Movement and Artificial Intelligence

M2 – INF 3002

AIDN: Interactive Analysis of Digital Data

Sylvie Gibet

Outline of the course

- **Lecture 1. Motion capture**
  - Motion capture: sensors, pre-processing,
  - Motion capture pipeline
  - Skeleton representation
    - TP mocap
  - Programming with motion capture

- **Lecture 2. Motion Processing**
  - Dynamic programming - DTW
    - TD/TP DTW using gestural traces (Wacom tablet)
  - Dimension-reduction – PCA
  - Motion descriptors

Programmation dynamique - Rappels

Principe général

- Récursivité : la décomposition récursive est un principe fondamental en algorithmique. Avec le principe « Diviser pour Régner », on obtient une décomposition qui conduit à des algorithmes de grande qualité.
- Cependant, pour certains problèmes, cette décomposition génère plusieurs sous-problèmes identiques -> on cherche alors à trouver un algorithme optimal qui calcule chaque sous-problème une seule fois.

Programmation dynamique

Principe général

- La programmation dynamique est une méthode d'optimisation opérant en phases et dont l'efficacité repose sur le principe d'optimalité de Bellman : « Toute politique optimale est composée de sous-politiques optimales ».

Exemple - Programmation dynamique

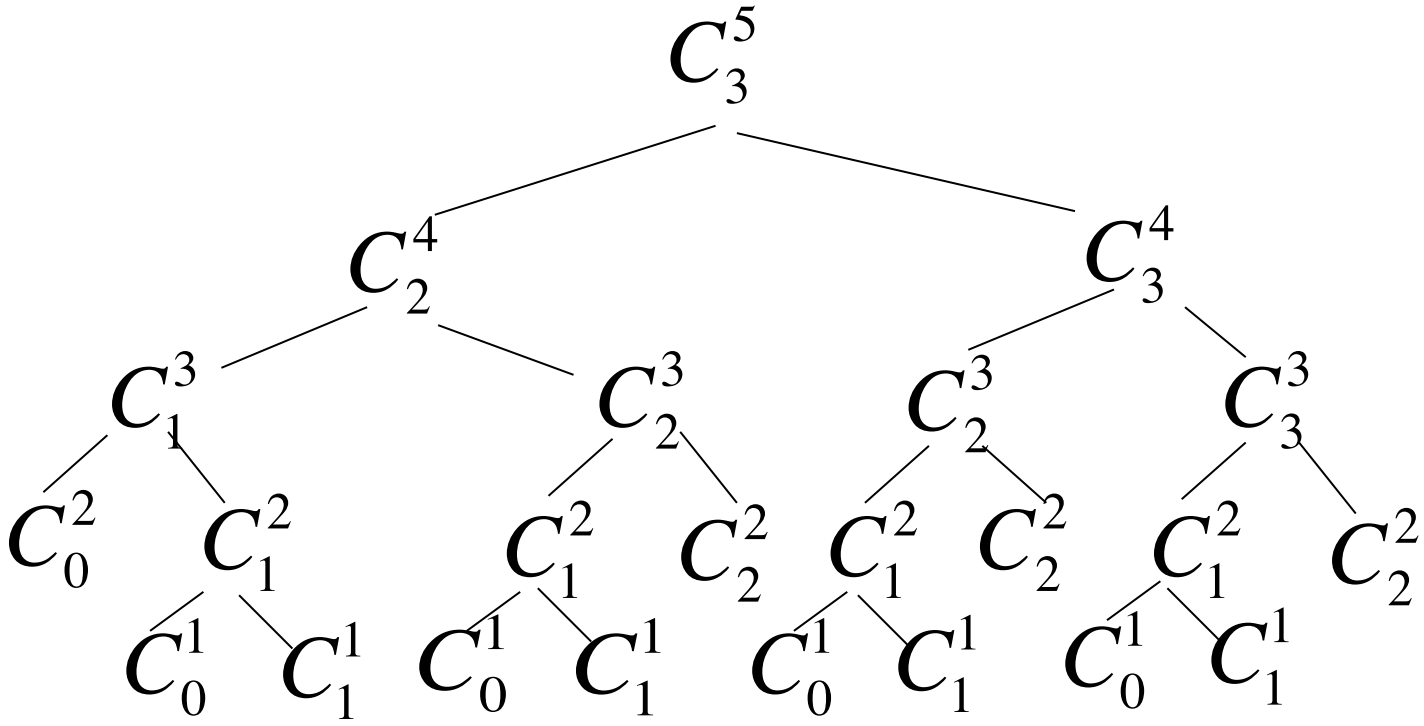
Calcul récursif des combinaisons C_k^n

- On veut calculer $C_k^n = \frac{n!}{k!(n-k)!}$
- Propriété :
  - $C_k^n = C_{k-1}^{n-1} + C_k^{n-1}$ pour 0<k<n
  - 1 sinon
- Approche « Diviser pour Régner »

  Fonction C(n,k) :
  - Si k=0 ou k=n alors retourner 1
  - Retourner C(n-1, k-1) + C(n-1,k)

Arbre de calcule de C(5,3)

Nombre d'appels récursifs : $T(n,k) = 2\ C(n,k) - 2$



Optimisation : triangle de Pascal

	0	1	2	3	…	n-1	n
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
.	. .	.	.				
n-1 1	n-1			…	1		
n	1	n	C_2^{n-1} C_2^n	C_3^{n-1} C_3^n	…	n	1

Triangle de Pascal

Principe

1. On remplit les k premières cases de chaque ligne de haut en bas
2. On arrête à la ligne n
3. Temps : $T(n,k) = O(nk)$

Remarque : il n'est pas nécessaire de mémoriser tout le tableau

Principe de la programmation dynamique

- Souvent utilisé lorsqu'une solution récursive se révèle inefficace
  1. **Solution récursive** : méthode descendante : on part de n et on exprime le problème en plusieurs sous-problèmes de taille p, avec p<n
  2. **Programmation dynamique** : méthodes ascendante : on part des cas de base et on calcule itérativement sans qu'il y ait de redondance (on évite de calculer plusieurs fois la même chose)

Algorithme − Plus longue sous-séquence commune (LCS)

- Soient deux sous-séquences $X_m = (x_1, \dots, x_m)$ et $Y_n = (y_1, ..., y_n)$, on a alors plusieurs cas de figure :
  - Si xm = yn alors le problème revient à trouver la LCS de $X_{m-1} = (x_1, \dots, x_{m-1})$ et $Y_{n-1} = (x_1, \dots, x_{m-1})$
  - Sinon, le problème revient à trouver la plus longue des chaînes parmi $LCS(X_{m-1}, Y_n)$ et $LCS(X_m, Y_{n-1})$

  (le raisonnement peut être fait sur les premières lettres)

Algorithme − Plus longue sous-séquence commune (LCS)

- Relation de récurrence

$$LCS\left(X_m, Y_n\right) = \begin{cases} 0 & \text{si } \ n = 0 \text{ ou } m = 0 \\ LCS\left(X_{m-1}, Y_{n-1}\right) + 1 & \text{si } x_m = y_n \\ \max\left(LCS\left(X_m, Y_{n-1}\right), LCS\left(X_{m-1}, Y_n\right)\right) & \text{si } x_m \neq y_n \end{cases}$$

- Question : nombre de sous-problèmes possibles pour deux chaînes de longueur *m* et *n* ?

Reconnaissance de chaînes de caractères bruitées

Principe général

- Applications importantes en bio-informatique
- On considère des chaînes sur un alphabet fini V. Une chaîne de référence $X = x_1..x_N$ est comparée avec une chaîne $Y = y_1..y_M$. La seconde est une version bruitée de la première, et les perturbations sont de trois types :

1. Disparition d'un élément de X
2. Insertion d'un élément parasite
3. Modification d'un élément

Reconnaissance de chaines bruitées

- À ces perturbations on attribue des coûts entiers positifs
  - Disparition : **X(p) → λ**, X(p) étant différent de λ
  - Insertion : **λ → Y(q)**, Y(q) étant différent de **λ**
  - Ssubstitution : **X(p) → Y(q)**, X(p) et Y(q) étant différents de λ
- Si X(p) = Y(q), le coût de substitution vaut 0, alors que si X(p) est différent de Y(q), il y a un coût de substitution non nul.
- On attribue à ces perturbations des coûts entiers positifs. Le coût d'une transformation de X à Y est défini comme étant la somme des coûts locaux des perturbations de caractères qui la composent.
- On cherche à calculer la distance de X à Y, en supposant que les chaînes sont stockées dans des tableaux.

Reconnaissance de chaînes de caractères bruitées

- Ce problème peut être considéré comme un problème de programmation dynamique : pourquoi ?
- Donnez la solution récursive naïve et montrez qu'elle conduit à des calculs redondants

Reconnaissance de chaînes de caractères bruitées

- Distance de Levenshtein

$$LEV\,(X_m, Y_n) = \begin{cases} n & \text{si } \; m = 0 \\ m & \text{si } \; n = 0 \\ \min \begin{cases} LEV(X_{m-1}, Y_n) + 1 \\ LEV(X_n, Y_{n-1}) + 1 \\ LEV(X_{m-1}, Y_{n-1}) + 1 & \text{si } x_m \neq y_n \\ LEV(X_{m-1}, Y_{n-1}) & \text{sinon} \end{cases} \end{cases}$$

- Au final, on obtient une matrice de coûts cumulés D, avec la valeur finale D(N,M) qui exprime la distance entre les 2 chaînes.

TD

- Implémenter les algorithmes : naïf, LCS, Levenshtein pour calculer les « distances » entre les chaînes :

X = CGTGGATCC

Y = CGTTGAATTC

- Complexité de l'algorithme : O(N.M)

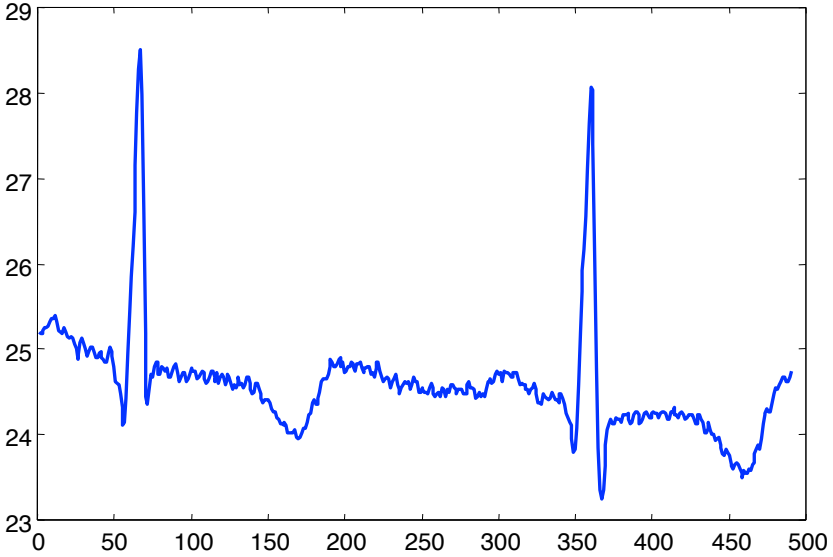
Avec N: longueur de la chaîne ***X***, et M : longueur de la chaîne ***Y***

Applications

- Alignement de chaînes de caractères
  - Détection de sous-séquences ADN
- Reconnaissance de motifs
  - Chaînes de caractères, motifs dans des images,…
- Les catégories d'algorithmes précédents peuvent également s'appliquer à des séries temporelles : X = X(t_i), avec i variant entre 1 à *N*

What are time series

- A time series is a collection of observations made sequentially in time.



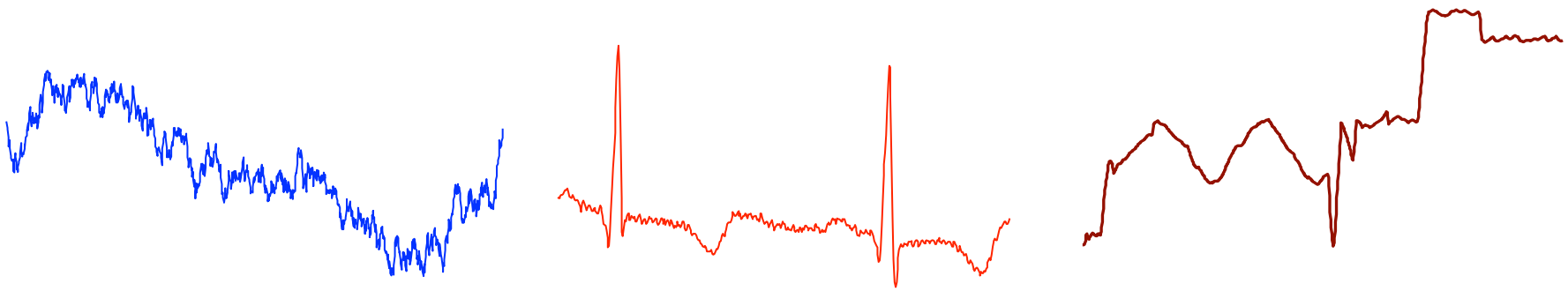
Note that virtually all similarity measurements, indexing and dimensionality reduction techniques discussed in this tutorial can be used with other data types.

What are time series

People measure things... BIG DATA

- *The presidents approval rating.*
- *Their blood pressure.*
- *The annual rainfall in Riverside.*
- *The value of their Yahoo stock.*
- *The number of web hits per second.*

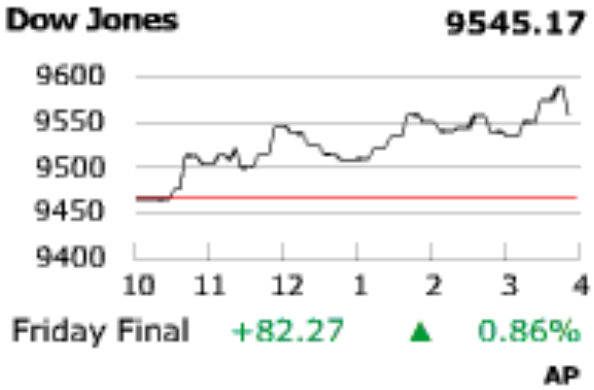
… and things change over time.

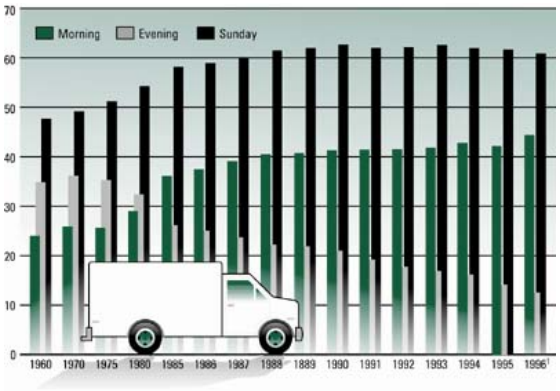


Thus time series occur in virtually every medical, scientific and businesses domain.

Time series are ubiquituous

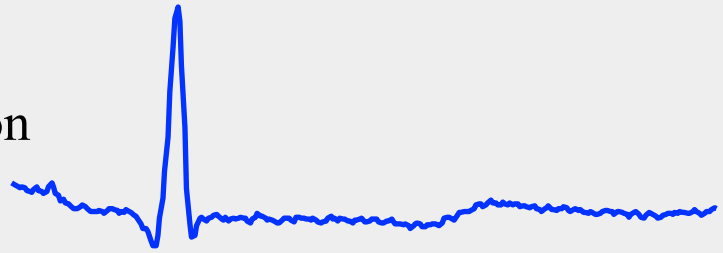
A random sample of 4,000 graphics from 15 of the worlds newspapers published from 1974 to 1989 found that more than 75% of all graphics were time series (Tufte, 1983).



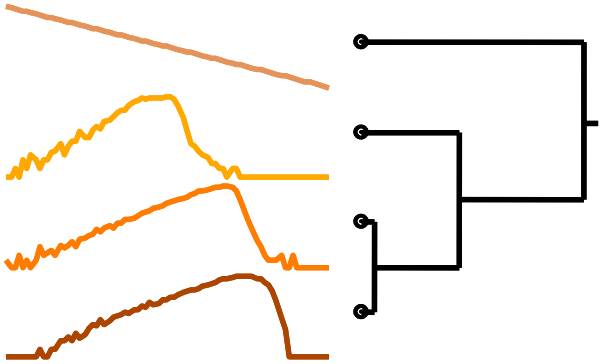


Time series similarities

Classification

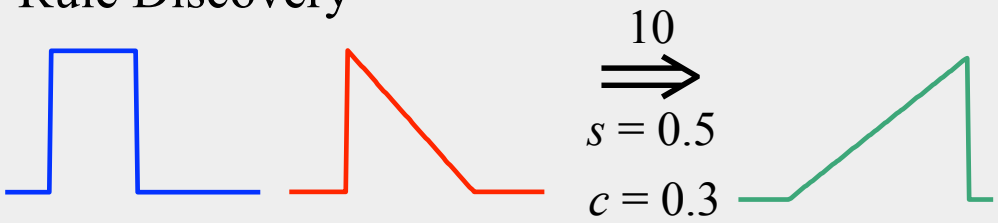


Clustering



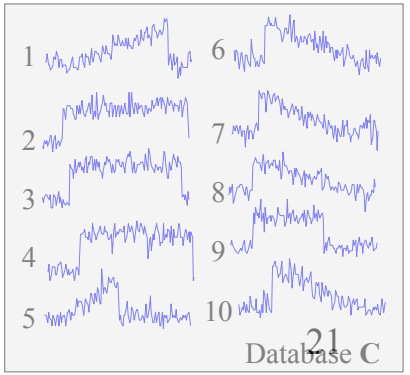
Defining the similarity between two time series is at the heart of most time series data mining applications/ tasks

Rule Discovery



Thus time series similarity will be the primary focus of this tutorial.

Query by Content



What is similarity?

The quality or state of being similar; likeness; resemblance; as, a similarity of features.

Similarity is hard to define, but…
"*We know it when we see it*"

The real meaning of similarity is a philosophical question.

We will take a more pragmatic approach.

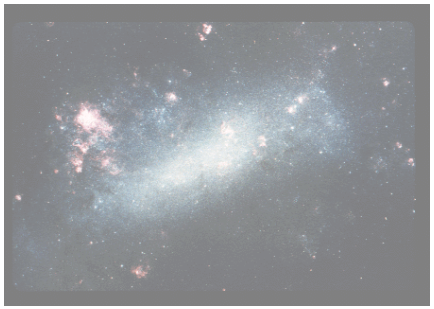
What is similarity?

- **Classification**: *Do other genes express themselves like this gene?*
- **Clustering**: *Grouping robot experiences*.
- **Association Rules:** *Peak followed plateau implies a downward trend with a confidence of 0.4 and a support of 0.2.*
- **Exploratory Data Analysis**: *Understanding the data by interacting with it.*

Why is Working With Time Series so Difficult? Part I

Answer: How do we work with very large databases?

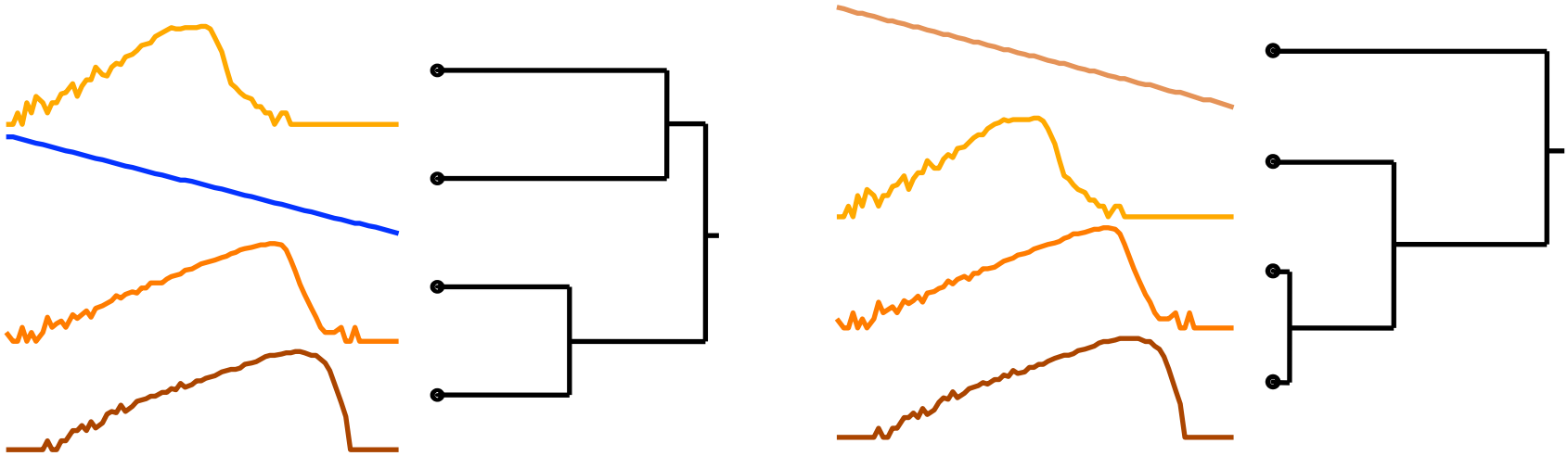
- 1 Hour of ECG (electrocardiagram) data: 1 Gigabyte.
- Typical Weblog: 5 Gigabytes per week.
- Space Shuttle Database: 158 Gigabytes and growing.
- Macho Database: 2 Terabytes, updated with 3 gigabytes per day
  (astronomical database of the intensities of about 20 million stars, recorded approximately every night for several years)



Since most of the data lives on disk, we need a representation of the data we can efficiently manipulate.

Why is Working With Time Series so Difficult? Part II

Answer: We are dealing with subjective notions of similarity



The definition of similarity depends on the user, the domain and the task at hand. We need to be able to handle this subjectivity.

Why is Working With Time Series so Difficult? Part III

Answer: Miscellaneous data handling problems.

- Differing data formats.
- Differing sampling rates.
- Noise, missing values, etc.

Two motivating Datasets

Cylinder-Bell-Funnel

$c(t) = (6+\eta)\ X_{[a,b]}(t) + \varepsilon(t)$

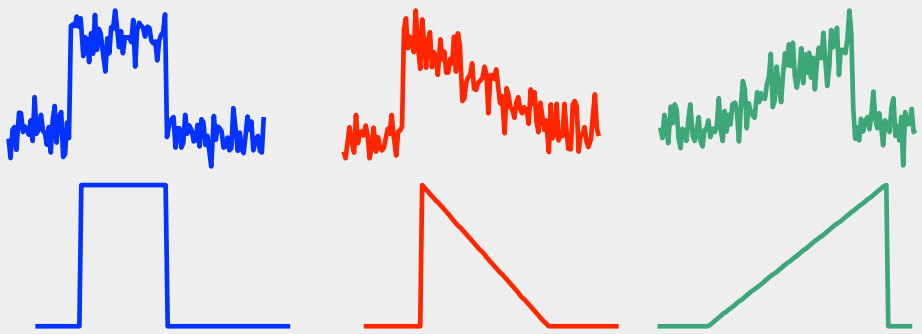
$b(t) = (6+\eta)\ X_{[a,b]}(t)\ (t-a)/(b-a) + \varepsilon(t)$

$f(t) = (6+\eta)\ X_{[a,b]}(t)\ (b-a)/(b-t) + \varepsilon(t)$

$X_{[a,b]} = \{\ 1, \text{ if } a \le t \le b, \text{ else } 0\ \}$

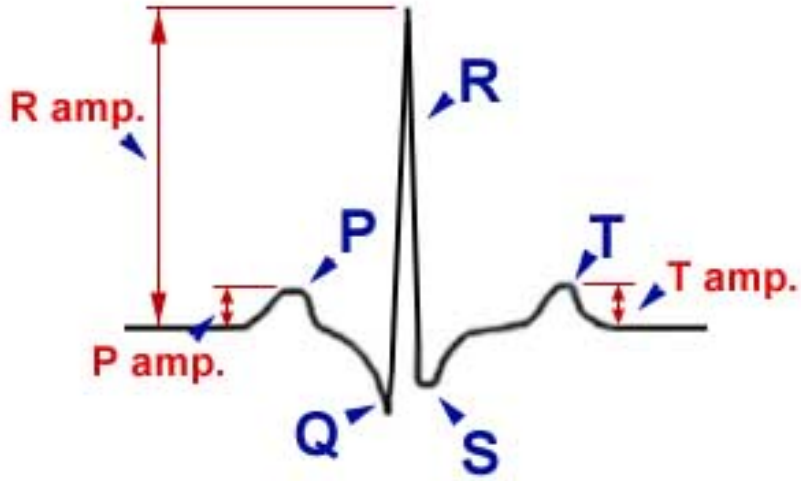
Where η and ε(t) are drawn from a standard normal distribution $N(0,1)$, a is an integer drawn uniformly from the range [16,32] and $(b-a)$ is an integer drawn uniformly from the range [32, 96].

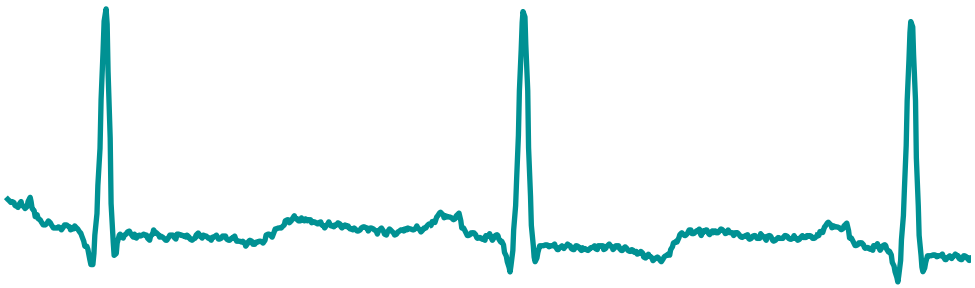
Kadous 1999; Manganaris,1997; Saito 1994; Rodriguez 2000; Geurts 2001



Cylinder Funnel Bell

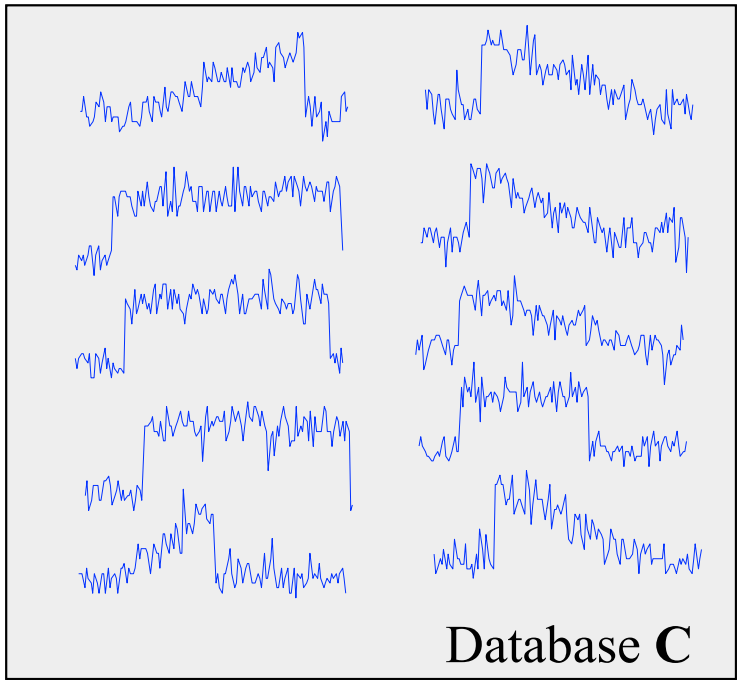
Electrocardiogram (ECGs)



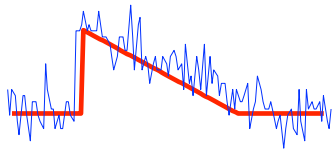


The similarity matching problem can come in two flavors I

Query Q
(template)



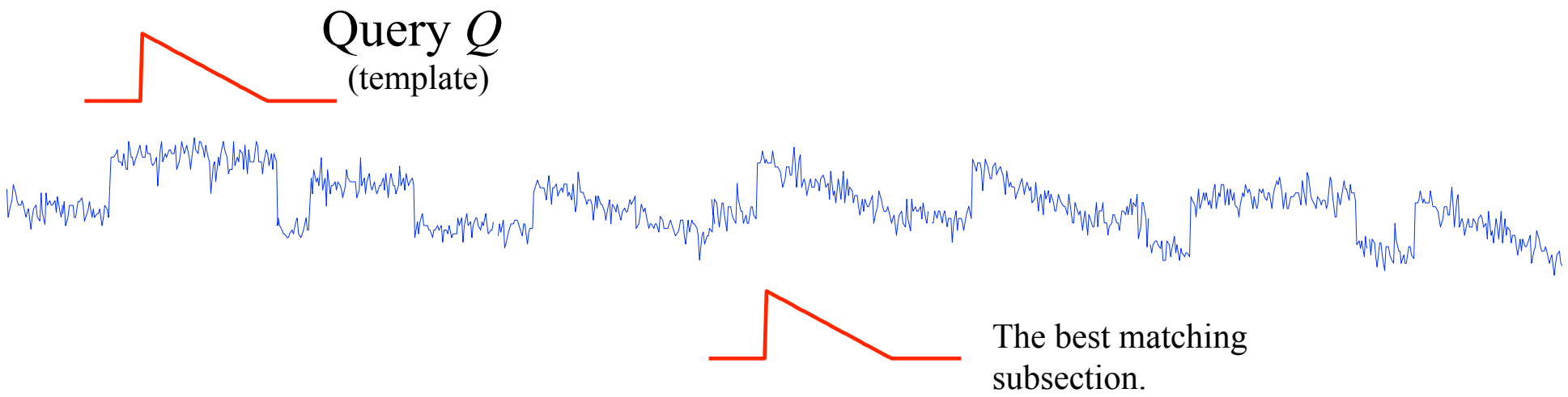
1: Whole Matching



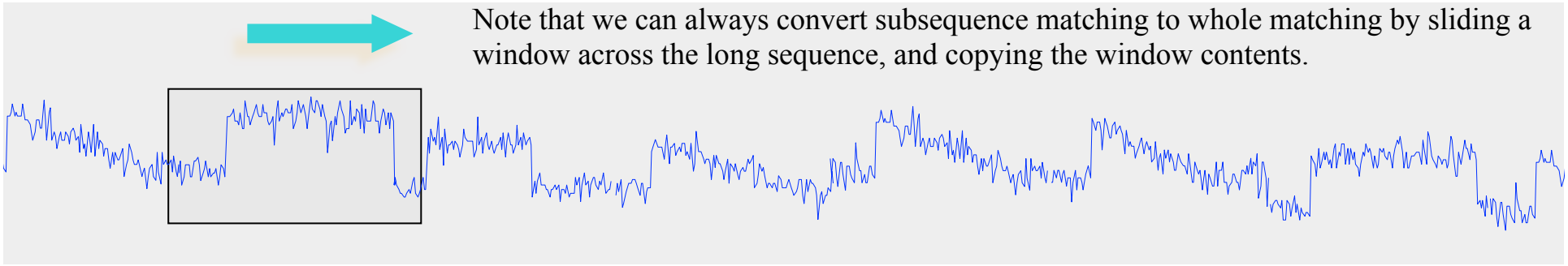
C_6 is the best match.

Given a Query Q, a reference database **C** and a distance measure, find the C_i that best matches Q.

The similarity matching problem can come in two flavors II



Given a Query Q, a reference database **C** and a distance measure, find the location that best matches Q.



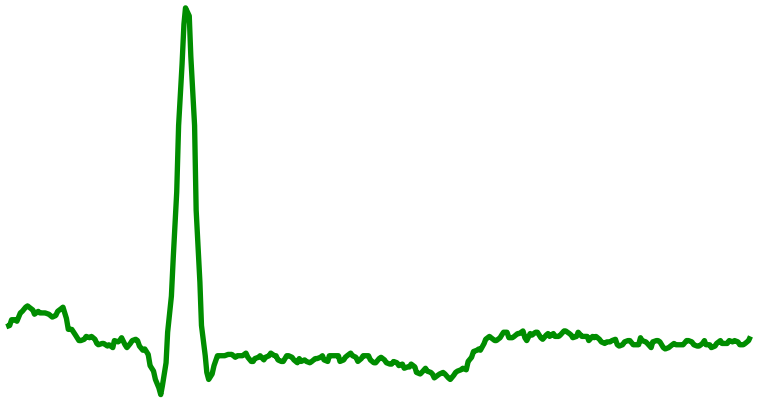
Motivations…

You go to the doctor because of chest pains. Your ECG looks strange…

You doctor wants to search a database to find similar ECGS, in the hope that they will offer clues about your condition...

Two questions:

- How do we define similar?
- How do we search quickly?



Defining Distance Measures

Definition: Let O_1 and O_2 be two objects from the universe of possible objects. The distance (dissimilarity) is denoted by $D(O_1,O_2)$

What properties should a distance measure have?

- $D(A,B) = D(B,A)$ *Symmetry*
- $D(A,A) = 0$ *Constancy of Self-Similarity*
- $D(A,B) = 0$ IIf A= B *Positivity*
- $D(A,B) \leq D(A,C) + D(B,C)$ *Triangular Inequality*

Intuitions behind desirable distance measure properties

$D(A,B) = D(B,A)$ *Symmetry*

Otherwise you could claim "Alex looks like Bob, but Bob looks nothing like Alex."

$D(A,A) = 0$ *Constancy of Self-Similarity*

Otherwise you could claim "Alex looks more like Bob, than Bob does."

$D(A,B) = 0$ IIf A=B *Positivity*

Otherwise there are objects in your database that are different, but you cannot tell apart.

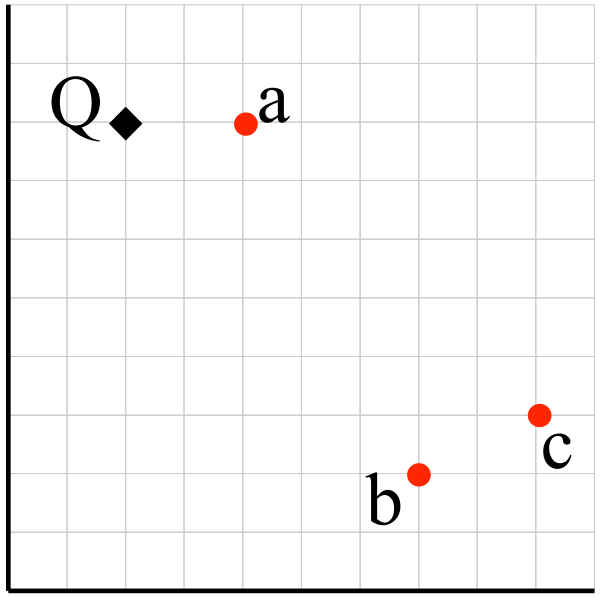
$D(A,B) \leq D(A,C) + D(B,C)$ *Triangular Inequality*

Otherwise you could claim "Alex is very like Bob, and Alex is very like Carl, but Bob is very unlike Carl."

Why is the Triangular Inequality so Important?

Virtually all techniques to index data require the triangular inequality to hold.

Suppose I am looking for the closest point to Q, in a database of 3 objects.
Further suppose that the triangular inequality holds, and that we have precompiled a table of distance between all the items in the database.



I find **a** and calculate that it is 2 units from Q, it becomes my *best-so-far*. I find **b** and calculate that it is 7.81 units away from Q.

I don't have to calculate the distance from Q to **c**!

I know $D(Q,\mathbf{b}) \leq D(Q,\mathbf{c}) + D(\mathbf{b},\mathbf{c})$

$D(Q,\mathbf{b}) - D(b,\mathbf{c}) \leq D(Q,\mathbf{c})$

$7.81 - 2.30 \leq D(Q,\mathbf{c})$

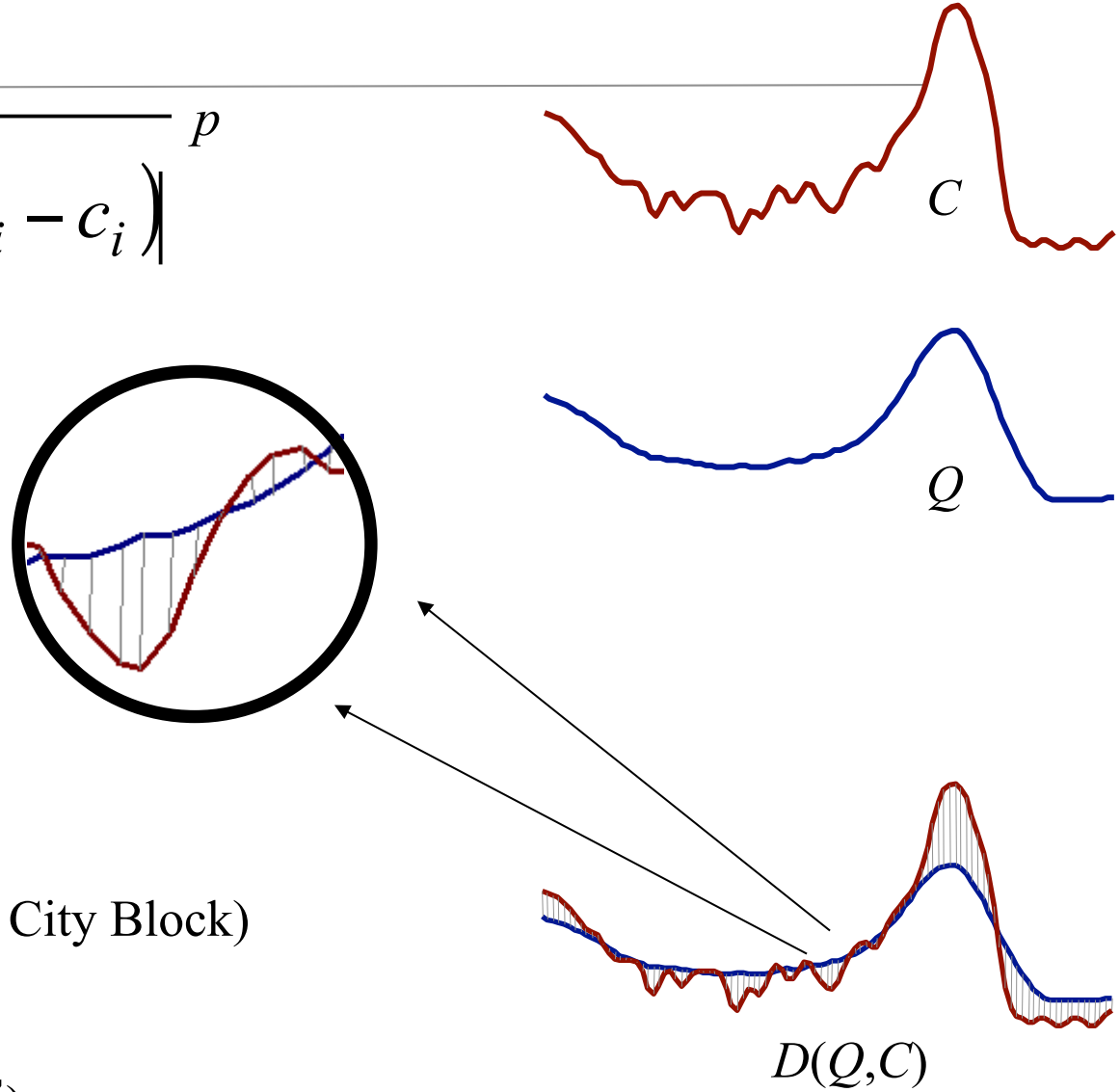
$5.51 \leq D(Q,\mathbf{c})$

So I know that **c** is at least 5.51 units away, but my *best-so-far* is only 2 units away.

	a	b	c
a		6.70	7.07
b			2.30
c			

The Minkowski Metrics

$$D(Q,C) \equiv \sqrt[p]{\sum_{i=1}^{n} \left|(q_i - c_i)\right|^p}$$



$p = 1$ Manhattan (Rectilinear, City Block)

$p = 2$ Euclidean

$p = \infty$ Max (Supremum, "sup")

Euclidian Distance Metrics

Given two time series

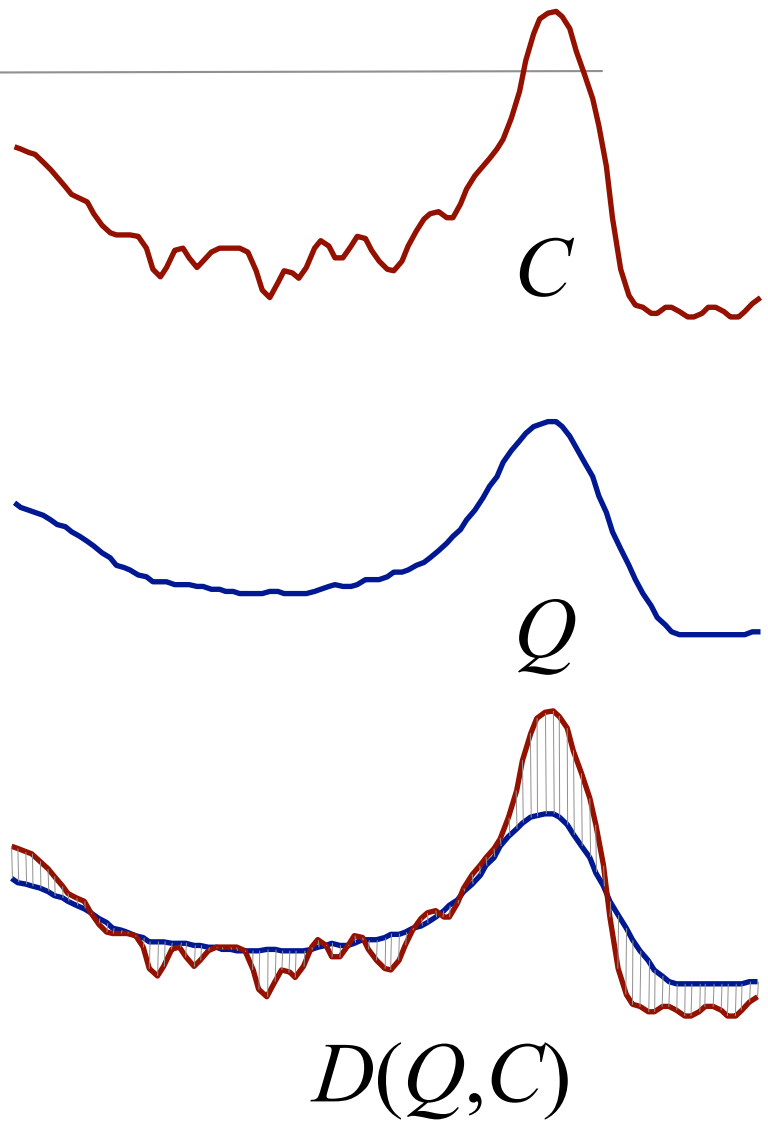
$$Q = q_1 \ldots q_n$$

and

$$C = c_1 \ldots c_n$$

their Euclidean distance is defined as:

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2}$$



Optimizing the Euclidian Distance Calculation

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$$

$$D_{squared}(Q,C) \equiv \sum_{i=1}^{n}(q_i - c_i)^2$$

Euclidean distance and Squared Euclidean distance are equivalent in the sense that they return the same rankings, clusterings and classifications.

Instead of using the
Euclidean distance
we can use the
Squared Euclidean distance

This optimization helps with CPU time, but most problems are I/O bound.

Preprocessing the data before distance calculation

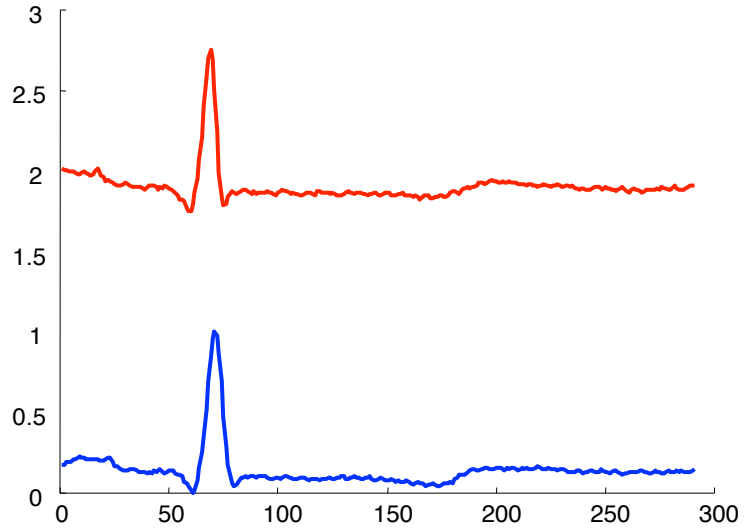
If we naively try to measure the distance between two "raw" time series, we may get very unintuitive results.

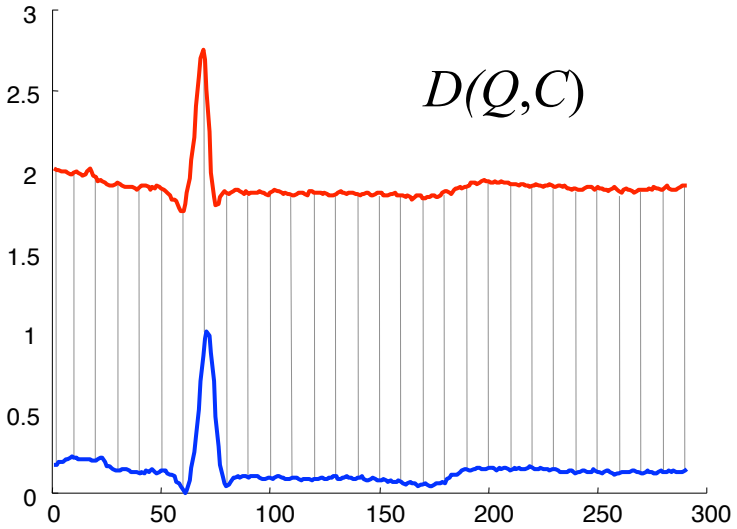
This is because Euclidean distance is very sensitive to some distortions in the data. For most problems these distortions are not meaningful, and thus we can and should remove them.

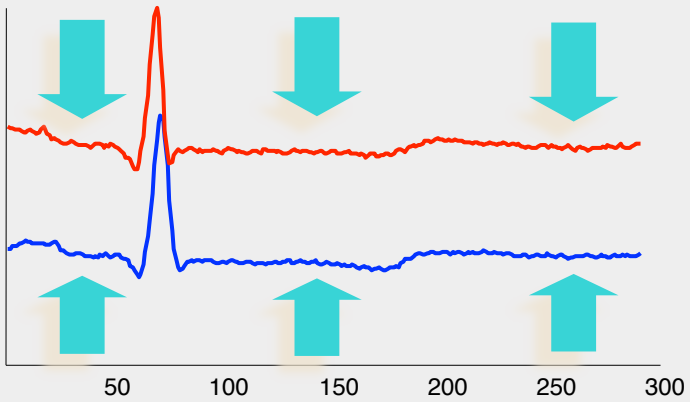
In the next 4 slides I will discuss the 4 most common distortions, and how to remove them.

- Offset Translation
- Amplitude Scaling
- Linear Trend
- Noise

Transformation I: Offset Translation



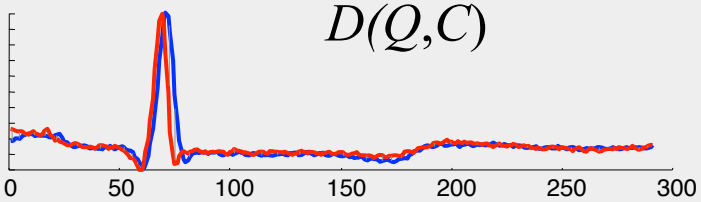




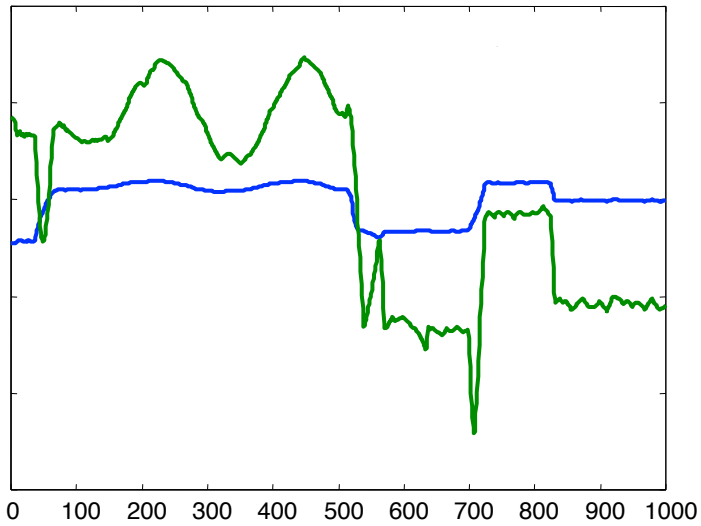
$Q = Q$ - mean(Q)

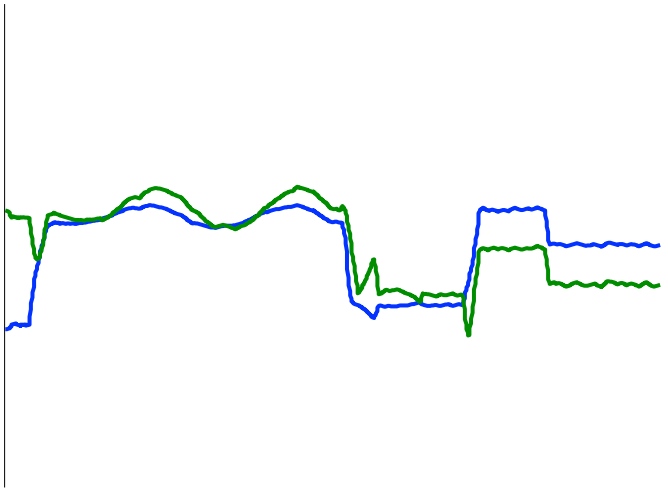
$C = C$ - mean(C)

$D(Q,C)$



Transformation II: Amplitude Scaling



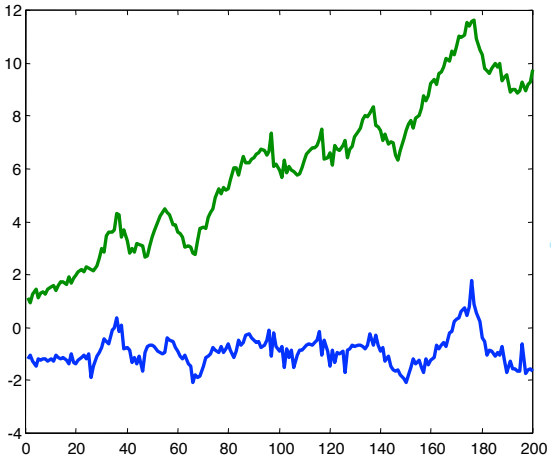


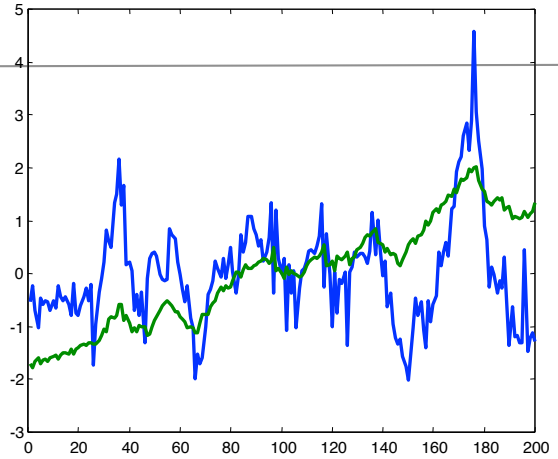
$$Q = (Q - \text{mean}(Q)) / \text{std}(Q)$$

$$C = (C - \text{mean}(C)) / \text{std}(C)$$

$$D(Q,C)$$

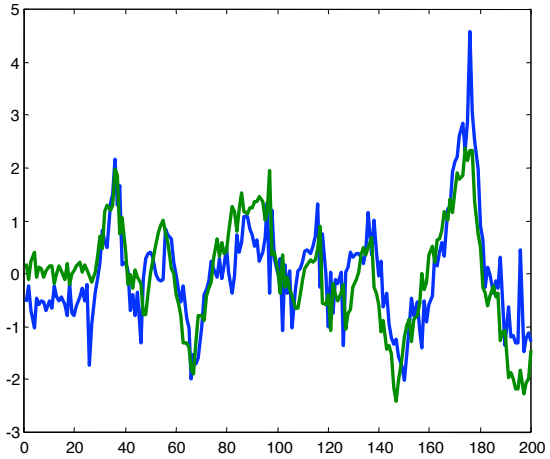
Transformation III: Linear Trend





Removed offset translation

Removed amplitude scaling



Removed **linear trend**

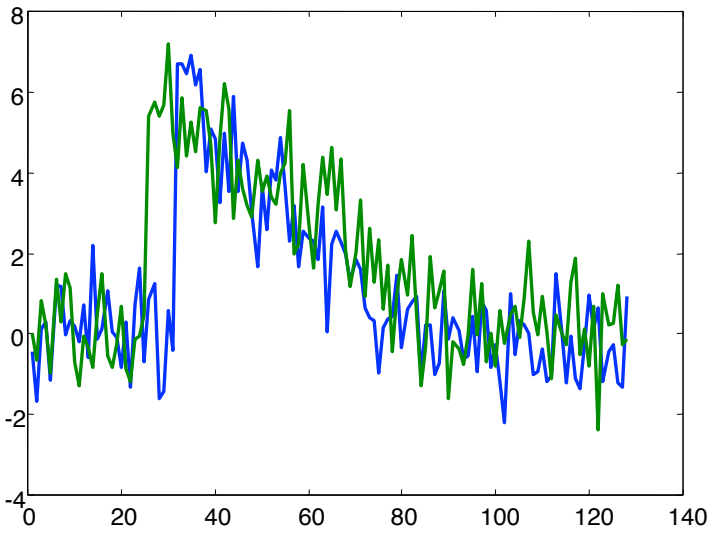
Removed offset translation

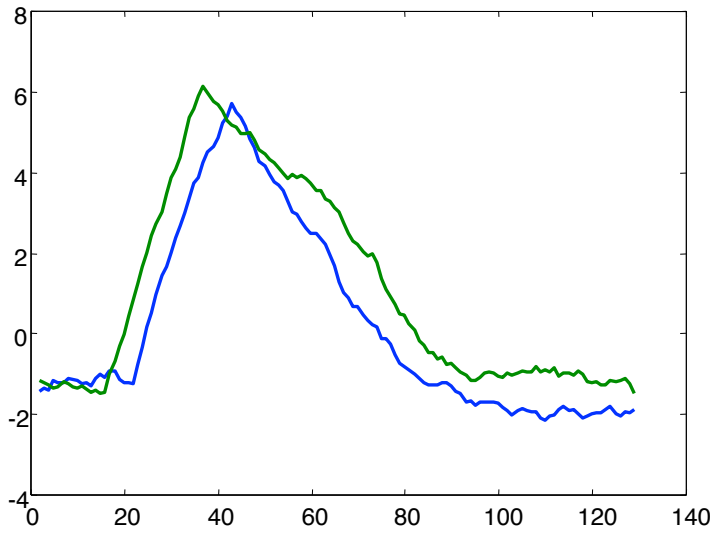
Removed amplitude scaling

The intuition behind removing linear trend is this.

Fit the best fitting straight line to the time series, then subtract that line from the time series.

Transformation IV: Noise





The intuition behind removing noise is this.

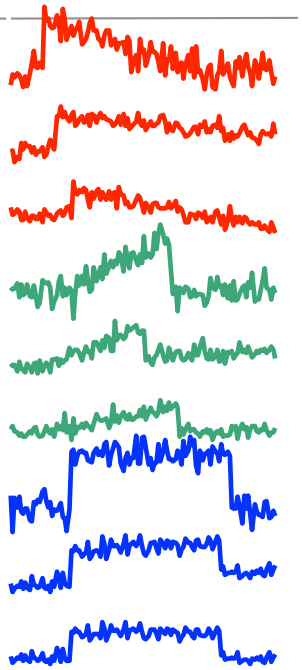
Average each data points value with its neighbors.

$Q = \text{smooth}(Q)$

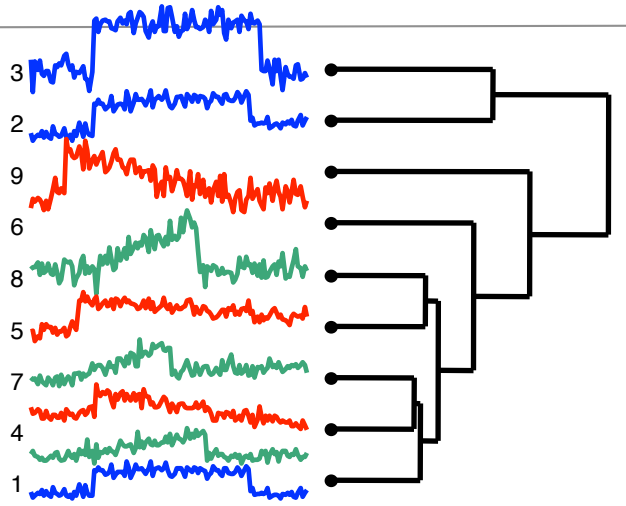
$C = \text{smooth}(C)$

$D(Q,C)$

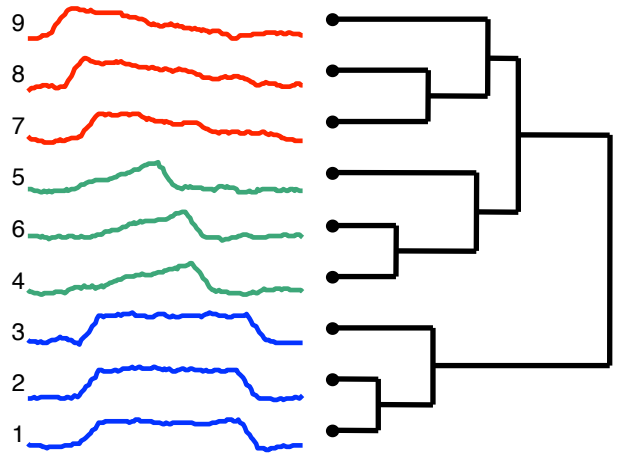
A Quick Experiment to Demonstrate the Utility of Preprocessing the Data



Instances from Cylinder-Bell-Funnel with small, random amounts of trend, offset and scaling added.



Clustered using Euclidean distance on the raw data



Clustered using Euclidean distance on the raw data, after removing noise, linear trend, offset translation and amplitude scaling.

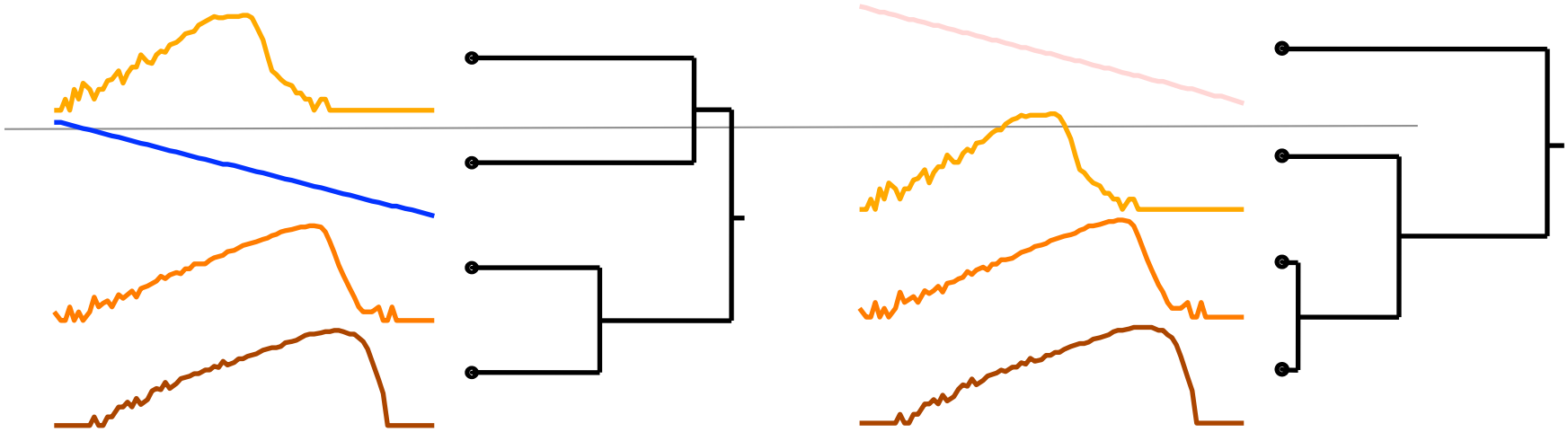
Summary of Preprocessing

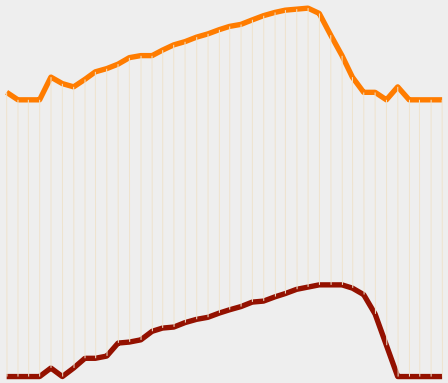
The "raw" time series may have distortions which we should remove before clustering, classification etc.

Of course, sometimes the distortions are the most interesting thing about the data, the above is only a general rule.

We should keep in mind these problems as we consider the high level representations of time series which we will encounter later (Fourier transforms, Wavelets etc). Since these representations often allow us to handle distortions in elegant ways.

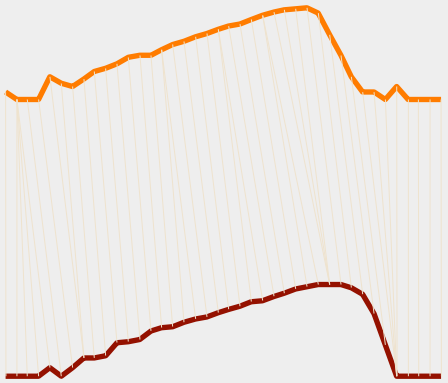
Dynamic Time Warping





Fixed Time Axis

Sequences are aligned "one to one".

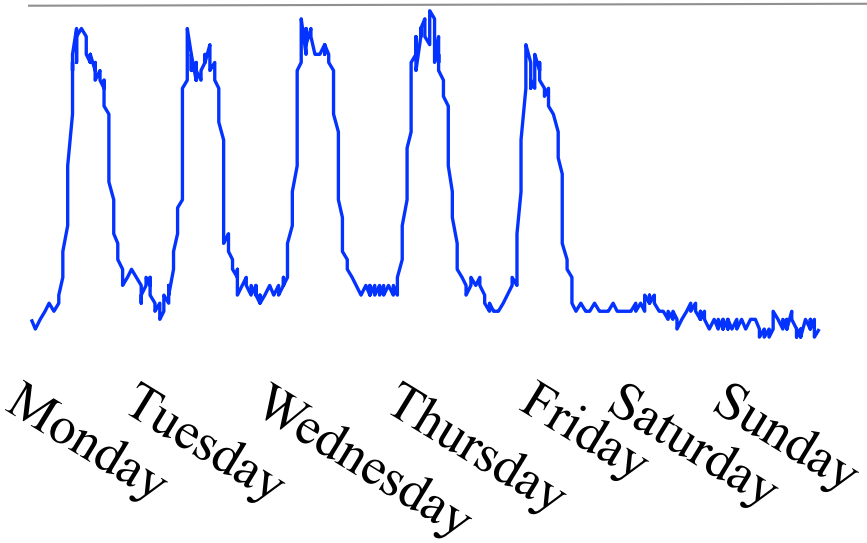


"Warped" Time Axis

Nonlinear alignments are possible.

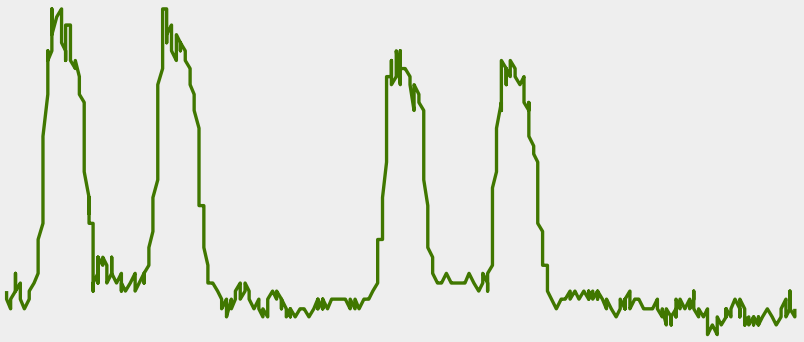
Note: We will first see the utility of DTW, then see how it is calculated.

Utility of Dynamic Time Warping: Example II, Data Mining



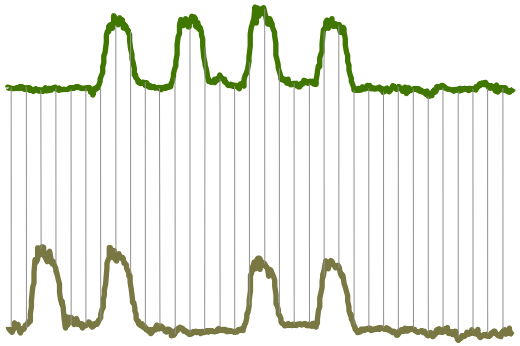
Power-Demand Time Series. Each sequence corresponds to a week's demand for power in a Dutch research facility in 1997 [van Selow 1999].

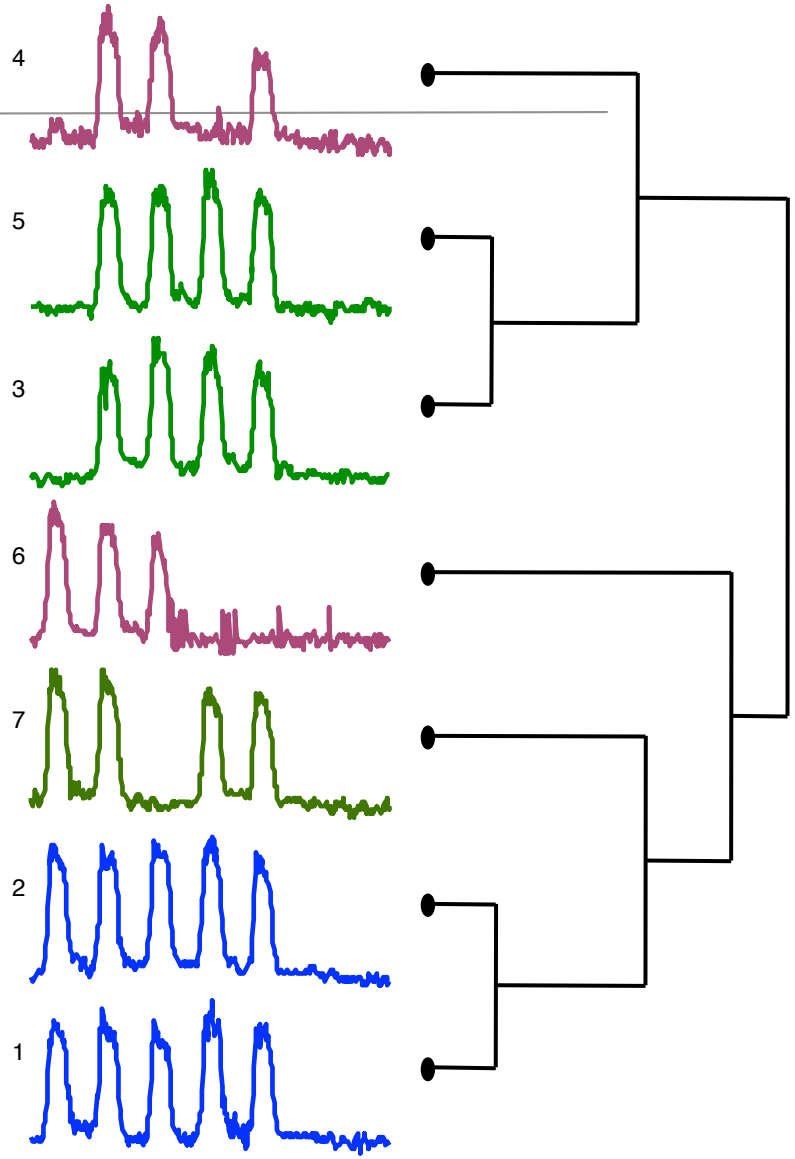
Wednesday was a national holiday



Hierarchical Clustering with Euclidean Distance.

<Group Average Linkage>





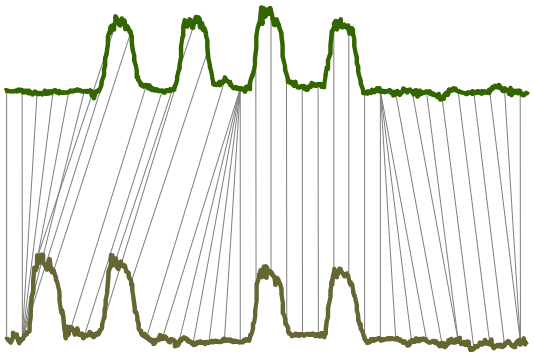
The two **5-day weeks** are correctly grouped.

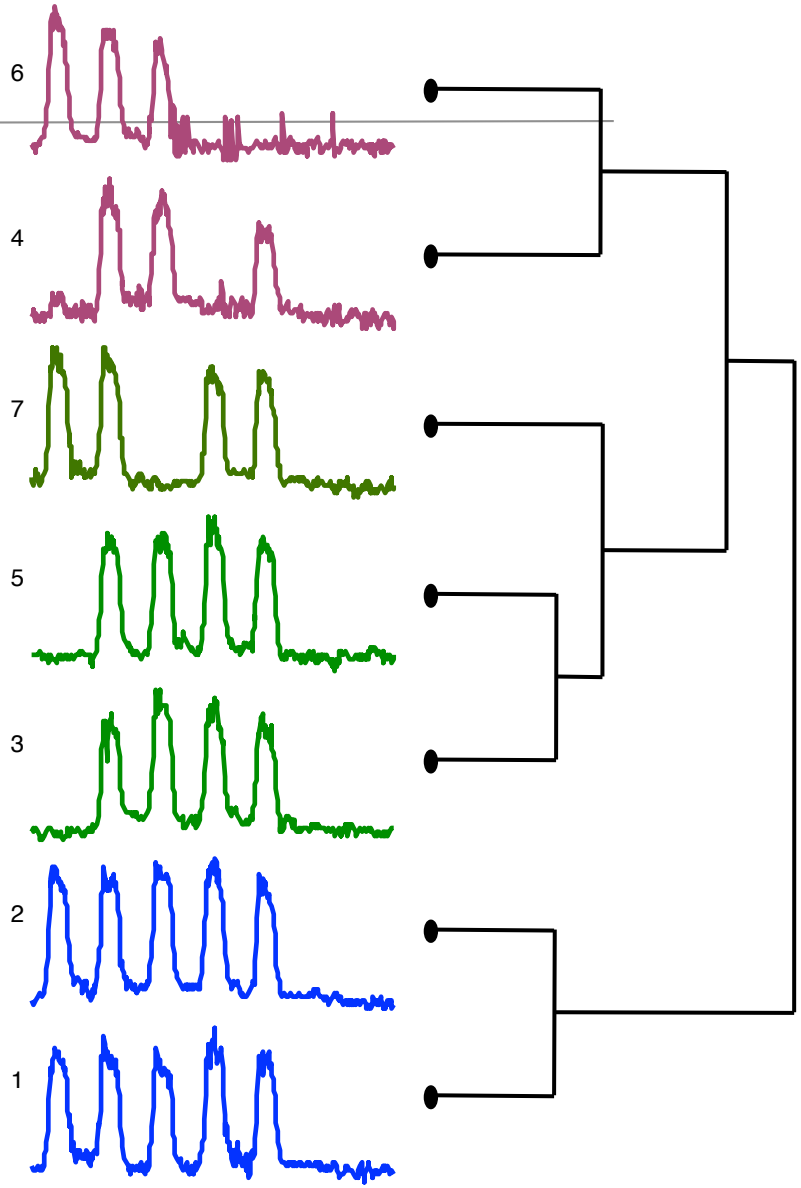
Note however, that the three **4-day weeks** are not clustered together.

Also, the two **3-day weeks** are also not clustered together.

Hierarchical Clustering with Dynamic Time Warping.

<Group Average Linkage>





The two **5-day weeks** are correctly grouped.

The three **4-day weeks** are clustered together.

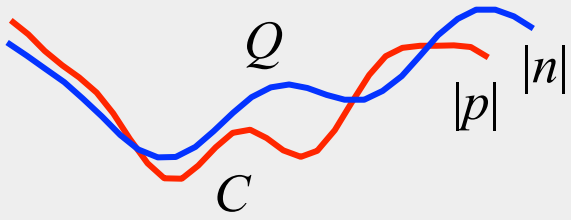
The two **3-day weeks** are also clustered together.

Time taken to create hierarchical clustering of power-demand time series

- Time to create dendrogram
  using Euclidean Distance 1.2 seconds

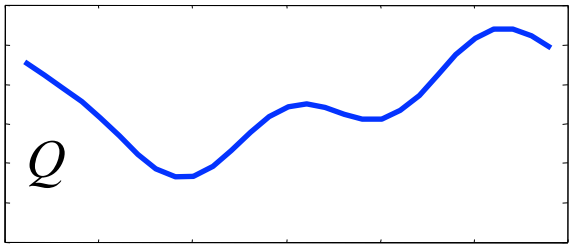
- Time to create dendrogram
  using Dynamic Time Warping 3.40 hours

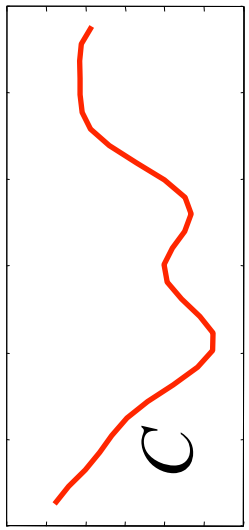
Computing the Dynamic Time Warp Distance

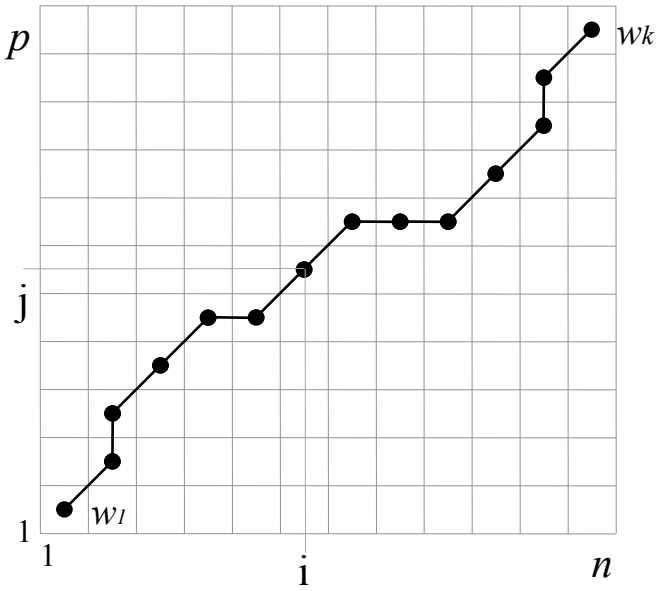


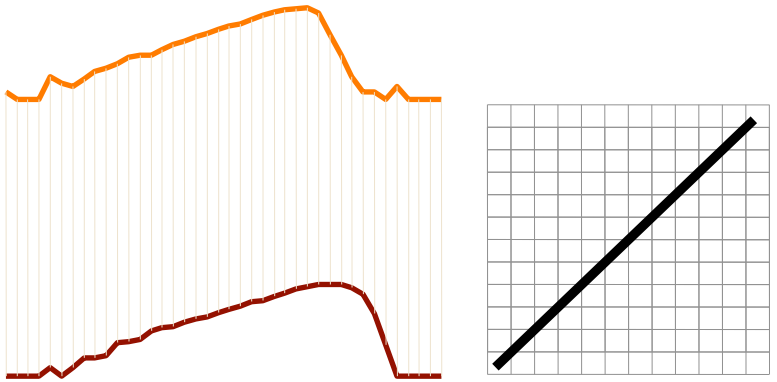
Note that the input sequences can be of

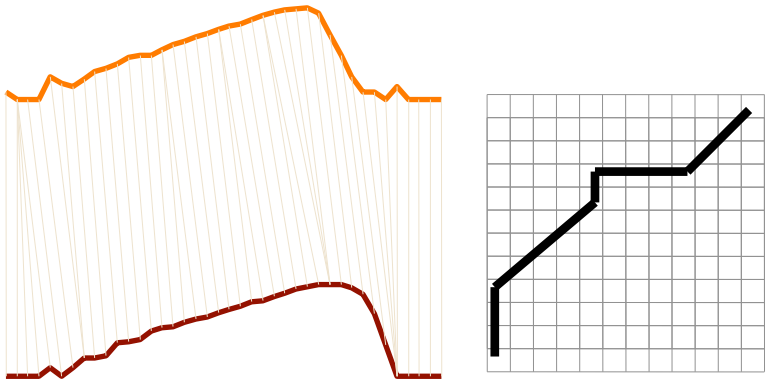
different lengths





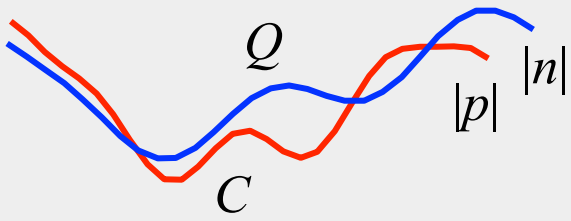


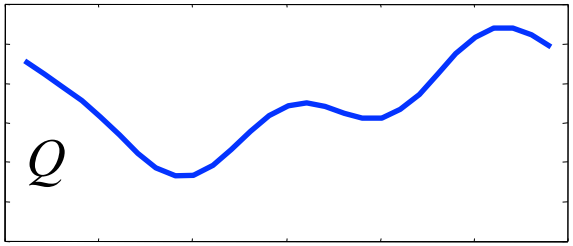


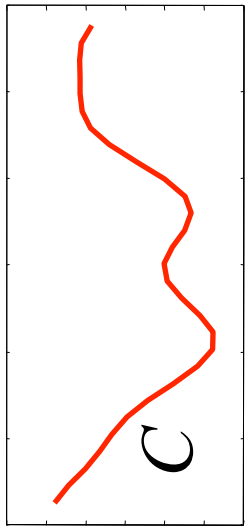


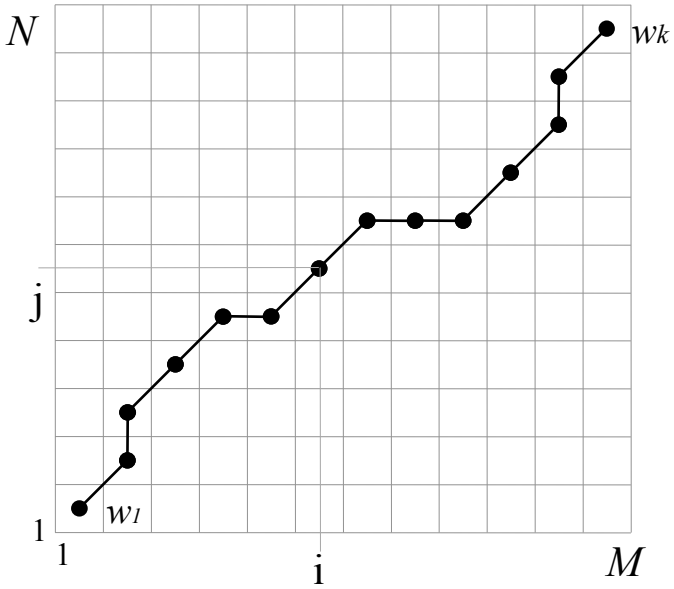
M. Müller

Computing the Dynamic Time Warp Distance









Every possible mapping from Q to C can be represented as a warping path in the search matrix.

We simply want to find the cheapest one…

$$DTW(Q,C) = \min\left\{ \sqrt{\sum_{k=1}^{K} w_k} \Big/ K \right\}$$

Although there are exponentially many such paths, we can find one in only quadratic time using dynamic programming.

$$D(i,j) = c(q_i,c_j) + \min\{ D(i-1,j-1) , D(i-1,j) , D(i,j-1) \}$$

Cost matrix C:

$$C(n,m) = c(x_n, y_m)$$

Dynamic programming:
Goal: find an alignment between X and Y having minimum overall cost

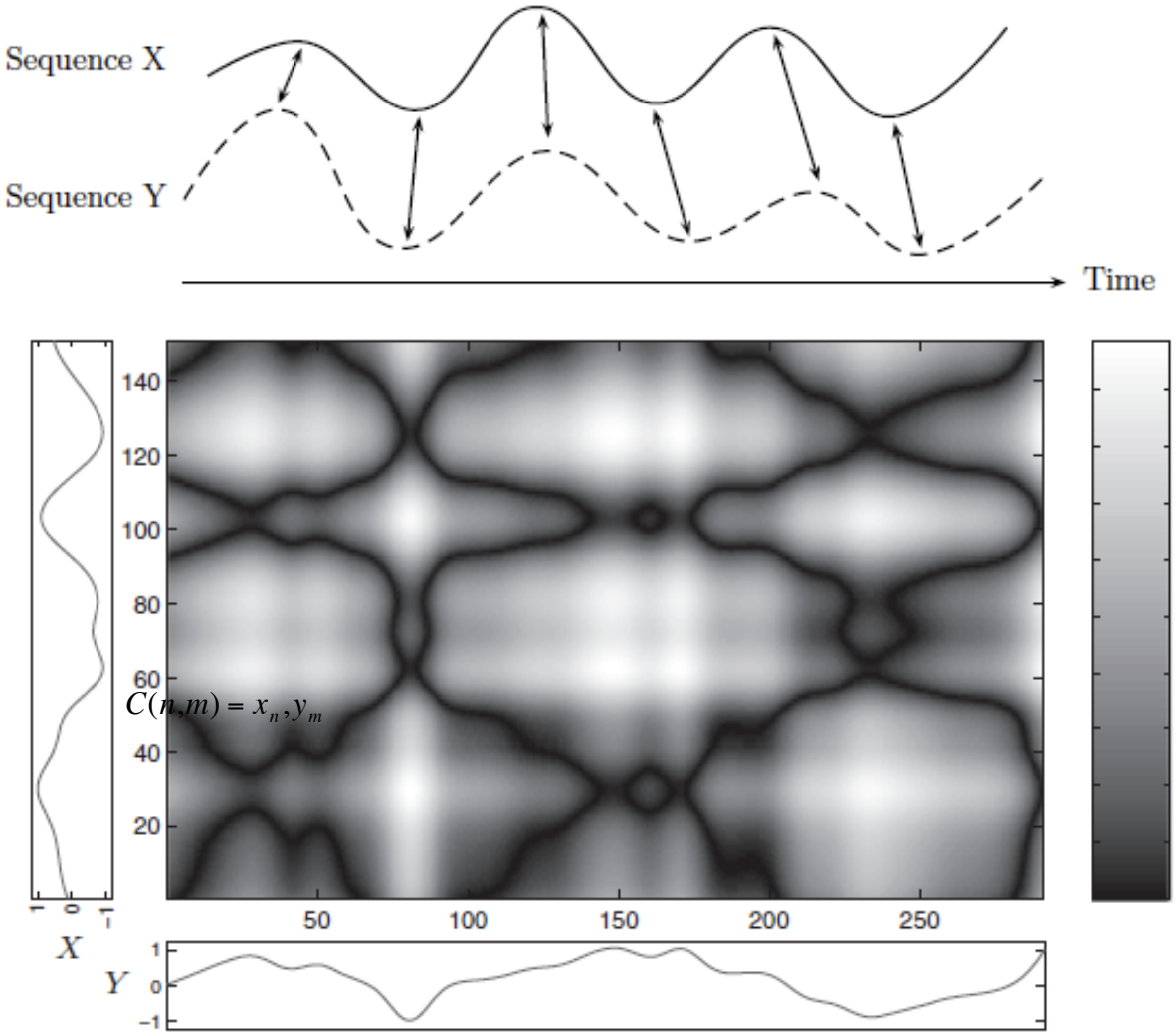


Fig. 4.2. Cost matrix of the two real-valued sequences X (*vertical axis*) and Y (*horizontal axis*) using the Manhattan distance (absolute value of the difference) as local cost measure c. Regions of low cost are indicated by *dark colors* and regions of high cost are indicated by *light colors*

Algorithme récursif

$$D(i,j) = c(x_i,y_j) + \min\{ D(i-1,j-1) , D(i-1,j) , D(i,j-1) \}$$

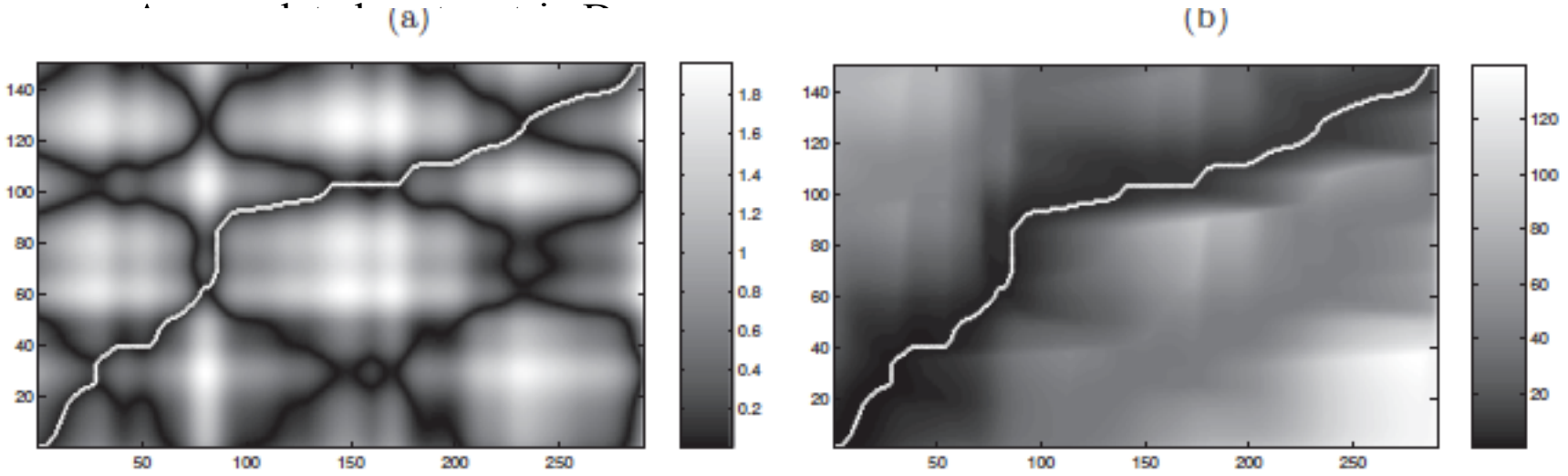


Fig. 4.4. (a) Cost matrix C as in Fig. 4.2 and (b) accumulated cost matrix D with optimal warping path p^* (*white line*)

Algorithm: OPTIMALWARPINGPATH

Input: Accumulated cost matrix D.
Output: Optimal warping path p^*.

Procedure: The optimal path $p^* = (p_1, \ldots, p_L)$ is computed in reverse order of the indices starting with $p_L = (N, M)$. Suppose $p_\ell = (n, m)$ has been computed. In case $(n, m) = (1, 1)$, one must have $\ell = 1$ and we are finished. Otherwise,

$$p_{\ell-1} := \begin{cases} (1, m-1), & \text{if } n = 1 \\ (n-1, 1), & \text{if } m = 1 \\ \text{argmin}\{D(n-1, m-1), & \\ \qquad D(n-1, m), D(n, m-1)\}, & \text{otherwise,} \end{cases} \tag{4.6}$$

where we take the lexicographically smallest pair in case "argmin" is not unique.

Step size condition

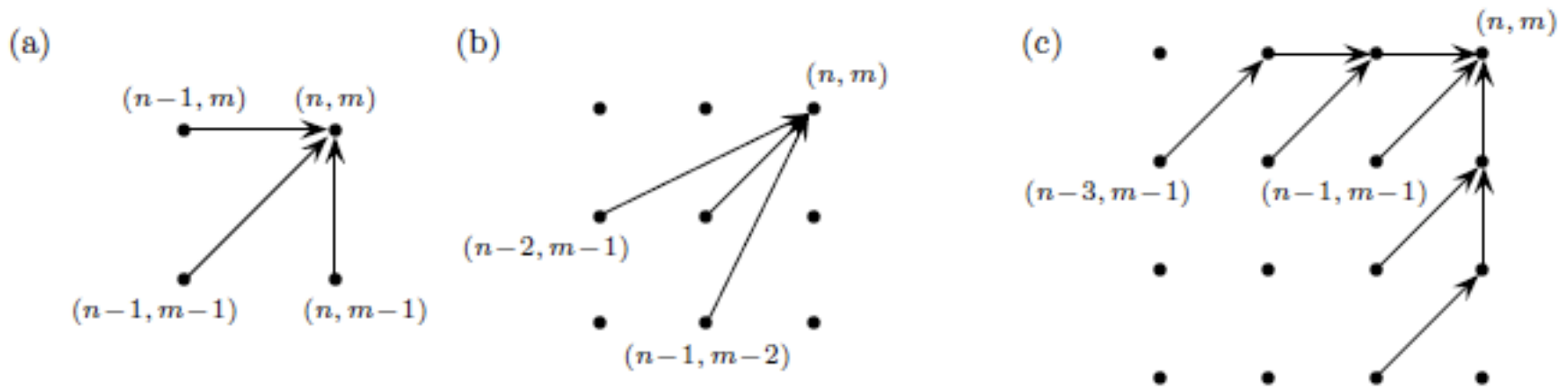
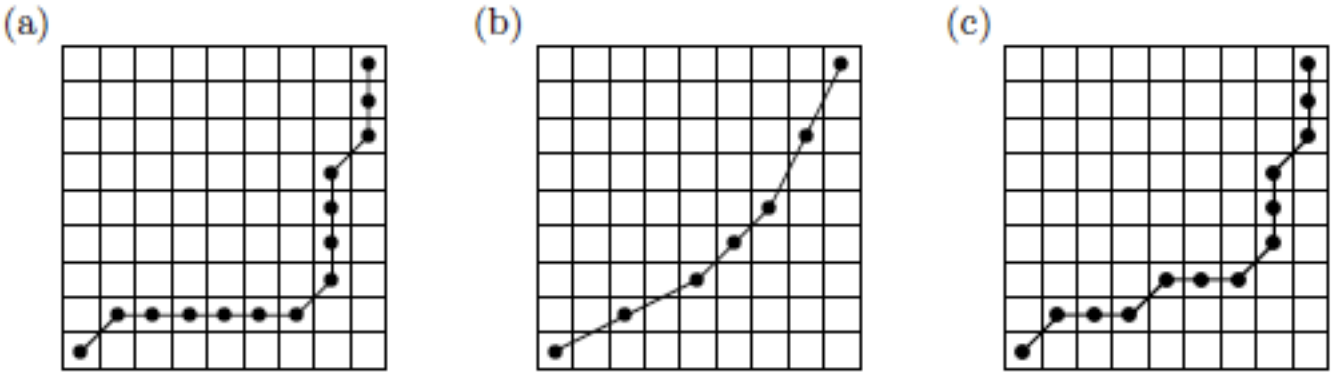


Fig. 4.5. Illustration of three different step size conditions, which express different local constraints on the admissible warping paths. (a) corresponds to the step size condition (iii) of Definition 4.1

$$D(n,m) = \min \begin{cases} D(n-1,m-1) + c(x_n, y_m) \\ D(n-2,m-1) + c(x_{n-1}, y_m) + c(x_n, y_m) \\ D(n-1,m-2) + c(x_n, y_{m-1}) + c(x_n, y_m) \\ D(n-3,m-1) + c(x_{n-2}, y_m) + c(x_{n-1}, y_m) + c(x_n, y_m) \\ D(n-1,m-3) + c(x_n, y_{m-2}) + c(x_n, y_{m-1}) + c(x_n, y_m) \end{cases}$$

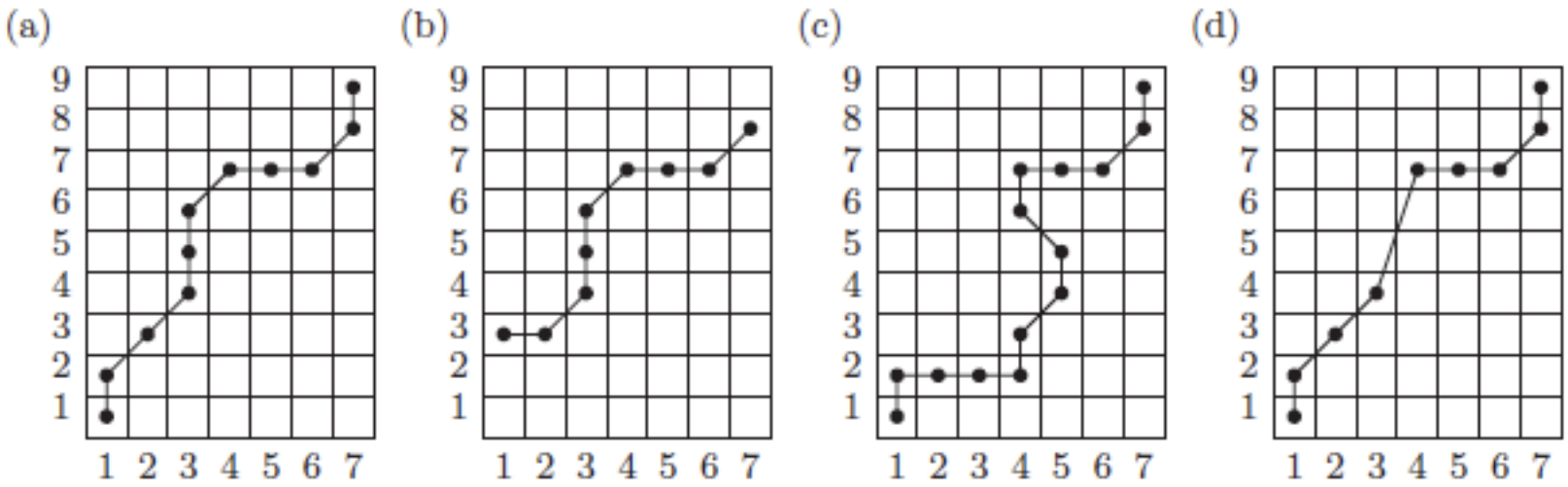


Warping pass

a) Degeneration of the warping path

b) Omission of elements in alignment of X and Y

c) Warping path with respect to figure 4.5.c)



Warping pass

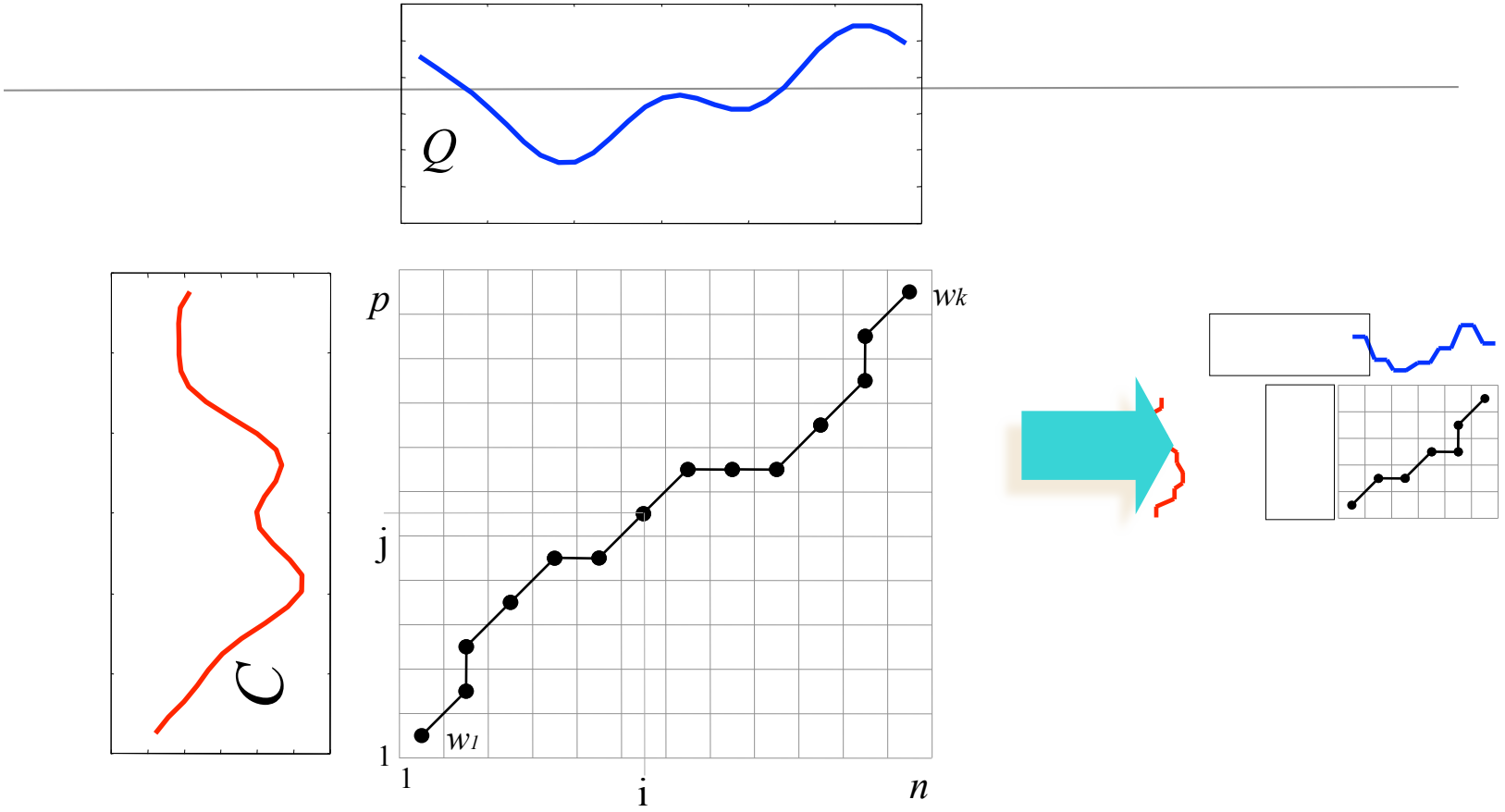
a) Admissible pass

b) Boundary condition is violated w1=(1,1), wK=(N,M)

c) Monotonicity condition is violated

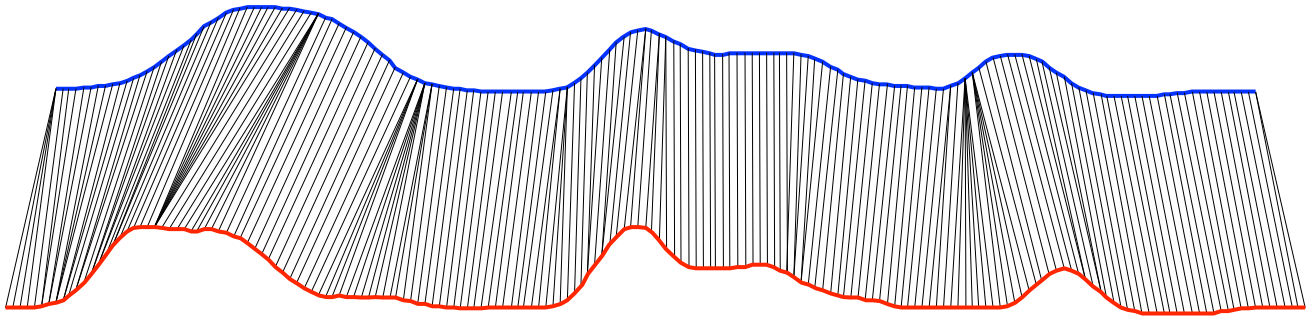
d) Step size condition is violated

Fast Approximations to Dynamic Time Warp Distance I

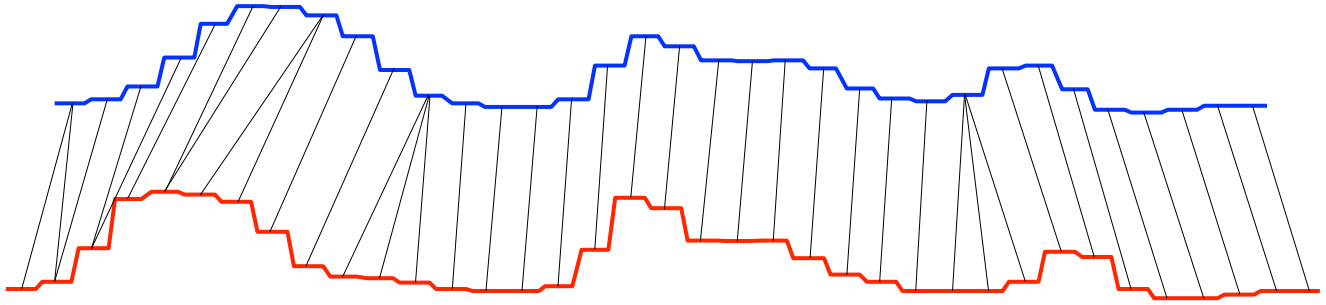


Simple idea: Approximate the time series with some compressed or downsampled representation, and do DTW on the new representation.

Fast Approximations to Dynamic Time Warp Distance II



22.7 sec



1.3 sec

.. strong visual evidence to suggests it works well.

Good experimental evidence the utility of the approach on clustering, classification and query by content problems also has been demonstrated.

Dynamic Time Warp is not a metric

- The triangular inequality does not hold for DTW
- An extension has been proposed that is a metric:
  - TWED « Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching » P.F. Marteau (2006, 2008, ...)

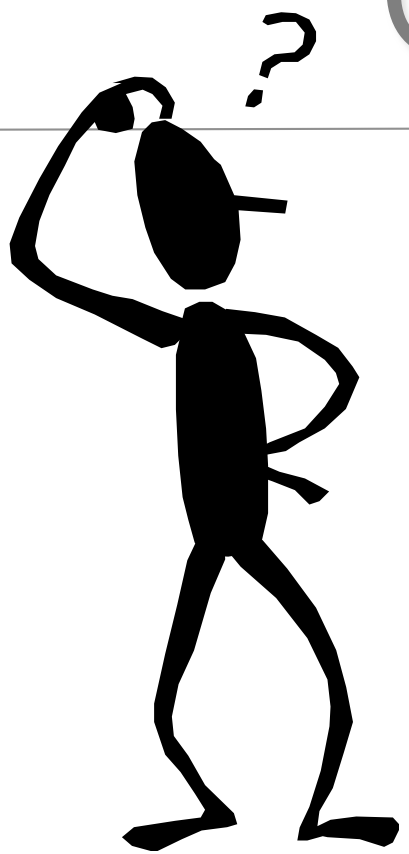
  https://en.wikipedia.org/wiki/Time_Warp_Edit_Distance
  - Elastic distances with kernels methods (classification)

Applications

- Dans le cas des séries temporelles, l'algorithme est la plupart du temps appliqué avec des signaux échantillonnés de manière régulière
- Le signal temporel X peut être multidimensionnel

- Reconnaissance de parole
- Transfert de style (mouvement)
- Chemins dans une image

Questions?



Thanks to

- Eamon Keogh
- Meinard Müller (Information Retrieval for Music and Motion)

Visit the Time Series Data Mining Archive for code, datasets, papers and pointers

http://www.cs.ucr.edu/~eamonn/