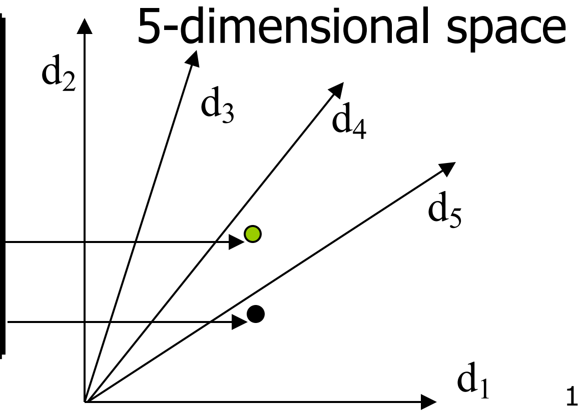
Dense Multidimensional Data

- Similarity search in high-dimensional spaces
- Motivating applications
- Indexing issues and the curse of dimensionality
- Indexing based on dimensionality reduction
- Indexing based on compression
- Indexing metric spaces

Projection of x Load	Projection of y load	Distance	Load	Thickness
10.23	5.27	15.22	2.7	1.2
12.65	6.25	16.22	2.2	1.1



Similarity Search and Applications

Dense Multidimensional Data

- Record data representation
  - Each column is an attribute, each row is an object, all objects have values in all attributes
    - assume interval-scaled attributes only

	Projection of x Load	Projection of y load	Distance	Load	Thickness
o_1	10.23	5.27	15.22	2.7	1.2
o_2	12.65	6.25	16.22	2.2	1.1

- Distance computation using a Minkowski Distance measure
  - E.g. Euclidean distance

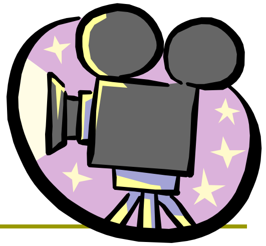
$$d(o_1, o_2) = \sqrt{(10.23 - 12.65)^2 + (5.27 - 6.25)^2 + (15.22 - 16.22)^2 + (2.7 - 2.2)^2 + (1.2 - 1.1)^2}$$

$$= 2.84$$

Similarity search

- Search is applied on a collection of objects O
- Given a query object q, find
  - objects o∈O for which dist(q,o) ≤ ε (range similarity query)
  - k nearest objects o∈O to q (kNN similarity query)
- Main applications
  - multimedia search-by-example
  - data mining (clustering, NN-based classification)
- Issues in query evaluation
  - poor performance of spatial indexes in high-dimensional spaces
  - curse of dimensionality

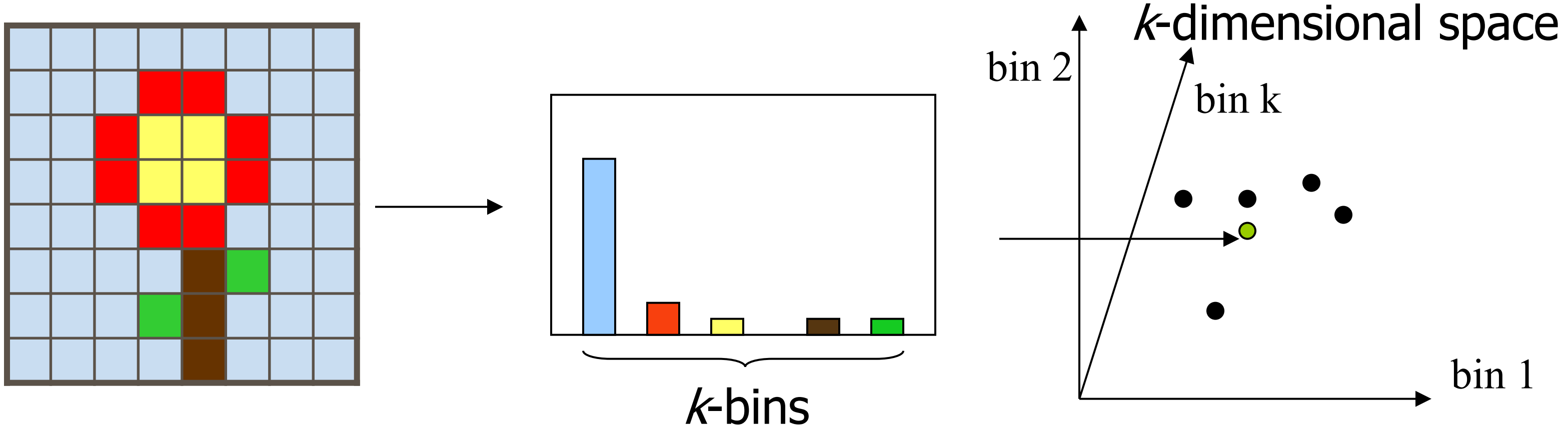
Application: Multimedia Data



- Objective: query and analyze vast amounts of multimedia data (e.g., images)
- *Content-based Image Retrieval*:
  - index and retrieve images based on their *visual contents* (e.g. color distribution)
- Methodology:
  - From each image extract a fixed set of features (e.g. color features)
  - Represent images as feature vectors
  - Index and query feature vectors instead of images
    - feature vectors are multidimensional objects

Example: Image Color Features

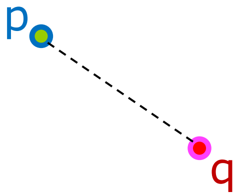
- To represent the color of an image compactly, a *color histogram* is used. Colors are partitioned to *k* groups according to their similarity and the *percentage* of each group in the image is measured.
- Images are transformed to *k*-dimensional points and a distance metric (e.g., Euclidean distance) is used to measure the similarity between them.



Distance Metrics in a Multidimensional Space (Minkowski distance)

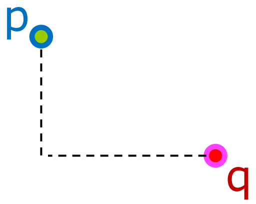
- Given two n-dimensional points
  - $p = p_1 \ldots p_n$
  - $q = q_1 \ldots q_n$
- their Euclidean distance is defined as:

$$L_2(p,q) \equiv \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$



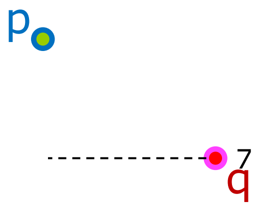
- also
  - Manhattan (city block) distance

$$L_1(p,q) \equiv \sum_{i=1}^{n} |p_i - q_i|$$

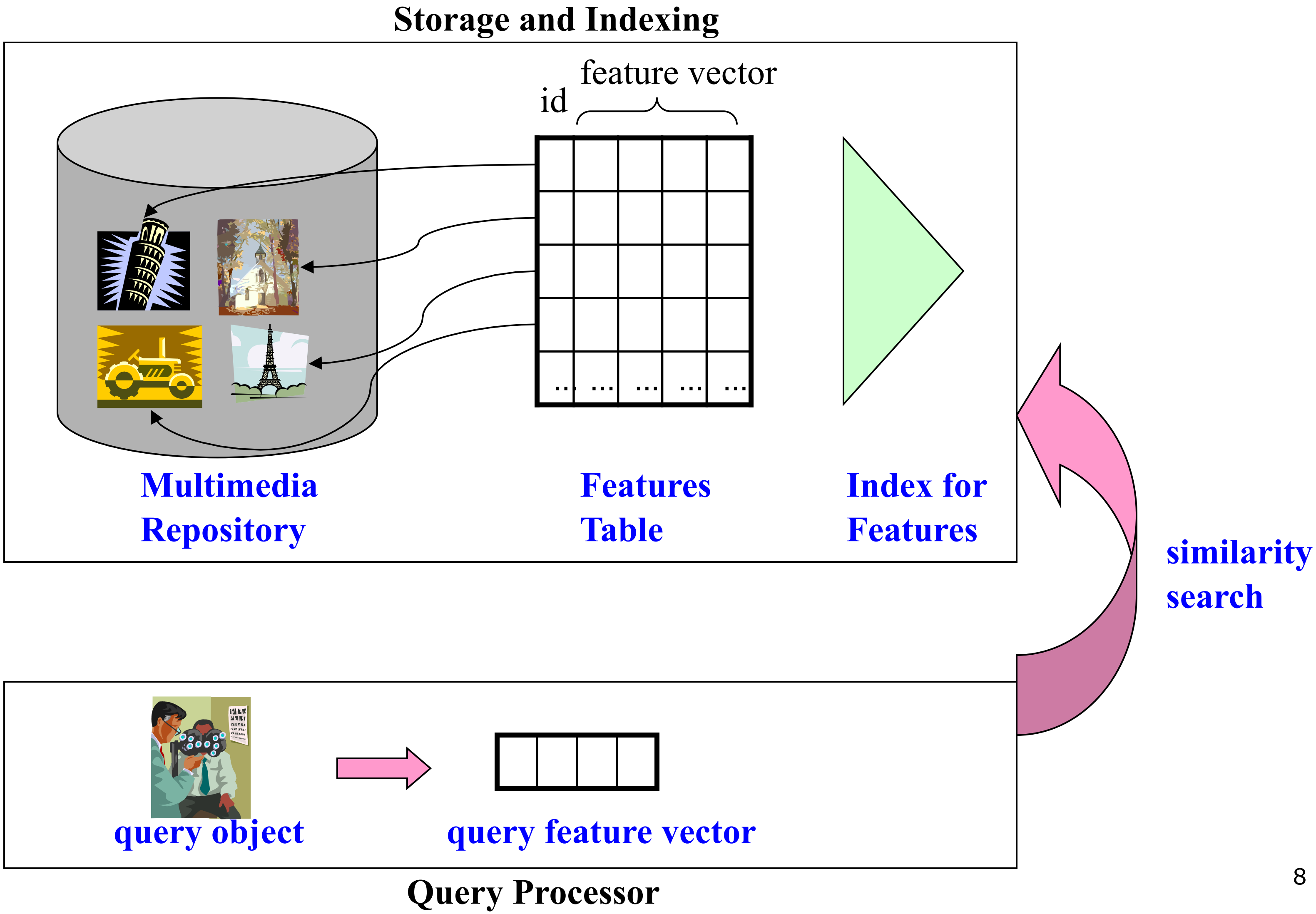


  - Max (supremum) distance

$$L_\infty(p,q) \equiv \max_{i=1}^{n} |p_i - q_i|$$



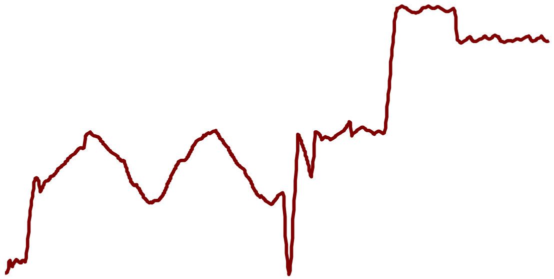
Architecture of a Multimedia Database



Application: Time-series Data

- A time-series is a sequential collection of values or events over time.
- Time series data are found in everywhere, e.g., stock market values, sensor indications, cardiograms.

real-valued time sequence
(e.g. stock prices)



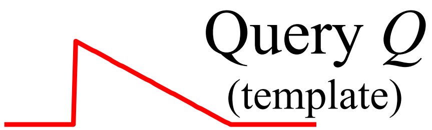
event sequence
(e.g., human activities during a day)

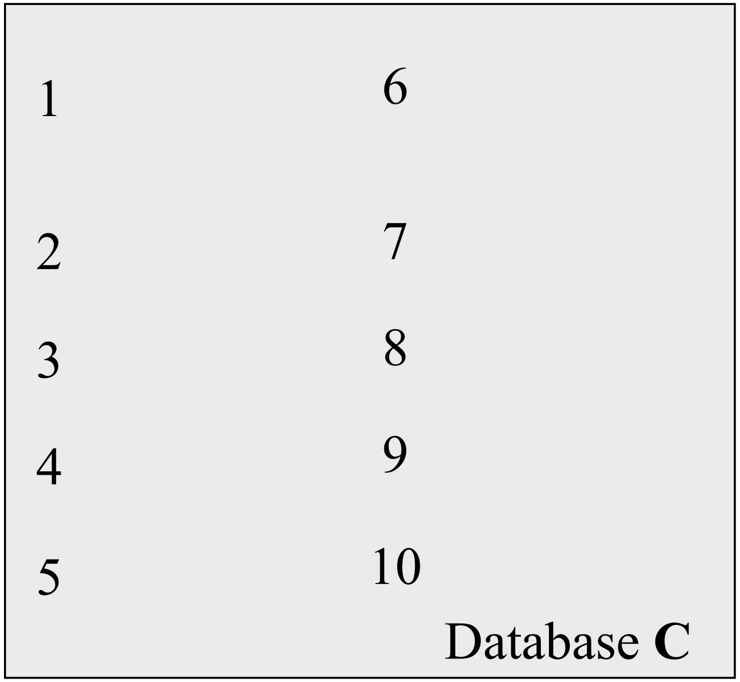
AABBBSSBABAGGTSBAK

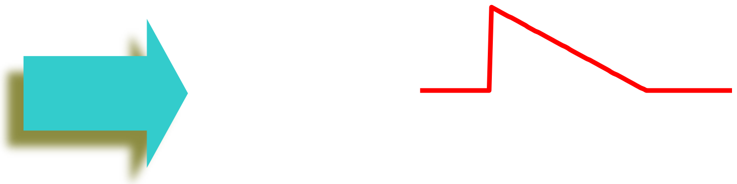
Queries and Analysis Tasks on Time-series Data

- find the most similar sequence to a query sequence q

1: Whole Matching



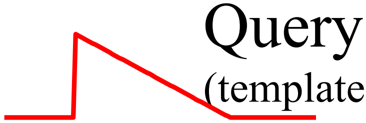




C_6 is the best match.

Queries and Analysis Tasks on Time-series Data

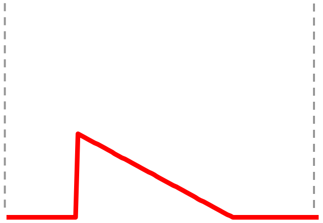
- find (approximate/exact) occurrences of a query subsequence q in a long sequence T.



Query *Q*
(template)

2: Subsequence Matching

long sequence **T**

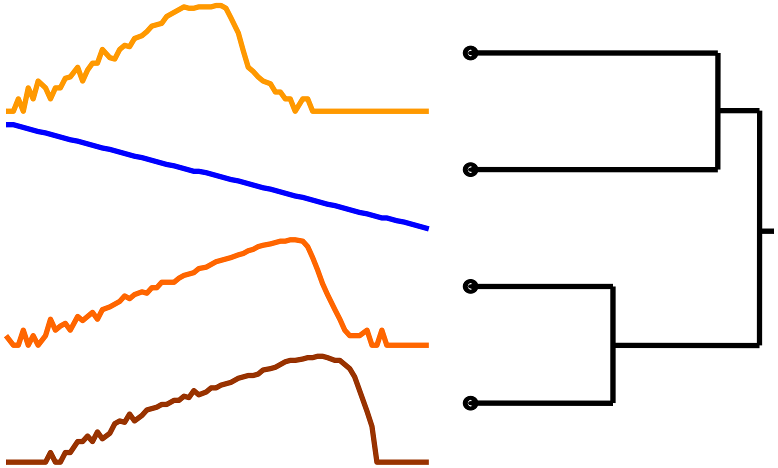


The best matching subsection.

Note that we can always convert subsequence matching to whole matching by sliding a window across the long sequence, and copying the window contents.

Queries and Analysis Tasks on Time-series Data

- Classification and Clustering (same are found in multimedia databases)

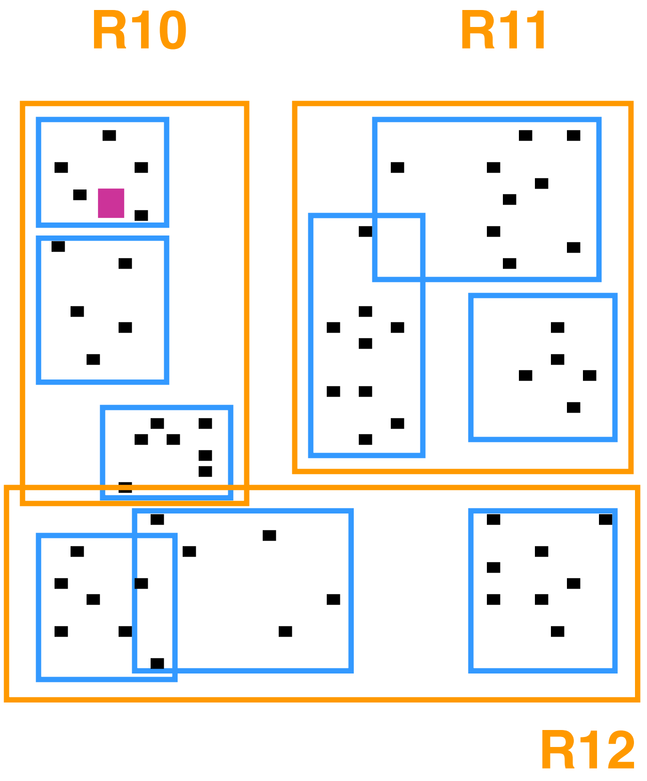


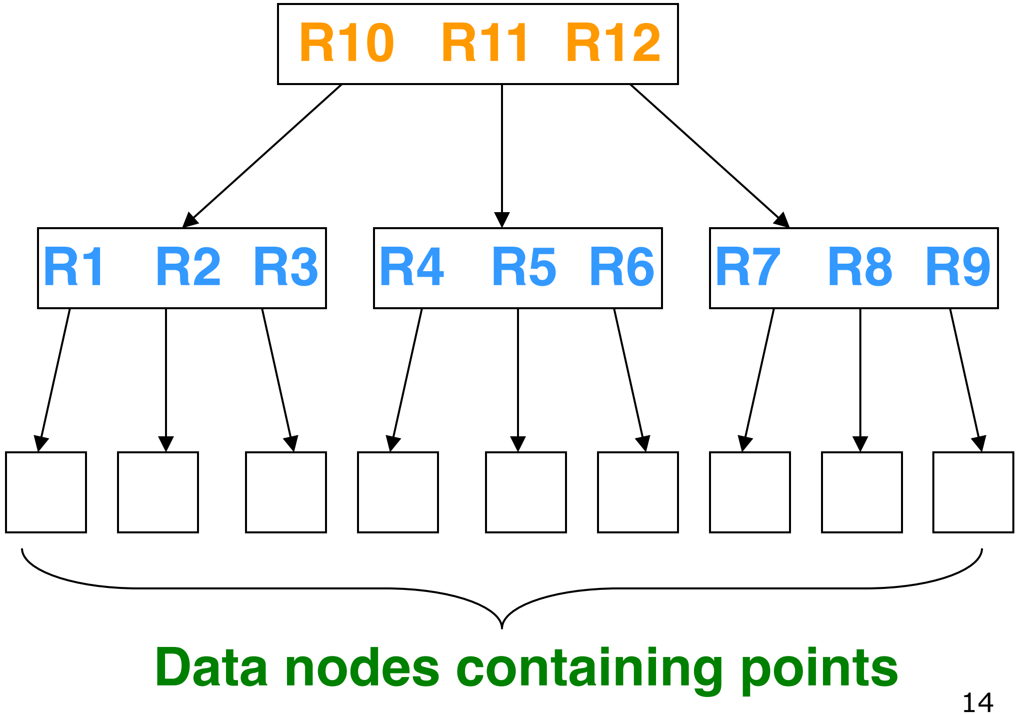
- Discovery of rules and trends
  - If stock A moves up for 10 days in a row then it will move down the next 5 days with high probability

Indexing in High-dimensional Spaces

Problem: Indexing feature vectors or time sequences for fast similarity search

- Possible solution: represent each vector as a point in the multi-dimensional feature space, index them by an R-tree and use spatial query methods (e.g., BF kNN search)





Problem!

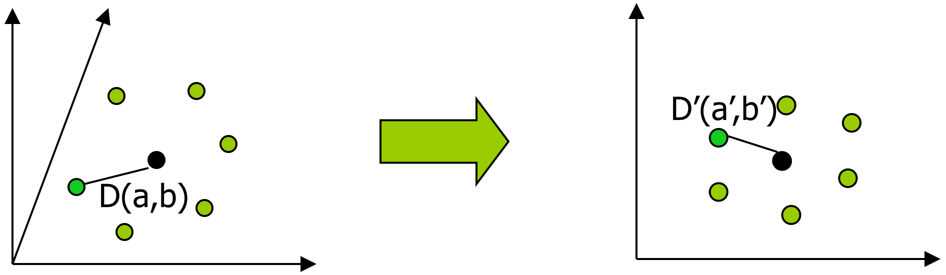
- The R-tree does not scale well for many dimensions. Somewhere above 6-20 dimensions search using the R-tree is slower compared to linear scan
  - Non-leaf entries have large MBRs, a lot of empty space
  - not all dimensions are used for splitting
    - a query point is inside ALL MBRs in most dimensions!
- Feature vectors and time-series are long (hundreds of dimensions)
- Distances between objects become meaningless even with a few noise dimensions (dimensionality curse)

Solutions

- The problem can be alleviated by:
  - dimensionality reduction and application of multi-step search algorithms
  - data compression and linear scan
  - indexing the metric distance space

Dimensionality Reduction

- In many cases the embedded dimensionality of a search problem is much lower than the actual dimensionality
- Some methods apply transformations on the data and approximate them with low-dimensional vectors
- The aim is to reduce dimensionality and at the same time maintain the data characteristics

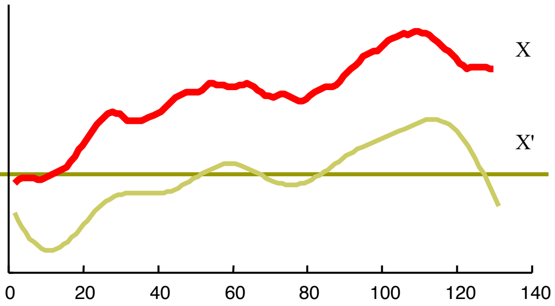


A simple dimensionality reduction approach: project data to a small set of dominant dimensions

GEMINI GEneric Multimedia INdexIng

- Establish a distance metric D from a domain expert
  - e.g., Euclidean distance
- Produce a dimensionality reduction technique that reduces the dimensionality of the data from n to N, where N can be efficiently handled by a spatial access method (e.g., R-tree)
  - f(o) = o’
- Produce a distance measure D’ defined on the N dimensional representation of the data, and prove that for any pair of points (a,b) it obeys
  - $D'(a',b') \leq D(a,b)$
  - the above is called the lower bounding property
- Plug into an off-the-shelf spatial-access-method (e.g., R-tree).

Dimensionality Reduction Example: Discrete Fourier Transform (DFT)

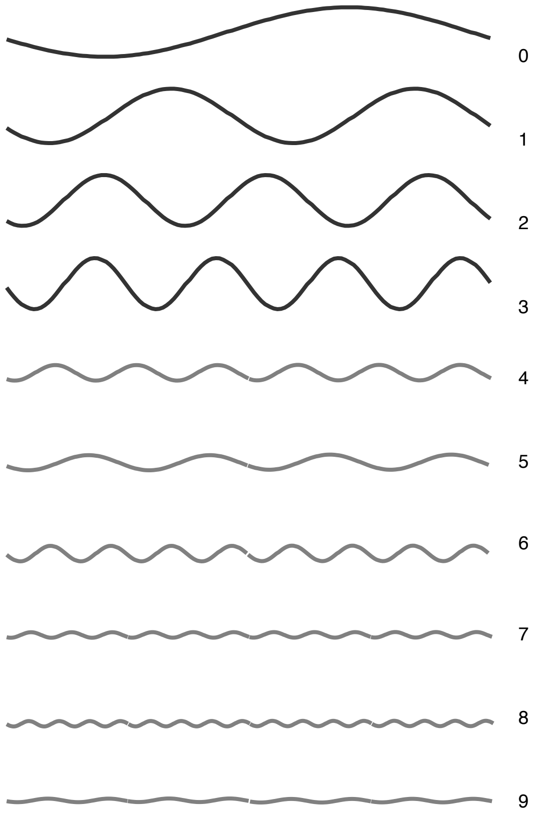


- How to represent a time-series (or a color histogram) using only *n* numbers (in a *n*-dimensional space)?

- Basic Idea: Represent the time series as a linear combination of sines and cosines, but keep only the first *n/2* coefficients.

Why *n*/2 coefficients? Because each wave requires 2 numbers, for the phase (*w*) and amplitude (*A*,*B*).

$$C(t) = \sum_{k=1}^{n} \left(A_k \cos(2\pi w_k t) + B_k \sin(2\pi w_k t) \right)$$

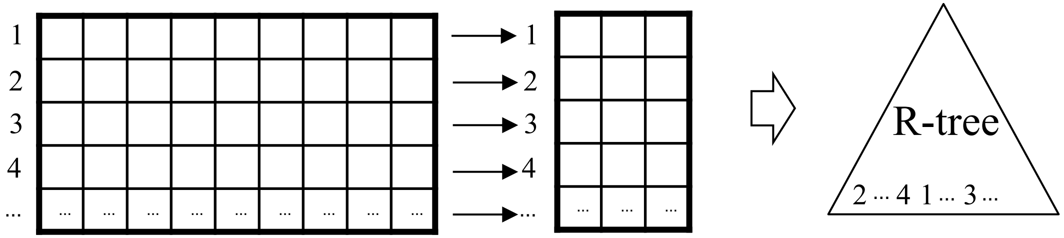


Other Dimensionality Reduction Techniques

- Other popular transformations include:
  - Discrete Wavelet Transform. The sequence is transformed to a linear combination of Wavelet basis functions, but only the largest n coefficients are used.
  - Singular Value Decomposition. Similarly, the sequence is transformed to a linear combination of eigenwaves, and only the first n coefficients are used.
    - See also Principal Component Analysis (PCA)
  - Piecewise approximations are also used for time-series data.

Data representation and indexing

- Methodology:
  - Each vector p in S is transformed to a low-dimensional vector p’
  - An index R for low-dimensional transformed vectors (e.g., R-tree) is used to for all transformed vectors p’.
  - We define a distance function (e.g., Euclidean distance) D’(p’,q’) for the transformed vectors, such that
    D’(p’,q’) ≤ D(p,q) (lower bounding property)



Two-step processing of range similarity queries

Range Similarity Queries

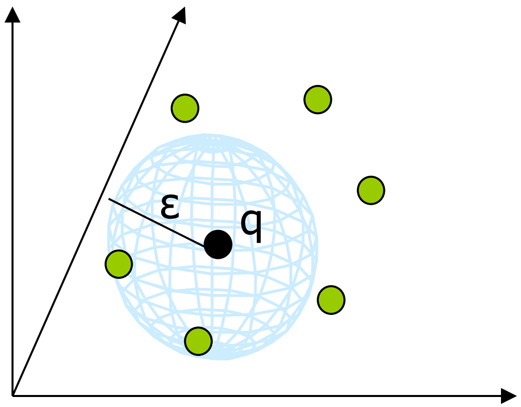
- Given:
  - A database S of feature vectors (or time sequences)
  - A distance function $D(p_1,p_2)$ that computes the dissimilarity between vectors p_1 and p_2
  - A query vector q, a distance threshold ε
- Find:
  - All vectors p in S, such that $D(p,q) \leq \varepsilon$

Two-step processing of range similarity queries

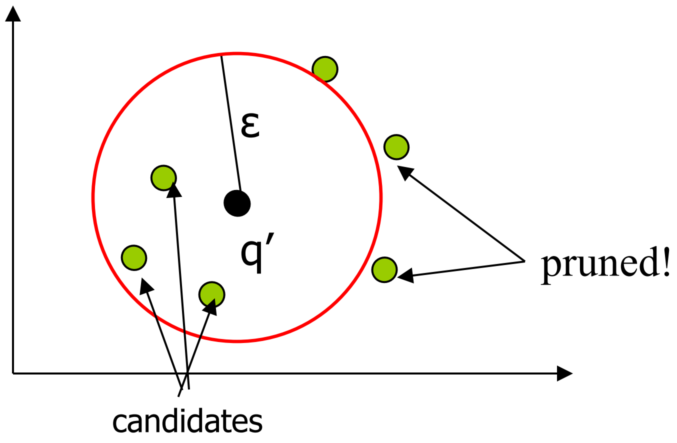
- Step 1:
  - convert q to q′ using the same dimensionality reduction technique used for the data objects
  - apply an R-tree range search to find fast a $S' \subseteq S$, such that for all p′ in S′, $D'(p',q') \leq \varepsilon$
- Step 2:
  - for each p′ in S′, use identifier of p′ to get high-dimensional vector p, compute D(p,q) and if $D(p,q) \leq \varepsilon$ add it to the response set

Two-step processing of range similarity queries

high-dimensional space



low-dimensional space



Q1: Will this method miss any results?

Q2: Will this method compute incorrect results?

Two-step processing of nearest neighbor similarity queries

Nearest Neighbor Similarity Queries

- Given:
  - A database S of multimedia feature vectors (or time sequences).
  - A distance function $D(p_1,p_2)$ that computes the dissimilarity between vectors p_1 and p_2.
  - A query vector q
- Find:
  - The most similar vector to q in S.
  - Or else, $p \in S$ such that $\forall s \in S$, $D(p,q) \leq D(s,q)$

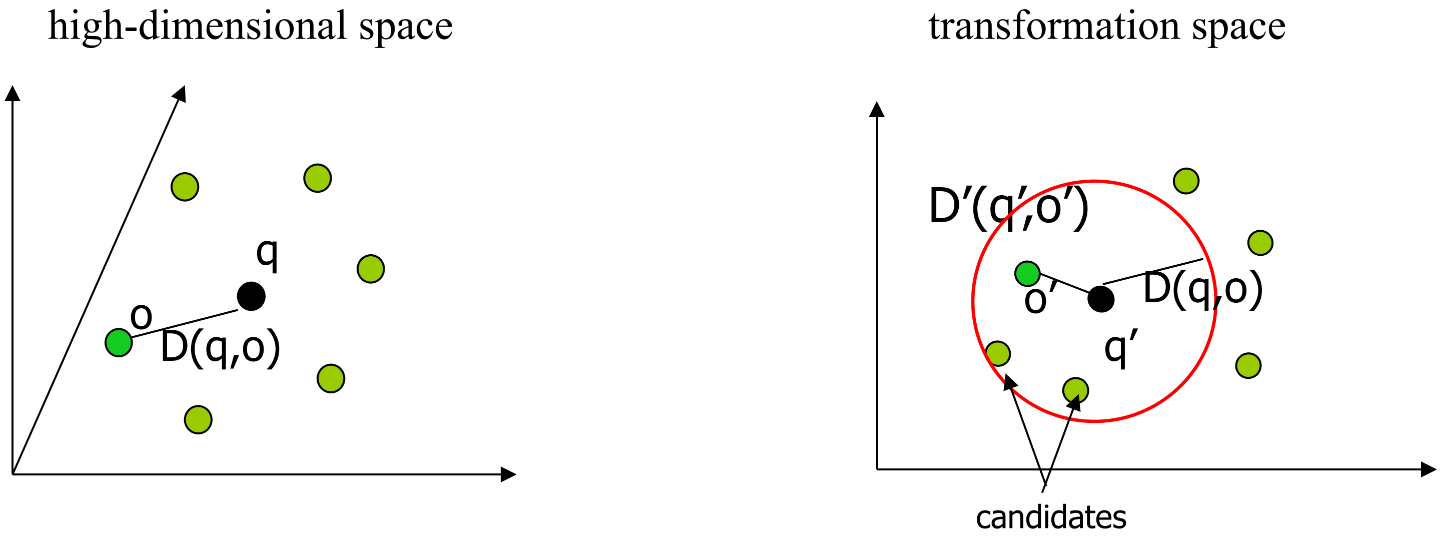
Two-step processing of nearest neighbor similarity queries

- Methodology:
  - Each vector p in S is transformed to a low-dimensional vector p’
  - An index R for low-dimensional transformed vectors (e.g., R-tree) is used to for all transformed vectors p’.
  - We define a distance function (e.g., Euclidean distance) D’(p’,q’) for the transformed vectors, such that
    D’(p’,q’) ≤ D(p,q) (lower bounding property)

Two-step processing of nearest neighbor similarity queries

- Step 1:
  - convert q to q′ using the same dimensionality reduction technique
  - apply an R-tree nearest neighbor search to find fast the nearest p′ to q′.
  - Let p∈S be the corresponding high-dim vector to p′. Compute D(q,p). Apply an R-tree range search to find fast a S′⊆S, such that for all points s′ in S′, D′(s′,q′) ≤ D(q,p)
- Step 2:
  - for each s′ in S′ compute D(s,q) and return the one with the smallest D(s,q)

Two-step processing of nearest neighbor similarity queries


high-dimensional space

transformation space

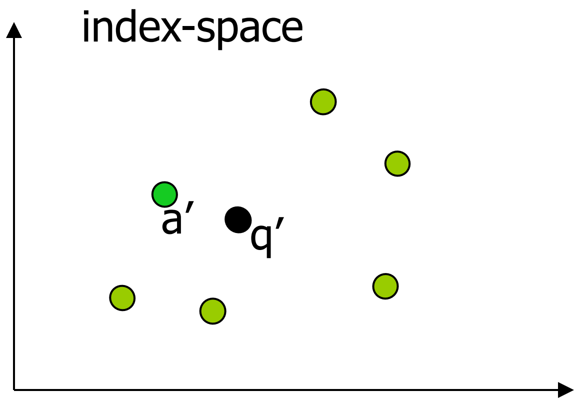
- Q: Will this method compute the correct result? why?

Multi-step processing of nearest neighbor similarity queries

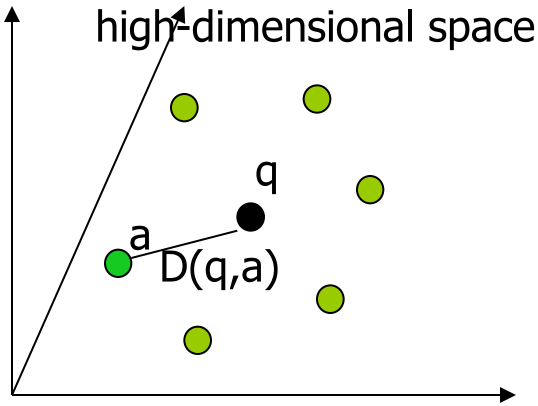
- Convert q to q′ using the same dimensionality reduction technique.
- NN = NULL; D(q,NN) = ∞;
- Repeat:
  - apply an incremental R-tree nearest neighbor search to find fast the next nearest p′ to q′.
  - If D′(q′,p′) < D(q,NN) compute actual D(q,p).
    - If D(q,p)<D(q,NN) then NN = p
- Until D′(q′,p′) ≥ D(q,NN)

- Q: Will this method better than two-step processing?

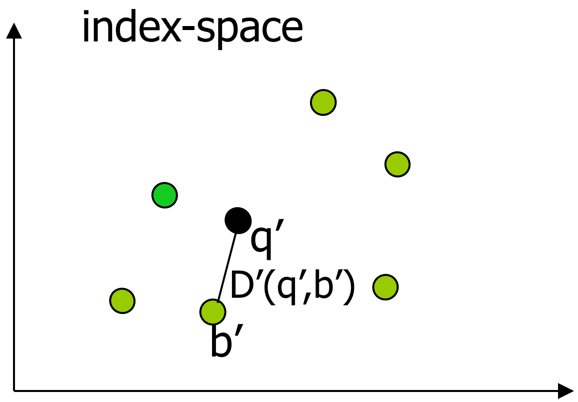
Example of multi-step processing



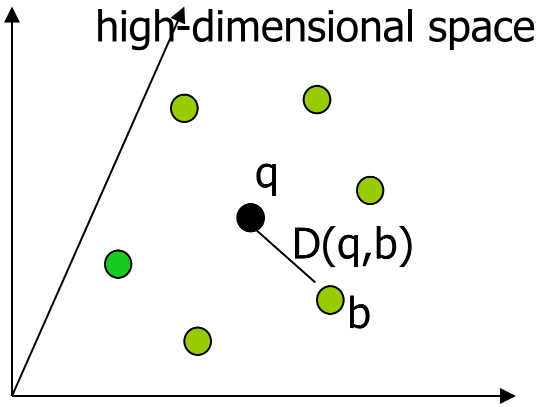
1. Get 1st NN (a′)



2. Compute a′s distance from q. Set cur_{NN}=a.

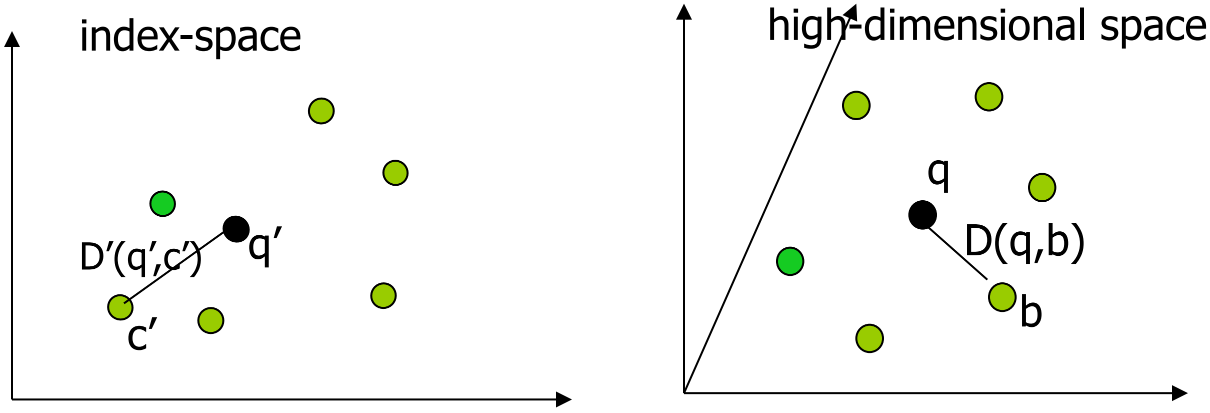


3. Get 2nd NN (a′). D′(q′,b′)< D(q,a), so goto step 4



4. Compute b′s distance from q. Since D(q,b)<D(q,a), Set cur_{NN}=b.

Example of multi-step processing



5. Get 3rd NN (c′). D′(q′,c′)> D(q,b), so terminate and report b as NN.

Compression-based indexing

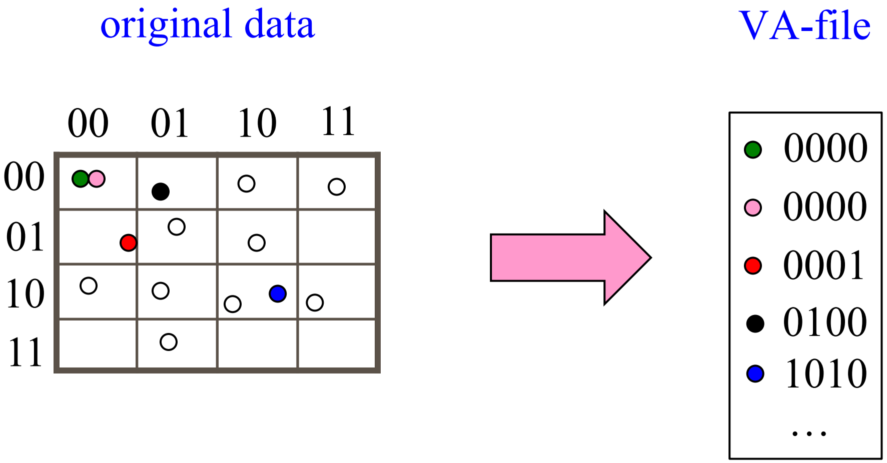
- Reading: VAfile paper
- Motivation:
    - In very high dimensional spaces dimensionality reduction can be expensive and ineffective
        - Should examine multiple possible dimension-sets to potentially reduce
    - Sometimes we have to resort to linear scan
        - Expensive because the entire (large) set of feature vectors have to be scanned and for each of them we need an (expensive) distance computation

Compression-based indexing

- Idea:
  - Partition the space by a grid
  - Approximate each vector by a bitstring that designates the partition where it belongs
  - Linearly scan all bitstrings and use bounds to eliminate most of the objects
  - Perform exact distance computations for the objects that survive the scan

The Vector-Approximation File

- data preprocessing phase



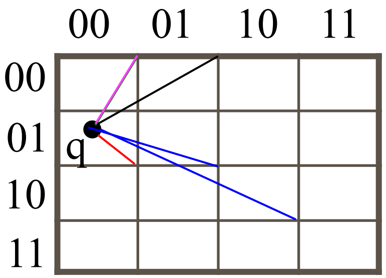
original data

VA-file

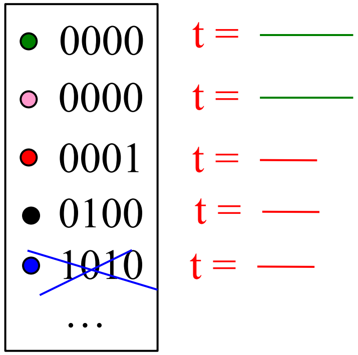
● 0000
● 0000
● 0001
● 0100
● 1010
…

The Vector-Approximation File

- similarity search (step 1)
  - scan VA-file and for each bitstring b
    - compute upper bound $dist_u(q,b)$ to q
      - keep track of smallest upper bound t
    - compute lower bound $dist_l(q,b)$ to q
    - if $dist_l(q,b) \geq t$ then filter out b
    - else put b.obj to candidates set



VA-file


● 0000 t =

○ 0000 t =

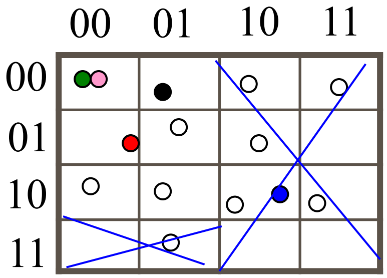
● 0001 t =

● 0100 t =

● ~~1010~~ t =

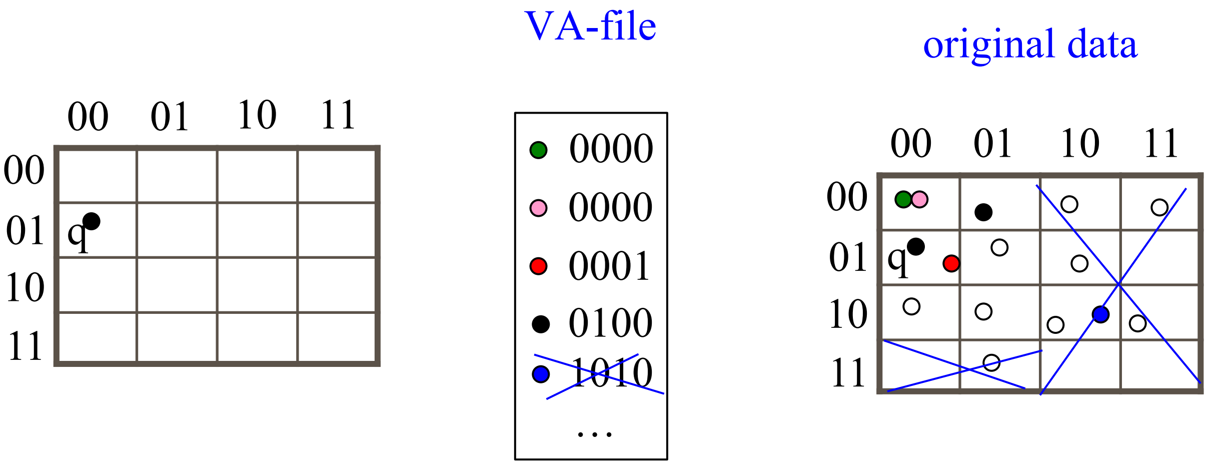
…

original data



The Vector-Approximation File

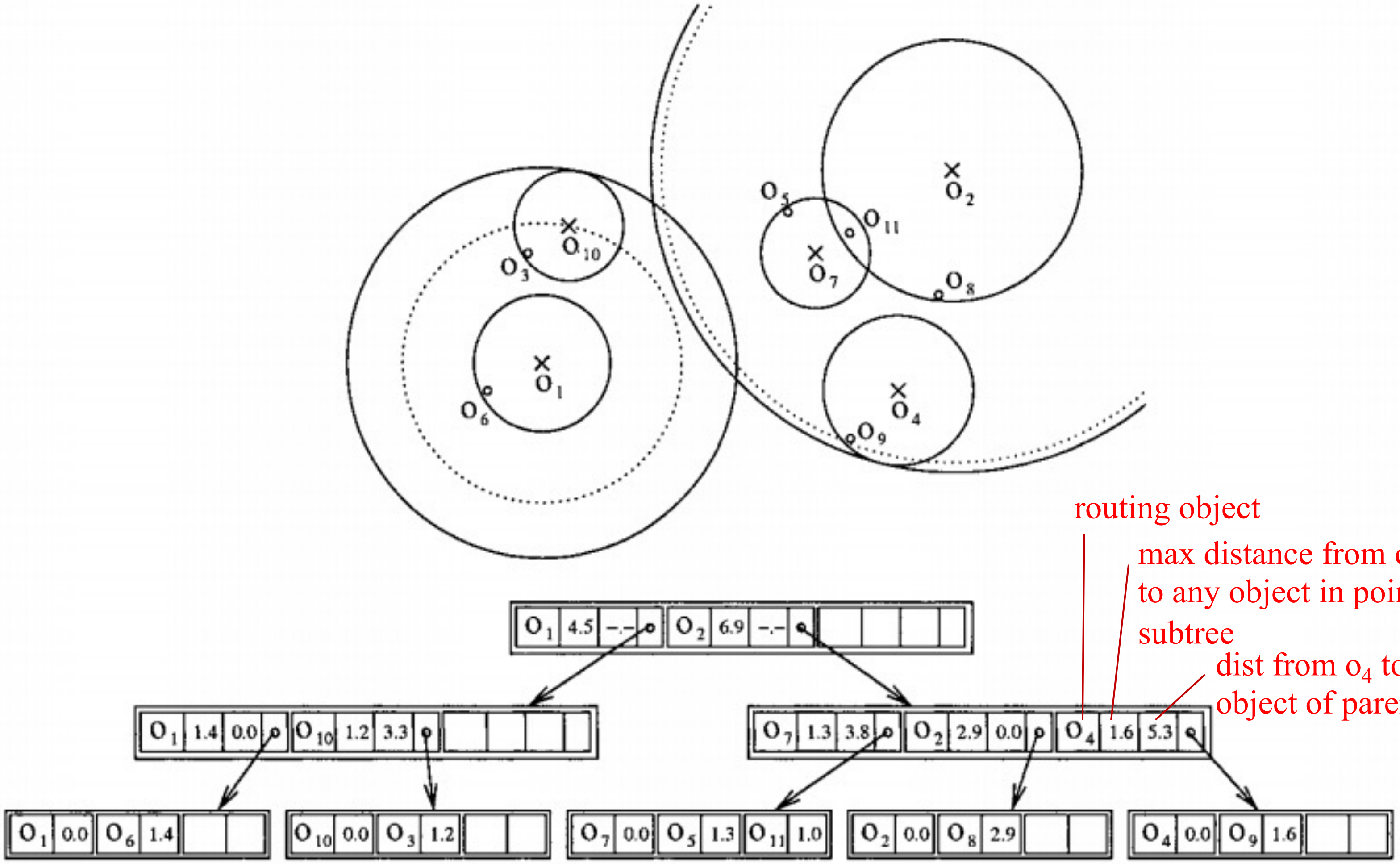
- similarity search (step 2)
  - Sort non-filtered candidates by $dist_l(q,b)$, scan them and for each bitstring b
    - compute lower bound $dist_l(q,b)$ to q
    - if $dist_l(q,b) \geq t$ then filter out b
    - else compute actual dist(q,b.obj) and update current actual NN and t



Indexing metric spaces: the M-tree

- Reading: M-tree paper
- Idea: Instead of indexing the feature space of the objects, index them by their distances
- Group objects into index nodes hierarchically by putting objects near each other to the same group
  - Applicable even if the objects have unknown attributes and only the distances between them are known
- Each node n has a routing object o and a radius r (stored in the entry pointing to n)
  - For every object o' in the sub-tree of n it should be dist(o,o') $\leq$ r
  - All data objects appear in leaf nodes

The M-tree: example



routing object

max distance from o_4 to any object in pointed subtree

dist from o_4 to routing object of parent node (o_2)

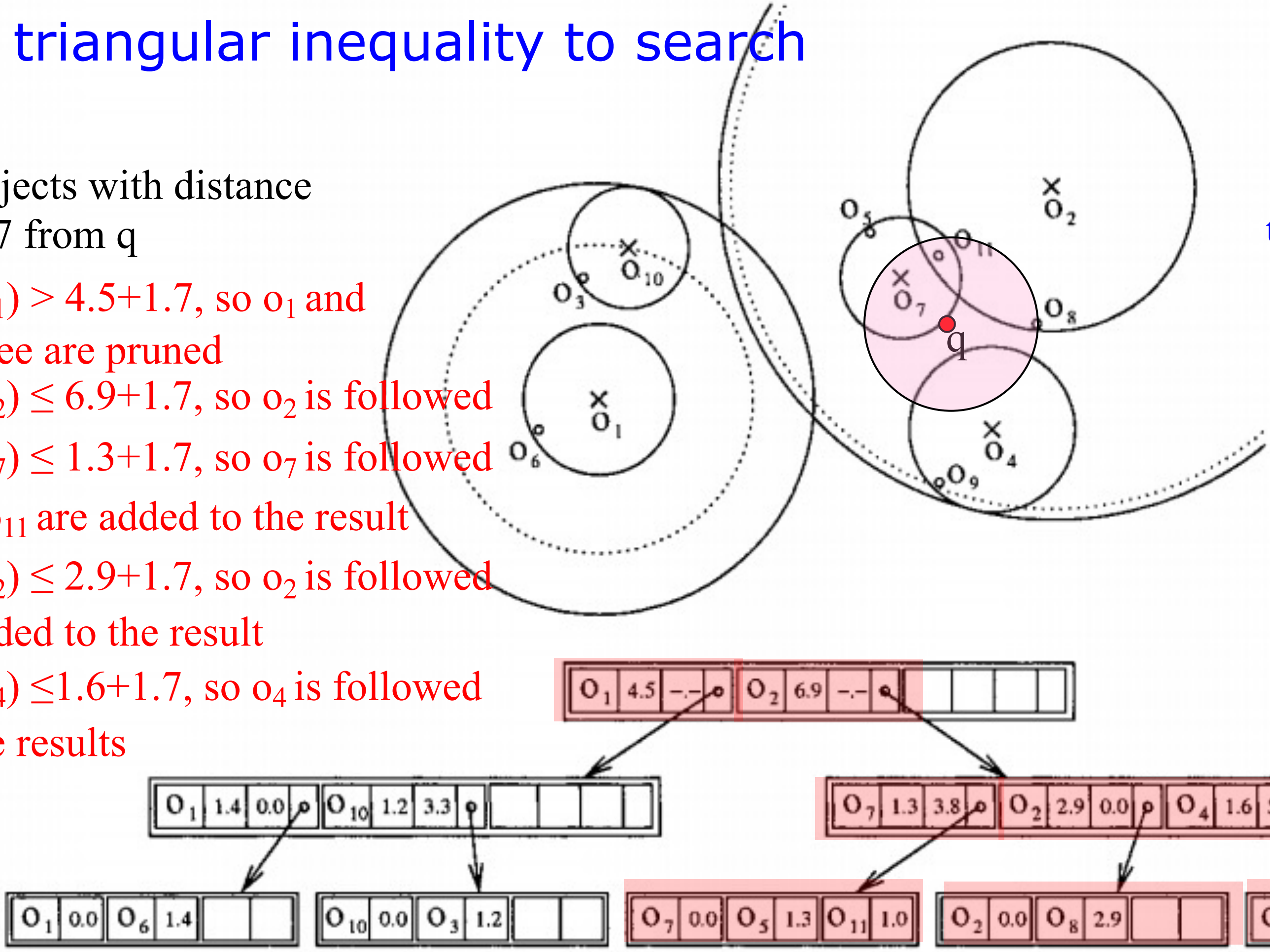
[Figure taken from http://itu-algorithms.github.io/events/2015/05/20/Zezulacourse]

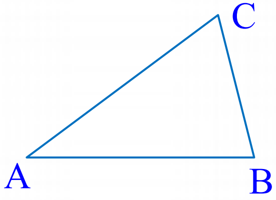
The M-tree: queries

- Use triangular inequality to search

example:
find all objects with distance
at most 1.7 from q

1) $dist(q,o_1) > 4.5+1.7$, so o_1 and its subtree are pruned
2) $dist(q,o_2) \leq 6.9+1.7$, so o_2 is followed
3) $dist(q,o_7) \leq 1.3+1.7$, so o_7 is followed
4) o_7 and o_{11} are added to the result
5) $dist(q,o_2) \leq 2.9+1.7$, so o_2 is followed
6) o_8 is added to the result
7) $dist(q,o_4) \leq 1.6+1.7$, so o_4 is followed
8) no more results





triangular inequality:
$AB \leq AC+CB$
$AB \geq |AC-CB|$

[Figure taken from http://itu-algorithms.github.io/events/2015/05/20/Zezulacourse]

The M-tree: updates

- Insertions:
  - “Search” the tree, by recursively following the routing object which is the closest to the new object o
  - If the leaf node fits o, insert it there, otherwise split the leaf node
    - Partition the objects of the leaf + o to two new leaf nodes with two new routing objects for them and replace the old routing object in the parent by the two new routing objects
      - New routing objects should minimize volume and overlap of the new leaf nodes they define
- Deletions: as in R-tree

Indexing metric spaces: use pivots

- Data preparation:
  - Select a small number of points, called pivots
  - For each data point o, compute distances to all pivots and store them in a matrix
- Observation:
  - Given a query point q and a data point o, we have (triangle inequality)
    - dist(q,o) ≥ |dist(p,o) – dist(p,q)|, for each pivot p
- Methodology:
  - For each pivot, compute dist(p,q)
  - For each data point o
    - if |dist(p,o) – dist(p,q)| > ε, for some p
      - mark o as non-result; break
    - if o is not marked, compute dist(q,o) and verify o

Pivot-based distance bound

- O is the set of data points
- P is the set of pivots (P⊂O)
- q is a query point

$$dist(o,q) \geq \max_{p \in P} \left| \overset{\text{precomputed distance}}{dist(p,o)} - \overset{\text{computed once for each query}}{dist(p,q)} \right|$$

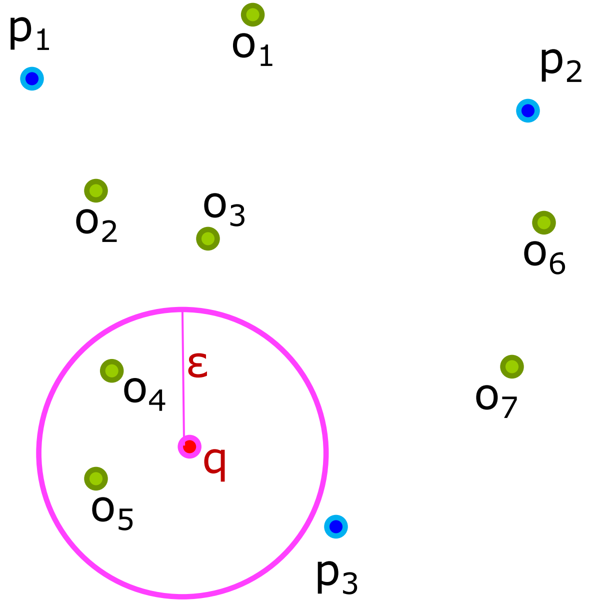
- |P| distance computations are required per query to derive lower distance bounds for all objects
  - This is important because distance computations in high dimensional spaces are expensive
- Pivots are effective if they are far from each other

Indexing metric spaces: use pivots

- Example

	o_1	o_2	o_3	o_4	o_5	o_6	o_7
p_1	3	2	4	6	7	8	9
p_2	5	7	6	8	9	2	4
p_3	9	7	6	5	4	6	4

precomputed distance matrix



- query q, ε=3

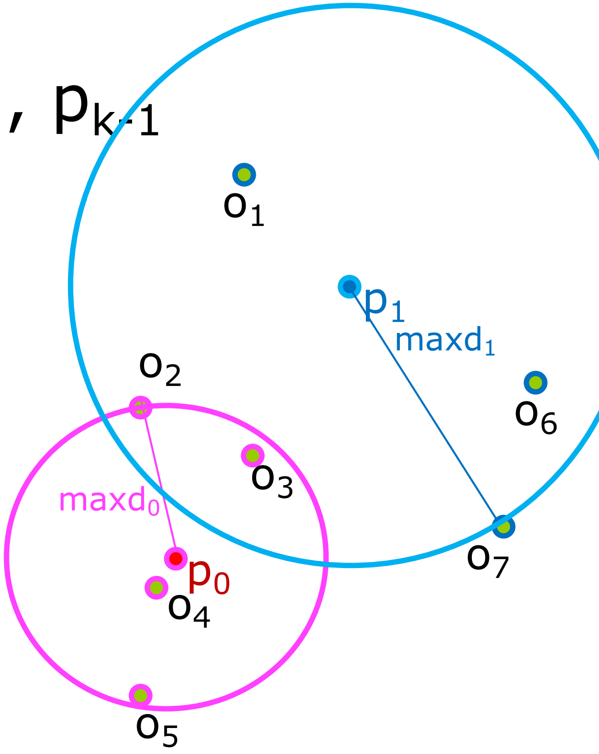
	q
p_1	7
p_2	8
p_3	4

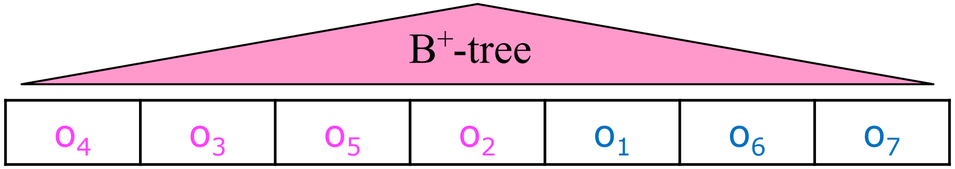
computed once for each query

- $|dist(p_1,o_1)-dist(p_1,q)|=|3-7|=4>\varepsilon$ o_1 eliminated!
- $|dist(p_1,o_2)-dist(p_1,q)|=|2-7|=4>\varepsilon$ o_2 eliminated!
- $$\left.\begin{array}{l}|dist(p_1,o_3)-dist(p_1,q)|=|4-7|=3\\ |dist(p_2,o_3)-dist(p_2,q)|=|6-8|=2\\ |dist(p_3,o_3)-dist(p_3,q)|=|6-4|=2\end{array}\right\}$$ o_3 not eliminated, must compute dist(o_3,q)

iDistance index

- Reading: iDistance paper
- Define a set P of k pivots: $p_0, p_1, \ldots, p_{k-1}$
  - Cluster centers can be used as pivots
- Each object o is mapped to a 1D value
  - Let p_i be the nearest pivot to object o
  - Object o is mapped to value $v(o) = maxd*i + dist(o,p_i)$
    - for each pivot p_i, keep track of distances $mind_i$, $maxd_i$
    - $maxd = max(maxd_i)$
  - Use a B^+-tree to index <v(o), o.id> pairs

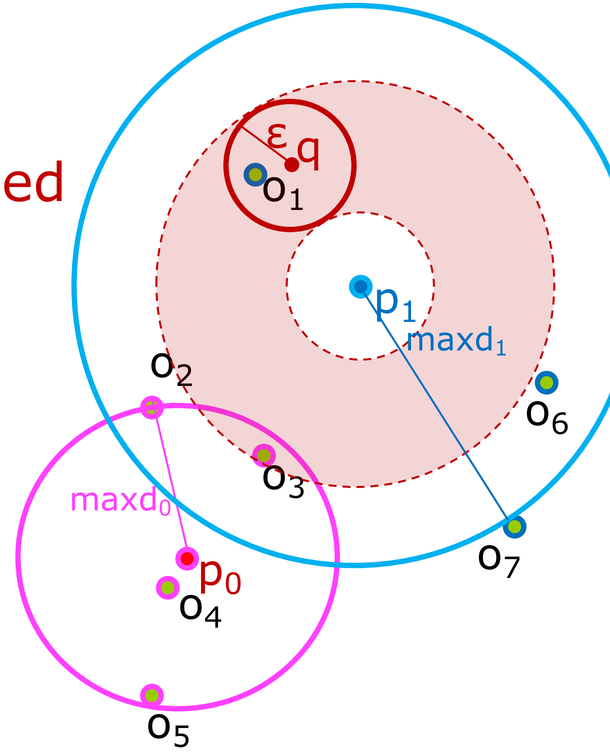




o_4	o_3	o_5	o_2	o_1	o_6	o_7

iDistance index: query evaluation

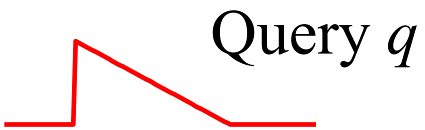
- Range query: (q,ε)
- For each pivot p_i:
  - If dist(q,p_i) - $maxd_i$> ε, pivot p_i is pruned
  - Else specify B^+-tree range to search for pi:
    - Lower bound: maxd*i + dist(q,p_i) – ε
    - Upper bound: maxd*i + dist(q,p_i) + ε
- k-NN query: (q,k)
  - Find kNN for nearest pivot to q
  - Use k-th distance as bound
    - Progressively shrink distance while searching other pivots

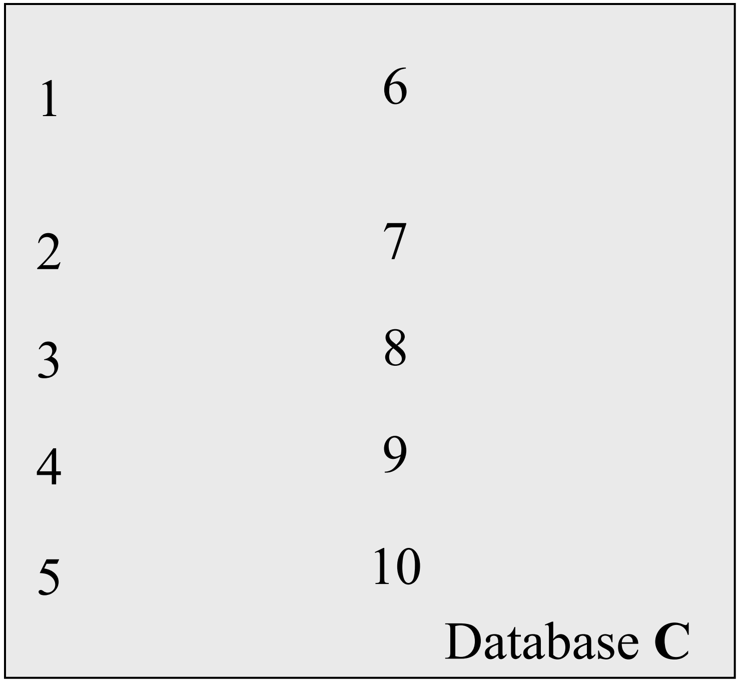


Time-series Search

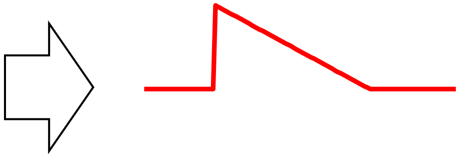
Searching for similar time-series

- find similar series to a query series q





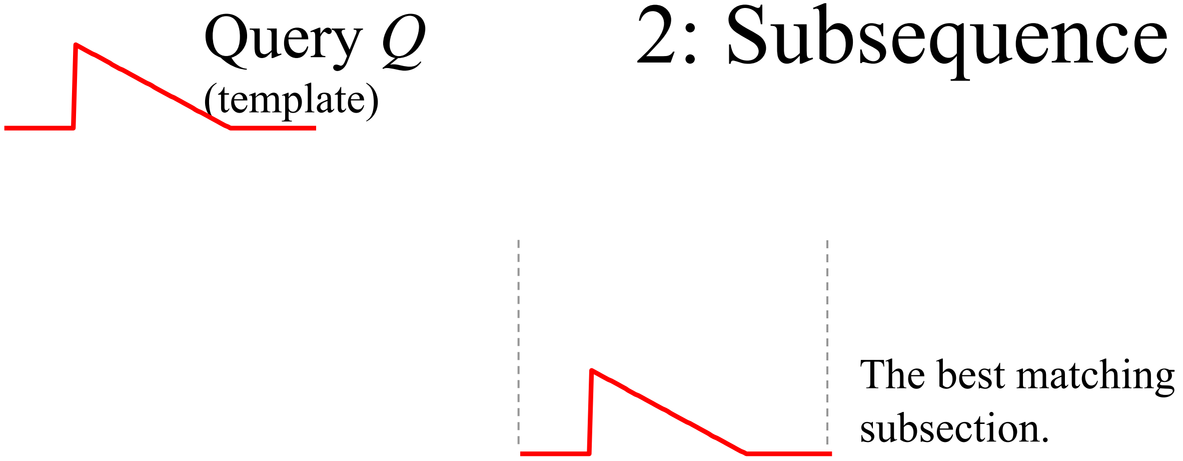
- Each position is a dimension
- All series have exactly the same list of posititions
- Simple approach: use Euclidean distance as for the case of feature vectors
- Drawbacks:
  - Ignores the fact that consecutive positions are semantically relevant
  - Cannot compute similarity between series of different lengths



C_6 is the best match.

Subsequence Matching

- find (approximate/exact) occurrences of a query subsequence q in a long sequence T.



Query *Q*
(template)

2: Subsequence Matching

long sequence **T**

The best matching
subsection.

Subsequence Matching

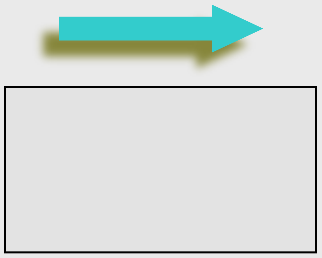
- Problem:
  - Given one or possibly more long real-valued sequences (i.e., time-series), develop an index for subsequence matching queries:
  - Subsequence Matching: given a database of long sequences S an a query sequence q, find all subsequences s in S, such that $D(s,q) \leq \varepsilon$, where ε is a distance threshold.

Subsequence Matching

- Determine a short sliding window w
- Assume for the moment that every query q is of length w
  - shorter queries than w are not interesting
  - we will discuss about longer ones soon
- Assume that D= Euclidean distance

Subsequence Matching

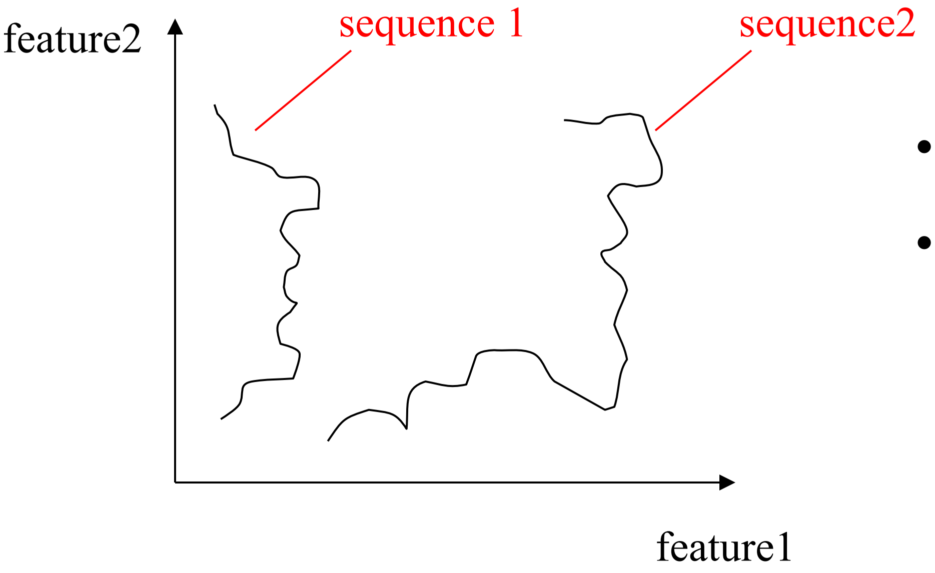
- Each position of the window defines a subsequence of length w
- Each subsequence can be transformed to a point in a low dimensional space
  - E.g., using DFT, WDT, PAA, etc.
- The Euclidean low-dimensional distance lower-bounds D.



Sliding window w

Subsequence Matching

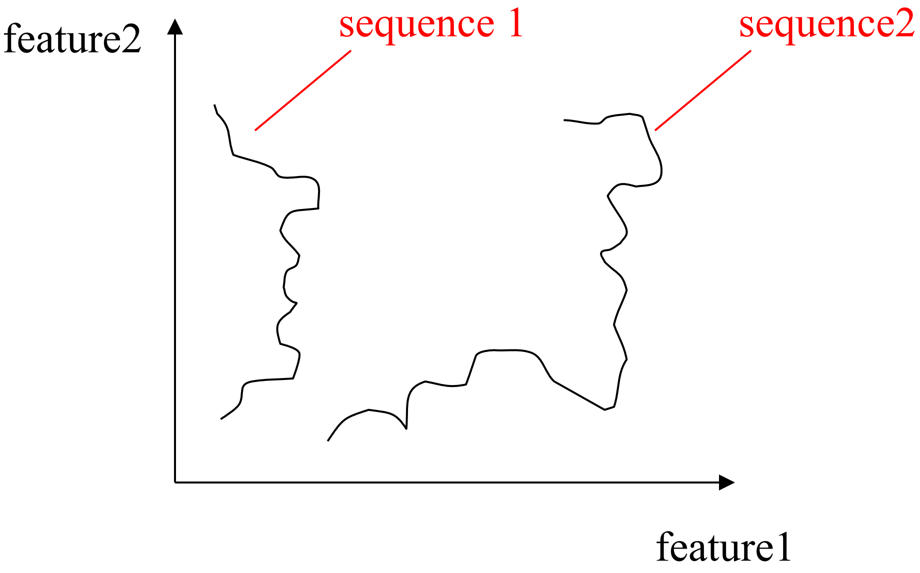
- The low-dimensional vectors of consecutive subsequences define a trail in the low-dimensional space



- Each point on a trail is a subsequence
- we can build an R-tree for these points

Subsequence Matching

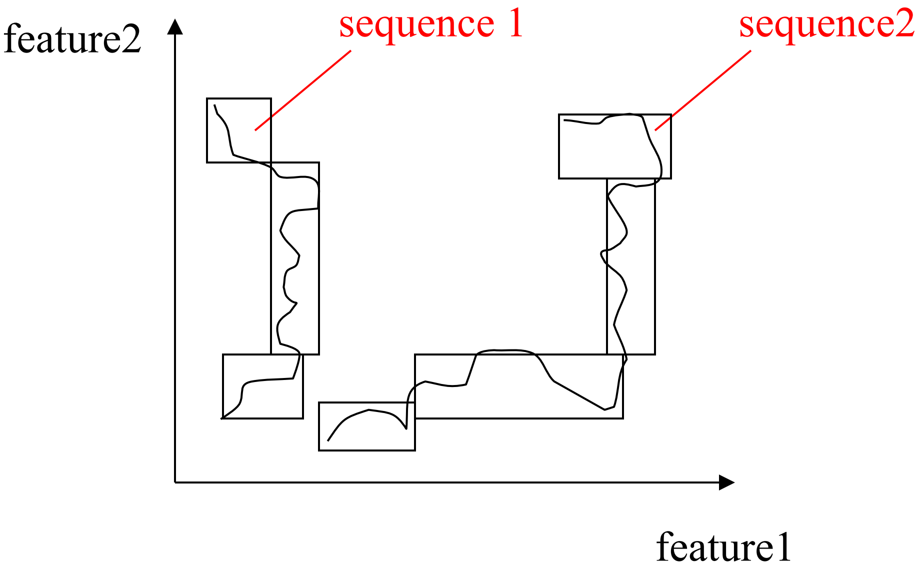
- However, the number of points can be too large, resulting in a large (and slow index)



- Example: 10M elements in a long vector, result in about 10M points

Subsequence Matching

- Idea: divide the trails and approximate them by MBRs (hyper-rectangles)

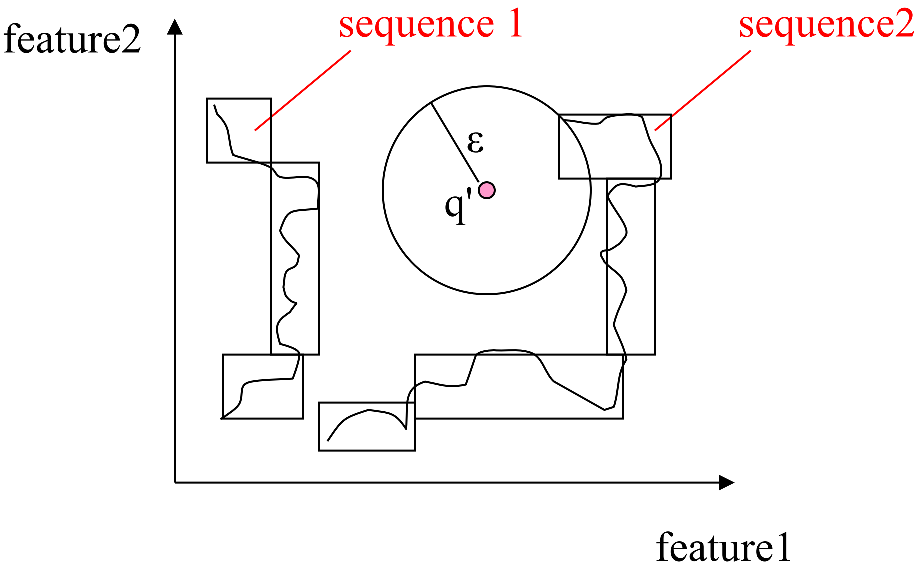


- Each indexed MBR unit stores
  - a) the id of the sequence
  - b) t_{start},t_{end} of the sub-trail

- Example: the 10M sequence can be divided into 10000 trails (about 1000 points per trail), thus the index will now have 10000 entries (instead of 10M)

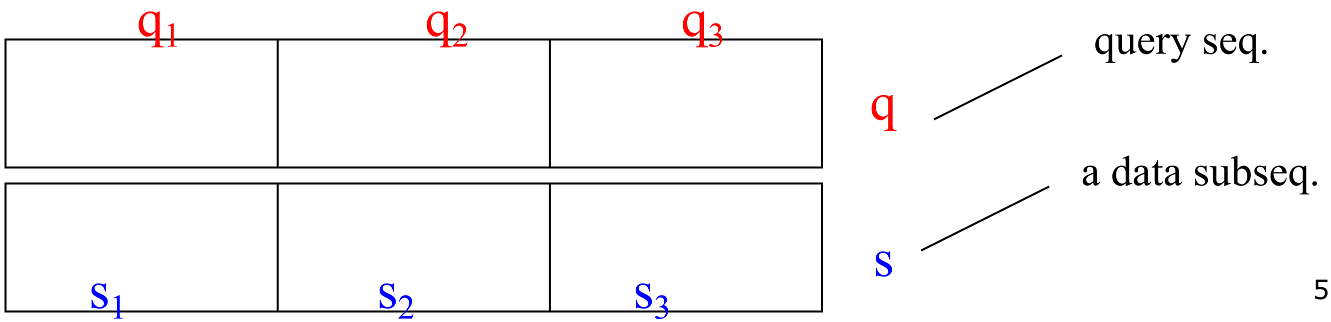
Subsequence Matching

- Searching:
  - Use the R-tree to find fast the sub-trails close to the query in the low-dimensional space
  - Linear-scan these trails to discard false-alarms



Subsequence Matching

- Searching for subsequences q longer than w:
  - divide q into p segments, each of length w
  - for each segment q_{seg}, apply an $\varepsilon/\sqrt{p}$ range query to find candidate subsequences that are close to the segment.
  - Unify the results for all segmental queries and examine the corresponding subsequences to discard false alarms
- Lemma:
  - If two sequences s and q are within distance ε from each other then at least one pair of segments s_i and q_i should be within distance $\varepsilon/\sqrt{p}$ from each other



Subsequence Matching: Example

- T={3,3,4,3,5,6,7,7,8,9,10,11,9,8,9,10}
- w = 3
- Subsequences: (3,3,4), (3,4,3),(4,3,5),…
- Assume no dim. reduction (already 3 dimensions only!)
- Consider query q = {3,4,3,5,6,5} and ε=4 (ε^2=16)
- Step 1: break q into q1={3,4,3} and q2={5,6,5}
- Step 2:
  - use the index to search for subsequences with sq. distance at most $\varepsilon^2/2$ from q1 and put their position in S1
  - use the index to search for subsequences with sq. distance at most $\varepsilon^2/2$ from q2 and put their position in S2
  - Merge S1 and S2 to a set of candidate positions P to examine
- Step 3:
  - Perform random accesses to positions in P to examine the candidate q-length subsequences.

Subsequence Matching: Example

$$\begin{array}{c|cccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
T= & \{3, & 3, & 4, & 3, & 5, & 6, & 7, & 7, & 8, & 9, & 10, & 11, & 9, & 8, & 9, & 10\}
\end{array}$$

$q1=\{3,4,3\}$

$\varepsilon^2/2=8$

$d^2(q1,1)=0+1^2+1^2=2 \leq \varepsilon^2/2$ √

$d^2(q1,2)=0+0+0=0 \leq \varepsilon^2/2$ √

$d^2(q1,3)=1^2+1^2+2^2=6 \leq \varepsilon^2/2$ √

$d^2(q1,4)=0^2+1^2+3^2=10 > \varepsilon^2/2$ X

$d^2(q1,5)=2^2+2^2+4^2=24 > \varepsilon^2/2$ X

…. remaining positions have sq. distance > $\varepsilon^2/2$

S1 = {1,2,3}

Subsequence Matching: Example

$$\begin{array}{l} \quad\;\; 1\;\; 2\;\; 3\;\; 4\;\; 5\;\; 6\;\; 7\;\; 8\;\; 9\;\; 10\;\; 11\;\; 12\;\; 13\;\; 14\;\; 15\;\; 16 \\ T=\{3,3,4,3,5,6,7,7,8,9,10,11,9,8,9,10\} \end{array}$$

$q2=\{5,6,5\}$

$\varepsilon^2/2=8$

$d^2(q2,4)=2^2+1^2+1^2=6 \le \varepsilon^2/2$ √

$d^2(q2,5)=0^2+0^2+2^2=4 \le \varepsilon^2/2$ √

$d^2(q2,6)=1^2+1^2+2^2=6 \le \varepsilon^2/2$ √

…. remaining positions have sq. distance > $\varepsilon^2/2$

S2 = {4,5,6}

…we also computed S1 = {1,2,3}

Positions in S1 correspond to candidate positions for q, positions in S2 correspond to candidate positions for q, shifted by 3 time units on the right.

→ therefore, positions for q that correspond to S2 are {1,2,3}

P = merge(S1,S2) = {1,2,3}

We only need to check those positions for finding out subsequences for which q = {3,4,3,5,6,5} is at most ε=4 distance away

Subsequence Matching: Example

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

T={3,3,4,3,5,6,7,7,8,9,10,11,9,8,9,10}

$d^2(q,1)=0+1^2+1^2+2^2+1^2+1^2=8 \leq \varepsilon^2$ ⇨ result!

$d^2(q,1)=0+0+0+0+0+2^2=4 \leq \varepsilon^2$ ⇨ result!

$d^2(q,1)=0+0+0+0+0+2^2=4 \leq \varepsilon^2$ ⇨ result!

q={3,4,3,5,6,5}

$\varepsilon^2=16$

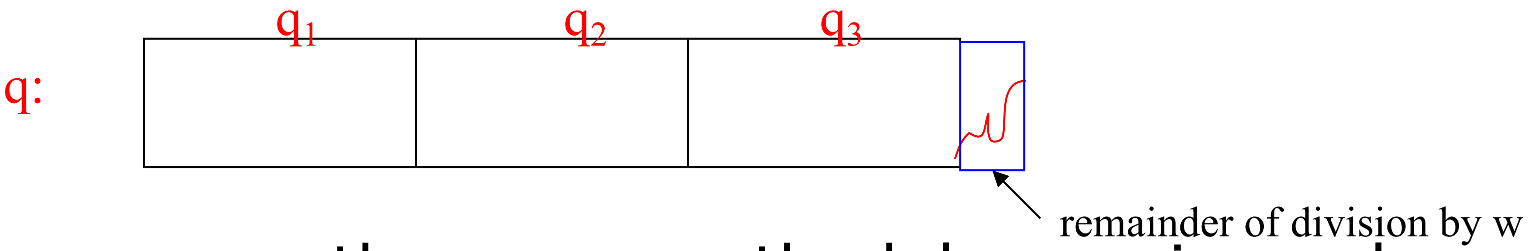
P = {1,2,3}

Access positions in P and verify results!

In fact, in this example, we could be sure that P = {1,2,3} is the result, without checking (WHY?)

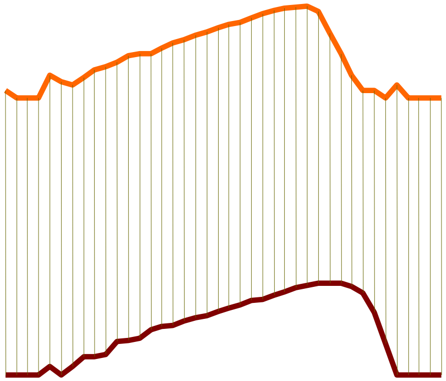
Subsequence Matching (cont'd)

- What happens if q is not a multiple of w?



- Q: Can we use the same methodology using only q1,q2,q3?
  - Yes: We can prove that if D(q,s) ≤ ε then there for at least one of q1,q2,q3, D(qi,si) ≤ ε/√3.
  - Based on the following truth:
    - If $D(q_{1..n},s_{1..n}) \leq \varepsilon$ for n-length sequences $q_{1..n}$ and $s_{1..n}$, then for any subsequence pair $q_{i..j},s_{i..j}$, $1 \leq i \leq j \leq n$, also $D(q_{i..J},s_{i..J}) \leq \varepsilon$ holds

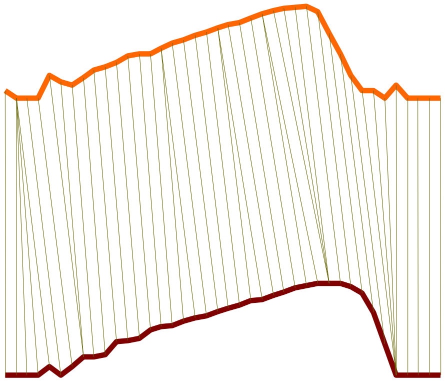
Dynamic Time Warping



Fixed Time Axis

Sequences are aligned "one to one".

e.g., Euclidean distance



"Warped" Time Axis

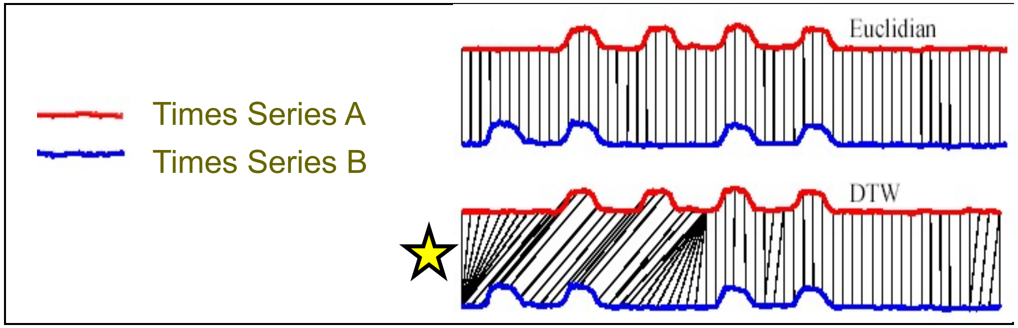
Nonlinear alignments are possible.

- DTW: A more robust distance measure between time-series
- considers shifting/distortion in time

Why Dynamic Time Warping?

Classification experiment on Cylinder-Bell-Funnel dataset

- Training data consists of 10 exemplars from each class.
- (One) Nearest Neighbor Algorithm.
- "Leaving-one-out" evaluation, averaged over 100 runs.



Comparison of the two approaches

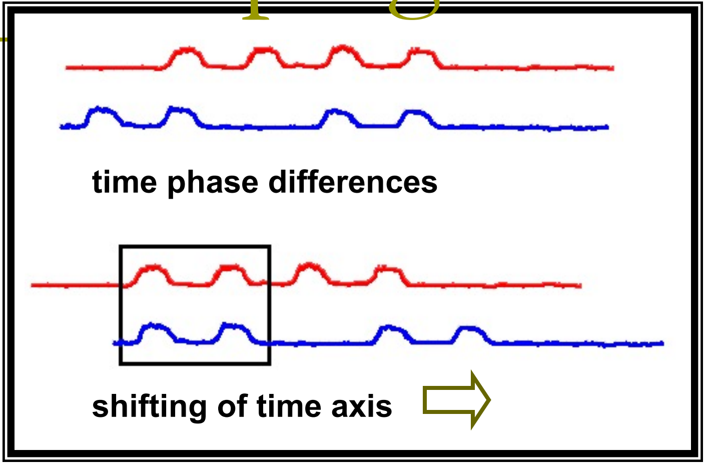
	mean error rate	speed
Euclidean Distance Metric	**0.2734**	★ **X**
Dynamic Time Warping (DTW)	★ **0.0269**	**230X**

- **DTW is very robust, but too slow to apply directly on sequences**
- **We need filtering/indexing techniques for DTW**

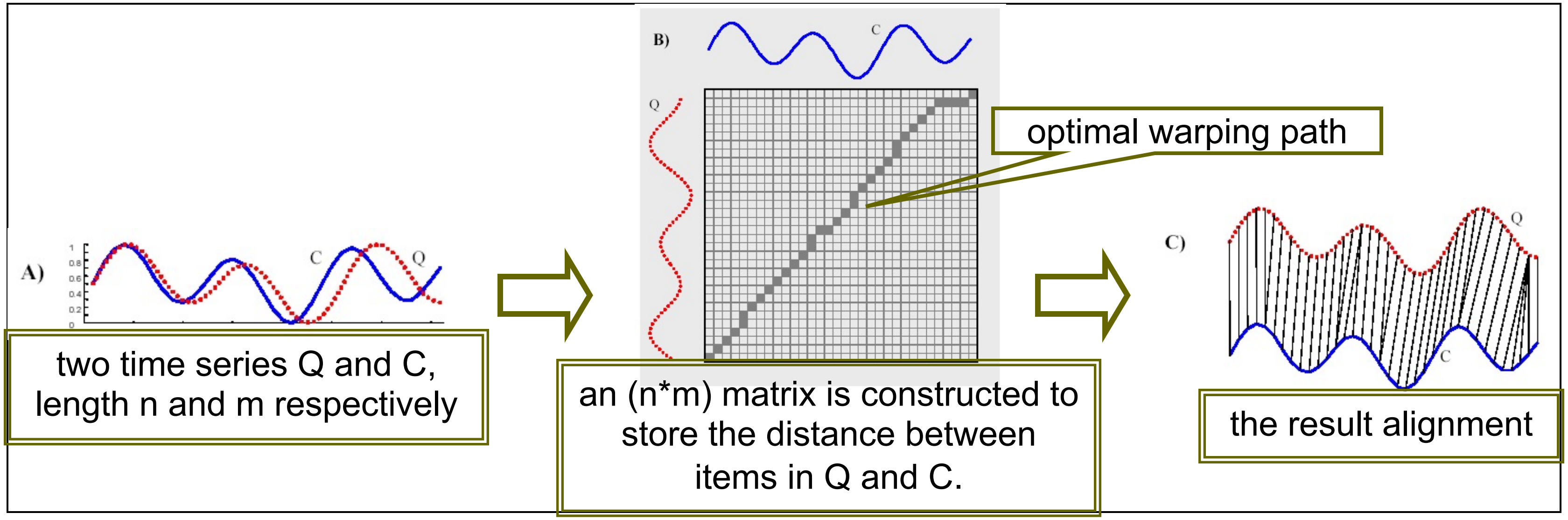
What is Dynamic Time Warping?

Given: two sequences $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$

Objective: align two sequence base on a common time-axis



Aligning time series with Dynamic Programming Matrix



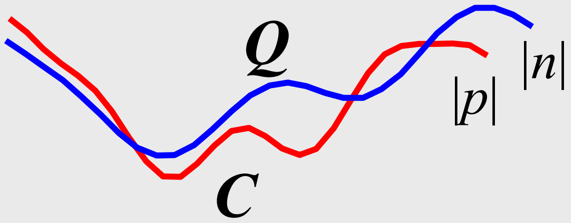
two time series Q and C, length n and m respectively

an (n*m) matrix is constructed to store the distance between items in Q and C.

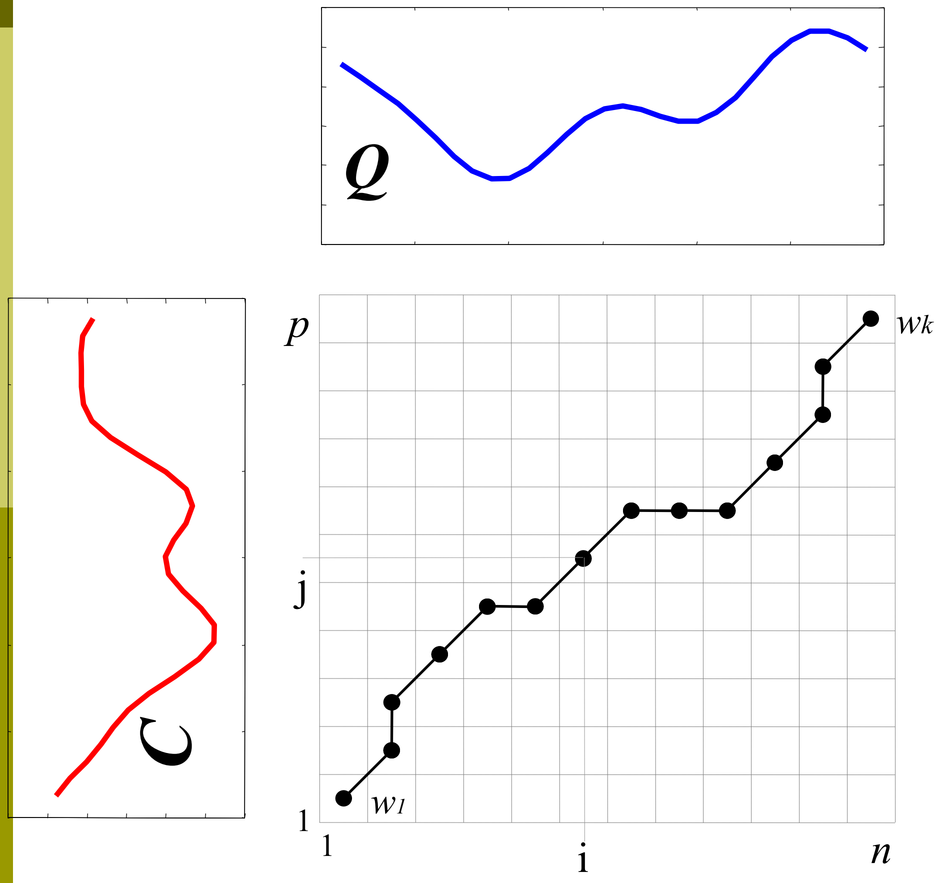
optimal warping path

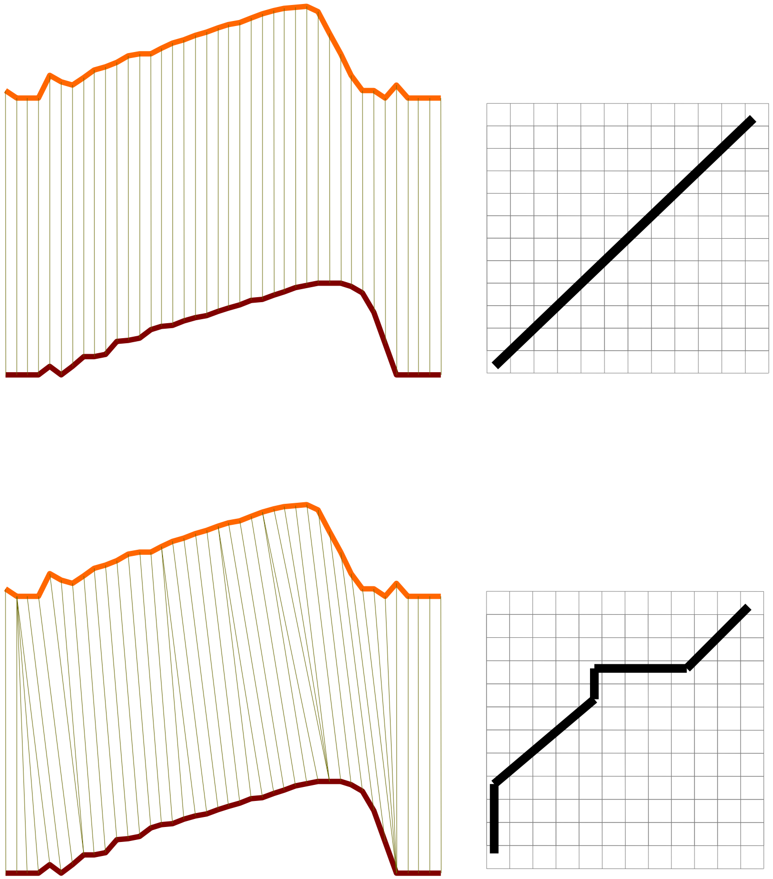
the result alignment

Computing the Dynamic Time Warp Distance I

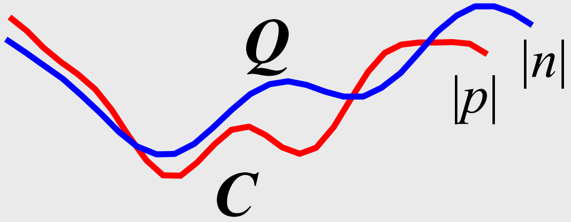


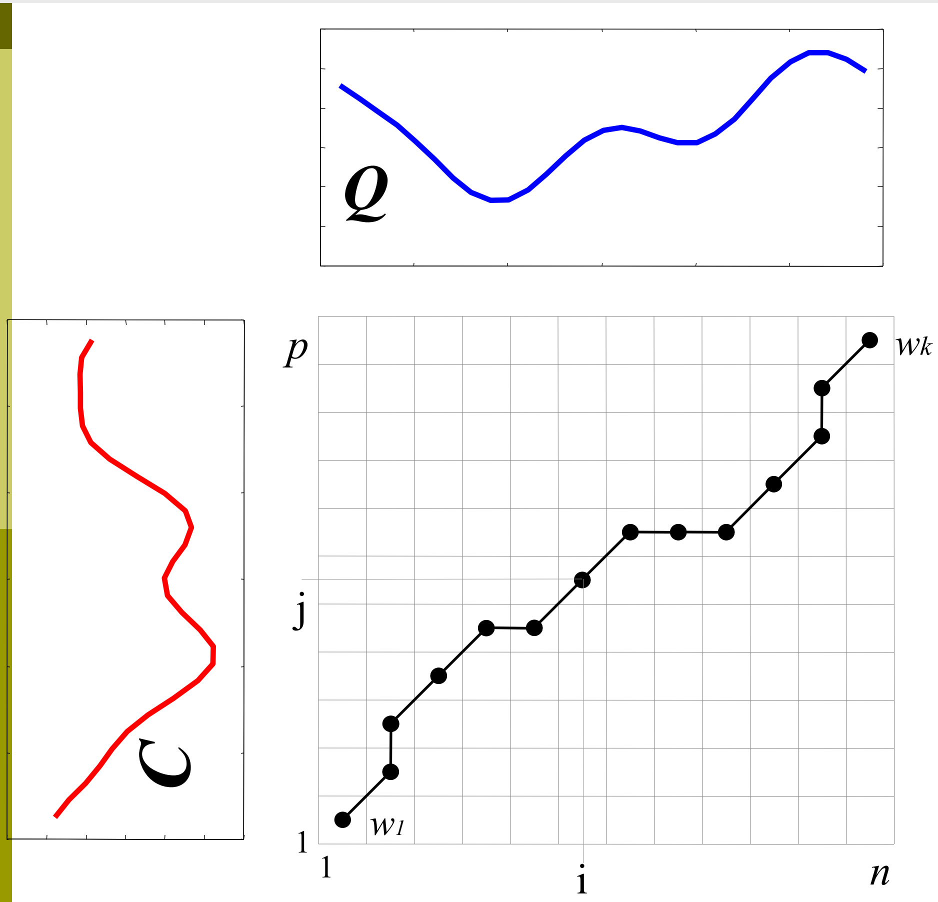
Note that the input sequences can be of different lengths





Computing the Dynamic Time Warp Distance II





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…

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

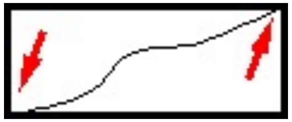
$$\gamma(\mathrm{i},\mathrm{j}) = d(q_\mathrm{i},c_\mathrm{j}) + \min\{\ \gamma(i\text{-}1,j\text{-}1)\ ,\ \gamma(i\text{-}1,j\ )\ ,\ \gamma(i,j\text{-}1)\ \}$$

Warping Constraints

- There're three basic constraints for time warping

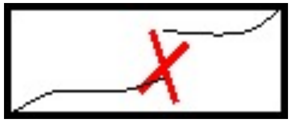
Boundary conditions

-the first (last) element of one sequence should match with the first (last) element of the other



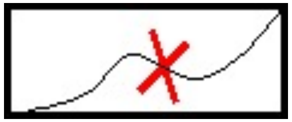
Continuity

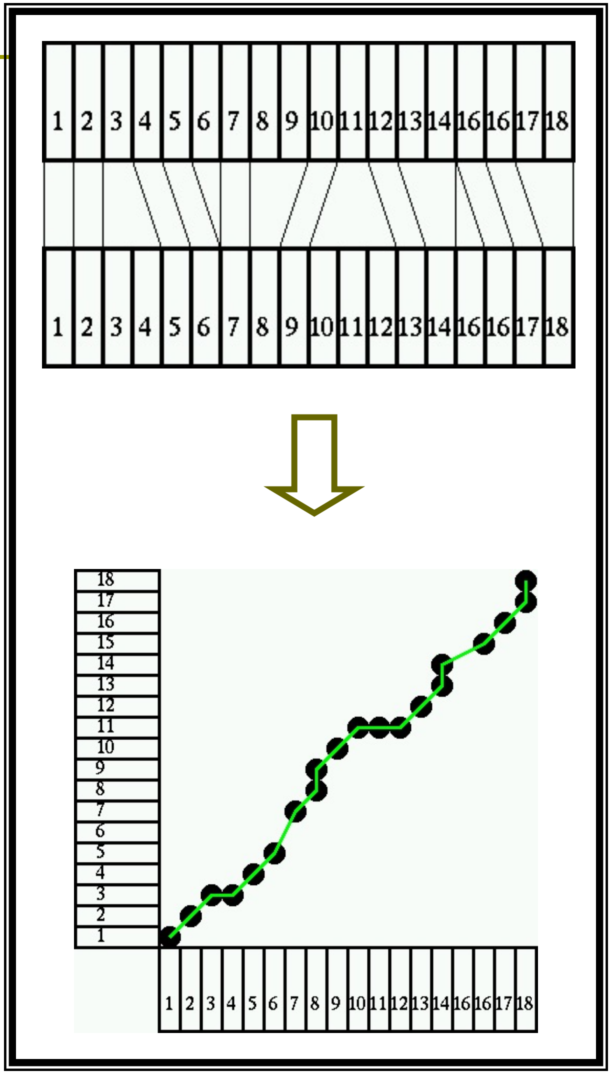
-no jumps



Monotonicity

- we can't go back in time





$$\gamma(\mathrm{i,j}) = d(q_i,c_j) + \min\{\, \gamma(i-1,j-1) \,,\, \gamma(i-1,j\,) \,,\, \gamma(i,j-1) \,\}$$

Unit distance:
$$d(q_i,c_j) = |q_i-c_j|^2$$

DTW Example

- q=<-0.06, 0.46, -0.64, -2.23, 0.09, 0.04, -0.30, 0.90, 1.74>
- s=<1.88, 2.78, 1.22, -1.10, -1.75, -0.10, -0.31, -1.43, -1.18>

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
q_1	3.76	8.07	1.64	1.08	2.86	0.00	0.06	1.88	1.25
q_2	2.02	5.38	0.58	2.43	4.88	0.31	0.59	3.57	2.69
q_3	6.35	11.70	3.46	0.21	1.23	0.29	0.11	0.62	0.29
q_4	16.89	25.10	11.90	1.28	0.23	4.54	3.69	0.64	1.10
q_5	3.20	7.24	1.28	1.42	3.39	0.04	0.16	2.31	1.61
q_6	3.39	7.51	1.39	1.30	3.20	0.02	0.12	2.16	1.49
q_7	4.75	9.49	2.31	0.64	2.10	0.04	0.00	1.28	0.77
q_8	0.96	3.53	0.10	4.00	7.02	1.00	1.46	5.43	4.33
q_9	0.02	1.08	0.27	8.07	12.18	3.39	4.20	10.05	8.53

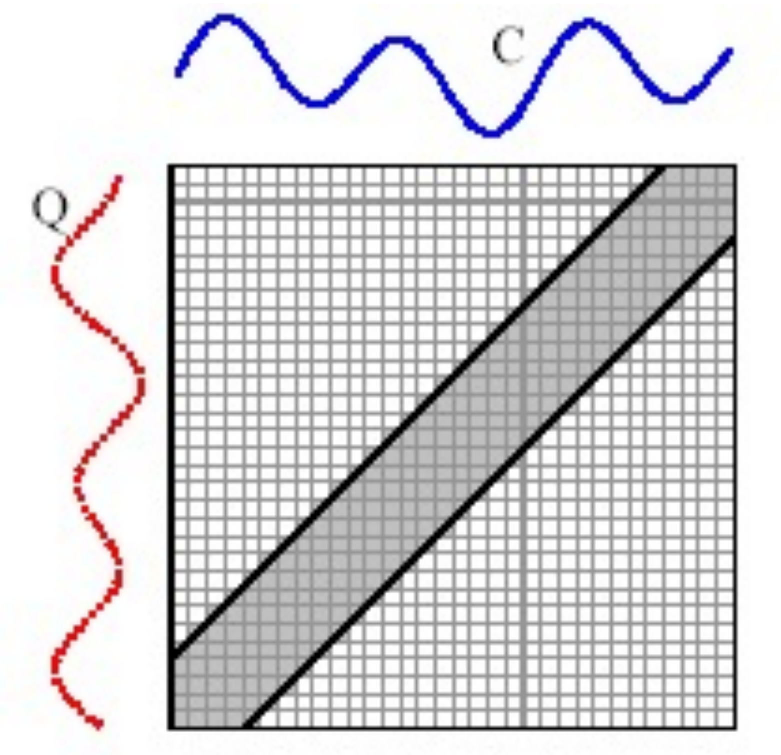
distance matrix (d)
$d[i][j] = d(q_i,s_j)^2$

DTW dist = √24.18=4.9

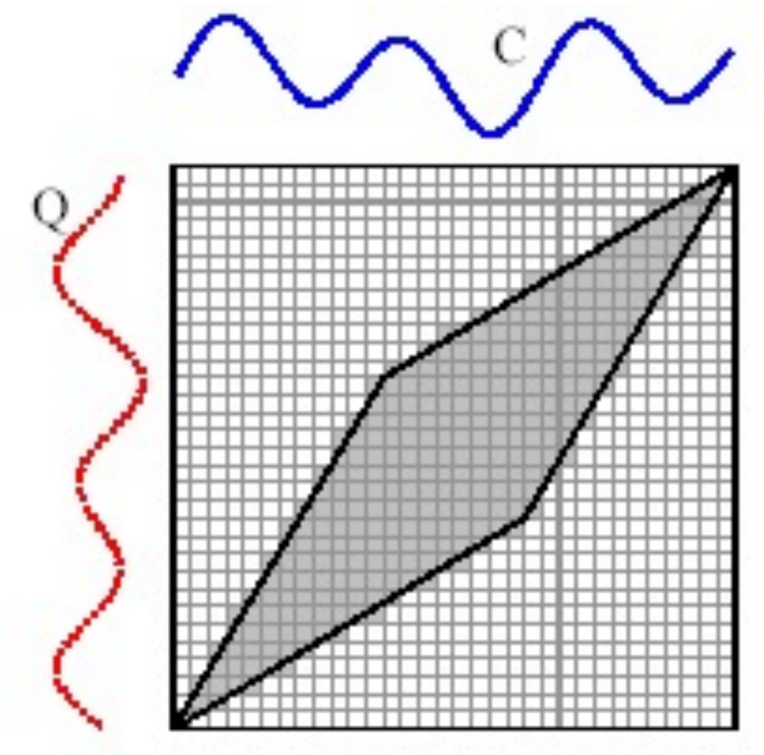
warping matrix (γ)
$\gamma[i][j] = d(q_i,s_j)^2 +$
$\min\{\gamma[i-1][j-1],\ \gamma[i-1][j],\ \gamma[i][j-1]\}$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
q_1	**3.76**	11.83	13.47	14.55	17.41	17.41	17.47	19.35	20.60
q_2	5.78	**9.14**	**9.72**	12.15	17.03	17.34	17.93	21.04	22.04
q_3	12.13	17.48	12.60	**9.93**	11.16	11.45	11.56	12.18	12.47
q_4	29.02	37.23	24.50	11.21	**10.16**	14.70	15.14	12.20	13.28
q_5	32.22	36.26	25.78	12.63	13.55	**10.20**	10.36	12.67	13.81
q_6	35.61	39.73	27.17	13.93	15.83	**10.22**	10.32	12.48	13.97
q_7	40.36	45.10	29.48	14.57	16.03	10.26	**10.22**	11.50	12.27
q_8	41.32	43.89	29.58	18.57	21.59	11.26	11.68	**15.65**	15.83
q_9	41.34	42.40	29.85	26.64	30.75	14.65	15.46	21.73	**24.18**

DTW speed-up with warping constraints



Sakoe-Chiba Band



Itakura Parallelogram

Allow each element from Q to match only with a range of elements from C (and vice versa)

Indexing for DTW

- Several methods have been proposed for indexing time-series for time-warping similarity.
- These methods extract some abstractions of the series that can be indexed and define lower bounds for DTW distance using these approximations.

DTW similarity search speed-up with lower bounds

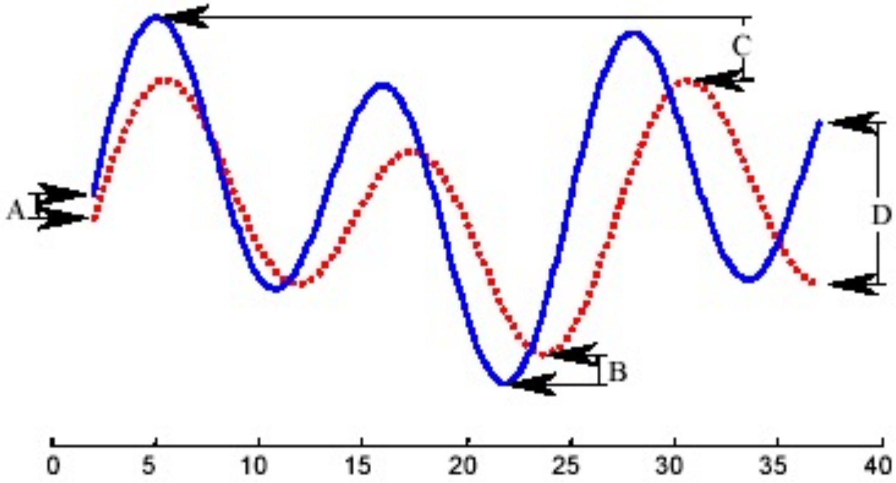
LB(Q,C) ≤ DTW(Q,C)

How to define a good Lower Bounding Function ?

A good lower bound should

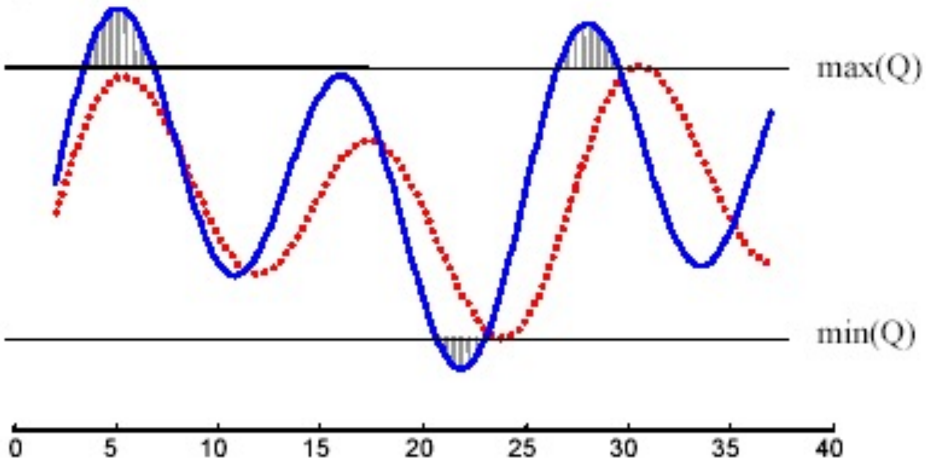
- be fast to compute
- be a relatively tight lower bound (in order to minimize exact DTW computations)

[LB_Kim]



max of first (A), last (D), min (B) and max (C) differences

[LB_Yi]



sum of squared diff of elements outside the min_max, max_min band

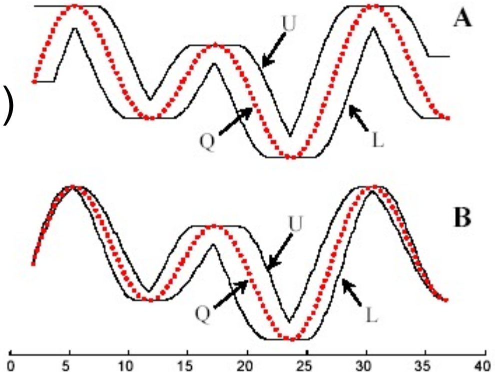
A lower bound using warping constraints

Notation

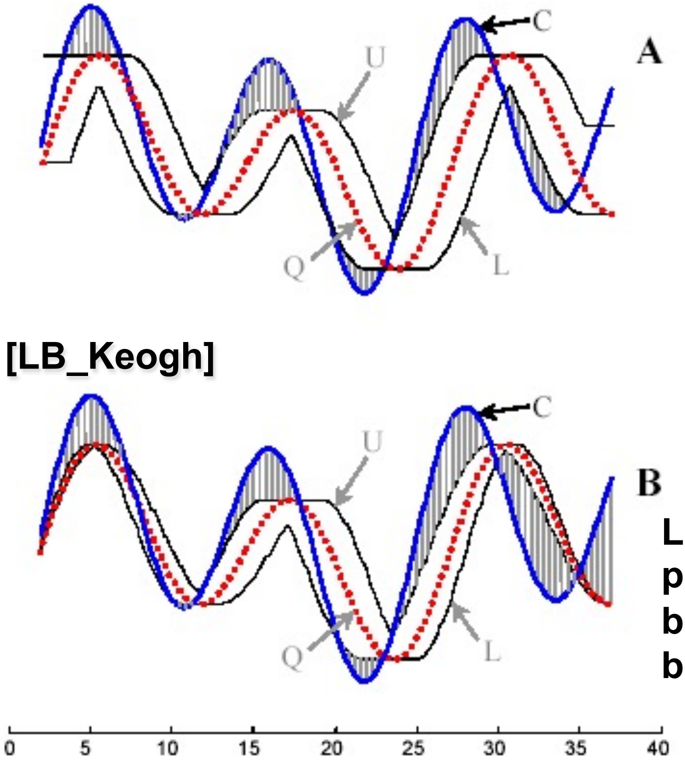
A: bounding envelope - **Sakoe-Chiba Band (**global constraint)

B: bounding envelope - **Itakura Parallelogram (**global constraint)

Q: original sequence   U: Upper   L: Lower



[LB_Keogh]

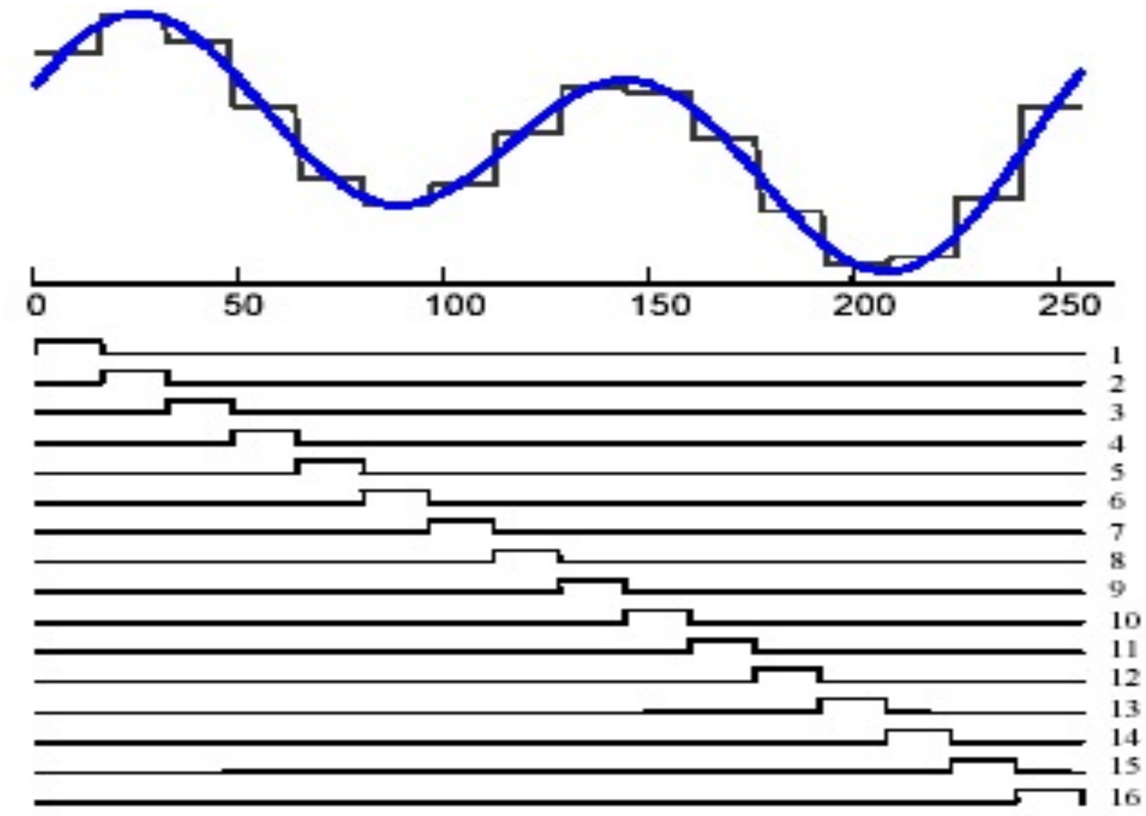


The envelope denotes the range within which each element of Q can be matched with some element of C

LB_Keogh = the squared sum of the distances from every part of the candidate sequence C not falling within the bounding envelope, to the nearest orthogonal edge of the bounding envelope.

Indexing time-series for DTW matching

- Step 1: Approximate each time-series with a Piecewise Constant Approximation (PAA)



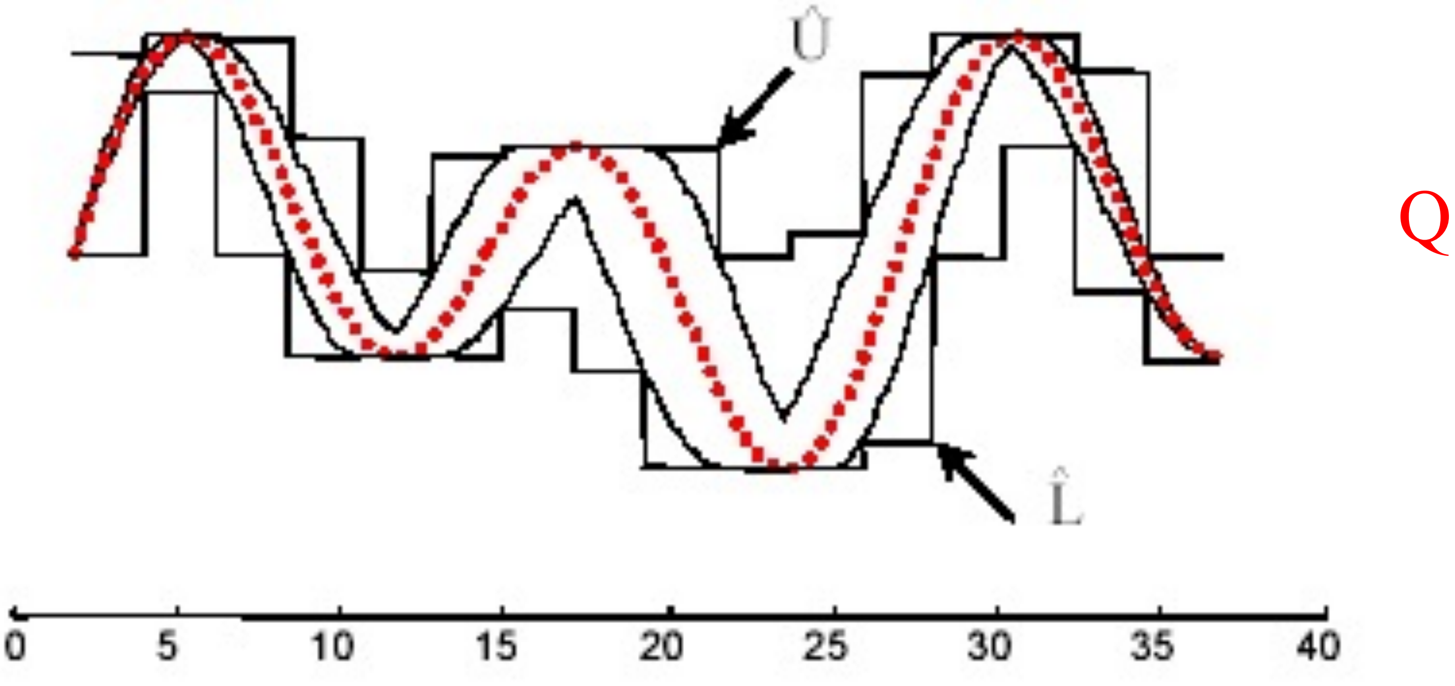
E.g., a sequence of length 256 is reduced to 16 dimensions

Indexing time-series for DTW matching

- Step 2: Index the approximations (low-dimensional space) using an R-tree
  - Each PAA segment position corresponds to a dimension
  - E.g. 16 segments in previous slide means a 16-dimensional R-tree
- Issue:
  - From the PAA approximations derive a lower bound for DTW distance

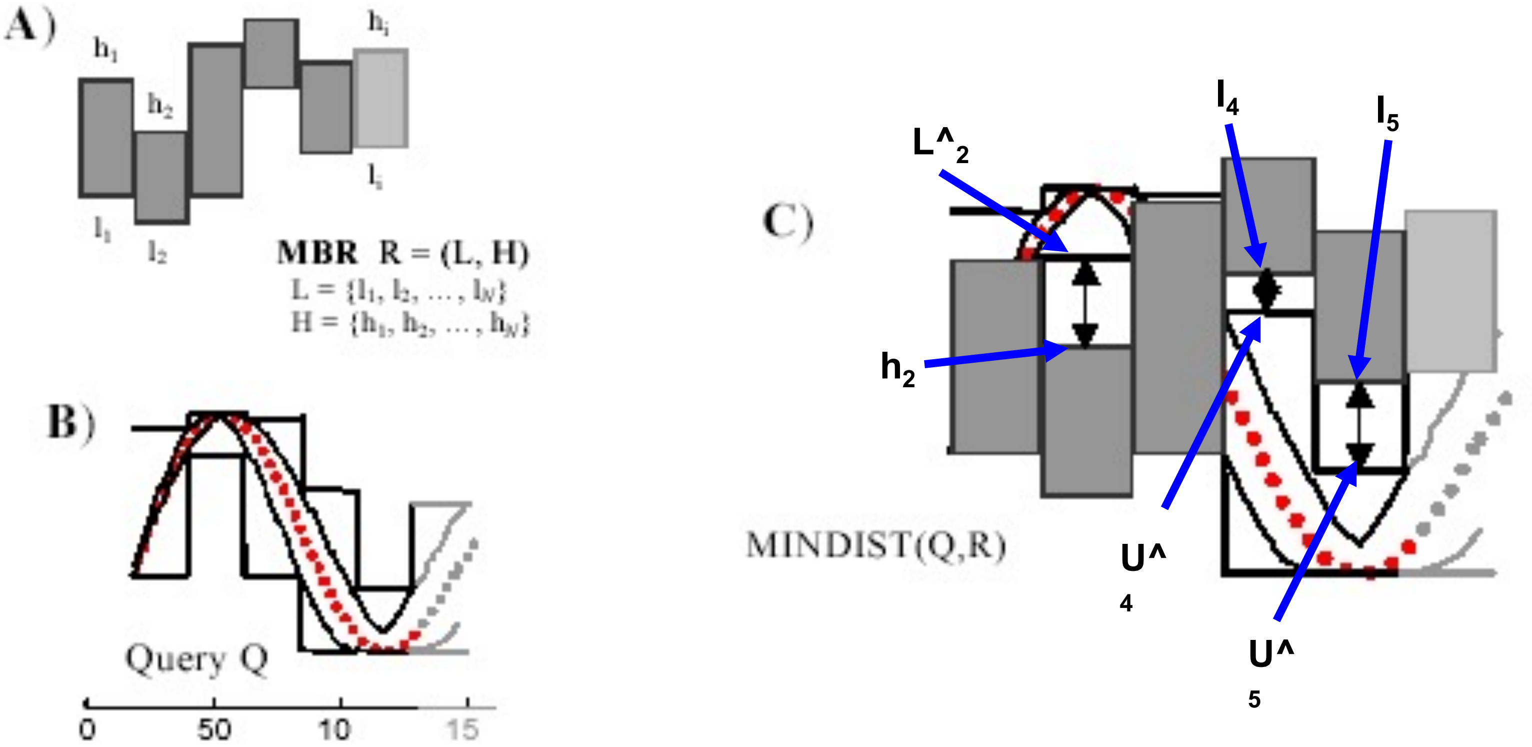
Indexing time-series for DTW matching

- Approximate the bounding envelope of the query sequence by a low-dimensional MBR:



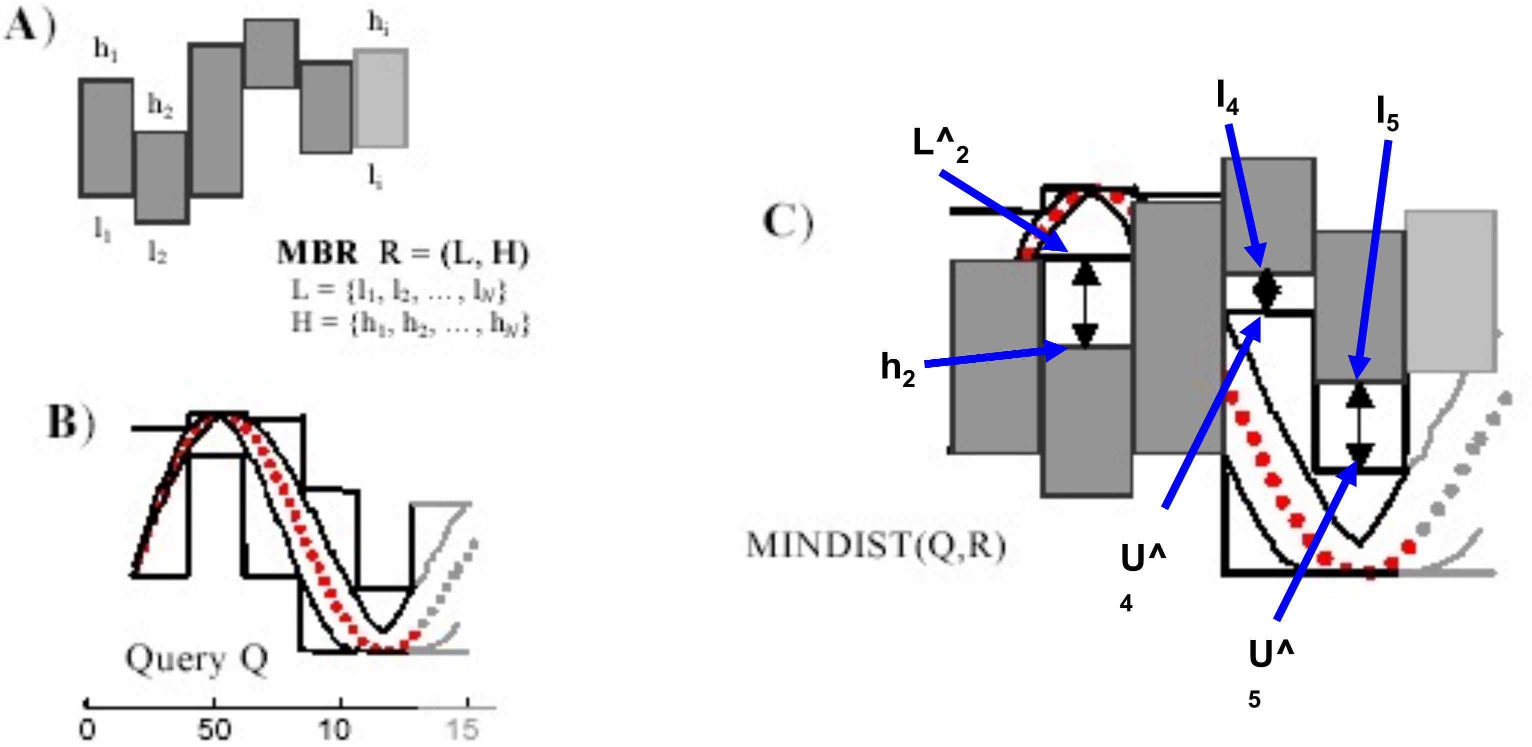
Indexing time-series for DTW matching

- Use the R-tree MBRs corresponding to MBRs of groups of sequence PAAs



Indexing time-series for DTW matching

- Thus, we can derive a distance bound between Q and any sequence in R



L^$_2$

l$_4$

l$_5$

h$_2$

U^ 4

U^ 5

Indexing time-series for DTW matching

- Range similarity queries and Nearest Neighbor queries can be now processed in two steps:
  - Use the R-tree of the low PAA approximations to find candidate sequences with LB_PAA(C,Q)≤ε.
  - For each candidate C, compute exact DTW

Summary

- Images and complex multimedia objects in general are approximated by feature vectors.
- Similarity search is performed at the feature space
  - Feature-based similarity is the typical way to model multimedia object similarity
- Indexing and searching high-dimensional feature vectors is expensive
  - Use low-dimensional approximations, define distance at the low-dimensional space, and lower bounds
  - Use multi-step query evaluation
  - Use compression or metric-space indexing
- Time-series data are also treated as multi-dimensional feature vectors
  - Subsequence search is a special problem for time-series dsta
  - Dynamic Time Warping is a more effective distance measure compared to Minkowski distance measures