Chapter 3
Dynamic Programming

Algorithm Theory
WS 2014/15

Fabian Kuhn

Edit Distance

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

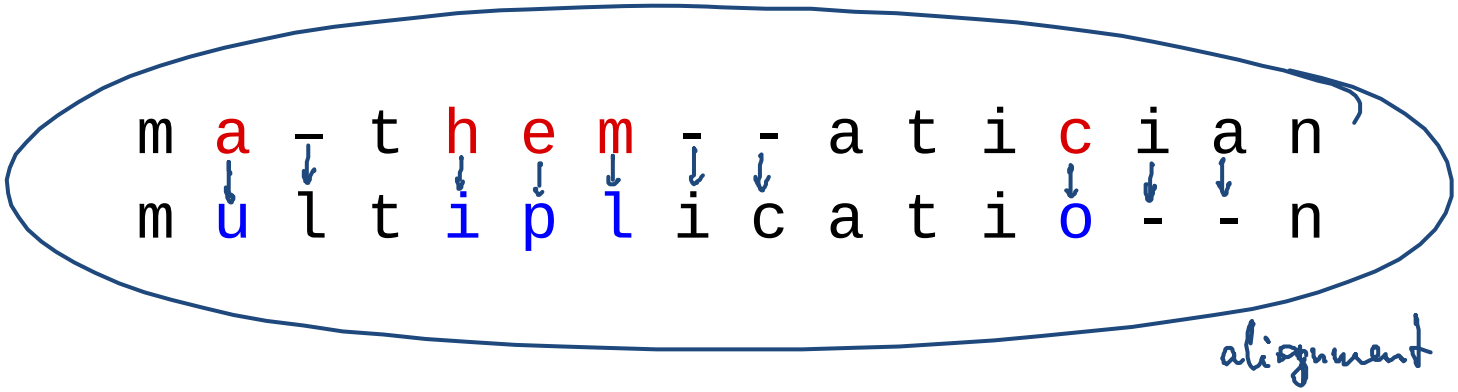
Goal: Determine the minimum number $D(A, B)$ of edit operations required to transform A into B

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



Edit Distance – Cost Model

- Cost for **replacing** character a by b: $\boldsymbol{c(a,b) \geq 0}$

- Capture insert, delete by allowing $a = \varepsilon$ or $b = \varepsilon$:
  - Cost for **deleting** character a: $\boldsymbol{c(a,\varepsilon)}$
  - Cost for **inserting** character b: $\boldsymbol{c(\varepsilon,b)}$

- **Triangle inequality**:

$$c(a,c) \leq c(a,b) + c(b,c)$$

→ each character is changed at most once!

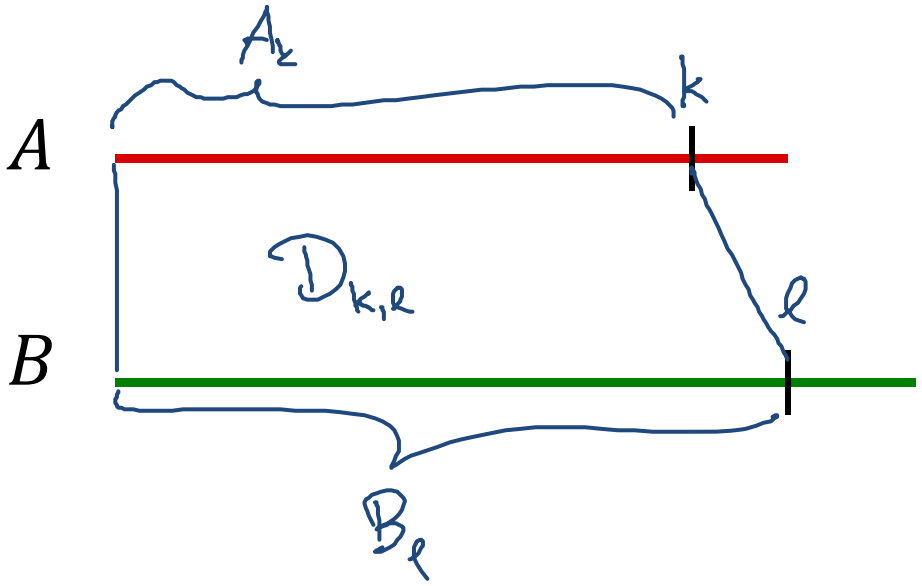
- **Unit cost model**: $c(a,b) = \begin{cases} 1, & \text{if } a \neq b \\ 0, & \text{if } a = b \end{cases}$

Computation of the Edit Distance

Let $A_k := a_1 \dots a_k$, $B_\ell := b_1 \dots b_\ell$, and

$$D_{k,\ell} := D(A_k, B_\ell)$$

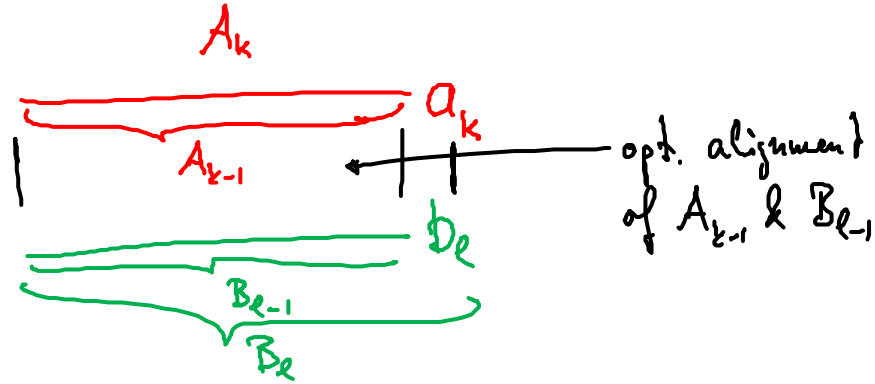
dynamic programming: recursive structure



Optimal "Alignment" between A_k and B_ℓ

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

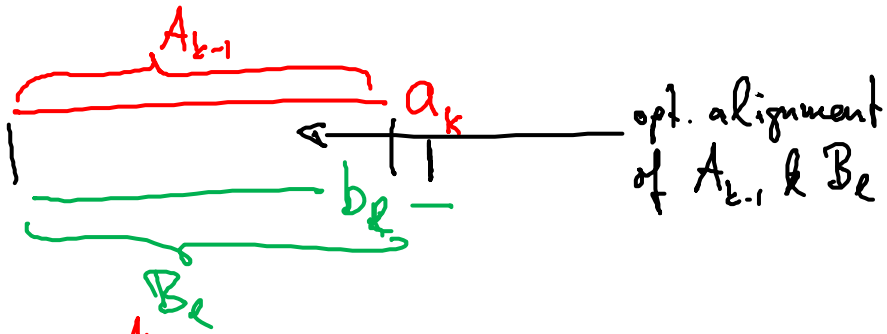
1)



case 1

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

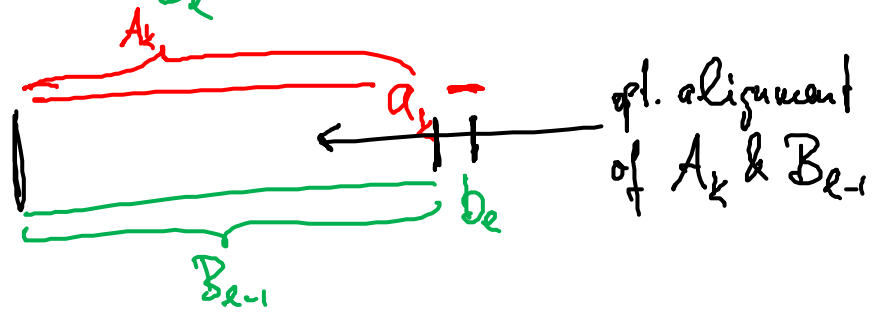
2)



case 2

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

3)



case 3

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

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:

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

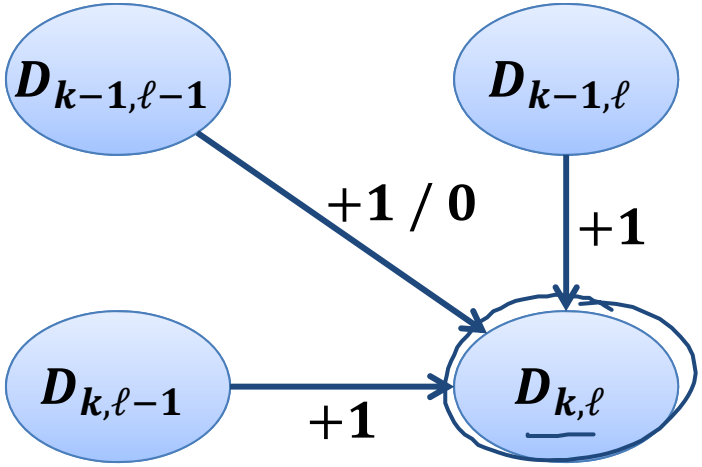
Computing the Edit Distance

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

$$D_{k,\ell} = \min \begin{Bmatrix} D_{k-1,\ell-1} + c(a_k, b_\ell) \\ D_{k-1,\ell} + c(a_k, \varepsilon) \\ D_{k,\ell-1} + c(\varepsilon, b_\ell) \end{Bmatrix} = \min \begin{Bmatrix} D_{k-1,\ell-1} + 1 \,/\, 0 \\ D_{k-1,\ell} + 1 \\ D_{k,\ell-1} + 1 \end{Bmatrix}$$

unit cost model

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



Recurrence Relation for the Edit Distance

Base cases:

$$D_{0,0} = D(\varepsilon, \varepsilon) = 0$$

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

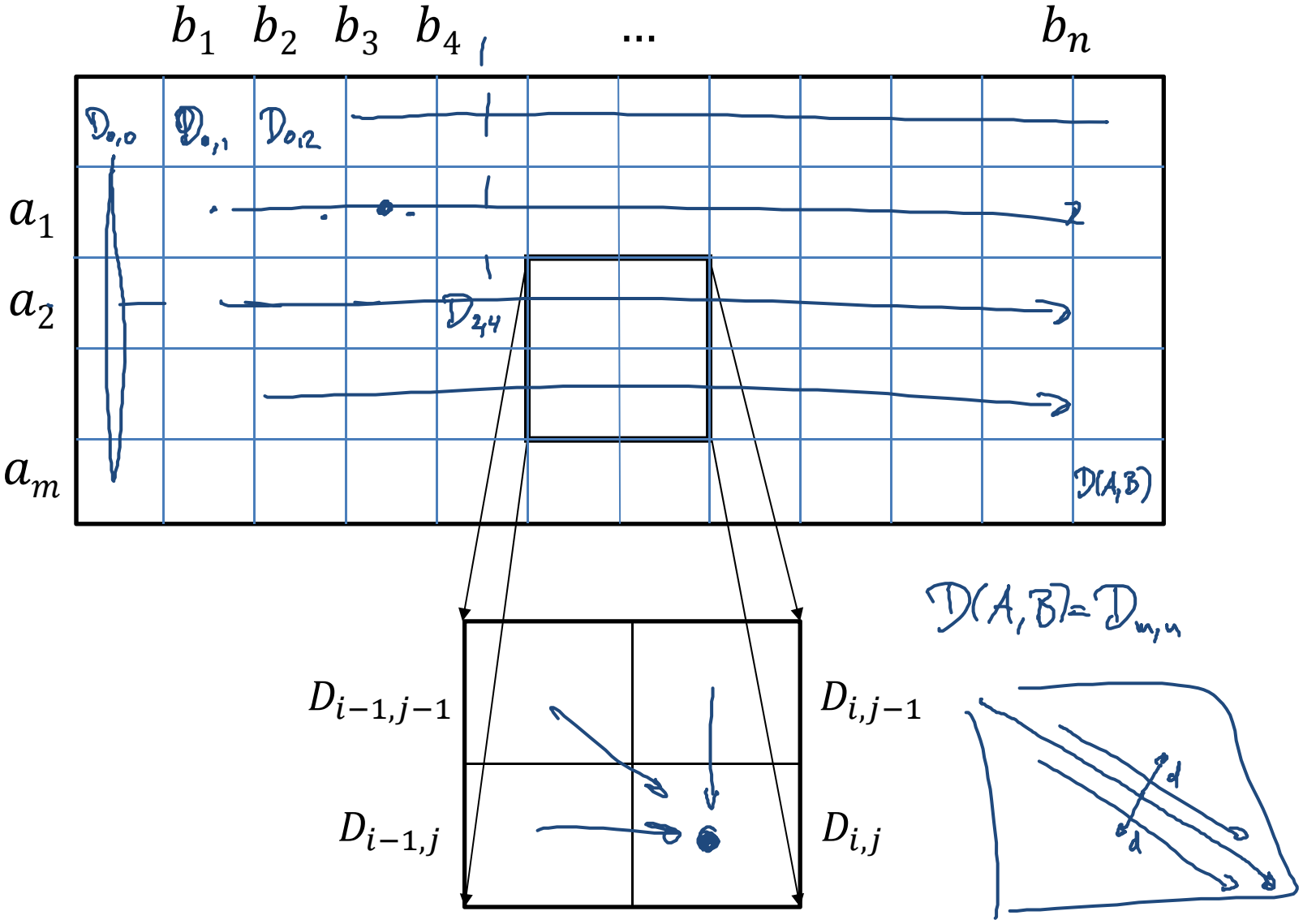
$$D_{i,0} = D(A_i, \varepsilon) = D_{i-1,0} + c(a_i, \varepsilon)$$

unit cost model: $D_{0,j} = j$, $D_{i,0} = i$

Recurrence relation:

$$D_{i,j} = \min \begin{cases} D_{k-1,\ell-1} + c(a_k, b_\ell) \\ D_{k-1,\ell} + c(a_k, \varepsilon) \\ D_{k,\ell-1} + c(\varepsilon, b_\ell) \end{cases}$$

Order of solving the subproblems



Algorithm for Computing the Edit Distance

Algorithm *Edit-Distance*

Input: 2 strings $A = a_1 \dots a_m$ and $B = b_1 \dots b_n$

Output: matrix $D = (D_{ij})$

1 $D[0,0] := 0;$

2 **for** $i := 1$ **to** m **do** $D[i,0] := i;$

3 **for** $j := 1$ **to** n **do** $D[0,j] := j;$

4 **for** $i := 1$ **to** m **do**

5 **for** $j := 1$ **to** n **do**

6 $$D[i,j] := \min \begin{Bmatrix} D[i-1,j] & + c(a_i, \varepsilon) \\ D[i,j-1] & + c(\varepsilon, b_j) \\ D[i-1,j-1] & + c(a_i, b_j) \end{Bmatrix};$$

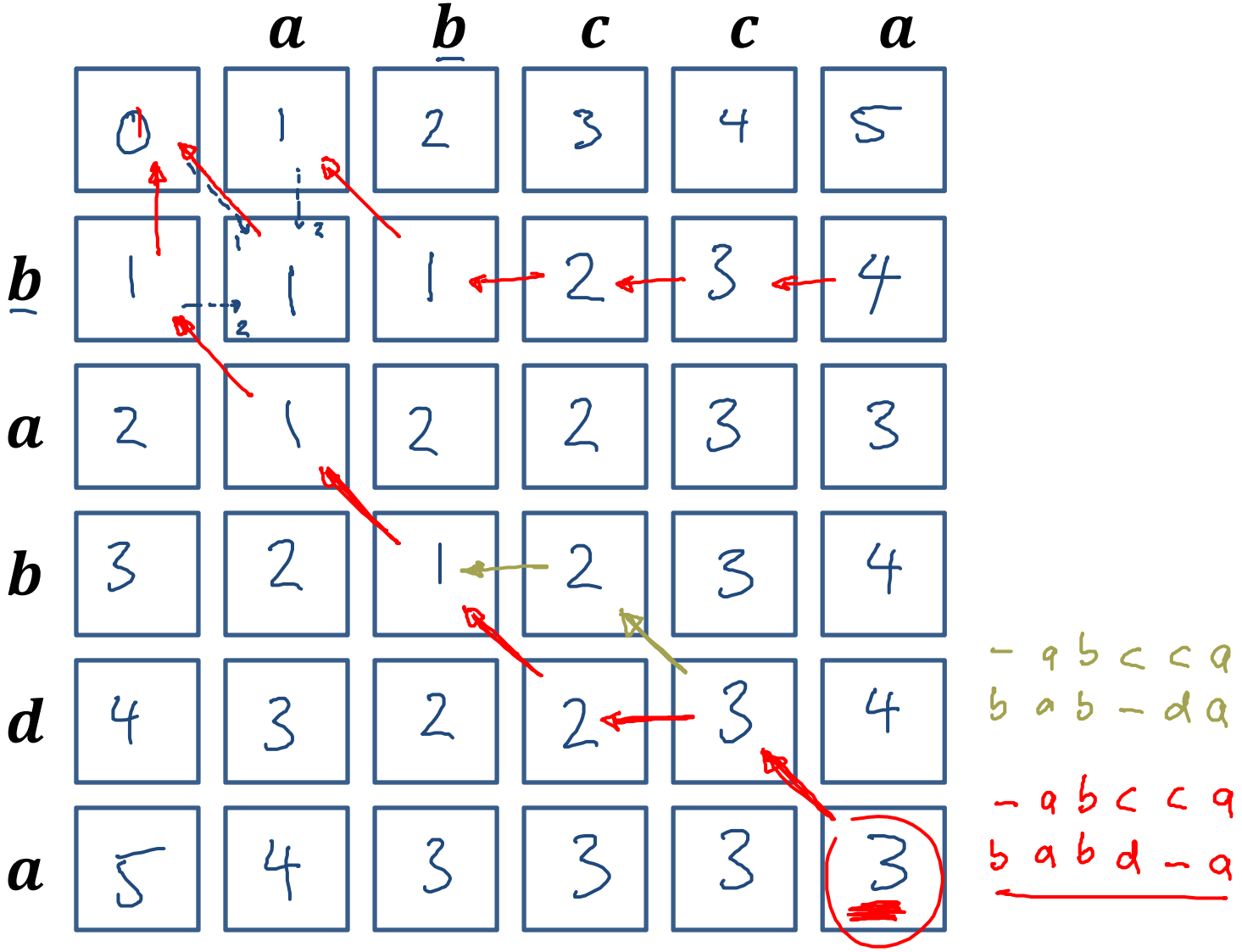
unit cost model

$D[i-1,0] + c(a_i, \varepsilon)$

$D[0,j-1] + c(\varepsilon, b_j)$

general cost

Example (unit cost model)



Computing the Edit Operations

Algorithm *Edit-Operations*(i,j)
Input: matrix D (already computed)
Output: list of edit operations

1 **if** $i = 0$ **and** $j = 0$ **then return** empty list

2 **if** $i \neq 0$ **and** $D[i,j] = D[i-1,j] + 1$ **then**
3 **return** *Edit-Operations*$(i-1,j) \circ$ „delete a_i“

4 **else if** $j \neq 0$ **and** $D[i,j] = D[i,j-1] + 1$ **then**
5 **return** *Edit-Operations*$(i,j-1) \circ$ „insert b_j“

6 **else** // $D[i,j] = D[i-1,j-1] + c(a_i, b_j)$
7 **if** $a_i = b_i$ **then return** *Edit-Operations*$(i-1,j-1)$
8 **else return** *Edit-Operations*$(i-1,j-1) \circ$ „replace a_i by b_j“

Initial call: *Edit-Operations*(*m*,*n*)

Edit Operations

		a	b	c	c	a
	0	1	2	3	4	5
b	1	1	1	2	3	4
a	2	1	2	2	3	3
b	3	2	1	2	3	4
d	4	3	2	2	3	4
a	5	4	3	3	3	3

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"

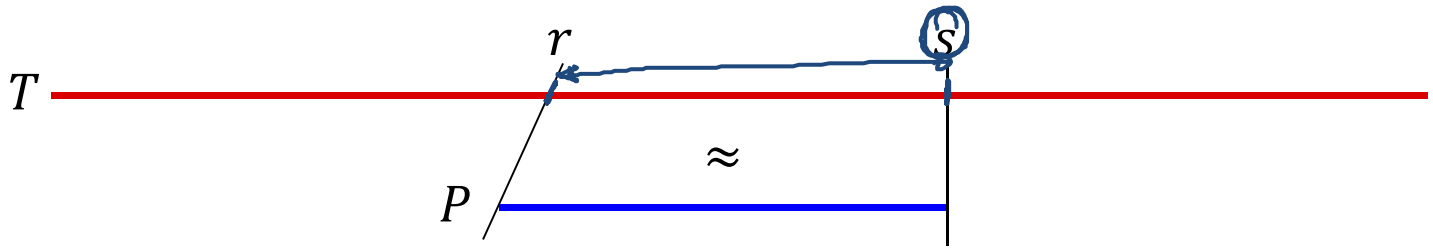
- Unit cost model:
  - interesting special case
  - each edit operation costs 1

Approximate String Matching

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

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

$$\arg\min_{1 \leq r \leq s \leq n} D(T_{r,s}, P)$$



Approximate String Matching

$|P| = m$

Naive Solution:

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

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

choose the minimum

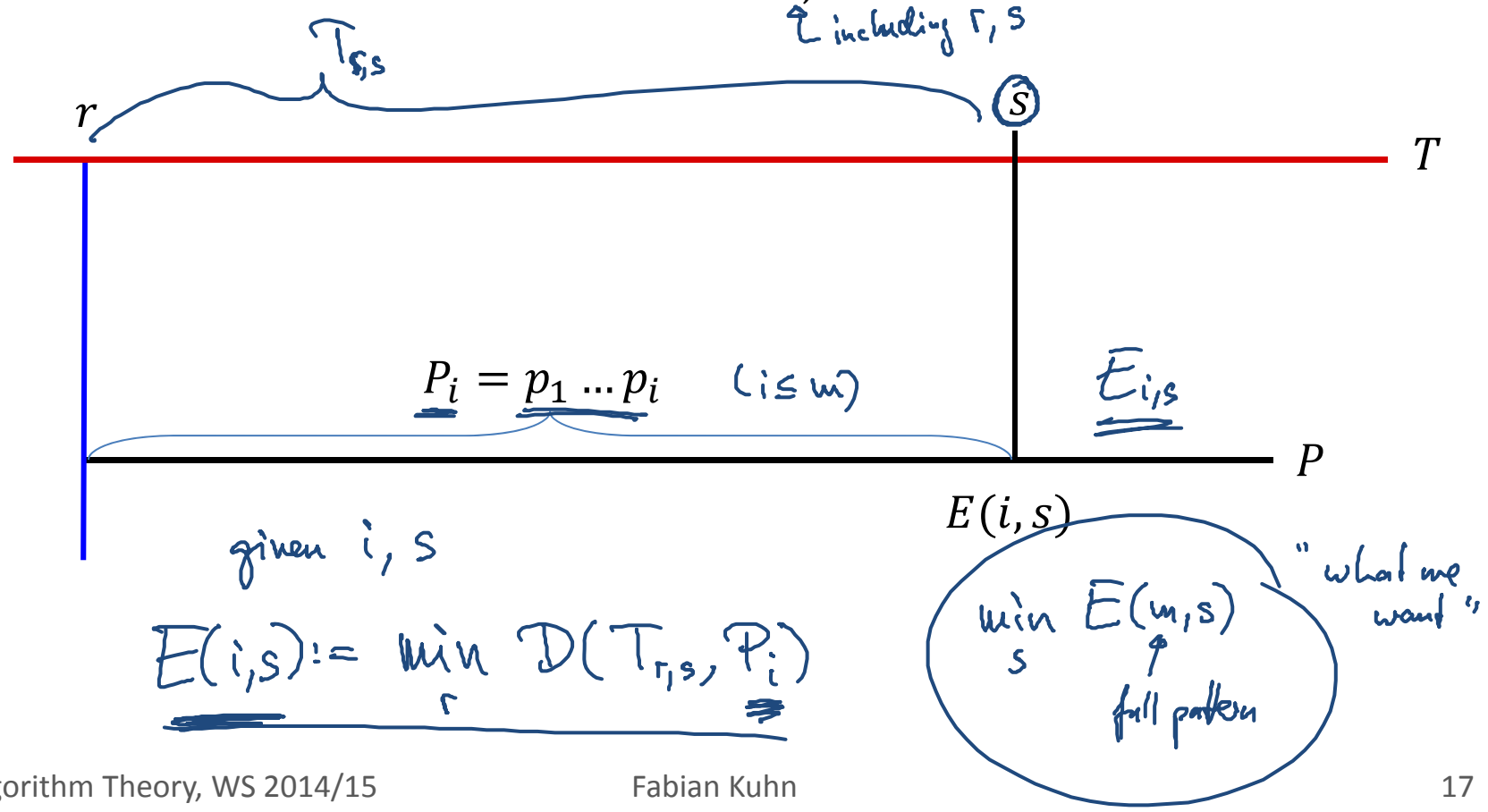
time:

$O(n^3 \cdot m)$

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.

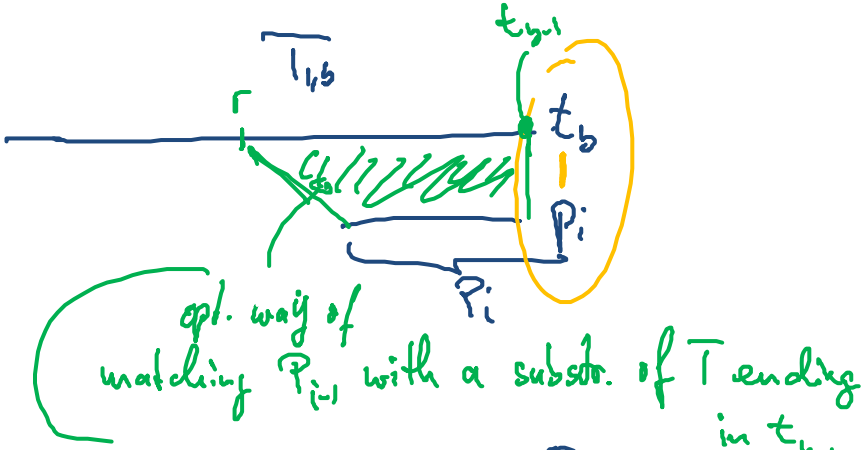


Approximate String Matching

Three ways of ending optimal alignment between T_b and P_i:

1. t_b is replaced by p_i:

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

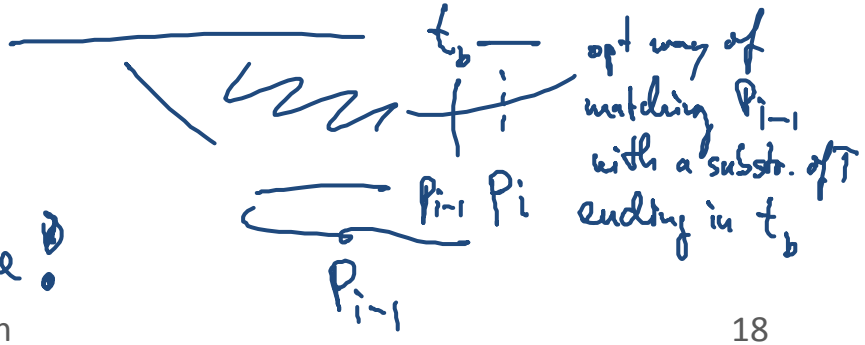


2. t_b is deleted:

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

3. p_i is inserted:

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



$D_{k,\ell} = \begin{cases} \\ \\ \end{cases}$ recursive rule looks exactly the same !

Approximate String Matching

Recurrence relation (unit cost model):

$$E_{b,i} = \min \begin{cases} E_{b-1,i-1} + 1/0 \\ E_{b-1,i} + 1 \\ E_{b,i-1} + 1 \end{cases}$$

(0 iff $t_b = p_i$)

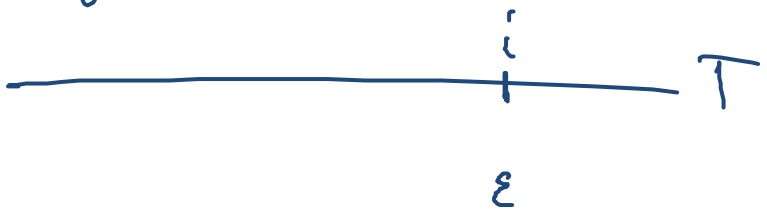
Base cases:

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

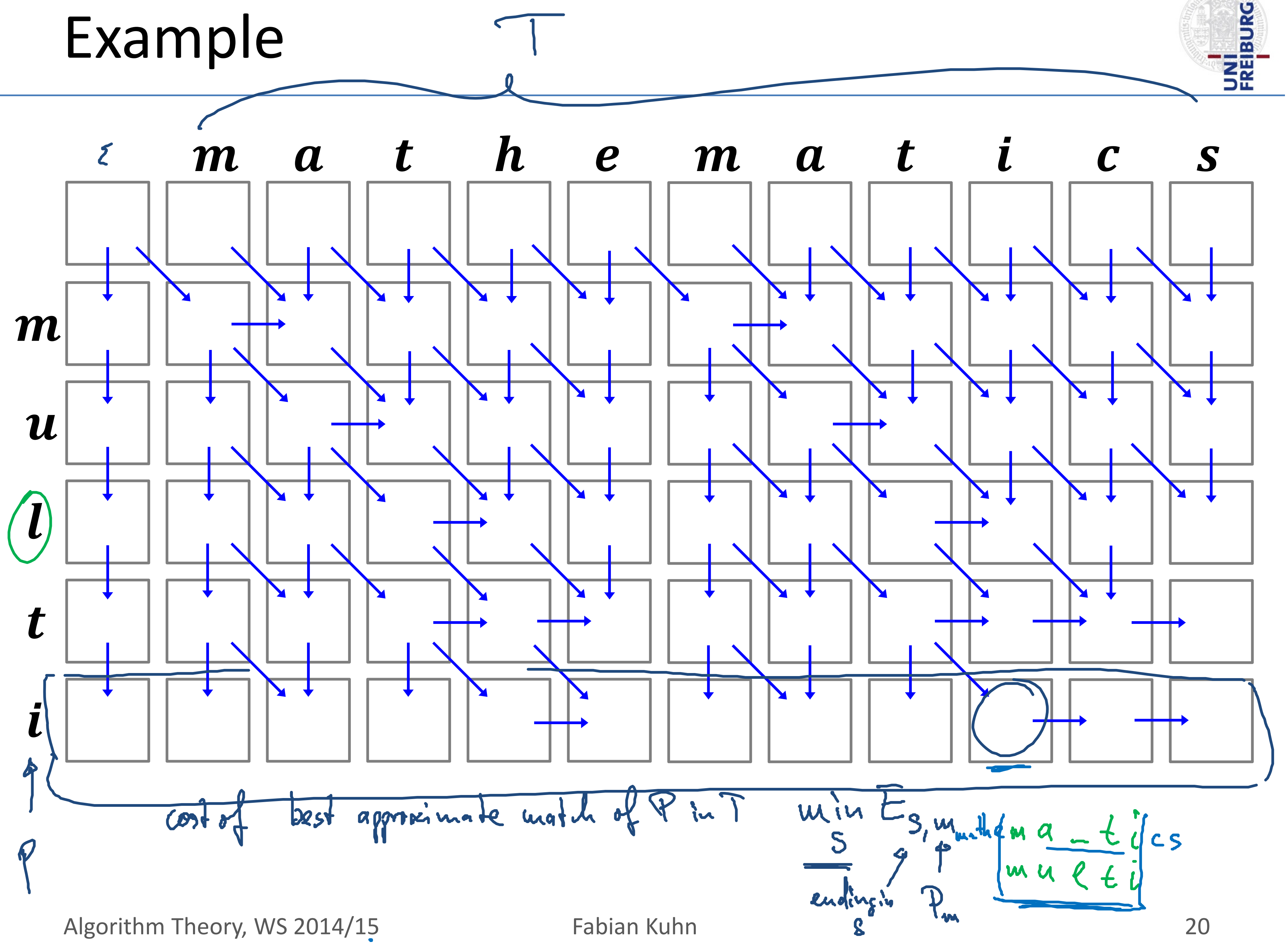
$E_{0,0}$: cost 0

$E_{0,i}$: matching P_i with a substring of T "ending" in pos 0 (empty substring)

$E_{i,0}$: matching $P_0 = \varepsilon$ with a substr. of T ending in pos i



Example


T
ε m a t h e m a t i c s
m
u
l
t
i
P
cost of best approximate match of P in T
min $E_{s,m}$
s
ending in s
P_m
m a _ t i c s
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 from 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)