



MASTER IN  
COMPUTER  
SCIENCE

UNIVERSITÉ DE FRIBOURG  
UNIVERSITÄT FREIBURG

# Pattern Recognition

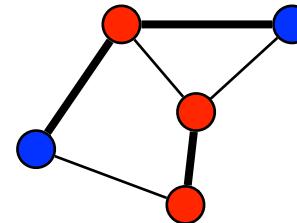
## Lecture 7 : String Matching I

Dr. Andreas Fischer  
[andreas.fischer@unifr.ch](mailto:andreas.fischer@unifr.ch)

# Statistical vs Structural Pattern Recognition

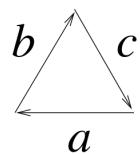
- Statistical pattern recognition
  - + rich mathematical structure, low computational complexity
  - fixed-size representation, no relation modeling
- Structural pattern recognition
  - + flexible representation, binary relations
  - lack of mathematical structure, high computational complexity

$(x_1, \dots, x_n)$

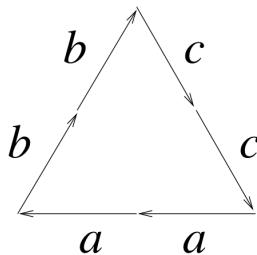


# String Representation

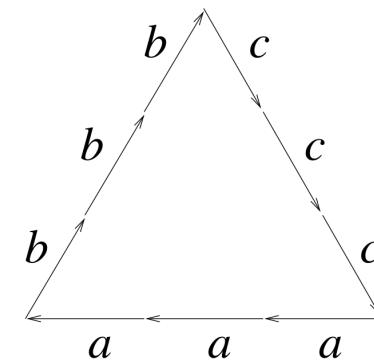
## Example: Shapes



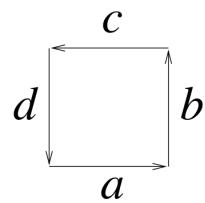
$abc$



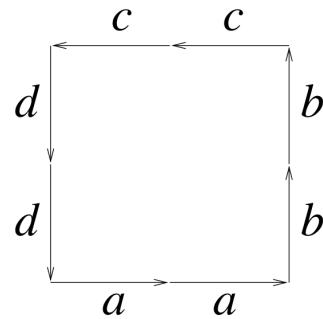
$aabbcc$



$aaabbccccc$



$abcd$

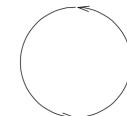


$aabbccdd$

⋮

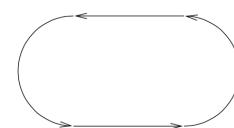
# Example: Shapes

*a*



*cd*

*b*



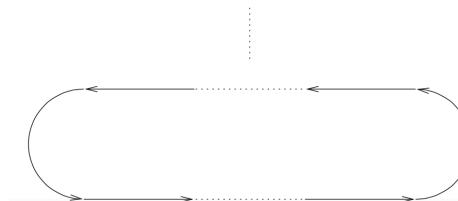
*cadb*

*c*



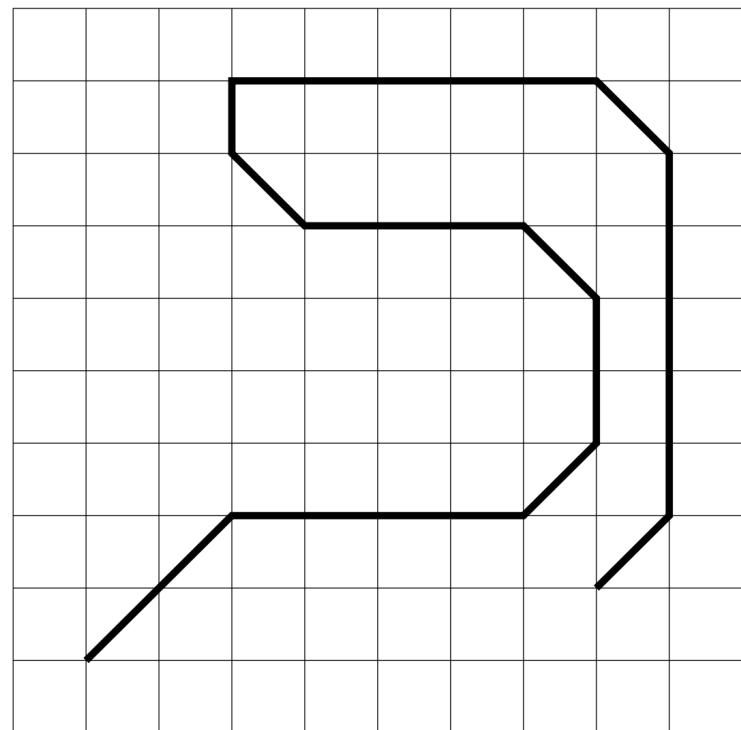
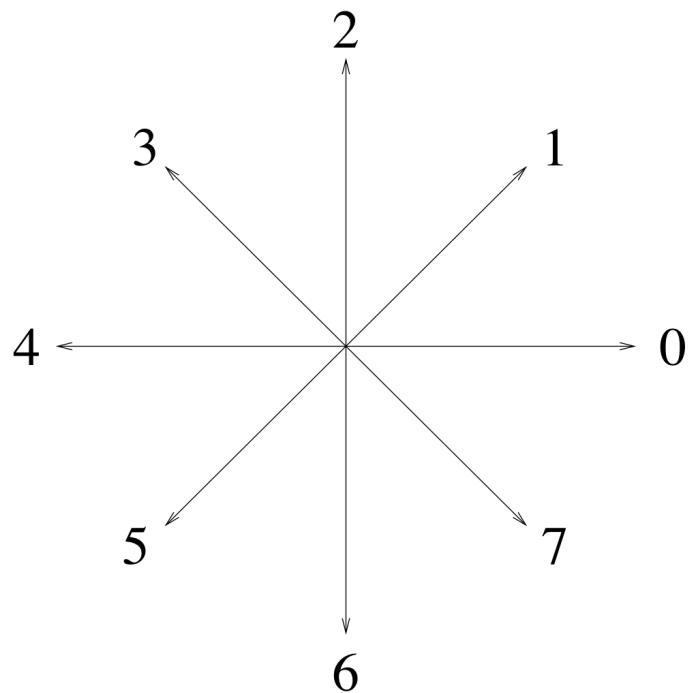
*caadbb* or *ca<sup>2</sup>db<sup>2</sup>*

*d*



*ca<sup>n</sup> db<sup>n</sup>*

## Example: Shapes



110000122344432000007666665

# String

- An *alphabet*  $A = \{a, b, \dots\}$  is a finite set of symbols.
- A *string* or *word*  $x = x_1 x_2 \dots x_n$  is a sequence of symbols.

$$x_i \in A, 1 \leq i \leq n$$

- *length* :  $|x| = n$ 
  - *empty string*  $\varepsilon : |\varepsilon| = 0$
- *concatenation* :  $z = xy = x_1 \dots x_n y_1 \dots y_m$ 
  - $a \dots a = a^i$
  - $a\varepsilon = \varepsilon a = a$
- *substring* :  $x_i x_{i+1} \dots x_{i+m}$ 
  - pattern  $\rightarrow$  pat, att, tte, ter, ern
- *subsequence* :  $x_{i_1} x_{i_2} \dots x_{i_m}$  with  $i_1 < i_2 < \dots < i_m$  for  $j < k$ 
  - pattern  $\rightarrow$  ptrn, at, tn, ...
- *prefix* :  $x_1 x_2 \dots x_m$ 
  - pattern  $\rightarrow$  p, pa, pat, ...
- *suffix* :  $x_{n-m} x_{n-m+1} \dots x_n$ 
  - pattern  $\rightarrow$  n, rn, ern, ...

# Formal Language

- $A = \{a, b, c\}$
- $A^n$  : words of length  $n$ 
  - $A^0 = \{\epsilon\}$
  - $A^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
- $A^*$  : all words over the alphabet  $A$

$$A^* = \bigcup_{i=0}^{\infty} A^i$$

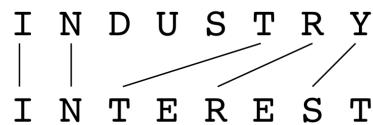
- $A^* = \{\epsilon, a, b, c, aa, ab, \dots\}$
- $A^+ = A^* \setminus \{\epsilon\} = \{a, b, c, aa, ab, \dots\}$
- A *formal language* is a subset of  $A^*$ 
  - $L = \{aaabb, ba, aab\}$
  - $L = a^n b^n c^n$

# **String Edit Distance**

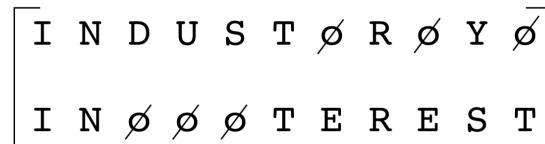
# Edit Operations

- Three canonical edit operations:
  - Substitution ( $a \rightarrow b$ )
  - Deletion ( $a \rightarrow \epsilon$ )
  - Insertion ( $\epsilon \rightarrow b$ )

**Trace**



**Alignment  
or  
Matching**



INDUSTRY

Delete D

INUSTRY

Delete U

INSTRY

Substitute Y by S

INSTRS

Insert E

INSTERS

Insert E

INSTERES

Delete S

INTERES

Insert T

INTEREST

# String Edit Distance (SED)

- Assign costs to all edit operations  $c(a \rightarrow b)$ ,  $c(a \rightarrow \epsilon)$ ,  $c(\epsilon \rightarrow b)$ .
- Edit operations correspond with pattern distortions. Large, improbable distortions should have high costs.
- Let  $(e_1, \dots, e_n)$  be an *edit path* that transforms string  $x$  into  $y$ . Furthermore,  $E$  is the set of all valid edit paths.
- The *string edit distance* or *Levenshtein distance*  $d(x,y)$  is then defined as the minimum cost for transforming  $x$  into  $y$ :

$$d(x, y) = \min_{(e_1, \dots, e_n) \in E} \sum_{i=1}^n c(e_i)$$

# SED Properties

- Lemma. The string edit distance is a metric if

$$c(a \rightarrow b) \geq 0 \quad \forall a, b \in A \cup \varepsilon$$

$$c(a \rightarrow b) = 0 \Leftrightarrow a = b \quad \forall a, b \in A$$

$$c(a \rightarrow c) \leq c(a \rightarrow b) + c(b \rightarrow c) \quad \forall a, b, c \in A \cup \varepsilon$$

$$c(a \rightarrow b) = c(b \rightarrow a) \quad \forall a, b \in A \cup \varepsilon$$

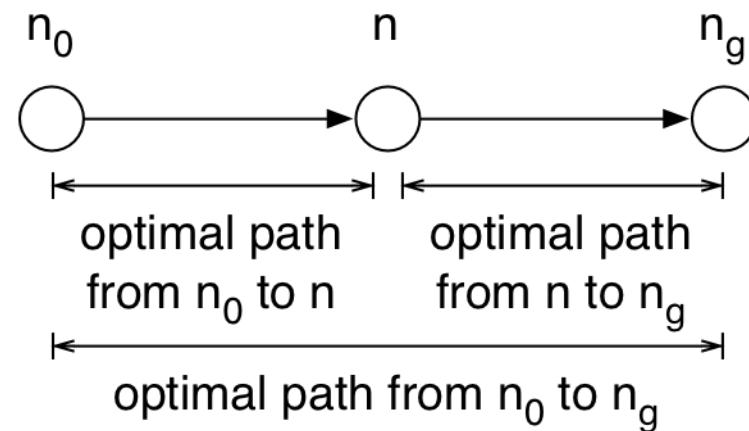
- In the following, we consider the weaker conditions

$$c(a \rightarrow b) \geq 0 \quad \forall a, b \in A \cup \varepsilon$$

$$c(a \rightarrow a) = 0 \quad \forall a \in A$$

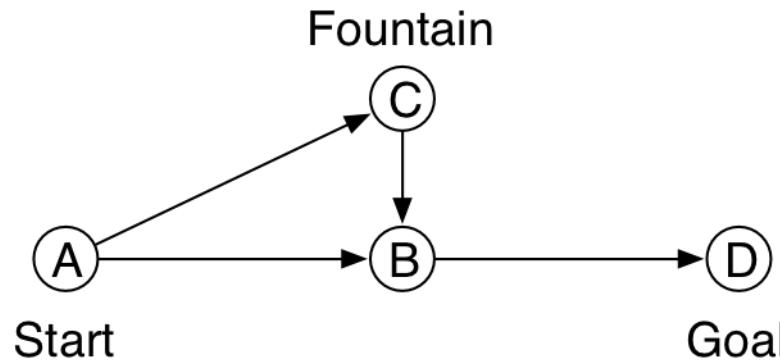
# Dynamic Programming (DP)

- SED efficiently computed by means of dynamic programming.
- DP is a method to find the optimal path for a certain class of graph search problems.
  - Necessary and sufficient condition for DP: If the optimal path from the start node  $n_0$  to a goal node  $n_g$  passes through node  $n$ , then the optimal path consists of the best path from  $n_0$  to  $n$ , followed by the best path from  $n$  to  $n_g$  (divide and conquer).



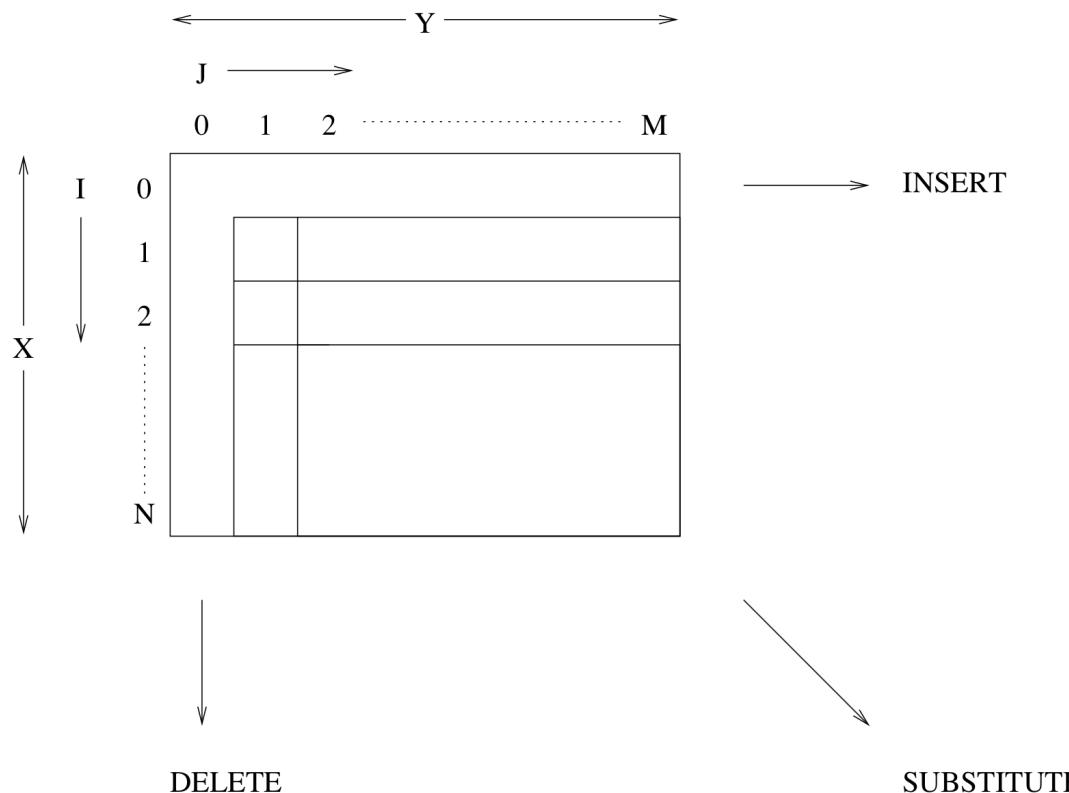
## Counterexample (Not DP)

- Consider a trip from A (Start) to D (Goal). Enough water is available to reach B. However, in order to reach D it is necessary to replenish water at C (Fountain).
  - Best path from A to B is direct without visiting C.
  - However, the best and only possible path from A to D goes over C.



# SED Computation

- Divide the string matching problem into prefix matching subproblems.
- Dynamic programming scheme for computation of  $d(x,y)$  between strings  $x = x_1x_2\dots x_N$  and  $y = y_1y_2\dots y_M$  :



# SED Algorithm

- Time-complexity and space-complexity:  $O(nm)$

Input:  $x = x_1 \dots x_n, y = y_1 \dots y_m, c(a \rightarrow b)$

Output: cost matrix  $D(i, j)$  and pointers

```
1.    $D(0, 0) := 0;$ 
2.   for  $i = 1$  to  $n$  do  $D(i, 0) := D(i - 1, 0) + c(x(i) \rightarrow \varepsilon);$ 
3.   for  $j = 1$  to  $m$  do  $D(0, j) := D(0, j - 1) + c(\varepsilon \rightarrow y(j));$ 
4.   for  $i = 1$  to  $n$  do
5.     for  $j = 1$  to  $m$  do
6.       begin
7.          $m_1 := D(i - 1, j - 1) + c(x(i) \rightarrow y(j));$ 
8.          $m_2 := D(i - 1, j) + c(x(i) \rightarrow \varepsilon);$ 
9.          $m_3 := D(i, j - 1) + c(\varepsilon \rightarrow y(j));$ 
10.         $D(i, j) = \min(m_1, m_2, m_3);$ 
11.        if  $m_1 = D(i, j)$  then  $\text{pointer}(i, j) := (i - 1, j - 1)$ 
12.                      else if  $m_2 = D(i, j)$  then  $\text{pointer}(i, j) := (i - 1, j)$ 
13.                      else  $\text{pointer}(i, j) := (i, j - 1);$ 
14.      end;
```

## SED Example

$$c(A \rightarrow \varepsilon) = c(B \rightarrow \varepsilon) = c(\varepsilon \rightarrow A) = c(\varepsilon \rightarrow B) = 1$$

$$c(A \rightarrow B) = c(B \rightarrow A) = 2 \quad A \neq B$$

	B	A	B	A	A	A
0	1	2	3	4	5	6
A	1	2	1	2	3	4
B	2	1	2	1	2	3
A	3	2	1	2	1	2
B	4	3	2	1	2	3
B	5	4	3	2	3	4
B	6	5	4	3	4	5

## SED Example

A B A B B B  
B A B A A A

	$\epsilon$	B	A	B	A	A	A
$\epsilon$	0	1	2	3	4	5	6
A	1	2	1	2	3	4	5
B	2	1	2	1	2	3	4
A	3	2	1	2	1	2	3
B	4	3	2	1	2	3	4
B	5	4	3	2	3	4	5
B	6	5	4	3	4	5	6

## SED Variants

- Block substitution. Substitute not only one character but instead ( $u \rightarrow v$ ) with  $u = u_1 \dots u_k$  and  $v = v_1 \dots v_l$ :

$$D(i, j) = \min_{u \rightarrow v} \{ D(i - k, j - l) + c(u \rightarrow v) \}$$

- Context-dependent costs:

$$c(a \rightarrow b : a\underline{aa}) \neq c(a \rightarrow b : b\underline{ab})$$

- *Needleman–Wunsch* algorithm for aligning DNA sequences with alphabet  $A = \{G, A, C, T\}$ .
  - Find the maximum similarity  $S(x, y)$  with  $s(a \rightarrow a) = 1$  (“match”),  $s(a \rightarrow b) = -1$  (“mismatch”),  $s(a \rightarrow \epsilon) = s(\epsilon \rightarrow a) = -1$  (“gap”).
  - Other similarity values possible similar to the costs of SED.
  - Equivalent to finding the minimum dissimilarity with SED.

# Longest Common Subsequence (LCS)

- The SED algorithm can be used to determine the *longest common subsequence*  $\text{lcs}(x,y)$  using the following cost function:
  - $c(a \rightarrow b) = 2$  if  $a \neq b$
  - $c(a \rightarrow \epsilon) = c(\epsilon \rightarrow b) = 1$
  - $c(a \rightarrow a) = 0$
- Intuitive explanation: based on this cost function, the optimal edit path can be found without substitutions ( $a \rightarrow b$ ) where  $a \neq b$ . The algorithm will search for the maximum number of diagonal steps ( $a \rightarrow a$ ) with zero cost. These diagonal steps correspond with the longest common subsequence. Therefore,

$$d(x,y) = |x| + |y| - 2|\text{lcs}(x,y)|$$

$$|\text{lcs}(x,y)| = \frac{|x| + |y| - d(x,y)}{2}$$

# LCS Similarity

- Two similarity measures can be defined considering

$$0 \leq |lcs(x, y)| \leq \min(|x|, |y|)$$

- $S_1(x, y) = 1$  if  $x$  is a subsequence of  $y$  or *vice versa*

$$S_1(x, y) = \frac{|lcs(x, y)|}{\min(|x|, |y|)}; 0 \leq S_1(x, y) \leq 1$$

- $S_2(x, y) = 1$  if  $x = y$

$$S_2(x, y) = \frac{2|lcs(x, y)|}{|x| + |y|}; 0 \leq S_2(x, y) \leq 1$$

# String Classification

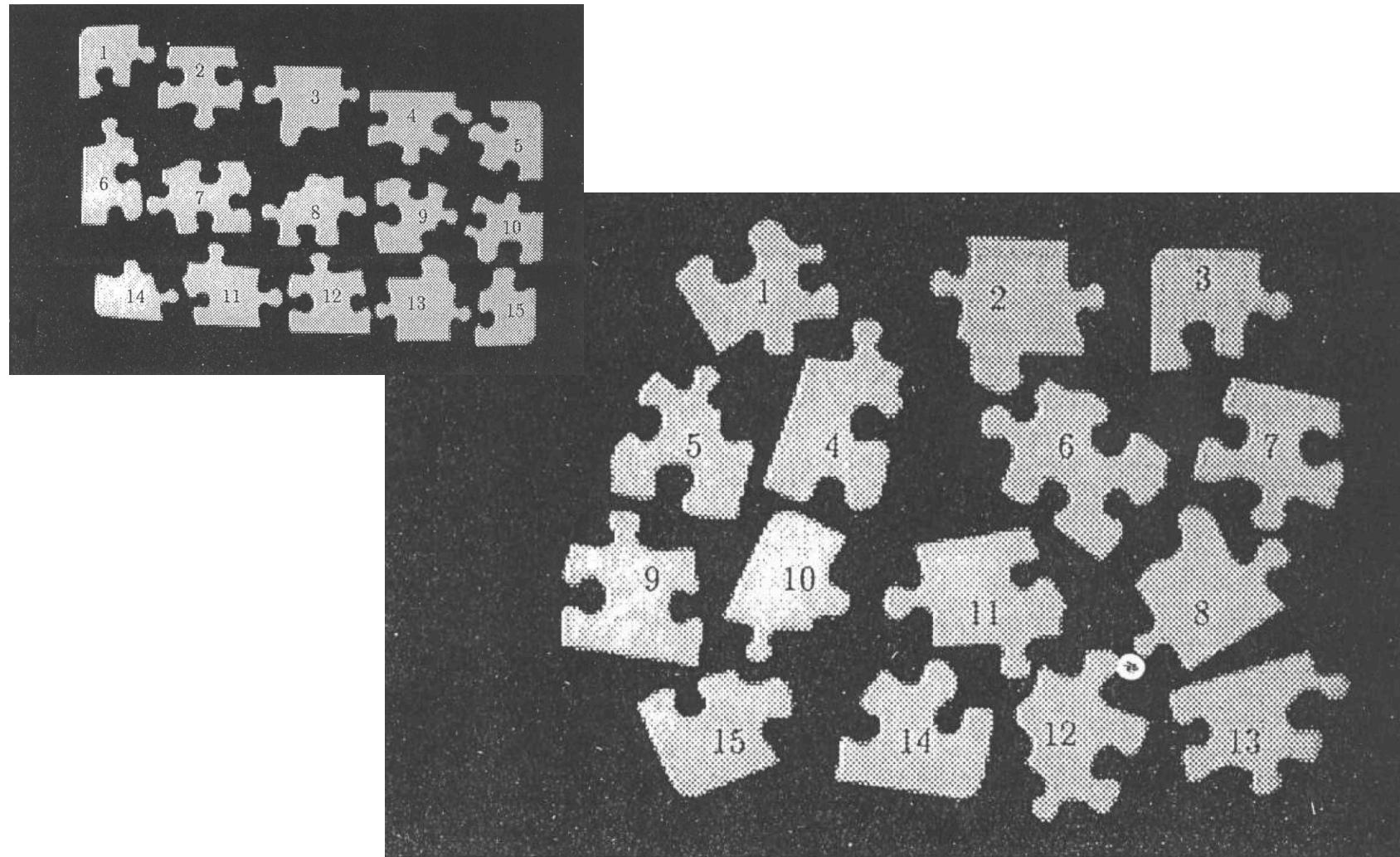
# Nearest Neighbor Classification

- NN classification based on string edit distance:
  - $X = A^*$
  - $\theta = \{ \text{learning samples} \}$
  - $f_\theta$  : assign the class label of the most similar known sample

$$f_\theta(x) = C_i \Leftrightarrow \operatorname{argmin}_{p \in \theta} d(x, p) \in C_i$$

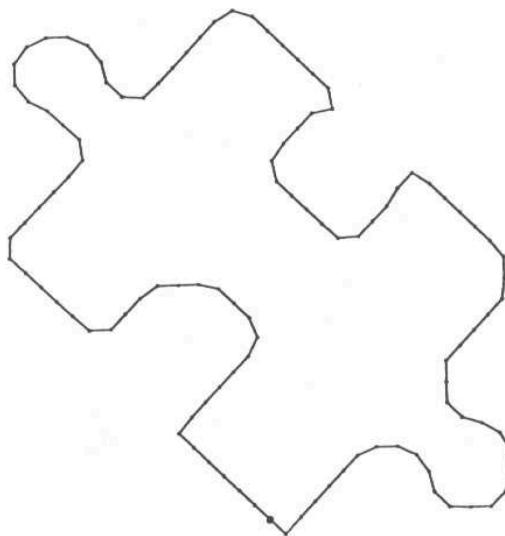
## Example: Shape Matching

- Unknown shapes are rotated and scaled versions of known shapes.



## Example: Shape Matching

- Represent shapes with a string of equidistance contour points, labeled with their angle (0 if continuing in the same direction,  $\neq 0$  for turns).

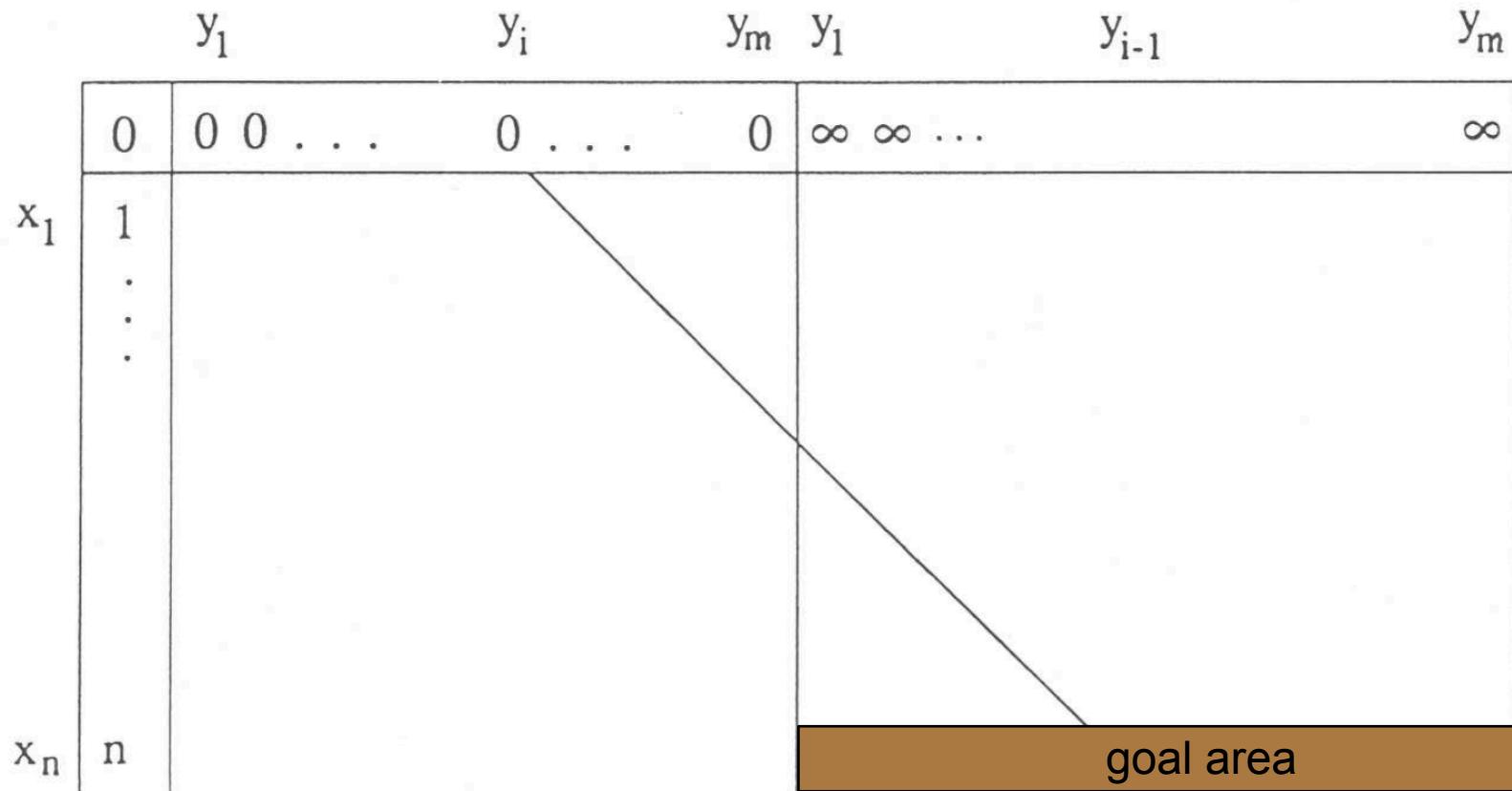


0	0	0	0	0	0	-86	-4	0	0
0	17	49	24	0	32	13	0	34	11
0	-44	-45	-1	0	0	0	-48	-42	0
0	0	5	46	39	0	16	-24	-37	-37
-39	-24	-25	-30	-22	32	41	40	3	0
0	0	-15	-48	-27	0	0	0	0	-36
-89	34	1	9	47	34	0	0	2	45
43	0	12	-12	-78	-12	0	0	0	-6
-39	-8	-37	0	0	0	44	1	44	28
-13	-30	-29	-19	-28	-43	-3	-42	-31	20
30	26	23	22	0	0	0	0	-90	

## Example: Shape Matching

- Rotation-invariance: take into account all possible starting points.
- Scale-invariance: use block substitution to compute angular sum.

$$c(u \rightarrow v) = \sum_{i=1}^k u_i - \sum_{i=1}^l v_l$$



# Dynamic Time Warping

# Dynamic Time Warping (DTW)

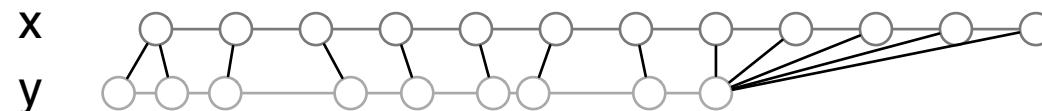
- The SED algorithm can be used to compute a dissimilarity  $d(x,y)$  between two feature vector sequences

$$x = x_1, \dots, x_N; x_i \in R^n$$

$$y = y_1, \dots, y_M; y_i \in R^n$$

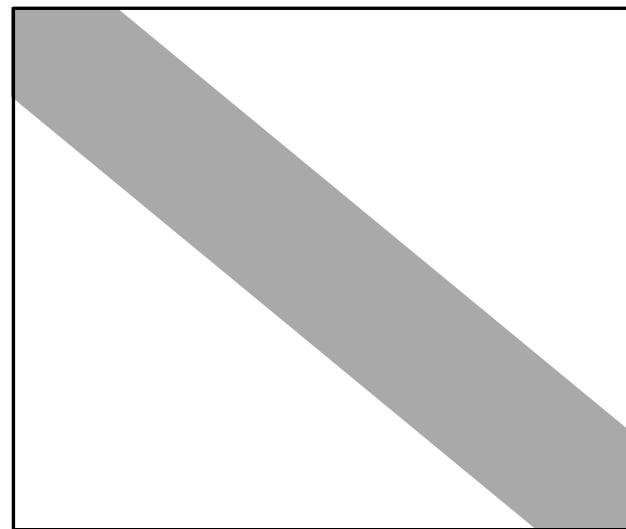
- Dynamic time warping aligns two sequences along a common time axis by considering only substitutions ( $x_i \rightarrow y_j$ ), usually with Euclidean cost:

$$c(x_i \rightarrow y_j) = \|x_i - y_j\| = \sqrt{\sum_{k=1}^n (x_{i,k} - y_{j,k})^2}$$



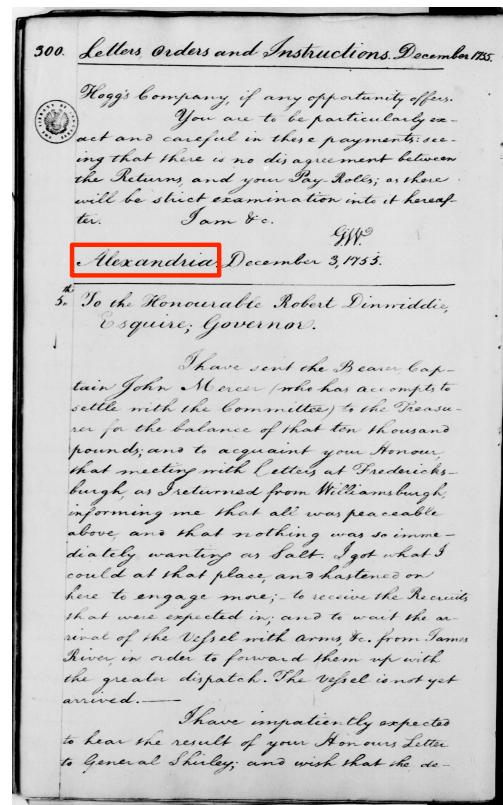
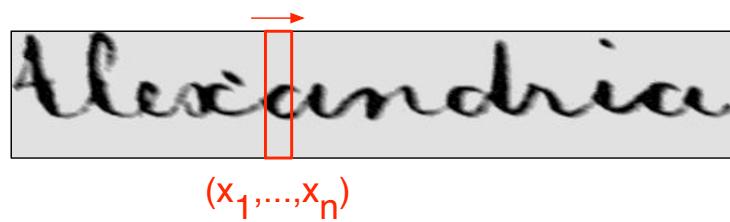
# Sakoe-Chiba Band

- Only consider edit paths close to the diagonal during DP.
- Suboptimal solution:
  - Excludes abnormal edit paths.
  - Speeds up the computation.



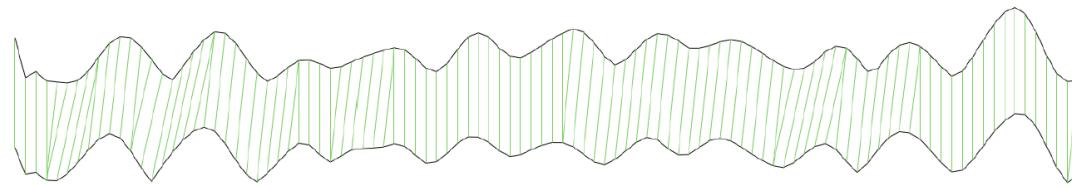
# Example: Keyword Spotting

- Describe words as feature vector sequences with a sliding window.
- Retrieve words similar to a query word from the document.

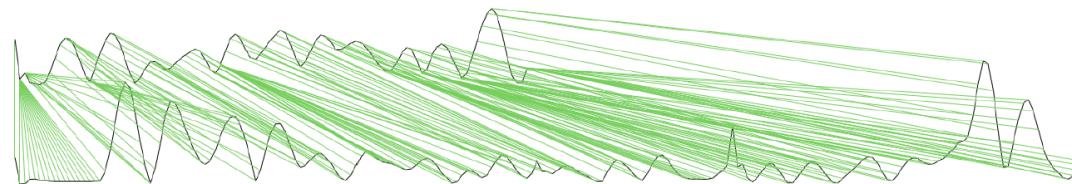


# Example: Signature Verificaiton

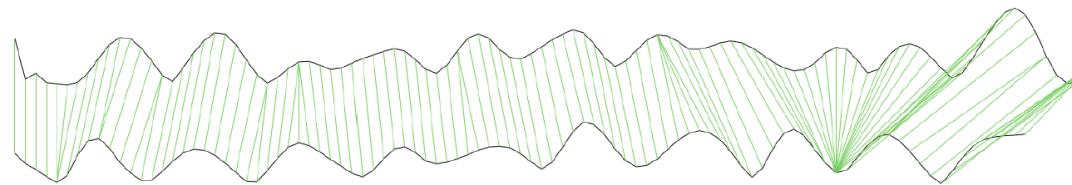
- Match the velocity  $x_i = (v_x, v_y) \in \mathbb{R}^2$  between two pen-tip trajectories recorded with an electronic device (tablet, smartphone, ...).



(a) genuine signature



(b) random forgery



(c) skilled forgery