TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel

Spring 2017

Lecture 15:
Structured Prediction

- No class Monday May 29 (Memorial Day)
- Final class is Wednesday May 31

- Assignment 3 has been posted, due Thursday June 1
- Final project report due Friday, June 9

Modeling, Inference, Learning

inference: solve argmax

modeling: define score function

$$\text{classify}(x, \boldsymbol{\theta}) = \underset{y}{\text{argmax}}\ \text{score}(x, y, \boldsymbol{\theta})$$

learning: choose $\boldsymbol{\theta}$

Structured Prediction:

size of output space is exponential in size of input or is unbounded (e.g., machine translation)

(we can't just enumerate all possible outputs)

- 2 categories of structured prediction:
  score-based and search-based

Score-Based Structured Prediction

- focus on defining the score function of the structured input/output pair:

$$\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

- cleanly separates score function, inference algorithm, and training loss

Modeling in Score-Based SP

- define score as a sum or product over "parts" of the structured input/output pair:

$$\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \sum_{\langle \boldsymbol{x}_r, \boldsymbol{y}_r \rangle \in \mathrm{parts}(\boldsymbol{x}, \boldsymbol{y})} \mathrm{score}(\boldsymbol{x}_r, \boldsymbol{y}_r, \boldsymbol{\theta})$$

$$\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \prod_{\langle \boldsymbol{x}_r, \boldsymbol{y}_r \rangle \in \mathrm{parts}(\boldsymbol{x}, \boldsymbol{y})} \mathrm{score}(\boldsymbol{x}_r, \boldsymbol{y}_r, \boldsymbol{\theta})$$

Parts Functions in Score-Based SP

- for an HMM:

$$\mathrm{parts}_{\mathrm{HMM}}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle x_t, y_t \rangle\}_{t=1}^{T} \cup \{\langle \emptyset, y_{t-1:t} \rangle\}_{t=1}^{T}$$

- each word-label pair forms a part, and each label bigram forms a part

Parts Functions in Score-Based SP

- for a linear chain CRF:

$$\text{parts}_{\text{chainCRF}}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle \boldsymbol{x}, y_{t-1:t} \rangle\}_{t=1}^{T}$$

- each label bigram forms a part (each of which includes entire input!)

Parts Functions in Score-Based SP

- for a PCFG:

$$\mathrm{parts}_{\mathrm{PCFG}}(\boldsymbol{x}, \boldsymbol{y}) = \left(\bigcup_{(y \to x) \in \boldsymbol{y}} \langle x, y \rangle \right) \cup \left(\bigcup_{(y \to y_1, y_2) \in \boldsymbol{y}} \langle \emptyset, \langle y, y_1, y_2 \rangle \rangle \right)$$

- each context-free grammar rule forms a part

Parts Functions in Score-Based SP

- for an arc-factored dependency parser:

$$\text{parts}_{\text{arcdep}}(\boldsymbol{x}, \boldsymbol{y}) = \{\langle \boldsymbol{x}, y_t \rangle\}_{t=1}^{T}$$

where y_t is the index of the parent of x_t in $\boldsymbol{y}$

- each dependency arc forms a part

Inference in Score-Based SP

- inference algorithms are defined based on decomposition of score into parts

$$\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \sum_{\langle \boldsymbol{x}_r, \boldsymbol{y}_r \rangle \in \mathrm{parts}(\boldsymbol{x}, \boldsymbol{y})} \mathrm{score}(\boldsymbol{x}_r, \boldsymbol{y}_r, \boldsymbol{\theta})$$

- smaller parts = easier to define efficient exact inference algorithms

Inference Algorithms for Score-Based SP

- exact inference algorithms are often based on **dynamic programming**

Dynamic Programming (DP)

- a class of algorithms that break problems into smaller pieces and reuse solutions for pieces
  - applicable if problem has certain properties (**optimal substructure** and **overlapping sub-problems**)

- in NLP, we use DP to iterate over exponentially-large output spaces in polynomial time
  - Viterbi and forward/backward for HMMs
  - CKY for PCFGs
  - Eisner algorithm for (arc-factored) dependency parsing

Viterbi Algorithm

- recursive equations + memoization:

base case:
returns score of sequence starting with label *y* for first word

$$V(1, y) = \mathrm{score}(x_1, y)\ \mathrm{score}(\emptyset, \langle, \text{<s>}, y\rangle)$$
$$V(t, y) = \max_{y' \in \mathcal{L}} \left(\mathrm{score}(x_t, y)\ \mathrm{score}(\emptyset, \langle y', y\rangle)\ V(t-1, y') \right)$$

recursive case:
computes score of max-scoring label sequence
that ends with label *y* at position *t*

final value is in: $V(|\boldsymbol{x}| + 1, \text{</s>})$

Viterbi Algorithm

- space and time complexity?
- can be read off from the recursive equations:

space complexity:
size of memoization table, which is # of unique indices of recursive equations

length of sentence ***** **number of labels**

$$V(t, y) = \max_{y' \in \mathcal{L}} \left(\mathrm{score}(x_t, y) \, \mathrm{score}(\emptyset, \langle y', y \rangle) \, V(t-1, y') \right)$$

so, space complexity is O(|*x*| |*L*|)

Viterbi Algorithm

- space and **time** complexity?
- can be read off from the recursive equations:

time complexity:
size of memoization table * complexity of computing each entry

length of sentence * **number of labels** * **each entry requires iterating through the labels**

$$V(t, y) = \max_{y' \in \mathcal{L}} \left(\mathrm{score}(x_t, y)\, \mathrm{score}(\emptyset, \langle y', y \rangle)\, V(t-1, y') \right)$$

so, time complexity is O(|*x*| |*L*| |*L*|) = O(|*x*| |*L*|^2)

Feature Locality

- **feature locality**: how big are the parts?
- for efficient inference (w/ DP or other methods), we need to be mindful of this
- parts can be arbitrarily big in terms of input, but not in terms of *output*!
- HMM parts are small in both the input and output (only two pieces at a time)

Learning with Score-Based SP: Empirical Risk Minimization

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathcal{D}} \operatorname{cost}(\boldsymbol{y}, \operatorname{predict}(\boldsymbol{x}, \boldsymbol{\theta}))$$

$$\operatorname{predict}(\boldsymbol{x}, \boldsymbol{\theta}) = \underset{\boldsymbol{y}}{\operatorname{argmax}} \ \operatorname{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

Cost Functions

- **cost function**: how different are these two structures?

$$\text{cost} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$$

- typically used to compare predicted structure to gold standard
- should reflect evaluation metric for task

- usual conventions: $\text{cost}(\boldsymbol{y}, \boldsymbol{y}) = 0$

$$\text{cost}(\boldsymbol{y}, \boldsymbol{y}') = \text{cost}(\boldsymbol{y}', \boldsymbol{y})$$

Cost Functions

- for classification, we used:

$$\mathrm{cost}(y, y') = \mathbb{I}[y \neq y']$$

- how about for sequences?
  - "Hamming cost": $$\mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}') = \sum_{t=1}^{|\boldsymbol{y}|} \mathbb{I}[y_t \neq y'_t]$$
  - "0-1 cost": $$\mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}') = \mathbb{I}[\boldsymbol{y} \neq \boldsymbol{y}']$$

Empirical Risk Minimization

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathcal{D}} \operatorname{cost}(\boldsymbol{y}, \operatorname{predict}(\boldsymbol{x}, \boldsymbol{\theta}))$$

$$\operatorname{predict}(\boldsymbol{x}, \boldsymbol{\theta}) = \underset{\boldsymbol{y}}{\operatorname{argmax}} \ \operatorname{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$$

- this is intractable so we typically minimize a **surrogate loss function** instead

Loss Functions for Score-Based SP

name	loss	where used
cost ("0-1")	$\text{cost}(\boldsymbol{y}, \text{predict}(\boldsymbol{x}, \boldsymbol{\theta}))$	
percep-tron		
hinge		

Loss Functions for Score-Based SP

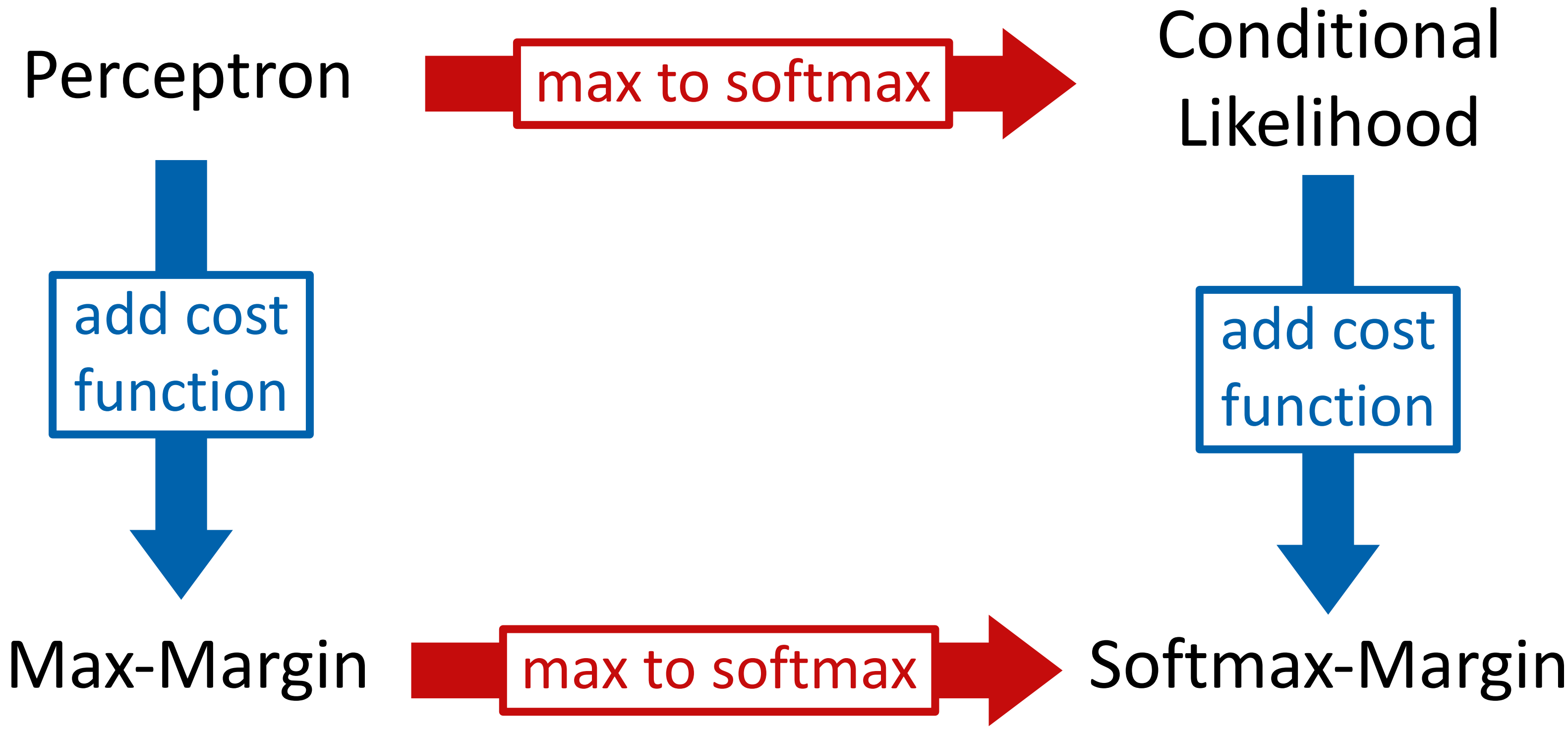
name	loss	where used
cost ("0-1")	$\mathrm{cost}(\boldsymbol{y}, \mathrm{predict}(\boldsymbol{x}, \boldsymbol{\theta}))$	MERT (Och, 2003)
percep-tron	$-\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} \ \mathrm{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta})$	structured perceptron (Collins, 2002)
hinge	$-\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} \ (\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta}) + \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}'))$	structured SVMs (Taskar et al., *inter alia*)

Loss Functions for Score-Based SP

name	loss	where used
cost ("0-1")	$\text{cost}(\boldsymbol{y}, \text{predict}(\boldsymbol{x}, \boldsymbol{\theta}))$	MERT (Och, 2003)
percep-tron	$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} \text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta})$	structured perceptron (Collins, 2002)
hinge	$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} (\text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta}) + \text{cost}(\boldsymbol{y}, \boldsymbol{y}'))$	structured SVMs (Taskar et al., *inter alia*)
log	$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \log \sum_{\boldsymbol{y}'} \exp \{\text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta})\}$	CRFs (Lafferty et al., 2001)

Loss Functions for Score-Based SP

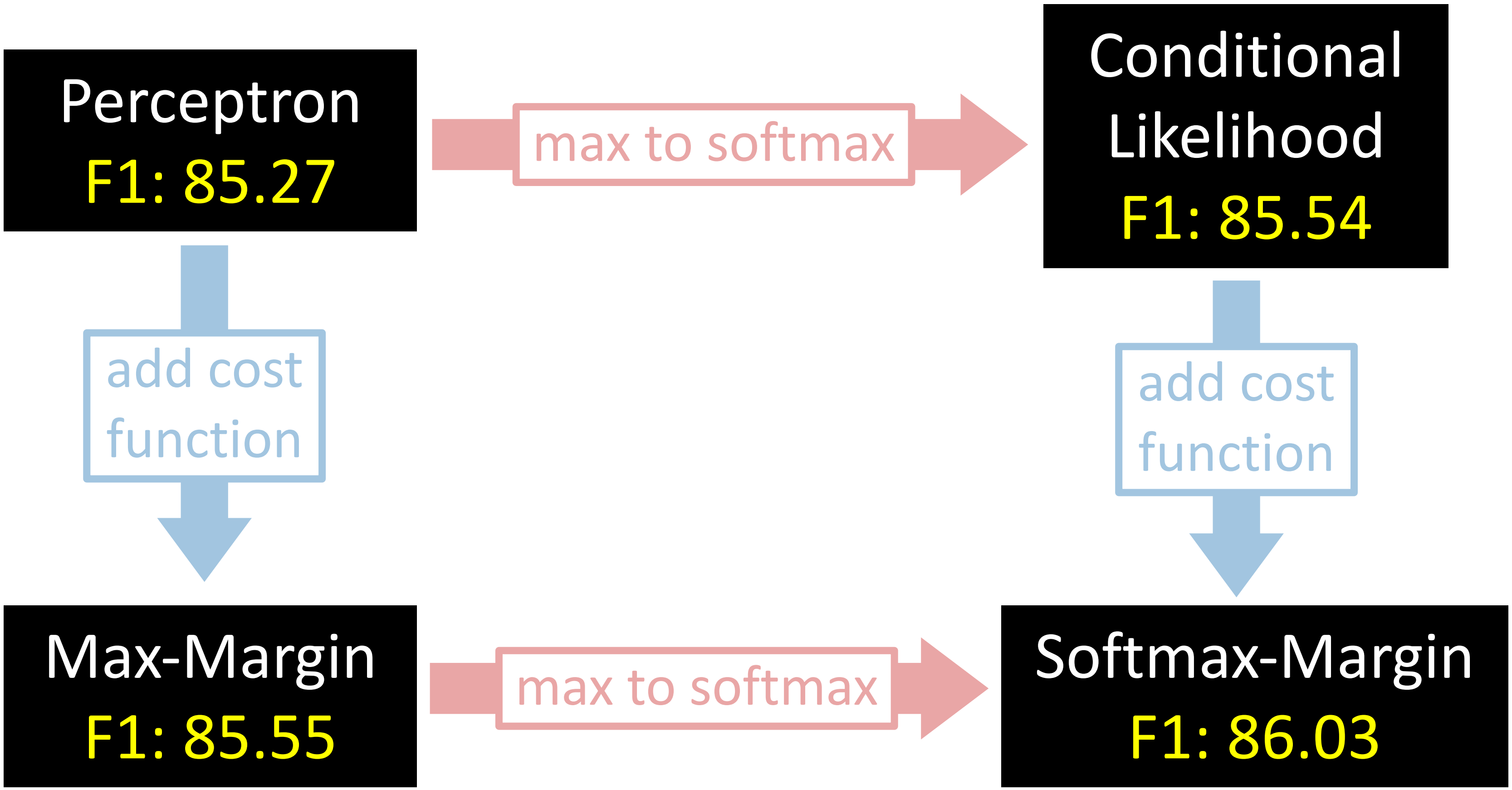
name	loss	where used
cost ("0-1")	$\mathrm{cost}(\boldsymbol{y}, \mathrm{predict}(\boldsymbol{x}, \boldsymbol{\theta}))$	MERT (Och, 2003)
percep-tron	$-\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} \ \mathrm{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta})$	structured perceptron (Collins, 2002)
hinge	$-\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \max_{\boldsymbol{y}'} \ (\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta}) + \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}'))$	structured SVMs (Taskar et al., *inter alia*)
log	$-\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \log \sum_{\boldsymbol{y}'} \ \exp\left\{\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta})\right\}$	CRFs (Lafferty et al., 2001)
softma x-margin	$-\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \log \sum_{\boldsymbol{y}'} \ \exp\left\{\mathrm{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta}) + \mathrm{cost}(\boldsymbol{y}, \boldsymbol{y}')\right\}$	Povey et al. (2008), Gimpel & Smith (2010)



$$-\text{score}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) + \log \sum_{\boldsymbol{y}'} \exp\left\{\text{score}(\boldsymbol{x}, \boldsymbol{y}', \boldsymbol{\theta}) + \text{cost}(\boldsymbol{y}, \boldsymbol{y}')\right\}$$

Results: Named Entity Recognition

(Gimpel & Smith, 2010)



Inference Algorithms for Score-Based SP

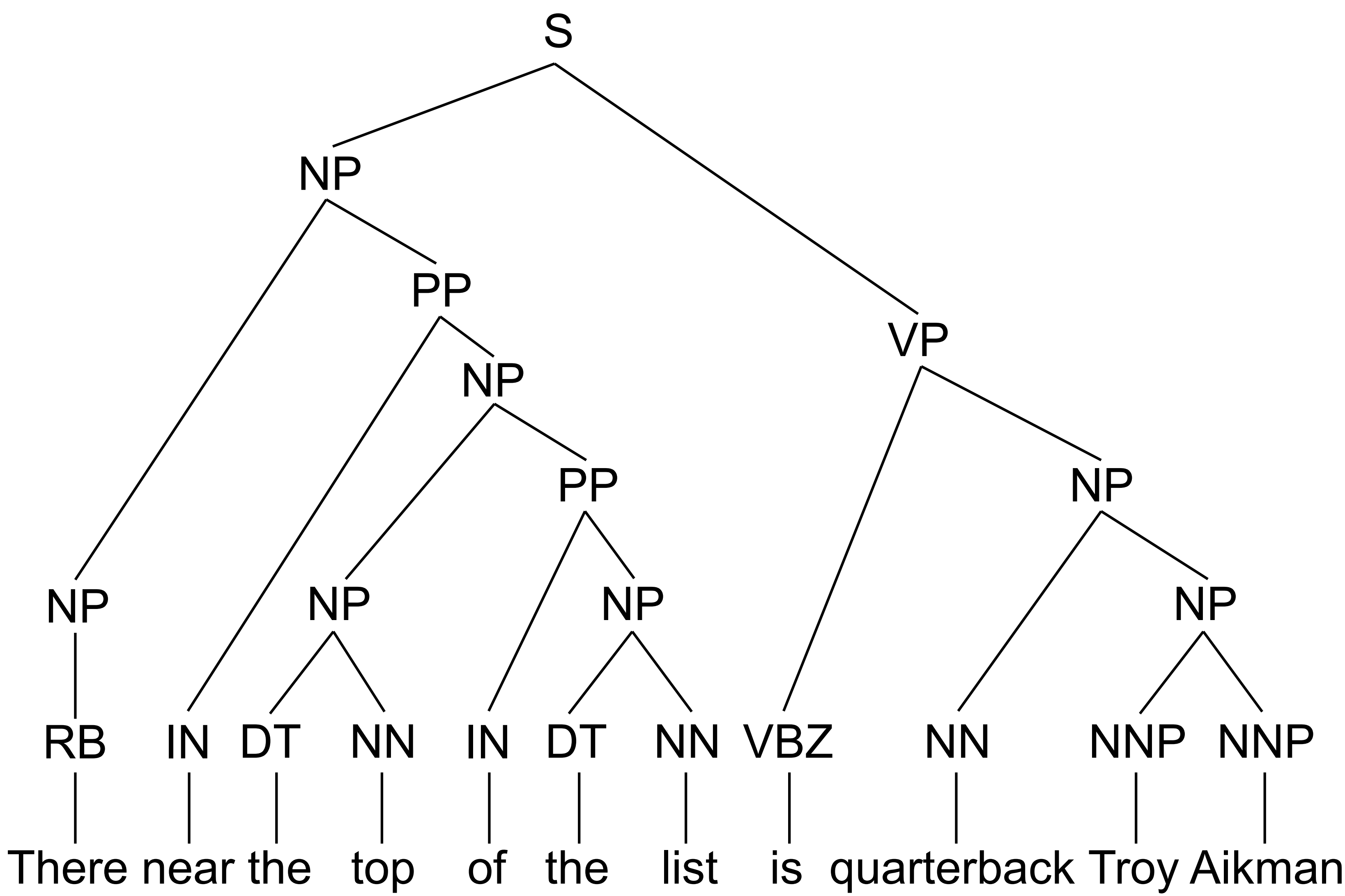
- dynamic programming
  - exact, but parts must be small for efficiency
- dynamic programming + “cube pruning”
  - permits approximate incorporation of large parts (“non-local features”) while still using dynamic program backbone
- integer linear programming

Cube Pruning
(Chiang, 2007; Huang & Chiang, 2007)

- Modification to dynamic programming algorithms for decoding to use non-local features approximately

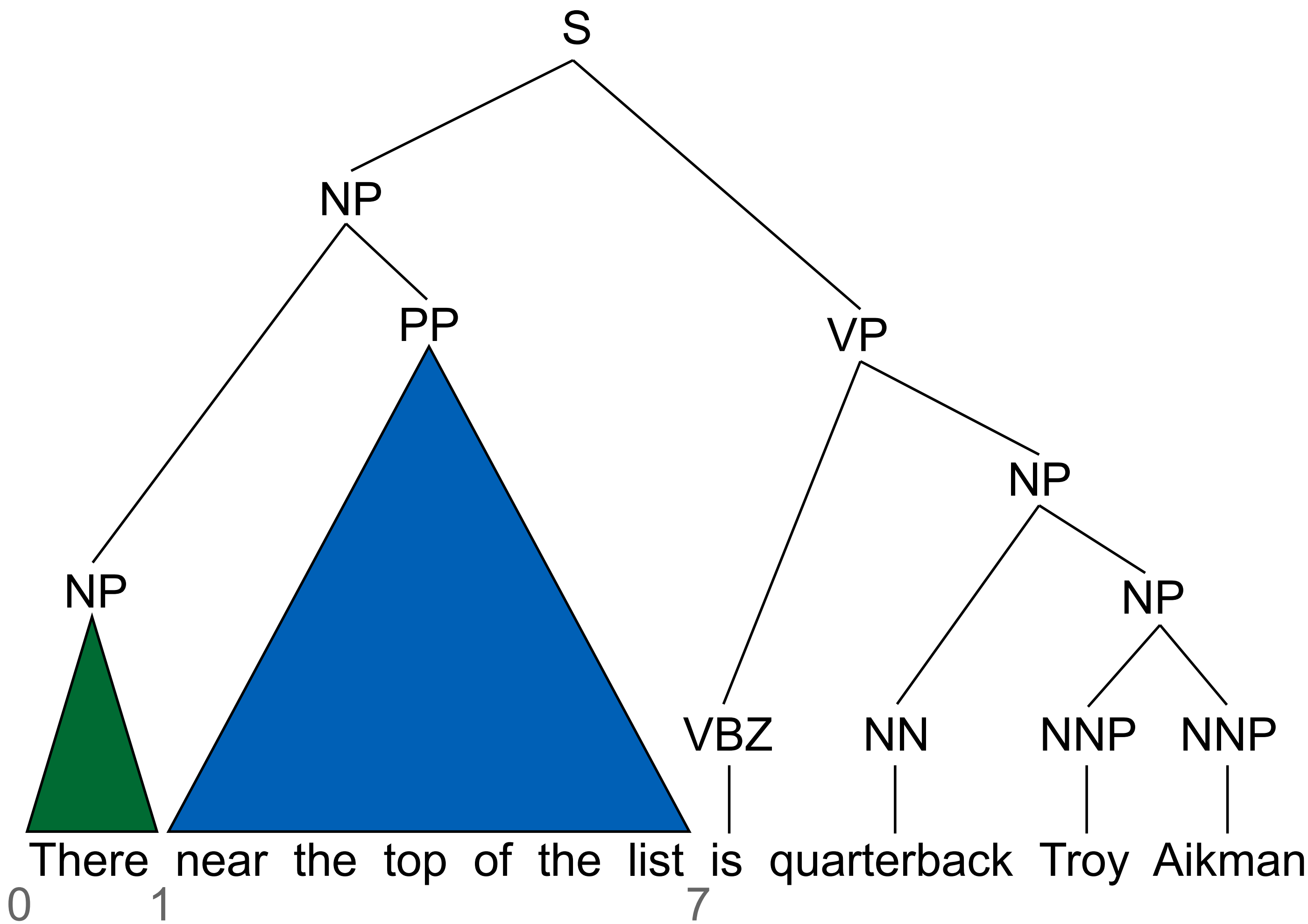
- Keeps a k-best list of derivations for each item

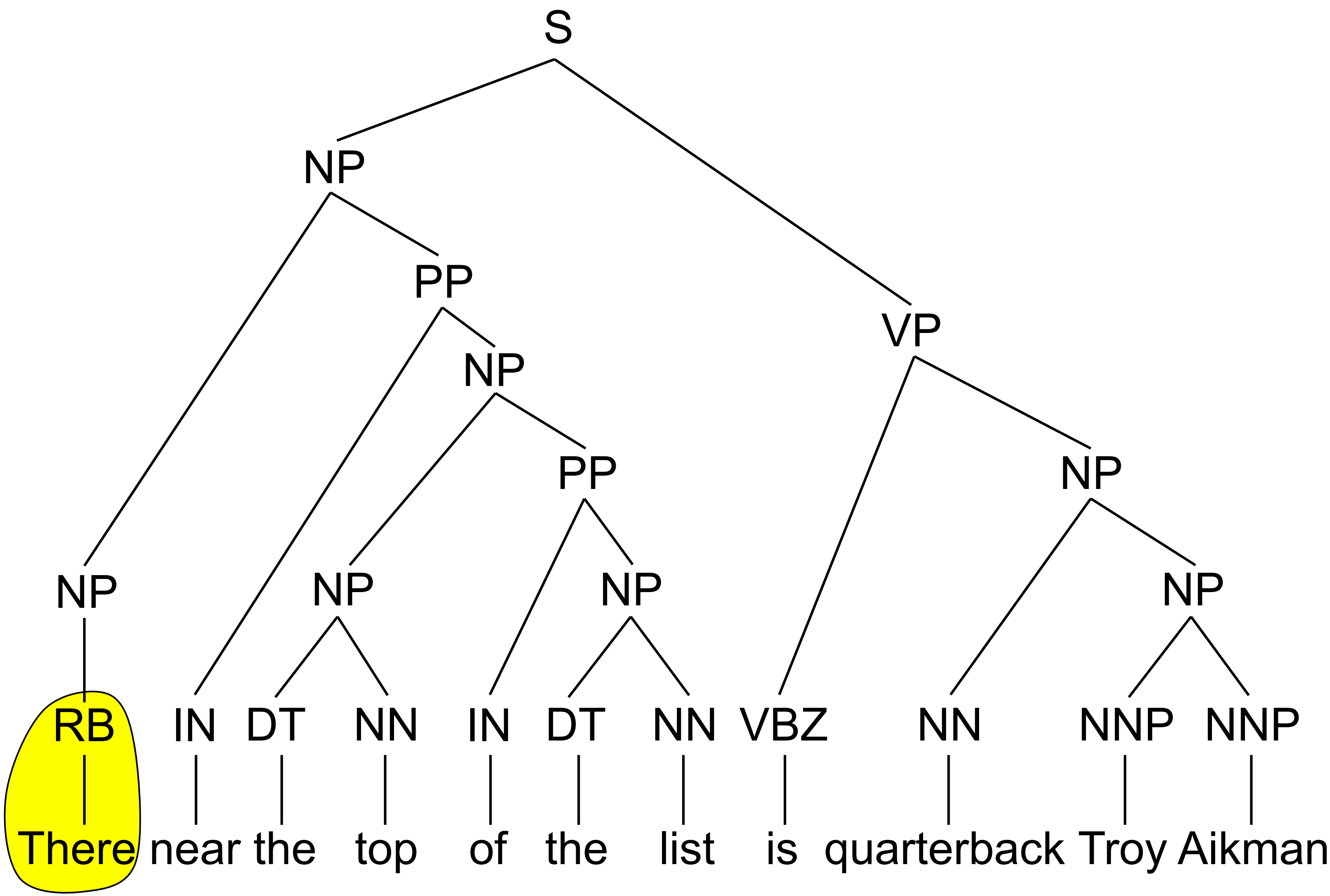
- Applies non-local feature functions on these derivations when defining new items

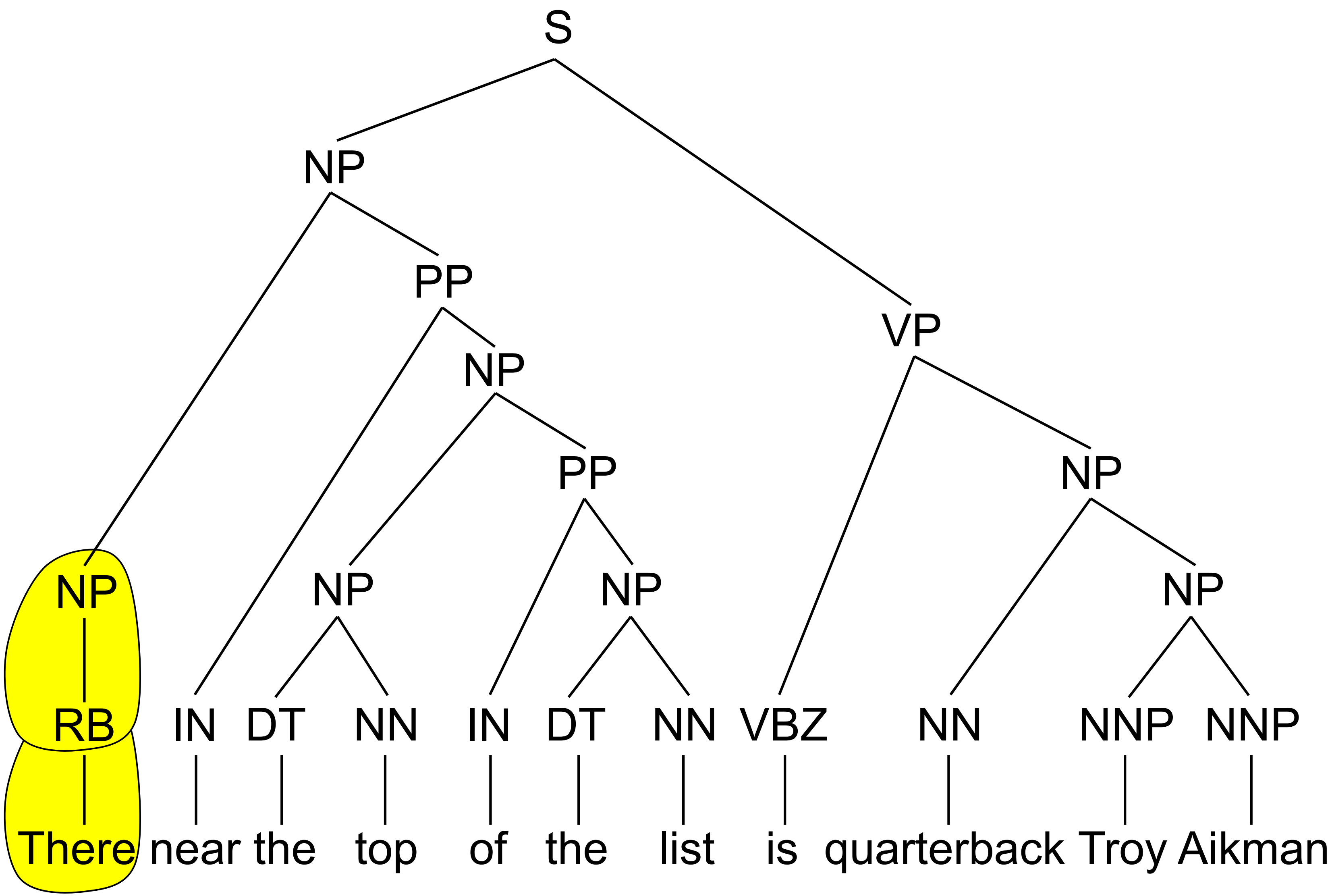

S
NP
VP
PP
NP
NP
PP
NP
NP
NP
NP
NP
RB IN DT NN IN DT NN VBZ NN NNP NNP
There near the top of the list is quarterback Troy Aikman

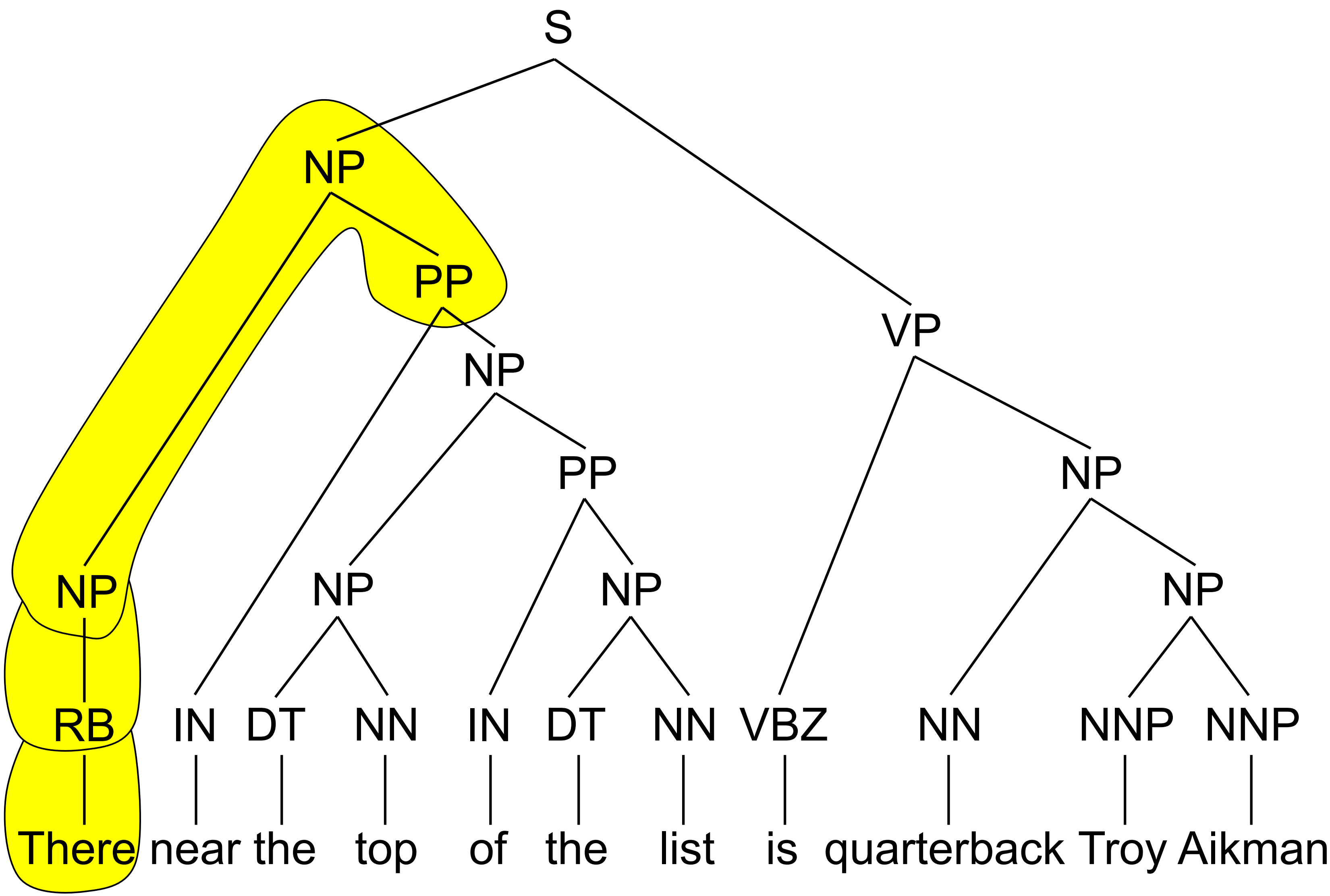
CKY Algorithm

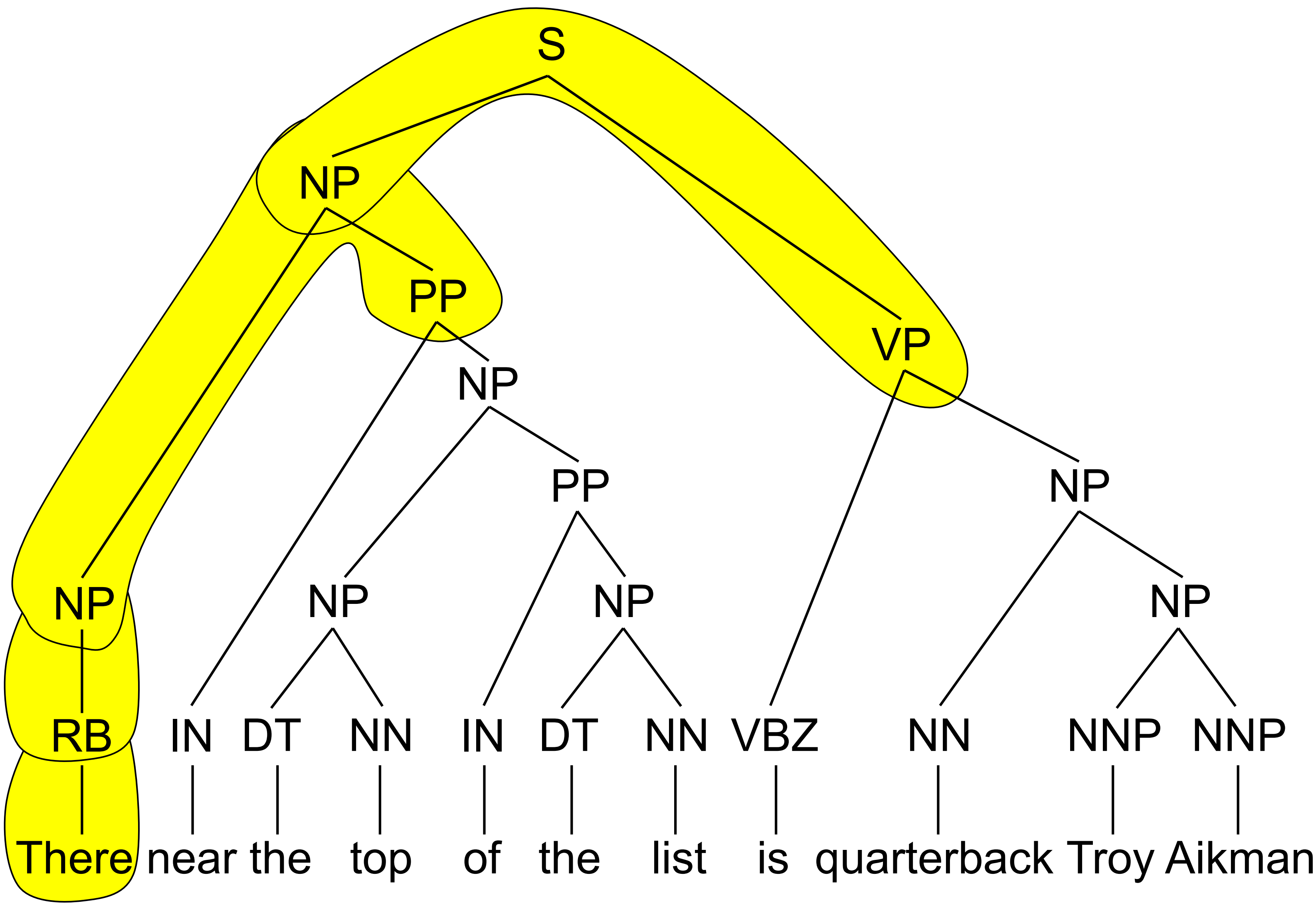
$$C(Z, i, j) = \max_{k} \max_{A,B} \left(C(A, i, k)\, C(B, k, j)\, \mathrm{score}(\langle Z \to A\, B \rangle) \right)$$



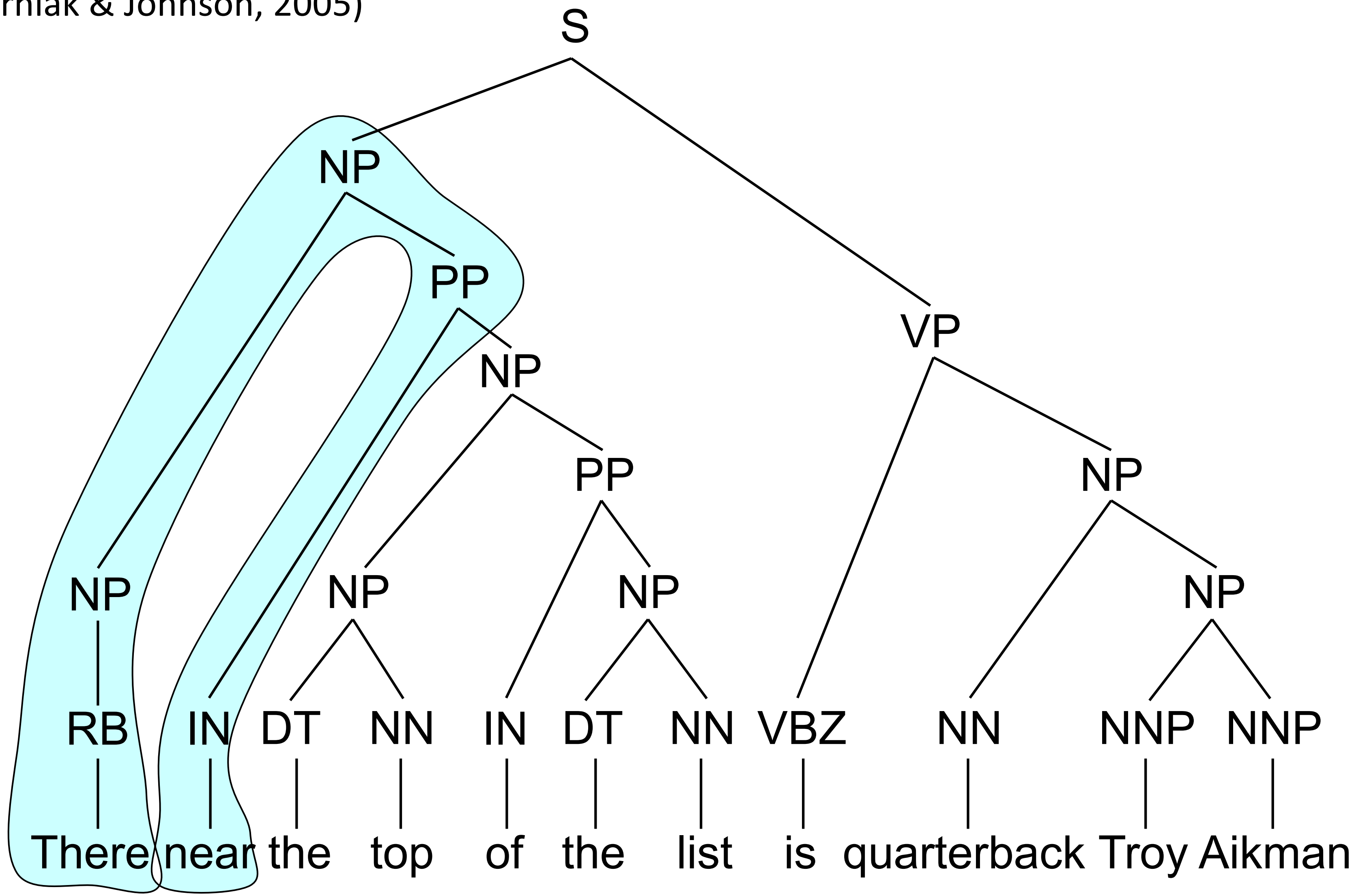

S
NP
PP
VP
NP
PP
NP
NP
NP
NP
NP
RB
IN
DT
NN
IN
DT
NN
VBZ
NN
NNP
NNP
There near the top of the list is quarterback Troy Aikman


S
NP
PP
VP
NP
PP
NP
NP
NP
NP
NP
RB IN DT NN IN DT NN VBZ NN NNP NNP
There near the top of the list is quarterback Troy Aikman

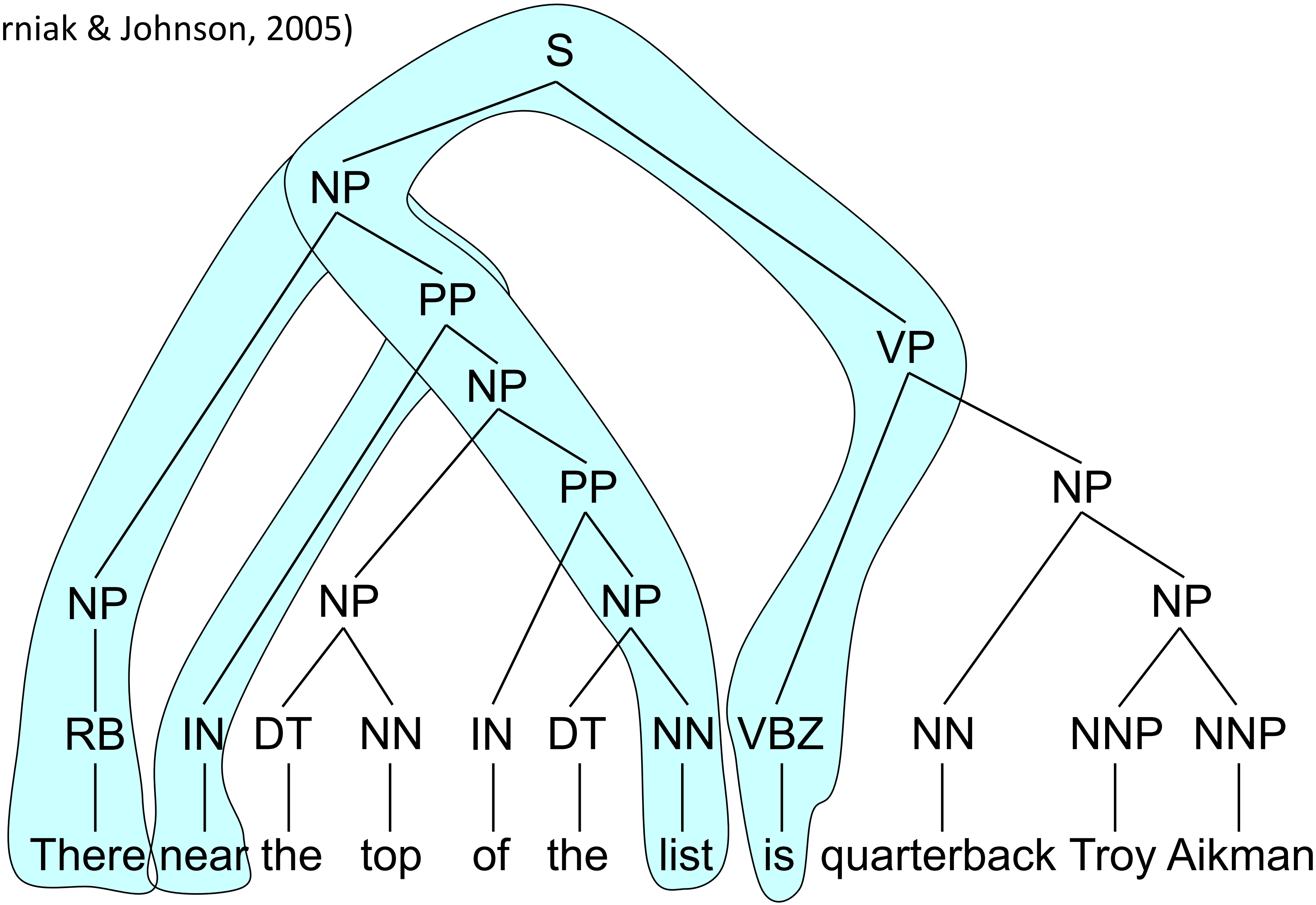

S
NP
VP
PP
NP
PP
NP
NP
NP
NP
NP
RB
IN
DT
NN
IN
DT
NN
VBZ
NN
NNP
NNP
There near the top of the list is quarterback Troy Aikman


S
NP
VP
PP
NP
NP
PP
NP
NP
NP
NP
RB
IN
DT
NN
IN
DT
NN
VBZ
NN
NNP
NNP
There near the top of the list is quarterback Troy Aikman

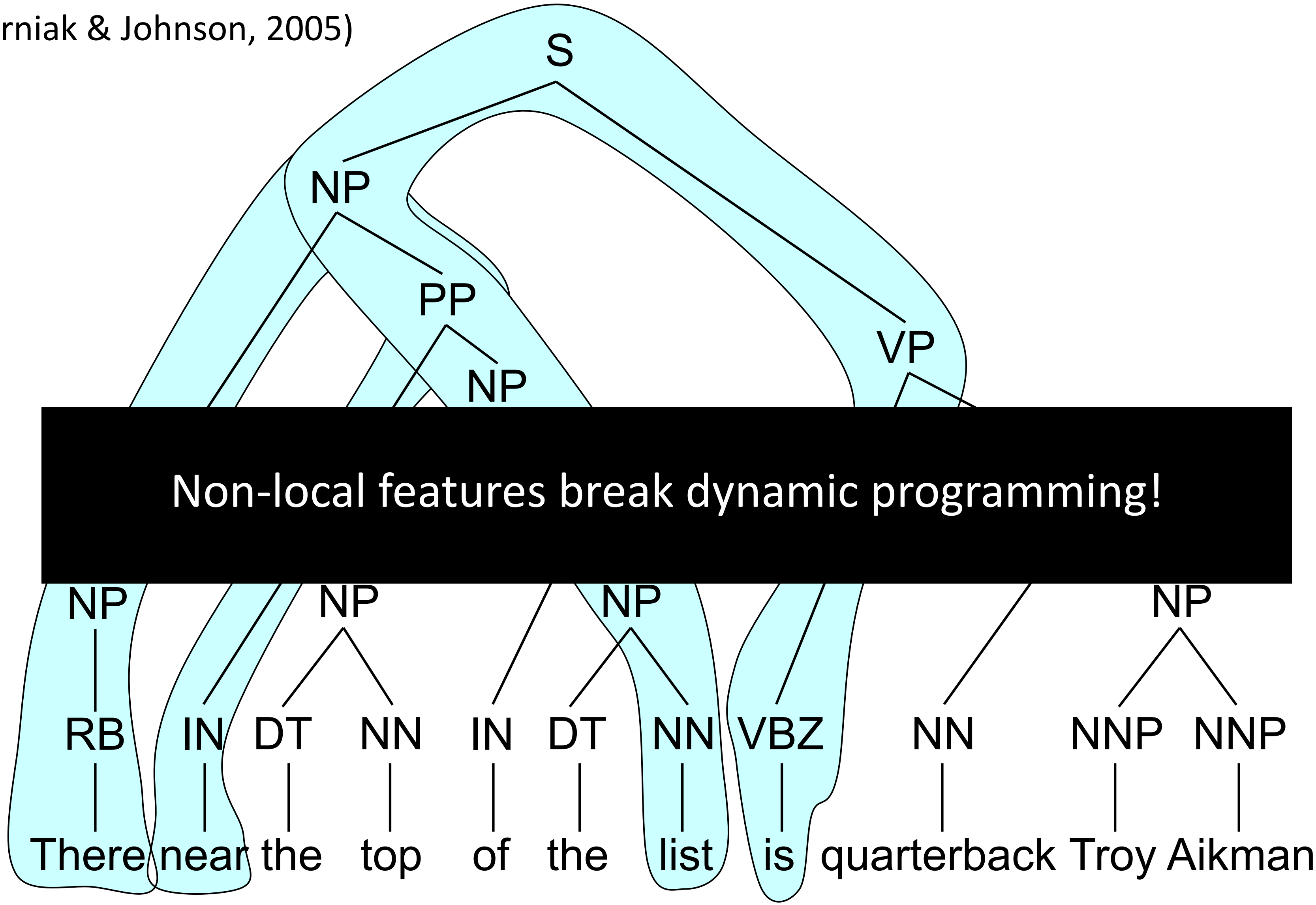
"NGramTree" feature
(Charniak & Johnson, 2005)



“NGramTree” feature
(Charniak & Johnson, 2005)

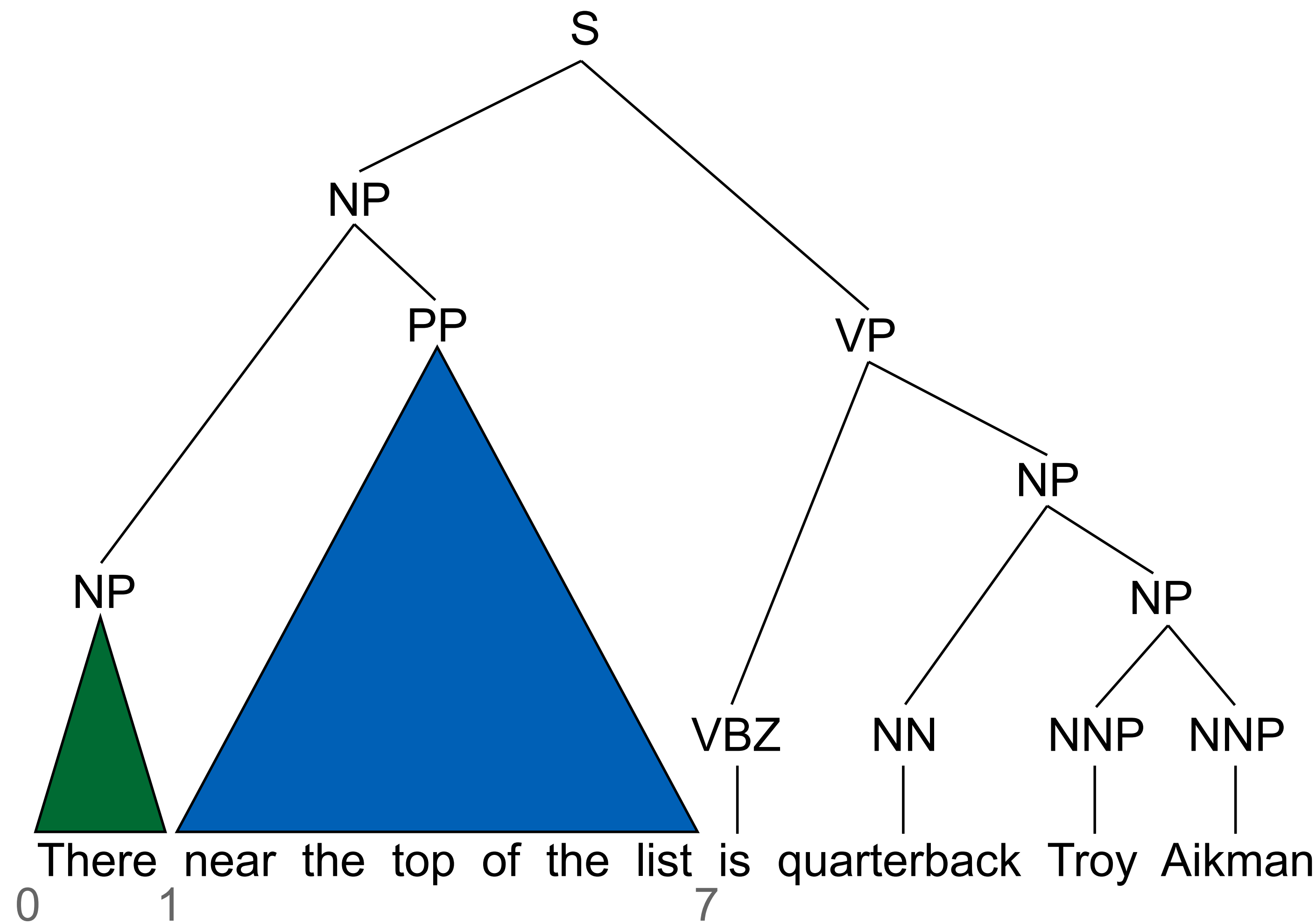


“NGramTree” feature
(Charniak & Johnson, 2005)

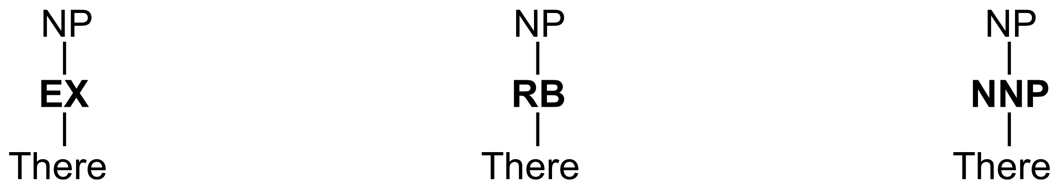


Non-local features break dynamic programming!

$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \to NP\ PP}$$

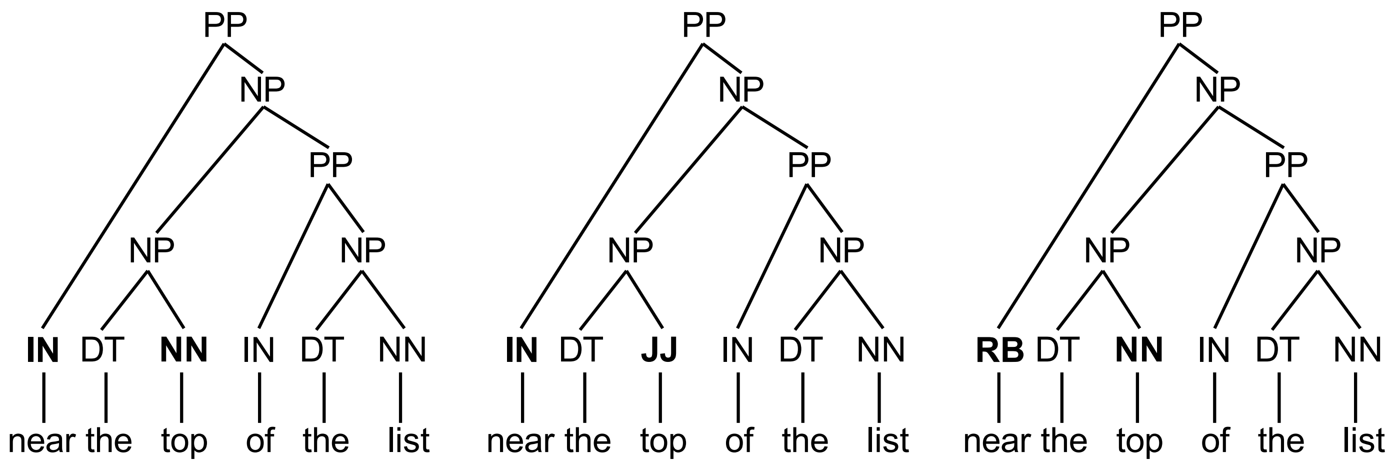


$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$



$C_{NP,0,1} =$

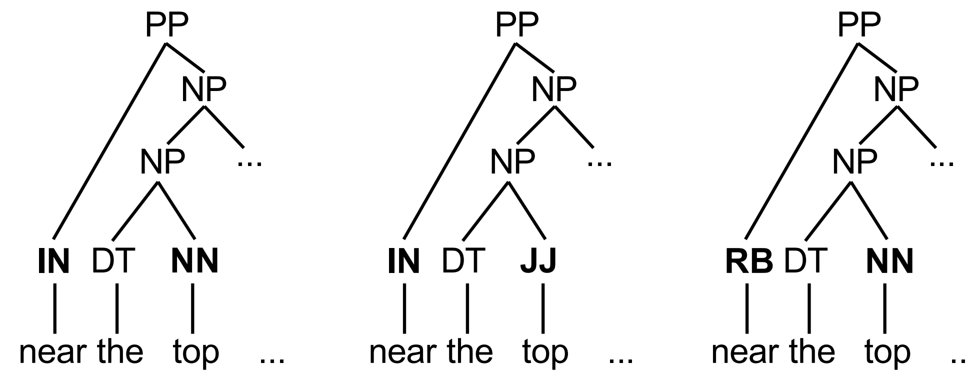
0.4	0.3	0.02

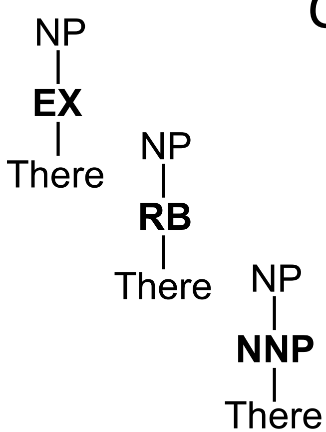


$C_{PP,1,7} =$

0.2	0.1	0.05

$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$

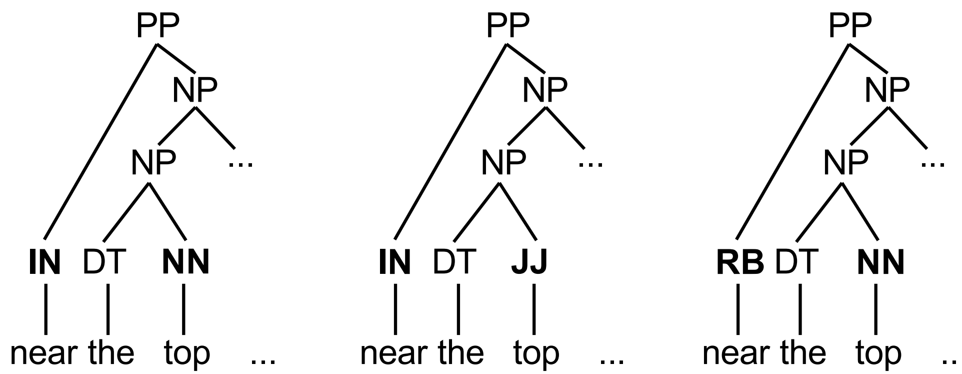


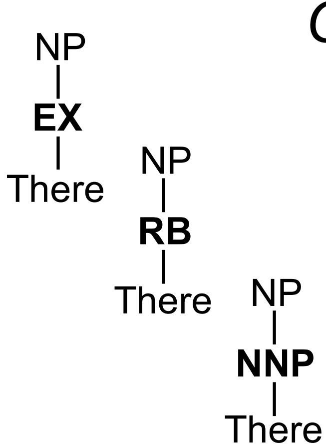


$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.08	0.04	0.02
0.3	0.06	0.03	0.015
0.02	0.004	0.002	0.001

$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$

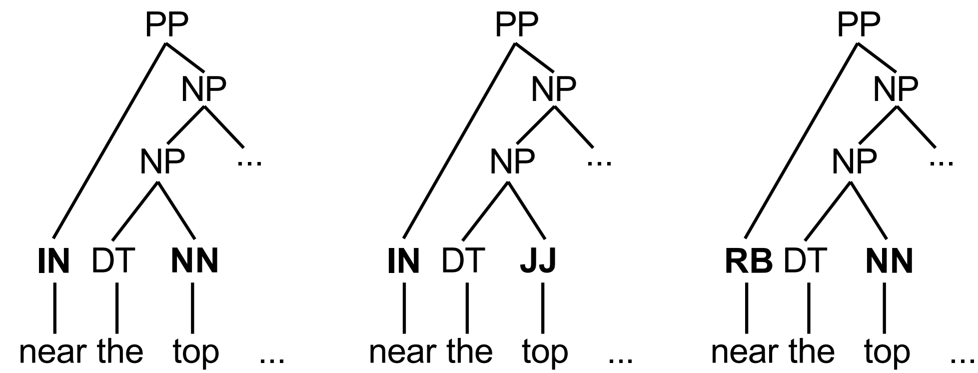
$$\lambda_{NP \rightarrow NP\ PP} = 0.5$$

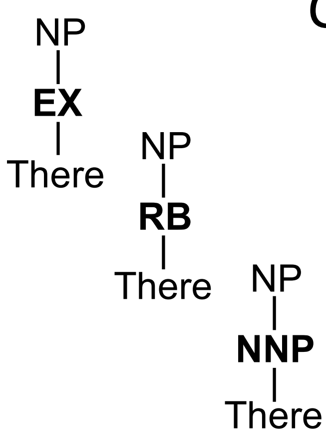




$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.08 × 0.5	0.04 × 0.5	0.02 × 0.5
0.3	0.06 × 0.5	0.03 × 0.5	0.015 × 0.5
0.02	0.004 × 0.5	0.002 × 0.5	0.001 × 0.5

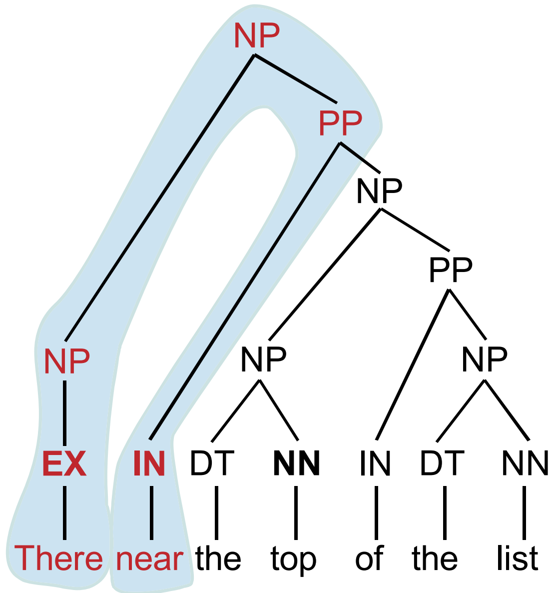
$$C_{NP,0,7} = C_{NP,0,1} \times C_{PP,1,7} \times \lambda_{NP \rightarrow NP\ PP}$$

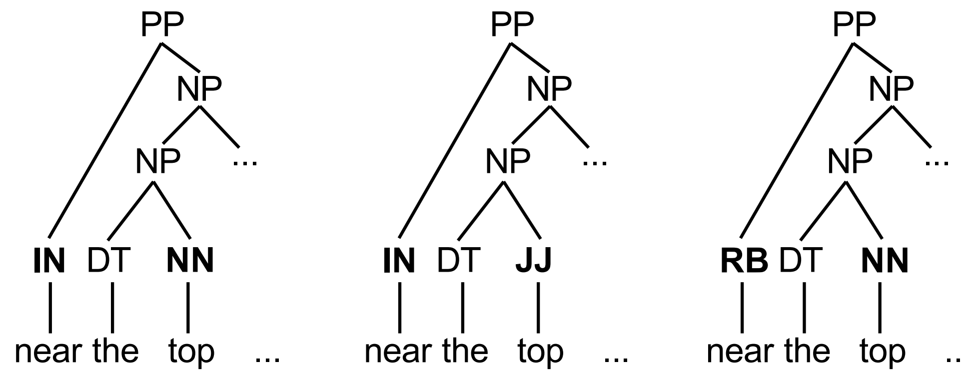


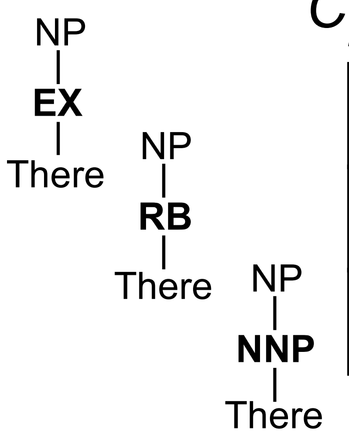


$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.04	0.02	0.01
0.3	0.03	0.015	0.0075
0.02	0.002	0.001	0.0005

$$\lambda_{There\ EX\ NP\ NP\ PP\ IN\ near} = 0.2$$







$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.04 × 0.2	0.02 × 0.2	0.01
0.3	0.03	0.015	0.0075
0.02	0.002	0.001	0.0005

$\lambda_{There\ EX\ NP\ NP\ PP\ IN\ near} = 0.2$

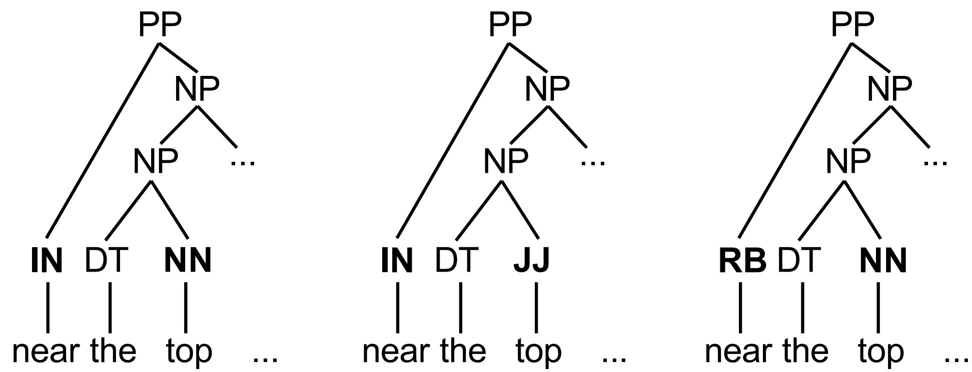
$\lambda_{There\ RB\ NP\ NP\ PP\ IN\ near} = 0.6$

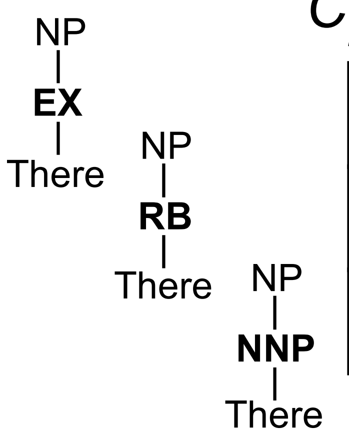
$\lambda_{There\ NNP\ NP\ NP\ PP\ IN\ near} = 0.1$

$\lambda_{There\ EX\ NP\ NP\ PP\ RB\ near} = 0.1$

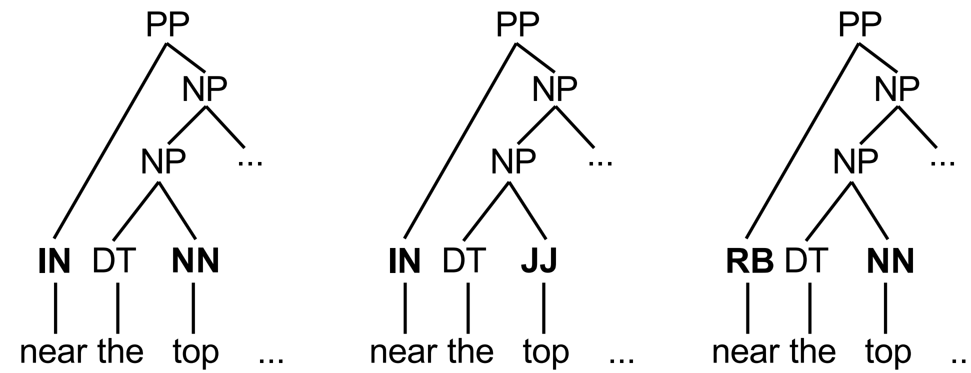
$\lambda_{There\ RB\ NP\ NP\ PP\ RB\ near} = 0.4$

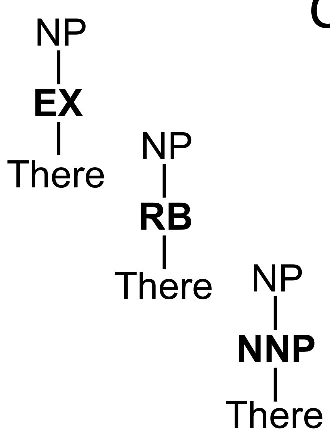
$\lambda_{There\ NNP\ NP\ NP\ PP\ RB\ near} = 0.2$



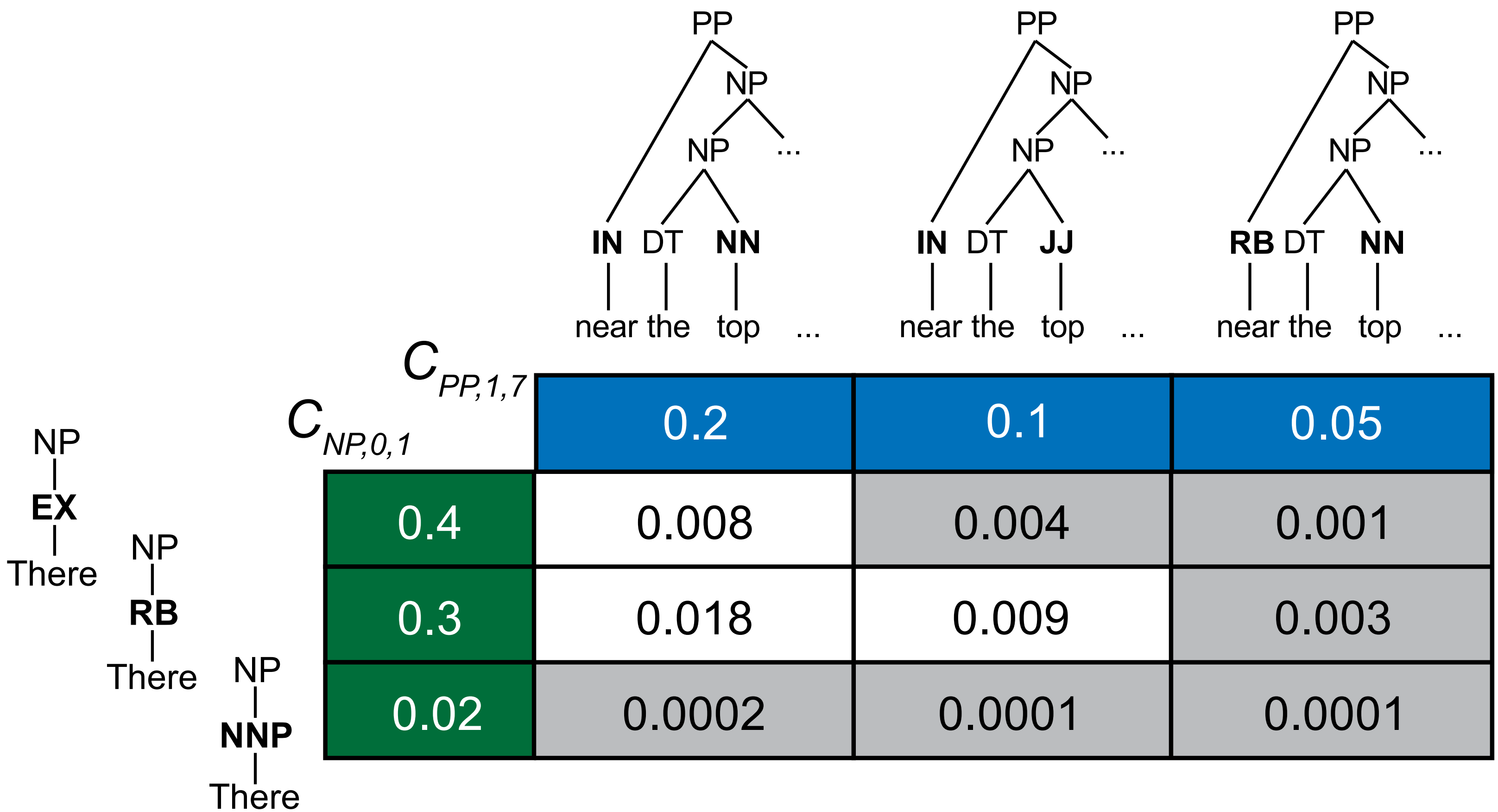


$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.04 × 0.2	0.02 × 0.2	0.01 × 0.1
0.3	0.03 × 0.6	0.015 × 0.6	0.0075 × 0.4
0.02	0.002 × 0.1	0.001 × 0.1	0.0005 × 0.2

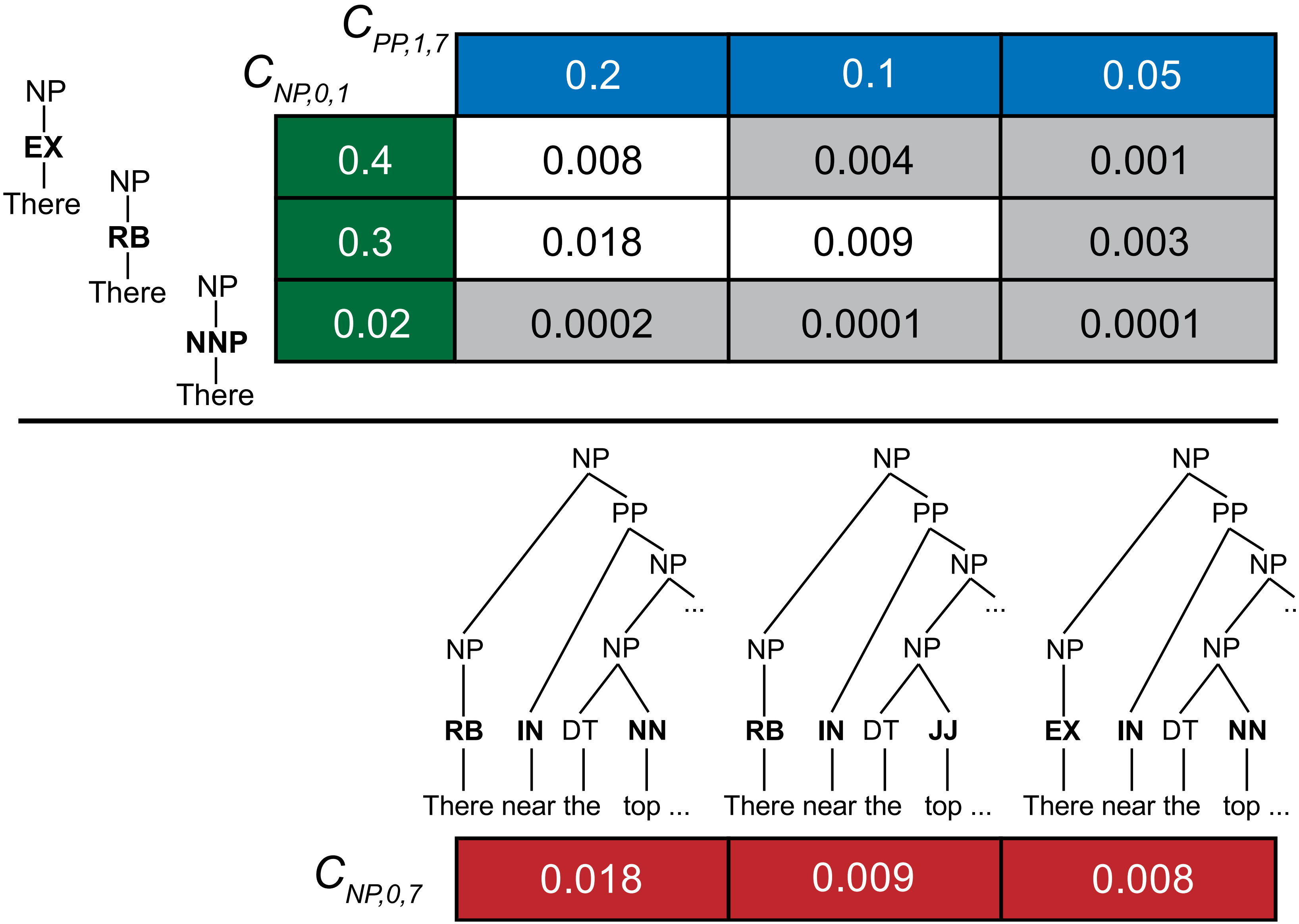




$C_{NP,0,1}$ \ $C_{PP,1,7}$	0.2	0.1	0.05
0.4	0.008	0.004	0.001
0.3	0.018	0.009	0.003
0.02	0.0002	0.0001	0.0001



PP
NP
NP
...
IN DT NN
near the top ...
PP
NP
NP
...
IN DT JJ
near the top ...
PP
NP
NP
...
RB DT NN
near the top ...
$C_{PP,1,7}$
$C_{NP,0,1}$
0.2
0.1
0.05
NP
EX
There
NP
RB
There
NP
NNP
There
0.4
0.008
0.004
0.001
0.3
0.018
0.009
0.003
0.02
0.0002
0.0001
0.0001



C PP,1,7
C NP,0,1
0.2
0.1
0.05
0.4
0.008
0.004
0.001
0.3
0.018
0.009
0.003
0.02
0.0002
0.0001
0.0001
NP
EX
There
NP
RB
There
NP
NNP
There
NP
PP
NP
...
NP
NP
RB
IN
DT
NN
There near the top ...
NP
PP
NP
...
NP
NP
RB
IN
DT
JJ
There near the top ...
NP
PP
NP
...
NP
NP
EX
IN
DT
NN
There near the top ...
C NP,0,7
0.018
0.009
0.008

Clarification

- Cube pruning does not actually expand all k^2 proofs as this example showed

- It uses a fast approximation that only looks at $O(k)$ proofs

Integer Linear Programming

- (on board)