Parsing Natural Language

Parsing Without Grammar: Graph Based Parsing

Grammarless Parsing – Graph based

- The second approach often found in the literature is called „Graph-based parsing“
- Usually applied to dependency parsing problems
- In this framework, we basically score trees again and have an efficient way of determining the tree with the highest score
  - This is different to CKY, since we have no grammar!

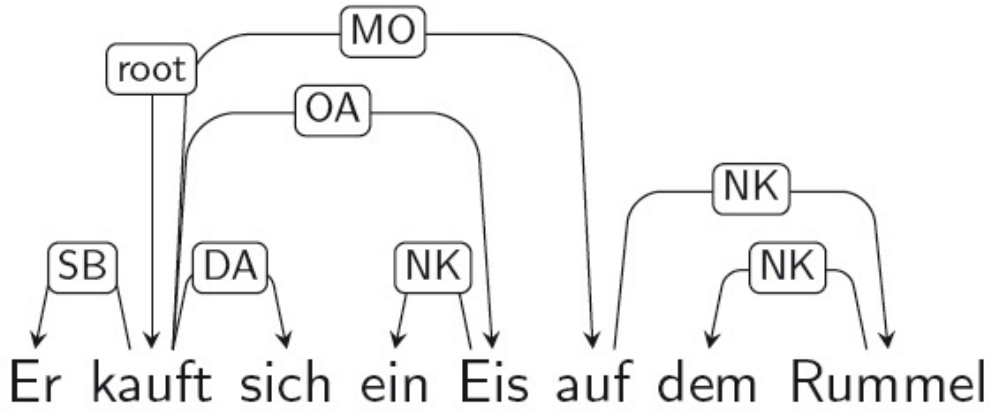
Grammarless Dependency Parsing

- Given a data set D, consisting of a set of pairs (x, G), with x being a sentence and G being a dependency tree
- We are looking for a function, which produces a valid dependency tree for our input x
- For this purpose we are determining all **valid trees** and find the best one via scoring as usual:

$$G = \underset{g \in valid\ trees}{\operatorname{argmax}} \; score(x, g)$$

Valid Dependency Trees

- It was easy to understand, what valid means, once we had a grammar at hand, but now we don’t, so what does valid mean in this context?
- Attributes of the tree:
  - Connected (you can reach root from every token)
  - Acyclic (cause it is a tree)
  - Single Head per token!



Grammarless Dependency Parsing

$$G = \underset{g \in valid\ trees}{\text{argmax}} \; score(x, g)$$

- Recall:

Transforming a structured problem into an easier learnable one can be done by abusing the repetitive structure and recombining the global problem by combining the results of the best local models

➔A graph consists of edges!

➔ Arc-Factored Model

Arc-Factored Model

- We score our tree as the sum of the score of its edges:

$$G = \underset{g \in valid\ trees}{argmax}\ score(x, g)$$

$$score(x, g) = \sum_{a \in A} score(x, a)$$

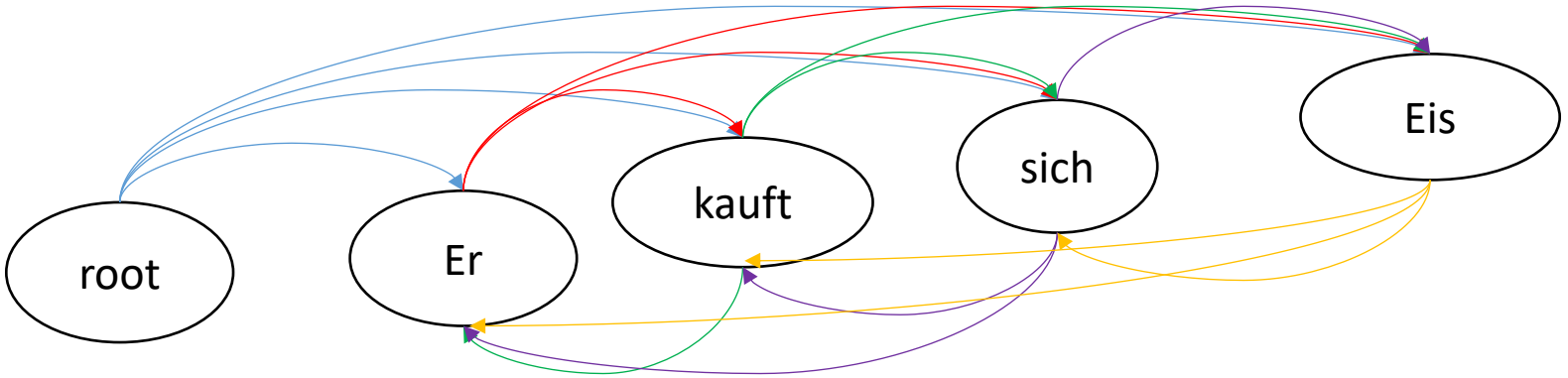
- 2 problems:
  1. How to find all valid trees, and how to get the best among them
  2. How to score an edge

For now let us assume we know how to score an edge…

Arc-Factored Model

- General sketch of graph based dependency parsing:

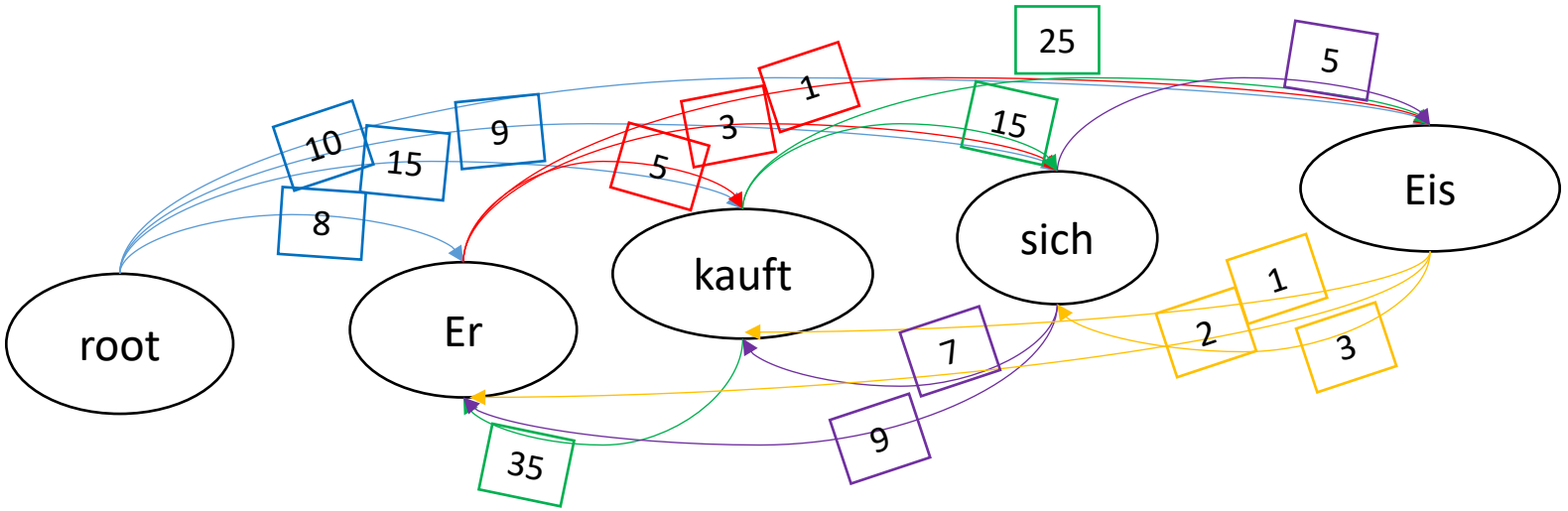
1. Add all possible edges („arcs“) into the graph



Dependency Parsing – Arc-Factored Model

- General sketch of graph based dependency parsing:

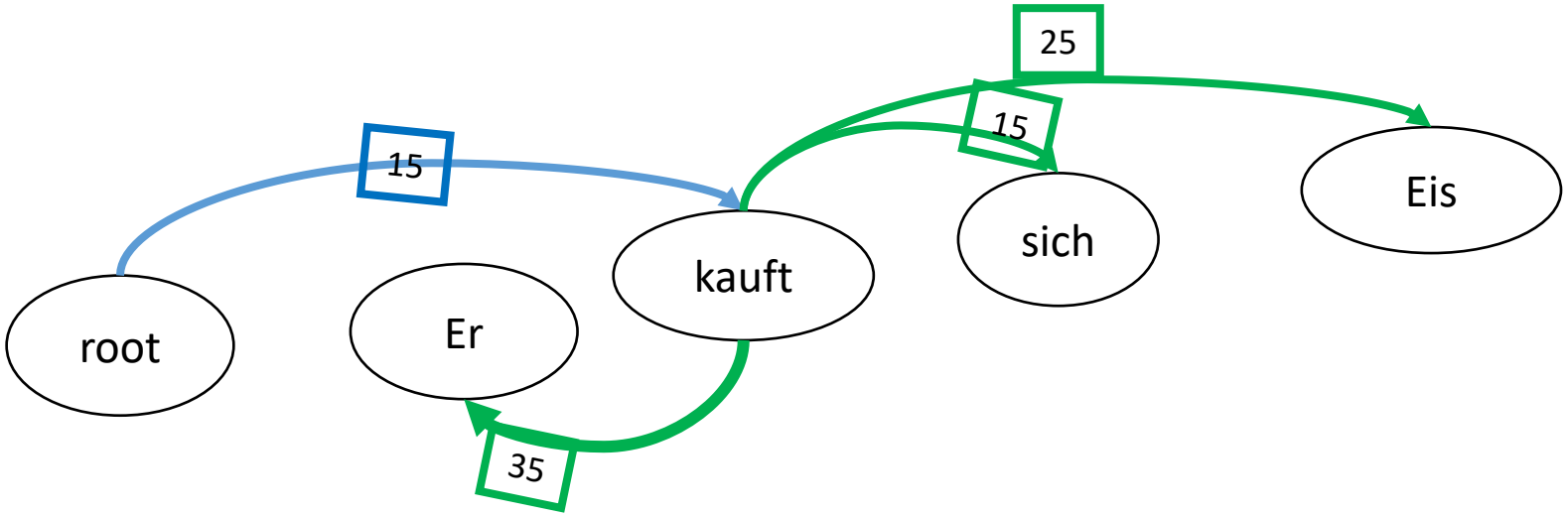
1. Add all possible edges („arcs“) into the graph
2. Score every edge



Dependency Parsing – Arc-Factored Model

- General sketch of graph based dependency parsing:

1. Add all possible edges („arcs“) into the graph
2. Score every edge
3. Find the Maximum Spanning tree according to these scores



Maximum spanning tree

- A spanning tree in a graph is a connected tree with n nodes and $n - 1$ edges
- The Maximum spanning tree is the tree with the highest sum of edge-scores

➔ Efficient algorithms from graph-theory are available!

Solved via Chu-Liu Edmonds!

$$G = \underset{g \in valide\ trees}{argmax}\ score(x, g)$$

Dependency Parsing

Graph-based Parsing: Chu-Liu Edmonds algorithm

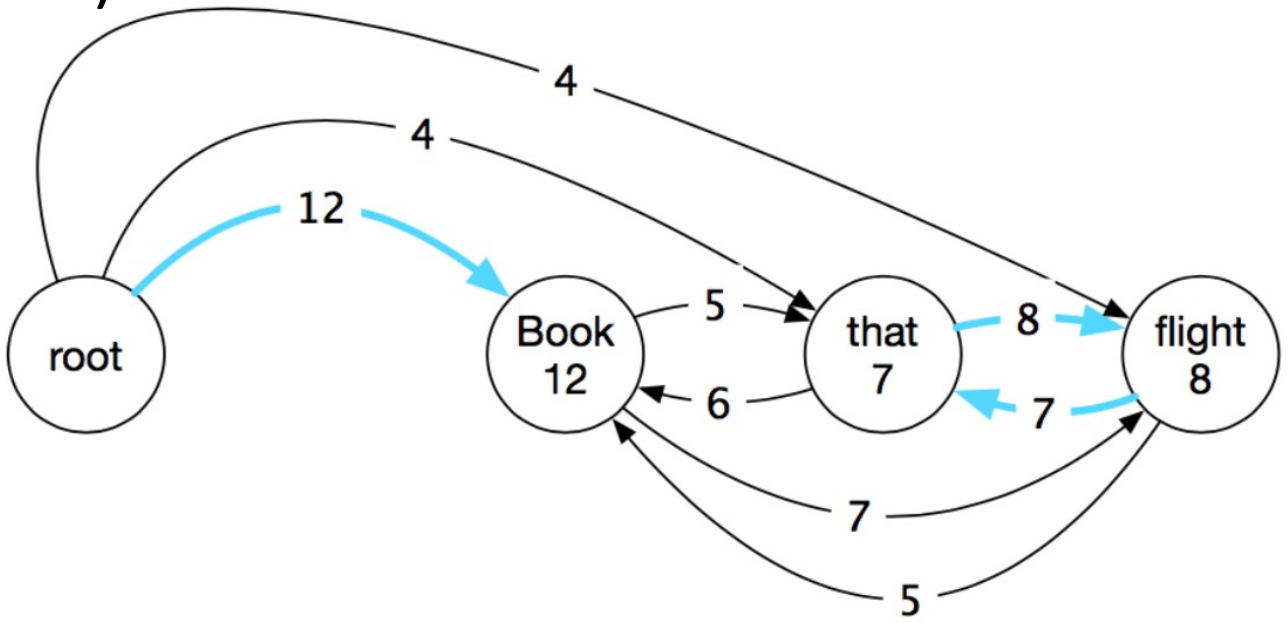
Dependency Parsing – Chu Liu Edmonds CLE

- Rather easy algorithm to find the Maximum Spanning tree
- Input: Given is a graph G having a score at every edge („edge-weights“)

- Output: The according Maximum Spanning tree

Dependency Parsing – Chu Liu Edmonds CLE

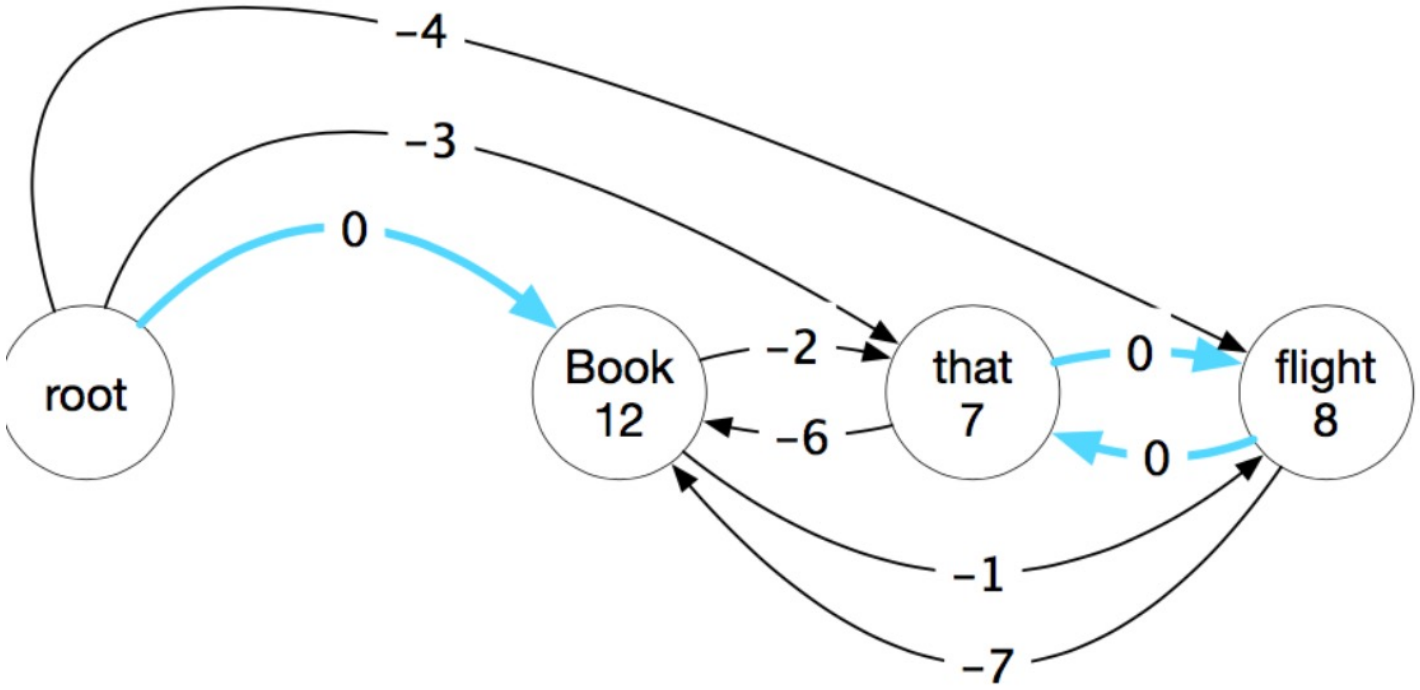
- For every node, find the incoming edge with maximum weight (and store it in the node)



- If this is a valid tree, return it!

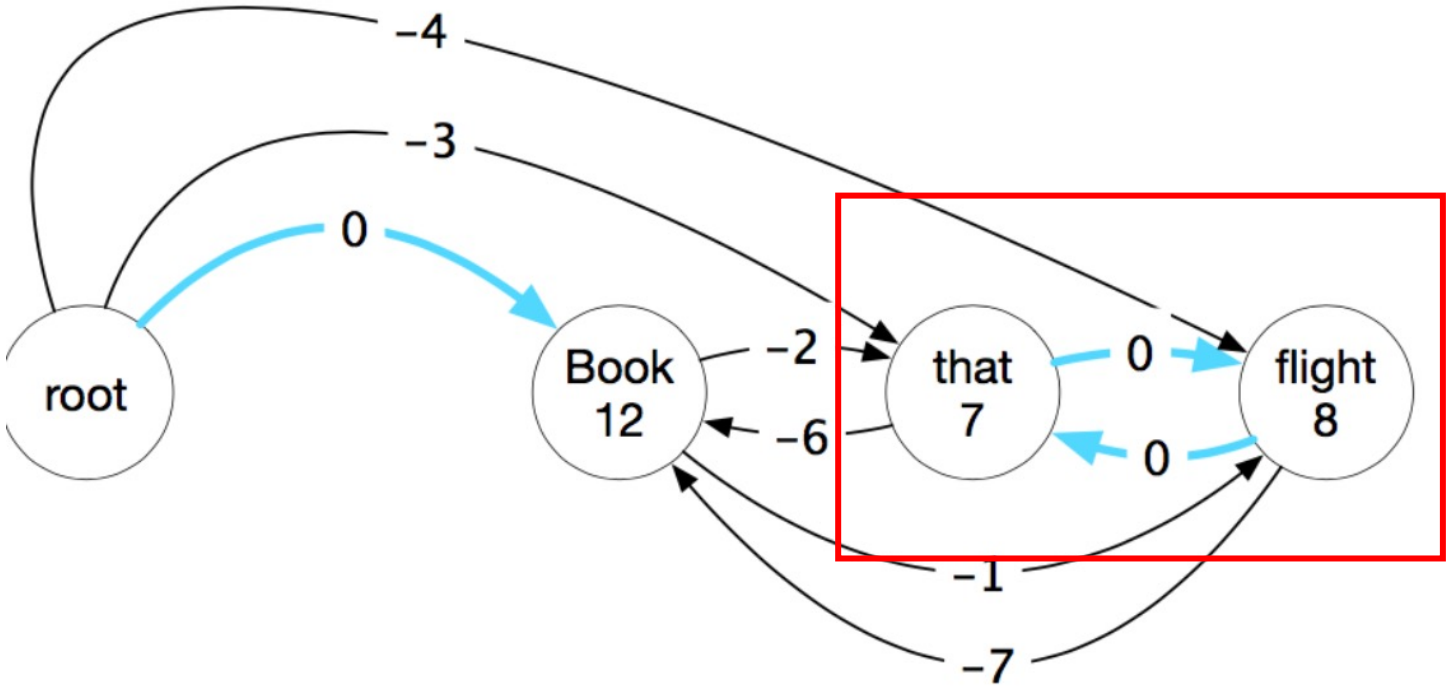
Dependency Parsing – Chu Liu Edmonds CLE

- If there are circles, reduce each incoming edge by weight stored in the node



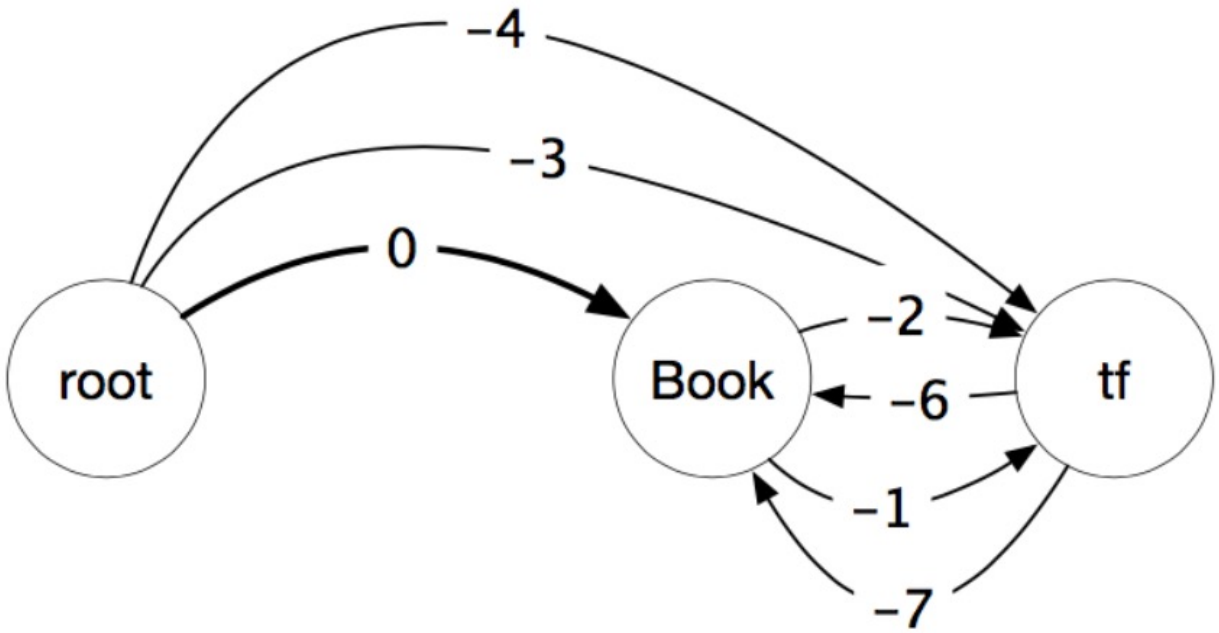
Dependency Parsing – Chu Liu Edmonds CLE

- Detect the edges involved in the circle (DFS in the nodes)



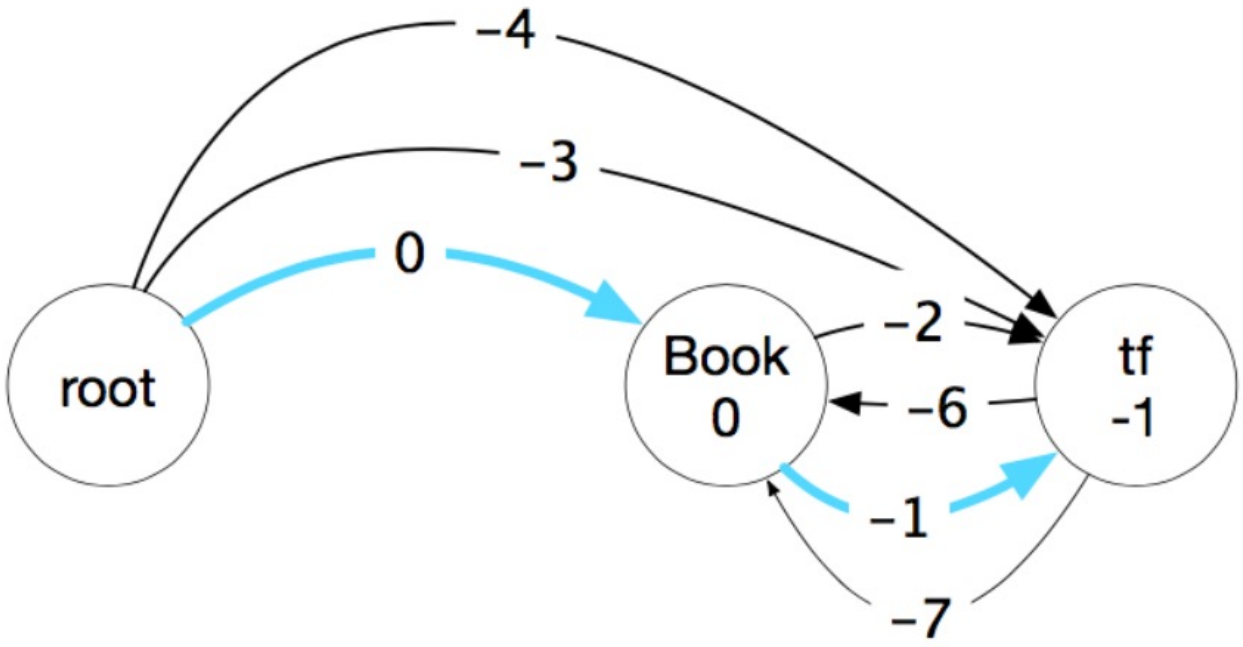
Dependency Parsing – Chu Liu Edmonds CLE

- Contract the involved nodes into a single node.
- Every incoming (to the new node) edge remains in the graph



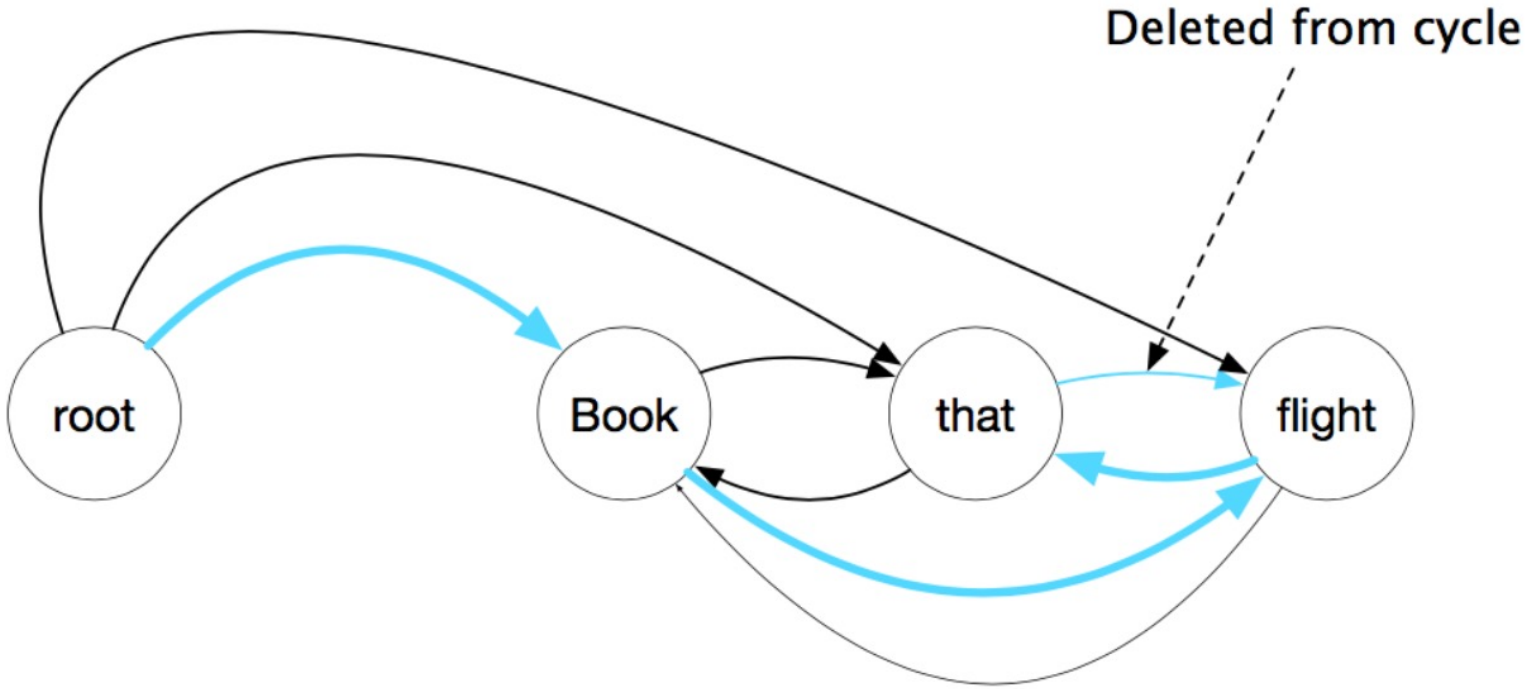
Dependency Parsing – Chu Liu Edmonds CLE

- Apply the algorithm **recursively** on the resulting graph



Dependency Parsing – Chu Liu Edmonds CLE

- Expand the node and delete an appropriate edge
- If no further cycles → done



Dependency Parsing – Chu Liu Edmonds CLE

function MAXSPANNINGTREE($G=(V,E)$, *root*, *score*) **returns** *spanning tree*

$F \leftarrow []$
$T' \leftarrow []$
score' $\leftarrow []$
for each $v \in V$ **do**
  bestInEdge $\leftarrow \text{argmax}_{e=(u,v)\in E}$ *score*[*e*]
  $F \leftarrow F \cup$ *bestInEdge*
  for each $e=(u,v) \in E$ **do**
    score'[*e*] $\leftarrow$ *score*[*e*] $-$ *score*[*bestInEdge*]

if $T=(V,F)$ is a spanning tree **then return** it
else
  $C \leftarrow$ a cycle in F
  $G' \leftarrow$ CONTRACT(G, C)
  $T' \leftarrow$ MAXSPANNINGTREE(G', *root*, *score'*)
  T $\leftarrow$ EXPAND(T', C)
  return T

function CONTRACT(G, C) **returns** *contracted graph*

function EXPAND(T, C) **returns** *expanded graph*

Dependency Parsing – CLE

- The algorithm of Chu-Liu and Edmond detects the Maximum Spanning tree in $O(n^3)$
- But remember: We had to score all edges in **advance**! So we can only score using features over a single edge and do not have access to partial structures as is the case for Shift-Reduce-Parsers
- **How to determine edge-scores?**

Edge Scores

- Describe an edge using arbitrary features
  - POS-Tags
  - Edge-direction
  - Distance of tokens
  - Text of tokens
  - Etc…
- Use the structured Perceptron and inference based learning!

Structured Perceptron for Dependency Parsing

1. Start with all feature weights at 0
2. Score all edges (sum of all active feature weights)
3. Apply CLE to get the best predicted tree under the current weights
4. Score the correct tree (as stored in the data)
5. If the Hinge-Loss is not 0:
   - Update the features using the Perceptron update rule („Gradient Descent“)
6. Repeat