Shift-Reduce Parsing for Phrase Structure Grammars

- A stack S and a queue Q are prepared
- Initially the input sentence is set in Q
- Two operations:
  - Shift: The first word in Q is moved into S
  - Reduce: If the sequence on the top of S matches with the right-hand side of a phrase grammar rule, reduce (delete) the sequence and push the symbol of the left-hand side of the rule into S.

(a) sentence → np, vp	(b) np → det, noun
(c) vp → verb, np	(d) det → “the”
(e) det → “a”	(f) noun → “boy”
(g) noun → “dog”	(h) verb → “hits”

Steps of Shift-Reduce Parsing

	stack S	queue Q	operation
(1)		the boy hits a dog	shift
(2)	the	boy hits a dog	reduce (d)
(3)	det	boy hits a dog	shift
(4)	det boy	hits a dog	reduce (f)
(5)	det noun	hits a dog	reduce (b)
(6)	np	hits a dog	shift
(7)	np hits	a dog	reduce (h)
(8)	np verb	a dog	shift
(9)	np verb a	dog	reduce (e)
(10)	np verb det	dog	shift
(11)	np verb det dog		reduce (g)
(12)	np verb det noun		reduce (b)
(13)	np verb np		reduce (c)
(14)	np vp		reduce (a)
(15)	sentence		

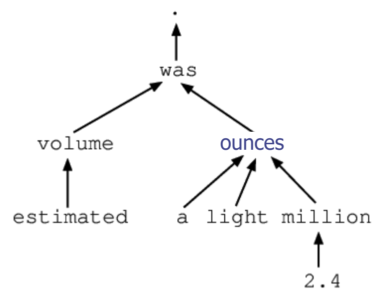
Contents of this lecture

- Morphological Analysis
  - Tokenization
  - Part-of-speech tagging
- Syntactic Parsing Algorithms
  - Phrase structure parsing
  - Word dependency parsing

A phrase structure tree and a dependency tree

```
                      S
                      |
       NP-------------VP--------.
        |              |        |
  VBN------NN   VBD-----------NP .
   |        |    |             |
Estimated volume was DT--JJ------QP------NNS
                      |   |      |        |
                      a light CD----CD ounces
                               |     |
                              2.4 million
```

(Penn Treebank)



Non-projective dependency tree
交差あり 依存構造木

Non-projective = Crossing dependencies are permitted

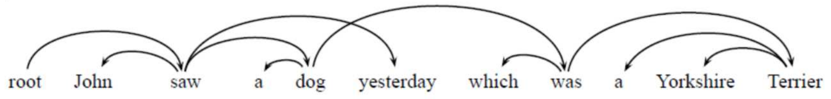


Figure 2: An example non-projective dependency structure.

*taken from: R. McDonald and F. Pereira, "Online Learning of Approximate Dependency Parsing Algorithms," European Chapter of Association for Computational Linguistics, 2006.

Dependency parsing algorithms

- Transition-based
  - (Kudo & Matsumoto (VLC 00, CoNLL 02): Japanese)
  - Yamada & Matsumoto (IWPT 03)
  - Nivre (IWPT 03, COLING 04, ACL 05)
  - Chen & Manning (EMNLP 14)
- Graph-based
  - Maximum Spanning Tree Algorithm
    - McDonald et al (ACL 05, EMNLP 05, EACL 06)
  - CKY-based
    - Eisner (COLING 96, Kluwer 00)
- Integer Linear Programming
  - Riedel & Clarke (EMNLP 06), Martins (ACL 09)

Transition-based Parsing: Deterministic linear time dependency parser based on Shift-Reduce parsing

[Nivre 03,04]

- There are two stacks S and Q
  - Initialization: S[w1] [w2, w3, …, wn]Q
  - Termination: S[…] []Q
  - Parsing actions:
    - **SHIFT**: S[…] [wi,…]Q → S[…, wi] […]Q
    - **Left-Arc**: S[…, wi] [wj, …]Q → S[…] [wj, …]Q
      ↑
      wi
    - **Right-Arc**: S[…, wi] [wj,…]Q → S[…, wi, wj] […]Q
    - **Reduce**: S[…, wi, wj] […]Q → S[…, wi] […]Q
      ↑
      wj

Though the original parser uses memory-based learning, recent implementation uses SVMs to select actions

→ MaltParser.ppt

Learning Actions

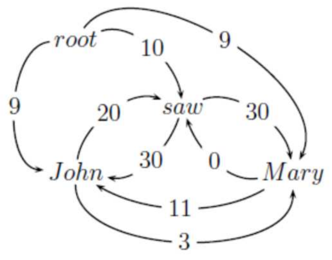
- The best action for each configuration is learned by a classifier (such as SVM)
- Recently, Neural Networks are used to predict the best action taking wider context into consideration.

Graph-based dependency parsing

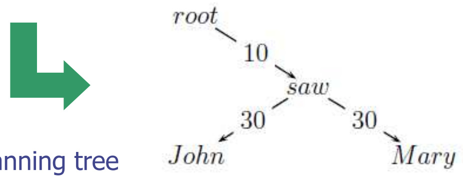
Maximum Spanning Tree algorithm [McDonald 05]

- R. McDonald, F. Pereira, K. Ribarov, and J. Hajiˇc, “Non-projective dependency parsing using spanning tree algorithms,”
  In *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, 2005.

Dependency parsing by maximum spanning tree



“John saw Mary”
all of dependency edges with scores
(root is a dummy node representing the head of sentence)



Spanning tree
with the highest score

Figures taken from [McDonald et al 2005]

Dependency Parsing by MST

1. Learn the weight vector **w** from the training data
   - In [McDonal 05] they used MIRA (a perceptron algorithm)
2. For a given sentence:
   - Calculate weights for all pairs of words and the pairs between root and all words
3. Search for the maximum spanning tree
   - Chu-Liu-Edmonds algorithm is used (Computation time is O(n^2) for a sentence of length n)

Notation and definition

$\mathrm{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$	training data
$\boldsymbol{x}_t$	t-th sentence in T
$\boldsymbol{y}_t = \{(i, j)\}_{i=1}^{n}$	Dependency structure for t-th sentence (defined by a set of edges)
$\mathbf{f}(i, j)$	The vector of feature functions for an edge
$\mathrm{dt}(\boldsymbol{x})$	The set of all dependency trees producible from x

Definition of target function

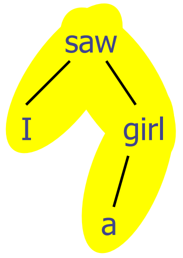
$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j)\in y} s(i, j) = \sum_{(i,j)\in y} \mathbf{w}\cdot\mathbf{f}(i, j)$$

$s(i, j)$	Score of i → j dependency
$\mathbf{f}(i, j)$	Feature set related with the dependency i → j, represented as a feature vector
$\mathbf{w}$	weight vector for the feature vector so as to maximize $\mathbf{w}\cdot\mathbf{f}(i, j)$.

The Basic Idea of MST parsing

The score function of dependency trees is defined by the sum of the scores of edges, which are defined by the weighted sum of feature functions

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j)\in y} s(i, j) = \sum_{(i,j)\in y} \mathbf{w}\cdot\mathbf{f}(i, j)$$



The target optimization problem:

$$\min \|\mathbf{w}\|$$
$$\text{s.t.}\quad s(\boldsymbol{x}, \boldsymbol{y}) - s(\boldsymbol{x}, \boldsymbol{y}') \geq L(\boldsymbol{y}, \boldsymbol{y}')$$
$$\forall(\boldsymbol{x}, \boldsymbol{y}) \in T,\quad \boldsymbol{y}' \in \mathrm{dt}(\boldsymbol{x})$$

L(y,y') is defined as the number of words in y' that have different parent compared with y.

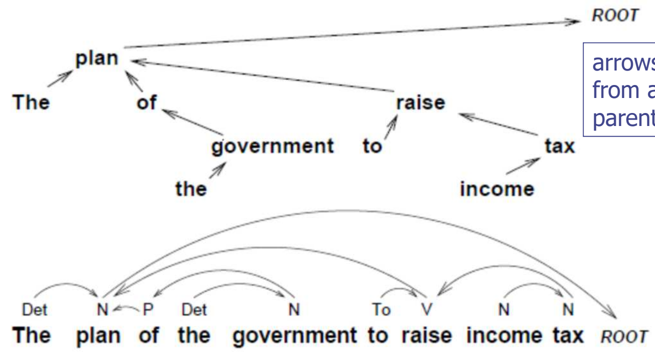
CKY-style dependency parsing

- Simple application of CKY parsing algorithm to dependency parsing
  - requires O(n^5) time (cf. O(n^3) for CFG)
- Why is it so time-consuming?
- It is possible to concurrent dependency parsing in O(n^3) time
  - ⇒ Eisner parsing algorithm [Eisner 00]

Jason Eisner, "Bilexical Grammars and Their Cubic-Time Parsing Algorithms," In Harry Bunt and Anton Nijholt (eds.), *Advances in Probabilistic and Other Parsing Technologies*, Kluwer Academic, Chapter 3, pp. 29-62. 2000.

CKY-style Parsing of Dependency Grammar

- Binary lexicalized grammar rules
  - $w \rightarrow w\ v$ or $w \rightarrow v\ w$
    (w is the head (parent) of v (dependant)

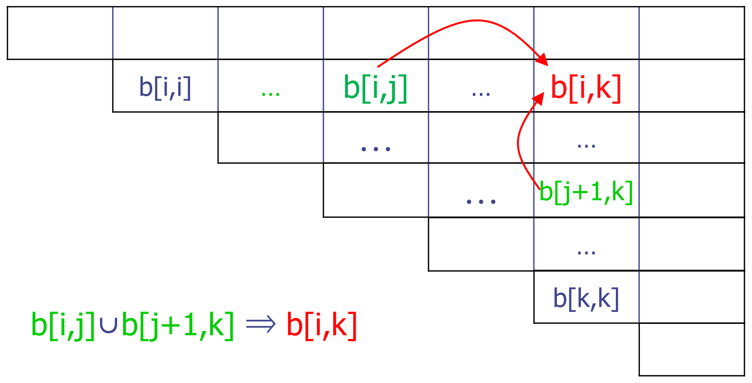


arrows are drawn from a child to the parent

Efficiency of binary lexicalized grammar

- CKY-style dependency parsing requires $O(n^5)$ time
  - CKY box b[i,j] contains subtrees spanning $w_{i,j}$
  - any word in $w_{i,j}$ can be the head in b[i,j]
    - when constructing items in b[i,k] from b[i,j] and b[j+1,k], there are min(n,|V|) possible words in b[i,j] and b[j+1,k] that can be the head (V: the vocabulary size)
    - making the time complexity $O(n^3) \times O(\min(n,|V|)^2) = O(n^5)$ (since $n<|V|$)

CKY Table

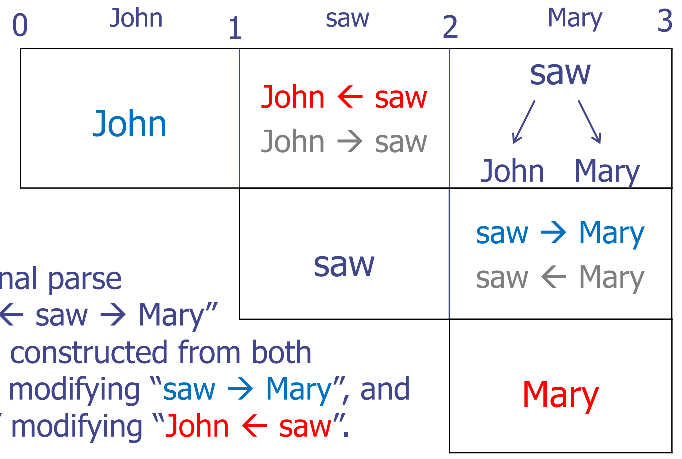


$b[i,j] \cup b[j+1,k] \Rightarrow b[i,k]$

Another Problem: Spurious ambiguities

- Simple application of CKY algorithm to Binary Lexical grammar rules causes another problem: spurious ambiguities
- When a word has modifiers from both sides (left and right), left and right modifiers can be intermingled. (Processing left modification first and processing right modification first produce the same results.)

Simple CKY parsing of binary lexicalized grammar: spurious ambiguity



The final parse
"John ← saw → Mary"
can be constructed from both
"John" modifying "saw → Mary", and
"Mary" modifying "John ← saw".

How to resolve inefficiency

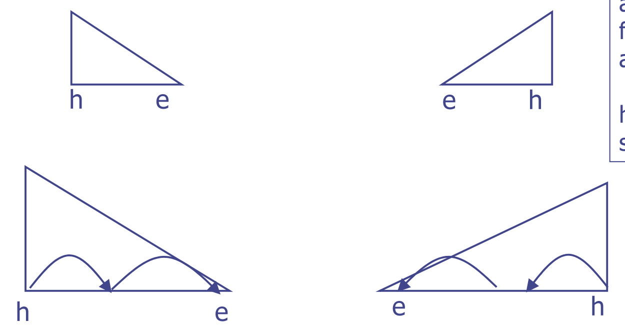
- For resolving Spurious ambiguity
  - It is possible to constrain CKY parsing process, so that left and right modifiers are not intermingled. E.g., left modifiers are delayed until all the right modifiers are considered
- For resolving O(n) factor in boxes
  - Span-based extension to CKY algorithm (Eisner algorithm)
  - Every span has either the left-most or right-most node as the head.

Eisner Algorithm (CKY style dependency parsing algorithm)

- Consists of two types of spans
  - Complete span: no modifiers comes from the right / from the left
  - Incomplete span: modifiers may come from the right / from the left
- Consists of two types of combination rules
  - Combine an incomplete span and a complete span, producing a complete span
  - Combine two complete span, producing an incomplete span

Complete spans

- Complete span: no modifiers comes from the right / from the left



arrows are drawn from a parent to a child

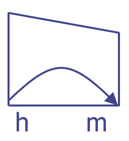
h: head of the span (subtree)

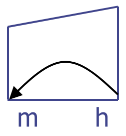
Incomplete spans

- Incomplete span: modifiers may come from the right / from the left

arrows are drawn from a parent to a child

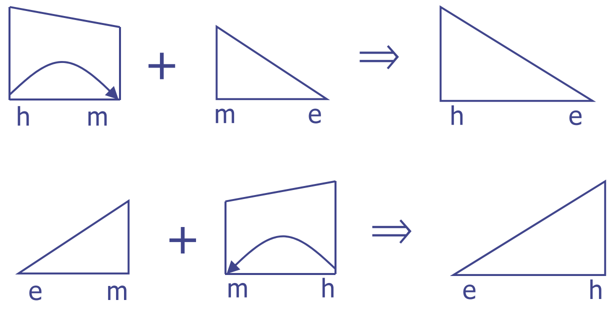
h: head of the span (subtree)





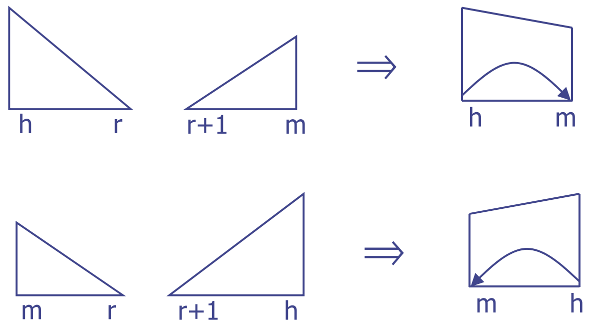
Combination rule (1)

Combine an incomplete span and a complete span, producing a complete span



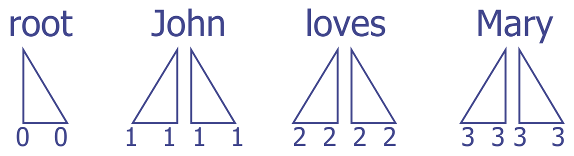
Combination rule (2)

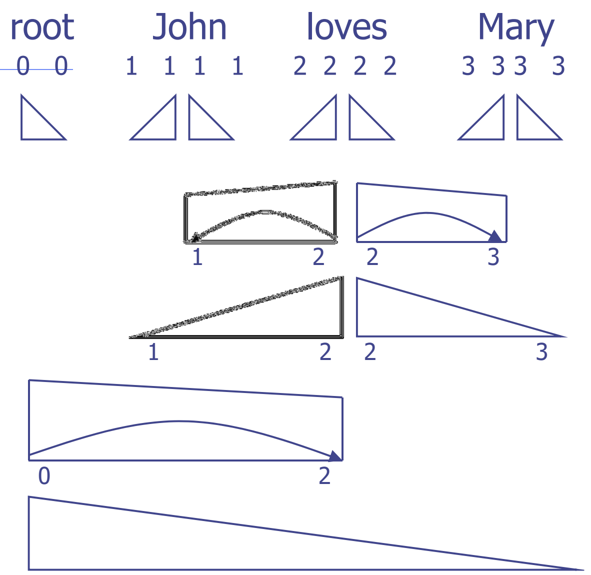
Combine two complete span, producing an incomplete span



Initial Configuration

- For each word in a given sentence w_i, build a left and right complete span
- Add a root with only right span with label 0 (or equivalently left span with label n+1 if we put root at the end of the sentence)





Eisner Algorithm: summary

- Given an input sentence
- Construct the initial configuration
  - complete spans (right and left spans) for each word, together with a root span
- CKY style parsing using two combination rules
  - This procedure constructs all possible dependency structures in $O(n^3)$ time

Dependency Parsing with Global Constraints

- Use of Integer Linear Programming (ILP)
  - The constraints for an input sentence to be a dependency tree with the highest score are described by equations with integer variables
  - Then ILP is used to get the best result (maximum spanning tree)
  - First work done by [Riedel and Clarke 2006]

Riedel, S. and Clarke, J., "Incremental Integer Linear Programming for Non-projective Dependency Parsing," Conference on Empirical Methods in Natural Language Processing, pp.129-137, 2006.

Formulation in ILP

$d_{i,j}$ = 1 if i is the parent of j
= 0 otherwise
defined for all word pairs except for root, which has only outgoing edge
(root: 0, other nodes: 1…n,)

Two basic constraints:

For all nodes j :

$$\sum_i d_{i,j} = 1$$ The parent of j is unique

$$\sum_i d_{i,0} = 0$$ root doesn't have a parent

Other Constraints and Object Function

- When the solution that maximizes the object function includes cycles, add the following constraint and repeat finding the solution
  - For edges belonging to the cycle c

$$\sum_{(i,j)\in c} d_{i,j} \le |c| - 1$$

- Object Function (to be maximized)
  - Same as the MST parser

$$\sum_{i,j} s(i,j)\cdot d_{i,j}$$

Other formalization of cycle-freeness

- In [Martins 2009], "cycle-free" is described by the following constraints

  "Connectivity of a tree"
  - New variables for edges $\phi_{i,j}$ representing the total number of nodes included in its descendants

$\sum_j \phi_{0,j} = n$	Root has n descendants
$\sum_i \phi_{i,j} = \sum_k \phi_{j,k} + 1$	Parent node (j)'s descendants are one plus all descendants of its children
$\phi_{i,j} \le n \cdot d_{i,j}$	No descendant for non-selected edge

A. Martins, N. Smith and E. Xing, "Concise Integer Linear Programming Formulations for Dependency Parsing," the Joint Conference of ACL and IJCNLP, 2009

Time Complexity of Dependency parsing algorithms

- Transition-based
  - Yamada & Matsumoto: O(n^2) (worst case)
  - Nivre (shift-reduce): O(n)
- Graph-based
  - Maximum Spanning Tree (first-order): O(n^2)
  - Second / third order MST = Second / third order Eisner algorithm
- Integer Linear Programming
  - No polynimial time algorithm is known in the worst case
  - Approximation by (Real) Linear Programming
- CKY parsing (Eisner algorithm)
  - First order, Second order (sibling): O(n^3)
  - Second order (grand parent): O(n^4)
  - Third order (grand parent + sibling, tri-sibling):O(n^4)

References

- Chen, D. and C. Manning, "A Fast and Accurate Dependency Parser using Neural Networks," EMNLP, pp.740-750, 2014.
- Eisner, J., "Bilexical grammars and their cubic-time parsing algorithms," In H. C. Bunt and A. Nijholt, editors, New Developments in Natural Language Parsing, pp.29–62, Kluwer Academic Publishers, 2000.
- A. Martins, N. Smith and E. Xing, "Concise Integer Linear Programming Formulations for Dependency Parsing," the Joint Conference of ACL and IJCNLP, pp.342-350, 2009
- R. McDonald, F. Pereira, K. Ribarov, and J. Hajic, "Non-projective dependency parsing using spanning tree algorithms," HLT/EMNLP, 2005.
- J. Nivre, "An efficient algorithm for projective dependency parsing," Proc. 8th International Workshop on Parsing Technologies (IWPT), 2003.
- S. Riedel and J. Clarke, "Incremental integer linear programming for non-projective dependency parsing," Proc. EMNLP, 2006.
- H. Yamada and Y. Matsumoto, "Statistical dependency analysis with support vector machines," Proc. 8th International Workshop on Parsing Technologies (IWPT), 2003.