UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

MASTER IN COMPUTER SCIENCE

Pattern Recognition

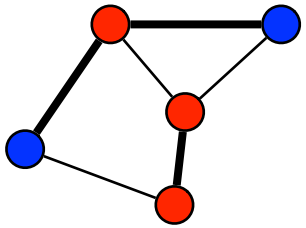
Lecture 7 : String Matching I

Dr. Andreas Fischer
andreas.fischer@unifr.ch

Statistical vs Structural Pattern Recognition

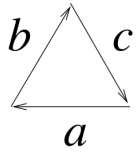
- Statistical pattern recognition
  + rich mathematical structure, low computational complexity
  - fixed-size representation, no relation modeling
- Structural pattern recognition
  + flexible representation, binary relations
  - lack of mathematical structure, high computational complexity

$(x_1,\ldots,x_n)$

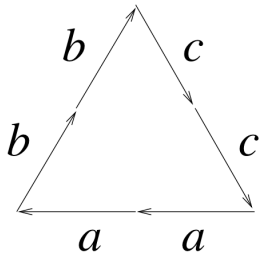


String Representation

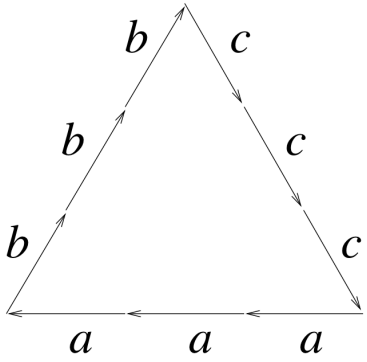
Example: Shapes



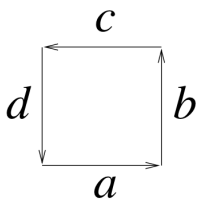
abc



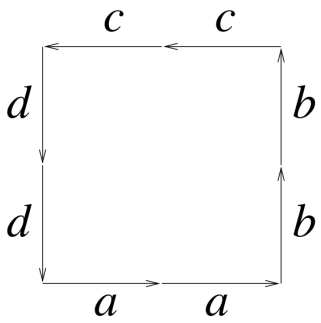
aabbcc



aaabbbccc



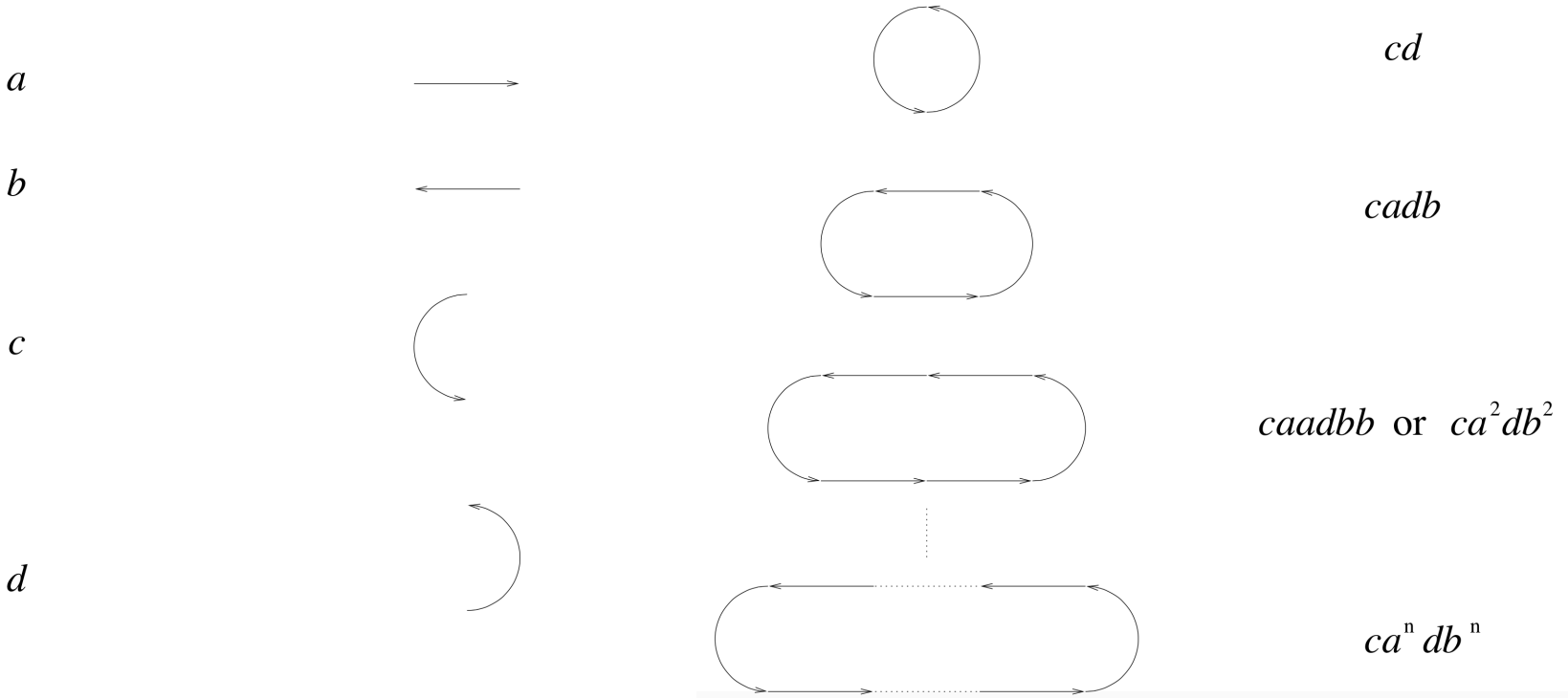
abcd



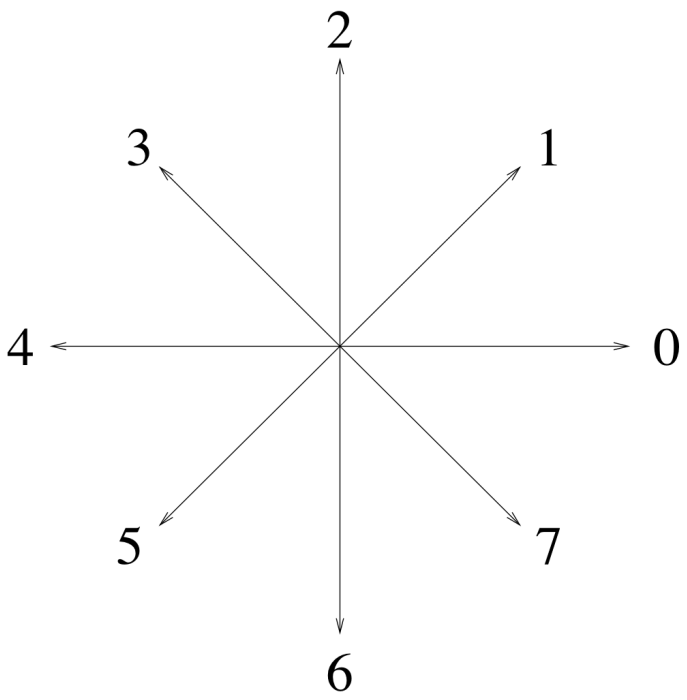
aabbccdd

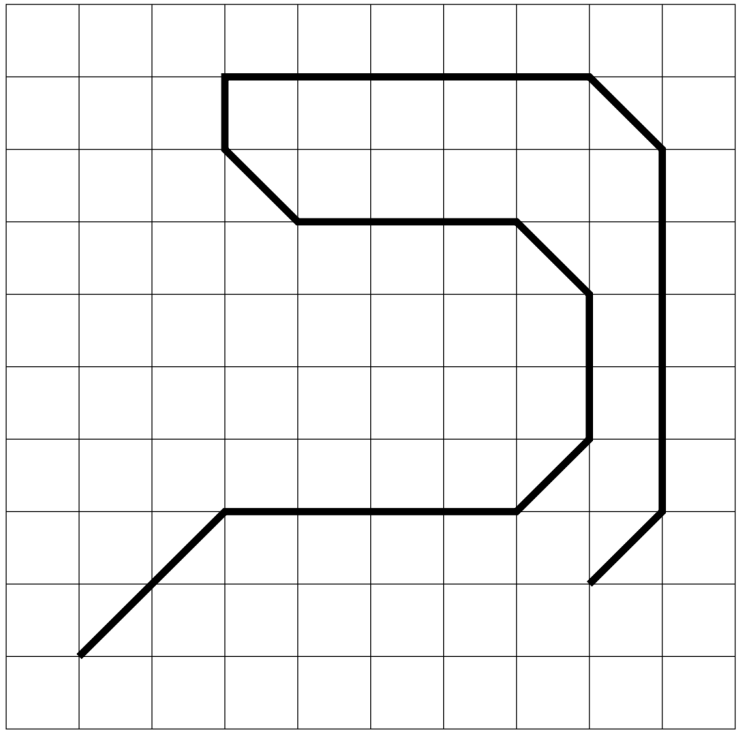
. . .

Example: Shapes



Example: Shapes





11000012234443200007666665

String

- An *alphabet* A = {a,b,…} is a finite set of symbols.
- A *string* or *word* $x = x_1x_2\ldots x_n$ is a sequence of symbols.

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

- *length* : |x| = n
  - *empty string* ε : |ε| = 0
- *concatenation* : $z = xy = x_1\ldots x_n y_1\ldots y_m$
  - $a\ldots a = a^i$
  - aε = εa = a
- *substring* : $x_i x_{i+1}\ldots x_{i+m}$
  - pattern −> pat, att, tte, ter, ern
- *subsequence* : $x_{i1}x_{i2}\ldots x_{im}$ with $i_j < i_k$ for j < k
  - pattern −> ptrn, at, tn, …
- *prefix* : $x_1x_2\ldots x_m$
  - pattern −> p, pa, pat, …
- *suffix* : $x_{n-m}x_{n-m+1}\ldots x_n$
  - pattern −> n, rn, ern, …

Formal Language

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

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

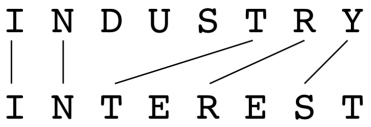
  - A* = {ε, a, b, c, aa, ab, …}
  - A^+ = A* \ {ε} = {a, b, c, aa, ab, …}
- A *formal language* is a subset of A*
  - L = {aaabb, ba, aab}
  - L = $a^n b^n c^n$

String Edit Distance

Edit Operations

- Three canonical edit operations:
  - Substitution (a → b)
  - Deletion (a → ε)
  - Insertion (ε → b)

Trace



Alignment or Matching

I	N	D	U	S	T	ø	R	ø	Y	ø
I	N	ø	ø	ø	T	E	R	E	S	T

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 → b), c(a → ε), c(ε → b).
- Edit operations correspond with pattern distortions. Large, improbable distortions should have high costs.
- Let (e_1,…,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:

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

SED Properties

- Lemma. The string edit distance is a metric if

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

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

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

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

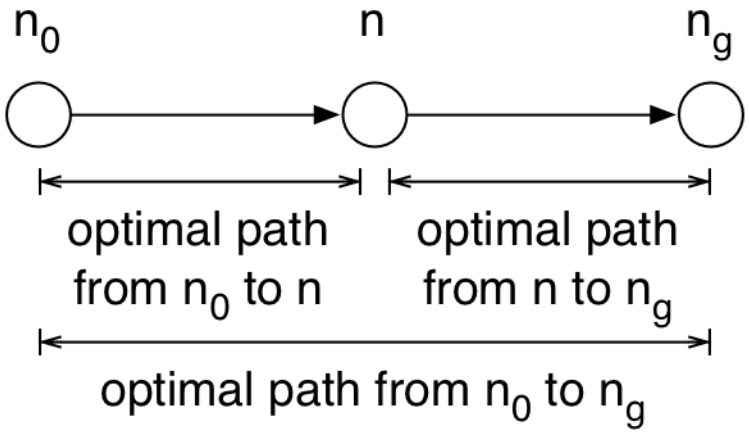
- In the following, we consider the weaker conditions

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

$$c(a \to a) = 0 \ \forall a \in A$$

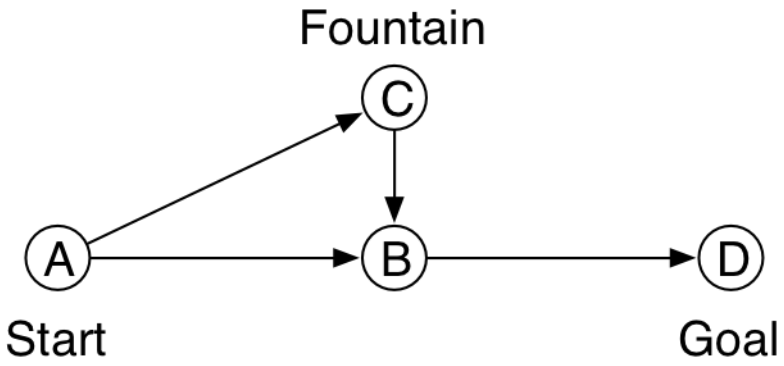
Dynamic Programming (DP)

- 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).



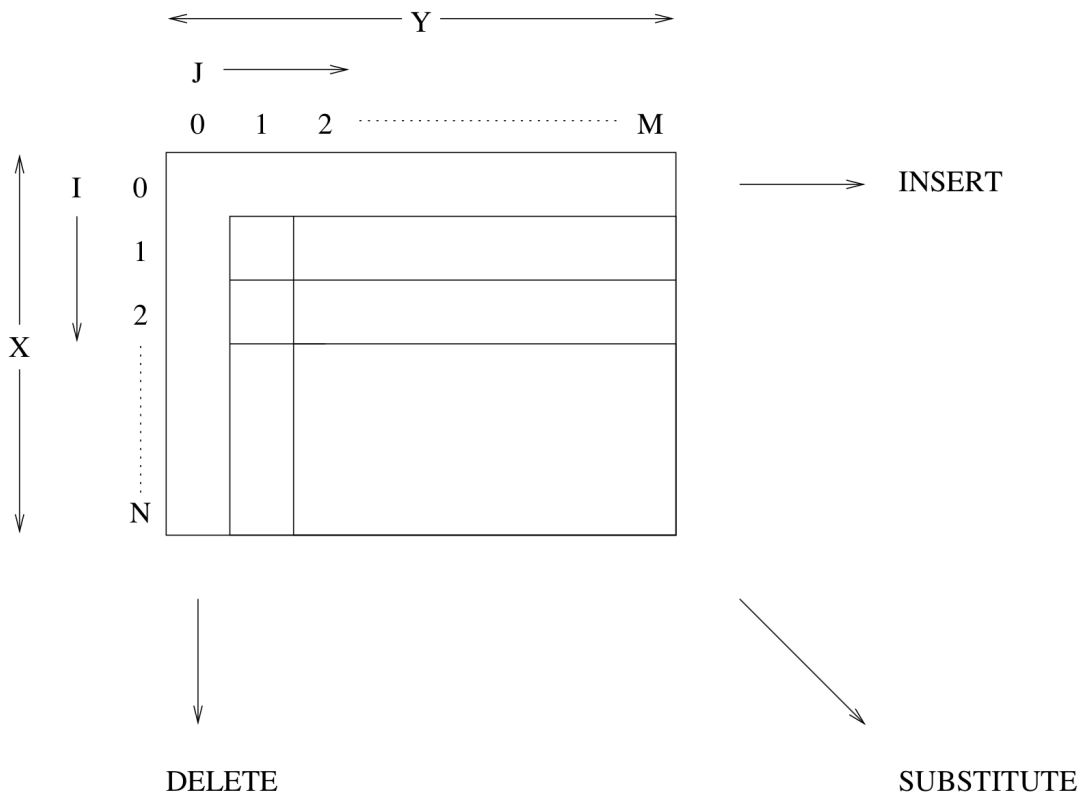
Counterexample (Not DP)

- 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 $x = x_1x_2\ldots x_N$ and $y = y_1y_2\ldots y_M$:



SED Algorithm

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

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

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** pointer(i,j):=$(i-1, j-1)$
12. **else if** $m_2 = D(i,j)$ **then** pointer(i,j):=$(i-1, j)$
13. **else** pointer(i,j):=$(i, j-1)$;
14. **end**;

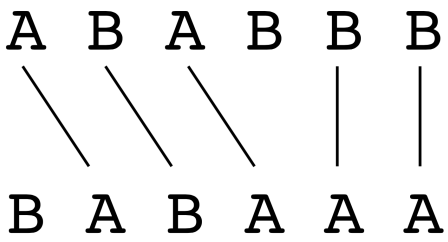
SED Example

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

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

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



	ε	B	A	B	A	A	A
ε	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 → v) with u = $u_1 \ldots u_k$ and v = $v_1 \ldots v_l$:

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

- Context-dependent costs:

$$c(a \to b : a\underline{a}a) \neq c(a \to b : b\underline{a}b)$$

- *Needleman–Wunsch* algorithm for aligning DNA sequences with alphabet A = {G, A, C, T}.
  - Find the maximum similarity S(x,y) with s(a→a) = 1 (“match”), s(a→b) = -1 (“mismatch”), s(a→ϵ) = s(ϵ→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* lcs(x,y) using the following cost function:
  - $c(a \rightarrow b) = 2$ if $a \neq b$
  - $c(a \rightarrow \varepsilon) = c(\varepsilon \rightarrow b) = 1$
  - $c(a \rightarrow a) = 0$
- Intuitive explanation: based on this cost function, the optimal edit path can be found without substitutions (a→b) where a ≠ b. The algorithm will search for the maximum number of diagonal steps (a→a) with zero cost. These diagonal steps correspond with the longest common subsequence. Therefore,

$$d(x,y) = |x| + |y| - 2|lcs(x,y)|$$

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

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

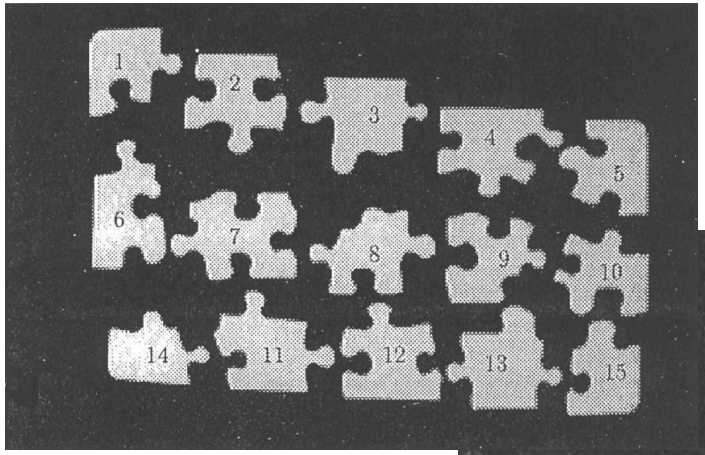
Nearest Neighbor Classification

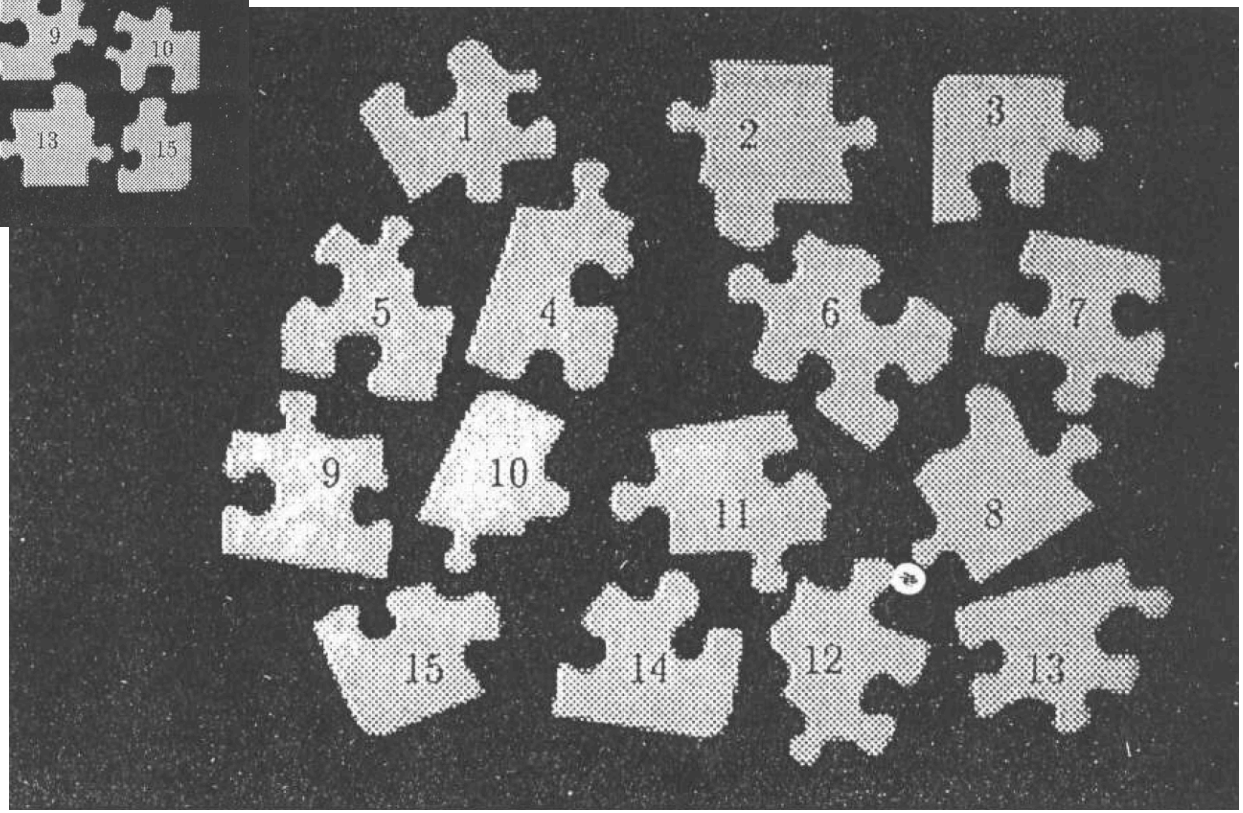
- NN classification based on string edit distance:
  - X = A*
  - θ = { learning samples }
  - f_θ : assign the class label of the most similar known sample

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

Example: Shape Matching

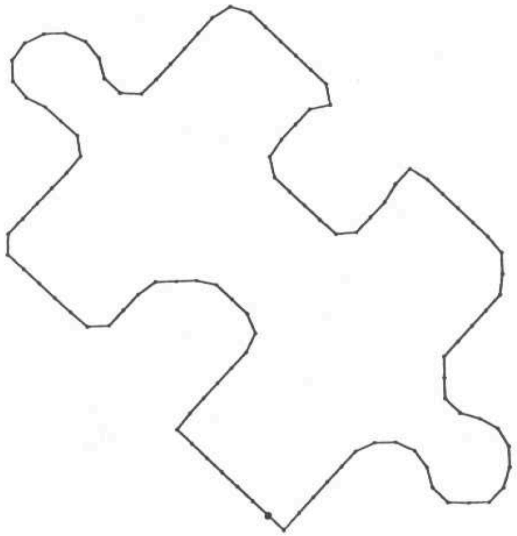
- Unknown shapes are rotated and scaled versions of known shapes.





Example: Shape Matching

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

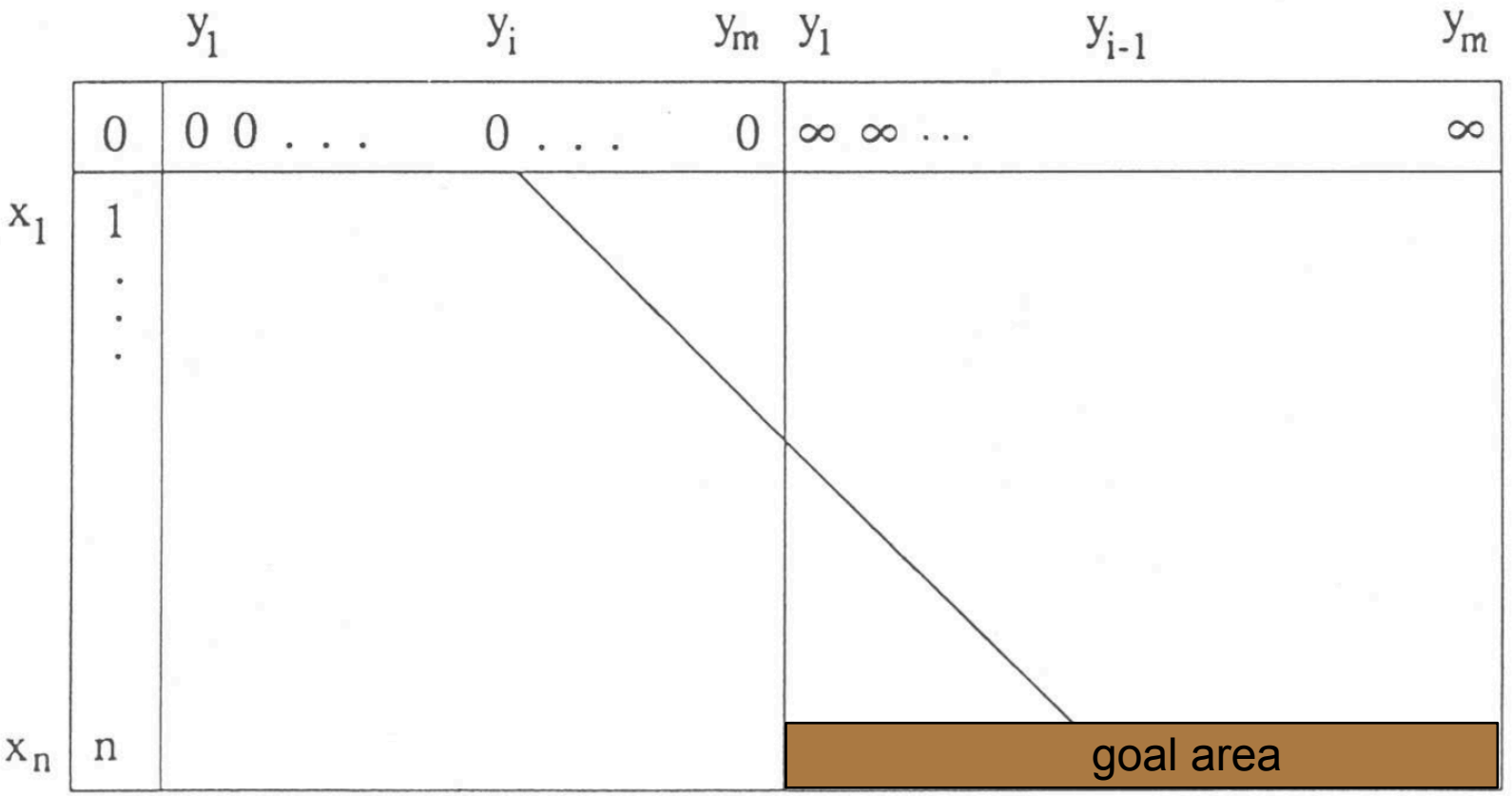


0	0	0	0	0	0	−86	−4	0	0
0	17	49	24	0	32	13	0	34	11
0	−44	−45	−1	0	0	0	−48	−42	0
0	0	5	46	39	0	16	−24	−37	−37
−39	−24	−25	−30	−22	32	41	40	3	0
0	0	−15	−48	−27	0	0	0	0	−36
−89	34	1	9	47	34	0	0	2	45
43	0	12	−12	−78	−12	0	0	0	−6
−39	−8	−37	0	0	0	44	1	44	28
−13	−30	−29	−19	−28	−43	−3	−42	−31	20
30	26	23	22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

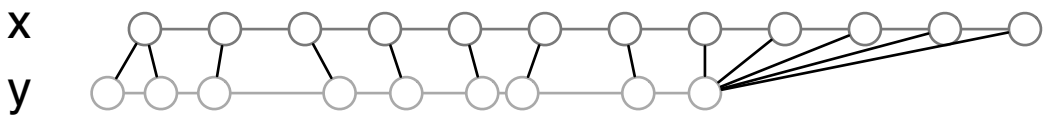
- The SED algorithm can be used to compute a dissimilarity d(x,y) between two feature vector sequences

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

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

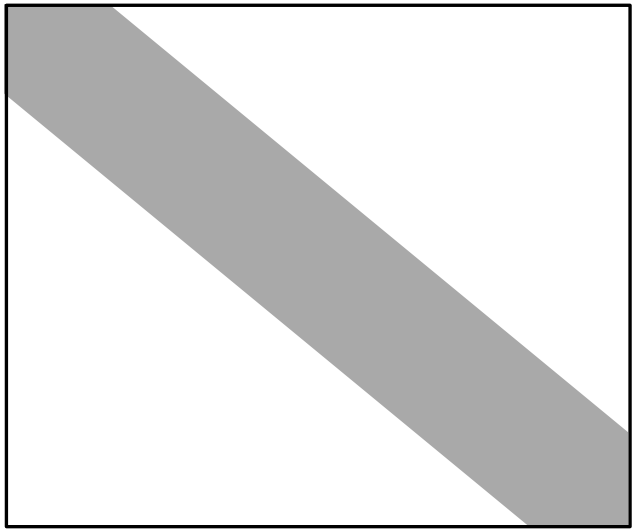
- Dynamic time warping aligns two sequences along a common time axis by considering only substitutions ($x_i \rightarrow y_j$), usually with Euclidean cost:

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



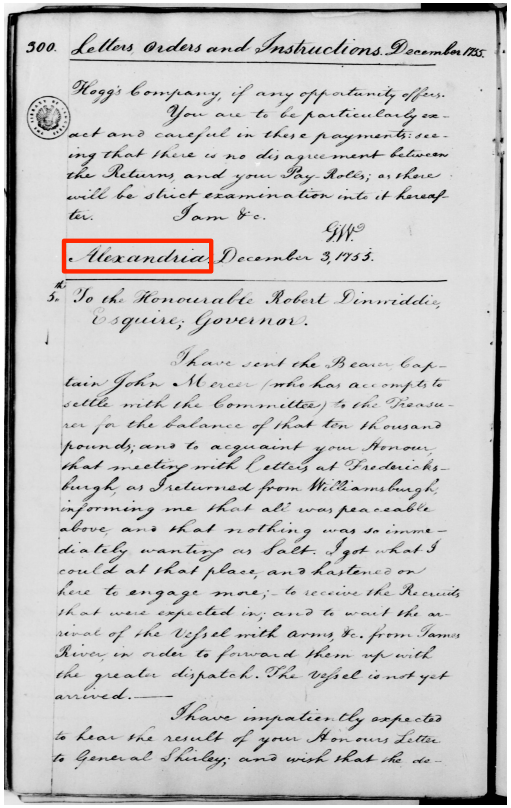
Sakoe-Chiba Band

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



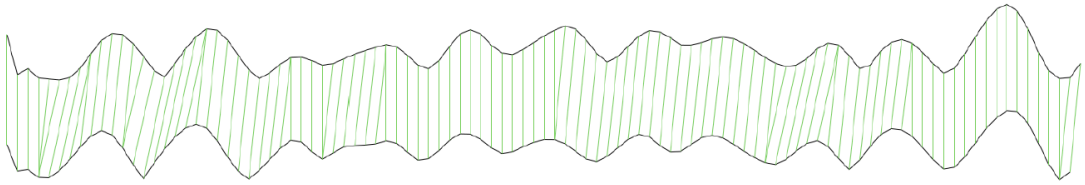
Example: Keyword Spotting

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

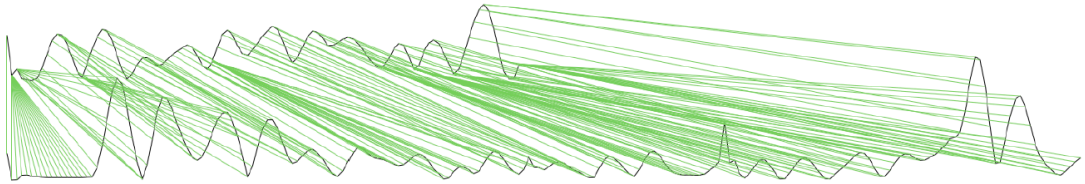


Example: Signature Verificaiton

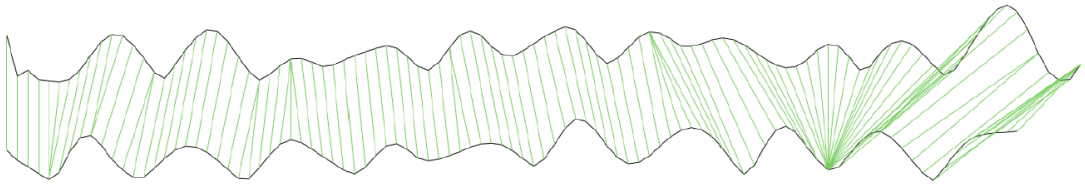
- Match the velocity $x_i=(v_x,v_y) \in R^2$ between two pen-tip trajectories recorded with an electronic device (tablet, smartphone, …).



(a) genuine signature



(b) random forgery



(c) skilled forgery