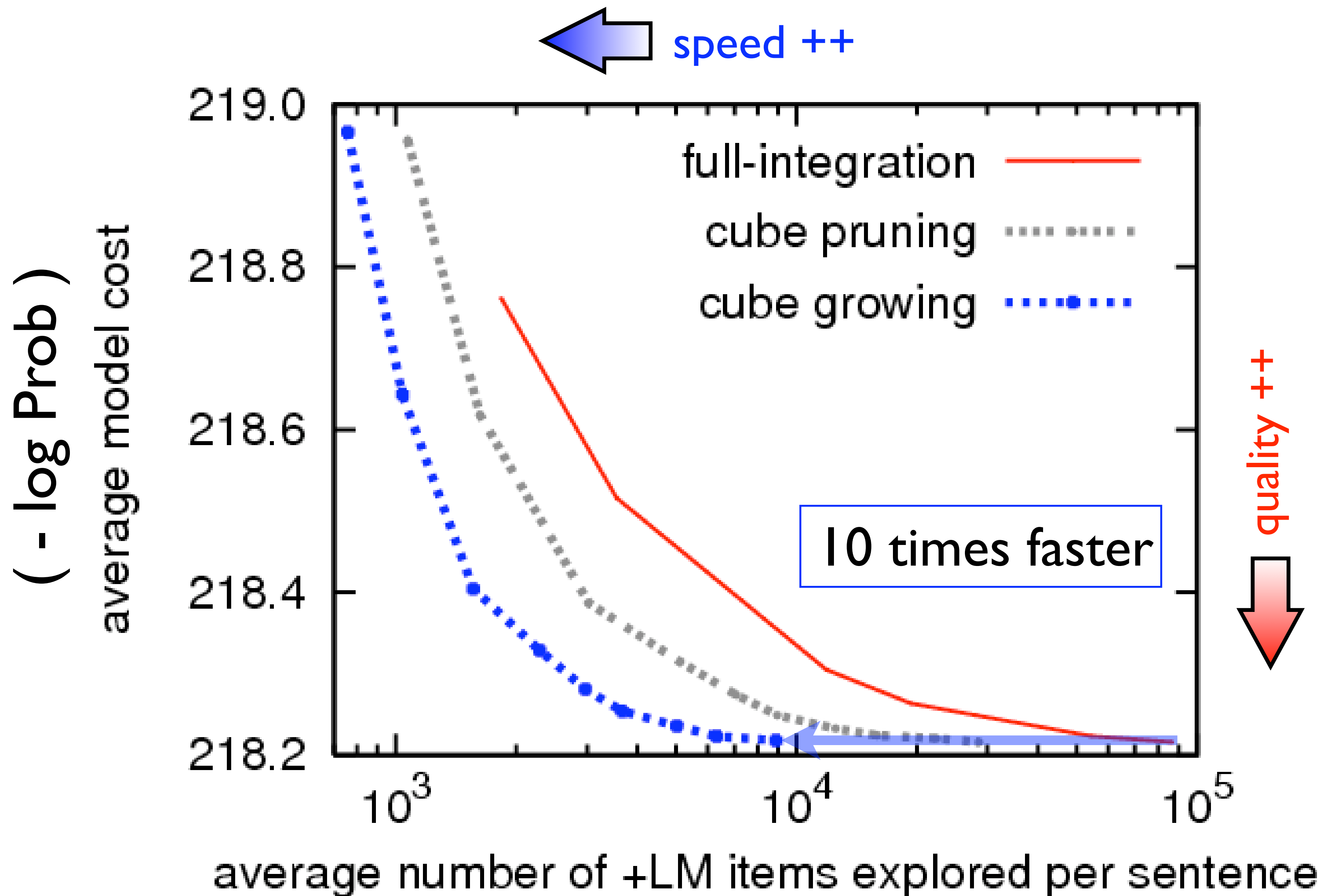


Speed vs. Search Quality

tested on our faithful clone of Pharaoh

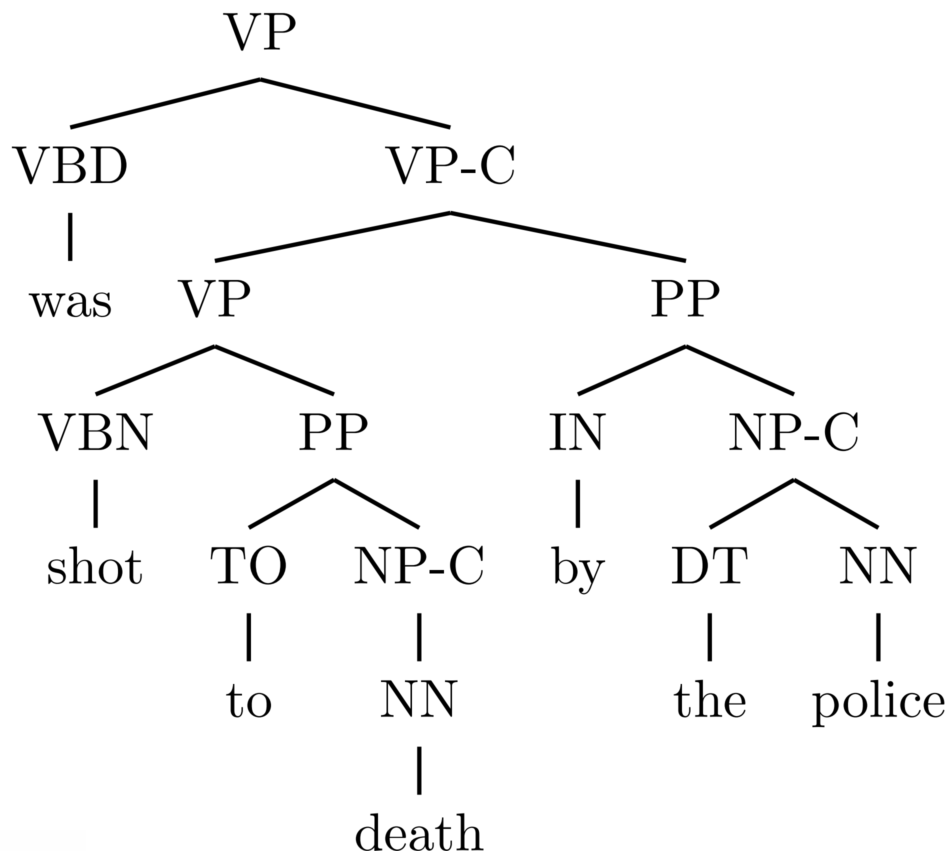


Syntax-based: Search Quality



Tree-to-String System

- syntax-directed, English to Chinese (Huang, Knight, Joshi, 2006)
- first parse input, and then recursively transfer



synchronous tree-substitution grammars (STSG)

(Galley et al., 2004; Eisner, 2003)

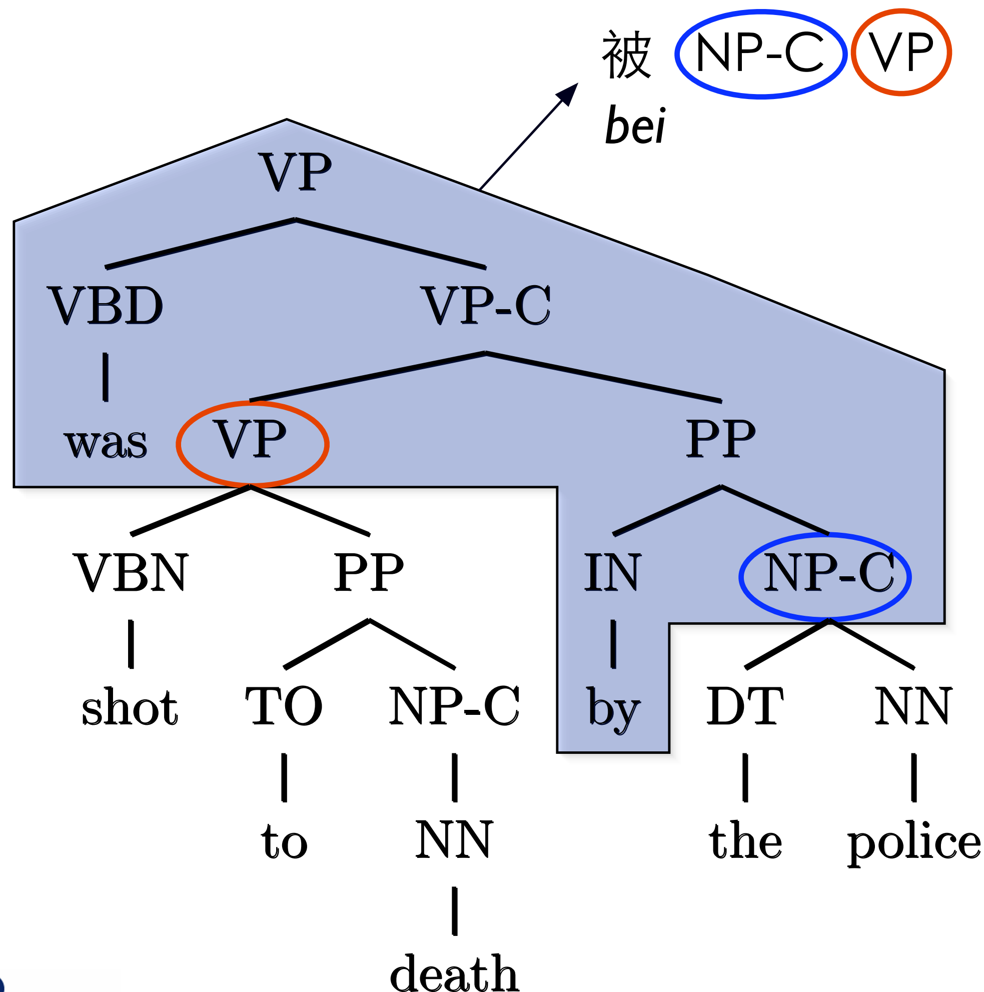
extended to translate a packed-forest instead of a tree

(Mi, Huang, Liu, 2008)

Penn UNIVERSITY of PENNSYLVANIA

Tree-to-String System

- syntax-directed, English to Chinese (Huang, Knight, Joshi, 2006)
- first parse input, and then recursively transfer



synchronous tree-substitution grammars (STSG)

(Galley et al., 2004; Eisner, 2003)

extended to translate a packed-forest instead of a tree

(Mi, Huang, Liu, 2008)

Penn University of Pennsylvania

Features

- extract features on the 50-best parses of train set
- cut off low-freq. features with count < 5
  - counts are "relative" -- change on at least 5 sentences
- feature templates
  - 4 local from (Charniak and Johnson, 2005)
  - 4 local from (Collins, 2000)
  - 7 non-local from (Charniak and Johnson, 2005)
- 800, 582 feature instances (30% non-local)
  - cf. C & J: 1.3 M feature instances (60% non-local)

Forest Oracle

the candidate tree that is closest to gold-standard

Optimal Parseval F-score

$$y_i^+ \triangleq \operatorname*{argmax}_{y \in cand(s_i)} F(y, y_i^*) \qquad F(y, y^*) \triangleq \frac{2PR}{P+R} = \frac{2|y \cap y^*|}{|y| + |y^*|}$$

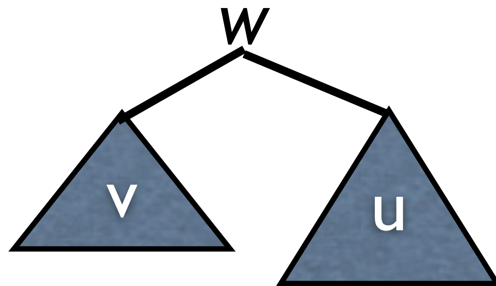
- Parseval F_1-score is the harmonic mean between labeled precision and labeled recall
    - can not optimize F-scores on sub-forests separately
- we instead use dynamic programming
    - optimizes the number of matched brackets per given number of test brackets
    - "when the test (sub-) parse has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

$$ora[v] : \mathbb{N} \mapsto \mathbb{N} \qquad ora[v](t) \triangleq \max_{y_v : |y_v| = t} |y_v \cap y^*|$$

Penn
University *of* Pennsylvania

Combining Oracle Functions

- combining two oracle functions along a hyperedge $e = \langle (v,u), w \rangle$ needs a convolution operator $\otimes$



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

t	f(t)
2	1
3	2

$\otimes$

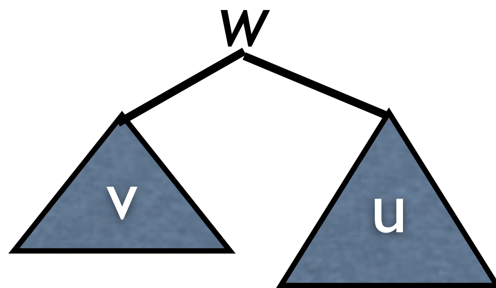
t	g(t)
4	4
5	4

=

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

Combining Oracle Functions

- combining two oracle functions along a hyperedge $e = \langle(v,u), w\rangle$ needs a convolution operator $\otimes$



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

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

N

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

ora[*w*]

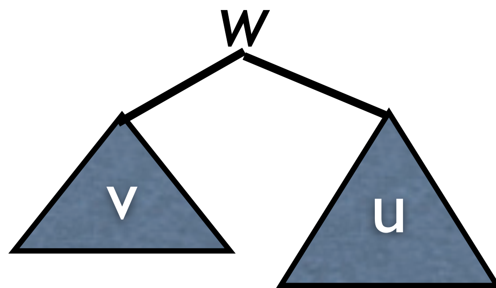
this node matched?

Y

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

Combining Oracle Functions

- combining two oracle functions along a hyperedge $e = \langle (v,u), w \rangle$ needs a convolution operator $\otimes$



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

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:

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

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 Pruning

a variant of Inside-Outside Algorithm

Pruning (J. Graehl, unpublished)

- prune by marginal probability (Charniak and Johnson, 2005)
  - but we prune hyperedges as well as nodes
- compute Viterbi inside cost $\beta(v)$ and outside cost $\alpha(v)$
- compute merit $\alpha\beta(e) = \alpha(\text{head}(e)) + \text{sum}_{u \in \text{tails}(e)}\, \beta(u)$
  - cost of the best derivation that traverses e
- prune away hyperedges that have $\alpha\beta(e) - \beta(\text{TOP}) > p$
- difference: a node can "partially" survive the beam
- can prune on average 15% more hyperedges than C&J

Penn
University of Pennsylvania