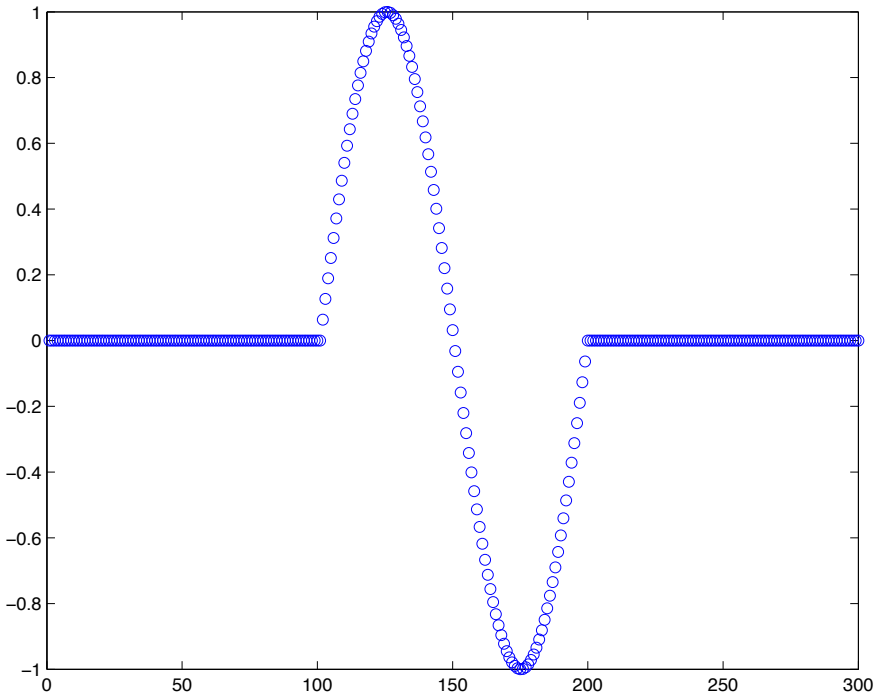
Dynamic Time Warping Algorithm

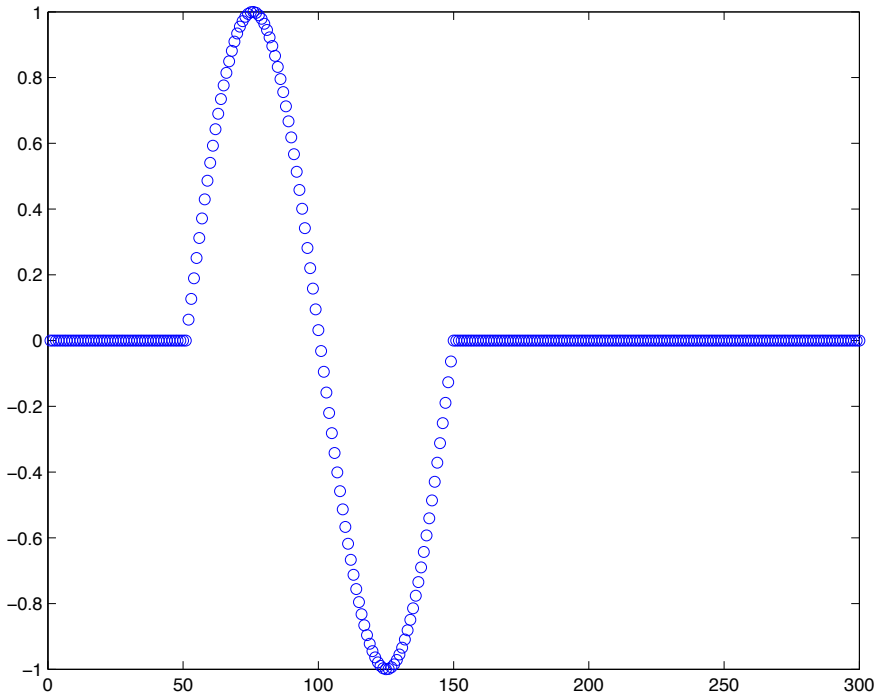
Slides from:

Elena Tsiporkova

Shift variance

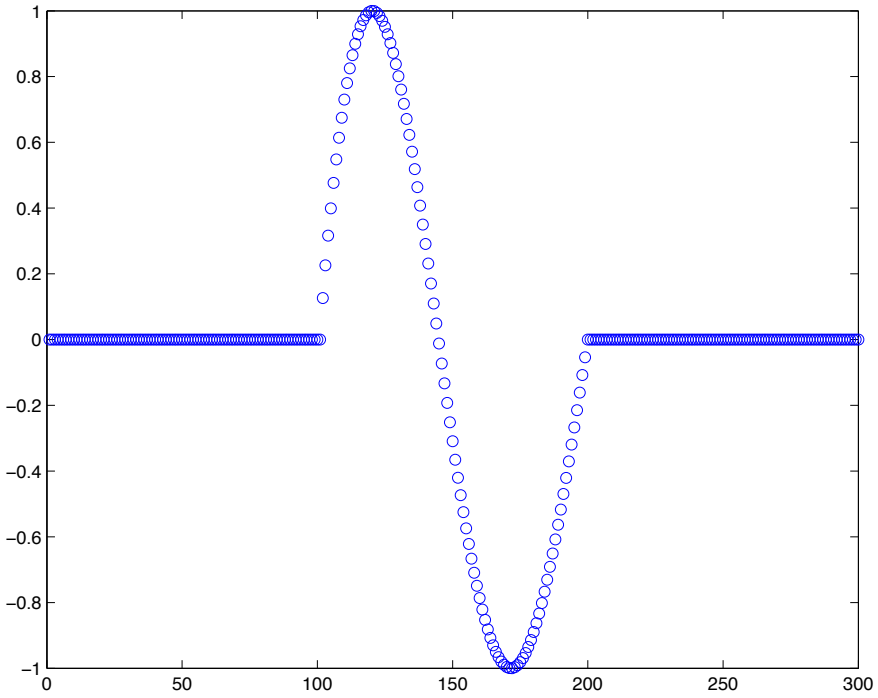
- Time series have shift variance
  - Are these two points close?

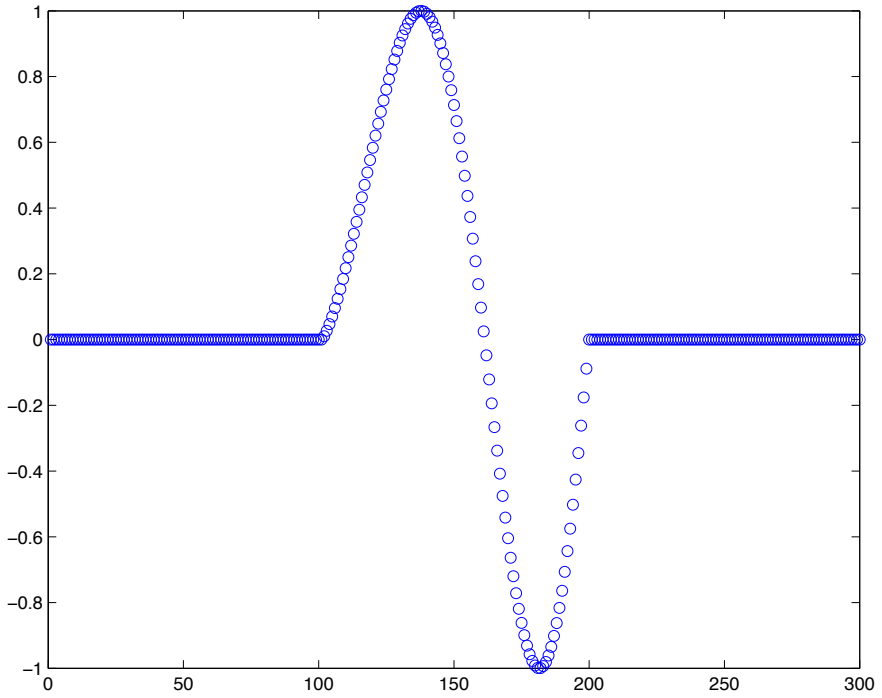




Time warp variance

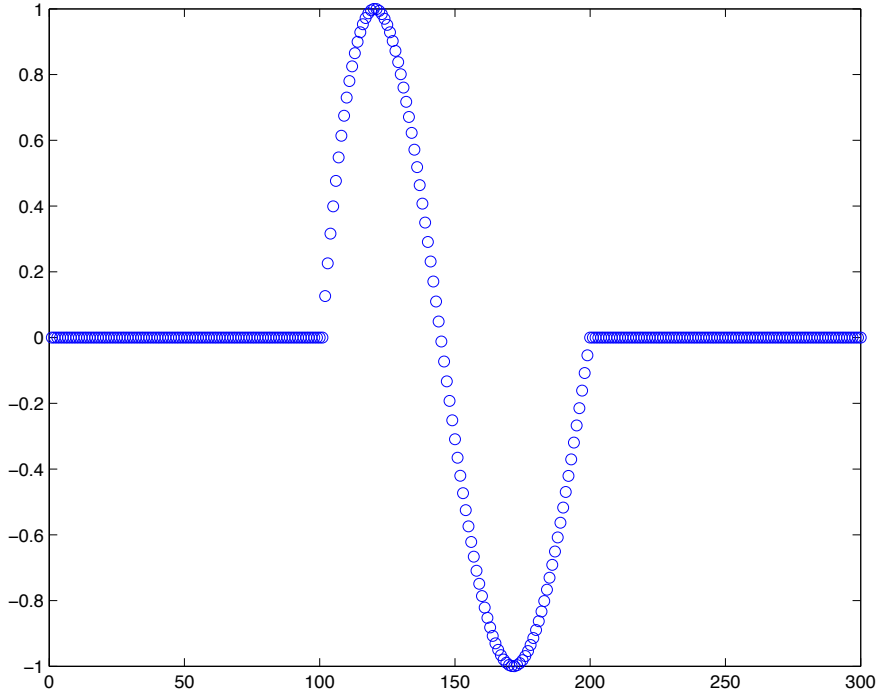
- Slight changes in timing are not relevant
  - Are these two point close?

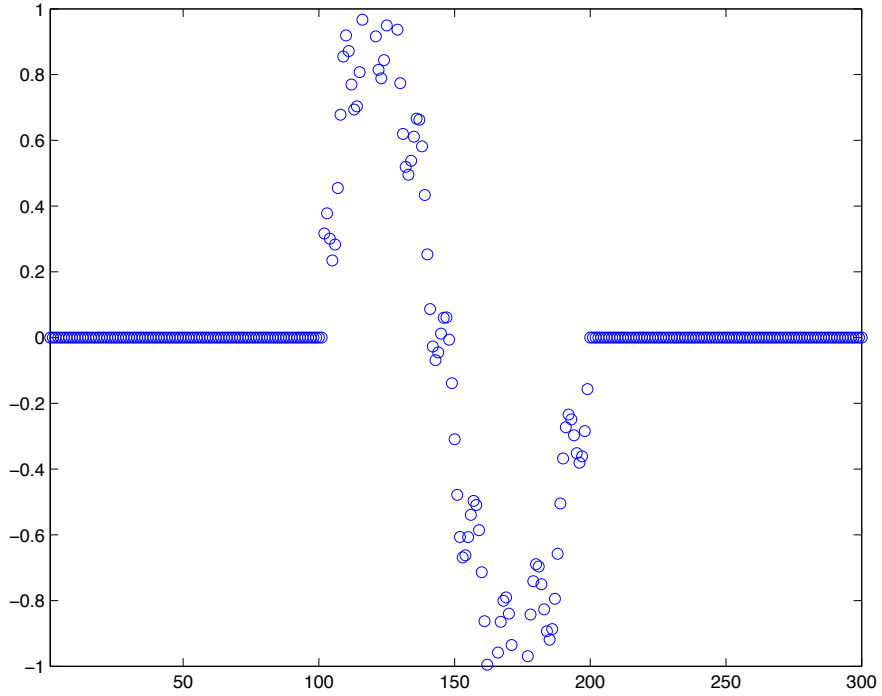




Noise/filtering variance

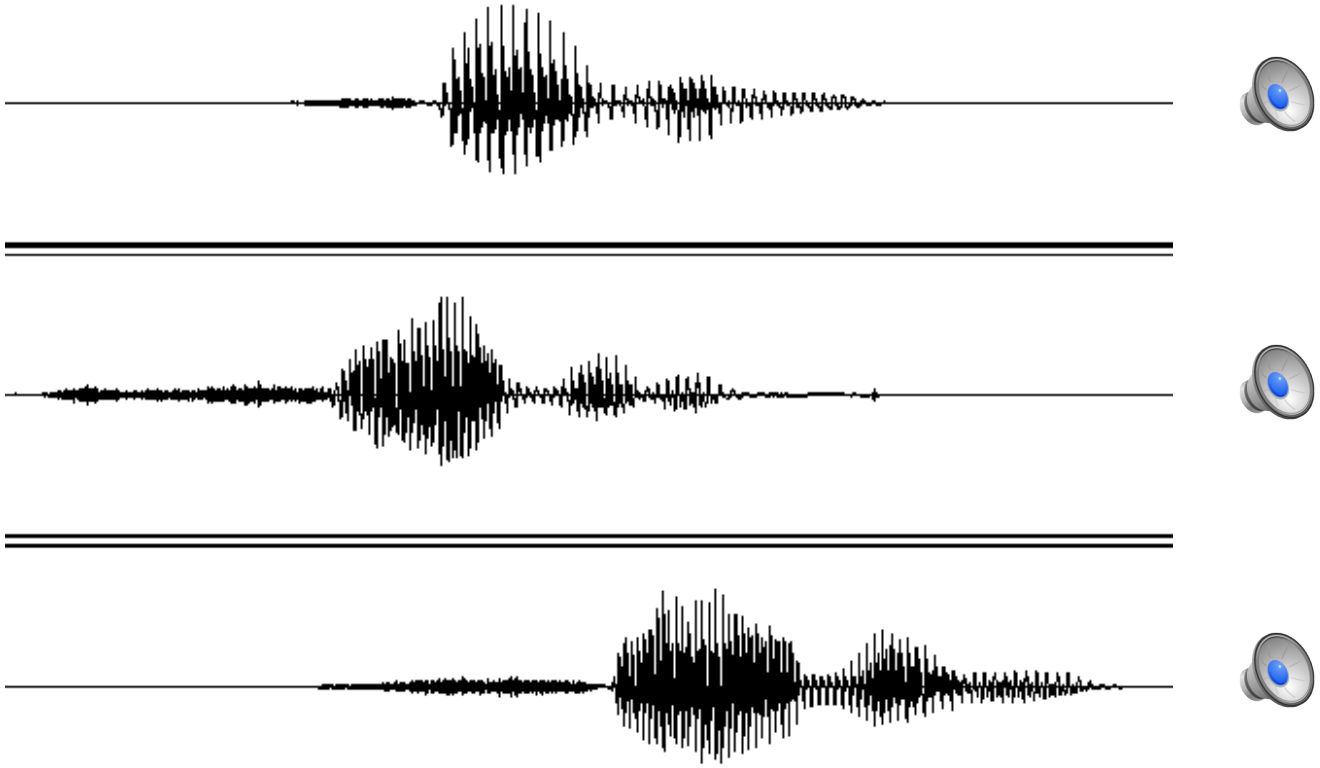
- Small changes can look serious
  - How about these two points?



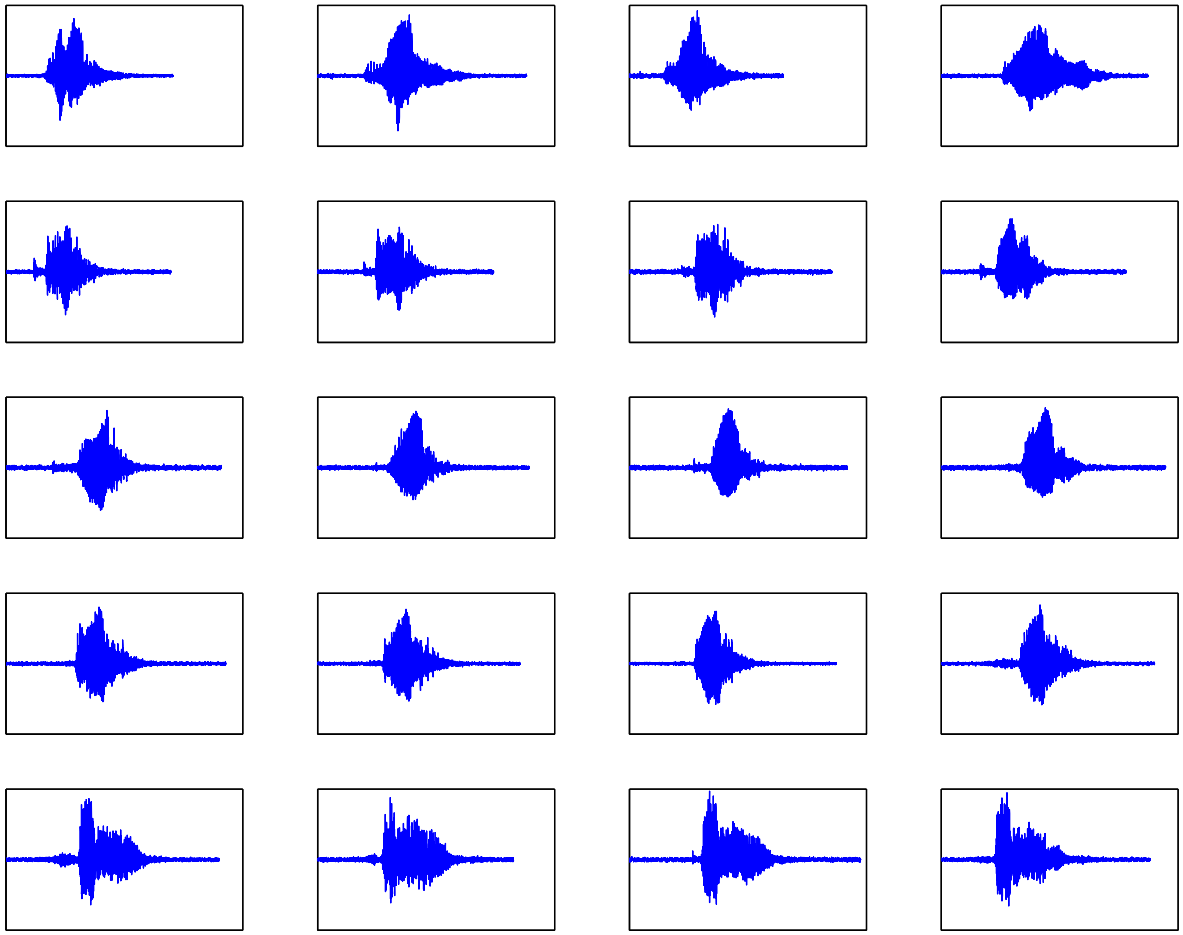


A real-world case

- Spoken digits

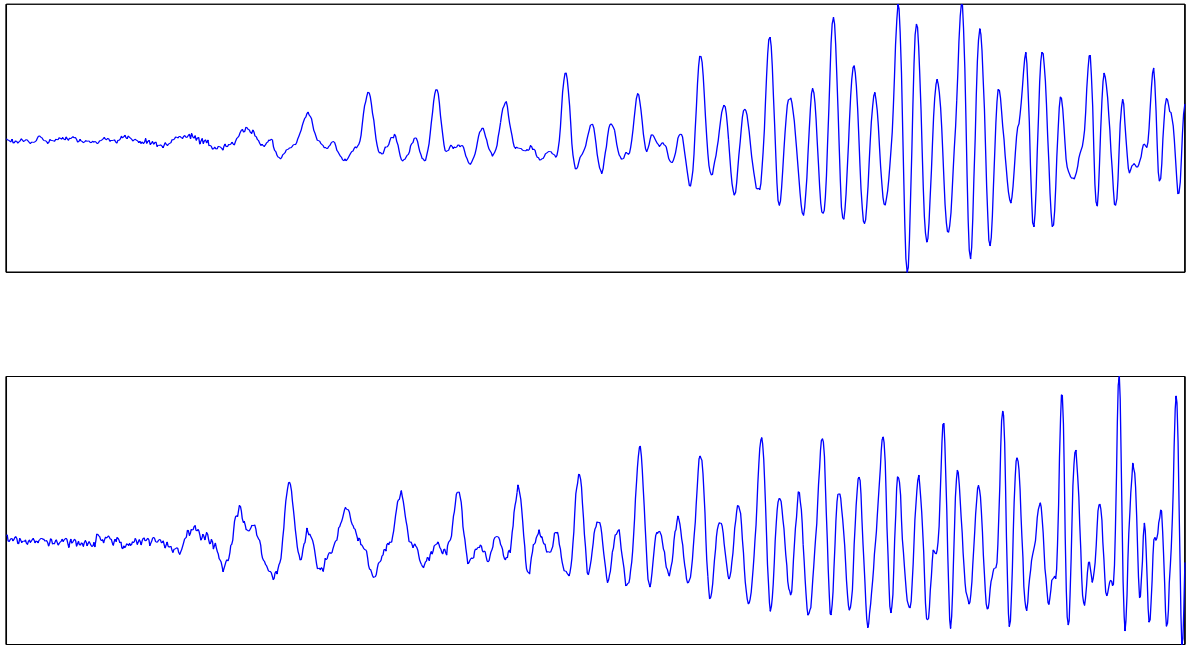


Example data



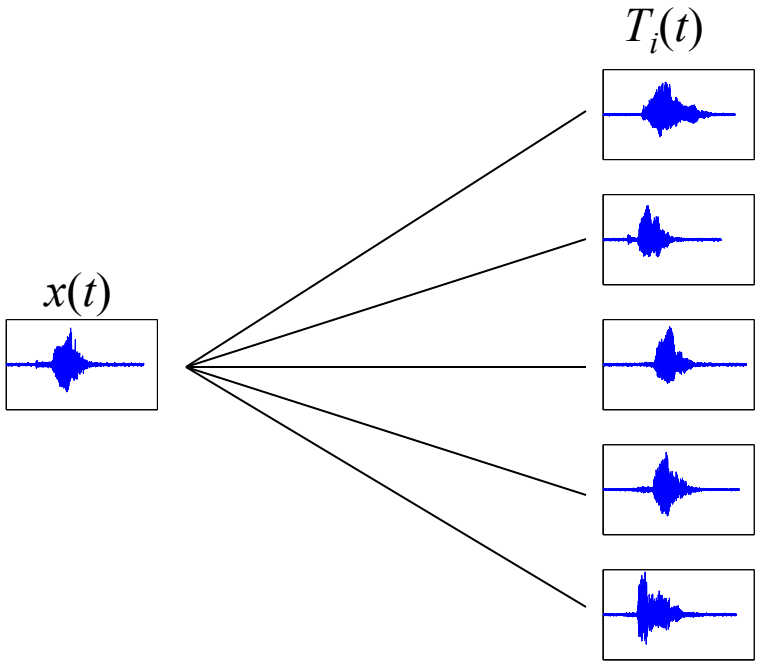
Going from fine to coarse

- Small differences are not important
  - Find features that obscure them



A basic speech recognizer

- Collect template spoken words $T_i(t)$
- Get their DTW distances from input $x(t)$
  - Smallest distance wins



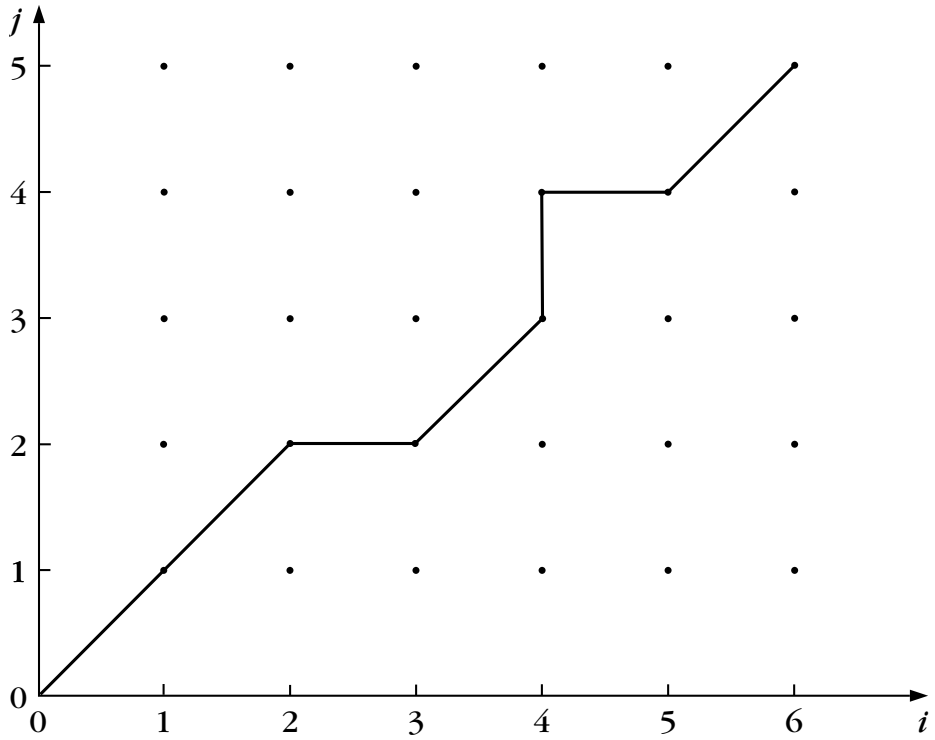
Clustering Time Series

- How do we cluster time series?
  - We can't just use k-means …
- We can use DTW for this

Matching warped series

- Represent the warping with a path

$$\mathbf{r}(i), i = 1, 2, \ldots, 6 \qquad \mathbf{t}(j), j = 1, 2, \ldots, 5$$

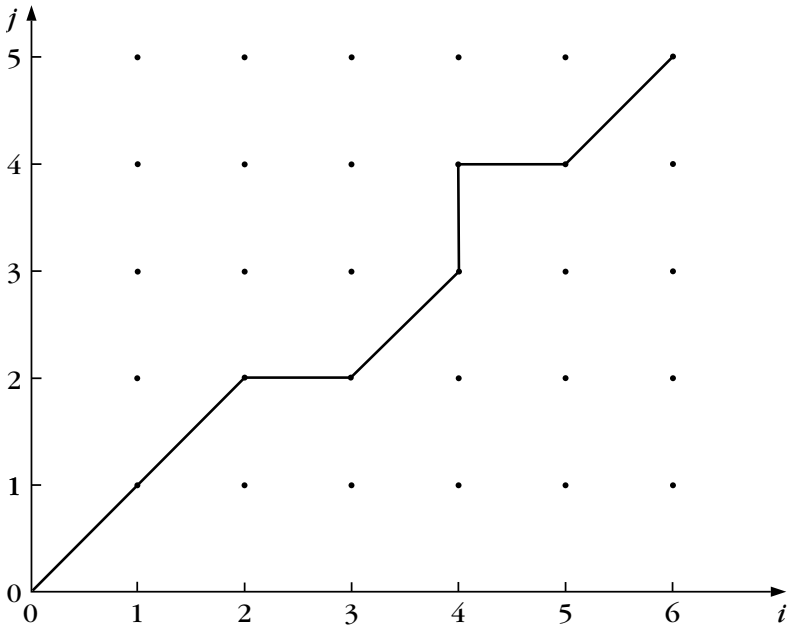


Finding the overall "distance"

- Each node will have a *cost*
  - e.g., $d(i,j) = \left\| \mathbf{r}(i) - \mathbf{t}(j) \right\|$
- Overall path *cost* is:

$$D = \sum_k d(i_k, j_k)$$

- Optimal D path defines the "distance" between two given sequences



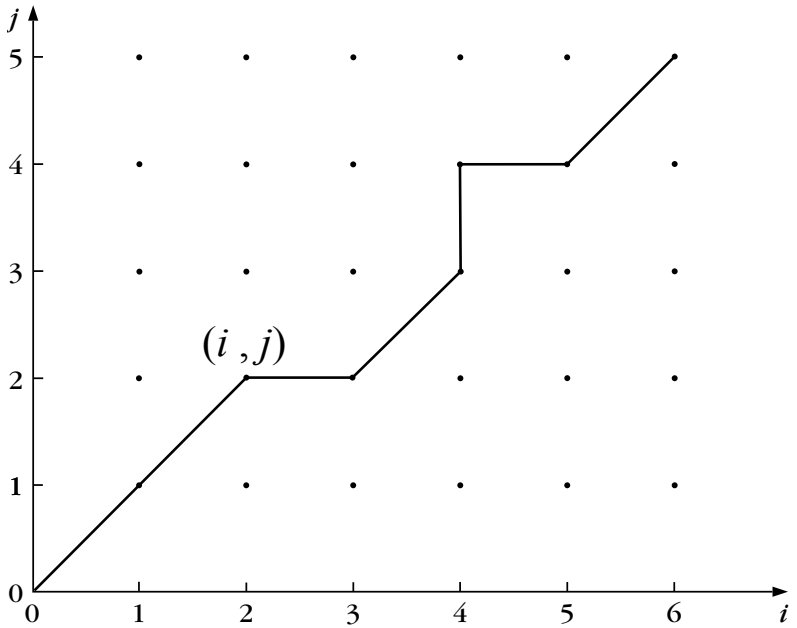
Bellman's optimality principle

- For an optimal path passing through $(i\ ,\ j)$:

$$(i_0, j_0) \overset{opt}{\longrightarrow} (i_f, j_f)$$

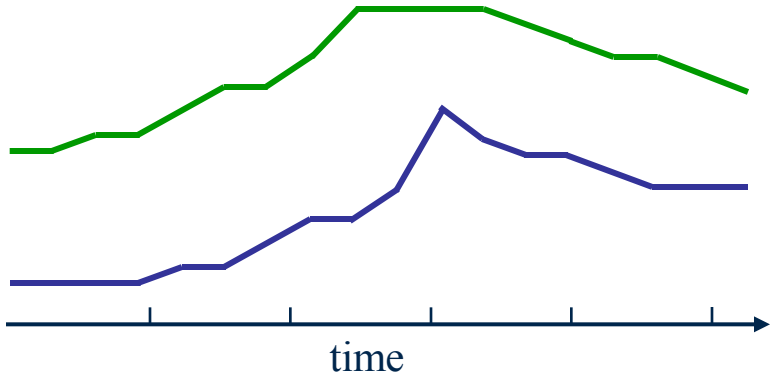
- Then:

$$(i_0, j_0) \overset{opt}{\longrightarrow} (i_f, j_f) = \left\{ (i_0, j_0) \overset{opt}{\longrightarrow} (i, j), (i, j) \overset{opt}{\longrightarrow} (i_f, j_f) \right\}$$

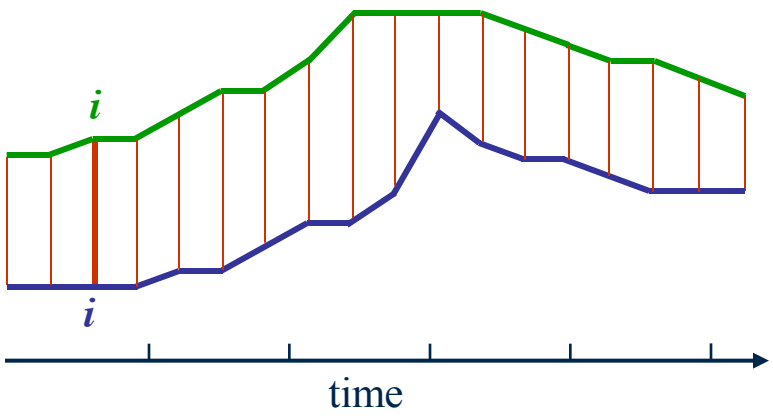


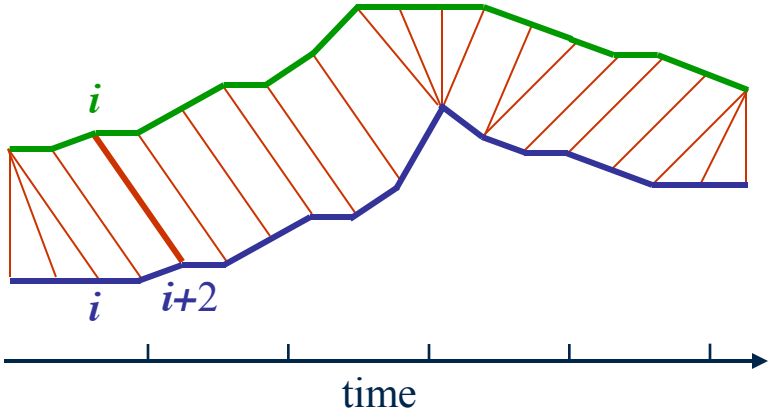
What is Special about Time Series Data?

Gene expression time series are expected to vary not only in terms of expression amplitudes, but also in terms of time progression since biological processes may unfold with different rates in response to different experimental conditions or within different organisms and individuals.



Why Dynamic Time Warping?

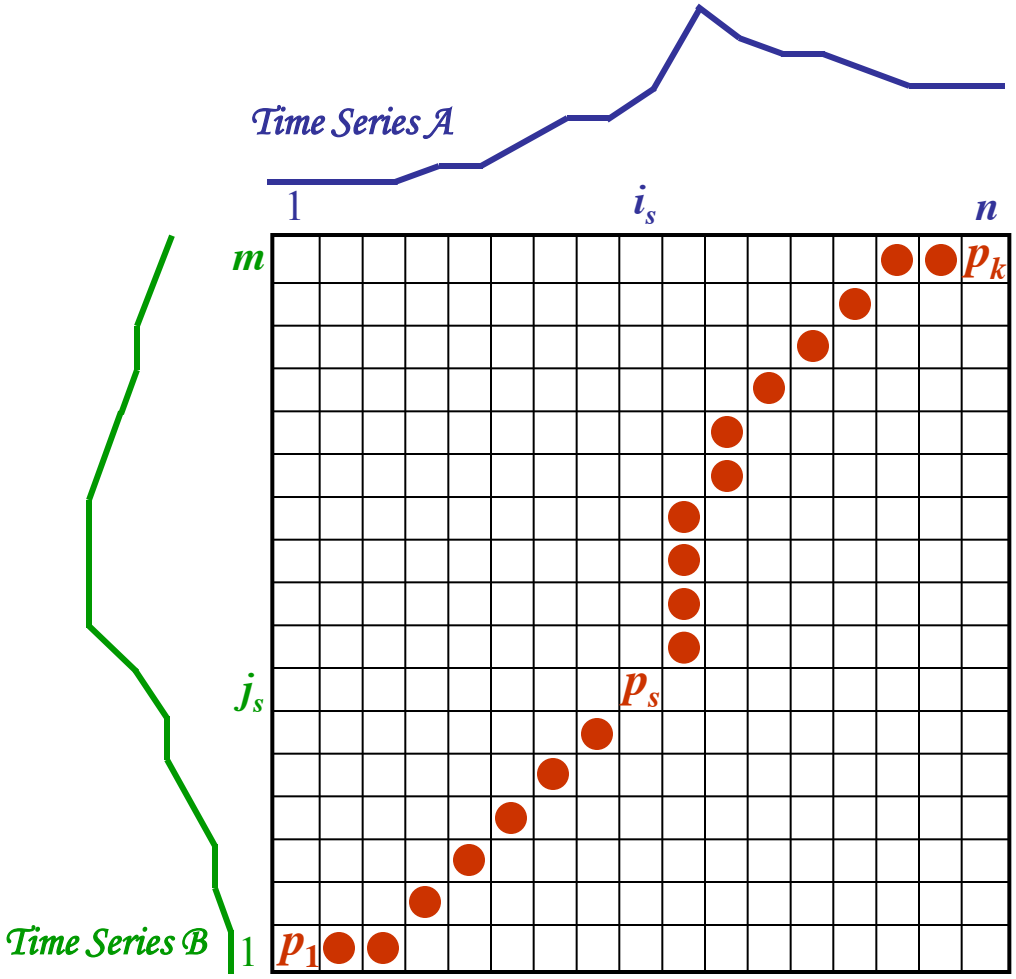




Any distance (Euclidean, Manhattan, …) which aligns the *i*-th point on one time series with the *i*-th point on the other will produce a poor similarity score.

A non-linear (elastic) alignment produces a more intuitive similarity measure, allowing similar shapes to match even if they are out of phase in the time axis.

Warping Function



To find the *best alignment* between $\mathcal{A}$ and $\mathcal{B}$ one needs to find the path through the grid

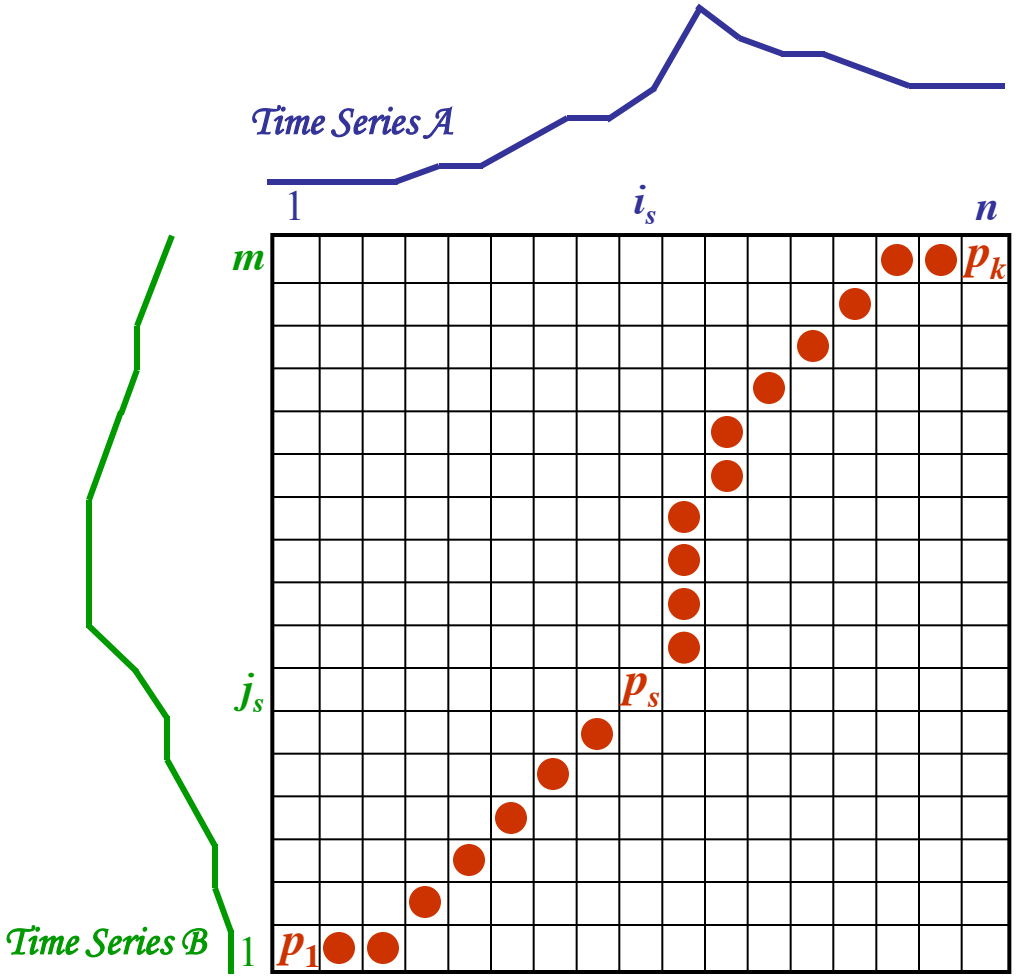
$P = p_1, \ldots, p_s, \ldots, p_k$

$p_s = (i_s, j_s)$

which *minimizes* the total distance between them.

P is called a *warping function*.

Time-Normalized Distance Measure



Time-normalized distance between $\mathcal{A}$ and $\mathcal{B}$:

$$D(\mathcal{A}, \mathcal{B}) = \left[\frac{\sum_{s=1}^{k} d(p_s) \cdot w_s}{\sum_{s=1}^{k} w_s} \right]$$

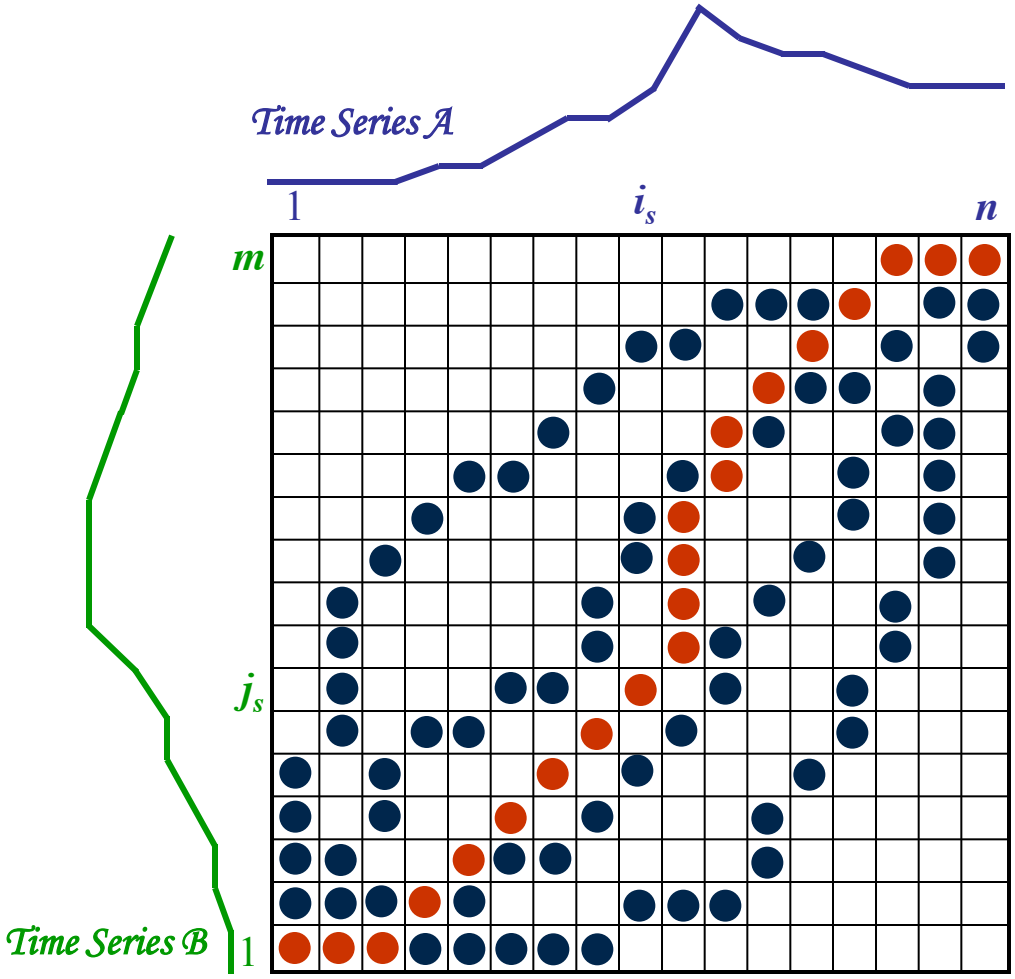
$d(p_s)$: distance between i_s and j_s

$w_s > 0$: weighting coefficient.

Best alignment path between $\mathcal{A}$ and $\mathcal{B}$:

$$P_0 = \arg\min_{P} (D(\mathcal{A}, \mathcal{B})).$$

Optimisations to the DTW Algorithm



The number of possible warping paths through the grid is exponentially explosive!

reduction of the search space

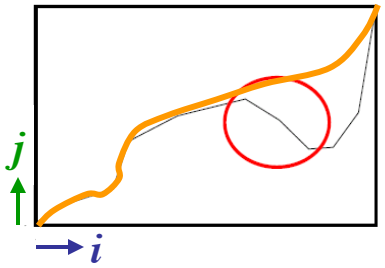
Restrictions on the warping function:

- monotonicity
- continuity
- boundary conditions
- warping window
- slope constraint.

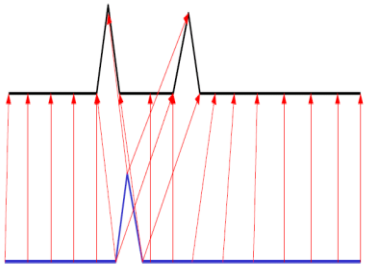
Restrictions on the Warping Function

Monotonicity: $i_{s-1} \leq i_s$ and $j_{s-1} \leq j_s$.

The alignment path does not go back in "*time*" index.

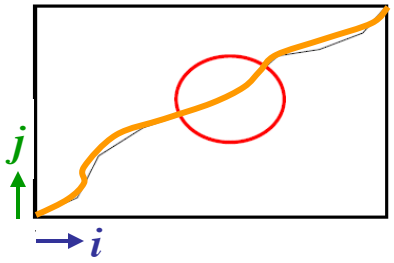


Guarantees that features are not repeated in the alignment.

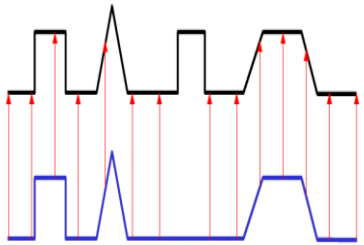


Continuity: $i_s - i_{s-1} \leq 1$ and $j_s - j_{s-1} \leq 1$.

The alignment path does not jump in "*time*" index.



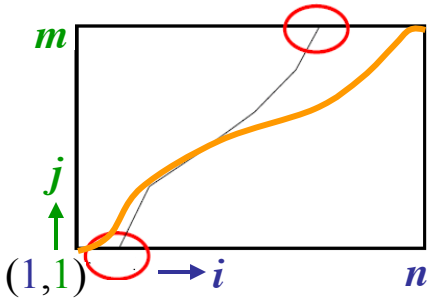
Guarantees that the alignment does not omit important features.



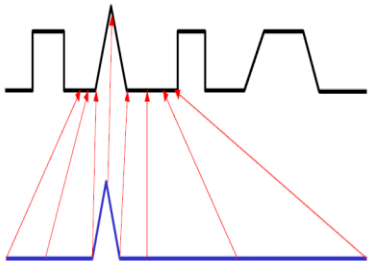
Restrictions on the Warping Function

Boundary Conditions: $i_1 = 1$, $i_k = n$ and $j_1 = 1$, $j_k = m$.

The alignment path starts at the bottom left and ends at the top right.

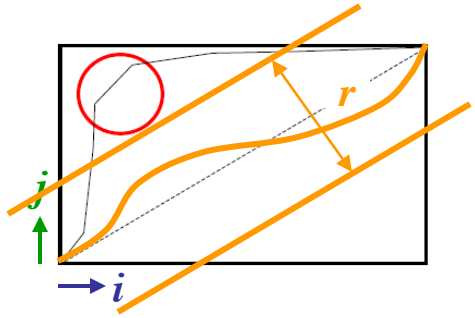


Guarantees that the alignment does not consider partially one of the sequences.

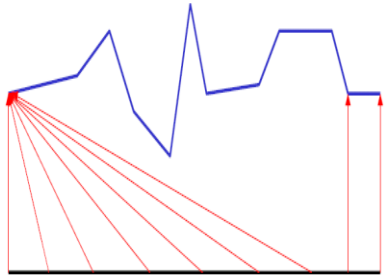


Warping Window: $|i_s - j_s| \leq r$, where $r > 0$ is the window length.

A good alignment path is unlikely to wander too far from the diagonal.



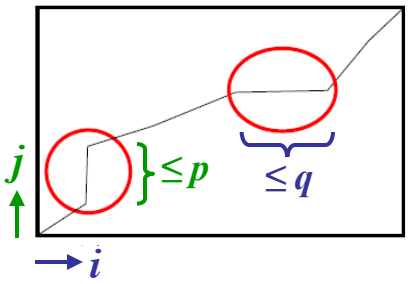
Guarantees that the alignment does not try to skip different features and gets stuck at similar features.



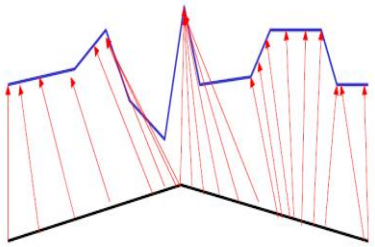
Restrictions on the Warping Function

Slope Constraint: $(j_{s_p} - j_{s_0}) / (i_{s_p} - i_{s_0}) \leq p$ and $(i_{s_q} - i_{s_0}) / (j_{s_q} - j_{s_0}) \leq q$, where $q \geq 0$ is the number of steps in the x-direction and $p \geq 0$ is the number of steps in the y-direction. After q steps in x one must step in y and vice versa: $S = p / q \in [0 , \infty]$.

The alignment path should not be too steep or too shallow.



Prevents that very short parts of the sequences are matched to very long ones.



The Choice of the Weighting Coefficient

Time-normalized distance between $\mathcal{A}$ and $\mathcal{B}$:

$$D(\mathcal{A},\mathcal{B}) = \min_{P}\left[\frac{\sum_{s=1}^{k} d(p_s)\cdot w_s}{\sum_{s=1}^{k} w_s}\right].$$

complicates optimisation

Seeking a weighting coefficient function which guarantees that:

$$C = \sum_{s=1}^{k} w_s$$

is independent of the warping function. Thus

$$D(\mathcal{A},\mathcal{B}) = \frac{1}{C}\min_{P}\left[\sum_{s=1}^{k} d(p_s)\cdot w_s\right]$$

can be solved by use of dynamic programming.

Weighting Coefficient Definitions

- *Symmetric form*

  $w_s = (i_s - i_{s-1}) + (j_s - j_{s-1})$,

  then $C = n + m$.

- *Asymmetric form*

  $w_s = (i_s - i_{s-1})$,

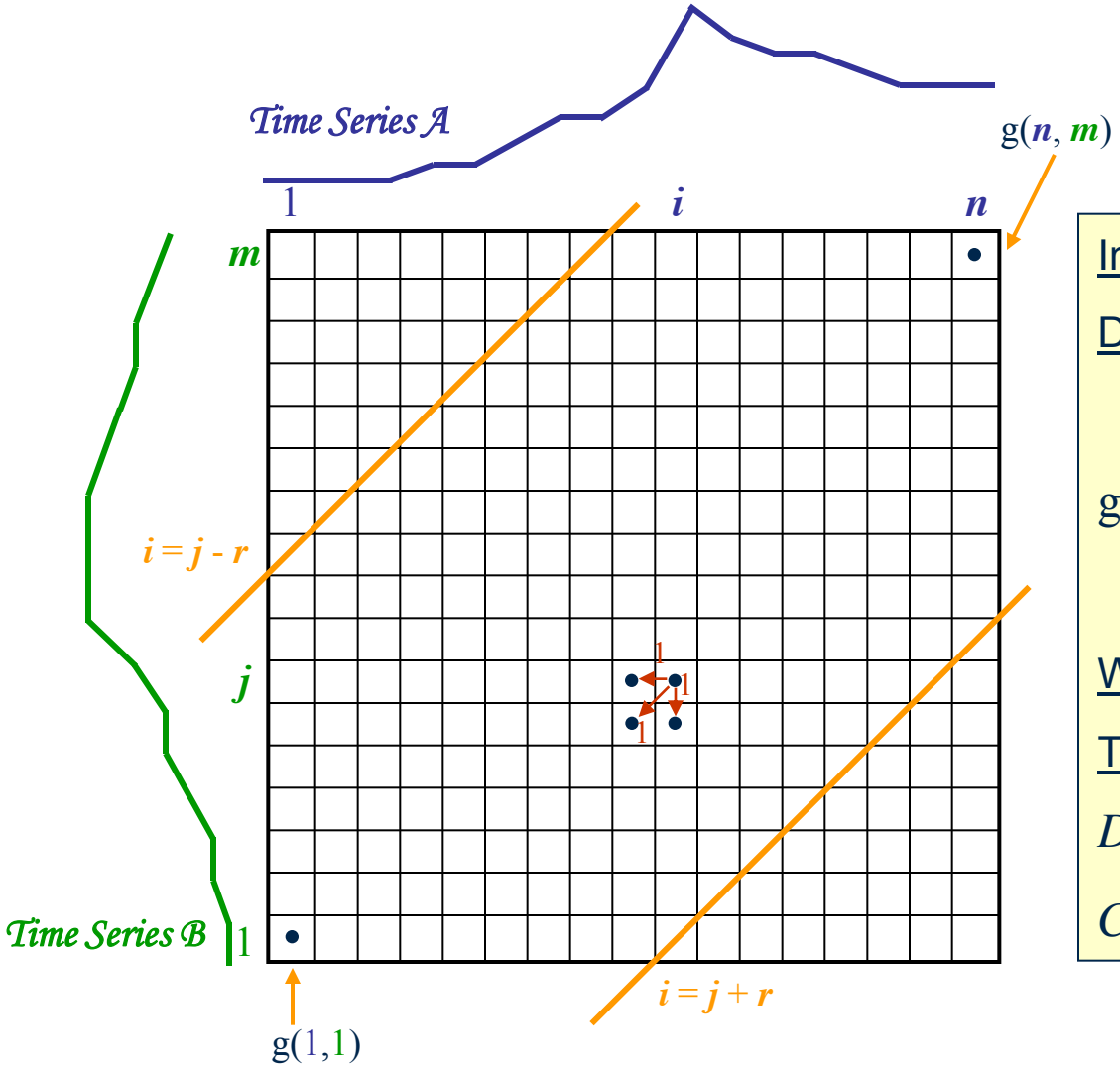
  then $C = n$.

  Or equivalently,

  $w_s = (j_s - j_{s-1})$,

  then $C = m$.

Quazi-symmetric DTW Algorithm (warping window, no slope constraint)



Initial condition: $g(1,1) = d(1,1)$.

DP-equation:

$$g(i,j) = \min \begin{pmatrix} g(i, j-1) + d(i,j) \\ g(i-1, j-1) + d(i,j) \\ g(i-1, j) + d(i,j) \end{pmatrix}.$$

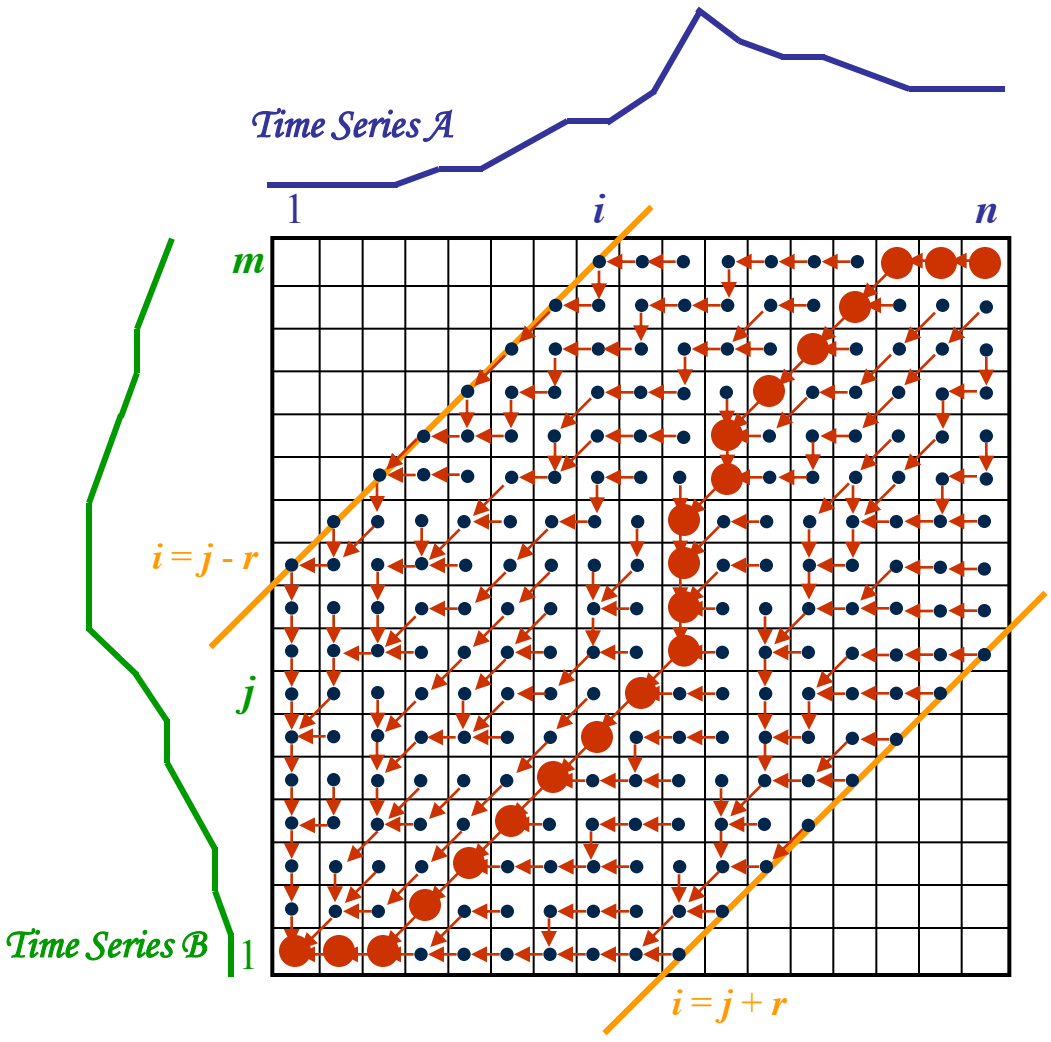
Warping window: $j - r \le i \le j + r$.

Time-normalized distance:

$D(\mathcal{A}, \mathcal{B}) = g(n, m) / C$

$C = n + m$.

DTW Algorithm at Work



Start with the calculation of g(1,1) = d(1,1).

Calculate the first row g(***i***, 1) = g(***i***–1, 1) + d(***i***, 1).

Calculate the first column g(1, ***j***) = g(1, ***j***) + d(1, ***j***).

Move to the second row g(***i***, 2) = min(g(***i***, 1), g(***i***–1, 1), g(***i*** – 1, 2)) + d(***i***, 2). Book keep for each cell the index of this neighboring cell, which contributes the minimum score (red arrows).

Carry on from left to right and from bottom to top with the rest of the grid g(***i***, ***j***) = min(g(***i***, ***j***–1), g(***i***–1, ***j***–1), g(***i*** – 1, ***j***)) + d(***i***, ***j***).

Trace back the best path through the grid starting from g(***n***, ***m***) and moving towards g(1,1) by following the red arrows.

DTW Algorithm: Example