12 Edit distance

Summer Term 2010

Robert Elsässer

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dynamic programming

- Algorithm design technique often used for optimization problems
- Generally usable for recursive approaches if the same partial solutions are required more than once
- Approach: store partial results in a table
- Advantage: better time complexity, often polynomial instead of exponential

Problem: similarity of strings

Edit distance

For two strings *A* and *B*, compute, as efficiently as possible, the edit distance *D(A,B)* and a minimal sequence of edit operations which transforms *A* into *B*.

i	n	f	-	-	-	o	r	m	a	t	i	k	-
i	n	t	e	r	p	o	l	-	a	t	i	o	n

Problem: similarity of strings

Approximate string matching

For a given text T, a pattern P, and a distance d,
find all substrings P' in T with $D(P,P') \leq d$

Sequence alignment

Find optimal alignments of DNA sequences

```
G A G C A - C T T G G A T T C T C G G
- - - C A C G T G G - - - - - - - - -
```

Edit distance

Given: two strings $A = a_1a_2 .... a_m$ and $B = b_1b_2 ... b_n$

Goal: find minimal cost $D(A,B)$ for a sequence of edit operations to transform A into B.

Edit operations:

1. Replace a character in A by a character from B
2. Delete a character from A
3. Insert a character from B

Edit distance

Cost model:

$$c(a,b) = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{if } a = b \end{cases}$$

$$a = \varepsilon,\ b = \varepsilon \text{ possible}$$

We assume the triangle inequality holds for *c*:

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

$\longrightarrow$ Each character is changed at most once

Edit distances

Trace as representation of edit sequences

```
A =     b a a c a a b c
B =   a b a c b c a c
```

or using indents

```
A =   - b a a c a - a b c
B =   a b a - c b c a - c
```

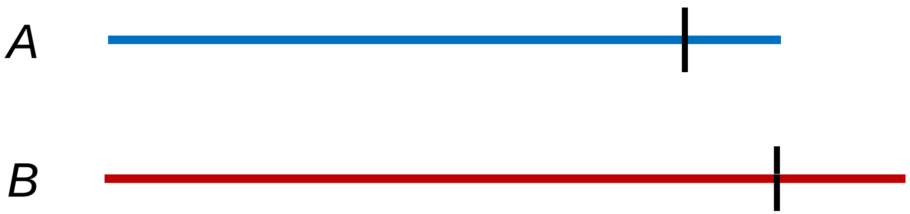
Edit distance (cost): 5

An optimal trace can be divided into two optimal subtraces
→ dynamic programming can be used

Computation of the edit distance

Let $A_i = a_1 \ldots a_i$ and $B_j = b_1 \ldots b_j$

$$D_{i,j} = D(A_i, B_j)$$



Computations of the edit distances

- Three possibilities of ending a trace:

$$a_1, \ldots, a_{m-1}, a_m$$
$$b_1, \ldots, b_{n-1}, b_n$$

- 1. a_m is replaced by b_n :
  $D_{m,n} = D_{m-1,n-1} + c(a_m, b_n)$

- 2. a_m is deleted: $D_{m,n} = D_{m-1,n} + 1$

- 3. b_n is inserted: $D_{m,n} = D_{m,n-1} + 1$

Computations of the edit distances

- Three possibilities of ending a trace:

- 1. a_m is replaced by b_n :
  $D_{m,n} = D_{m\text{-}1,n\text{-}1} + c(a_m, b_n)$

$a_1,\ldots,a_{m-1}, a_m$
$b_1,\ldots, b_{n-1}, b_n$

- 2. a_m is deleted: $D_{m,n} = D_{m-1,n} + 1$

$a_1,\ldots,a_{m-1}, a_m$
$b_1,\ldots, b_n$

- 3. b_n is inserted: $D_{m,n} = D_{m,n-1} + 1$

Computations of the edit distances

- Three possibilities of ending a trace:

- 1. a_m is replaced by b_n :
  $D_{m,n} = D_{m\text{-}1,n\text{-}1} + c(a_m, b_n)$

$a_1,\ldots,a_{m-1}, a_m$
$b_1,\ldots, b_{n-1}, b_n$

- 2. a_m is deleted: $D_{m,n} = D_{m\text{-}1,n} + 1$

$a_1,\ldots,a_{m-1}, a_m$
$b_1,\ldots, b_n$

- 3. b_n is inserted: $D_{m,n} = D_{m,n\text{-}1} + 1$

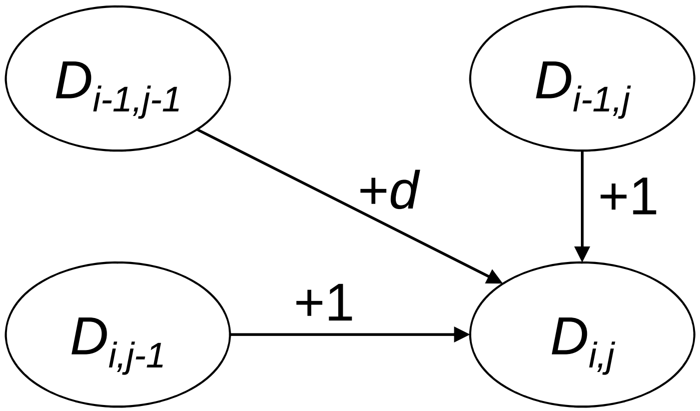
$a_1,\ldots,a_m$
$b_1,\ldots, b_{n-1}\ b_n$

Computation of the edit distance

- Recurrence relation, if $m,n \geq 1$:

$$D_{m,n} = \min \begin{Bmatrix} D_{m-1,n-1} & + & c(a_m, b_n), \\ D_{m-1,n} & + & 1, \\ D_{m,n-1} & + & 1 \end{Bmatrix}$$

- $\longrightarrow$ Computation of all $D_{i,j}$ is required, $0 \leq i \leq m$, $0 \leq j \leq n$.



Recurrence relation for the edit distance

Base cases:

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

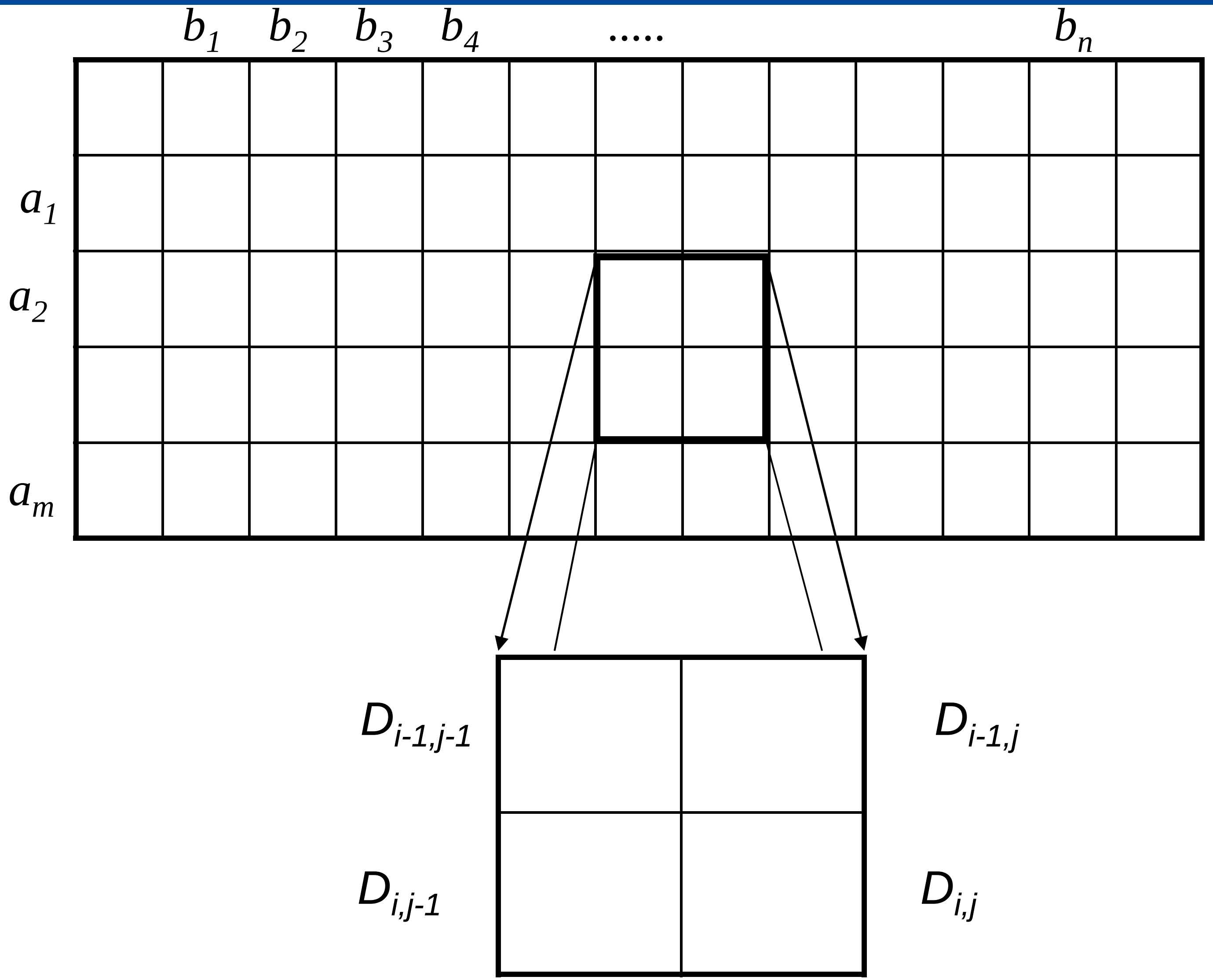
$$D_{0,j} = \mathrm{D}(\varepsilon, B_j) = j$$

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

Recurrence equation:

$$D_{i,j} = \min \left\{ \begin{array}{ccc} D_{i-1,j-1} & + & c(a_i, b_j) \\ D_{i-1,j} & + & 1, \\ D_{i,j-1} & + & 1 \end{array} \right\}$$

Order of computation for the edit distance



Algorithm for the edit distance

Algorithm edit_distance

Input: two strings $A = a_1 \ldots. a_m$ and $B = b_1 \ldots b_n$

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

```
D[0,0] := 0
for i := 1 to m do D[i,0] = i
for j := 1 to n do D[0,j] = j
for i := 1 to m do
    for j := 1 to n do
        D[i,j] := min( D[i - 1,j] + 1,
                       D[i,j - 1] + 1,
                       D[i –1, j – 1] + c(a_i,b_j))
```

Example

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

Example

		j	0	1	2	3	4
				a	b	a	c
i							
0			0	1	2	3	4
1	b		1				
2	a		2				
3	a		3				
4	c		4				

Example

	j	0	1	2	3	4
i			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}				
2	a	2_{del}				
3	a	3_{del}				
4	c	4_{del}				

Example

	j	0	1	2	3	4
			a	b	a	c
i						
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1			
2	a	2_{del}				
3	a	3_{del}				
4	c	4_{del}				

Example

			j: 0	1	2	3	4
				a	b	a	c
i							
0			0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b		1_{del}	1			
2	a		2_{del}				
3	a		3_{del}				
4	c		4_{del}				

D(ε , 2) +1 = 3

D(1 , 1) +1 = 2

D(ε , 1) +0 = 1

Example

j		0	1	2	3	4
i			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1			
2	a	2_{del}				
3	a	3_{del}				
4	c	4_{del}				

$D(\varepsilon, 2) + 1 = 3$

$D(1, 1) + 1 = 2$

$D(\varepsilon, 1) + 0 = 1$

Example

i \ j		0	1	2	3	4
			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1	1		
2	a	2_{del}				
3	a	3_{del}				
4	c	4_{del}				

Example

i \ j		0	1	2	3	4
			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1	1	2	3
2	a	2_{del}	1			
3	a	3_{del}	2			
4	c	4_{del}	3			

Example

i \ *j*		0	1	2	3	4
			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1	1	2	3
2	a	2_{del}	1	2		
3	a	3_{del}	2			
4	c	4_{del}	3			

Example

j		0	1	2	3	4
i			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1	1	2	3
2	a	2_{del}	1	2	1	2
3	a	3_{del}	2	2		
4	c	4_{del}	3	3		

Example

	j	0	1	2	3	4
i			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1	1	2	3
2	a	2_{del}	1	2	1	2
3	a	3_{del}	2	2	2	2
4	c	4_{del}	3	3	3	

Example

		j: 0	1	2	3	4
i			a	b	a	c
0		0	1_{ins}	2_{ins}	3_{ins}	4_{ins}
1	b	1_{del}	1	1	2	3
2	a	2_{del}	1	2	1	2
3	a	3_{del}	2	2	2	2
4	c	4_{del}	3	3	3	2

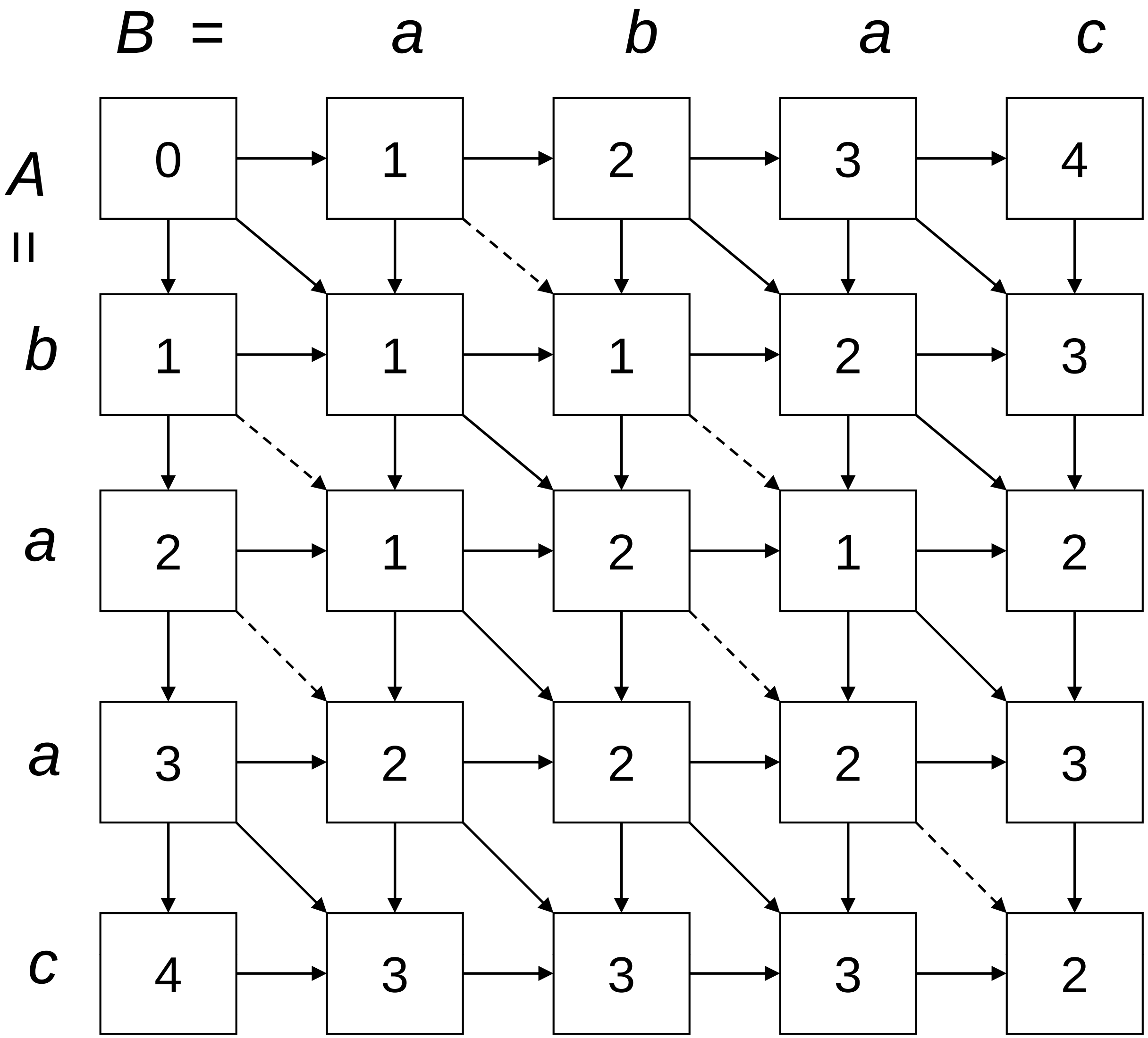
Computation of the edit operations

Algorithm edit_operations (*i,j*)
Input: matrix *D* (computed)

```
if i = 0 and j = 0 then return
if i ≠ 0 and D[i,j] = D[i – 1 , j] + 1
  then „delete a[i]“
           edit_operations (i – 1, j)
else if j ≠ 0 and D[i,j] = D[i, j – 1] + 1
   then „insert b[j]“
            edit_operations (i, j – 1)
else
  /* D[i,j] = D[i – 1, j – 1 ] + c(a[i], b[j])  */
         „replace a[i] by b[j] “
         edit_operations (i – 1, j – 1)
```

Initial call: edit_operations(*m,n*)

Trace graph of the edit operations



Sub-graph of the edit operations

Trace graph: All possible traces which transform A into B, directed edges from vertex (i, j) to ($i + 1, j$), ($i, j + 1$) and ($i + 1, j + 1$).

Weights of the edges represent the edit costs.

Costs are monotonically increasing along an optimal path.

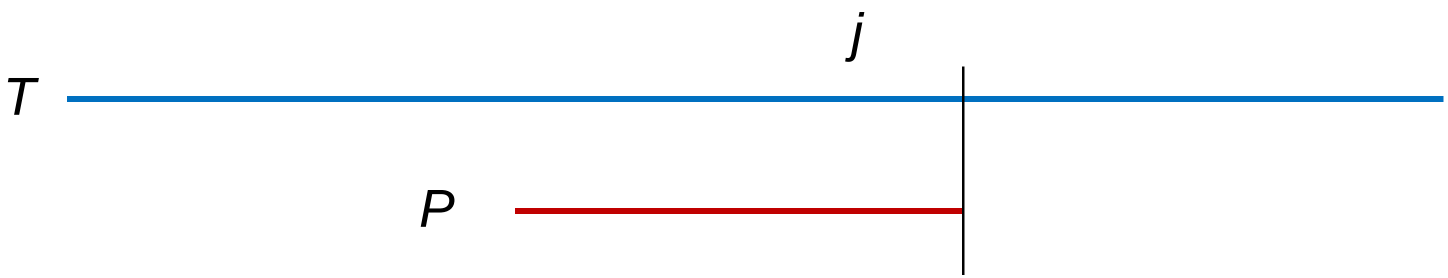
Each path from the upper left corner to the lower right corner represents an optimal trace.

Approximate string matching

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

- **Goal:** an interval $[j', j]$, $1 \le j' \le j \le n$, such that the substring
  $T_{j',j} = t_{j'} \ldots t_j$ of T is most similar to pattern P,
  i.e. for all other intervals $[k', k]$, $1 \le k' \le k \le n$:

$$D(P, T_{j',j}) \le D(P, T_{k',k})$$



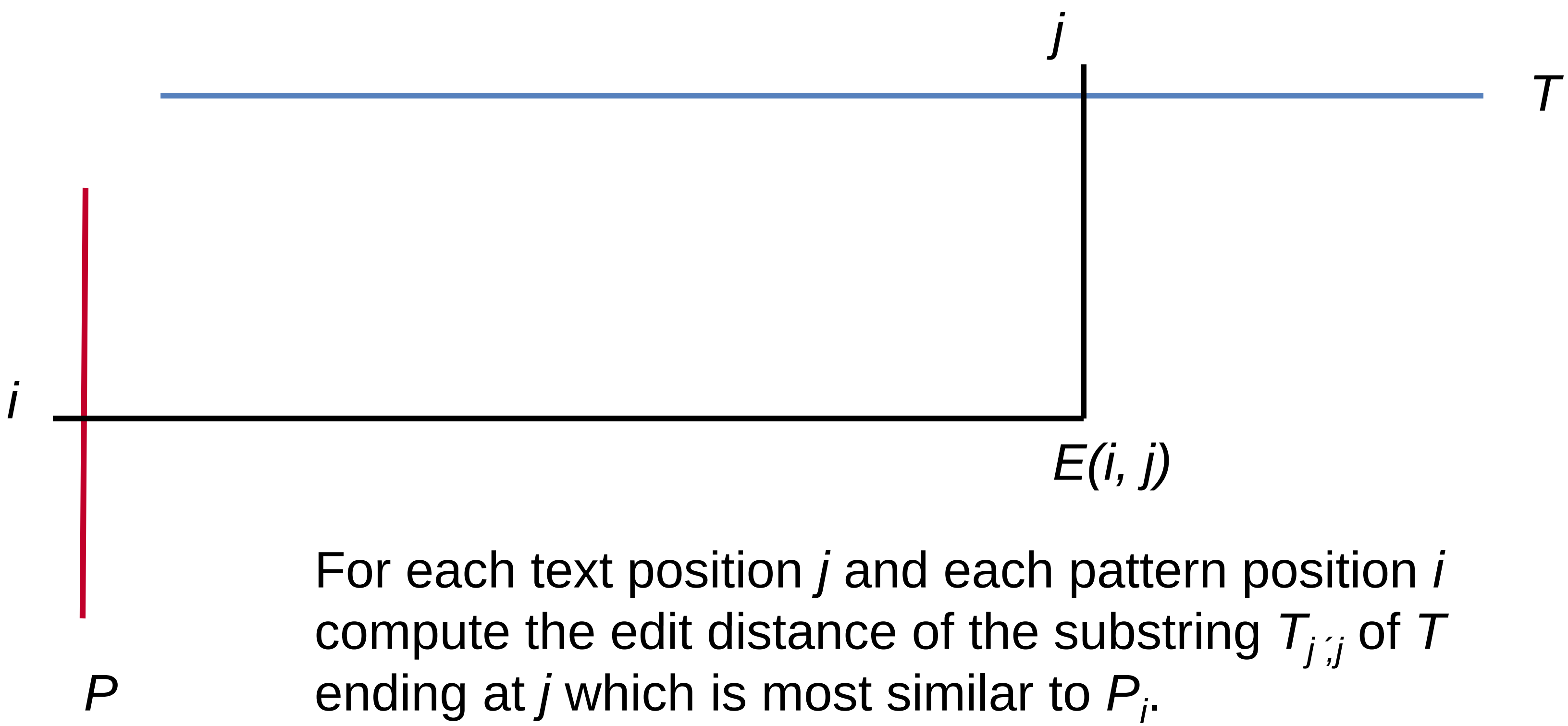
Approximate string matching

Naïve approach:

> **for all** $1 \le j' \le j \le n$ **do**
>
> compute $D(P, T_{j',j})$
>
> choose minimum

Approximate string matching

Consider a related problem:



For each text position j and each pattern position i compute the edit distance of the substring $T_{j',j}$ of T ending at j which is most similar to P_i.

Approximate string matching

Method:

for all $1 \leq j \leq n$ ***do***

compute j' such that $D(P, T_{j',j})$ is minimal

For $1 \leq i \leq m$ and $0 \leq j \leq n$ let:

$$E_{i,j} = \min_{1 \leq j' \leq j+1} D(P_i, T_{j',j})$$

Optimal trace:

$P_i =$ b a a c a a b c

$T_{j',j} =$ b a c b c a c

Approximate string matching

Recurrence relation:

$$E_{i,j} = \min \begin{Bmatrix} E_{i-1,j-1} + c(p_i, t_j), \\ E_{i-1,j} + 1, \\ E_{i,j-1} + 1 \end{Bmatrix}$$

Remark:

j´ can be completely different for $E_{i-1, j-1}$, $E_{i-1,j}$ and $E_{i, j-1}$.

A subtrace of an optimal trace is an optimal subtrace.

Approximate string matching

Base cases:

$$E_{0,0} = E(\varepsilon, \varepsilon) = 0$$
$$E_{i,0} = \mathrm{E}(P_j, \varepsilon) = i$$

but

$$E_{0,j} = E(\varepsilon, T_j) = 0$$

Observation:

The optimal edit sequence from P to $T_{j',j}$ does not start with an insertion of $t_{j'}$.

Approximate string matching

Dependency graph

	T =	a	b	b	d	a	d	c	b	c
P =	0	0	0	0	0	0	0	0	0	0
a	1	0	1	1	1	0	1	1	1	1
d	2	1	1	2	1	1	0	1	2	2
b	3	2	1	1	2	2	1	1	1	2
b	4	3	2	1	2	3	2	2	1	2
c	5	4	3	2	2	3	3	2	2	1

Approximate string matching

Theorem

If there is a path from $E_{0,\, j'-1}$ to $E_{i,\, j}$ in the dependency graph, then $T_{j',\, j}$ is a substring of T ending in j which is most similar to P_i and

$$D(P_i, T_{j',j}) = E_{i,\, j}$$