Dependency Parsing

Lecture 12

Kilian Evang Jakub Waszczuk

Heinrich Heine University Düsseldorf

Summer semester 2021

Today

- Previous homework
- Graph-based projective parsing (slides from the course at ESSLLI 2007 by Ryan McDonald and Joakim Nivre + our own inference rules)
- Learning the parameters (slides from the same course)

Dependency parsing methods

Graph-based methods

- Pros: exact inference; typically better results
- Cons: complexity (esp. non-projective parsing)

Transition-based methods

- Pros: fast; non-limited access into parsing history
- Cons: greedy (inference not exact)

Graph-based mathematically more sound, transition-based linguistically more attractive

Arc-factored model

Weight

We define the weight of a dependency graph $G = (V, A)$ as the product of the (non-negative) weights of the arcs in A:

$$w(G) = \prod_{(i,j,\ell) \in A} w_{ij}^{\ell}$$

Parsing

For a given sentence x, find a dependency graph $\hat{G}$ which maximizes $w(G)$:

$$\hat{G} = \arg\max_{G \in \mathcal{G}_x} \; w(G)$$

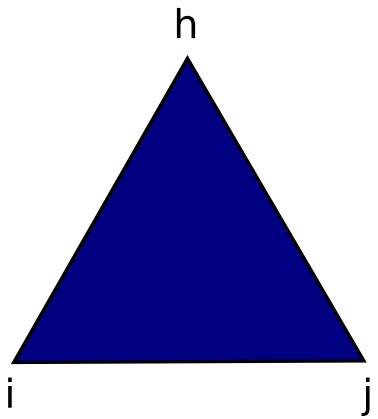
where $\mathcal{G}_x$ is the set of permissible structures for sentence x.

Permissible structures

Non-projective (last session) and projective (today)

Arc-factored Projective Parsing

- Projective dependency structures are nested
- Can use CFG like parsing algorithms – chart parsing
- Each chart item (triangle) represents the weight of the best tree rooted at word h spanning all the words from i to j
  - Analog in CFG parsing: items represent best tree rooted at non-terminal NT spanning words i to j
- **Goal**: Find chart item rooted at 0 spanning 0 to n

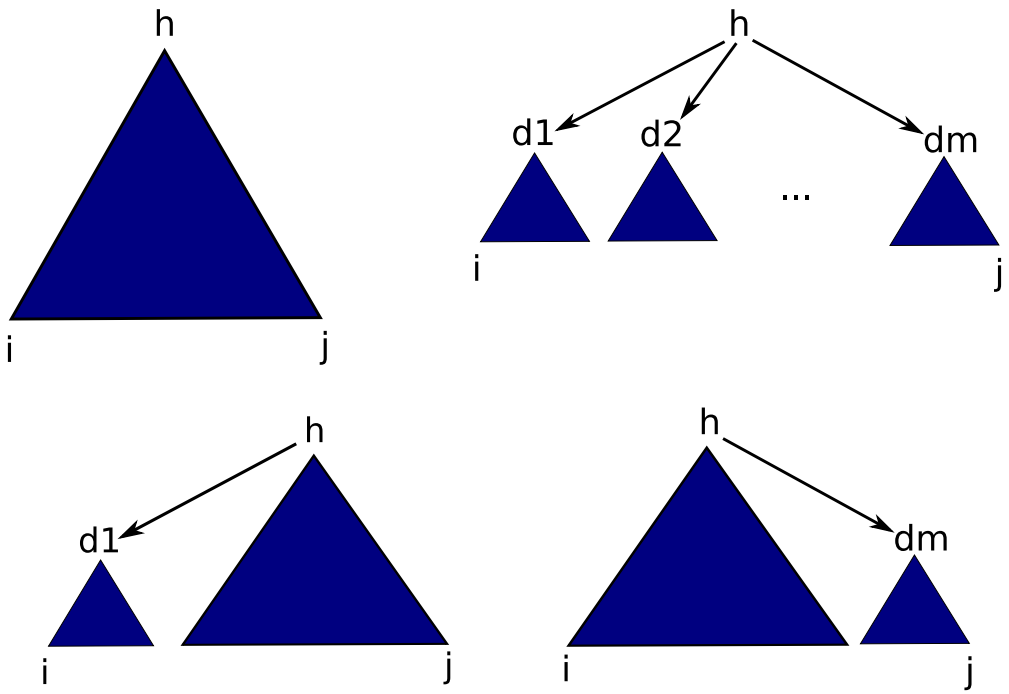


Base case

Length 1, $h = i = j$, has weight 1

Arc-factored Projective Parsing

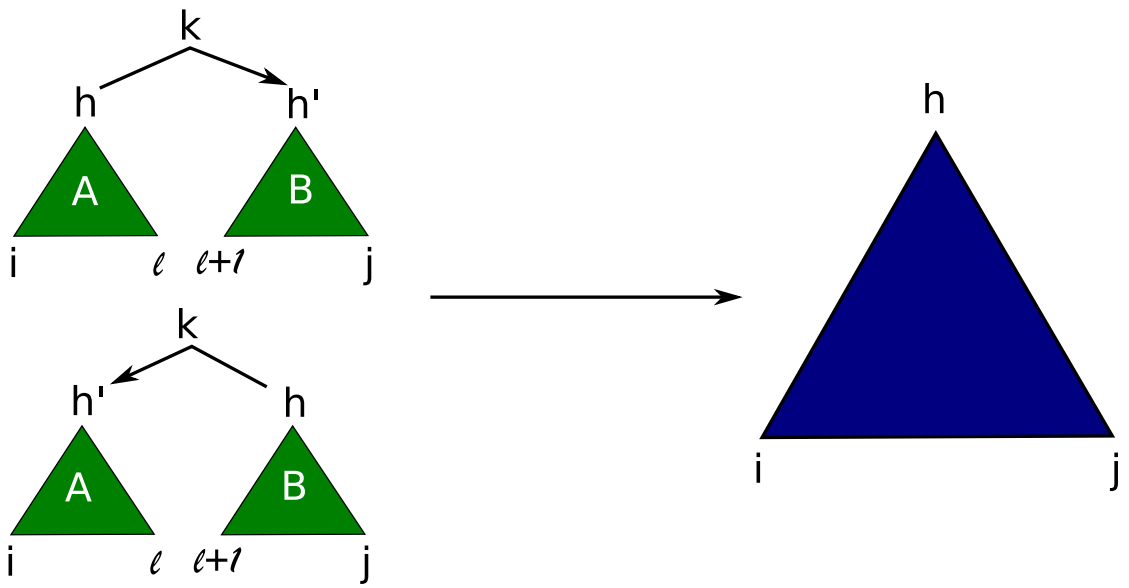
- All projective graphs can be written as the combination of two smaller adjacent graphs



- Inductive hypothesis – algorithm has calculated score of smaller items correctly (just like CKY)

Arc-factored Projective Parsing

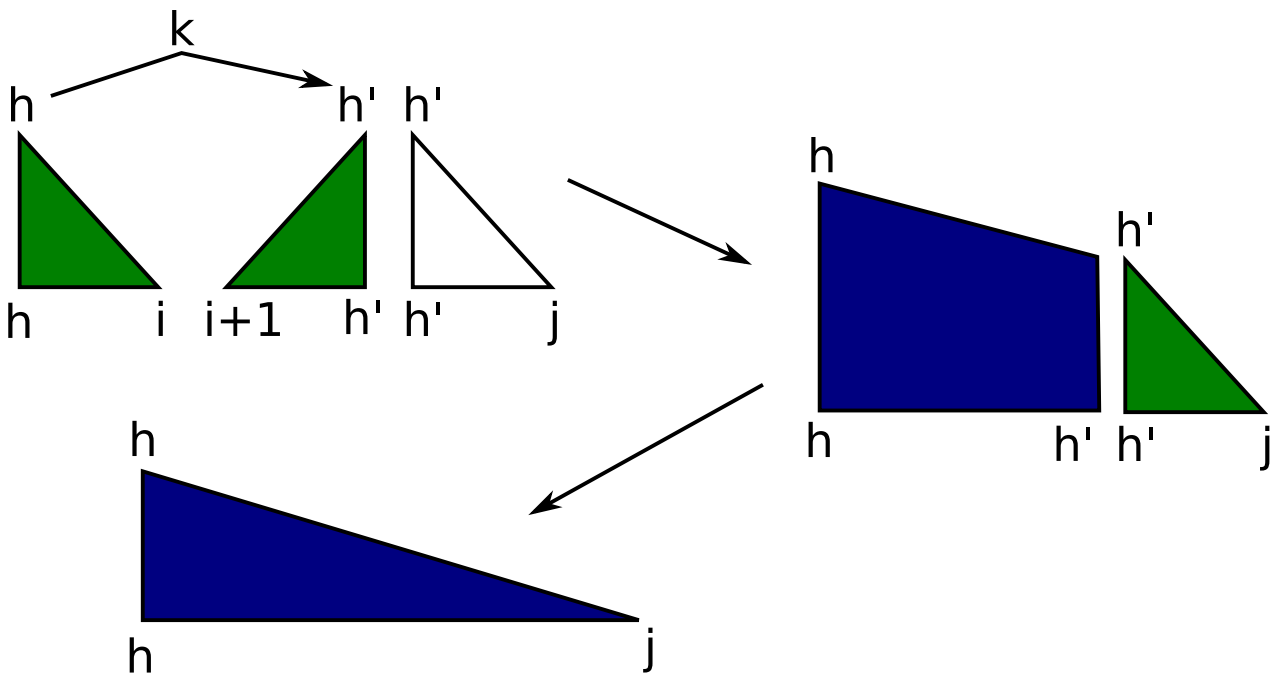
- Chart item filled in a bottom-up manner
  - First do all strings of length 1, then 2, etc. just like CKY



- Weight of new item: $\max_{l,j,k} \; w(A) \times w(B) \times w^{k}_{hh'}$
- Algorithm runs in $O(|L|n^5)$
- Use back-pointers to extract best parse (like CKY)

Arc-factored Projective Parsing

- $O(|L|n^5)$ is not that good
- [Eisner 1996] showed how this can be reduced to $O(|L|n^3)$
  - Key: split items so that sub-roots are always on periphery



Eisner's algorithm detailed: chart items

Chart items

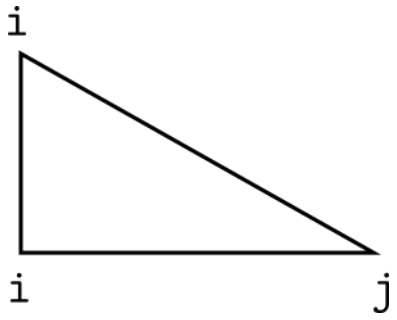
An *item* is a triple (i, j, t) where:

- i and j are positions in the input sentence and range from 1 to n
- $t \in \{L, R, B\}$ tells which endwords are active (left, right, or both, respectively)

Left

Item (i, j, L) asserts that a projective tree can be built over $i \ldots j$ such that:

$$\forall_{k:i \leq k \leq j} \, (i \rightarrow^{\star} k)$$



Eisner's algorithm detailed: chart items

Chart items

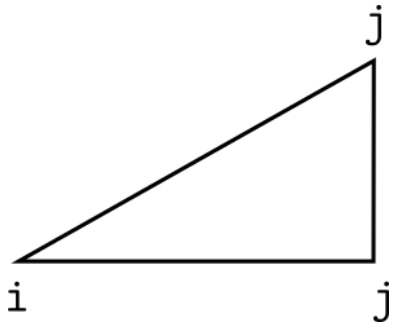
An *item* is a triple (i, j, t) where:

- i and j are positions in the input sentence and range from 1 to n
- $t \in \{L, R, B\}$ tells which endwords are active (left, right, or both, respectively)

Right

An item (i, j, R) asserts that a projective tree can be built over $i \ldots j$ such that:

$$\forall_{k:i \leq k \leq j} \, (j \rightarrow^{\star} k)$$



Eisner's algorithm detailed: chart items

Chart items

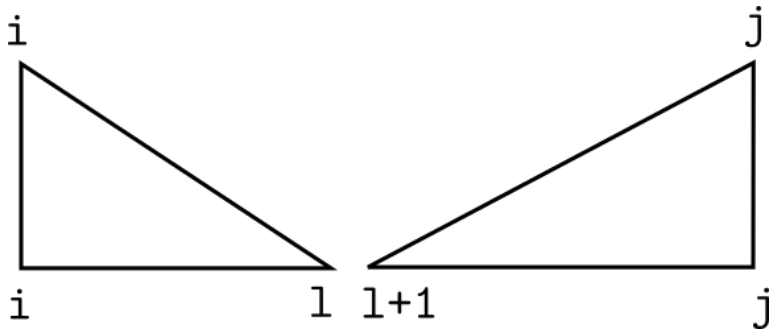
An *item* is a triple (i, j, t) where:

- i and j are positions in the input sentence and range from 1 to n
- $t \in \{L, R, B\}$ tells which endwords are active (left, right, or both, respectively)

Both

An item (i, j, B) asserts that $\exists_{l:i \leq l < j}$ such that two projective trees can be built over $i \ldots l$ and $l + 1 \ldots j$, respectively, and:

$$\forall_{k:i \leq k \leq l} \, (i \rightarrow^{\star} k) \quad \wedge \quad \forall_{k:l < k \leq j} \, (j \rightarrow^{\star} k)$$



Eisner's algorithm detailed: weighted items

Weighted items

A *weighted item* is a pair $((i, j, t), x)$, denoted $(i, j, t) : x$, where:

- (i, j, t) is an item
- $x \in \mathcal{R}_{\geq 0}$ is the corresponding weight

Assertion

Weighted item $(i, j, t) : x$ asserts that:

- A projective tree corresponding to (i, j, t) with weight x exists

Eisner's algorithm detailed: inference rules

Initialize

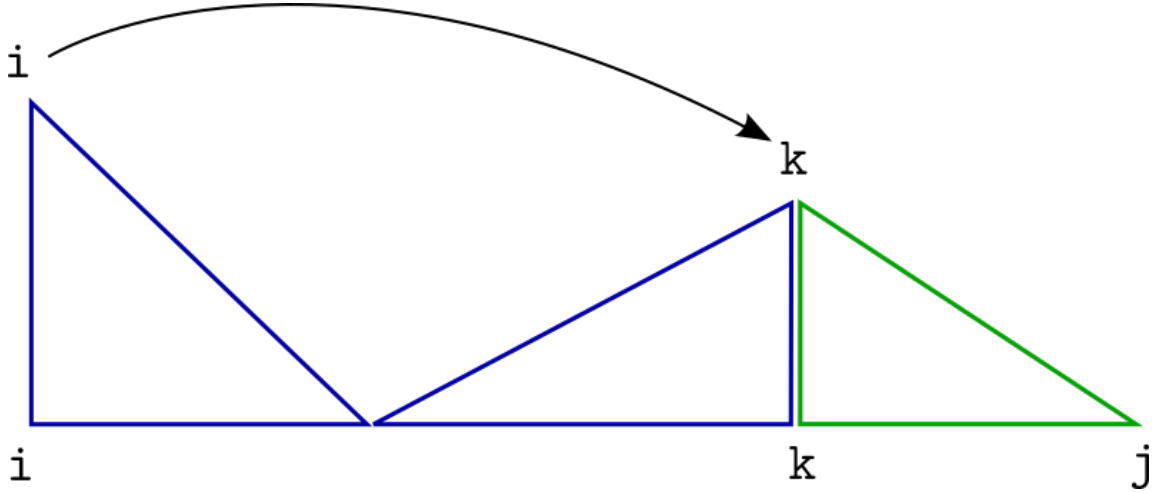
$$\text{init left:} \quad \frac{}{(i,i,L):1} \quad i \in \{1,\ldots,n\}$$

$$\text{init right:} \quad \frac{}{(i,i,R):1} \quad i \in \{1,\ldots,n\}$$

Eisner's algorithm detailed: inference rules

Combine

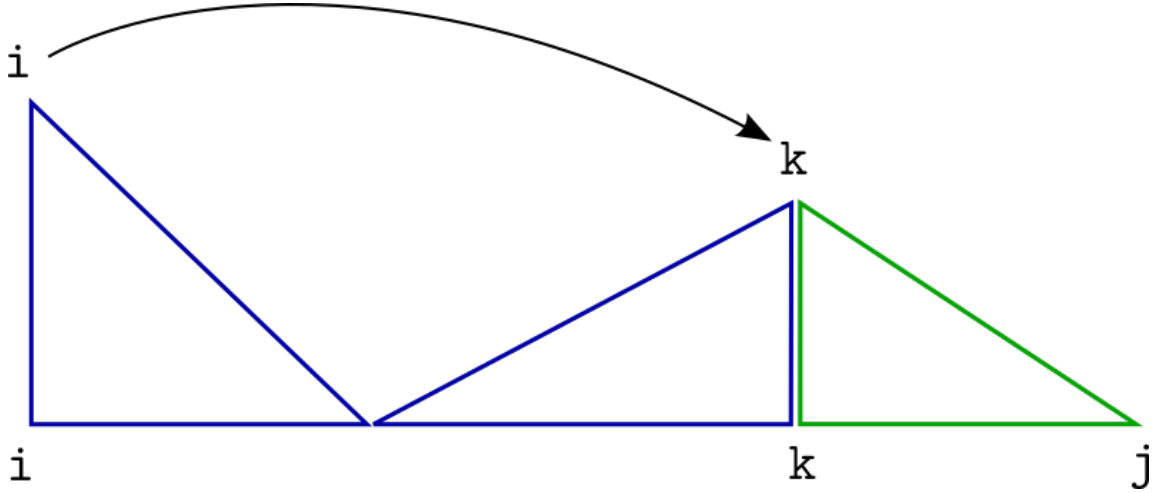
$$\text{combine left:} \quad \frac{(i,k,B) \qquad (k,j,L)}{(i,j,L)}$$



Eisner's algorithm detailed: inference rules

Combine

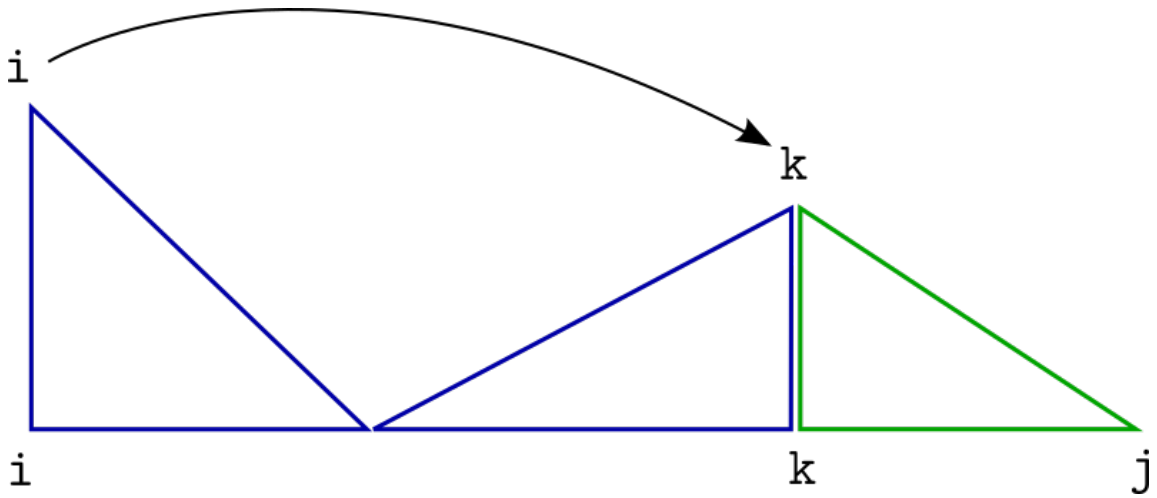
combine left: $$\frac{(i,k,B) : x \qquad (k,j,L) : y}{(i,j,L) : ?}$$



Eisner's algorithm detailed: inference rules

Combine

$$\text{combine left:} \quad \frac{(i,k,B) : x \qquad (k,j,L) : y}{(i,j,L) : x \cdot y \cdot w_{ik}^{\ell}} \qquad \ell \in \mathcal{L}$$

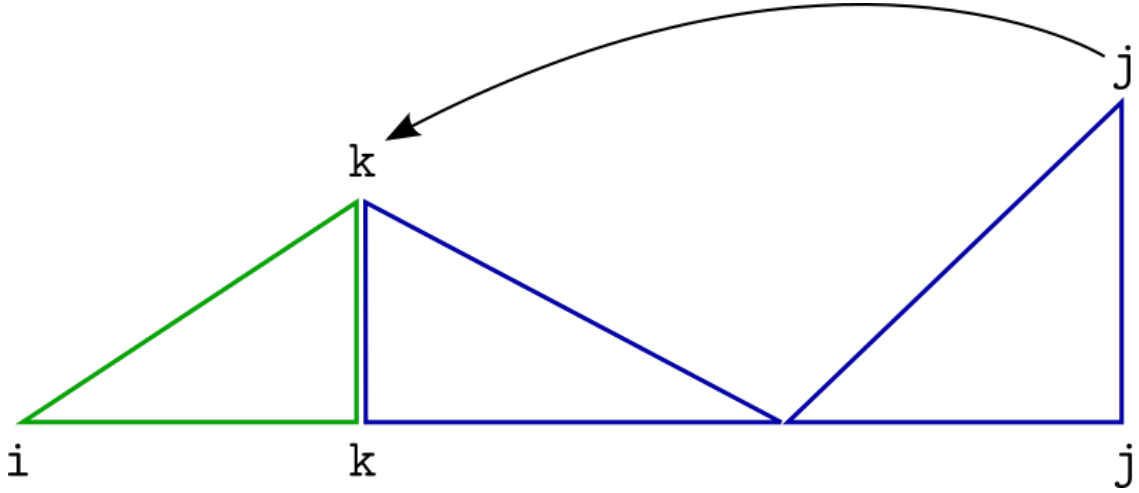


$\mathcal{L}$: set of permissible labels
w_{ik}^{ℓ}: weight of edge (i, j, ℓ)

Eisner's algorithm detailed: inference rules

Combine

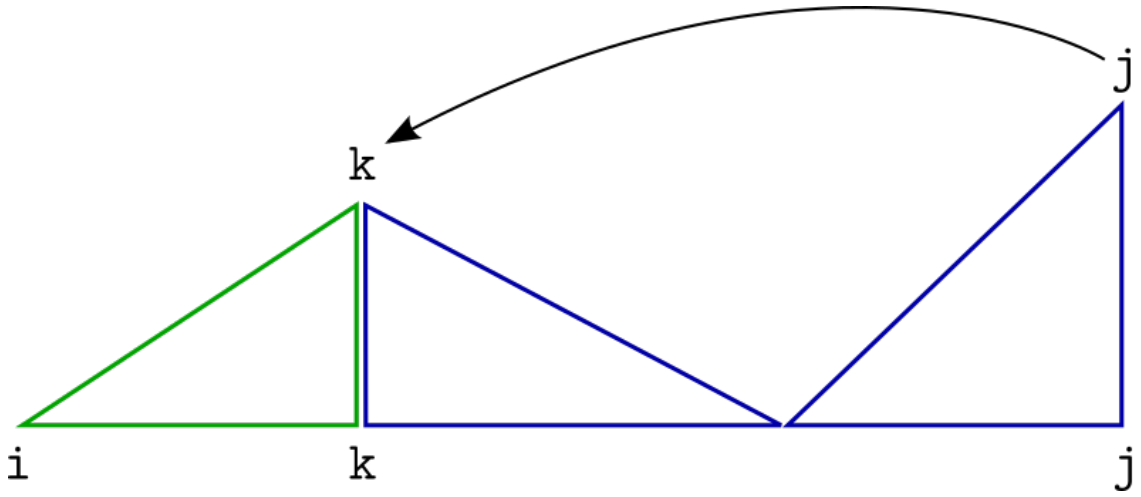
combine right: $$\frac{(i,k,R) \quad (k,j,B)}{(i,j,R)}$$



Eisner's algorithm detailed: inference rules

Combine

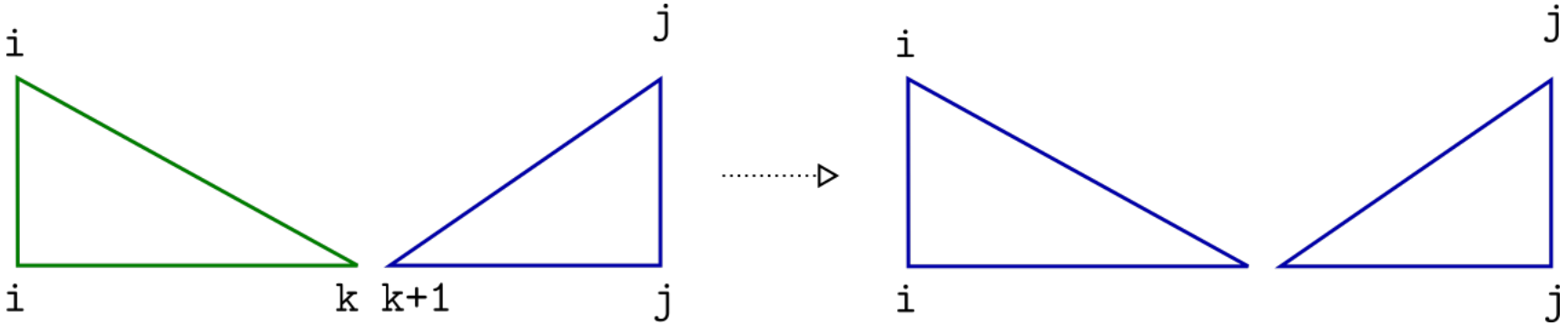
combine right: $$\frac{(i,k,R) : x \qquad (k,j,B) : y}{(i,j,R) : x \cdot y \cdot w^{\ell}_{jk}} \quad \ell \in \mathcal{L}$$



Eisner's algorithm detailed: inference rules

Combine

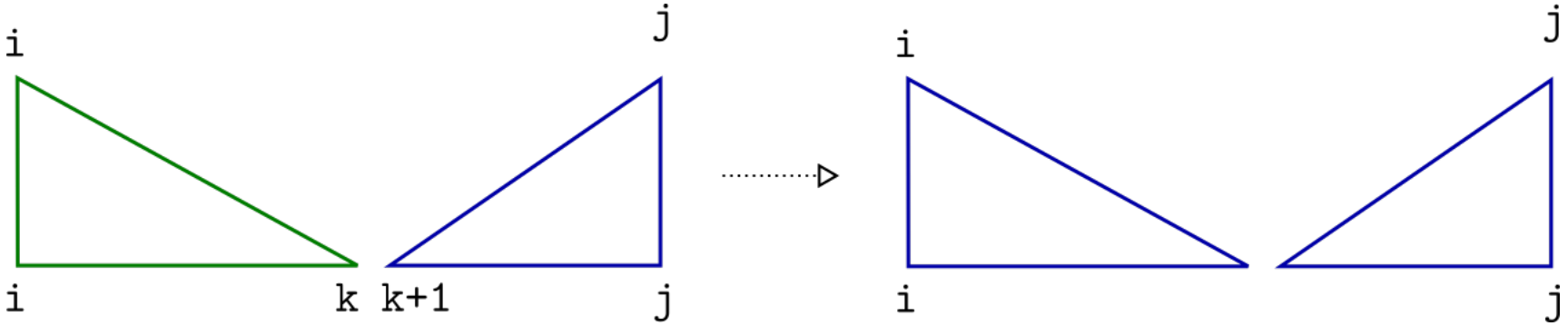
combine both: $$\frac{(i,k,L) \quad (k{+}1,j,R)}{(i,j,B)}$$



Eisner's algorithm detailed: inference rules

Combine

combine both: $$\frac{(i,k,L) : x \qquad (k{+}1,j,R) : y}{(i,j,B) : x{\cdot}y}$$



Inference in Arc-Factored Models

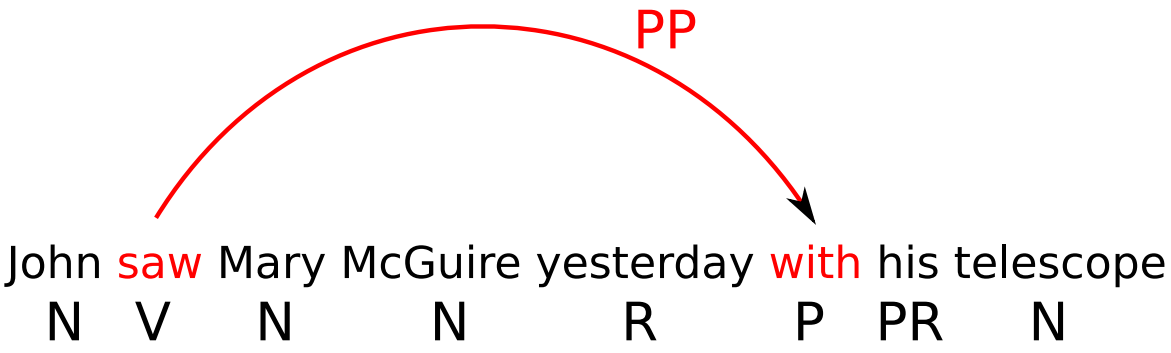
- Non-projective case
  - $O(|L|n^2)$ with the Chu-Liu-Edmonds MST algorithm
- Projective case
  - $O(|L|n^3)$ with the Eisner algorithm
- But we still haven't defined the form of w_{ij}^k
- Or how to learn these parameters

Arc weights as linear classifiers

$$w_{ij}^{k} = e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)}$$

- Arc weights are a linear combination of features of the arc, **f**, and a corresponding weight vector **w**
- Raised to an exponent (simplifies some math ...)
- What arc features?
- [McDonald et al. 2005] discuss a number of binary features

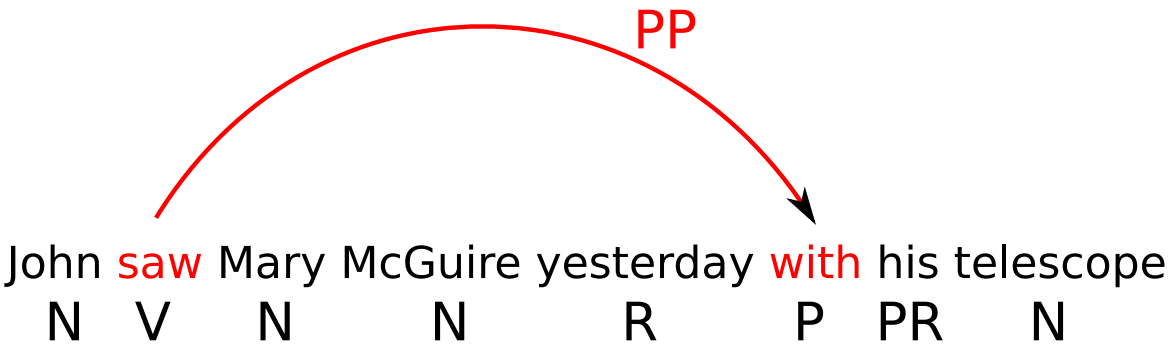
Arc Features: $\mathbf{f}(i, j, k)$



- Features from [McDonald et al. 2005]:
  - Identities of the words w_i and w_j and the label l_k

head=saw & dependent=with

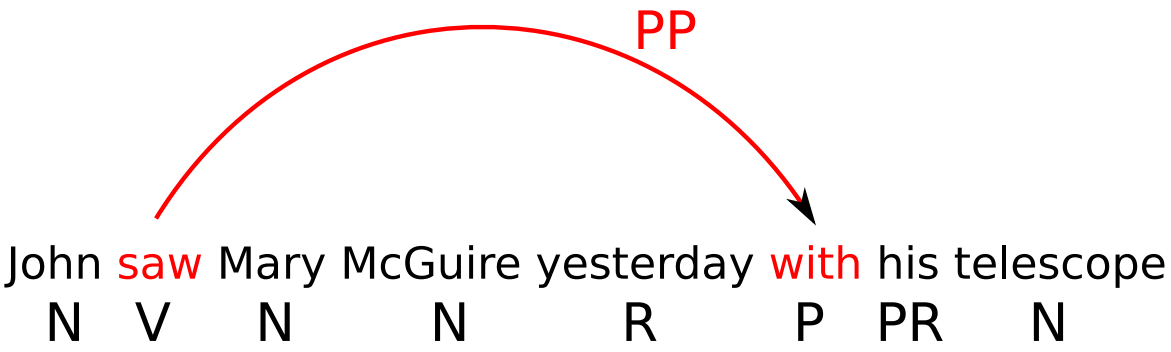
Arc Features: $\mathbf{f}(i, j, k)$



- Features from [McDonald et al. 2005]:
  - Part-of-speech tags of the words w_i and w_j and the label l_k

head-pos=Verb & dependent-pos=Preposition

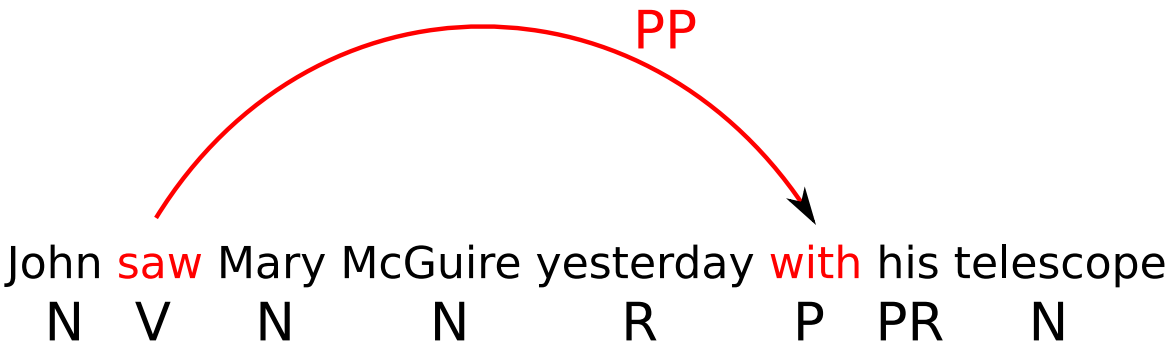
Arc Features: $\mathbf{f}(i, j, k)$



- Features from [McDonald et al. 2005]:
  - Part-of-speech of words surrounding and between w_i and w_j

inbetween-pos=Noun
inbetween-pos=Adverb
dependent-pos-right=Pronoun
head-pos-left=Noun
...

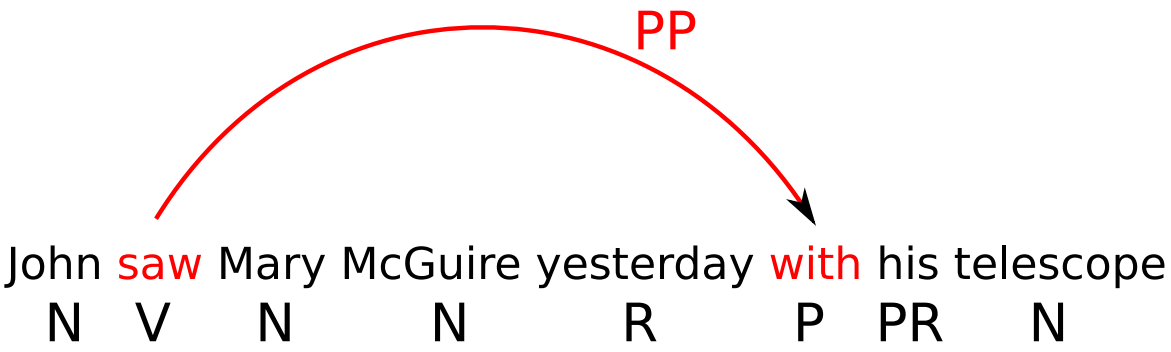
Arc Features: $\mathbf{f}(i, j, k)$



- Features from [McDonald et al. 2005]:
  - Number of words between w_i and w_j, and their orientation

arc-distance=3
arc-direction=right

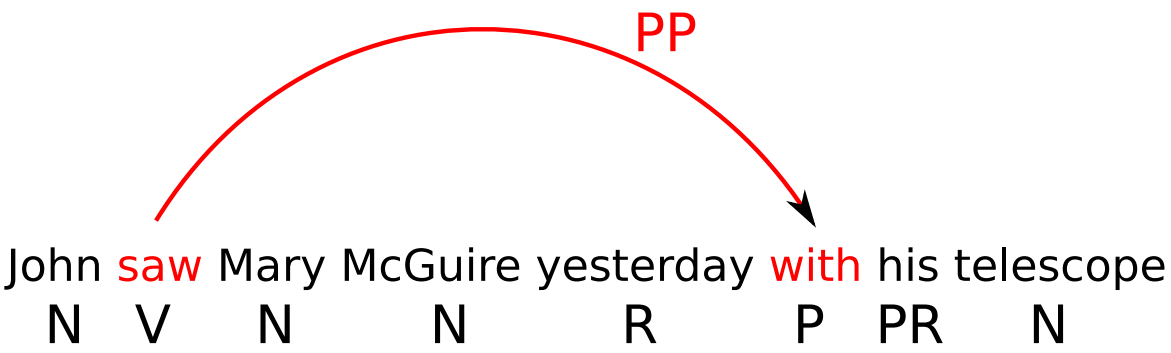
Arc Features: $\mathbf{f}(i, j, k)$



- Label features

arc-label=PP

Arc Features: $\mathbf{f}(i, j, k)$



- Combos of the above

head-pos=Verb & dependent-pos=Preposition & arc-label=PP
head-pos=Verb & dependent=with & arc-distance=3
...

- No limit: any feature over arc (i, j, k) or input x

Learning the parameters

- We can then re-write the inference problem

$$
\begin{aligned}
G &= \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k = \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)} \\
&= \arg\max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} e^{\mathbf{w} \cdot \mathbf{f}(i,j,k)} \\
&= \arg\max_{G \in T(G_x)} \sum_{(i,j,k) \in G} \mathbf{w} \cdot \mathbf{f}(i,j,k) \\
&= \arg\max_{G \in T(G_x)} \mathbf{w} \cdot \sum_{(i,j,k) \in G} \mathbf{f}(i,j,k) = \arg\max_{G \in T(G_x)} \mathbf{w} \cdot \mathbf{f}(G)
\end{aligned}
$$

- Which we can plug into online learning algorithms

Perceptron: Pseudo code

Training data: $\mathrm{T} = \{(x_i\ ,\ y_i\ )\}_{i=1}^{|T|}$

$w = 0$
$for\ \ n : 1..N$
  $for\ \ i : 1..|T|$
    $y^* = Parse(x^i, w)$
    $if\ \ y^* \neq y^i$
      $w = Update(w, y^*, y^i)$
$return\ \ w$

$Parse(x, w)$
$return\ \ argmax_{y \in GEN(x)} \sum_{(i,l,j) \in A_y} \sum_{k=1}^{K} f_k(i, l, j, x).w_k$

$Update(w, y^*, y^i)$
$for\ \ k : 1..K$
  $for\ \ (i, l, j) \in A_{y^*}$
    $w_k = w_k - f_k(i, l, j, x)$
  $for\ \ (i, l, j) \in A_{y^i}$
    $w_k = w_k + f_k(i, l, j, x)$