Dependency Parsing

Dr. Demetrios Glinos
University of Central Florida

CAP6640 – Computer Understanding of Natural Language

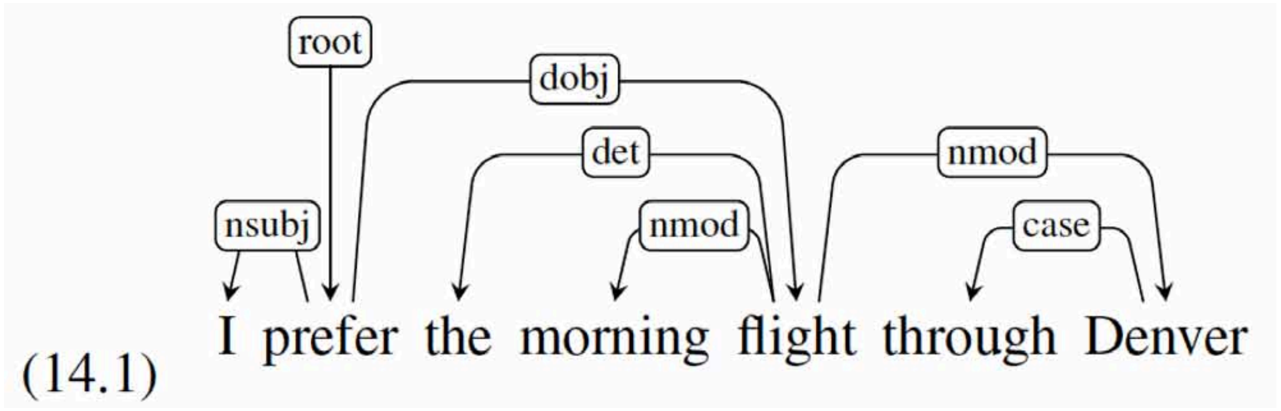
Today

- Dependency Parsing
- Dependency Relations
- Dependency Formalisms
- Dependency Treebanks
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluating Dependency Parses

Dependency Parsing

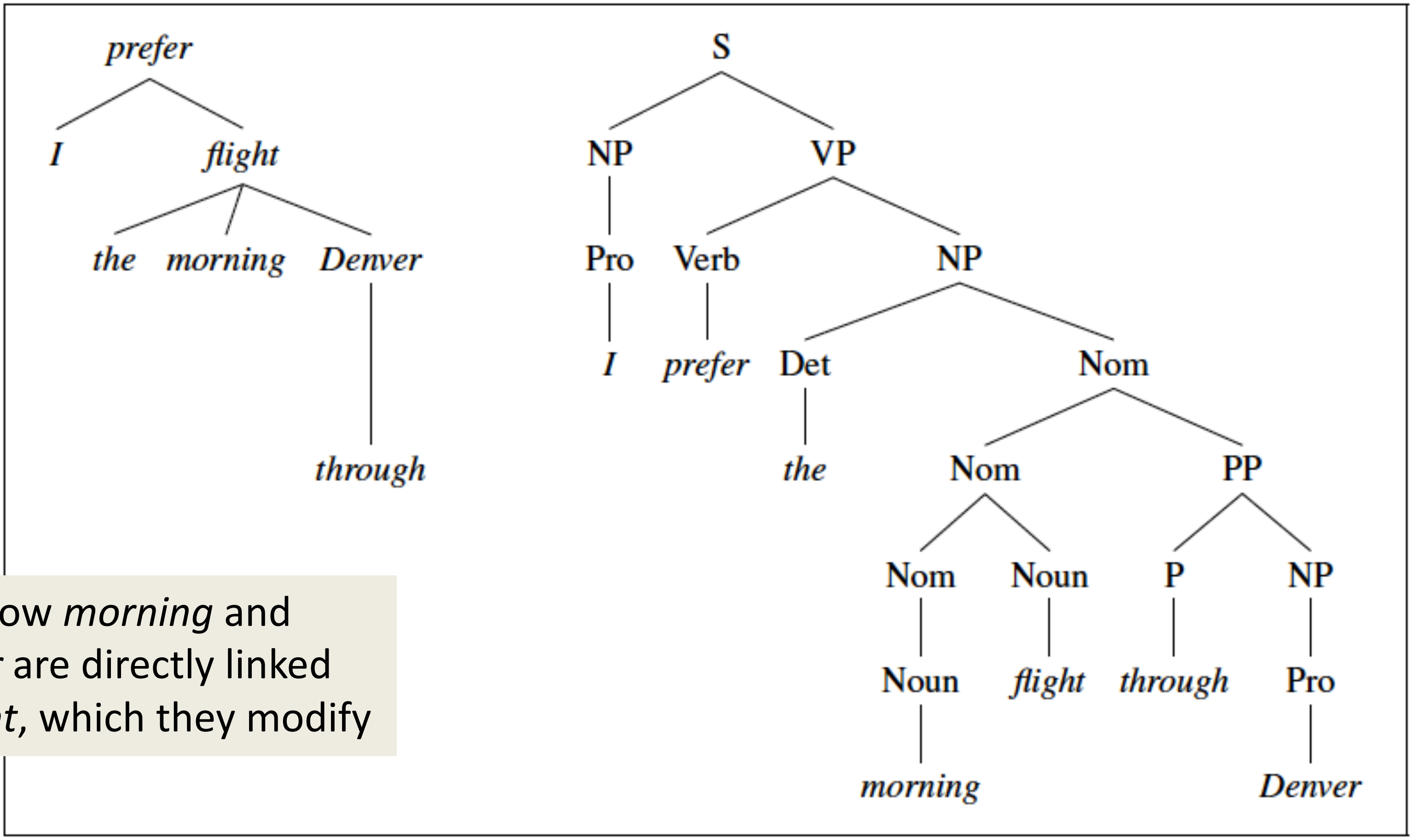
- Syntactic structure is described solely in terms of
  - the words (or lemmas), and
  - directed binary grammatical relations between words

- Root of parse structure is main verb
- Directed arcs go from head to dependent
- There are no non-terminals!



source: J&M (3d Ed. draft), Ch. 14

Comparison of Parse Trees



Note how *morning* and *Denver* are directly linked to *flight*, which they modify

Figure 14.1 A dependency-style parse alongside the corresponding constituent-based analysis for *I prefer the morning flight through Denver.*

source: J&M (3d Ed. draft) , Ch. 14

Advantages of Dependency Grammars

- Better able to handle morphologically rich languages, which have relatively free word order
    - e.g., in Czech, a grammatical object can appear either before or after a location adverbial
        - would need separate rules in a phrase-structure grammar
        - but dependency would have only 1 arc for this relation

- The head-dependent relations can serve as approximations to semantic relations between predicates and their arguments
    - useful for coreference resolution, question answering, information extraction
    - wold need head-finding rules to extract similar information from constituent parse trees

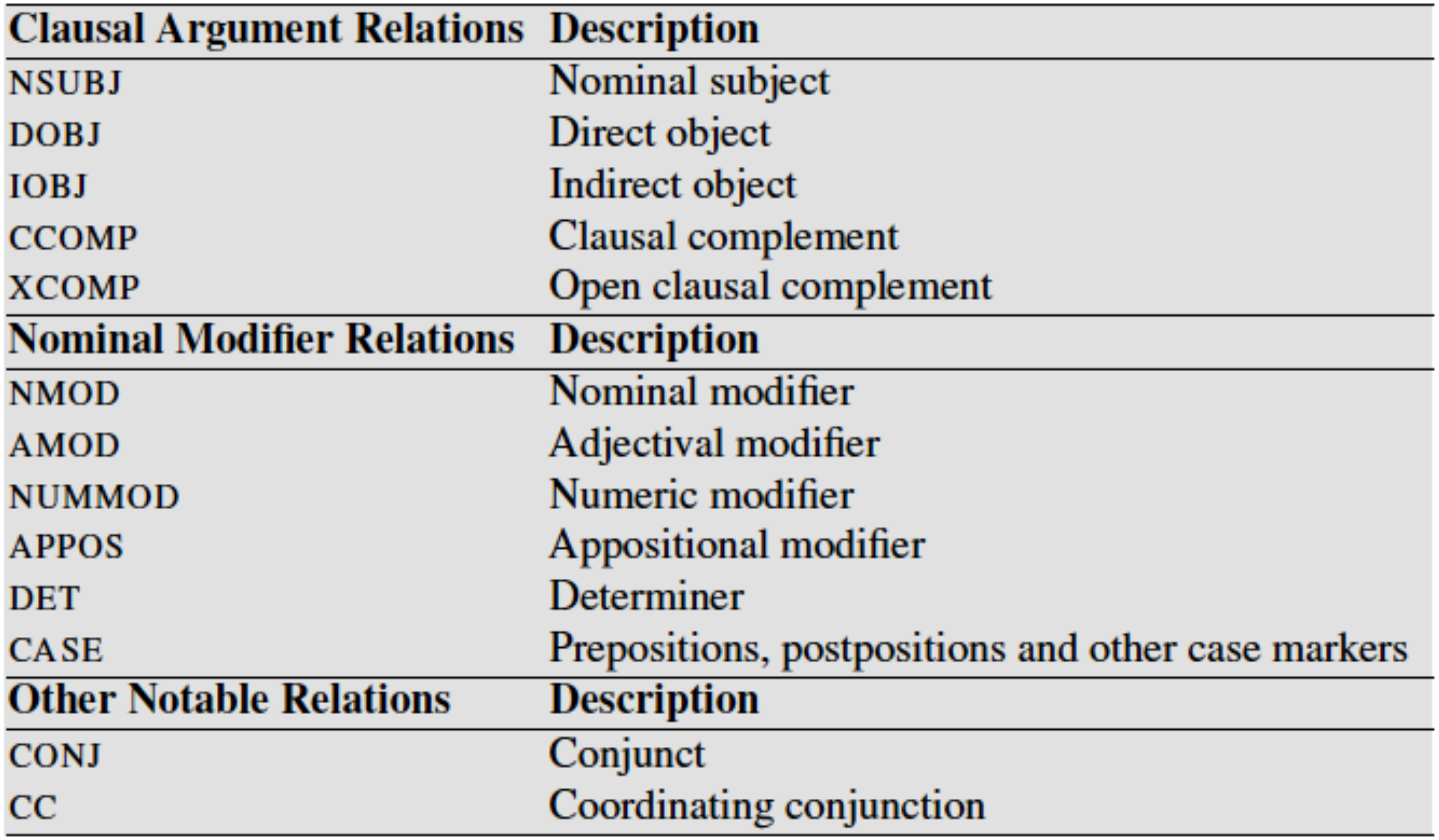
Today

- Dependency Parsing
- Dependency Relations
- Dependency Formalisms
- Dependency Treebanks
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluating Dependency Parses

Dependency Relations

- Consider the constituent: *on the morning flight*
  - head – central organizing word of a larger constituent (*on*)
  - remaining words in constituent are either direct or indirect dependents of the head
- Dependency relations
  - arcs connect head to *directly-related* dependents
  - labels on arc encode the *grammatical functions* expressed by the relation
    - e.g., subject, direct object, and indirect object

Dependency Relations

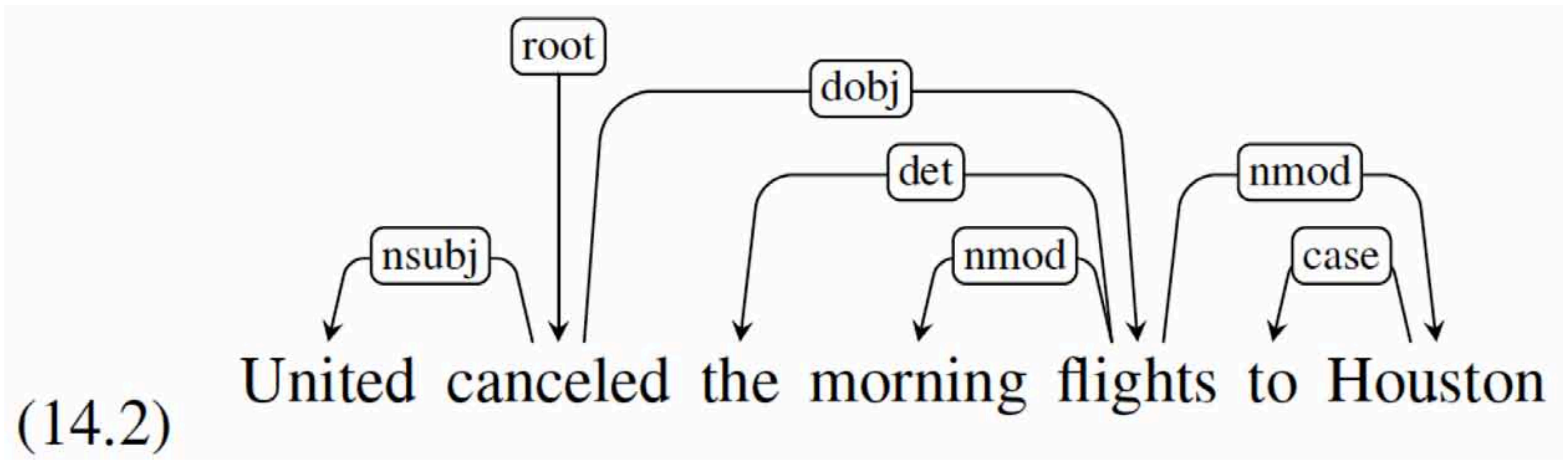


Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET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

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

source: J&M (3d Ed. draft) , Ch. 14

Example: Dependency Relations



source: J&M (3d Ed. draft) , Ch. 14

- nsubj – subject of predicate *cancel*
- dobj – direct object of predicate *cancel*
- nmod & det – modifiers of noun *flights*
- case – modifier of noun *Houston*

Today

- Dependency Parsing
- Dependency Relations
- Dependency Formalisms
- Dependency Treebanks
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluating Dependency Parses

Dependency Formalisms

Dependency tree G = (V, A)

where V = set of vertices (words/lemmas, punctuation, etc.)

A = set of dependency relation arcs

and which satisfies these constraints

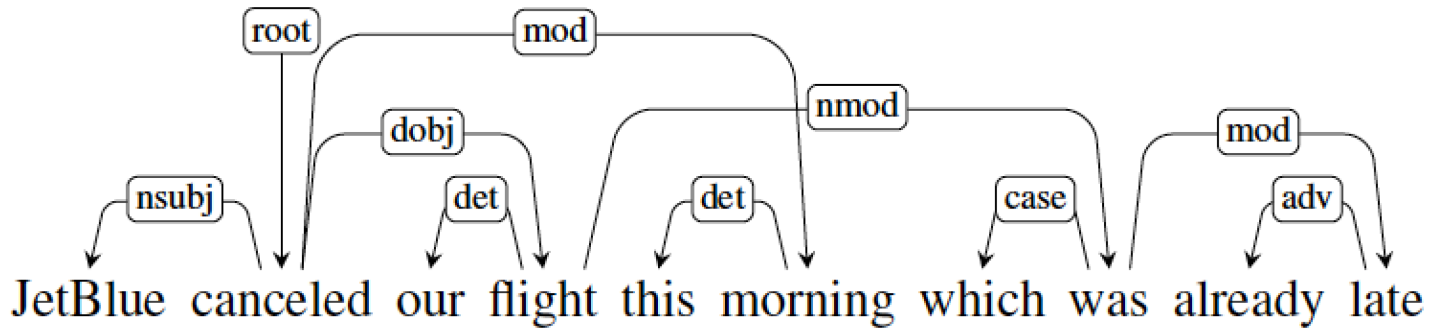
1. There is a single designated root node that has no incoming arcs
2. Each vertex (other than root) has exactly one incoming arc
3. There is a unique path from the root node to each vertex in V

→ Thus, each word has a single head and the graph is connected

Projectivity

- An arc is projective if
  - there is a path from the head to every word between the head and the dependent
- A dependency tree is projective if all arcs in it are projective
- In this example, the arc from *flight* to *was* is non-projective

(14.3)


root
mod
nmod
dobj
mod
nsubj
det
det
case
adv
JetBlue canceled our flight this morning which was already late

source: J&M (3d Ed. draft) , Ch. 14

- A dependency tree is projective if it can be drawn with no arcs that cross

Today

- Dependency Parsing
- Dependency Relations
- Dependency Formalisms
- Dependency Treebanks
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluating Dependency Parses

Dependency Treebanks

- Corpus construction approaches
  - directly annotated by humans or semi-supervised with humans in loop
  - mine existing constituent-based treebanks using head rules
- Directly annotated corpora
  - have been created for morphologically rich languages: Czech, Hindi, Finnish
  - e.g., Prague Dependency Treebank
- Mined resources
  - e.g., Wall Street Journal section from Penn Treebank
  - OntoNotes project also includes additional media for English, Chinese, Arabic

Mining Treebanks for Dependencies

- Identifying all the head-dependent relations
  - generally use "head rules" (as for lexicalized probabilistic parsers)
  - Head rule approach:
    1. use head rules to mark head child of each node in phrase structure
    2. make the head of each non-head child depend on the head of the head-child
- Identifying the correct dependency-grammar relations for the detected relations
  - here, can make use of additional information such as function tags that are often present in the treebank
- Result
  - a set of projective dependencies, due to the manner of construction
  - generally flat structure for most noun phrases

Today

- Dependency Parsing
- Dependency Relations
- Dependency Formalisms
- Dependency Treebanks
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluating Dependency Parses

Shift-Reduce Parsing

- Originally developed for programming languages
- Classic approach
  - CFG
  - a stack
  - a list of tokens to be parsed
- Basic operation
  - input tokens are successively shifted onto the stack
  - top 2 elements of stack are matched against the RHS of the rules in the grammar
  - when match found, the matched elements are replaced on the (now reduced) stack by the non-terminal from the LHS of the rule

Transition-Based Dependency Parsing

- Adapts shift-reduce parsing by
  - dispensing with the CFG
  - altering the reduce operation to introduce a dependency relation between a word and its head (instead of adding a non-terminal to the tree)
  - reduce actions can be:
    - assert a head-dependent relation between the word at the top of the stack and the word below it
    - or vice versa

Basic Transition-Based Parser

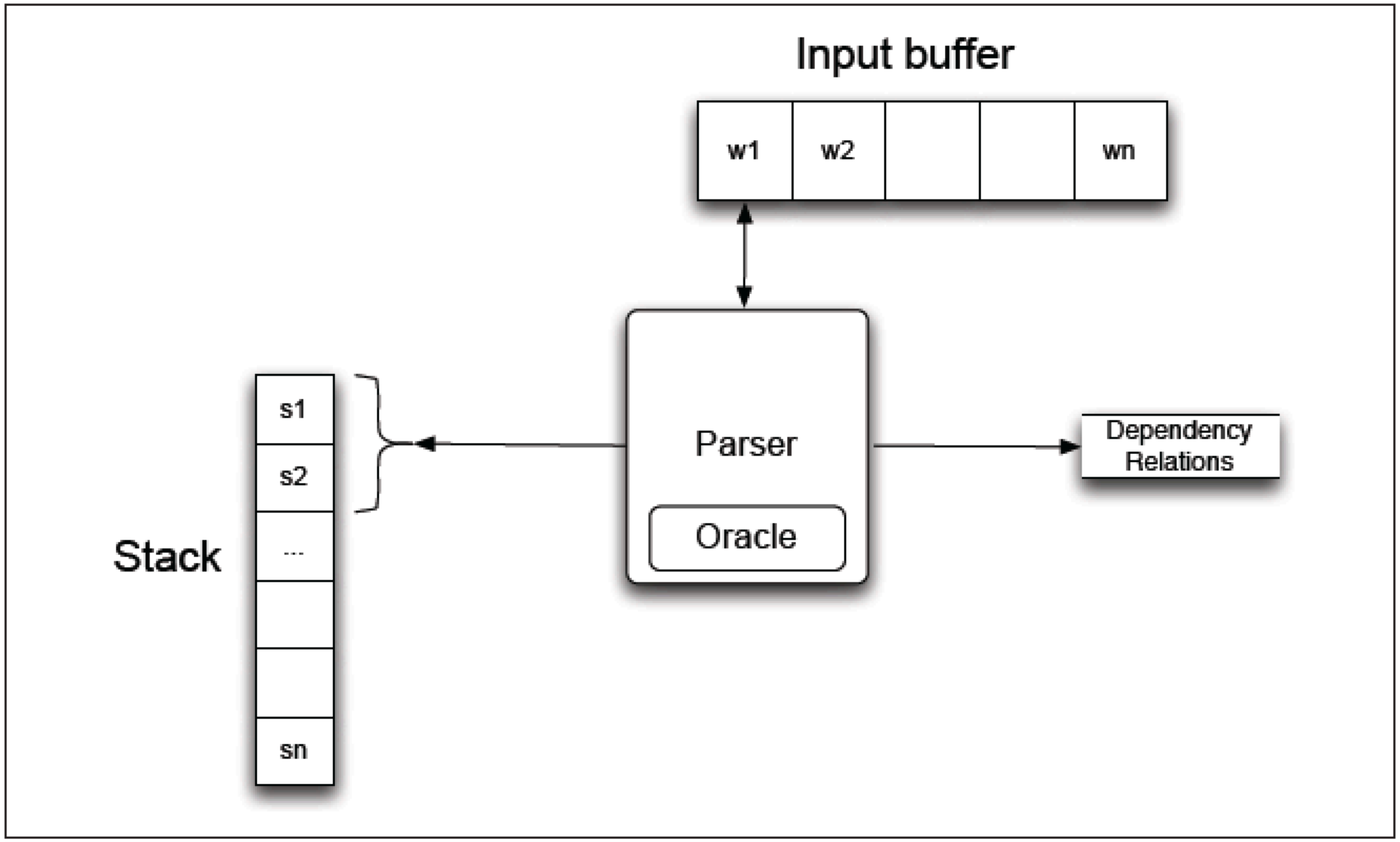


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

source: J&M (3d Ed. draft) , Ch. 14

Dependency Parsing as Search

- Configuration
  - represents the state in a state-space search
  - expressed as 3-tuple c = (Σ, B, A) where
    - Σ is a sublist of the input nodes V_x of some sentence x
    - B represents the remaining nodes of V_x
    - A is a set of dependency arcs over V_x
- Start state:
  - Σ contains root node; B contains the entire sentence; A is the empty set
- Goal:
  - a configuration where all the words have been accounted for and an appropriate dependency tree has been generated
- Solution:
  - the sequence of transitions from the start state to a goal state

Transition Operators

- These operators produce new configurations from a given configuration

LEFT-ARC

- assert a head-dependent relation between the word at top of stack with word below it
- remove the lower word from the stack

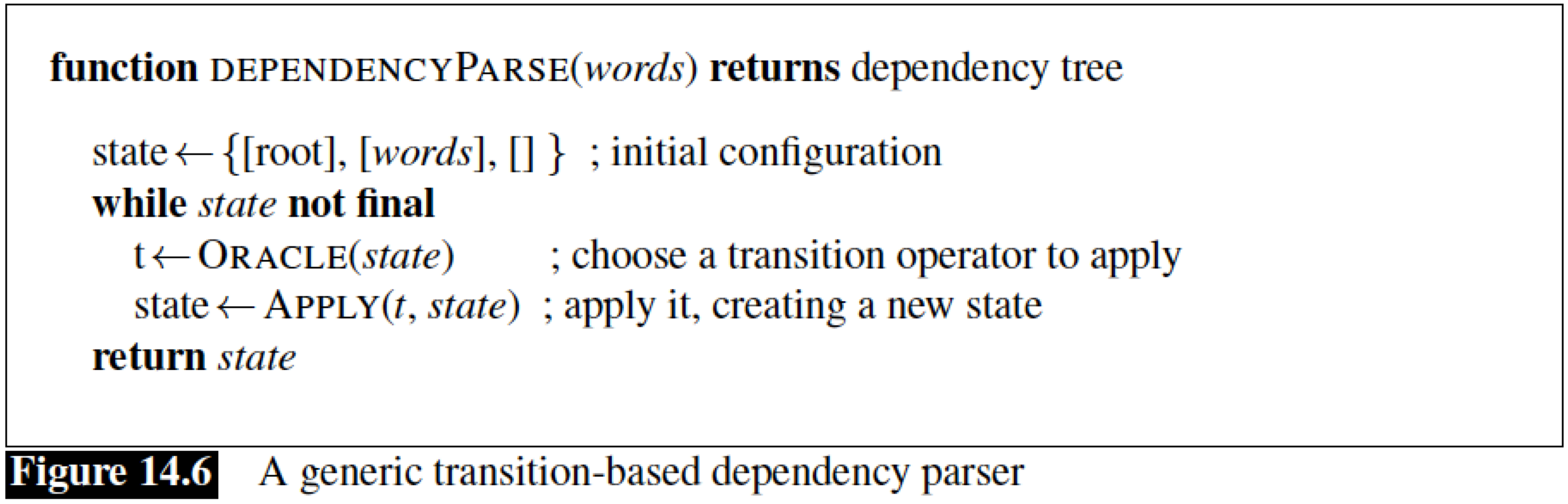
RIGHT-ARC

- assert a head-dependent relation between the second word on the stack with word at the top
- remove the lower word at the top of the stack

SHIFT

- remove the word from the front of the input buffer and push it onto the stack

Dependency Parsing Algorithm



```
function DEPENDENCYPARSE(words) returns dependency tree

  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
```

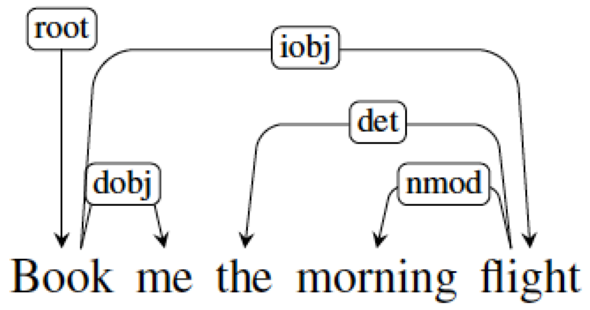
Figure 14.6 A generic transition-based dependency parser

source: J&M (3d Ed. draft) , Ch. 14

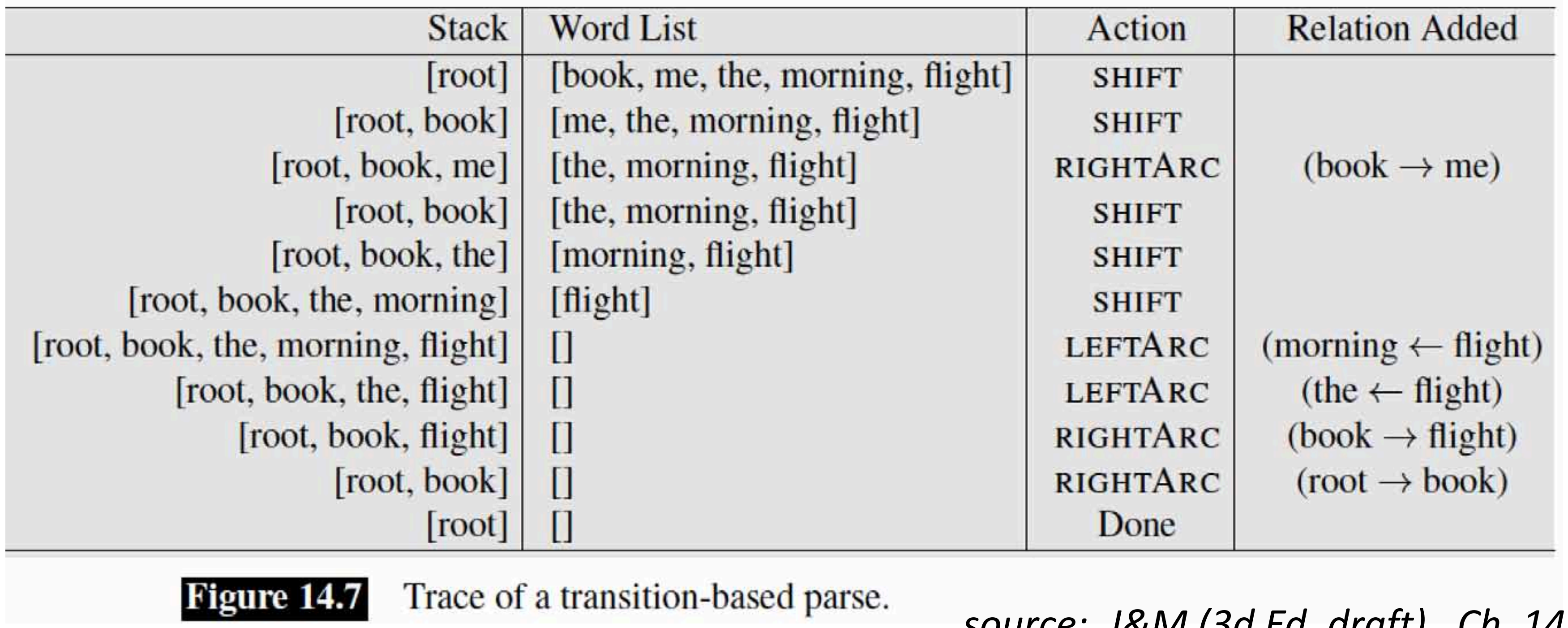
Notes

- an Oracle is used to determine the correct transition operator to apply
- the process ends when all the words in the sentence have been consumed and the root node is the only element remaining on the stack
- complexity is O(n), where n = # words in sentence, since based on a straightforward greedy approach that makes a single left-to-right pass through the words of the sentence

Example: Transition-Based Dependency Parse



(14.5)



Stack	Word List	Action	Relation Added
[root]	[book, me, the, morning, flight]	SHIFT	
[root, book]	[me, the, morning, flight]	SHIFT	
[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
[root, book]	[the, morning, flight]	SHIFT	
[root, book, the]	[morning, flight]	SHIFT	
[root, book, the, morning]	[flight]	SHIFT	
[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
[root, book, the, flight]	[]	LEFTARC	(the ← flight)
[root, book, flight]	[]	RIGHTARC	(book → flight)
[root, book]	[]	RIGHTARC	(root → book)
[root]	[]	Done	

Figure 14.7 Trace of a transition-based parse.

source: J&M (3d Ed. draft) , Ch. 14

Creating an Oracle

- Use supervised machine learning to train a classifier that serves as the Oracle
- Training data source: a treebank that contains dependency trees
- What we need: configurations + transition operators (class labels)
- But treebanks don't contain this
- We can get this by running the parsing algorithm using the reference parses from the treebank
- This is called using a training oracle
  - with correct parse in hand, we can choose correct transition, and add it to the configuration to create the training data set

Features

- We need to extract feature vectors for training our classifier
- Generally, features vary by language, genre, and type of classifier
- Typical features
  - word forms
  - lemmas
  - POS
  - head
  - dependency relation to head
- Focus of feature extraction
  - top levels of stack
  - words near front of buffer
  - dependency relations already associated with the above
  - combinations of the above (e.g., for to 2 on stack) are also useful
- Features of interest extracted and put into feature template

Learning Approaches

- Features tend to be large in number, resulting in sparse templates
- Traditional learning approaches
  - multinomial logistic regression
  - support vector machines
- Recently
  - neural network and deep learning, which eliminate the need for complex, hand-crafted features

Alternative Transition Systems

- Transition system described so far is called arc-standard transition system
- Arc eager transition system
  - modifies transition operators to operate on word at top of stack and word at head of buffer
  - also introduces a REDUCE operator to pop the stack
  - effect is to allow words to attach to their heads as soon as possible, rather than awaiting for all their dependents to be processed (reduces errors)
- Beam search
  - used to expand basic greedy approach to consider a fixed number of alternative parses
  - requires implementing a scoring system for configurations based on summing the scores of the transition operators that produced them

Alternative Transition Systems

- Basic transition parsing algorithm is quite general
- Transition systems have also been developed for
    - POS-tagging
    - generating non-projective dependency structures
    - assigning semantic roles
    - parsing texts containing multiple languages

Today

- Dependency Parsing
- Dependency Relations
- Dependency Formalisms
- Dependency Treebanks
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluating Dependency Parses

Graph-Based Dependency Parsing

- Basic idea
  - search through space of possible trees for a tree that maximizes some score

- Solves problem with greedy transition approach where accuracy declines significantly as distance between head and dependent increases

- Maximum spanning tree (MST) approach
  - uses an efficient greedy algorithm
  - edge weights (scores) generated from a model generated from training data

1. For each vertex, select the incoming edge (from a possible head) with the highest edge score
2. rescore edge costs by subtracting highest score incoming edge value from all incoming edge values to each vertex
3. recursively remove cycles by collapsing them into single nodes until can remove an edge; then restore cycles but exclude the removed edge

Example: MST

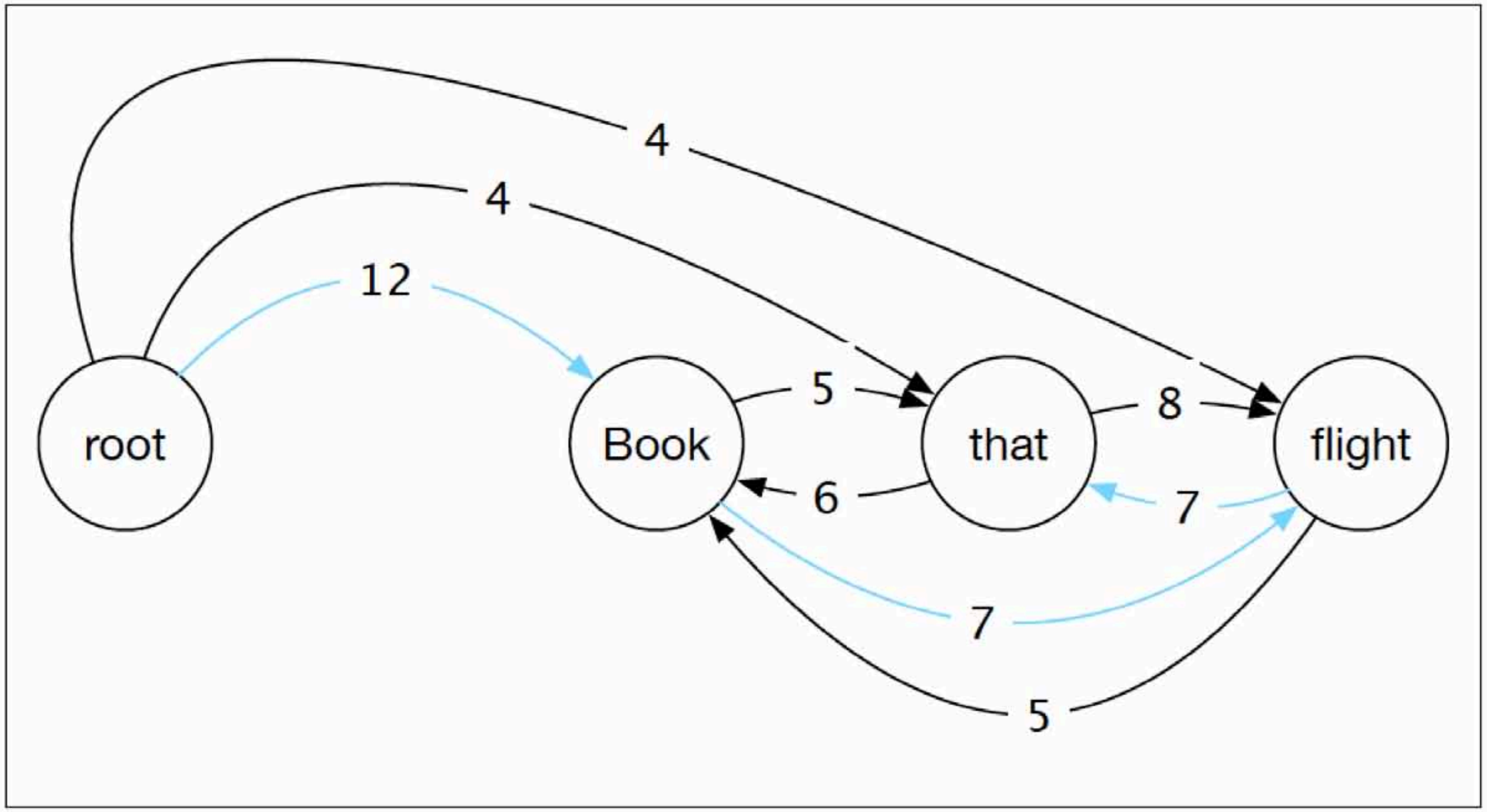
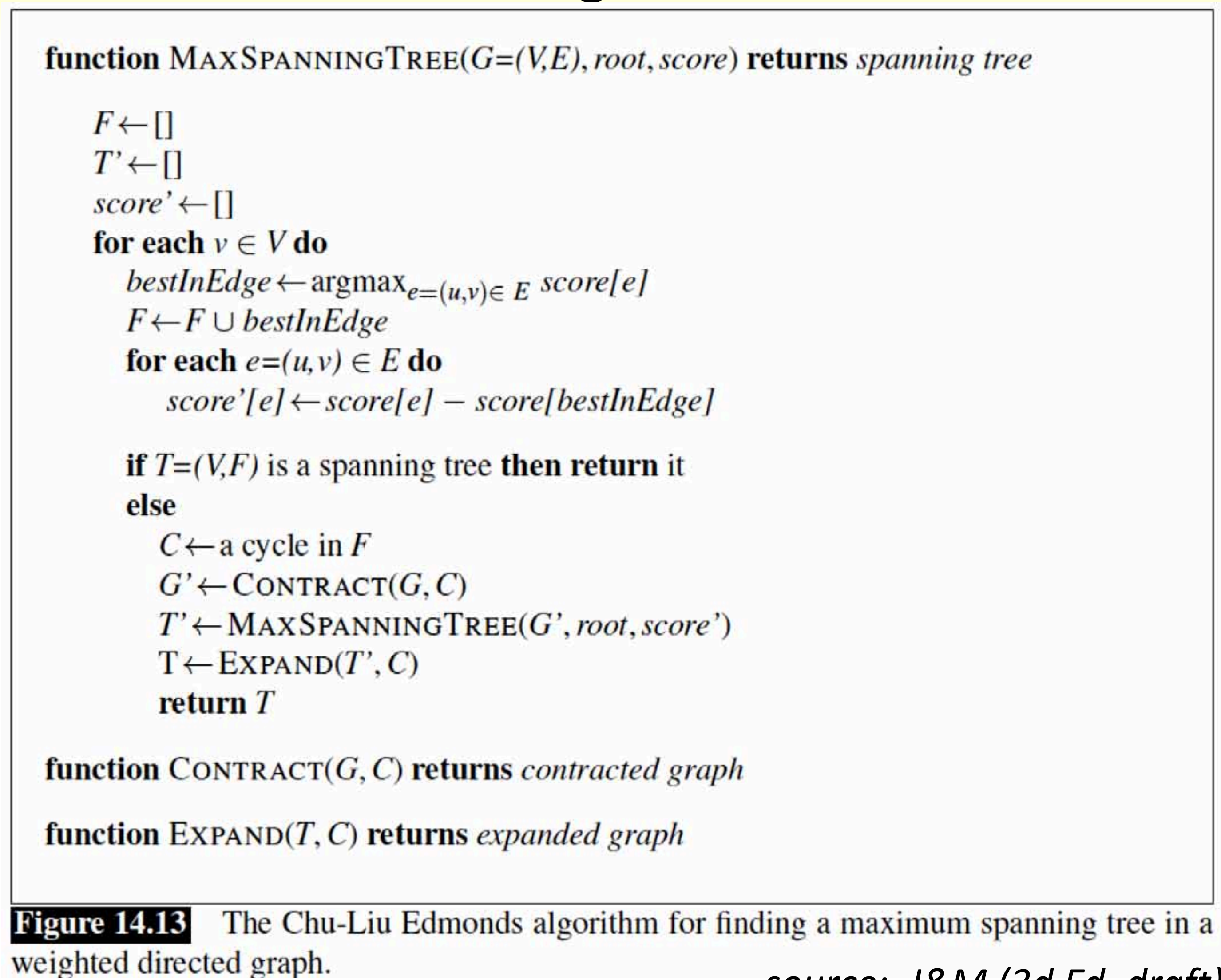


Figure 14.12 Initial rooted, directed graph for *Book that flight*.

source: J&M (3d Ed. draft) , Ch. 14

Note: MST is shown in blue

MST Algorithm



```
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
```

Figure 14.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

source: J&M (3d Ed. draft) , Ch. 14

Features

We illustrate edge-factored parsing, in which score for tree is sum of scores of its edges

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

where an edge score is a weighted sum of the features extracted from it

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

where the features mirror those used in training transition-based parsers:

- wordforms, lemmas, POS of the headword and its dependent
- features from the contexts before, after, and between the words
- the particular dependency relation between the words
- the direction of the relation (right or left)
- distance from head to dependent

Learning

- We seek to train a model that assigns higher scores to correct trees than incorrect trees
- This is different from training amodel that associates training instances with class labels
- Use inference-based learning plus the perceptron learning rule
  - parse sentence using random weights; if parse correct, do nothing
  - if parse incorrect, lower weights of features not in the correct parse
  - do this for each sentence in the training data set
  - repeat until convergence

Today

- Dependency Parsing
- Dependency Relations
- Dependency Formalisms
- Dependency Treebanks
- Transition-Based Dependency Parsing
- Graph-Based Dependency Parsing
- Evaluating Dependency Parses

Evaluating Dependency Parsing

- Exact-match (EM)
  - most sentences parsed as incorrect
  - too pessimistic to guide development

- Use labeled attachment accuracy and unlabeled attachment accuracy

  - labeled attachment – correct head + correct relation
  - unlabeled attachment – correct head only

  - labeled attachment score (LAS) - % words in input assigned correct head and relation
  - unlabeled attachment score (UAS) - % words in input assigned correct head

  - label accuracy score (LS) - % words in input assigned correct relation

Example: Parse Evaluation

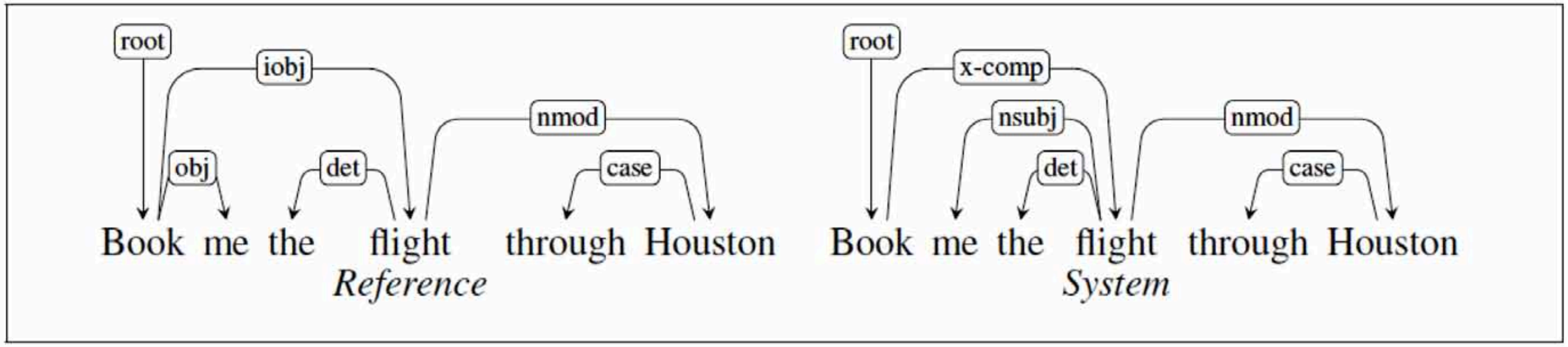

root
iobj
obj
det
nmod
case
Book me the flight through Houston
Reference
root
x-comp
nsubj
det
nmod
case
Book me the flight through Houston
System

Figure 14.15 Reference and system parses for *Book me the flight through Houston*, resulting in an LAS of 3/6 and an UAS of 4/6.

should be 5/6

should be 4/6

Note:
We can also evaluate performance on particular dependency relations (e.g., NSUBJ) using precision, recall, F_1, and confusion matrix, previously covered