The Eisner algorithm

Marco Kuhlmann
Department of Computer and Information Science

LINKÖPING UNIVERSITY

The Eisner algorithm

- The Eisner algorithm is an algorithm for computing the highest-scoring projective dependency tree under an arc-factored model.
- It solves this problem using bottom–up dynamic programming, storing solutions to sub-problems in a table.

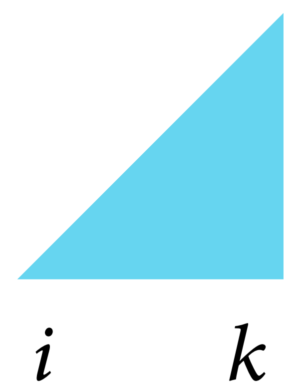
Overview of the Eisner algorithm

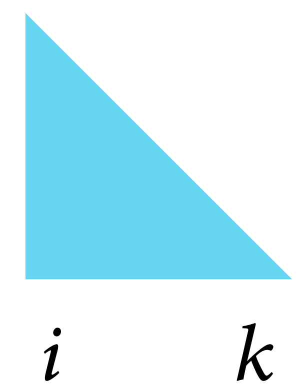
- We are given an input sentence x and a table A that holds the score of each possible arc: $A[h][d] = \text{score}(x, h \to d)$
- We will fill four tables T_t such that each entry $T_t[i][k]$ will hold the maximal possible score of a certain type of graph, where i and k identify the leftmost and the rightmost words in the graph.
- Once we are done, we will use the tables to compute the maximal possible score of a projective tree for the full sentence.

Sub-problems in the Eisner algorithm

type 1
right-rooted triangle

tree with
root at position k



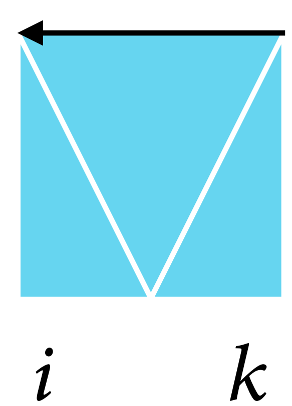


type 2
left-rooted triangle

tree with
root at position i

type 3
box with right-to-left arc

pair of trees with
arc from k to i



type 4
box with left-to-right arc

pair of trees with
arc from i to k

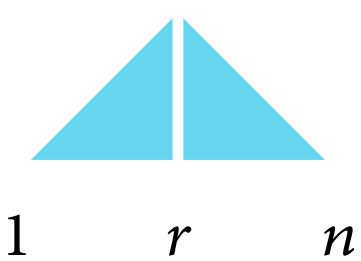
Filling the score tables

```
# Assume that all table entries are initialised with -inf

foreach i from 1 to n:

    T₁[i][i] = 0

    T₂[i][i] = 0

foreach k from 2 to n:

    foreach i from k-1 downto 1:

        T₄[i][k] = max over j from i to k-1 of T₂[i][j] + T₁[j+1][k] + A[i][k]

        T₃[i][k] = max over j from i to k-1 of T₂[i][j] + T₁[j+1][k] + A[k][i]

        T₂[i][k] = max over j from i+1 to k of T₄[i][j] + T₂[j][k]

        T₁[i][k] = max over j from i to k-1 of T₁[i][j] + T₃[j][k]
```

Computing the maximal score for the full sentence

- To obtain the maximal possible score of a projective dependency tree for a full sentence of length n, we compute

$$\max_r \left(T_1[1][r] + T_2[r][n]\right)$$



- Alternatively, we can introduce a special 'pseudo-root' with position 0 and directly extract $T_2[0][n]$, the maximal possible score of a triangle rooted at the pseudo-root.

  allows solutions with multiple 'real roots'

Filling the backpointer tables

```
# Assume that all table entries are initialised with None

foreach i from 1 to n:

    B₁[i][i] = None

    B₂[i][i] = None

foreach k from 2 to n:

    foreach i from k−1 downto 1:

        B₄[i][k] = argmax over j from i to k−1 of T₂[i][j] + T₁[j+1][k] + A[i][k]

        B₃[i][k] = argmax over j from i to k−1 of T₂[i][j] + T₁[j+1][k] + A[k][i]

        B₂[i][k] = argmax over j from i+1 to k of T₄[i][j] + T₂[j][k]

        B₁[i][k] = argmax over j from i to k−1 of T₁[i][j] + T₃[j][k]
```

Constructing the tree from the backpointers

```
def build(t, i, k):
    j = Bt[i][k]    # retrieve the entry from the backpointer table
    if j == None:  return ∅
    if t == 4:  return build(2, i, j) ∪ build(1, j+1, k) ∪ {(i, k)}
    if t == 3:  return build(2, i, j) ∪ build(1, j+1, k) ∪ {(k, i)}
    if t == 2:  return build(4, i, j) ∪ build(2, j, k)
    if t == 1:  return build(1, i, j) ∪ build(3, j, k)

# To construct the full tree (right-rooted triangle + left-rooted triangle):
arcs = build(1, 1, r) ∪ build(2, r, n)
```

Complexity analysis of the Eisner algorithm

Let n be the length of the input sentence.

- The space complexity of the Eisner algorithm is in $O(n^2)$; this corresponds to the number of cells in a table T_t.
- The runtime complexity of the Eisner algorithm is in $O(n^3)$; this corresponds to the number of nested *for* loops that we need to enumerate sub-problems and compute maximal values.

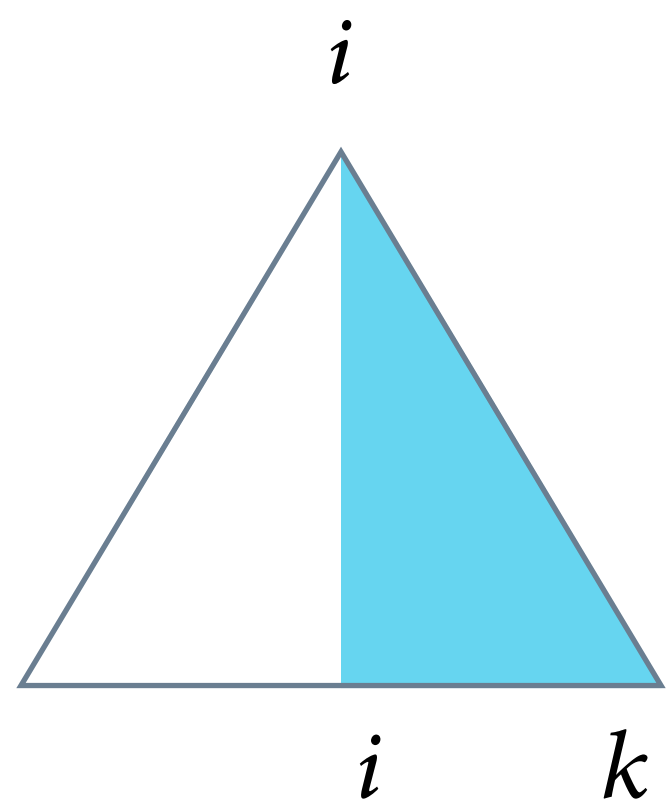
Eisner algorithm: Correctness

Lemma: For every type t and all $i \leq k$, the value $T_t[i][k]$ is the maximal possible score of a graph of type t with endpoints i and k.

Proof: by induction on the size of the graph, $k - i$

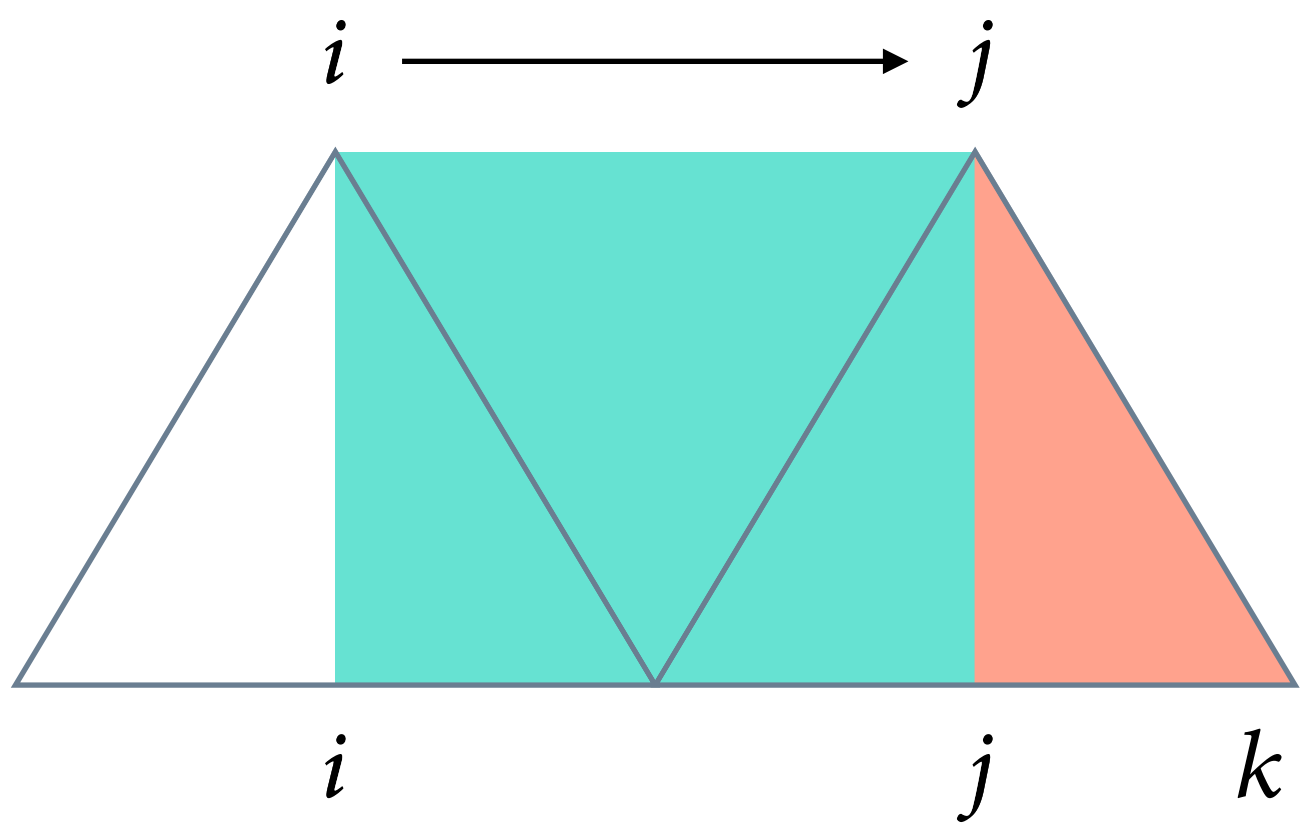
- In the simplest case, we have a graph with a single vertex and no arcs. The score of such a graph is zero.
- In the general case, we have a graph with at least one arc. We can then decompose the graph into two smaller graphs.

Eisner algorithm: Correctness



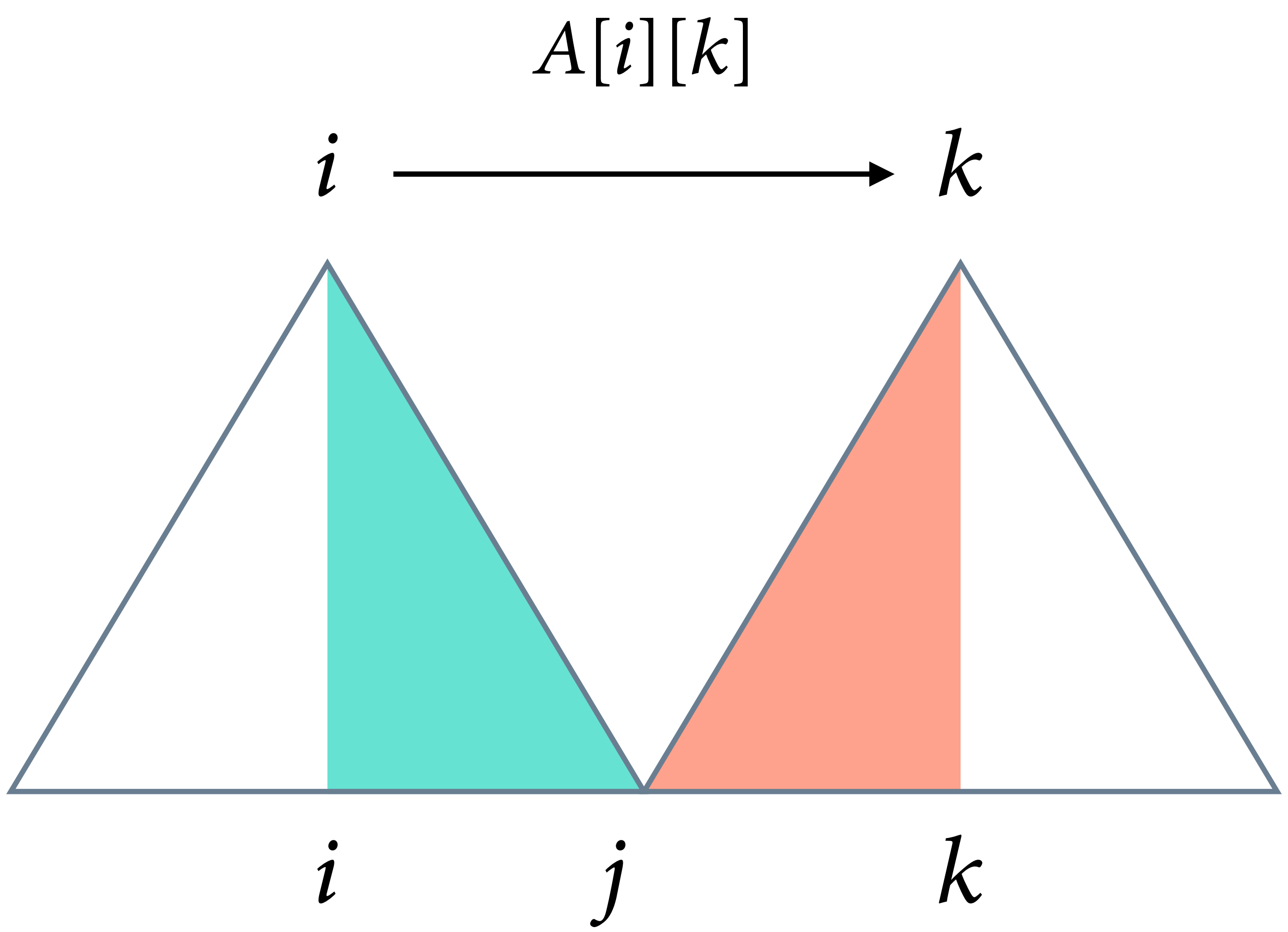
$T_2[i][k]$?

Eisner algorithm: Correctness



$$T_2[i][k] \;=\; \max_j \left(T_4[i][j] + T_2[j][k]\right)$$

Eisner algorithm: Correctness



$$T_4[i][k] \;=\; \max_j \left(\boxed{T_2[i][j]} + \boxed{T_1[j+1][k]} + A[i][k] \right)$$