School of CIT –
Social Computing
Research Group

Technical
University
of Munich

Natural Language Processing
IN2361

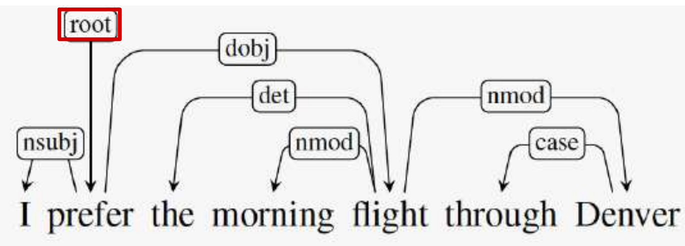
Prof. Dr. Georg Groh

Chapter 18
Dependency Parsing

- content is based on [1]
- certain elements (e.g. equations or tables) were taken over or taken over in a modified form from [1]
- citations of [1] or from [1] are omitted for legibility
- errors are fully in the responsibility of Georg Groh
- BIG thanks to Dan and James for a great book!

Dependency Parsing

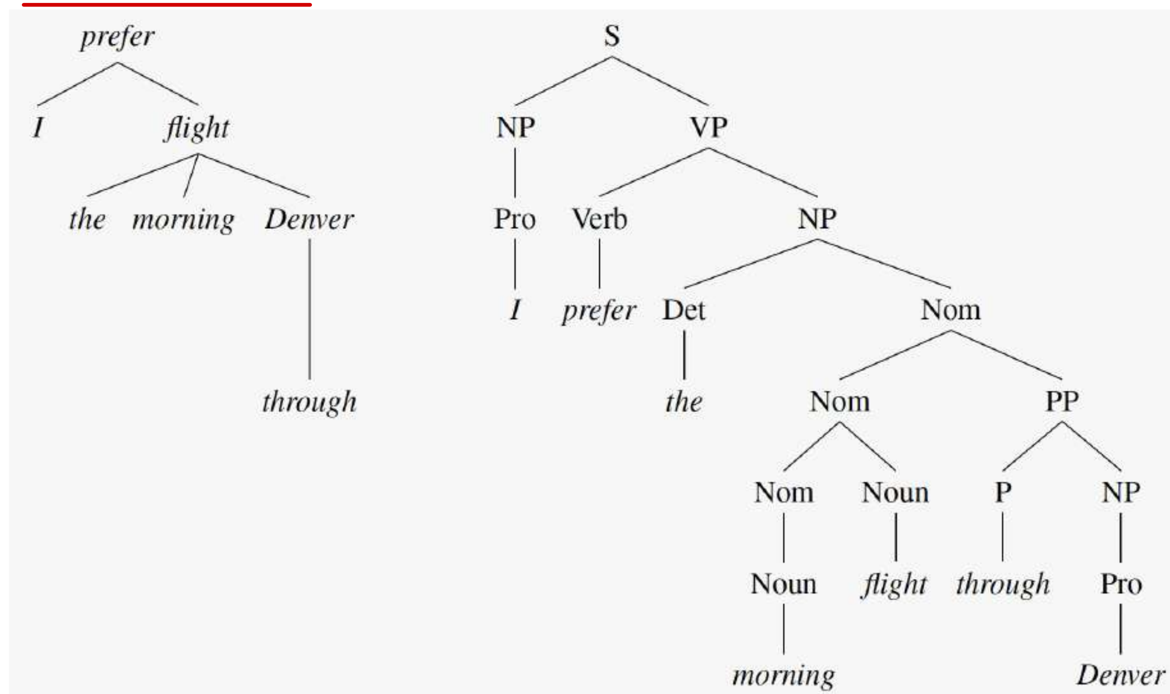
idea: syntactic structure of a sentence is described solely in terms of the words + associated set of directed binary relations among the words



simpler and more abstract

dependency parse

constituent parse



- relations: between head and dependent
- root: head of entire structure
- every major constituent has a head (e.g. flight, Denver)

Dependency Parsing

advantages:

- DP abstracts away from word-order, representing only the information that is necessary for the parse
  - (constituent (phrase-structure) grammar for languages with free word order: would require rule for every word order case)

- head-dependent relations provide an approximation to the semantic relationship between predicates and their arguments → useful for many applications (e.g. co-reference resolution, question answering and information extraction)
  - traditional parses also contain that information, in principle, but head needs to be determined with rules;
    difference in “information content” btw. parse tree (“more syntactical”) and dependency tree (“slightly more semantical”) is more significant in languages with free word order

Dependency Relations

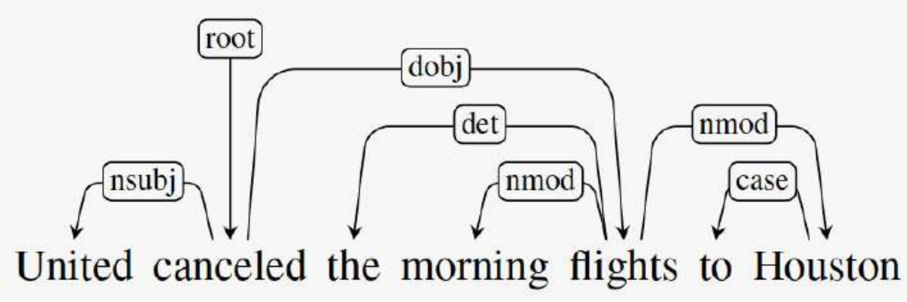
- grammatical relations link heads to dependents
- relation types ←→ grammatical functions
- Universal Dependencies project. examples:

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Dependency Relations

- relations:
  - clausal relations describe syntactic roles with respect to a predicate (often a verb)
  - modifier relations that categorize ways that words that can modify their heads.

example:



causal: NSUBJ, DOBJ
modifier: NMOD, DET, CASE

Dependency Relations

relation examples:

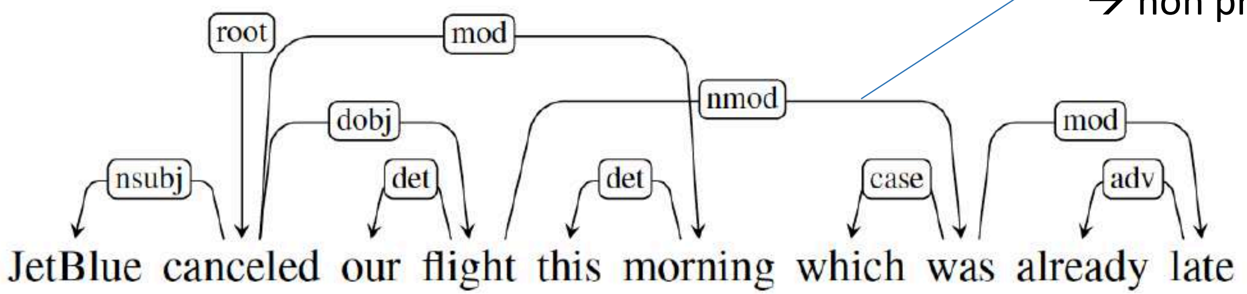
Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Dependency Trees and Projectivity

- DP: resulting graph of relations, often a dependency tree:
  - single designated root node with no incoming arcs
  - except root node, each vertex has exactly one incoming arc
  - ∃ unique path from the root node to each vertex

- arc (h,d) projective if ∃ path from h to every word w that in the sentence lies between h and d.
  tree is projective if all arcs are projective

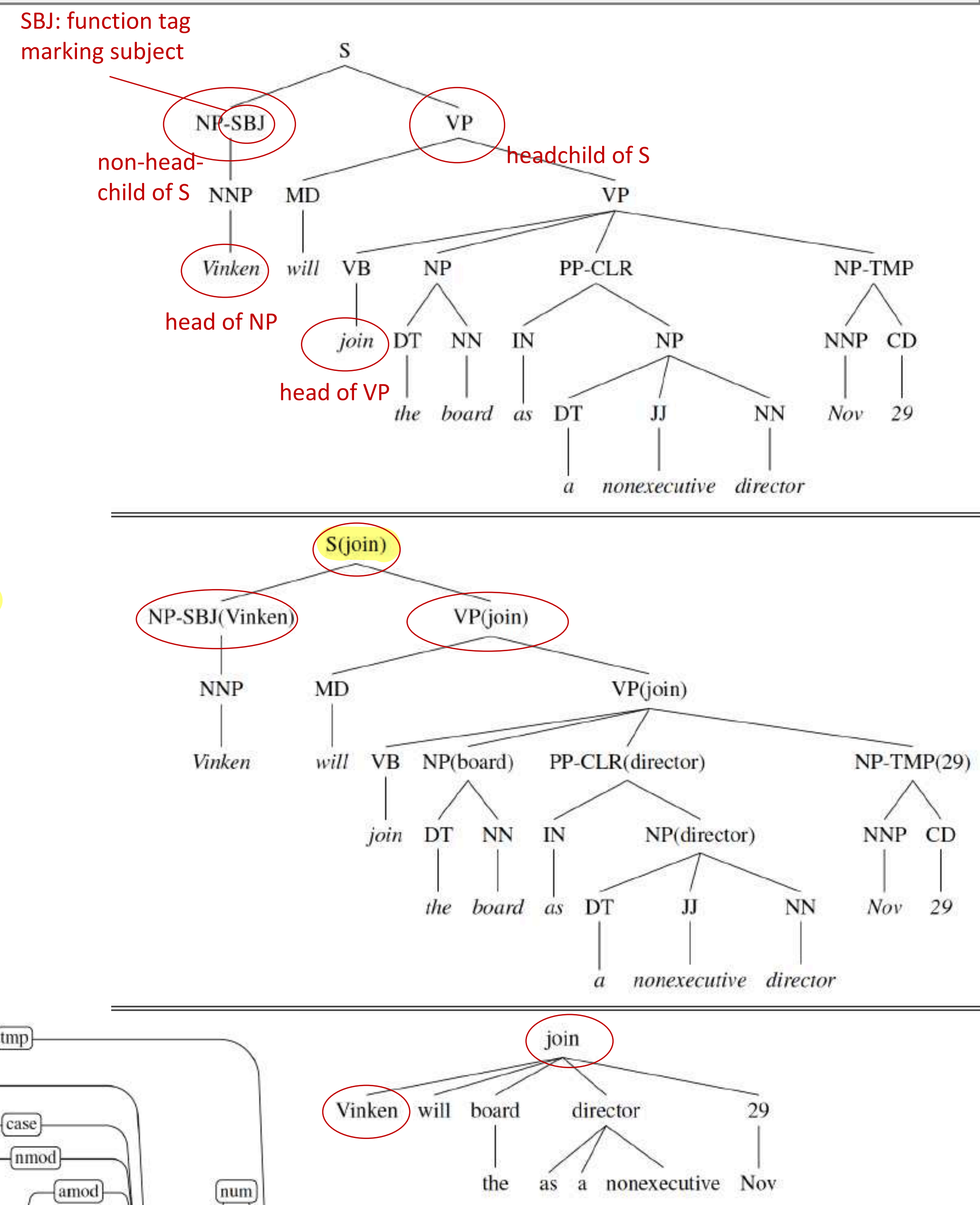
non-projective arc
→ non projective DP tree

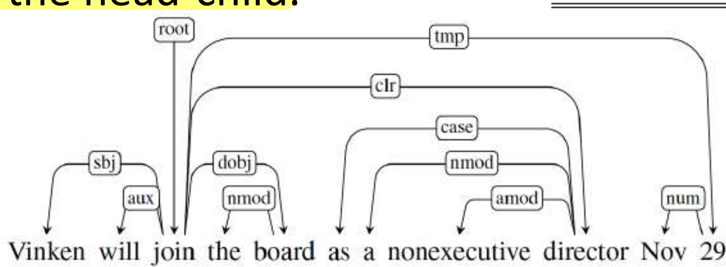


CF parse trees are necessarily projective → DP parse trees generated from CF treebank parse trees are also projective

Dependency Treebanks

- Either handcrafted or via hand-correcting automated DP parses or via "translating" constituent-based (CF) parse trees (e.g. from Penn Treebank)
- "translating" constituent-based parse trees into DP trees:
  - identify heads and dependents
    - Mark head child of each node using head rules developed for use in lexicalized probabilistic parsers
    - In the dependency structure, make the head of each non-head child depend on the head of the head-child.
  - classify arcs



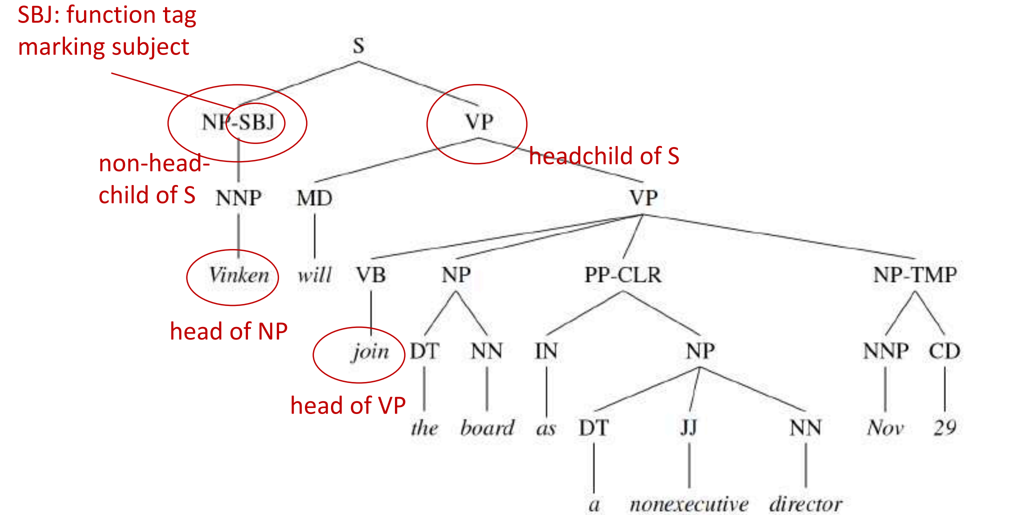


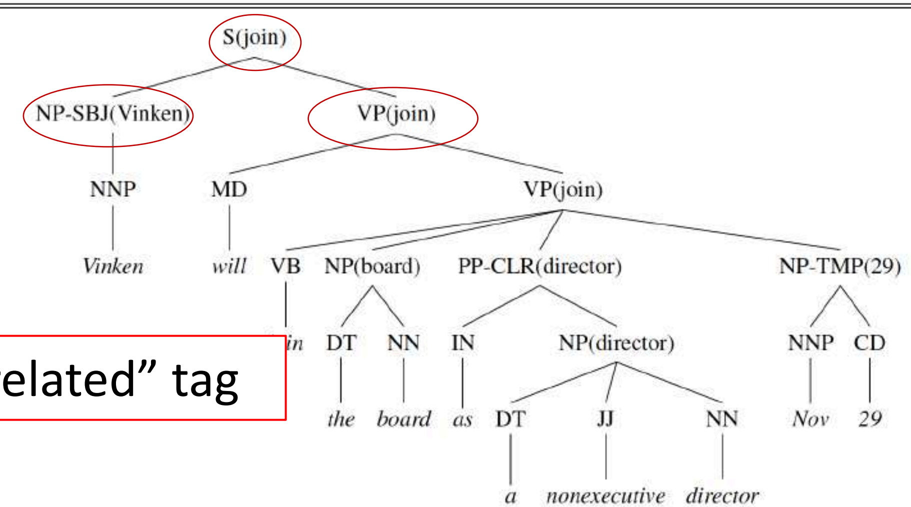
Dependency Treebanks

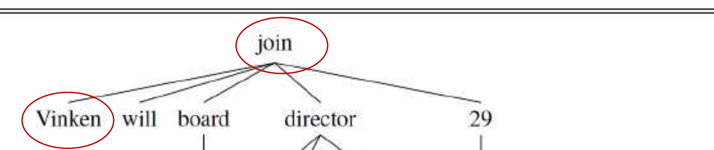
- Either handcrafted or via hand-correcting automated DP parses or via “translating” constituent-based (CF) parse trees (e.g. from Penn Treebank)
- “translating” constituent-based parse trees into DP trees:
  - identify heads and dependents
    - Mark head child of each node
      …eloped
      probabilistic parsers
    - In the dependency structure make the head of eac… head child depend on the head of the head-child.
  - classify arcs

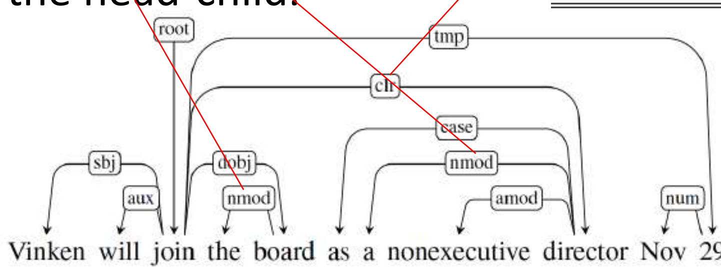
misclassified;
should be det instead

clr: “closely related” tag









Transition-Based Dependency Parsing

adaptation of Shift-Reduce parsing of CF grammars using a stack (Shift-Reduce: main operation: remove top two elements B and C from stack and replace with A if rule A→BC)

greedy, linear, left-to-right

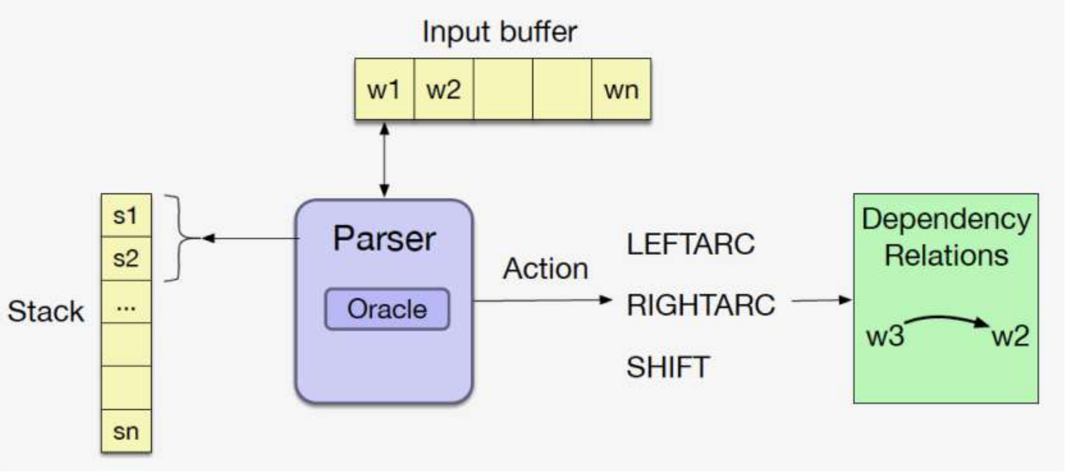
function DEPENDENCYPARSE(*words*) **returns** dependency tree

stack   input buffer   dependencies found

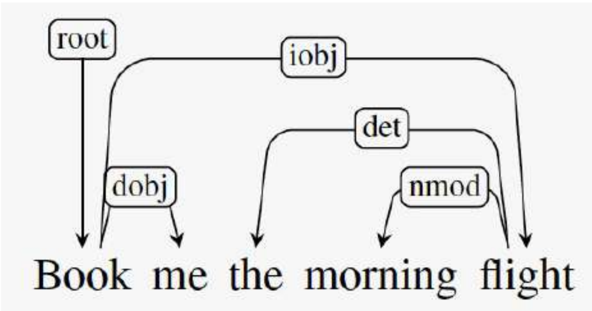
  state ← {[root], [*words*], [] } ; initial configuration
  while *state* **not final**
    t ← ORACLE(*state*) ; choose a transition operator to apply
    state ← APPLY(*t*, *state*) ; apply it, creating a new state
  return *state*

"Arc Standard" transition operators (operate on top two stack elements):

- **LEFT ARC**: Assert head-dependent relation between word at top of stack (head) and word directly beneath it (dependent); remove lower word from stack
- **RIGHT ARC**: Assert head-dependent relation between second word on stack (head) and word at top (dependent); remove word at top of stack
- **SHIFT**: Remove word from front of input buffer and push it onto stack.



Transition-Based Dependency Parsing



"Arc Standard" transition operators:

- **LEFT ARC** : Assert head-dependent relation between word at top of stack (head) and word directly beneath it (dependent);
  remove lower word from stack
- **RIGHT ARC** : Assert head-dependent relation between second word on stack (head) and word at top (dependent); remove word at top of stack
- **SHIFT** : Remove word from front of input buffer and push it onto stack.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Transition-Based Dependency Parsing

- ambiguity: there may be more than one path through the configuration space leading to same or different parse

- greedy: if oracle chooses wrong operator: no going back

- possibly include arc classification (n classes) into job of oracle: then need n types of left arcs and n types of right arcs: 3 → 2*n+1 operators

- transition-based approaches produce projective parse trees only → any such parse tree for a sentence with a fundamentally non-projective DP-structure will contain errors.

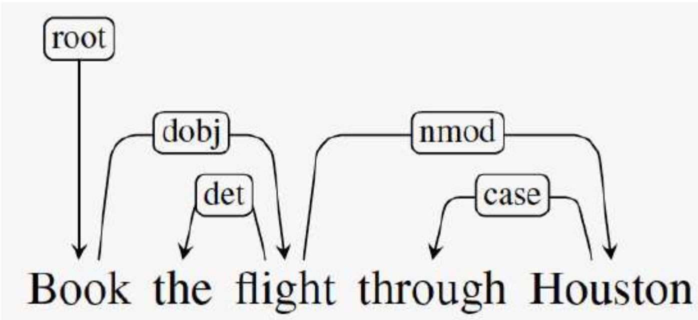
Creating an Oracle

- train oracle with supervised ML:
  input: (features of) configuration, output: transition operator

- create training data via simulating parsing process:
  - input: reference parse tree (vertices V, true dependence relations R_p)
  - output: sequence of configurations (stack, word-buffer, current set of "retrieved" true dep. relations R_c) + associated operators
  - for each configuration:

> - Choose LEFT ARC if it produces correct head-dependent relation given reference parse and current configuration
> - Otherwise, choose RIGHT ARC if
>   - (1) it produces correct head-dependent relation given reference parse and current configuration and
>   - (2) all dependents of word at top of stack have already been assigned,
> - Otherwise, choose SHIFT .

(2) ensures that a word is not popped from the stack, and thus lost to further processing, before all its dependents have been assigned to it



- Choose LEFT ARC if it produces correct head-dependent relation given reference parse and current configuration
- Otherwise, choose RIGHT ARC if
  - (1) it produces correct head-dependent relation given reference parse and current configuration and
  - (2) all dependents of word at top of stack have already been assigned,
- Otherwise, choose SHIFT .

S1: top of stack, S2: second on stack, r: type of relation;
node can have only one incoming arc, but potentially many outgoing arcs

LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$

RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t.(S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$

SHIFT: **otherwise**

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Features

- now we have sequence of configs + operators: what features of configs do we use?

- basic features used for part-of-speech tagging or partial parsing: e.g. word forms, lemmas, parts of speech, the head, dependency relation to the head

- feature templates for config (stack, buffer, R_c): e.g.

operator: left, right, or shift + label

POS tag of second word on stack

$$\langle s_1.w, op\rangle, \langle s_2.w, op\rangle \langle s_1.t, op\rangle, \langle s_2.t, op\rangle$$

word-form of first word on stack

$$\langle b_1.w, op\rangle, \langle b_1.t, op\rangle \langle s_1.wt, op\rangle$$

POS tag of first word in buffer

word-form conatenated with tag of first stack word

Features

- example config of
  United canceled the morning flights to Houston :

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

$$\langle s_1.w = flights, op = shift\rangle$$
$$\langle s_2.w = canceled, op = shift\rangle$$
$$\langle s_1.t = NNS, op = shift\rangle$$
$$\langle s_2.t = VBD, op = shift\rangle$$
$$\langle b_1.w = to, op = shift\rangle$$
$$\langle b_1.t = TO, op = shift\rangle$$
$$\langle s_1.wt = flightsNNS, op = shift\rangle$$
$$\langle \underbrace{s_1t.s_2t}_{s_1.t \circ s_2.t} = NNSVBD, op = shift\rangle$$

- feature templates typically used :

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Alternative: Neural Classifier

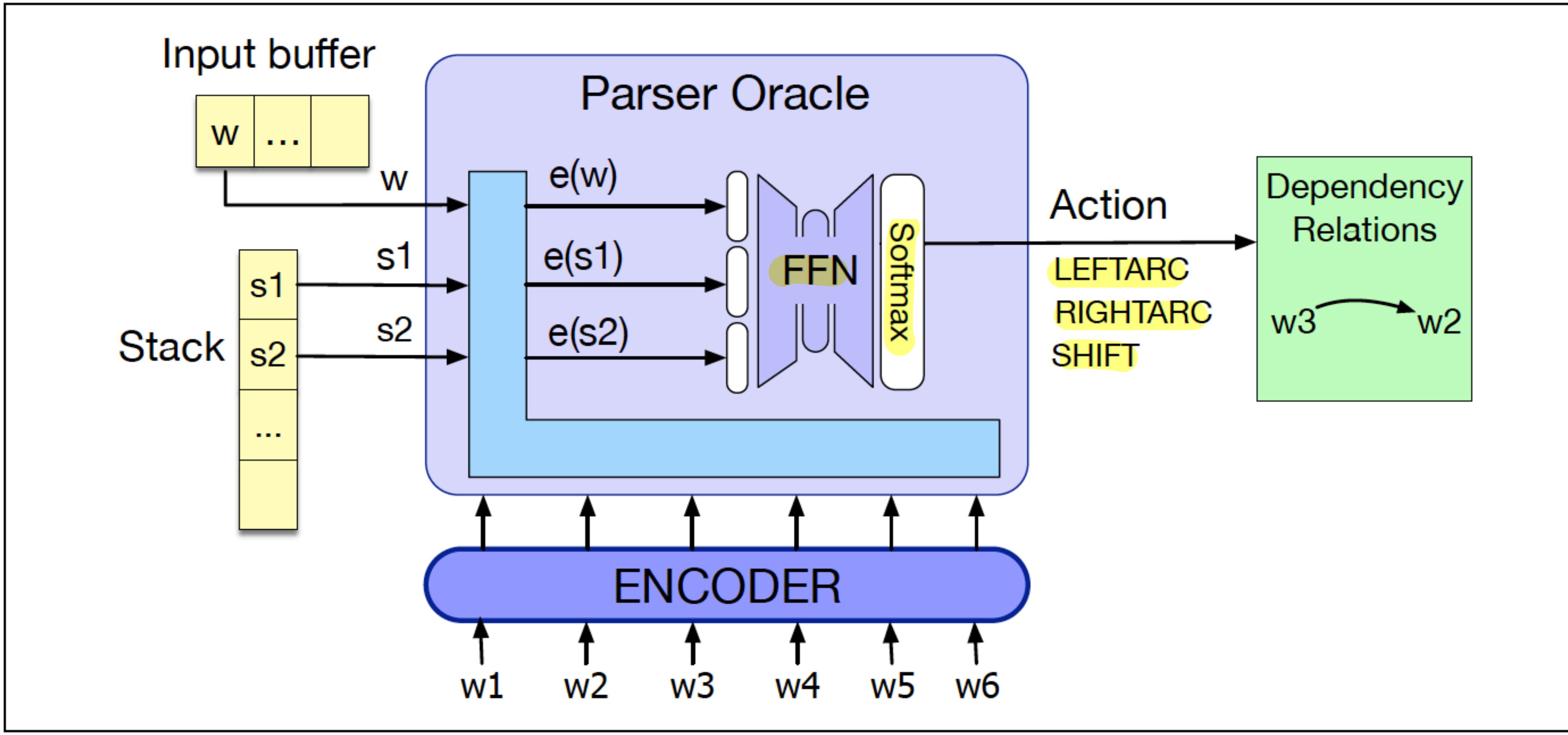
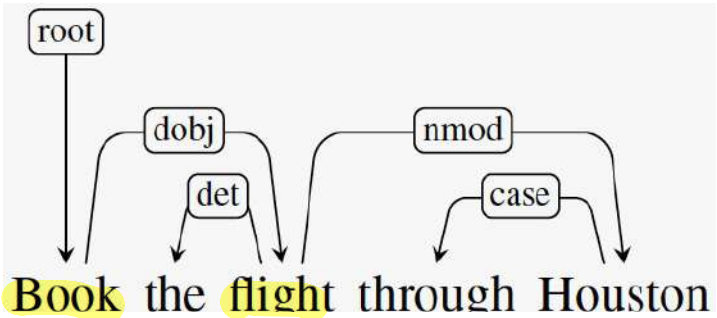


Figure 14.9 Neural classifier for the oracle for the transition-based parser. The parser takes the top 2 words on the stack and the first word of the buffer, represents them by their encodings (from running the whole sentence through the encoder), concatenates the embeddings and passing through a softmax to choose a parser action (transition).

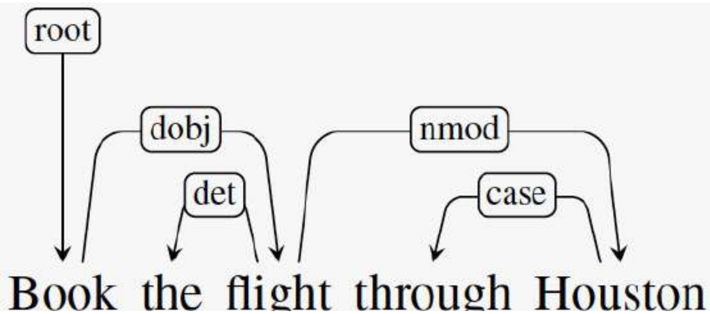
Alternative Transition Systems

- with arc-standard, the dep.rel. btw. flight and Book can be done late (in step 8): RIGHTARC cannot be applied earlier because modifiers *through Houston* have not been processed → delay: source of possible errors



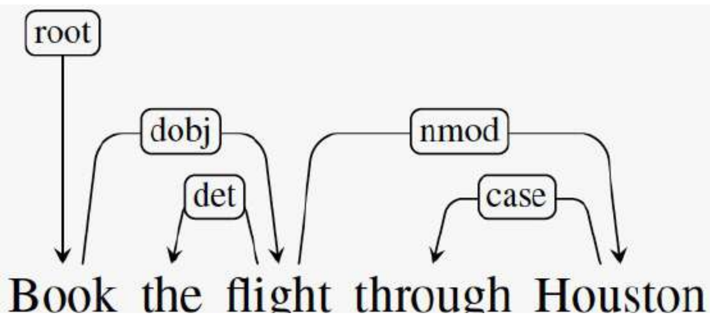
- with arc-eager ("eager to apply RIGHTARC earlier"), the dep.rel. btw. flight and Book can be established earlier (in step 4)

  - LEFT ARC : Assert head-dependent relation between word at front of input buffer and word at top of stack; pop the stack.
  - RIGHT ARC : Assert head-dependent relation between word on top of stack and word at front of input buffer; shift the word at front of input buffer to stack.
  - SHIFT : Remove word from front of input buffer and push it to stack.
  - REDUCE : Pop stack.



- LEFT ARC : Assert head-dependent relation between word at front of input buffer and word at top of stack; pop the stack.
- RIGHT ARC : Assert head-dependent relation between word on top of stack and word at front of input buffer; shift the word at front of input buffer to stack.
- SHIFT : Remove word from front of input buffer and push it to stack.
- REDUCE : Pop stack.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	



instead of removing it (as before).
now the word is still available for further dependents

- LEFT ARC : … d at front of … stack.
- RIGHT ARC : Assert head-dependent relation between word on top of stack and word at front of input buffer; **shift the word at front of input buffer to stack.**
- SHIFT : Remove word from front of input buffer and push it to stack.
- REDUCE : Pop stack.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	

Beam Search

- ML trained oracle: greedily gives best transition t given a configuration c

$$\hat{T}(c) = argmaxScore(t, c)$$

disadvantage: bad decisions cannot be undone despite good later evidence

- Beam-Search: systematically develop (by applying all possible transitions) a limited (beam width) number of possible configs (agenda); if limit is reached: only take up new configs if better than worst in agenda; score new config based on previous config and transition applied:

$$\begin{aligned} ConfigScore(c_0) &= 0.0 \\ ConfigScore(c_i) &= ConfigScore(c_{i-1}) + Score(t_i, c_{i-1}) \end{aligned}$$

Beam Search

function DependencyBeamParse(*words*, *width*) **returns** dependency tree

state ← {[root], [*words*], [], 0.0}   ;initial configuration
agenda ← ⟨*state*⟩;   initial agenda

while *agenda* **contains** non-final states
  newagenda ← ⟨⟩
  for each *state* ∈ *agenda* **do**
    for all {*t* | *t* ∈ ValidOperators(*state*)} **do**
      child ← Apply(*t*, *state*)
      newagenda ← AddToBeam(*child*, *newagenda*, *width*)
  agenda ← *newagenda*
return BestOf(*agenda*)

function AddToBeam(*state*, *agenda*, *width*) **returns** updated agenda

if Length(*agenda*) < *width* **then**
  agenda ← Insert(*state*, *agenda*)
else if Score(*state*) > Score(WorstOf(*agenda*))
  agenda ← Remove(WorstOf(*agenda*))
  agenda ← Insert(*state*, *agenda*)
return *agenda*

Graph-Based Dependency Parsing

- Parsing a sentence S is search in the space of all possible parse trees $\mathcal{G}_S$ for S:

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\operatorname{argmax}}\, score(t, S)$$

score of a tree: from score of tree edges (edge factored approach):

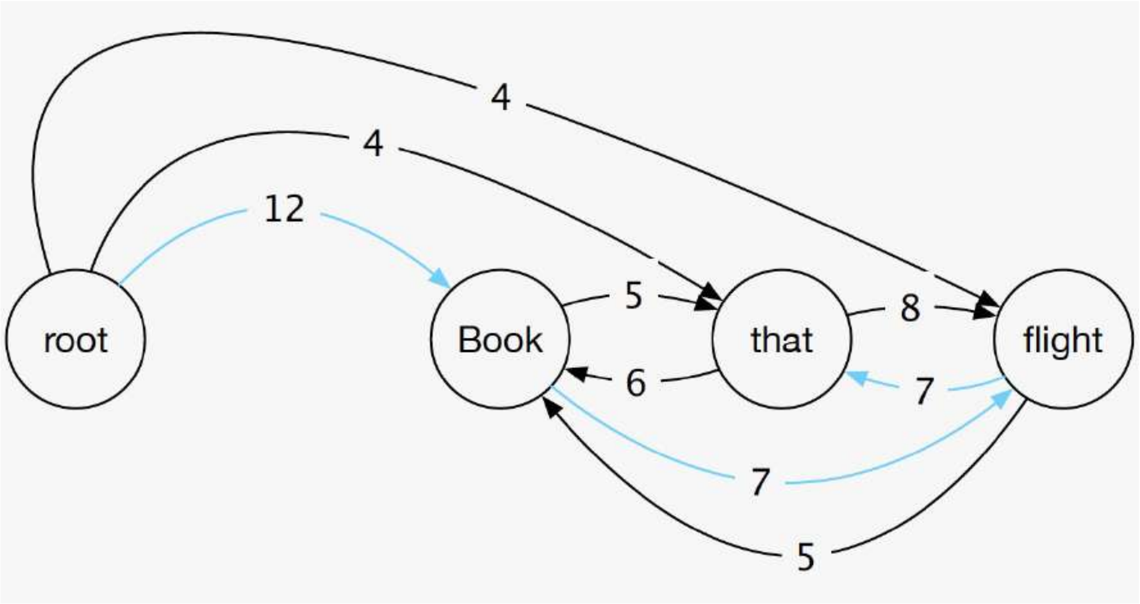
$$score(t, S) = \sum_{e \in t} score(e)$$

- advantages compared to transition-based approaches:
  - non-projective trees possible
  - better with long-range dependencies (especially in other languages than English) (scoring entire trees than just making greedy local decisions)

Maximum Spanning Tree Parsing

- Fully (except for root) connected graph:
  - vertices: words of sentence + ROOT
  - edges: all possible (head, dependent) relations
  - edge weights: edge scores

- search for maximum spanning tree in graph



graph for *Book that flight*; result: MST in blue

Search for Maximum Spanning Tree in Graph

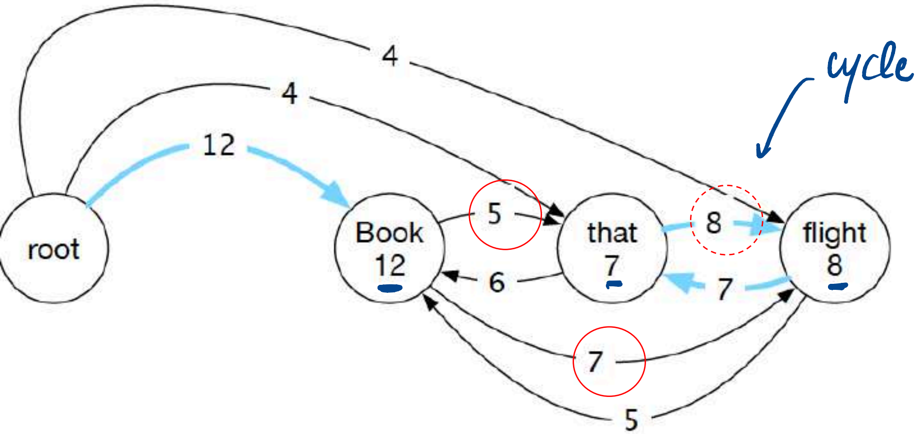
1. for each node v: choose incoming edge with max weight m(v)

2. if result is spanning tree (each node has one incoming edge and no cycles exist): done.

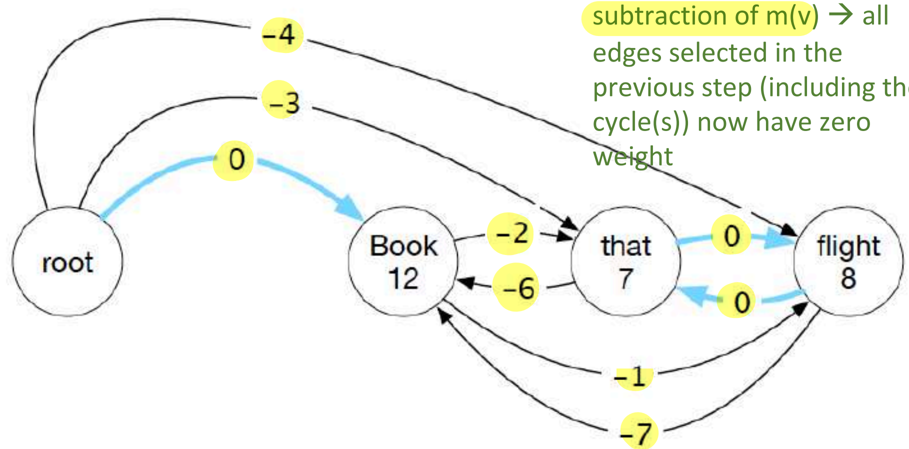
3. else: cycles exist: cleanup:

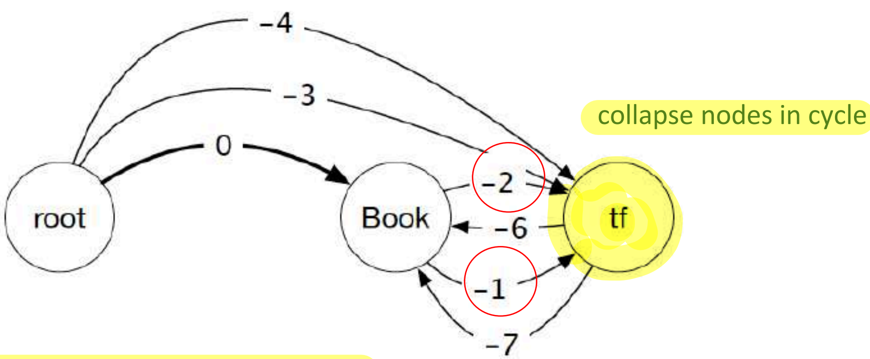
   - for each vertex v subtract m(v) from all incoming edge's weights. This does not change any MST (only relative choices among in-edges of a node matter)
   - collapse cycle into single (meta-)node

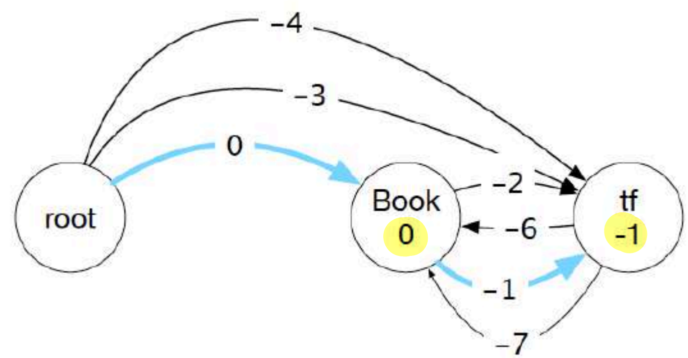
call process again recursively for cleansed graph → result: MST for cleansed graph; expand cycle (meta-)nodes back, breaking cycle by deleting obvious edge



subtraction of m(v) → all edges selected in the previous step (including the cycle(s)) now have zero weight



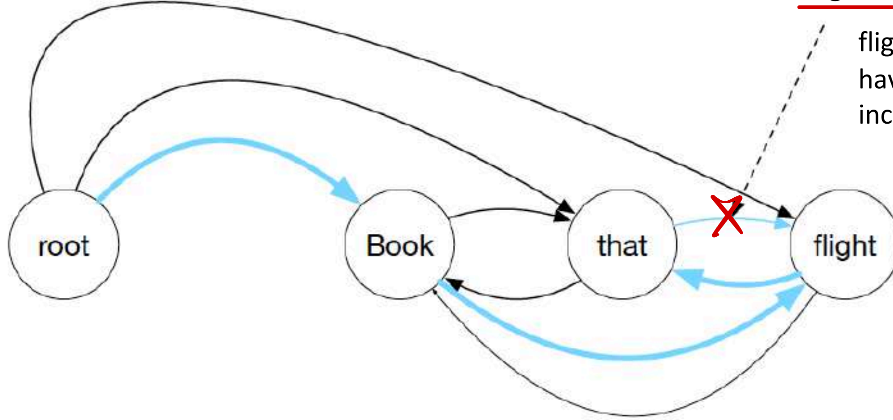




subtraction of m(v) is necessary to keep the right order among edges in next recursive call of the algorithm: if w(that, flight)=10 instead of 8, we would have had w(Book,flight)=-3 and (Book,that) would have been chosen instead of (Book,flight)

Obvious edge deleted from cycle

flight cannot have two incoming arcs



Search for Maximum Spanning Tree in Graph

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

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

Scoring

- sentence S, candidate tree T: edge-factored scoring:

$$score(S,T) = \sum_{e \in T} score(S,e)$$

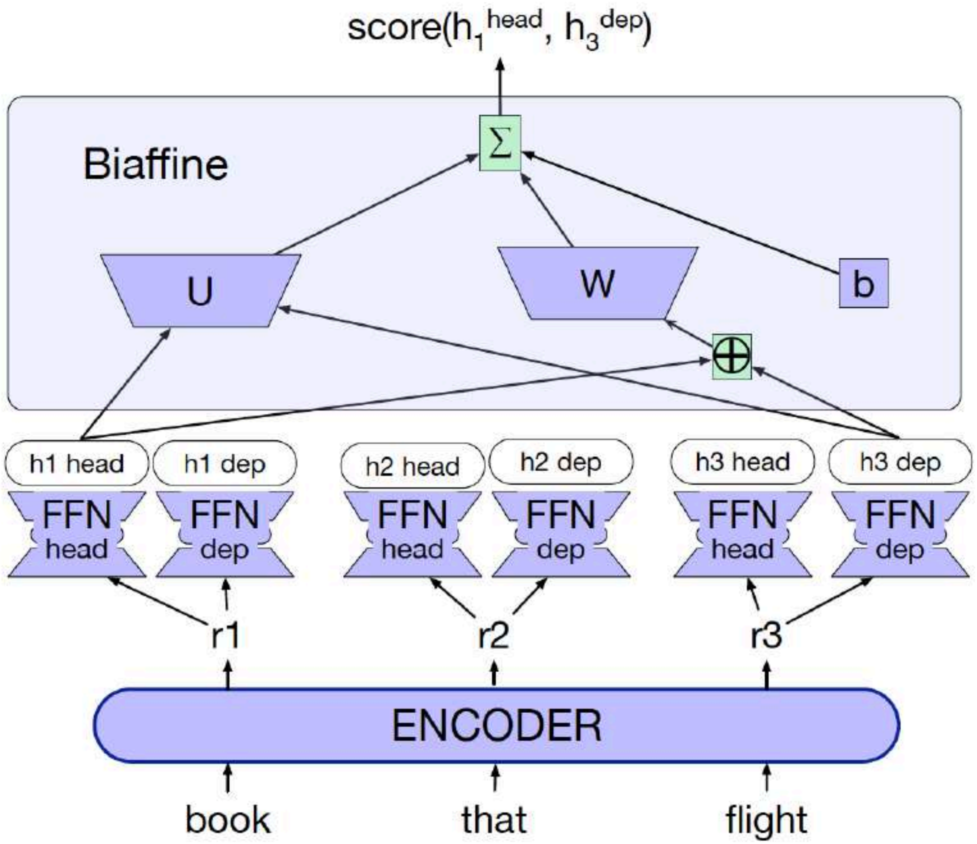
- edge scoring:

$$score(S,e) = \sum_{i=1}^{N} w_i f_i(S,e)$$

- features f_i : similar to transition-based parsing:
  - word-forms, lemmas, and parts of speech of the headword and its dependent.
  - corresponding features derived from the contexts before, after and between the words.
  - pre-trained word embeddings
  - the dependency relation itself.
  - the direction of the relation (to the right or left).
  - the distance from the head to the dependent.

NN for Scoring

$$p(i \to j) = \text{softmax}([\text{Score}(k \to j); \forall k \neq j, 1 \leq k \leq n])$$



$$\begin{aligned} \text{Score}(i \to j) &= \text{Biaff}(\mathbf{h}_i^{head}, \mathbf{h}_j^{dep}) \\ \text{Biaff}(\mathbf{x}, \mathbf{y}) &= \mathbf{x}^{\mathsf{T}}\mathbf{U}\mathbf{y} + \mathbf{W}(\mathbf{x} \oplus \mathbf{y}) + b \end{aligned}$$

$$\mathbf{h}_i^{head} = \text{FFN}^{head}(\mathbf{r}_i)$$

$$\mathbf{h}_i^{dep} = \text{FFN}^{dep}(\mathbf{r}_i)$$

Metrics

- Metrics for learning: e.g. Accuracies: Labeled Attachment Score (LAS) or Unlabeled Attachment Score (UAS)

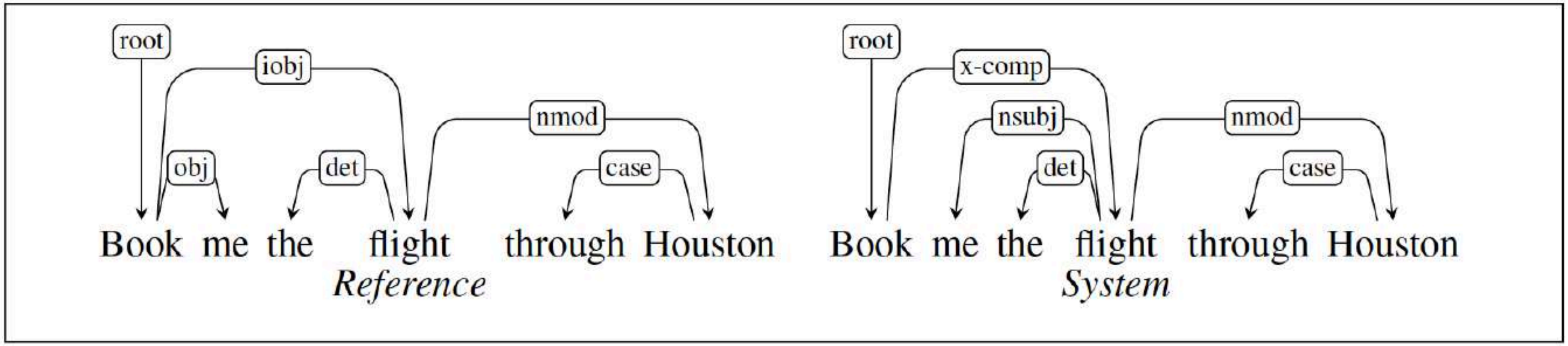


Figure 13.15 Reference and system parses for *Book me the flight through Houston*, resulting in an LAS of 4/6 and an UAS of 5/6.

Bibliography

(1) Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft, version Jan 2023); Online: https://web.stanford.edu/~jurafsky/slp3/ (URL, Oct 2023); this slide-set is especially based on chapter 18.

Recommendations for Studying

- minimal approach:

  work with the slides and understand their contents! Think beyond instead of merely memorizing the contents

- standard approach:

  minimal approach + read the corresponding pages in Jurafsky [1]

- interested students

  == standard approach