CS11-747 Neural Networks for NLP

Parsing with Dynamic Programming

Graham Neubig

Carnegie Mellon University

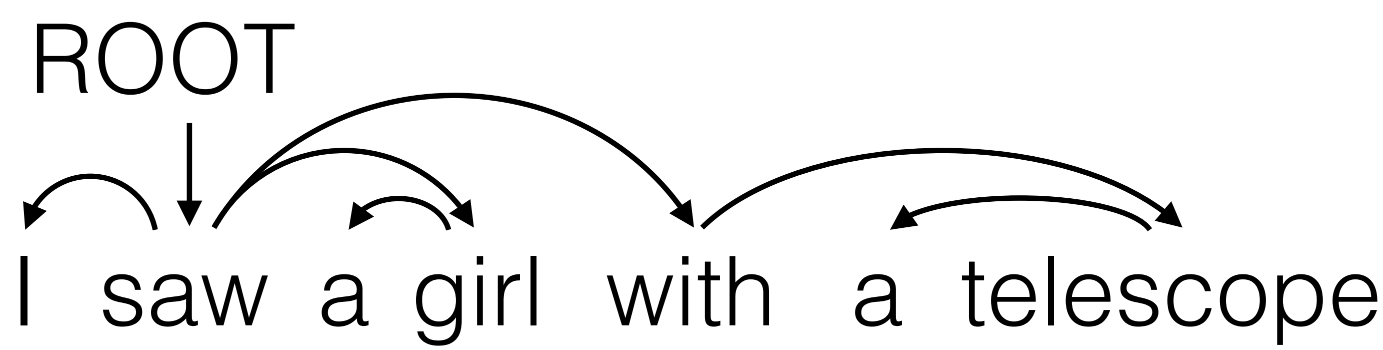
Language Technologies Institute

Site

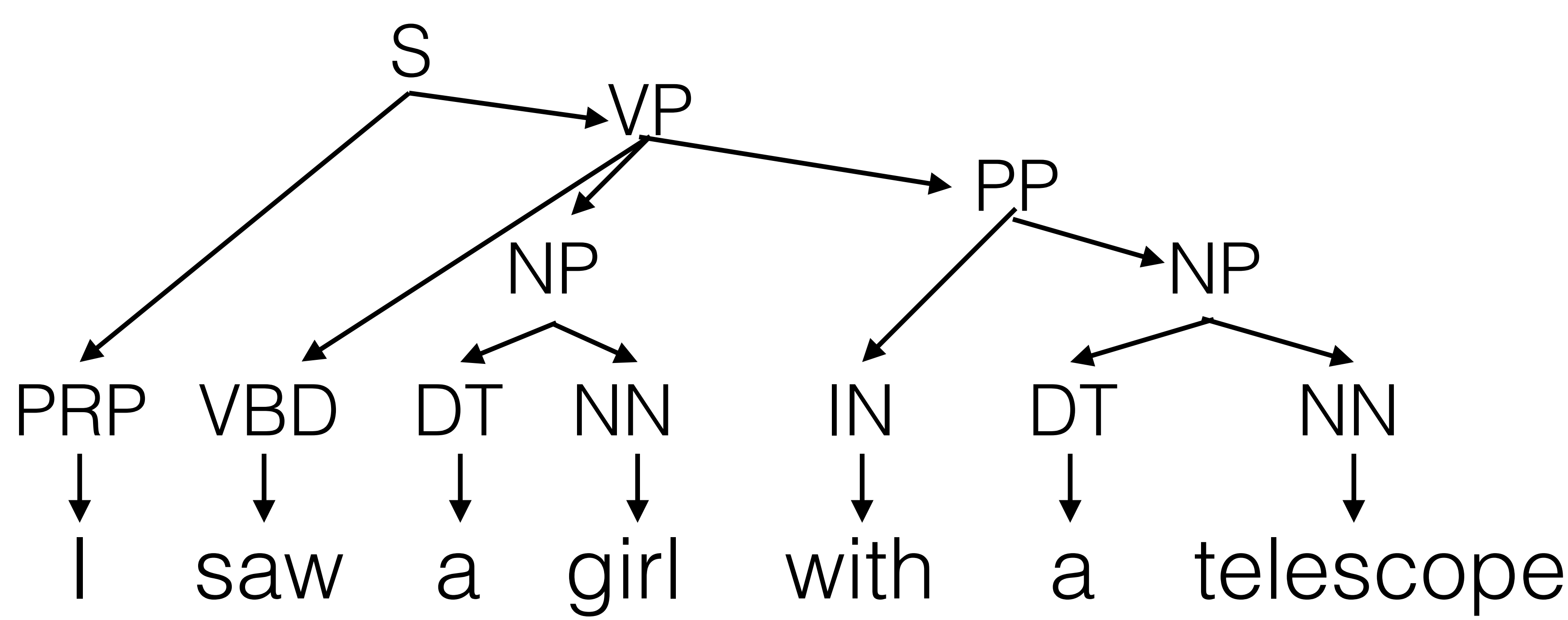
https://phontron.com/class/nn4nlp2020/

Two Types of Linguistic Structure

- **Dependency:** focus on relations between words



- **Phrase structure:** focus on the structure of the sentence



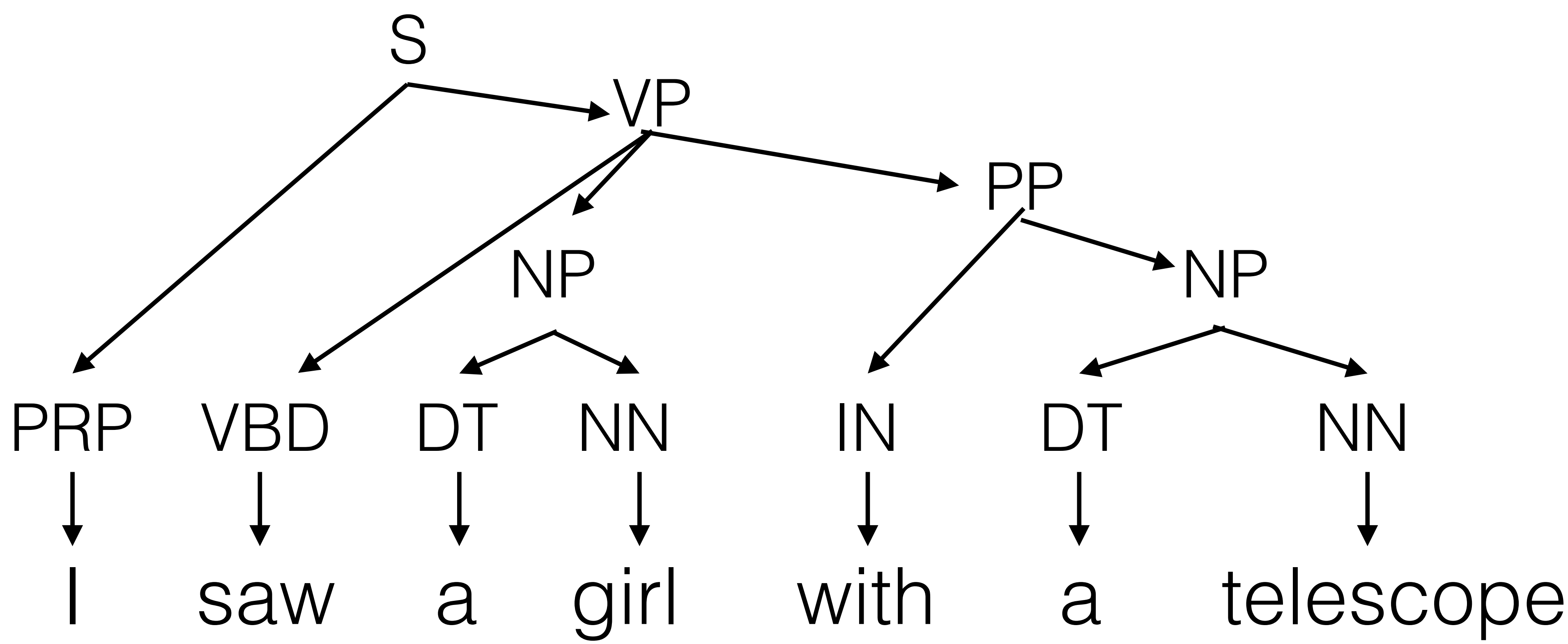
Parsing

- Predicting linguistic structure from input sentence
- **Transition-based models**
  - step through actions one-by-one until we have output
  - like history-based model for POS tagging
- **Dynamic programming-based models**
  - calculate probability of each edge/constituent, and perform some sort of dynamic programming
  - like linear CRF model for POS

Dynamic Programming for Phrase Structure Parsing

Phrase Structure Parsing

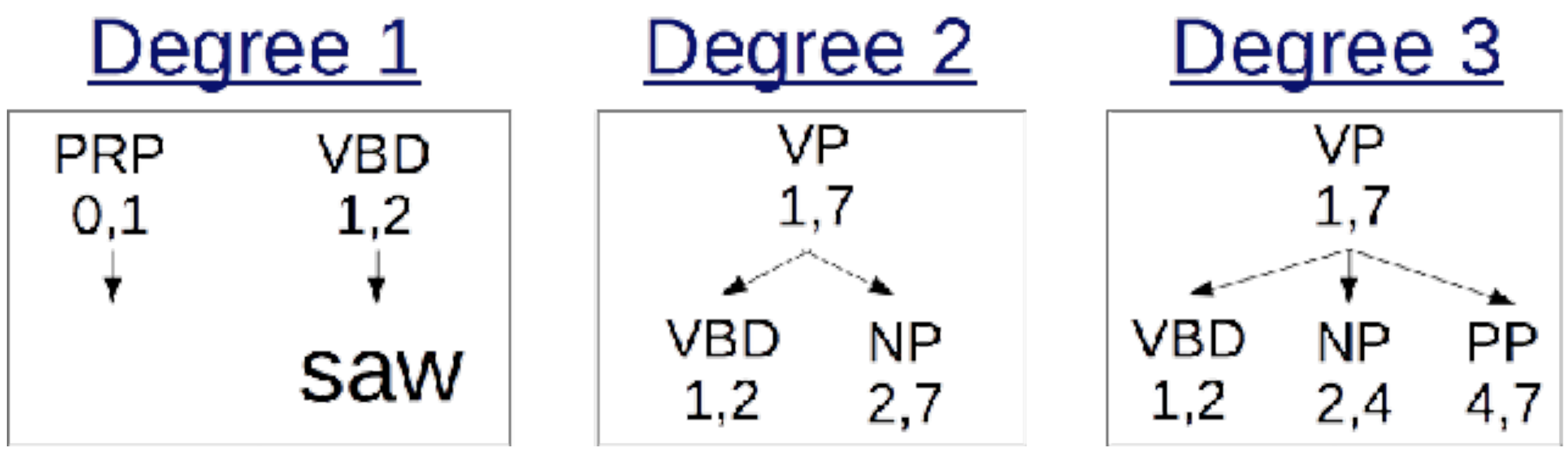
- Models to calculate phrase structure



- Important insight: parsing is similar to tagging
  - Tagging is search in a **graph** for the best **path**
  - Parsing is search in a **hyper-graph** for the best **tree**

What is a Hyper-Graph?

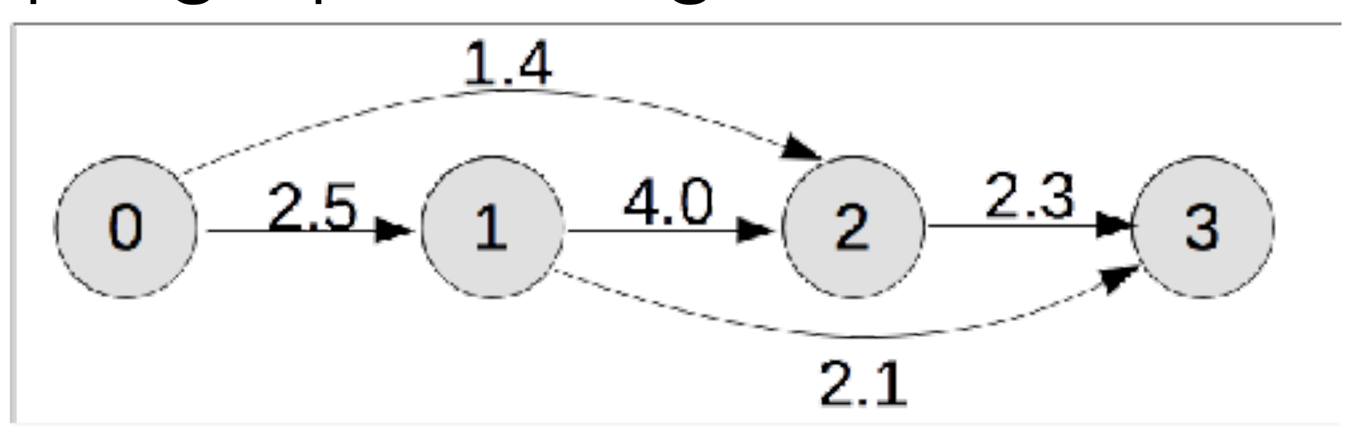
- The “degree” of an edge is the number of children



- The degree of a hypergraph is the maximum degree of its edges

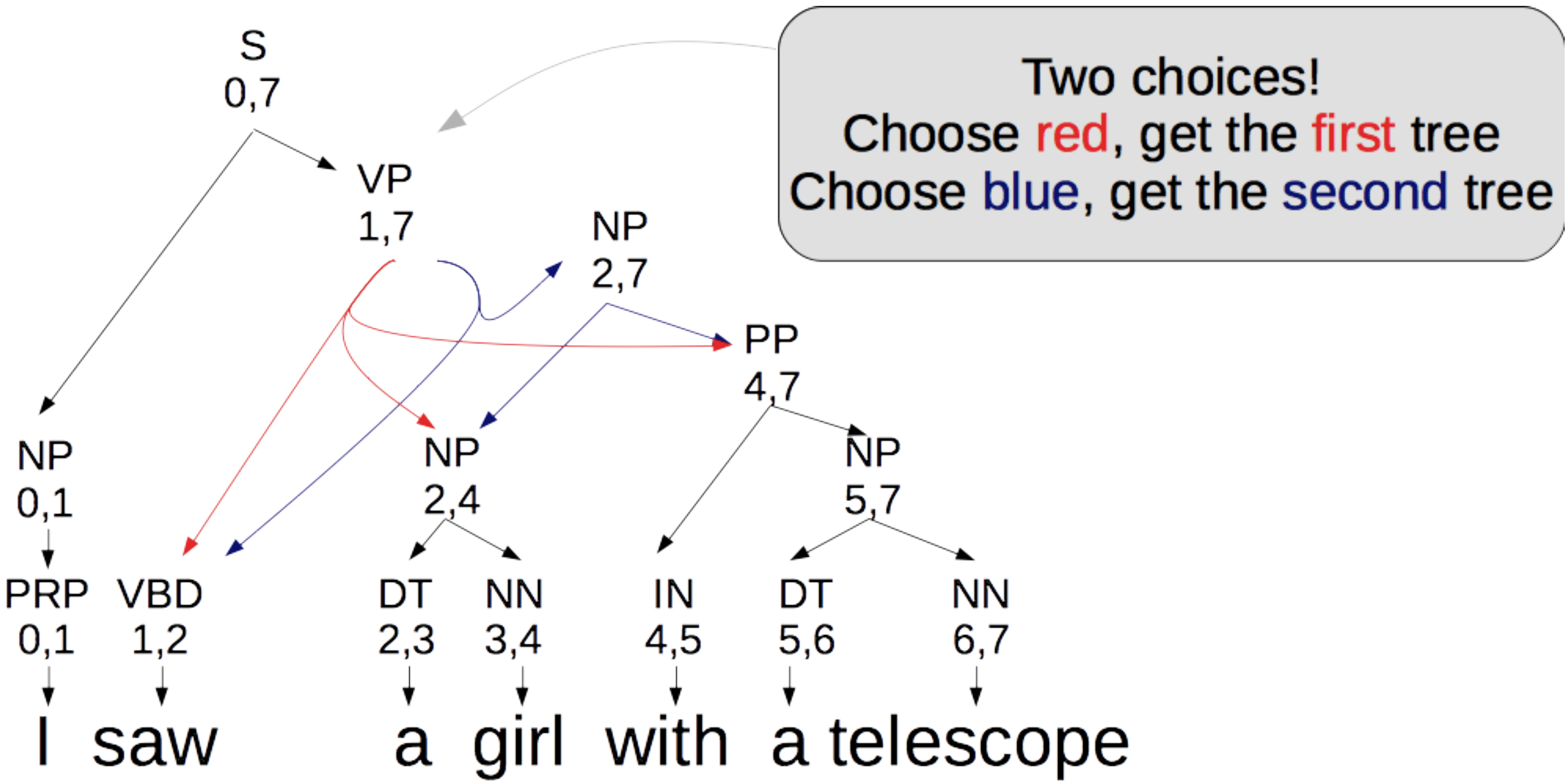
- A graph is a hypergraph of degree 1!

Example →



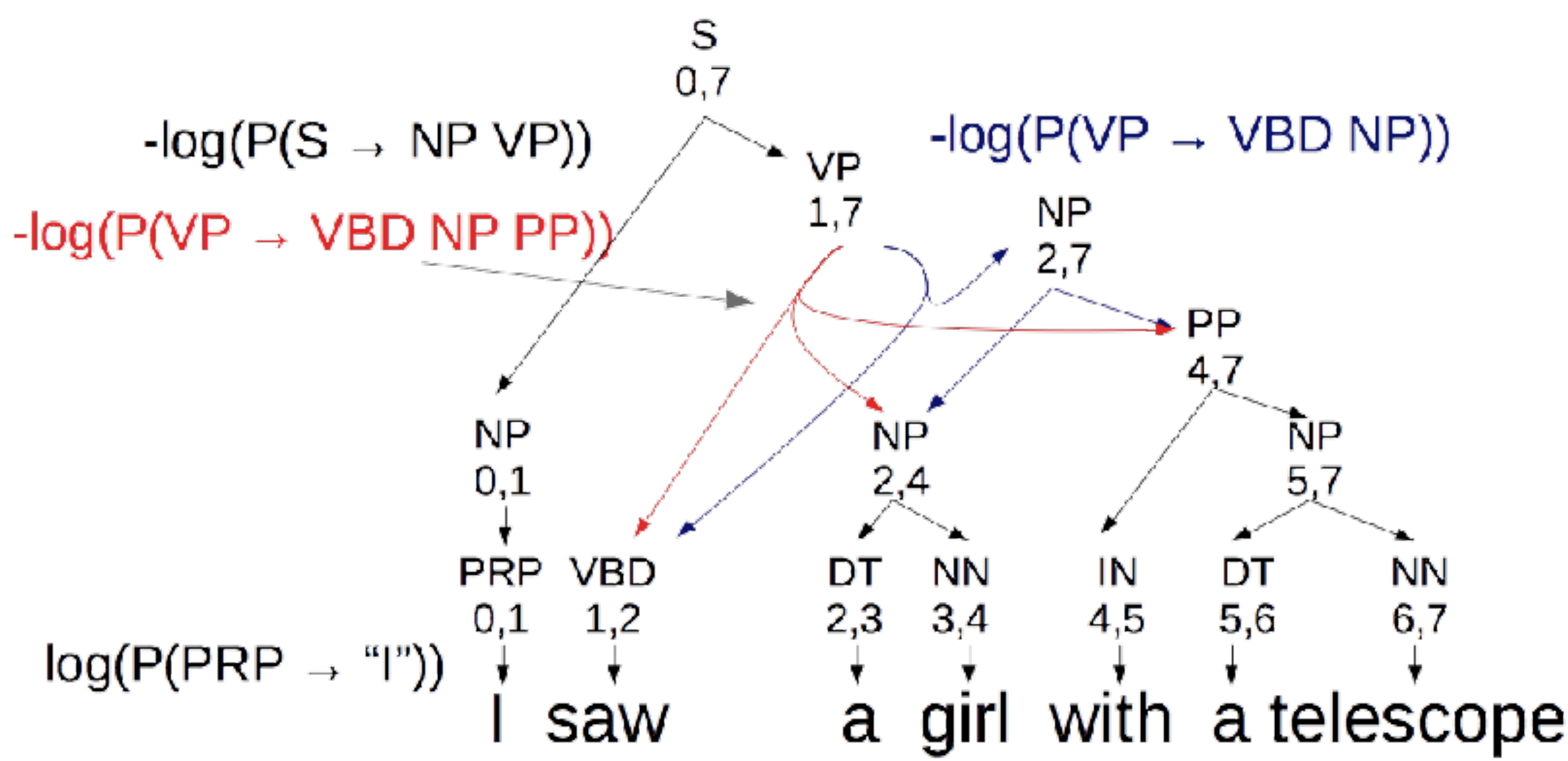
Tree Candidates as Hypergraphs

- With edges in one tree or another



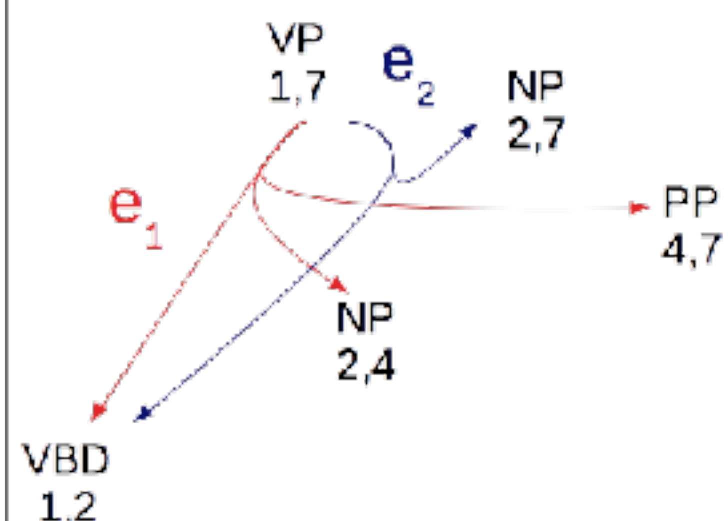
Weighted Hypergraphs

- Like graphs, can add weights to hypergraph edges
- Generally negative log probability of production



Hypergraph Search Example: CKY Algorithm

- Find the highest-scoring tree given a CFG grammar
- Create a hypergraph containing all candidates for a binarized grammar, do hypergraph search



score(e_1) =
 -log(P(VP → VBD NP PP)) +
 best_score[VBD1,2] +
 best_score[NP2,4] +
 best_score[NP2,7]

score(e_2) =
 -log(P(VP → VBD NP)) +
 best_score[VBD1,2] +
 best_score[VBD2,7]

best_edge[VB1,7] = argmin$_{e1,e2}$ score

best_score[VB1,7] =
 score(best_edge[VB1,7])

- Analogous to Viterbi algorithm, which is over graphs, but over hyper-graphs

Hypergraph Partition Function: Inside-outside Algorithm

- Find the marginal probability of each span given a CFG grammar

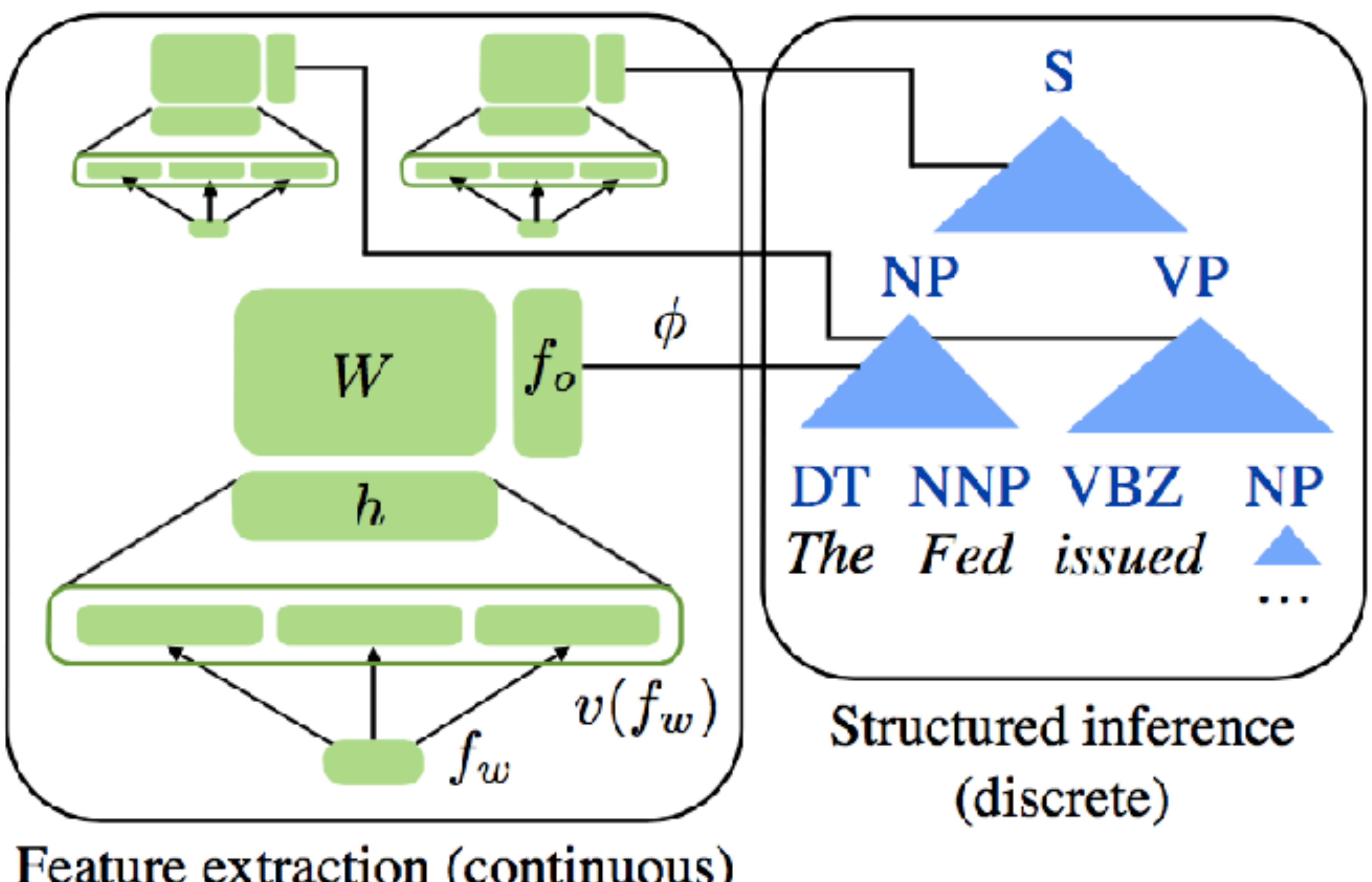
- Partition function us probability of the top span

- Same as CKY, except we logsumexp instead of max

- Analogous to forward-backward algorithm, but forward-backward is over graphs, inside-outside is over hyper-graphs

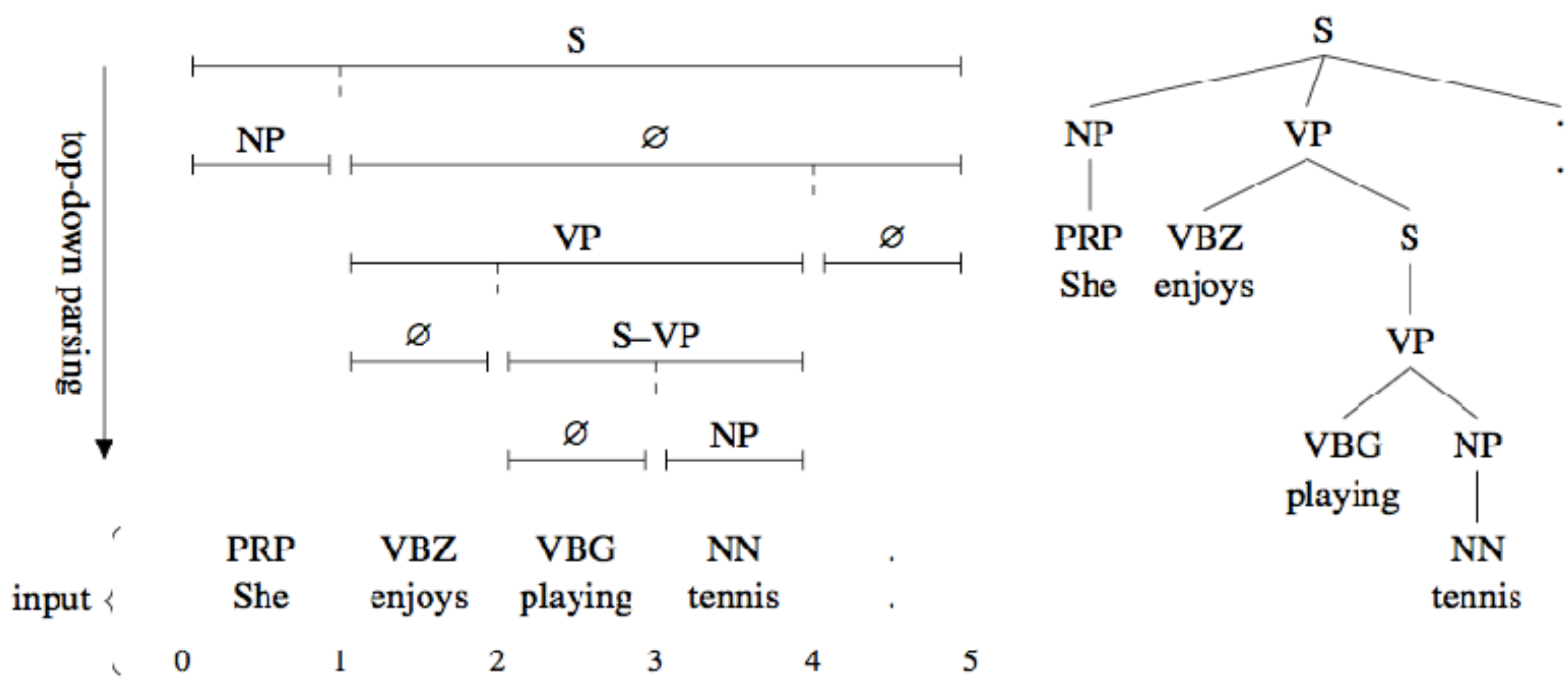
Neural CRF Parsing
(Durrett and Klein 2015)

- Predict score of each span using FFNN
- Do discrete structured inference using CKY, inside-outside



Span Labeling
(Stern et al. 2017)

- Simple idea: try to decide whether span is constituent in tree or not

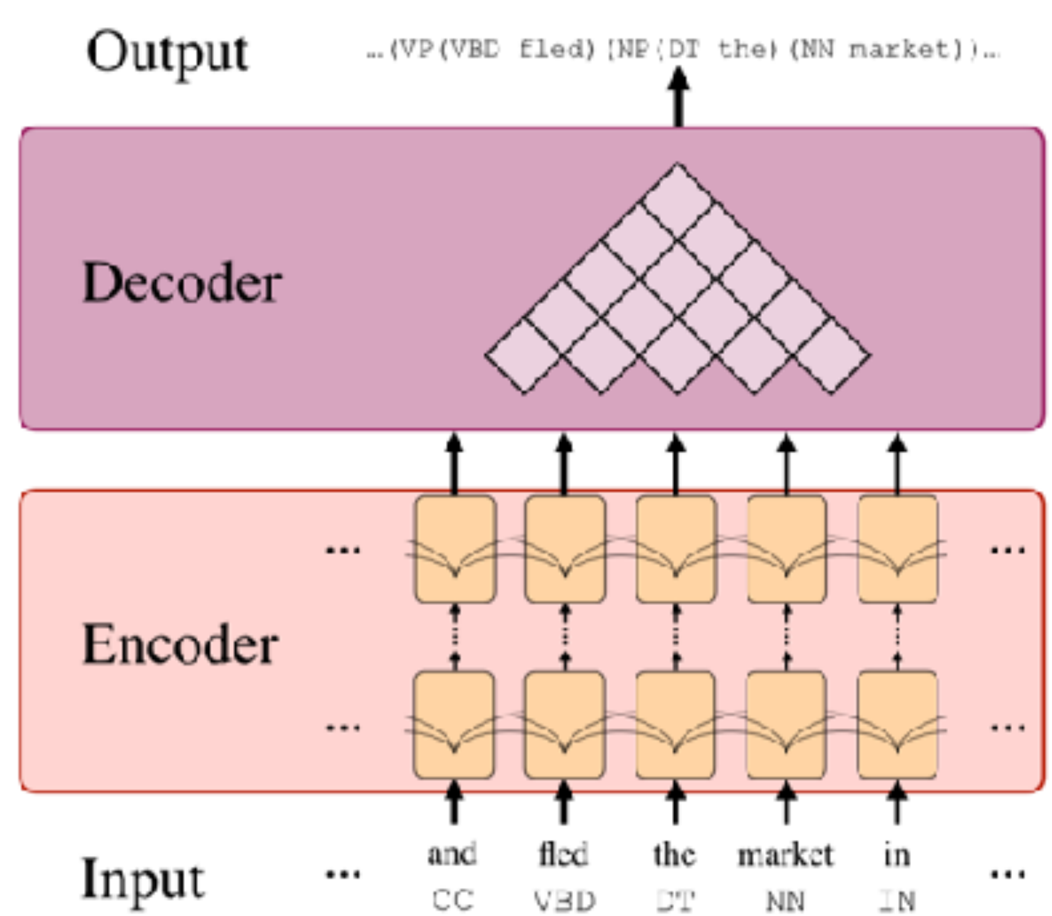


(a) Execution of the top-down parsing algorithm. (b) Output parse tree.

- Allows for various loss functions (local vs. structured), inference algorithms (CKY, top down)

Self-Attentional Encoding+Structured Inference (Kitaev et al. 2018)

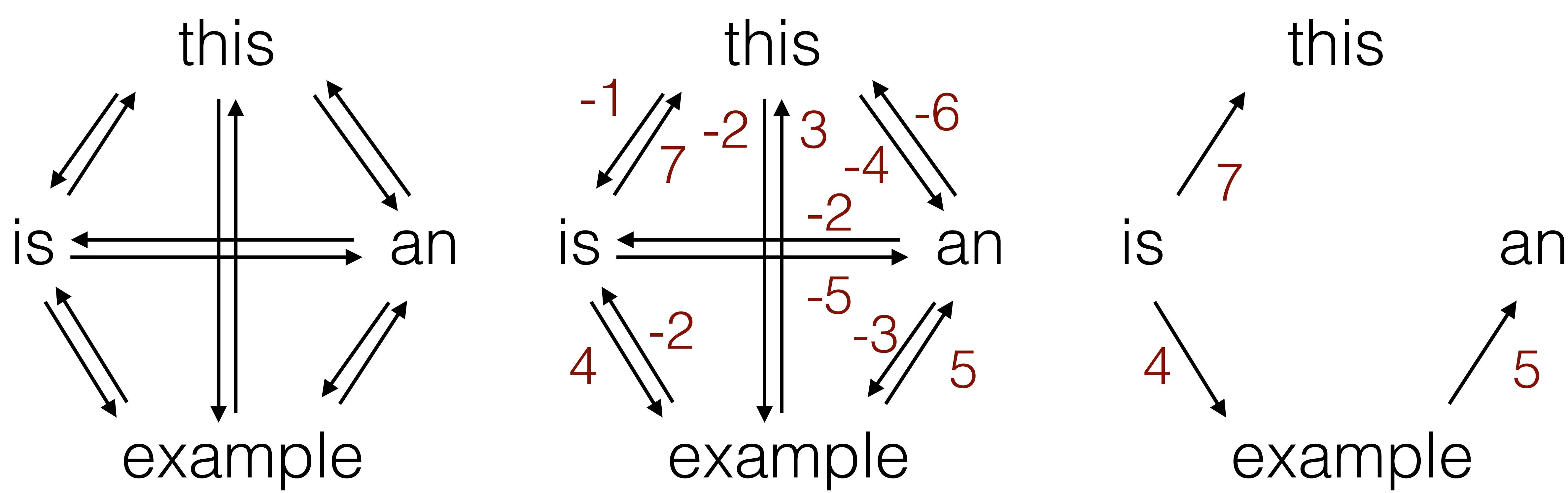
- Self-attention based encoding
- Structured margin-based inference
- Berkeley neural parser: https://github.com/nikitakit/self-attentive-parser



Dependency Parsing with Dynamic Programs

(First Order) Graph-based Dependency Parsing

- Express sentence as fully connected directed graph
- Score each edge independently
- Find maximal spanning tree



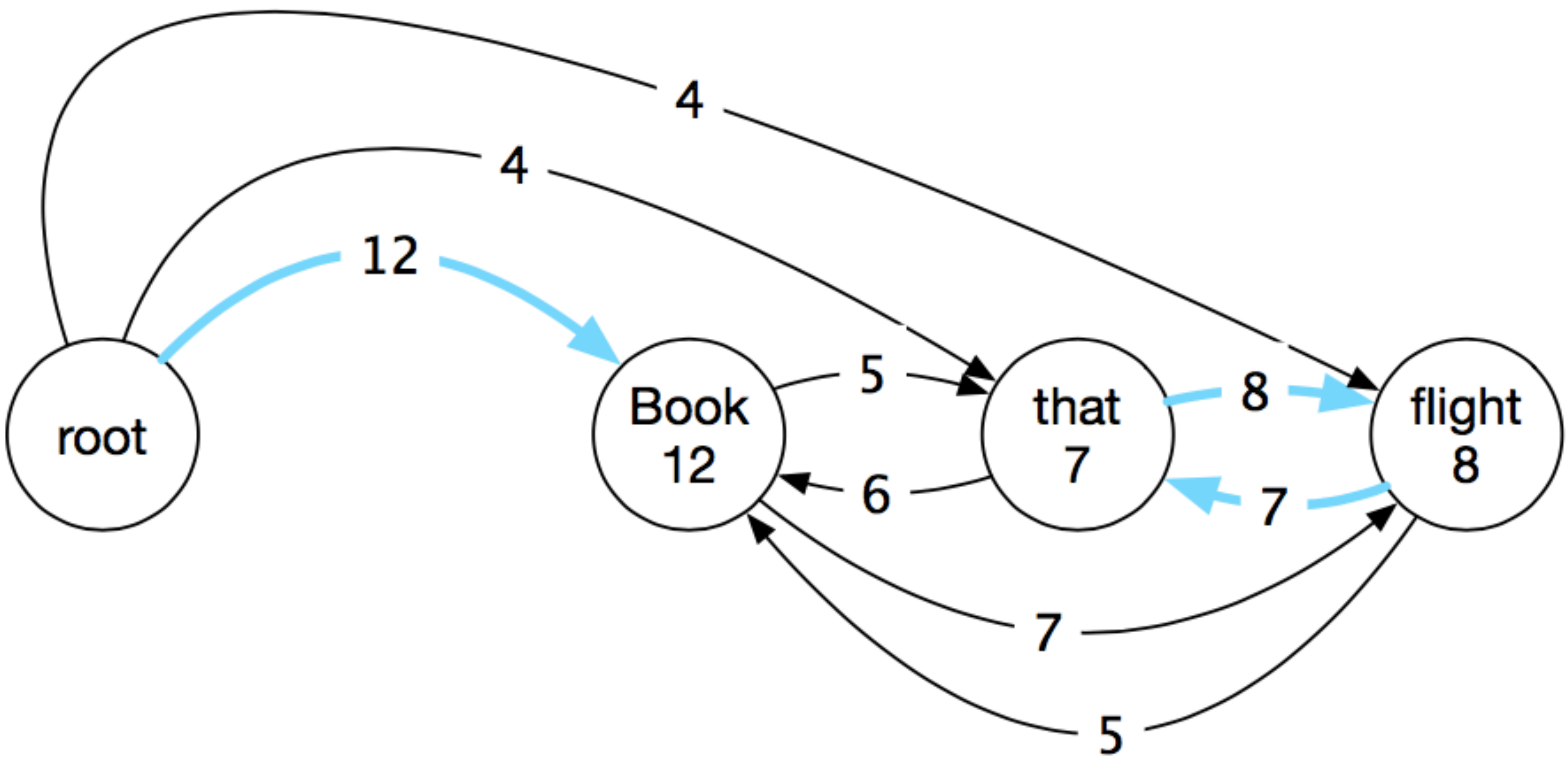
Graph-based vs. Transition Based

- **Transition-based**
  - + Easily condition on infinite tree context (structured prediction)
  - - Greedy search algorithm causes short-term mistakes
- **Graph-based**
  - + Can find exact best global solution via DP algorithm
  - - Have to make local independence assumptions

Chu-Liu-Edmonds (Chu and Liu 1965, Edmonds 1967)

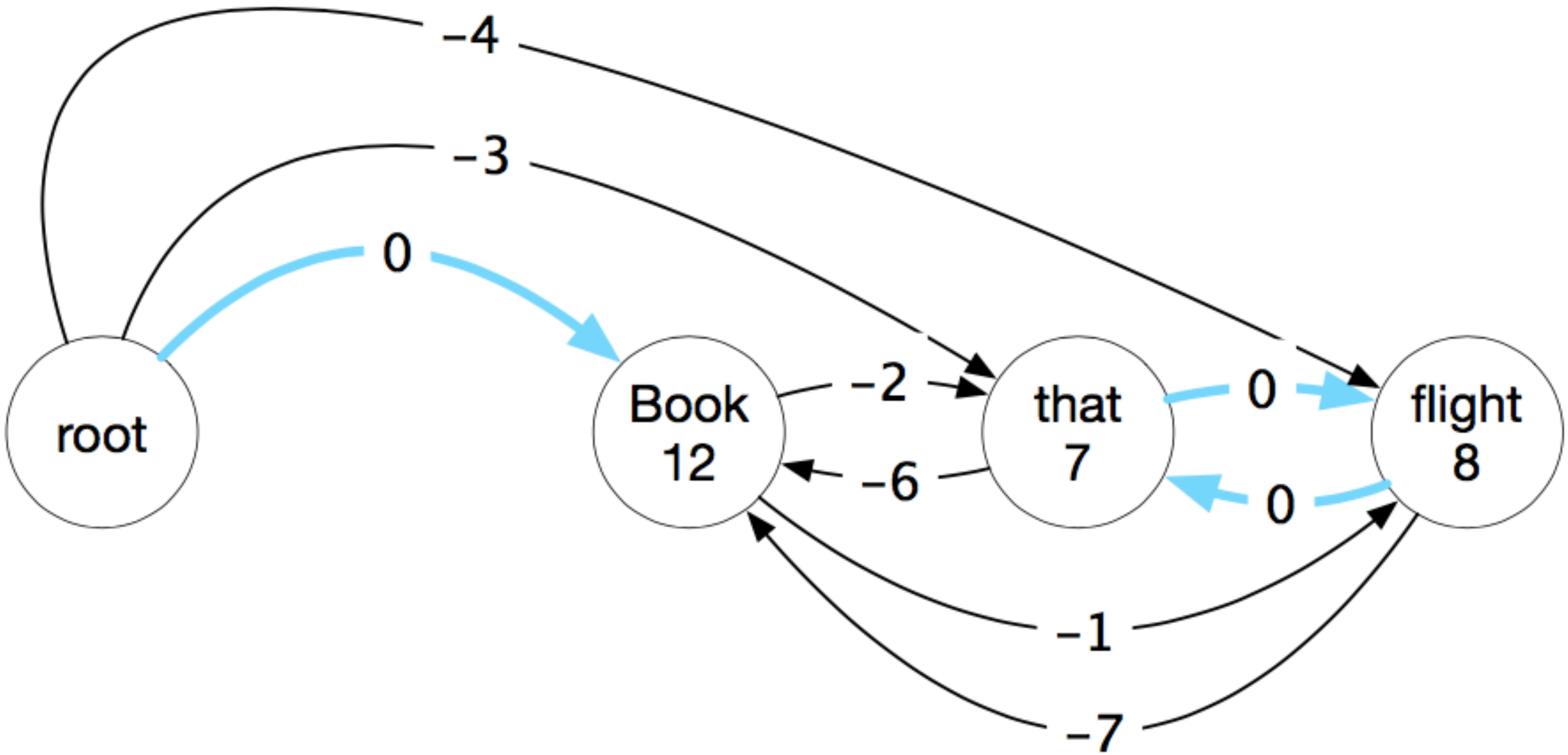
- We have a graph and want to find its spanning tree
- **Greedily select** the best incoming edge to each node (and subtract its score from all incoming edges)
- If there are cycles, select a cycle and **contract** it into a single node
- **Recursively call** the algorithm on the graph with the contracted node
- **Expand** the contracted node, deleting an edge appropriately

Chu-Liu-Edmonds (1): Find the Best Incoming



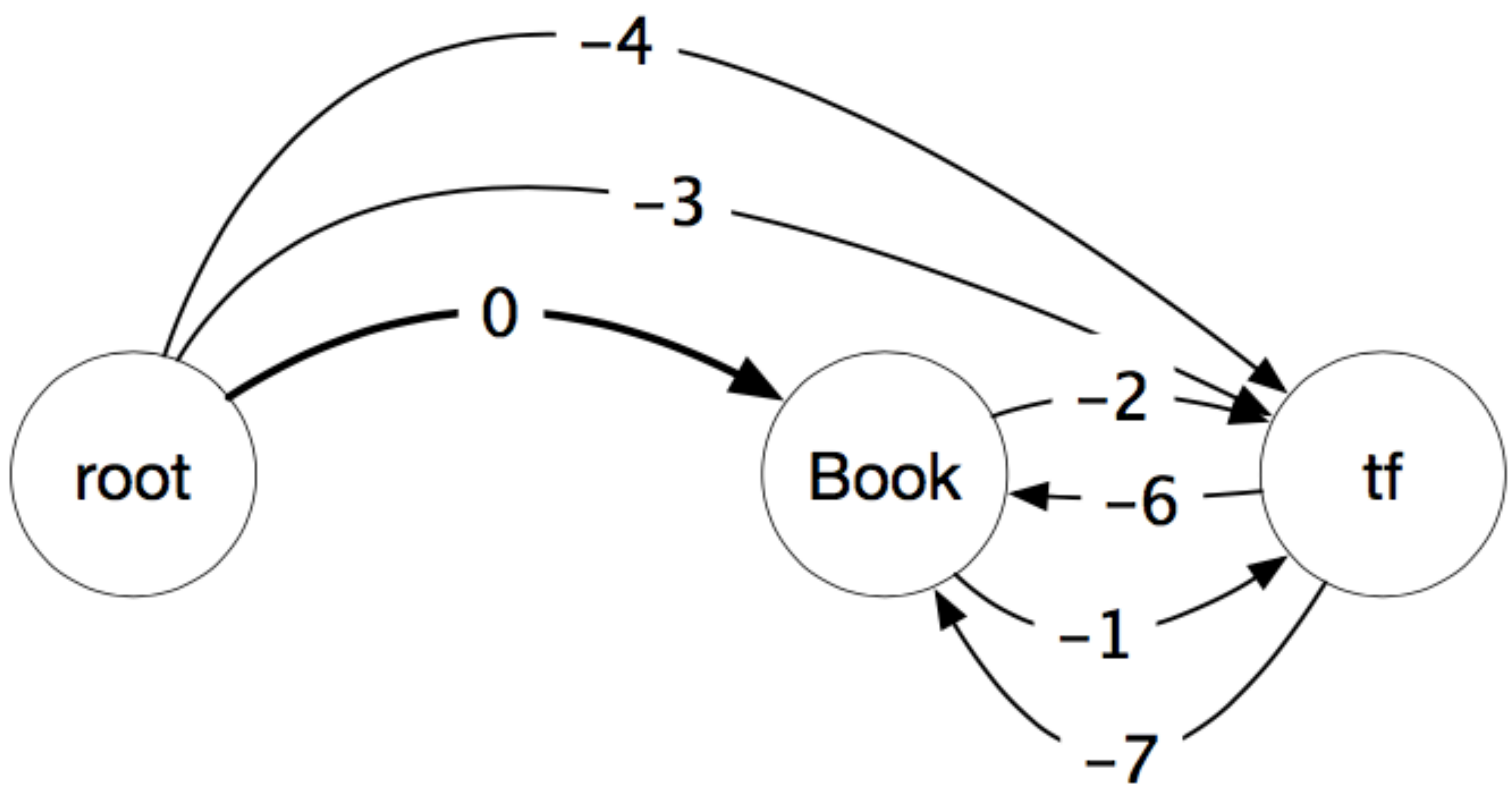
(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (2): Subtract the Max for Each



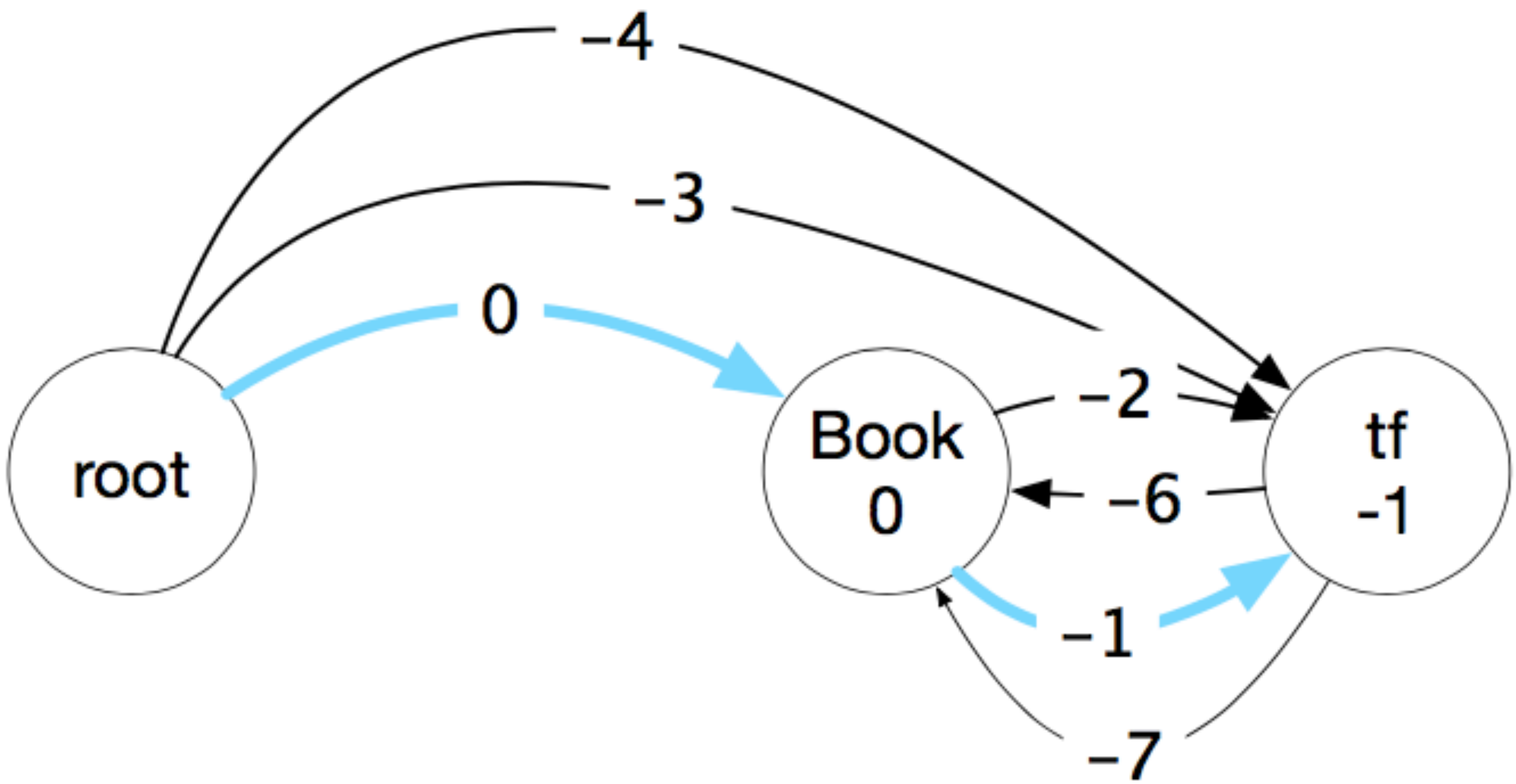
(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (3): Contract a Node



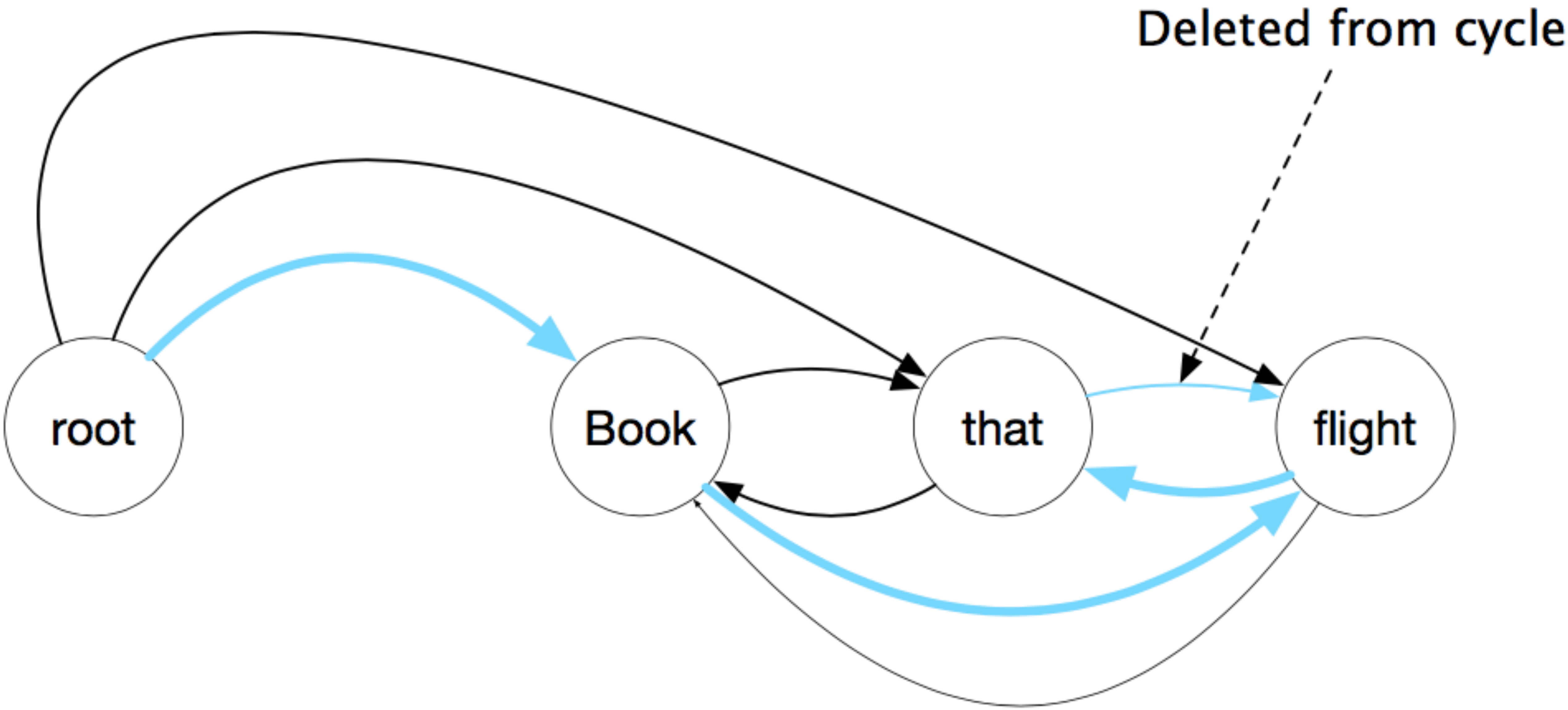
(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (4): Recursively Call Algorithm



(Figure Credit: Jurafsky and Martin)

Chu-Liu-Edmonds (5): Expand Nodes and Delete Edge



(Figure Credit: Jurafsky and Martin)

Other Dynamic Programs

- **Eisner's Algorithm** (Eisner 1996):
  - A dynamic programming algorithm to combine together trees in $O(n^3)$
  - Creates *projective* dependency trees (Chu-Liu-Edmonds is *non-projective*)
- **Tarjan's Algorithm** (Tarjan 1979, Gabow and Tarjan 1983):
  - Like Chu-Liu-Edmonds, but better asymptotic runtime $O(m + n \log n)$

Training Algorithm
(McDonald et al. 2005)

- Basically use **structured hinge loss** (covered in structured prediction class)
- Find the highest scoring tree, penalizing each correct edge by the margin
- If the found tree is not equal to the correct tree, update parameters using hinge loss

Features for Graph-based Parsing (McDonald et al. 2005)

- What features did we use before neural nets?

a)

Basic Uni-gram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

b)

Basic Big-ram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

c)

In Between POS Features
p-pos, b-pos, c-pos
Surrounding Word POS Features
p-pos, p-pos+1, c-pos-1, c-pos
p-pos-1, p-pos, c-pos-1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos-1, p-pos, c-pos, c-pos+1

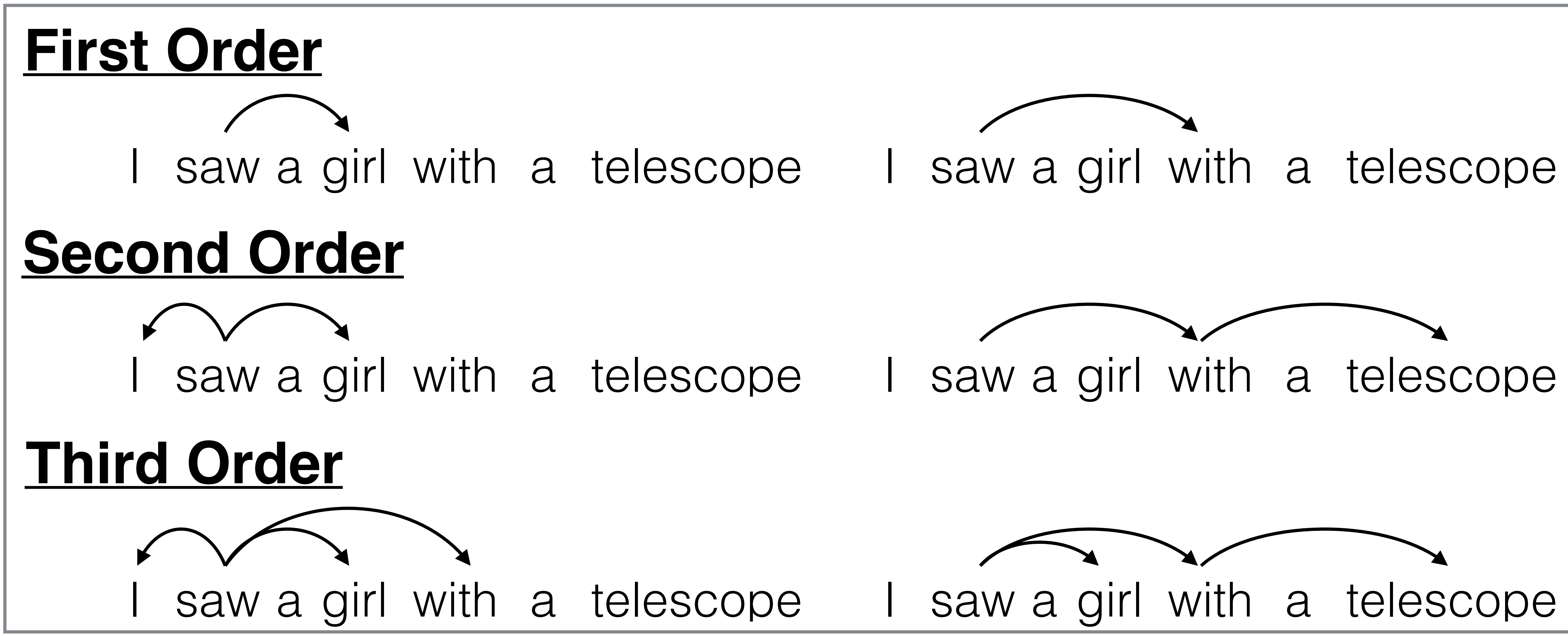
Table 1: Features used by system. p-word: word of parent node in dependency tree. c-word: word of child node. p-pos: POS of parent node. c-pos: POS of child node. p-pos+1: POS to the right of parent in sentence. p-pos-1: POS to the left of parent. c-pos+1: POS to the right of child. c-pos-1: POS to the left of child. b-pos: POS of a word in between parent and child nodes.

- All conjoined with arc direction and arc distance
- Also use POS combination features
- Also represent words w/ prefix if they are long

Higher-order Dependency Parsing

(e.g. Zhang and McDonald 2012)

- Consider multiple edges at a time when calculating scores



- + Can extract more expressive features
- - Higher computational complexity, approximate search necessary

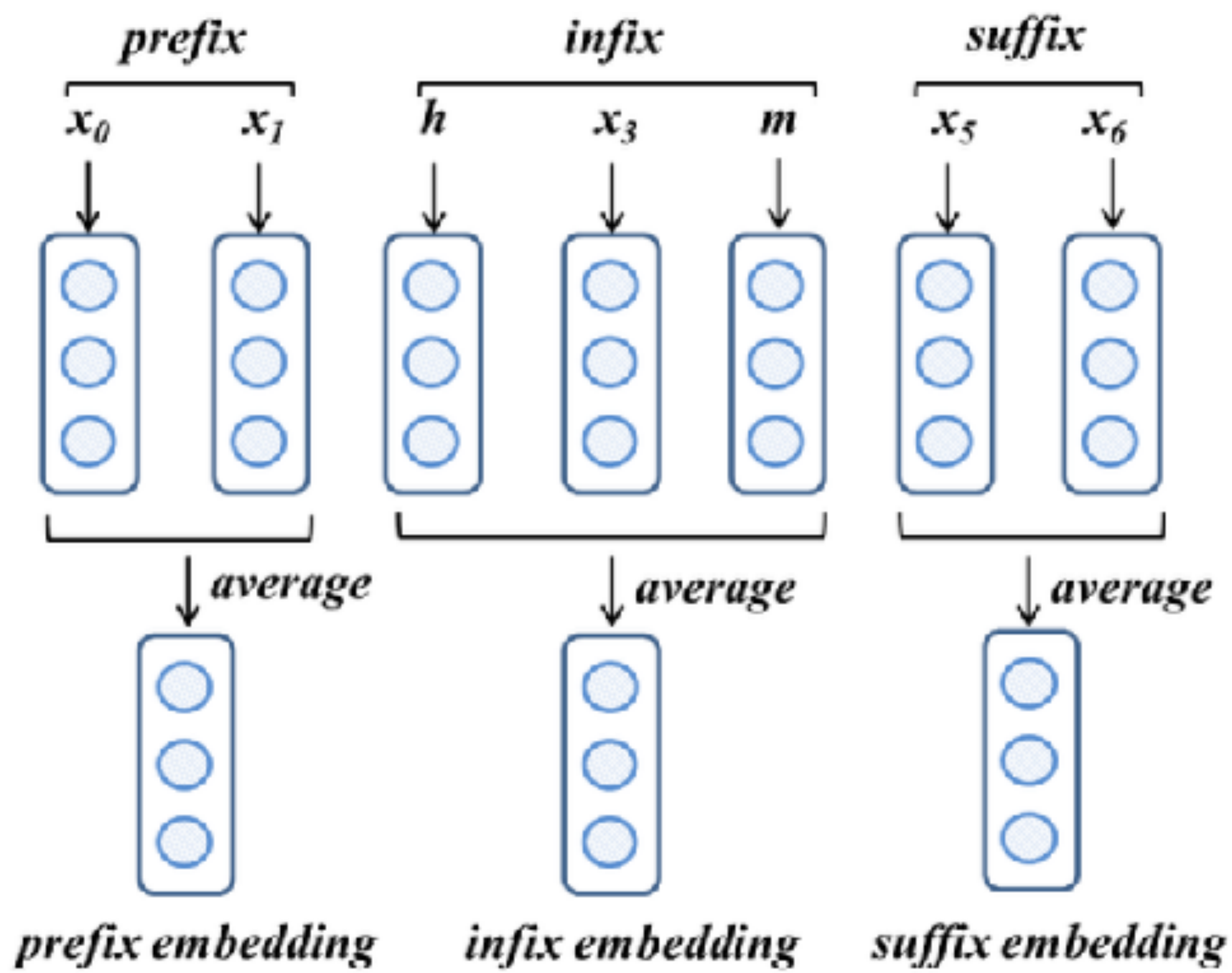
Neural Models for Graph-based Parsing

Neural Feature Combinators
(Pei et al. 2015)

- Extract traditional features, let NN do feature combination
    - Similar to Chen and Manning (2014)'s transition-based model
- Use **averaged embeddings of phrases**
- Use **second-order features**

Phrase Embeddings
(Pei et al. 2015)

- Motivation: words surrounding or between head and dependent are important clues
- Take average of embeddings

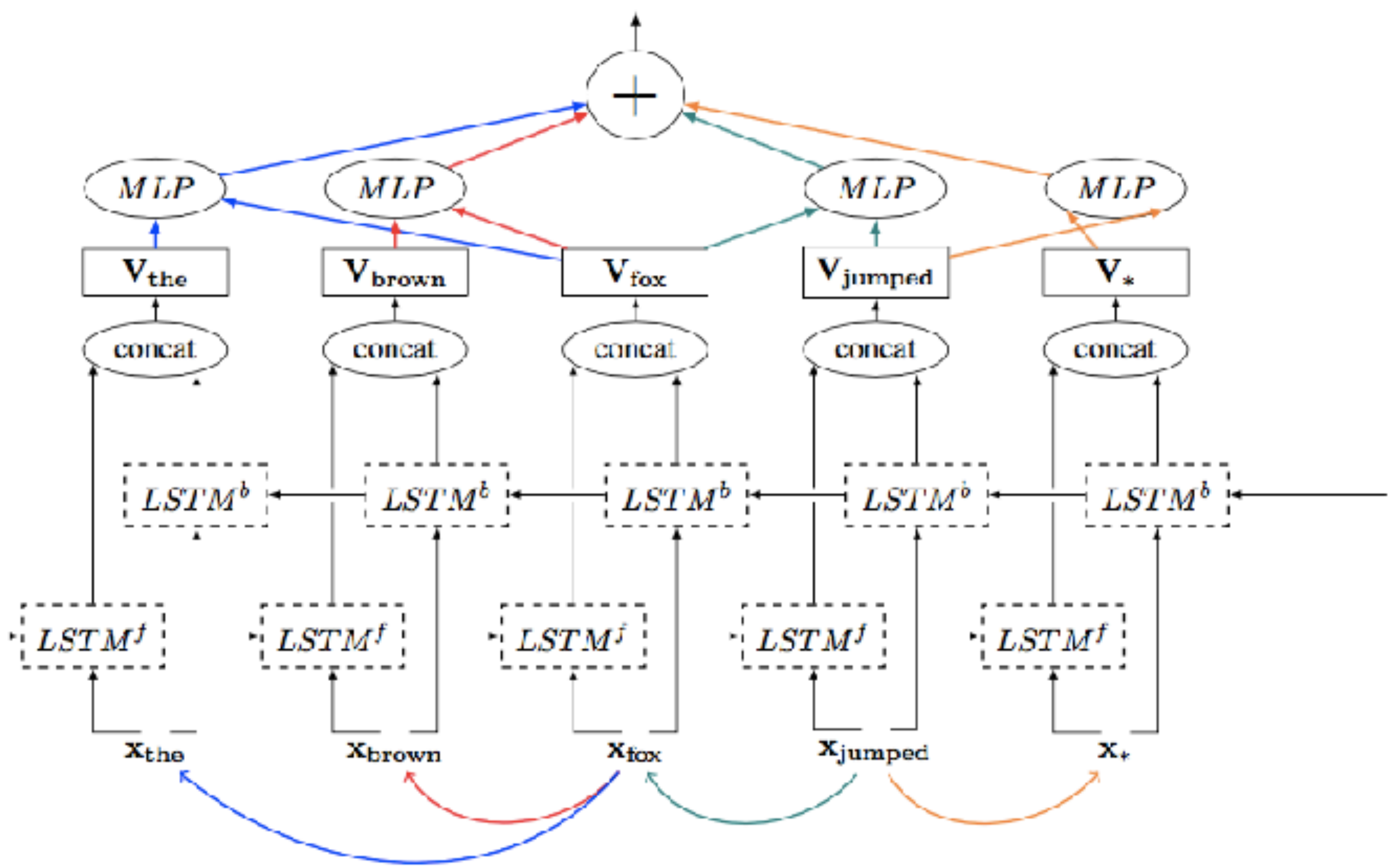


Do Neural Feature Combinators Help?

(Pei et al. 2015)

- Yes!
  - 1st-order: LAS 90.39->91.37, speed 26 sent/sec
  - 2nd-order: LAS 91.06->92.13, speed 10 sent/sec
- 2nd-order neural better than 3rd-order non-neural at UAS

BiLSTM Feature Extractors
(Kipperwasser and Goldberg 2016)



- Simpler and better accuracy than manual extraction

BiAffine Classifier

(Dozat and Manning 2017)

$$\mathbf{h}_i^{(arc\text{-}dep)} = \mathbf{MLP}^{(arc\text{-}dep)}(\mathbf{r}_i)$$

$$\mathbf{h}_j^{(arc\text{-}head)} = \mathbf{MLP}^{(arc\text{-}head)}(\mathbf{r}_j)$$

Learn specific representations for head/dependent for each word

$$\begin{aligned}\mathbf{s}_i^{(arc)} &= H^{(arc\text{-}head)} U^{(1)} \mathbf{h}_i^{(arc\text{-}dep)} \\ &+ H^{(arc\text{-}head)} \mathbf{u}^{(2)}\end{aligned}$$

Calculate score of each arc

- Just optimize the likelihood of the parent, no structured training
  - This is a local model, with global decoding using MST at the end
- Best results (with careful parameter tuning) on universal dependencies parsing task
- Implementation: https://github.com/XuezheMax/NeuroNLP2

Global Training

- Previously: margin-based global training, local probabilistic training
- What about global probabilistic models?

$$P(Y \mid X) = \frac{e^{\sum_{j=1}^{|Y|} S(y_j|X,y_1,\ldots,y_{j-1})}}{\sum_{\tilde{Y}\in V*} e^{\sum_{j=1}^{|\tilde{Y}|} S(\tilde{y}_j|X,\tilde{y}_1,\ldots,\tilde{y}_{j-1})}}$$

- Algorithms for calculating partition functions:
  - **Projective parsing:** Eisner algorithm is a bottom-up CKY-style algorithm for dependencies (Eisner et al. 1996)
  - **Non-projective parsing:** Matrix-tree theorem can compute marginals over directed graphs (Koo et al. 2007)
- Applied to neural models in Ma et al. (2017)

An Alternative:
Parse Reranking

An Alternative: Parse Reranking

- You have a nice model, but it’s hard to implement a dynamic programming decoding algorithm
- Try reranking!
    - Generate with an easy-to-decode model
    - Rescore with your proposed model

Examples of Reranking

- Inside-outside recursive neural networks (Le and Zuidema 2014)
- Parsing as language modeling (Choe and Charniak 2016)
- Recurrent neural network grammars (Dyer et al. 2016)

A Word of Caution about Reranking! (Fried et al. 2017)

- Your reranking model got SOTA results, great!
- But, it might be an effect of model combination (which we know works very well)
    - The model generating the parses **prunes down the search space**
    - The reranking model chooses the best parse **only in that space**!

	Scoring models		
Candidates	RD	RG	RD + RG
RD	92.22	93.45	93.87
RG	90.24	89.55	90.53
RD ∪ RG	92.22	92.78	93.92

Questions?