Chapter 3
Dynamic Programming

Algorithm Theory
WS 2016/17

Fabian Kuhn

Dynamic Programming = Recursion + Memoitation

„Memoization“ for increasing the efficiency of a recursive solution:

- Only the *first time* a sub-problem is encountered, its solution is computed and then stored in a table. Each subsequent time that the subproblem is encountered, the value stored in the table is simply looked up and returned

  (without repeated computation!).

- *Computing the solution*: For each sub-problem, store how the value is obtained (according to which recursive rule).

Dynamic Programming

Dynamic programming / memoization can be applied if

- Optimal solution contains optimal solutions to sub-problems (recursive structure)

- Number of sub-problems that need to be considered is small

Edit Distance

Given: Two strings $A = a_1 a_2 \dots a_m$ and $B = b_1 b_2 \dots b_n$

Goal: Determine the minimum number $D(A, B)$ of edit operations required to transform A into B

Edit operations:

a) **Replace** a character from string A by a character from B

b) **Delete** a character from string A

c) **Insert** a character from string B into A

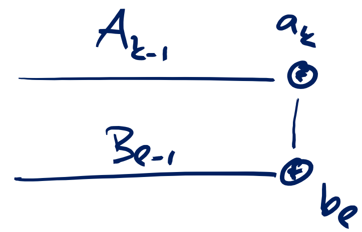
```
m a - t h e m - - a t i c i a n
m u l t i p l i c a t i o - - n
```

Computation of the Edit Distance

Three ways of ending an "alignment" between A_k and B_ℓ:

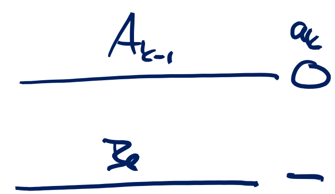
1. a_k is replaced by b_ℓ:

   $$D_{k,\ell} = D_{k-1,\ell-1} + c(a_k, b_\ell)$$



2. a_k is deleted:

   $$D_{k,\ell} = D_{k-1,\ell} + c(a_k, \varepsilon)$$



3. b_ℓ is inserted:

   $$D_{k,\ell} = D_{k,\ell-1} + c(\varepsilon, b_\ell)$$

Computing the Edit Distance

- Recurrence relation (for $k, \ell \geq 1$)

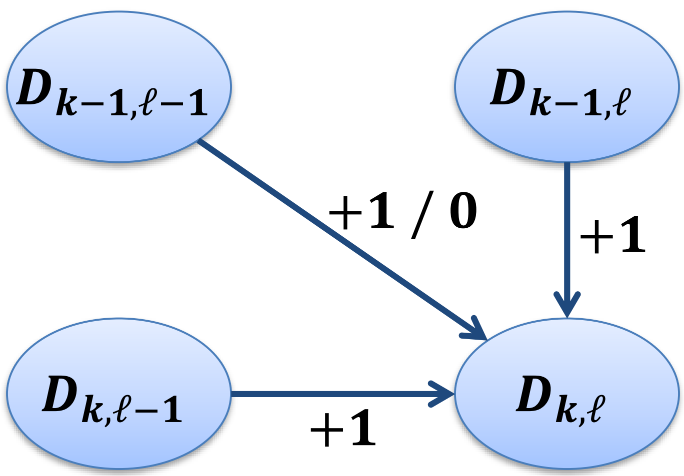
$$D_{k,\ell} = \min \begin{Bmatrix} D_{k-1,\ell-1} & + c(a_k, b_\ell) \\ D_{k-1,\ell} & + c(a_k, \varepsilon) \\ D_{k,\ell-1} & + c(\varepsilon, b_\ell) \end{Bmatrix} = \underbrace{\min \begin{Bmatrix} D_{k-1,\ell-1} & + 1\,/\,0 \\ D_{k-1,\ell} & + 1 \\ D_{k,\ell-1} & + 1 \end{Bmatrix}}_{\textbf{unit cost model}}$$

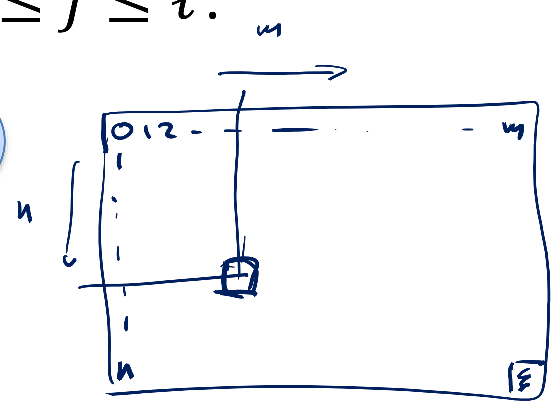
- Need to compute $D_{i,j}$ for all $0 \leq i \leq k$, $0 \leq j \leq \ell$: m

$D_{0,0} = 0$

$D_{0,j} = D_{0,j-1} + c(\varepsilon, b_j)$

$D_{i,0} = D_{i-1,0} + c(a_i, \varepsilon)$

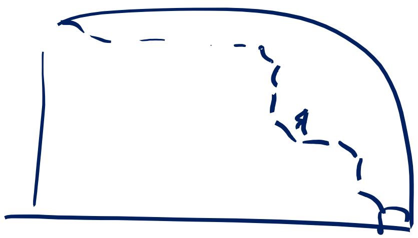




Edit Distance: Summary

- Edit distance between two strings of length m and n can be computed in $O(mn)$ time.

- Obtain the edit operations:
  - for each cell, store which rule(s) apply to fill the cell
  - track path backwards from cell (m, n)
  - can also be used to get all optimal “alignments”



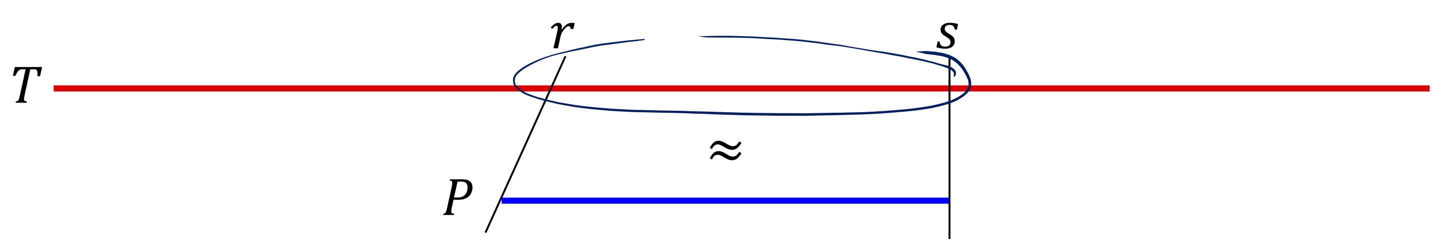
- Unit cost model:
  - interesting special case
  - each edit operation costs 1

Approximate String Matching $n \gg m$

Given: strings $T = t_1 t_2 \ldots t_n$ (text) and $P = p_1 p_2 \ldots p_m$ (pattern).

Goal: Find an interval $[r, s]$, $1 \leq r \leq s \leq n$ such that the sub-string $T_{r,s} := t_r \ldots t_s$ is the one with highest similarity to the pattern P:

$$\underset{1 \leq r \leq s \leq n}{\arg\min}\, D\left(T_{r,s}, P\right)$$



Approximate String Matching

Naive Solution:

for all $1 \leq r \leq s \leq n$ **do**

compute $D(T_{r,s}, P)$

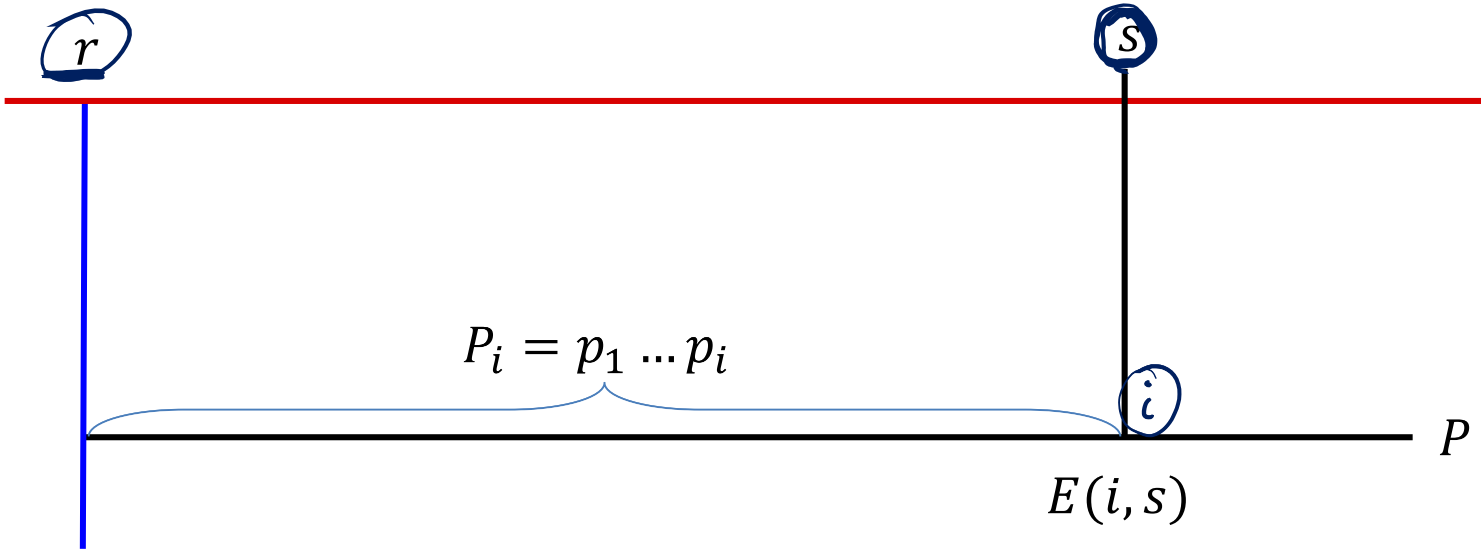
choose the minimum

$O(n^2 \cdot n \cdot m)$

Approximate String Matching

A related problem:

- For each position s in the text and each position i in the pattern compute the minimum edit distance $E(i, s)$ between $P_i = p_1 \dots p_i$ and any substring $T_{r,s}$ of T that ends at position s.

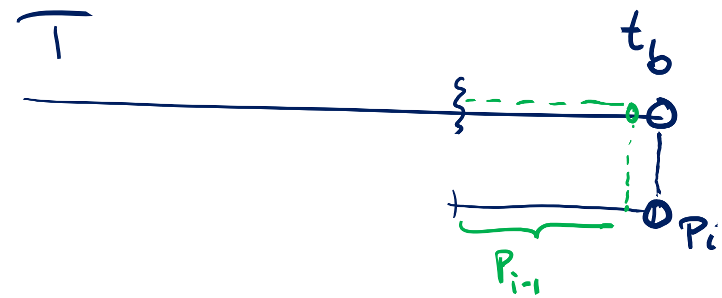


Approximate String Matching

Three ways of ending optimal alignment between T_b and P_i:

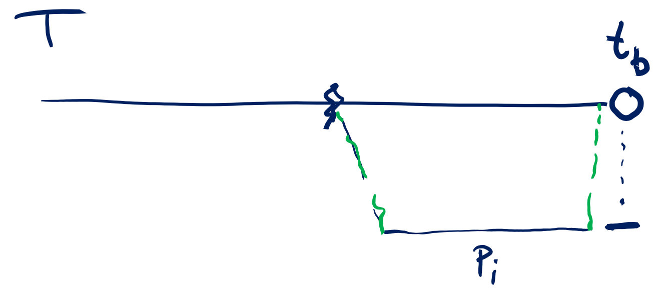
1. t_b is replaced by p_i:

   $$E_{b,i} = E_{b-1,i-1} + c(t_b, p_i)$$



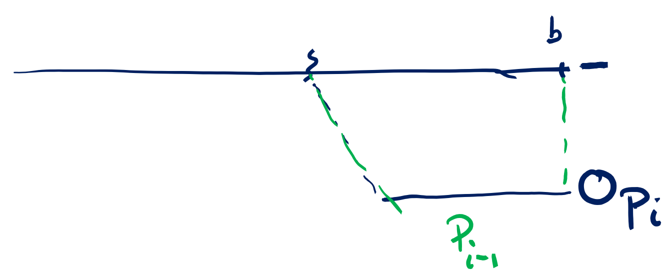
2. t_b is deleted:

   $$E_{b,i} = E_{b-1,i} + c(t_b, \varepsilon)$$



3. p_i is inserted:

   $$E_{b,i} = E_{b,i-1} + c(\varepsilon, p_i\ )$$



Approximate String Matching

Recurrence relation (unit cost model):

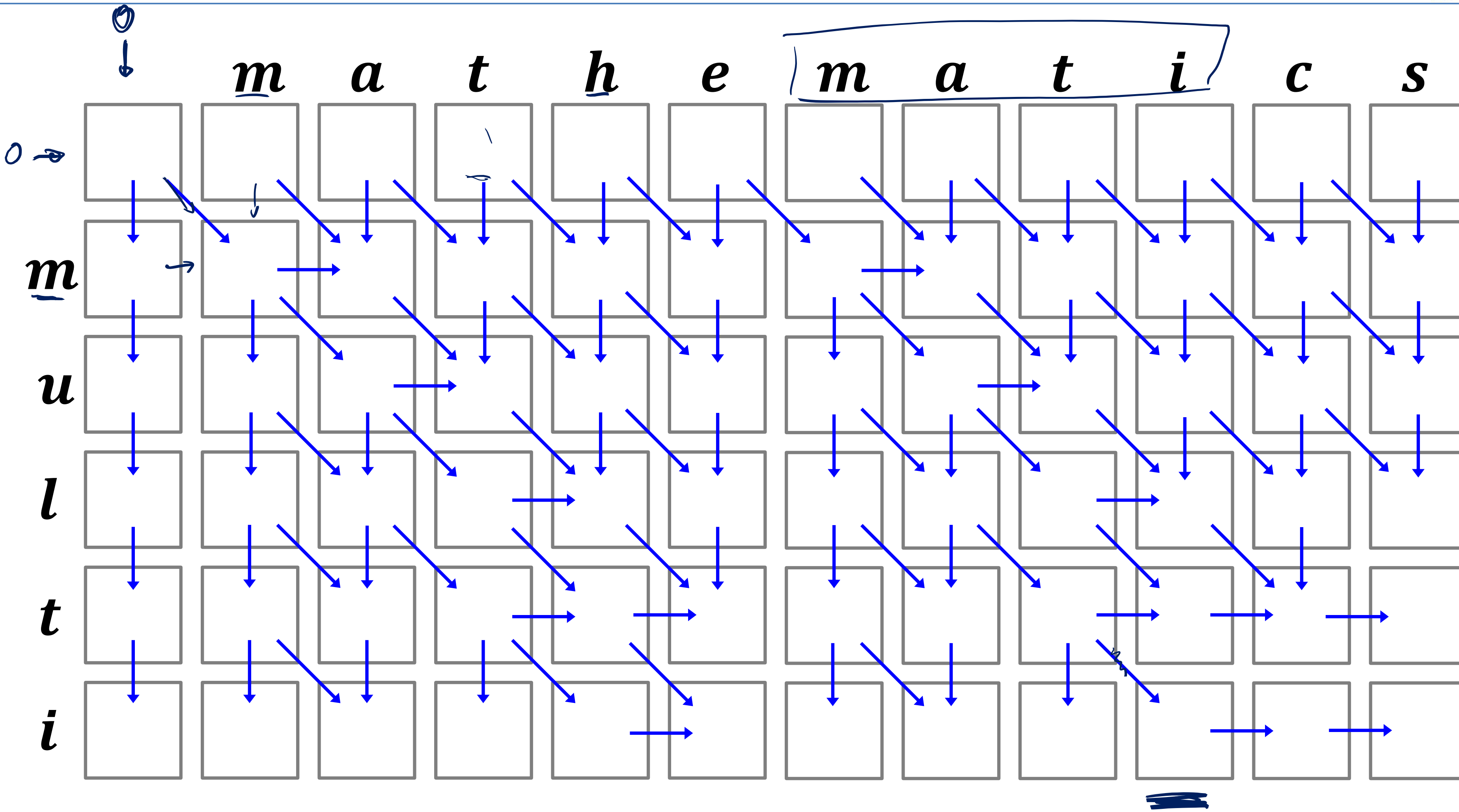
$$E_{b,i} = \min \begin{Bmatrix} E_{b-1,i-1} + 1 \\ E_{b-1,i} \quad + 1 \\ E_{b,i-1} \quad + 1 \end{Bmatrix}$$

Base cases:

$$E_{0,0} = 0$$
$$E_{0,i} = i$$
$$E_{i,0} = 0$$

Example

math|h m   mathem m $O(m \cdot n)$



$E_{b,i}$: cost of best alignment of P_i with a substr. of T ending in pos. b

m a - t i
m u l t i

Approximate String Matching

- Optimal matching consists of optimal sub-matchings
- Optimal matching can be computed in $O(mn)$ time
- Get matching(s):
  - Start from minimum entry/entries in bottom row
  - Follow path(s) to top row
- Algorithm to compute $E(b, i)$ identical to edit distance algorithm, except for the initialization of $E(b, 0)$

Related Problems in Bioinformatics

Sequence Alignment:

Find optimal alignment of two given DNA, RNA, or amino acid sequences.

```
G A – C G G A T T A G
G A T C G G A A T – G
```

Global vs. Local Alignment:

- *Global alignment*: find optimal alignment of 2 sequences
- *Local alignment*: find optimal alignment of sequence 1 (patter) with sub-sequence of sequence 2 (text)