UPPSALA
UNIVERSITET

Graph-based dependency grammar

Syntactic analysis (5LN455)

2023

Sara Stymne
Department of Linguistics and Philology

Partially based on slides from Marco Kuhlmann

UPPSALA UNIVERSITET

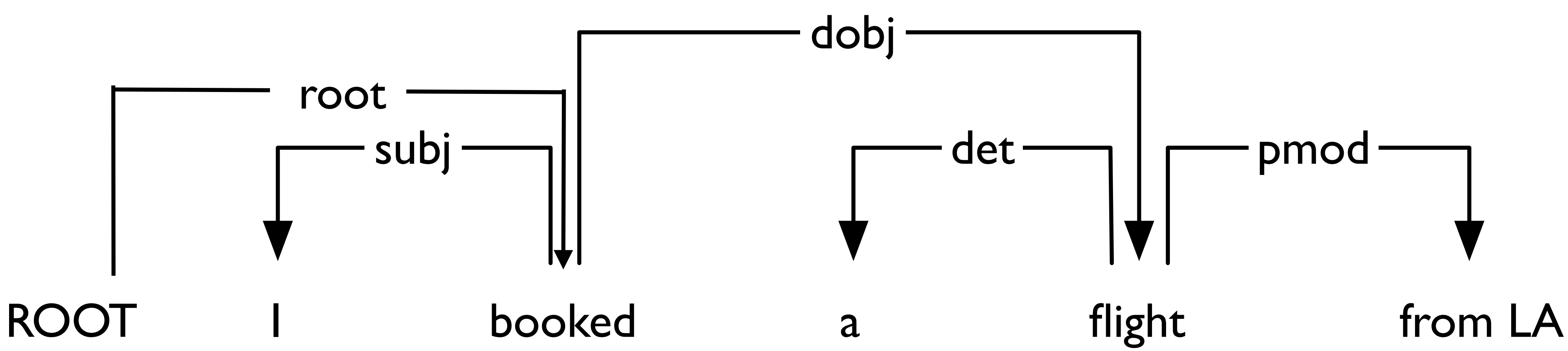
Overview

- Dependency grammar and projectivity
- Arc-factored dependency parsing
  - Collins' algorithm
  - Eisner's algorithm
- Evaluation of dependency parsers
- Transition-based dependency parsing
  - The arc-standard algorithm
- Advanced dependency parsing

UPPSALA
UNIVERSITET

Dependency grammar

Dependency trees



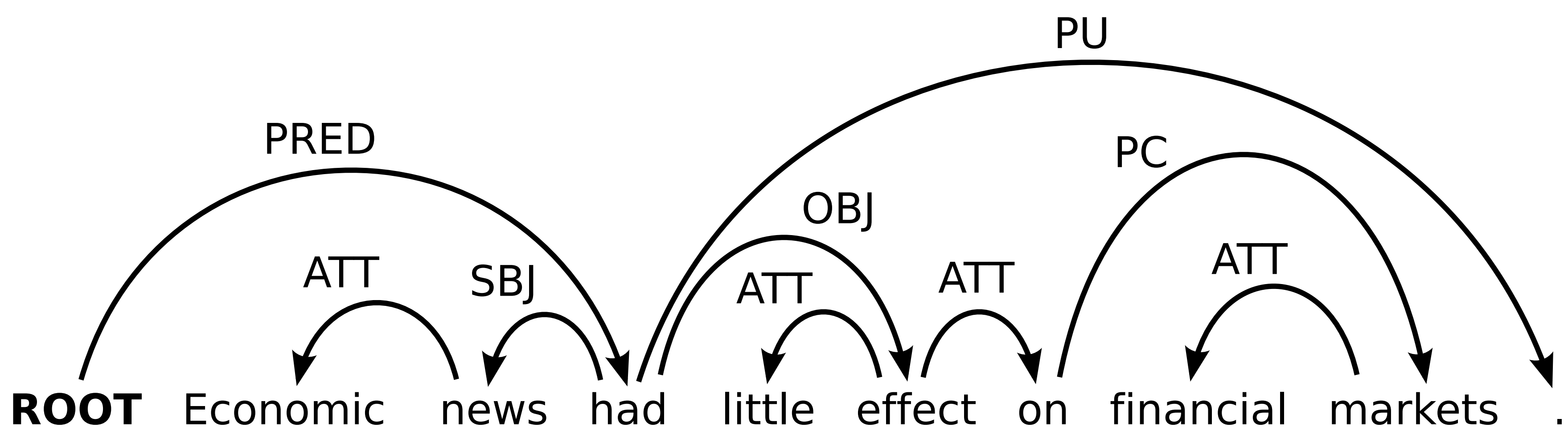
- In an arc $h \rightarrow d$, the word h is called the head, and the word d is called the dependent.
- The arcs form a rooted tree.
- Each arc has a label, l, and an arc can be described as (h, d, l)

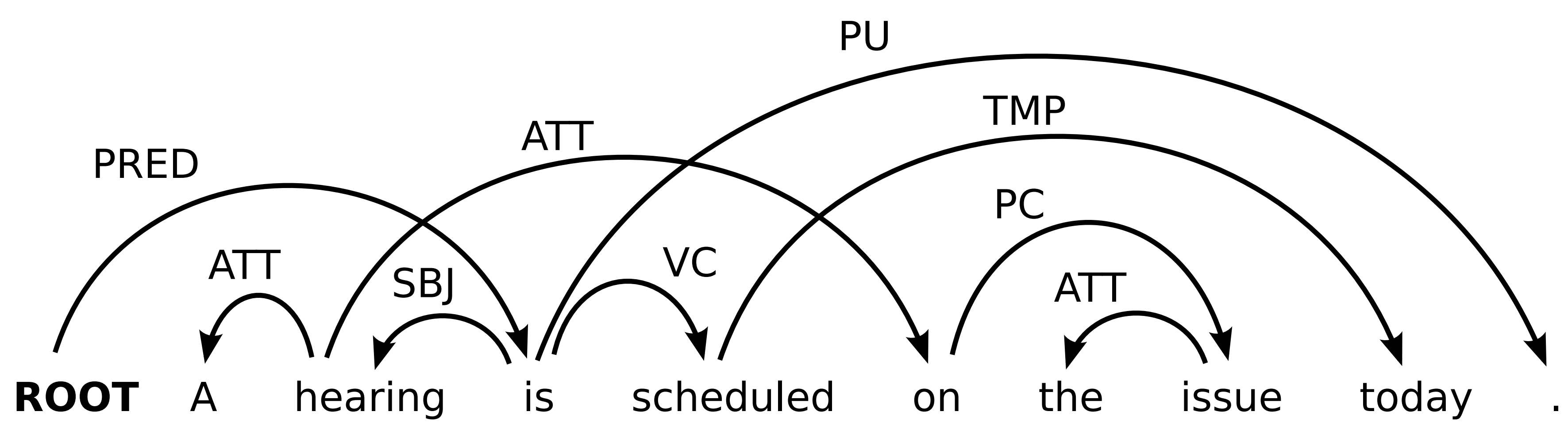
UPPSALA UNIVERSITET

Projectivity

- An important characteristic of dependency trees is projectivity
- A dependency tree is projective if:
  - For every arc in the tree, there is a directed path from the head of the arc to all words occurring between the head and the dependent (that is, the arc (i,l,j) implies that $i \rightarrow^* k$ for every k such that $\min(i, j) < k < \max(i, j)$)

Projective and non-projective trees





UPPSALA UNIVERSITET

Projectivity and dependency parsing

- Many dependency parsing algorithms can only handle projective trees
- Non-projective trees do occur in natural language
  - How often depends on the language (and treebank)
- In the course: in-depth discussion of projective algorithms, some discussion of non-projective algorithms

Main parsing strategies

- Graph-based dependency parsing:
    - Scores the dependency graph (tree)
- Transition-based dependency parsing:
    - Scores a sequence of transitions
- There are also grammar-based methods, which we will not discuss (not commonly used)

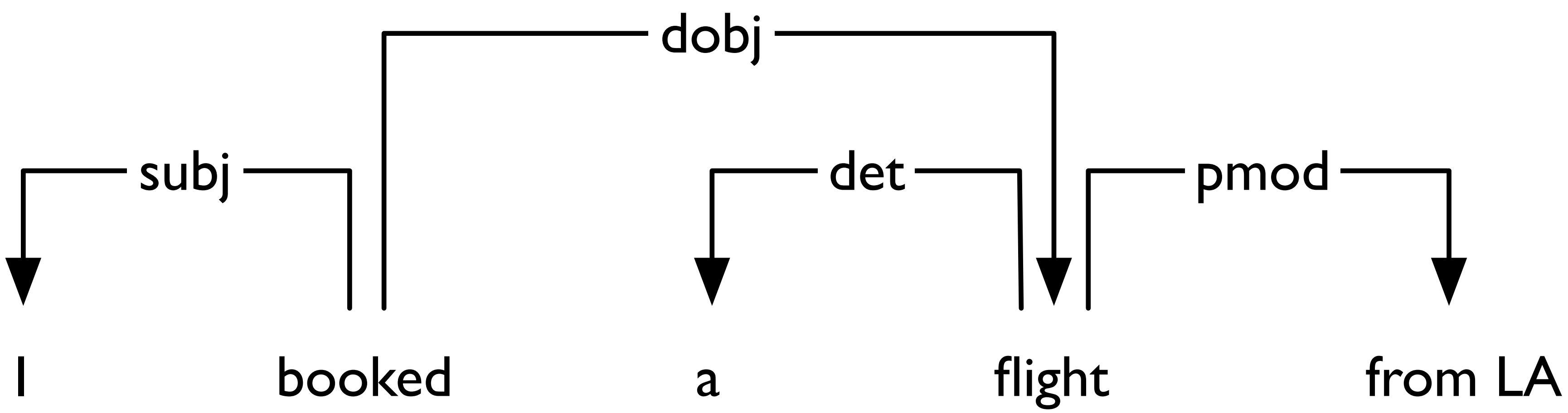
UPPSALA UNIVERSITET

Arc-factored dependency parsing

Uppsala Universitet

Ambiguity

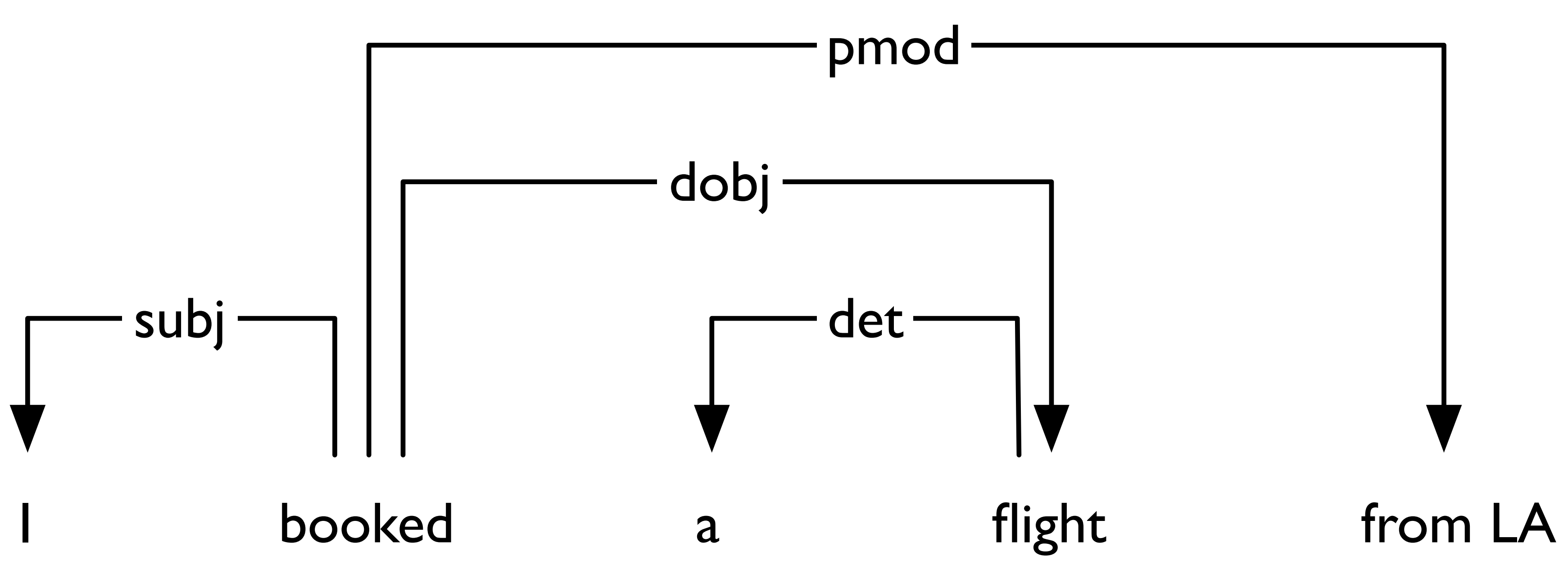
Just like phrase structure parsing, dependency parsing has to deal with ambiguity.



UPPSALA UNIVERSITET

Ambiguity

Just like phrase structure parsing, dependency parsing has to deal with ambiguity.



Disambiguation

- We need to disambiguate between alternative analyses.
- We develop mechanisms for scoring dependency trees, and disambiguate by choosing a dependency tree with the highest score.

Scoring models and parsing algorithms

Distinguish two aspects:

- Scoring model:
  How do we want to score dependency trees?
- Parsing algorithm:
  How do we compute a highest-scoring dependency tree under the given scoring model?

The arc-factored model

- Split the dependency tree t into parts $p_1, \ldots, p_n$, score each of the parts individually, and combine the score into a simple sum.

  score(t) = score(p_1) + … + score(p_n)

- The simplest scoring model is the arc-factored model, where the scored parts are the arcs of the tree.

Examples of classic features

- ‘The head is a verb.’
- ‘The dependent is a noun.’
- ‘The head is a verb
and the dependent is a noun.’
- ‘The head is a verb
and the predecessor of the head is a pronoun.’
- ‘The arc goes from left to right.’
- ‘The arc has length 2.’

Training using structured prediction

- Take a sentence *w* and a gold-standard dependency tree *g* for *w*.
- Compute the highest-scoring dependency tree under the current weights; call it *p*.
- Increase the weights of all features that are in *g* but not in *p*.
- Decrease the weights of all features that are in *p* but not in *g*.

Training using structured prediction

- Training involves repeatedly parsing (treebank) sentences and refining the weights.
- Hence, training presupposes an efficient parsing algorithm.

Higher order models

- The arc-factored model is a first-order model, because scored subgraphs consist of a single arc.
- An nth-order model scores subgraphs consisting of (at most) n arcs.
- Second-order: siblings, grand-parents
- Third-order: tri-siblings, grand-siblings
- Higher-order models capture more linguistic structure and give higher parsing accuracy, but are less efficient

Parsing algorithms

- Projective parsing
  - Inspired by the CKY algorithm
    - Collins' algorithm
    - Eisner's algorithm
- Non-projective parsing:
  - Minimum spanning tree (MST) algorithms
    - e.g. Chu-Liu-Edmunds algorithm (CLE)

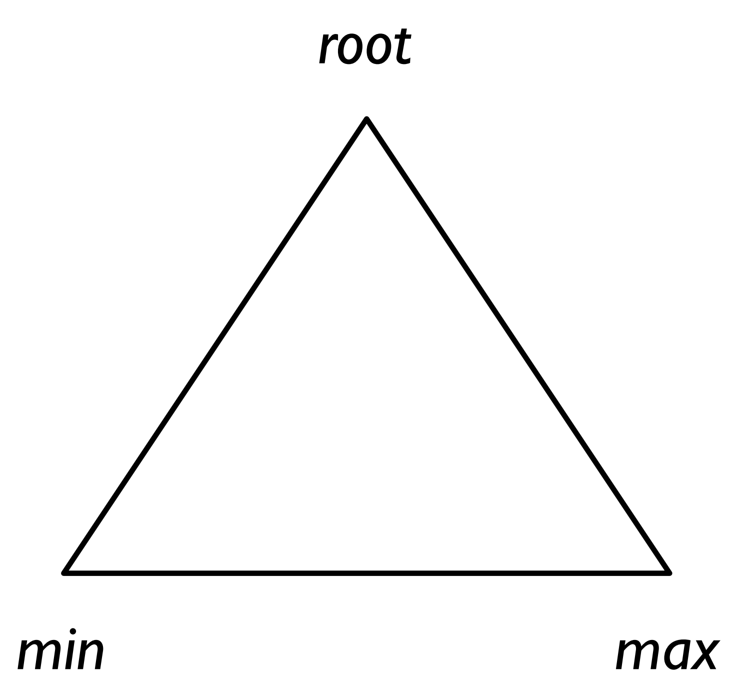
UPPSALA UNIVERSITET

Collins' algorithm

Collins' algorithm

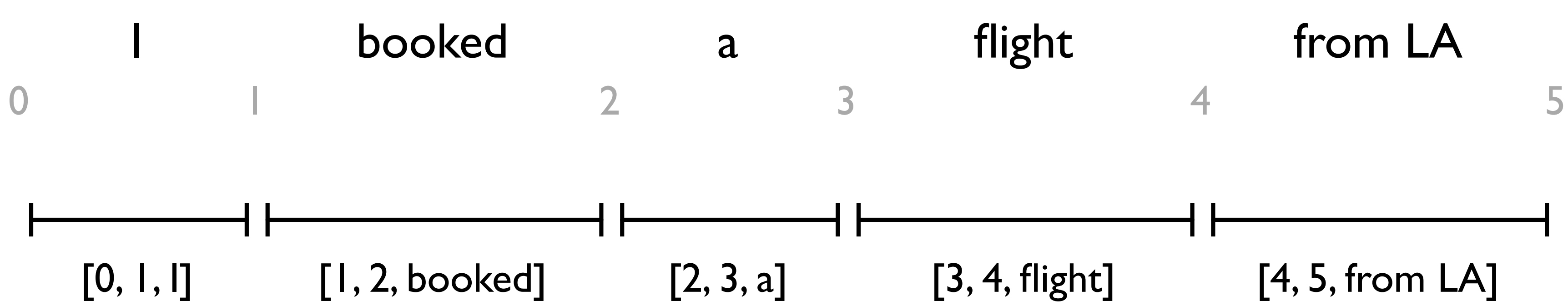
- Collin's algorithm is a simple algorithm for computing the highest-scoring dependency tree under an arc-factored scoring model.
- It can be understood as an extension of the CKY algorithm to dependency parsing.
- Like the CKY algorithm, it can be characterized as a bottom-up algorithm based on dynamic programming.

Signatures, Collins'

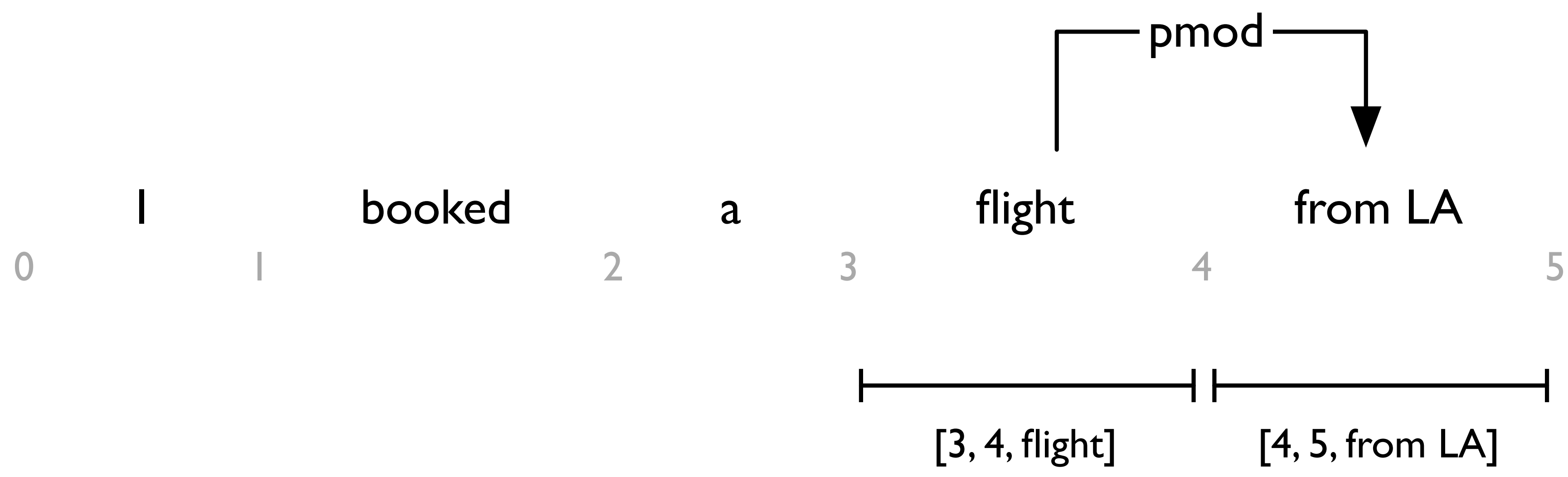


[*min*, *max*, *root*]

Initialization

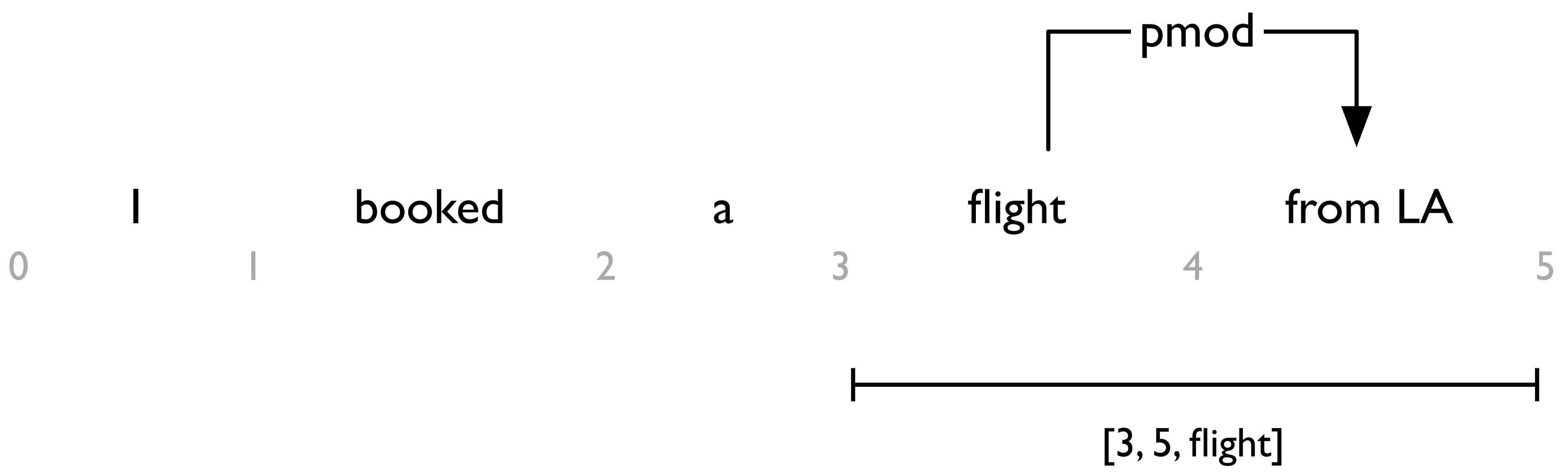


Adding a left-to-right arc

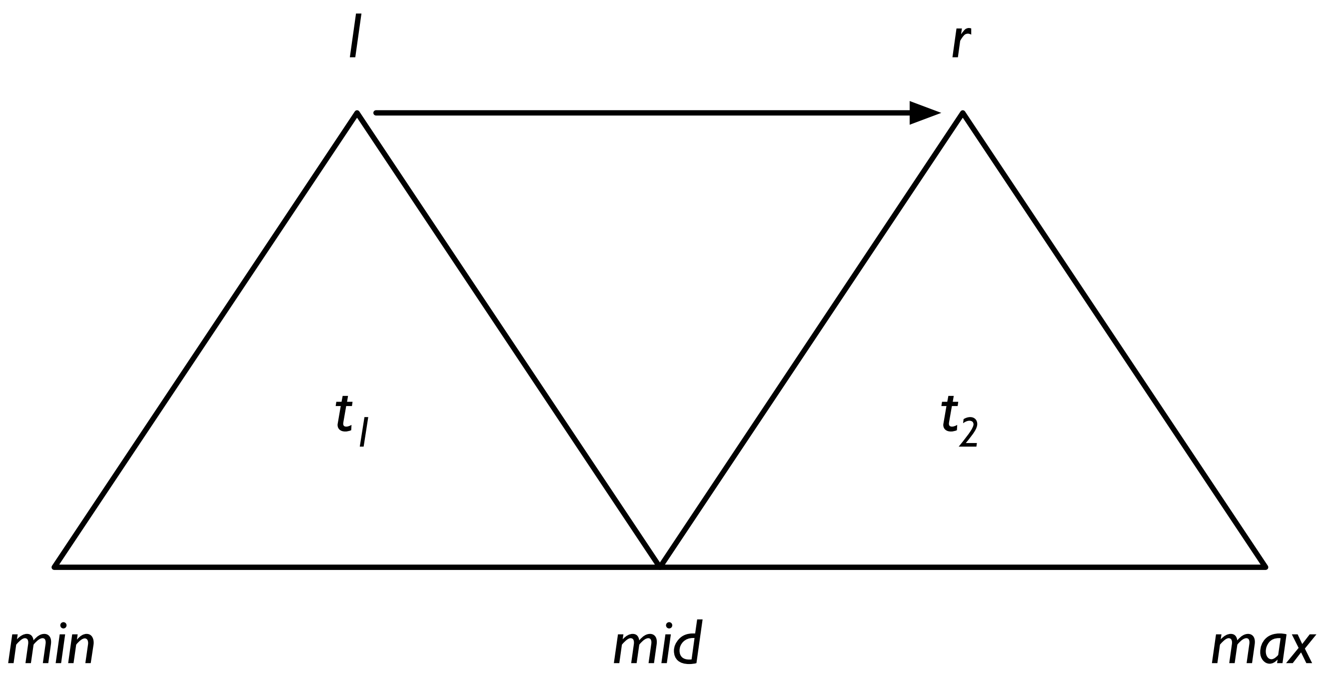


Collins' algorithm

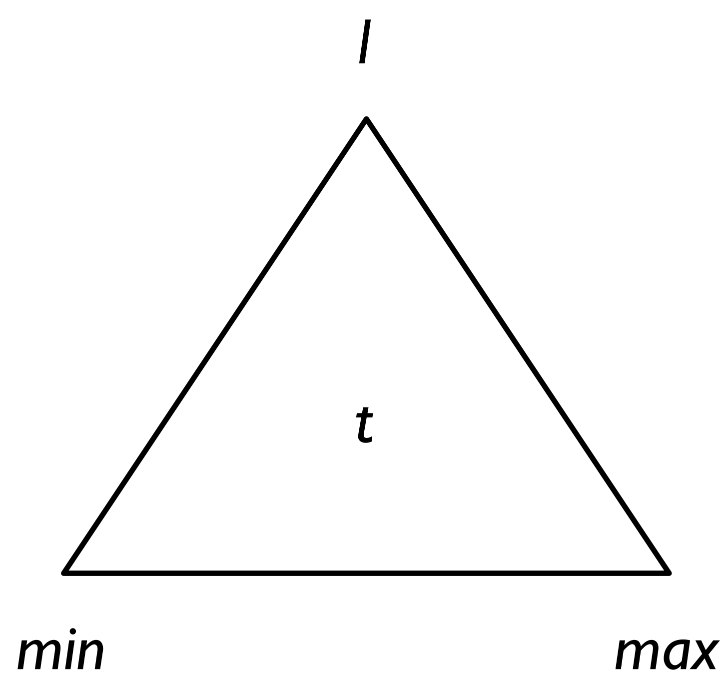
Adding a left-to-right arc



Adding a left-to-right arc



Adding a left-to-right arc



$$\text{score}(t) = \text{score}(t_1) + \text{score}(t_2) + \text{score}(l \rightarrow r)$$

Adding a left-to-right arc

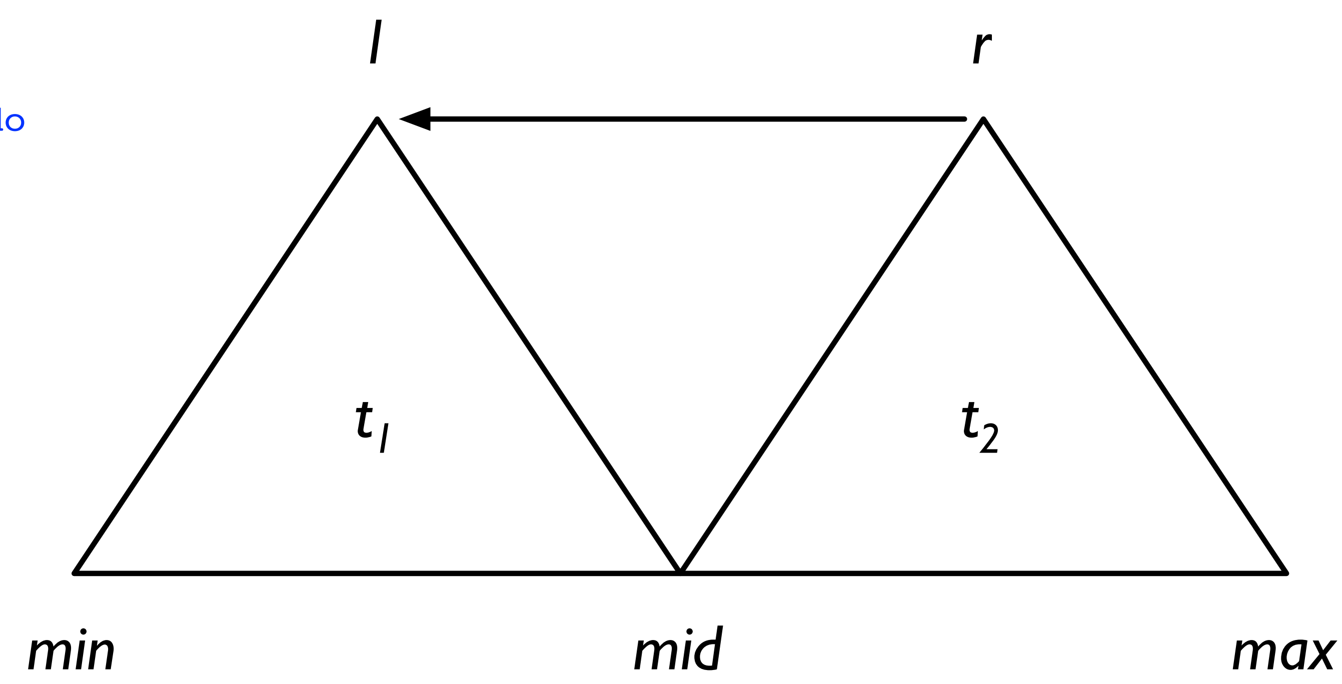
```
for each [min, max] with max - min > 1 do
  for each l from min to max - 2 do
    double best = score[min][max][l]
    for each r from l + 1 to max - 1 do
      for each mid from l + 1 to r do
        t1 = score[min][mid][l]
        t2 = score[mid][max][r]
        double current = t1 + t2 + score(l → r)
        if current > best then
          best = current
    score[min][max][l] = best
```

Collins' algorithm

Complexity analysis

- Runtime?
- Space?

```
for each [min, max] with max - min > 1 do
  for each r from min + 1 to max - 1 do
    double best = score[min][max][r]
    for each l from min to r - 1 do
      for each mid from l + 1 to r do
        t₁ = score[min][mid][l]
        t₂ = score[mid][max][r]
        double current = t₁ + t₂ + score(r → l)
        if current > best then
          best = current
    score[min][max][r] = best
```



UPPSALA UNIVERSITET

Complexity analysis

- Space requirement:
  $O(|w|^3)$
- Runtime requirement:
  $O(|w|^5)$

Extension to the labeled case

- It is important to distinguish dependencies of different types between the same two words.

  Example: subj, dobj

- For this reason, practical systems typically deal with labeled arcs.
- The question then arises how to extend Collins' algorithm to the labeled case.

Smart approach

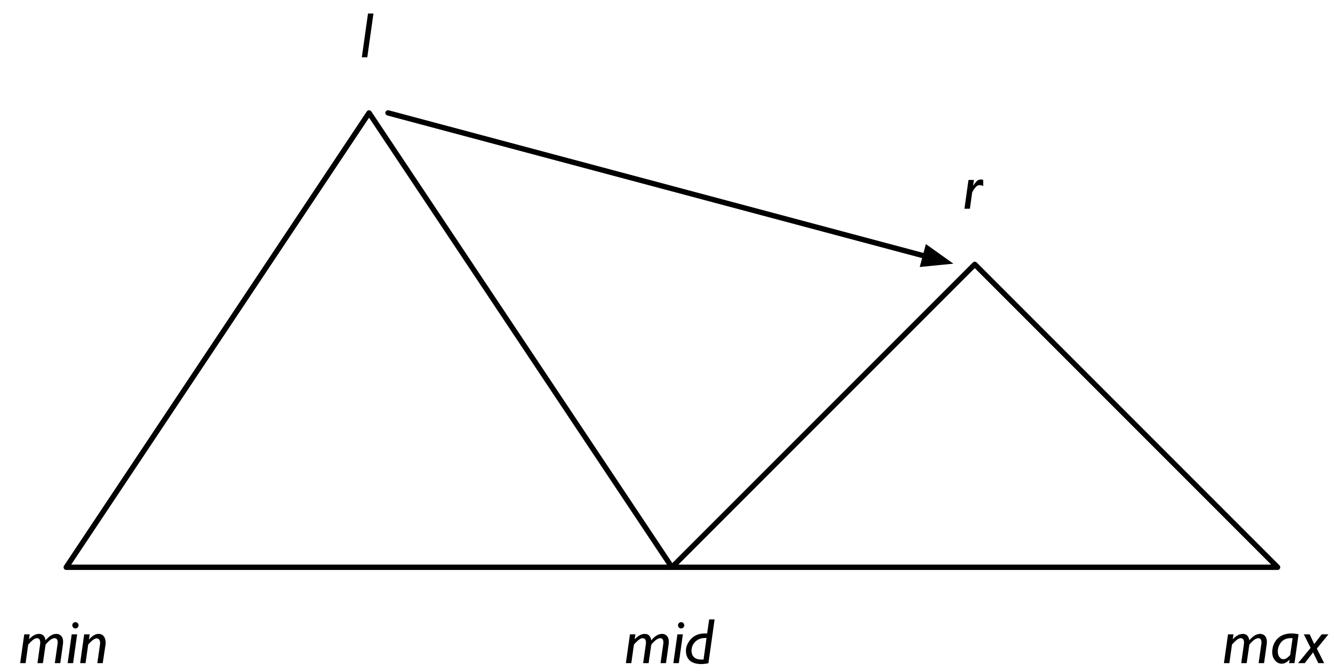
- Before parsing, compute a table that lists, for each head-dependent pair (h, d), the label that maximizes the score of arcs $h \rightarrow d$.
  - This is guaranteed to be the arcs that could be used in a highest-scoring tree
- During parsing, simply look up the best label in the pre-computed table.
- This adds (not multiplies!) a factor of $|L||w|^2$ to the overall runtime of the algorithm.

UPPSALA UNIVERSITET

Eisner's algorithm

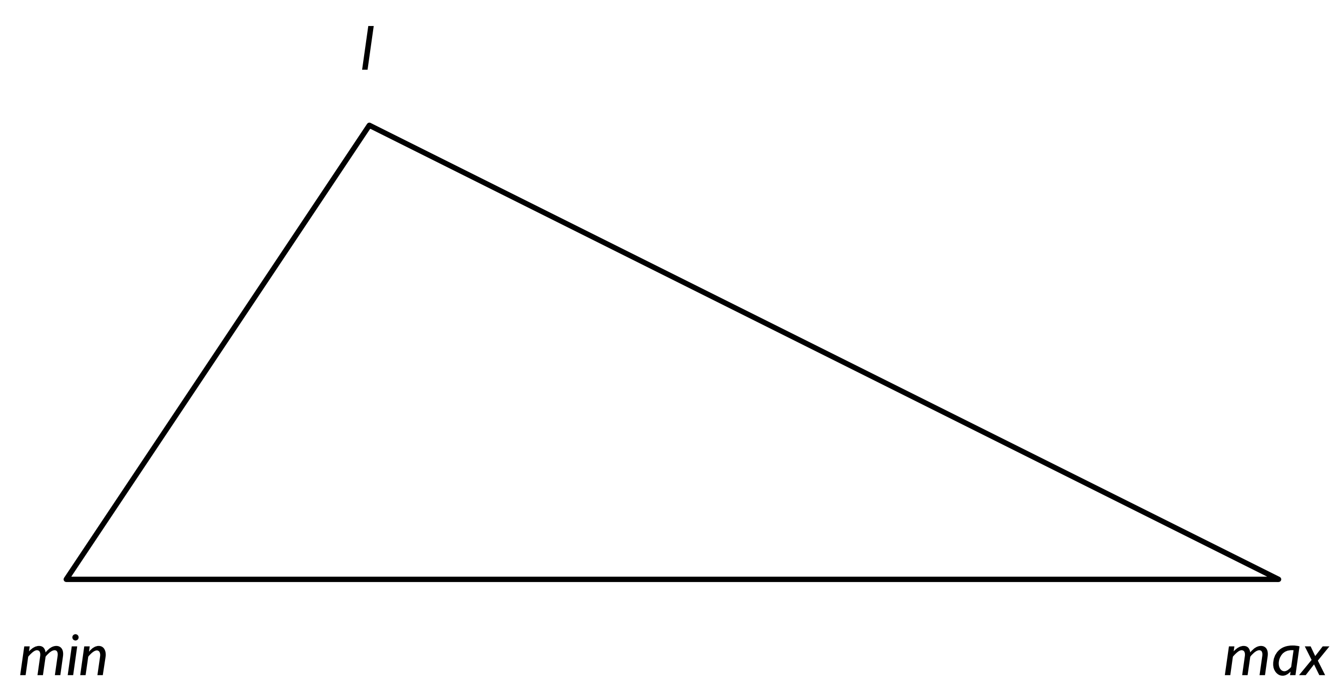
- With its runtime of $O(|w|^5)$, Collins' algorithm may not be of much use in practice.
- With Eisner's algorithm we will be able to solve the same problem in $O(|w|^3)$.
    - Intuition: collect left and right dependents independently

Basic idea



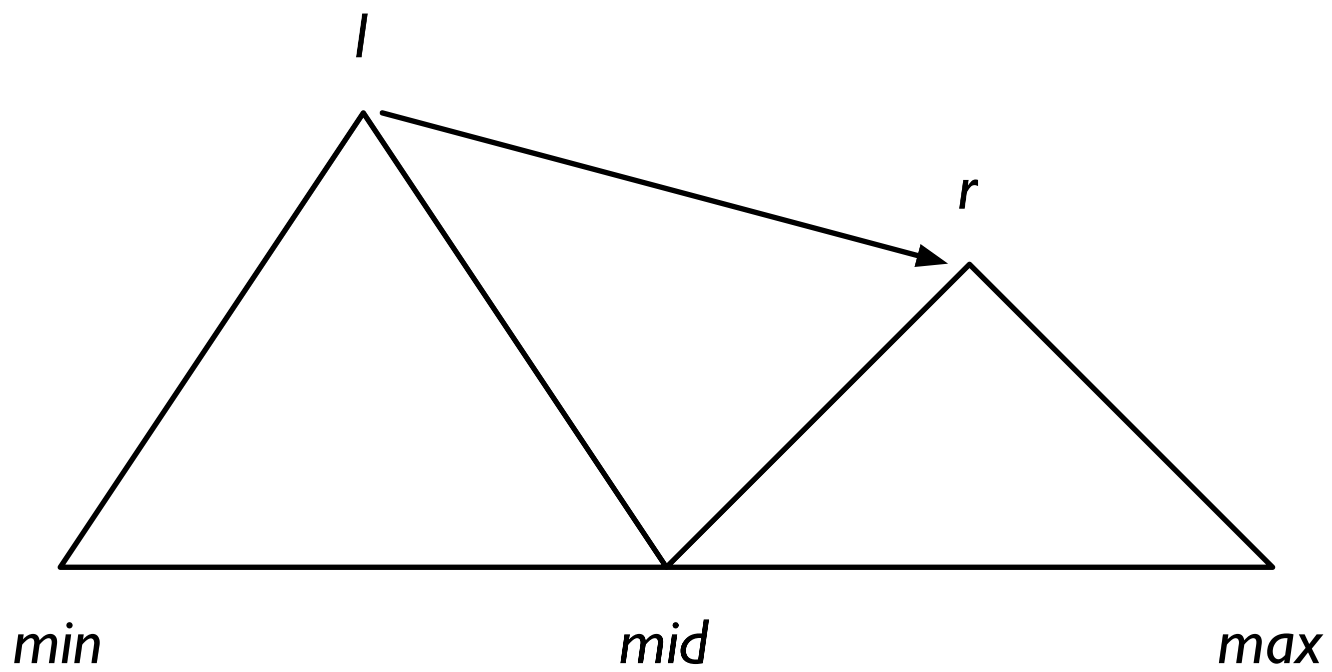
In Collins' algorithm, adding a left-to-right arc is done in one single step, specified by 5 positions.

Basic idea



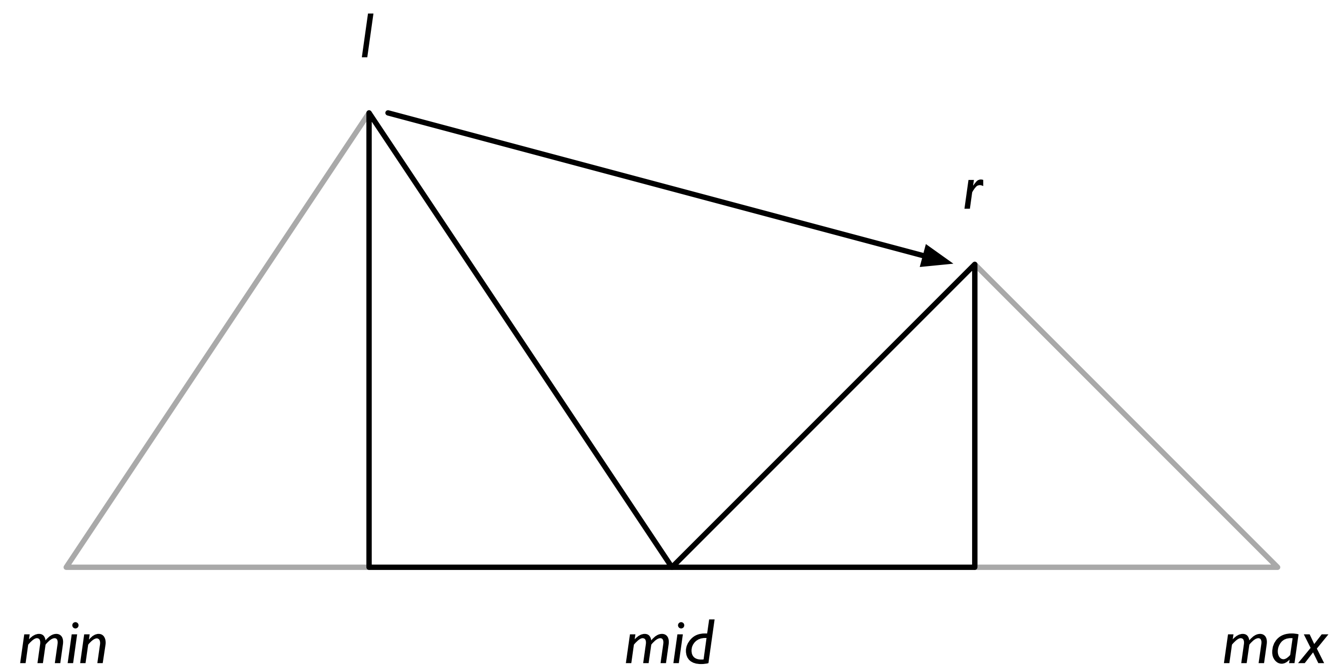
In Collins' algorithm, adding a left-to-right arc is done in one single step, specified by 5 positions.

Basic idea



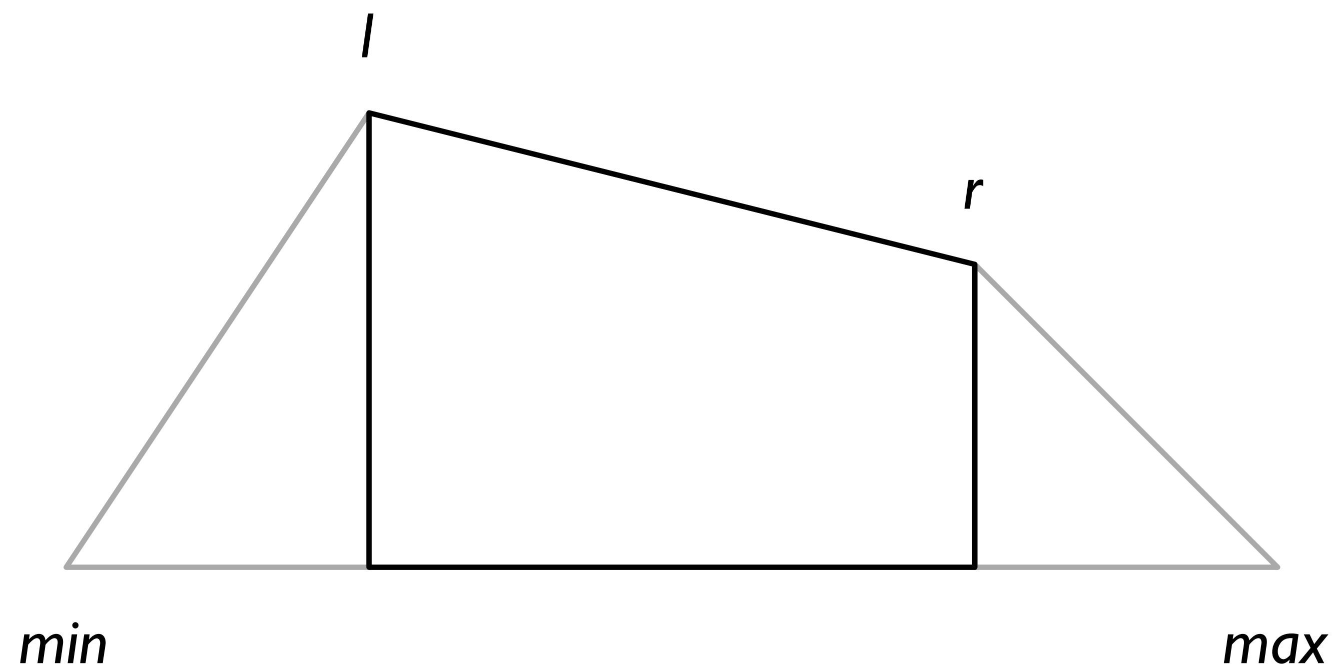
In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.

Basic idea



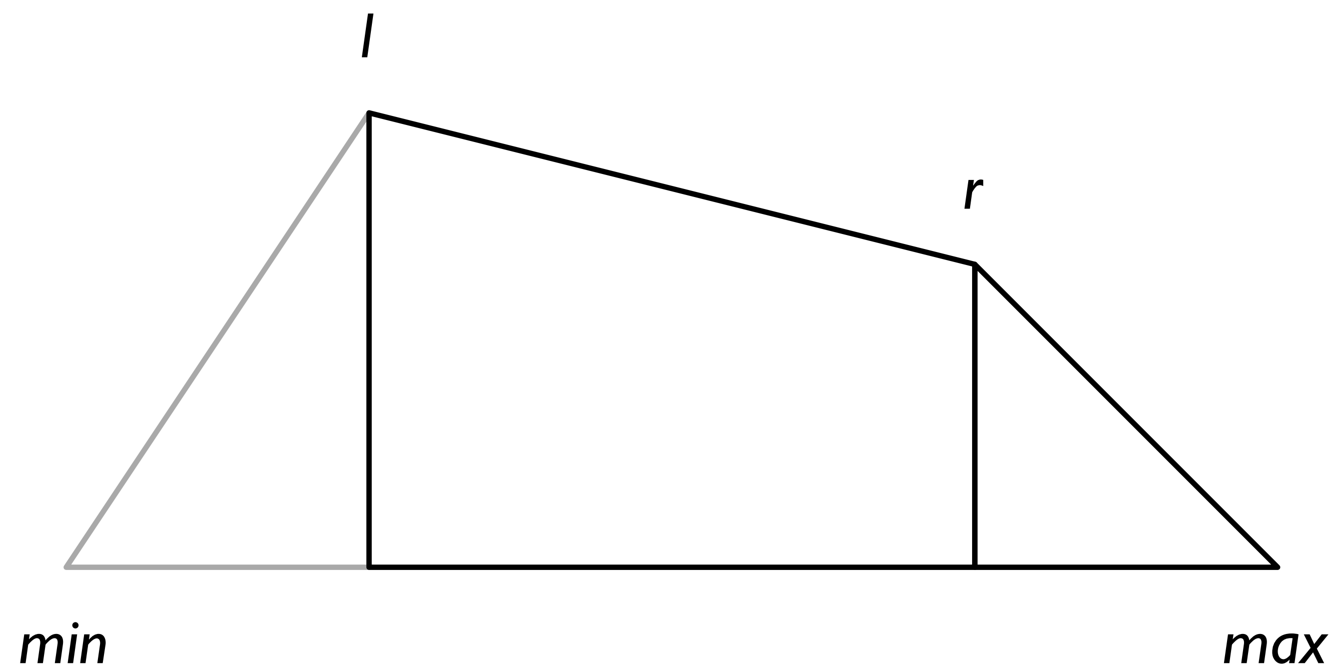
In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.

Basic idea



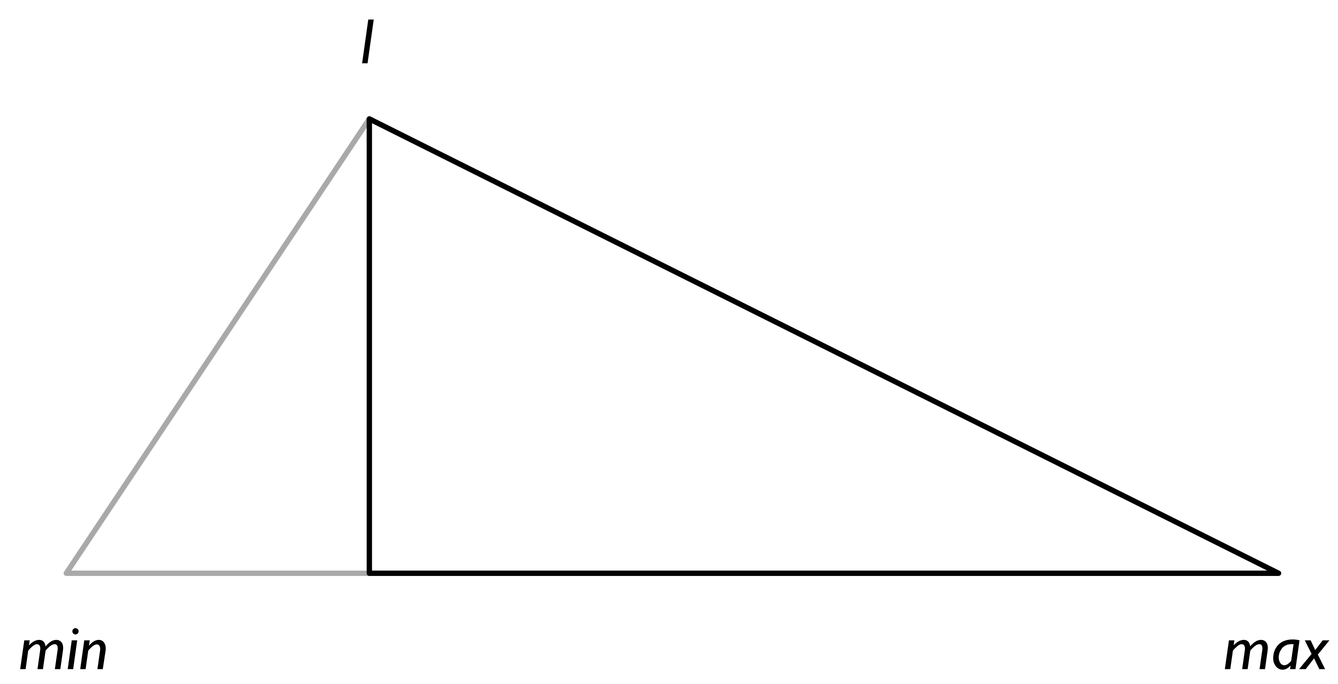
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.

Basic idea



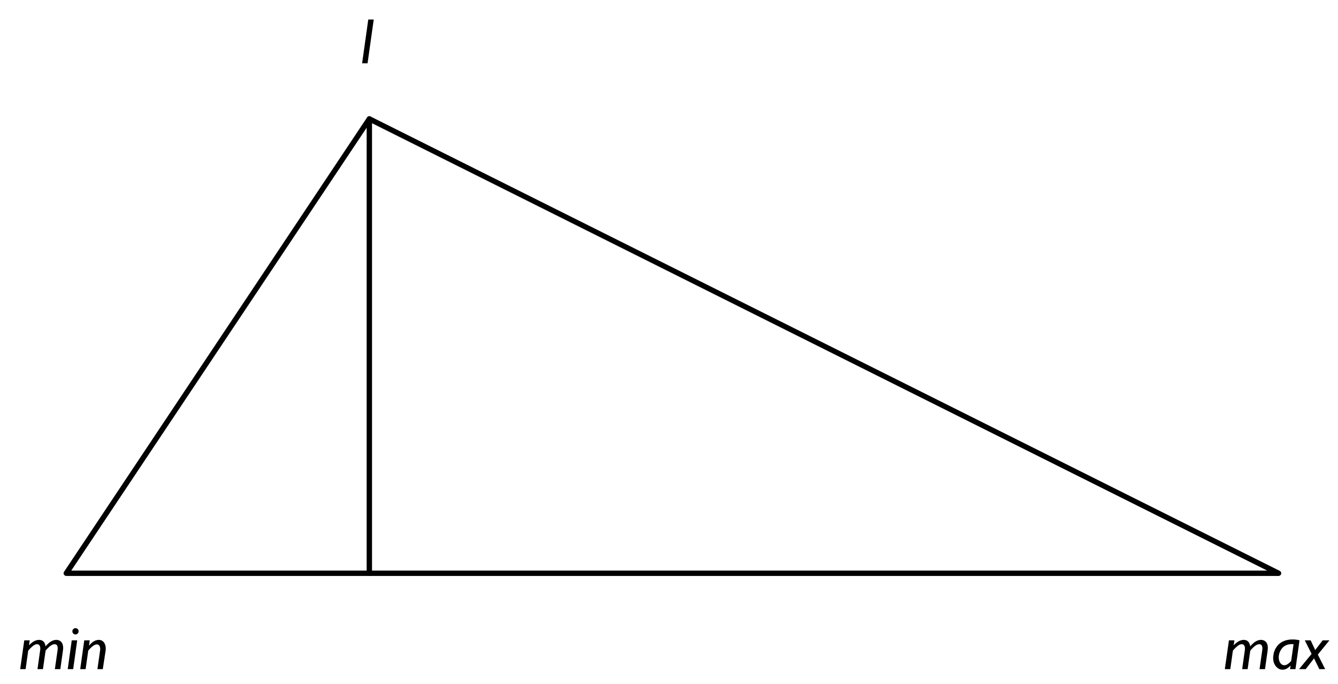
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.

Basic idea



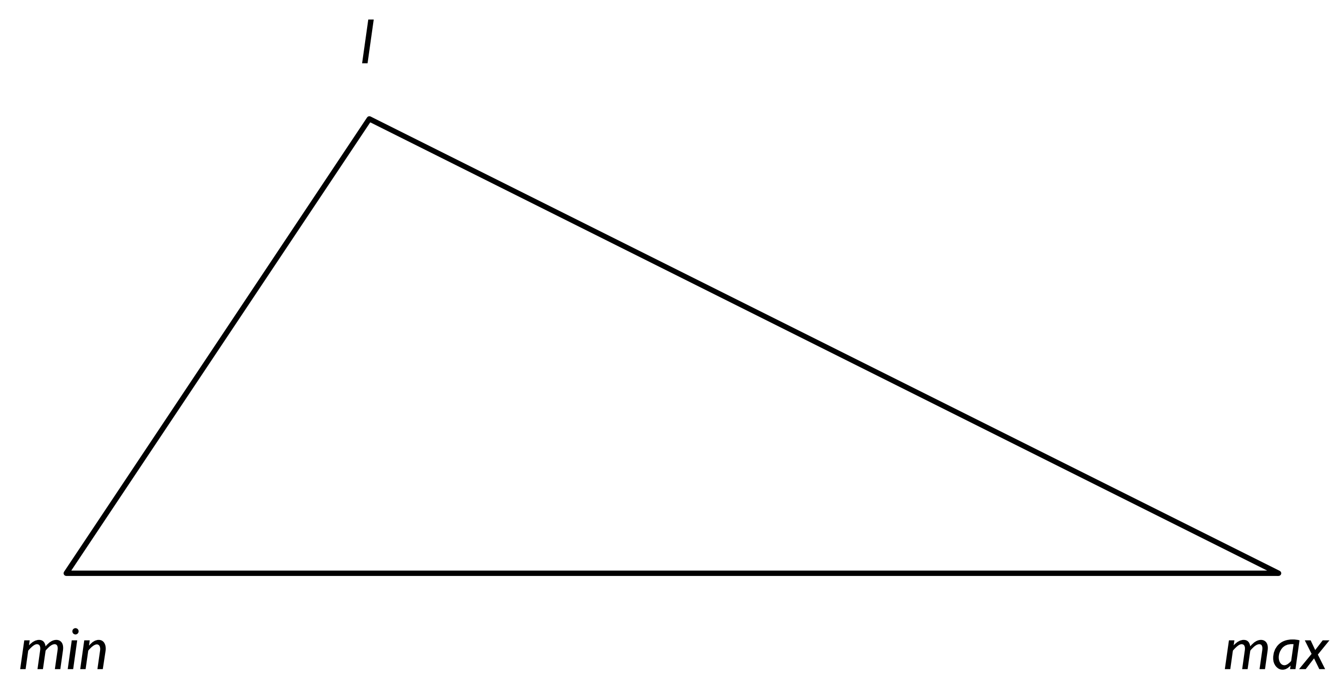
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.

Basic idea



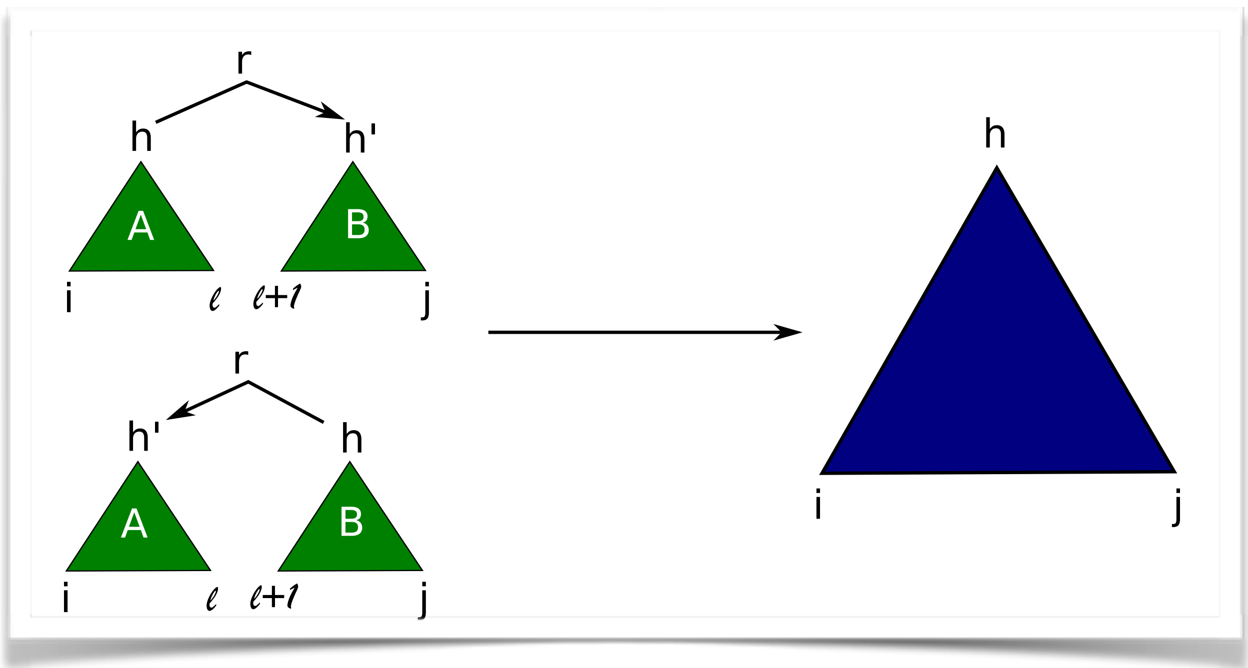
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.

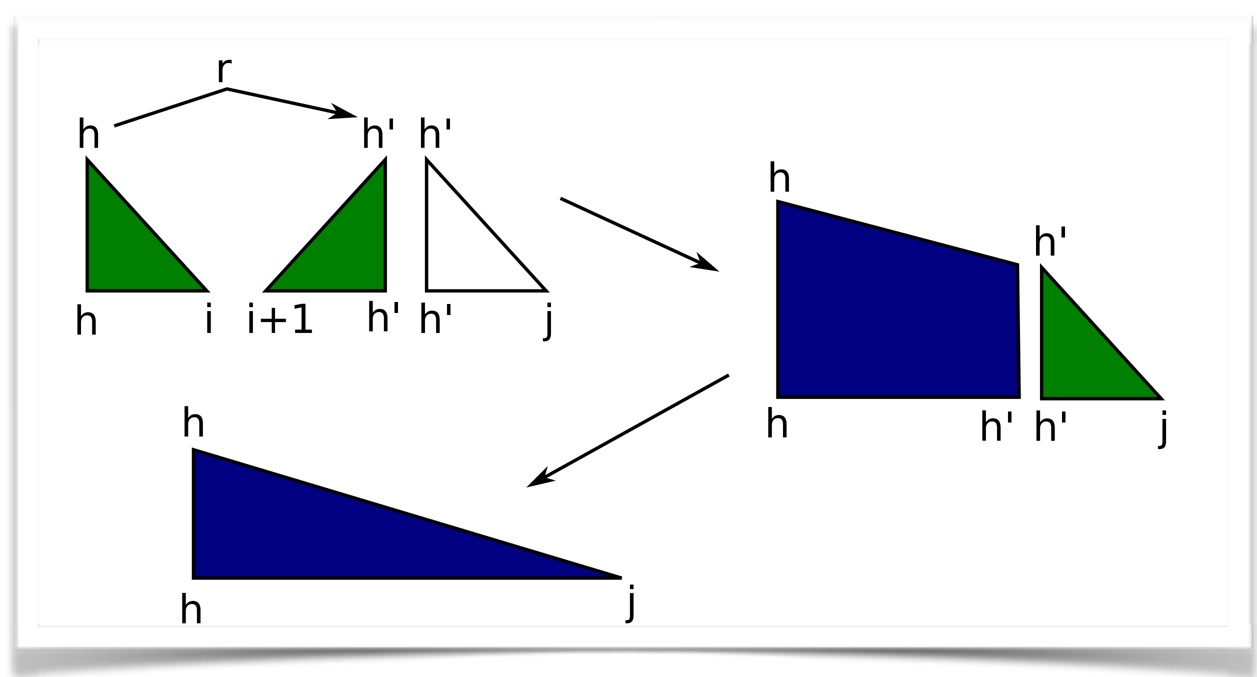
Basic idea



In Eisner's algorithm, the same thing is done in three steps, each one specified by 3 positions.

Comparison



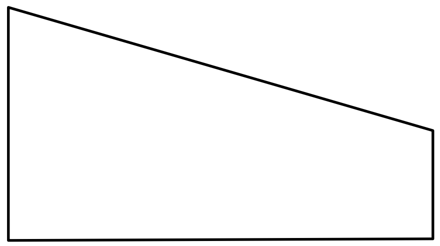


Dynamic programming tables

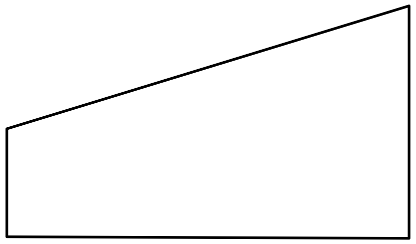
- Collins':
  - [min,max,head]
- Eisner's
  - [min,max,head-side,complete]
    - head-side (binary): is head to the left or right?
    - complete (binary:) is the non-head side still looking for dependents?

Graphic representation

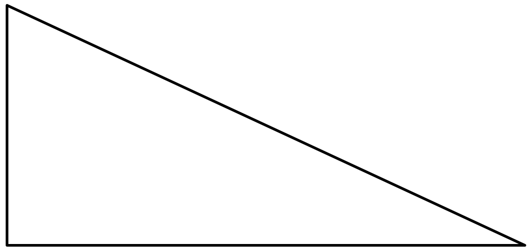
- [min,max,left,yes]



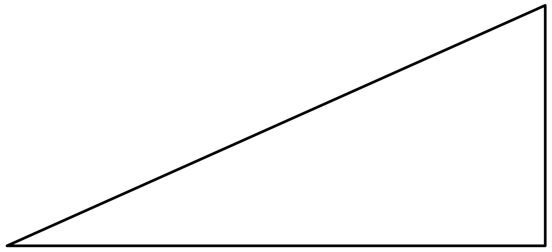
- [min,max,right,yes]



- [min,max,left,no]

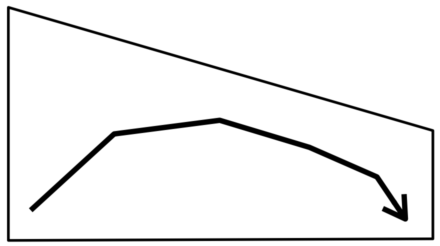


- [min,max,right,no]

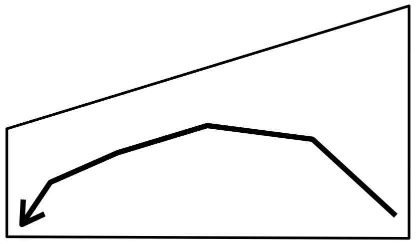


Graphic representation

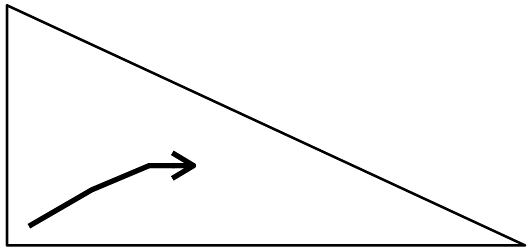
- [min,max,left,yes]



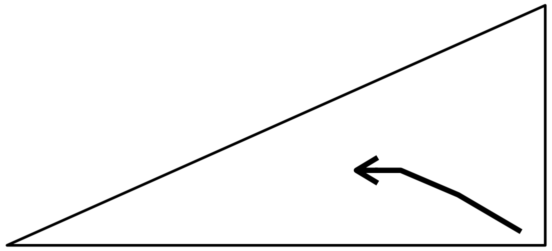
- [min,max,right,yes]



- [min,max,left,no]

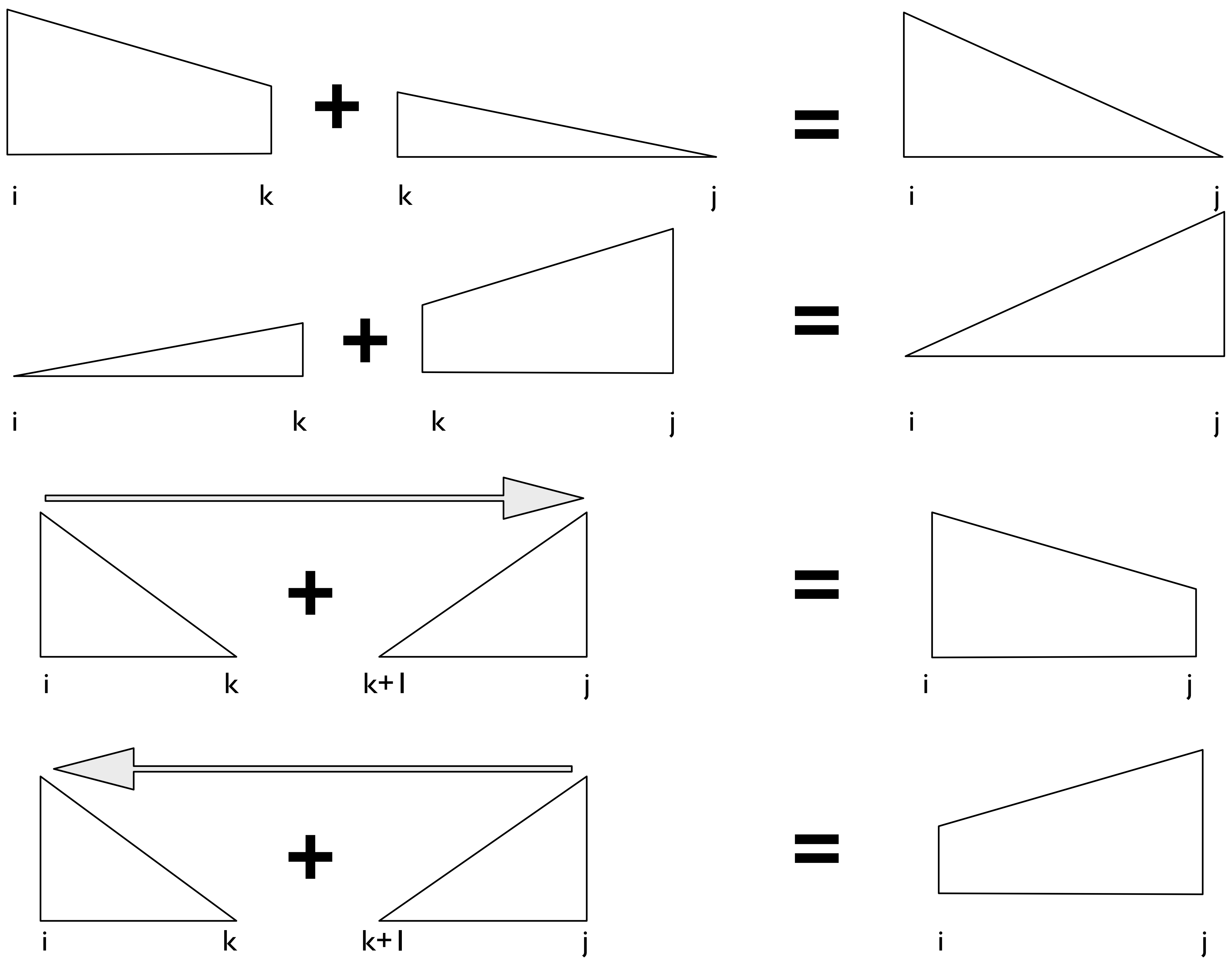


- [min,max,right,no]



Eisner's algorithm

Possible operations



Pseudo code

```
for each i from 0 to n and all d,c do
  C[i][i][d][c] = 0.0
for each m from 1 to n do
  for each i from 0 to n-m do
     j = i+m
     C[i][j][←][1] = max_{i≤q<j}(C[i][q][→][0] + C[q+1][j][←][0]+score(w_j,w_i)
     C[i][j][→][1] = max_{i≤q<j}(C[i][q][→][0] + C[q+1][j][←][0]+score(w_i,w_j)
     C[i][j][←][0] = max_{i≤q<j}(C[i][q][←][0] + C[q][j][←][1])
     C[i][j][→][0] = max_{i≤q<j}(C[i][q][→][1] + C[q][j][→][0])
return [0][n][→][0]
```

Summary

- Eisner's algorithm is an improvement over Collin's algorithm that runs in time $O(|w|^3)$.
- The same scoring model can be used.
- The same technique for extending the parser to labeled parsing can be used, adding $O(|L||w|^2)$ to the run time.
- Eisner's algorithm is the basis of current arc-factored dependency parsers.

Minimum-spanning tree parsing

- Based on graph algorithms to find the minimum spanning tree
  - Often: Chu-Liu-Edmonds algorithm (CLU)
- Directly produces non-projective trees
- First suggested in the MSTparser
- One of the most popular algorithms today

UPPSALA UNIVERSITET

Minimum-spanning tree parsing

- **Intuition:**
- Score all word pairs in both directions
- Create a fully connected graph with these scores
- Remove all edges going into ROOT
- For each node, greedily keep only the highest-scoring incoming arc
    - If this produces a tree: done!
    - Otherwise: handle each cycle in the graph

Evaluation of dependency parsers

- labelled attachment score (LAS):
percentage of correct arcs,
relative to the gold standard
- labelled exact match (LEM):
percentage of correct dependency trees,
relative to the gold standard
- unlabelled attachment score/exact match (UAS/UEM):
the same, but ignoring arc labels

UPPSALA UNIVERSITET

Accuracy vs precision/recall

- Attachment score is an accuracy score
- For phrase-structure parsing we reported precision and recall
- Why is that not done for dependency parsing?

UPPSALA UNIVERSITET

Coming up

- Monday, Feb 20: guest lecture, Paola Merlo, 2-K1023
- Wednesday, Feb 22, Lecture:
  - Transition-based parsing (watch videos first)
- Sign up for a project and hand in a proposal in Studium (DL: February 27)
- Literature seminar 2, March 2
- Do assignment 2, literature review (DL: March 6)
- Start looking at the dependency assignment (DL: March 13)
  - Supervision: Feb 27 and March 8