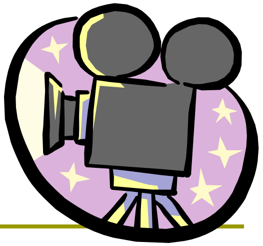
Multidimensional Data

- 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
- Subsequence matching in time-series

Similarity search

- Record data representation
- Given an object (record) o find
  - similar objects to o according to some upper distance bound ε
  - k-most similar objects to o
- Applications
  - multimedia search-by-example
  - data mining (clustering, NN-based classification)
- Issues
  - bad performance of spatial indexes in high-dimensional spaces
  - curse of dimensionality

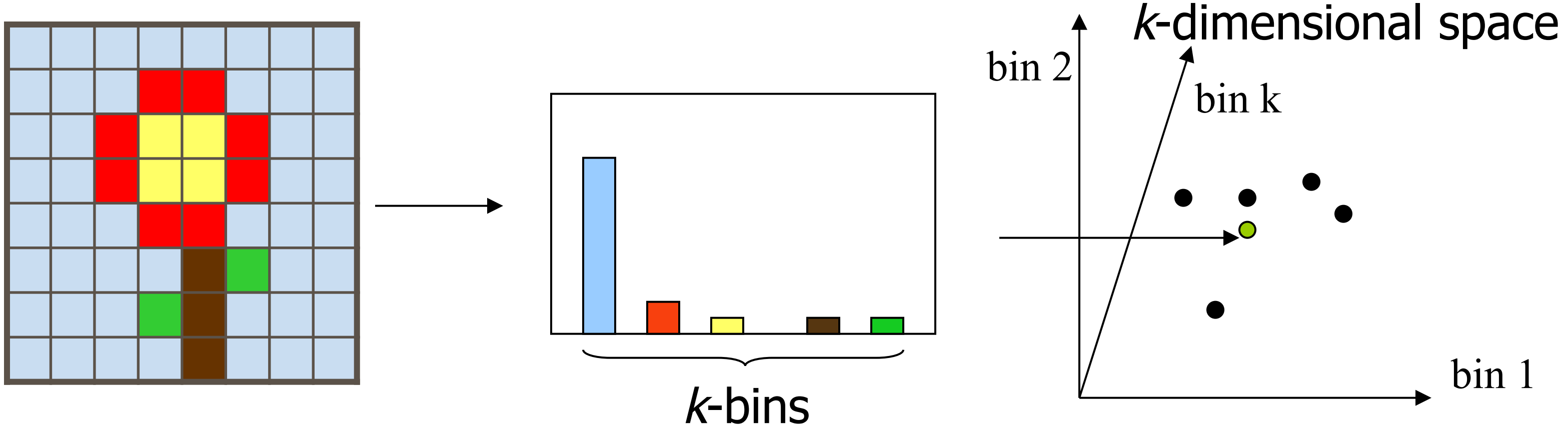
Application: Multimedia Data



- 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

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

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

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

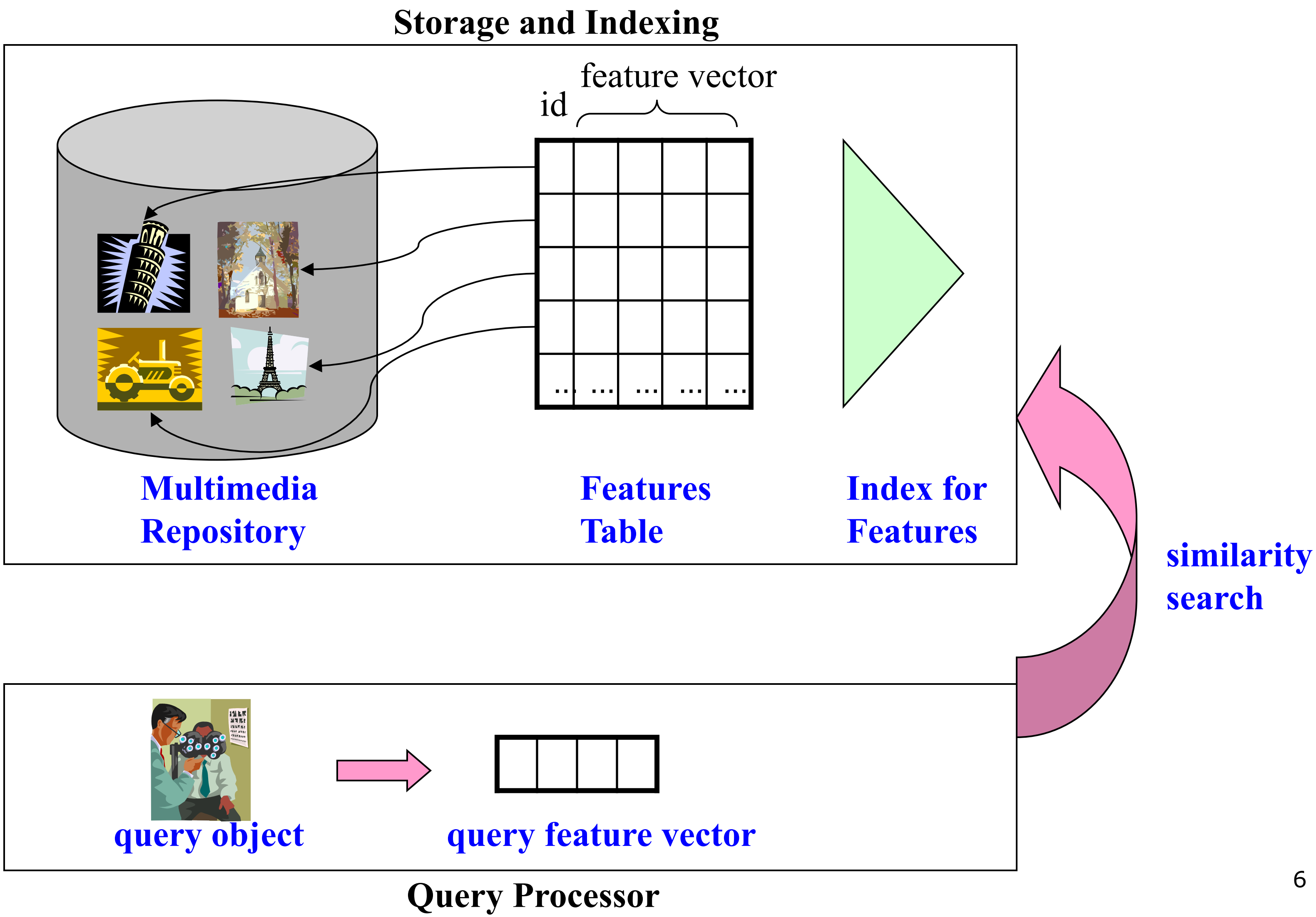
- also
  - Manhattan (city block) distance

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

  - Max (supremum) distance

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

Architecture of a Multimedia Database



Storage and Indexing

feature vector

id

Multimedia Repository

Features Table

Index for Features

similarity search

query object

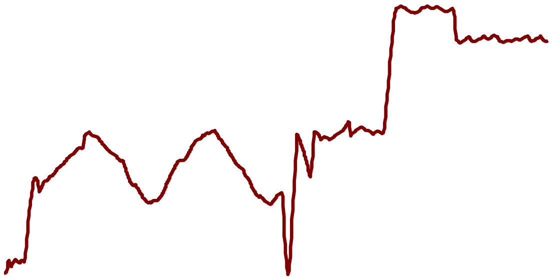
query feature vector

Query Processor

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)

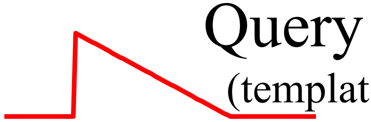
AABBBSSBABAGGTSBAK

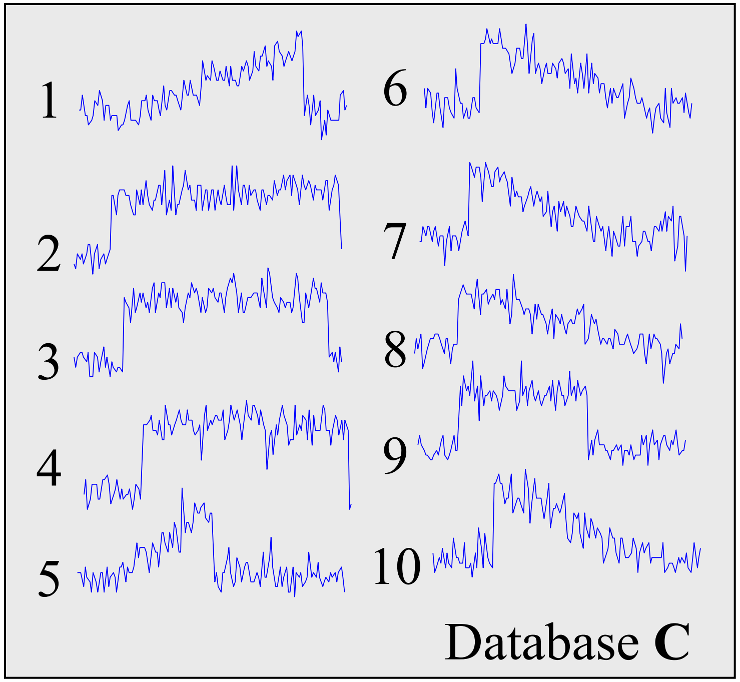
Queries and Analysis Tasks on Time-series Data

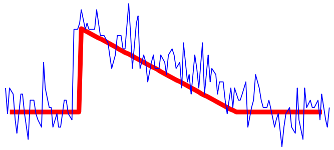
- find the most similar sequence to a query sequence q

1: Whole Matching

Query *Q*
(template)





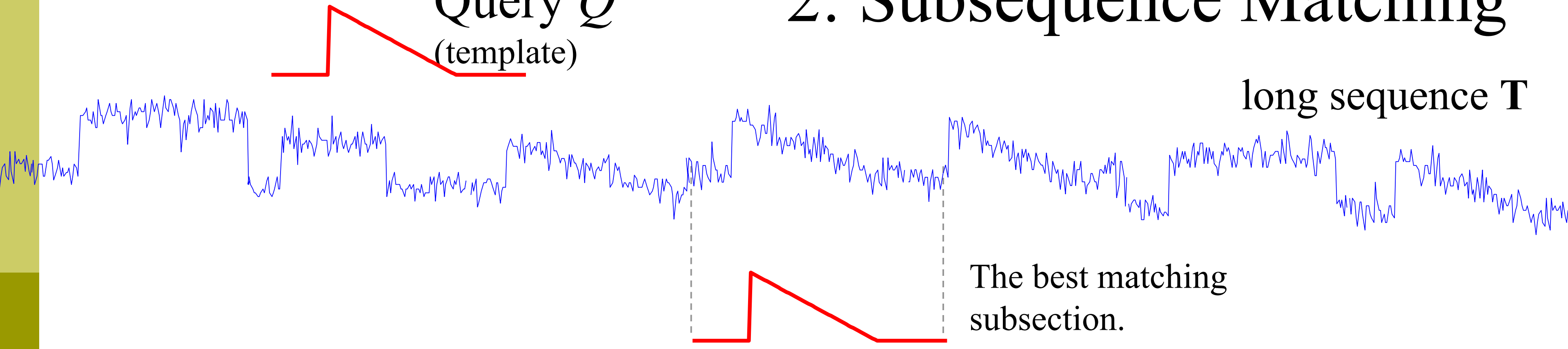


C_6 is the best match.

Queries and Analysis Tasks on Time-series Data

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

2: Subsequence Matching

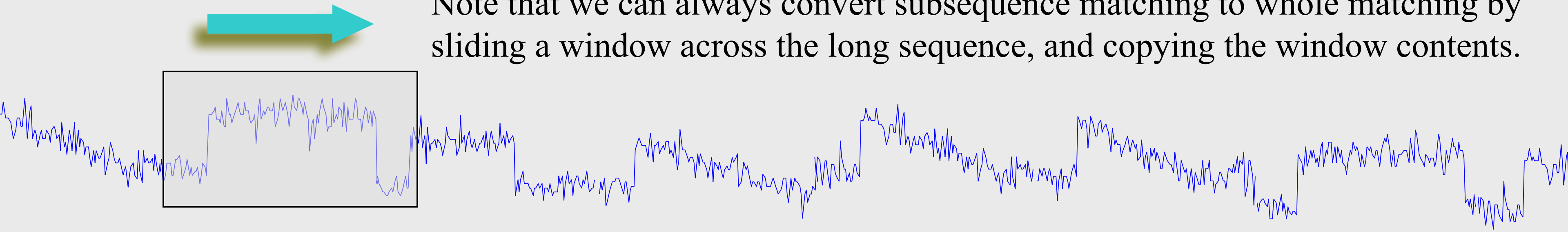


Query *Q* (template)

long sequence **T**

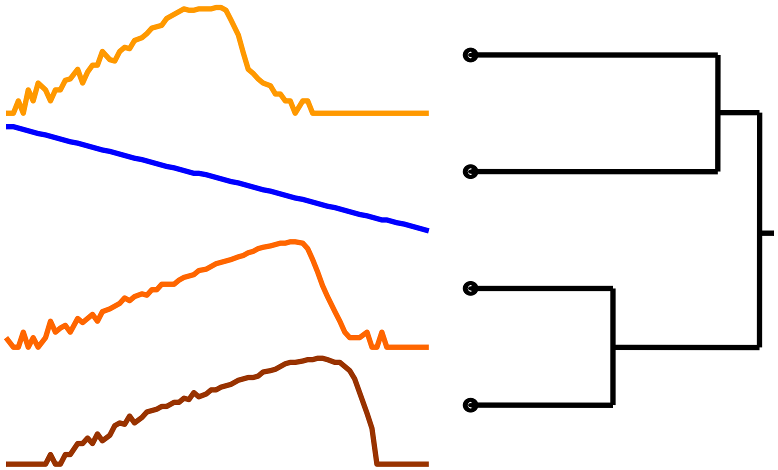
The best matching subsection.

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

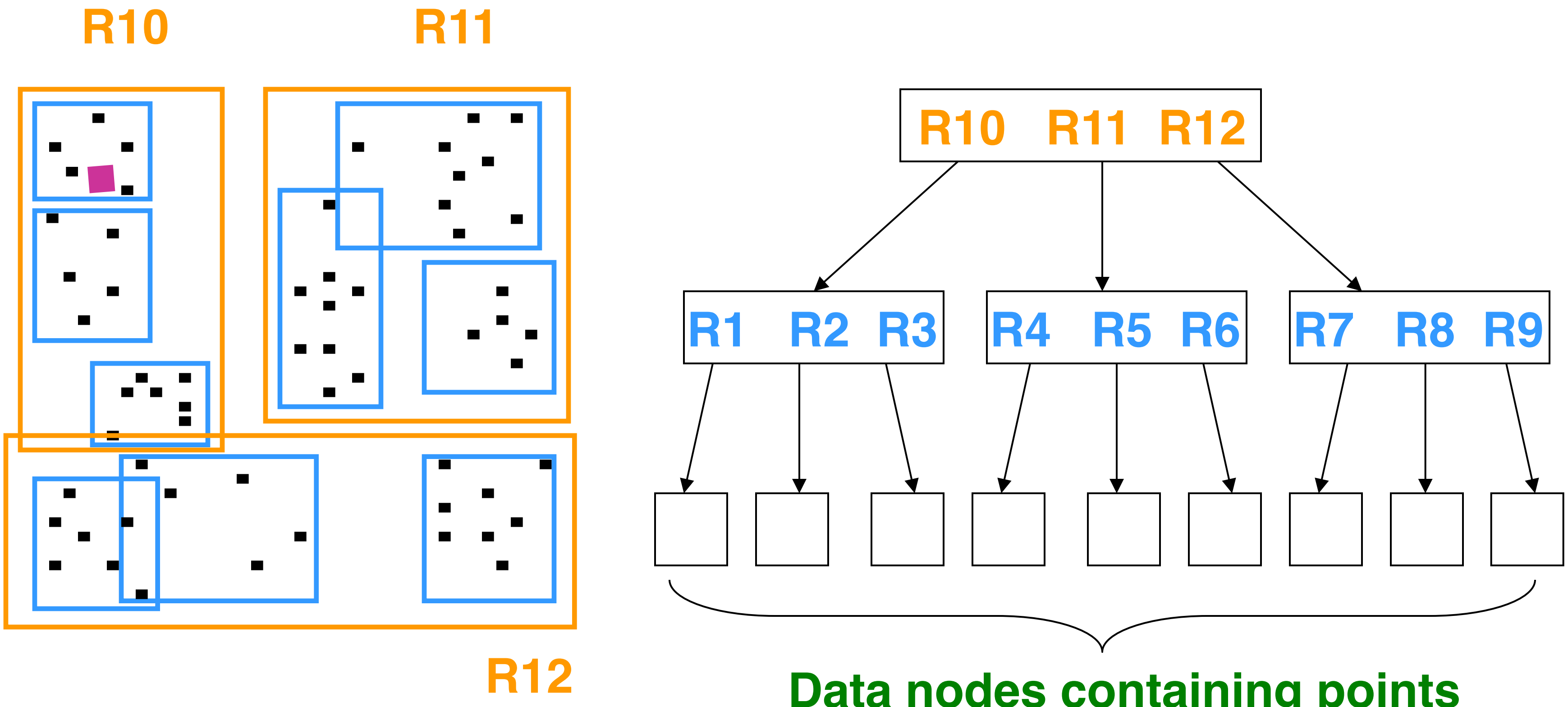
- Classification and Clustering (same are found in multimedia databases)



- 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

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



Problem!

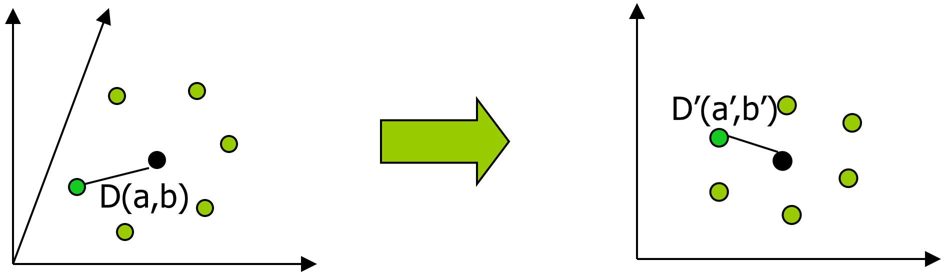
- The R-tree does not scale well for many dimensions. Somewhere above 6-20 dimensions search using the R-tree is slower compared to linear scan
  - large MBRs, a lot of empty space
  - not all dimensions are used for splitting
    - a query point is inside ALL MBRs in most dimensions!
- Feature vectors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

Possible Remedies

- 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
  - approximate search

Dimensionality Reduction

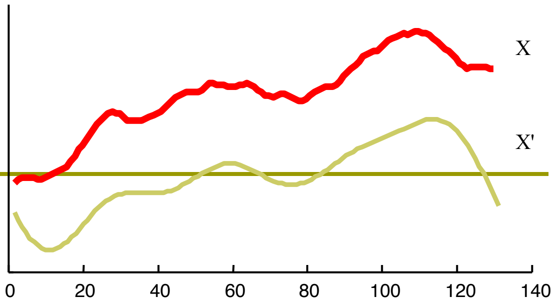
- In many cases the *embedded* dimensionality of a search problem is much lower than the actual dimensionality
- Some methods apply transformations on the data and approximate them with low-dimensional vectors
- The aim is to reduce dimensionality and at the same time maintain the data characteristics



GEMINI GEneric Multimedia INdexIng

- Establish a distance metric 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)
- Produce a distance measure defined on the N dimensional representation of the data, and prove that 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).

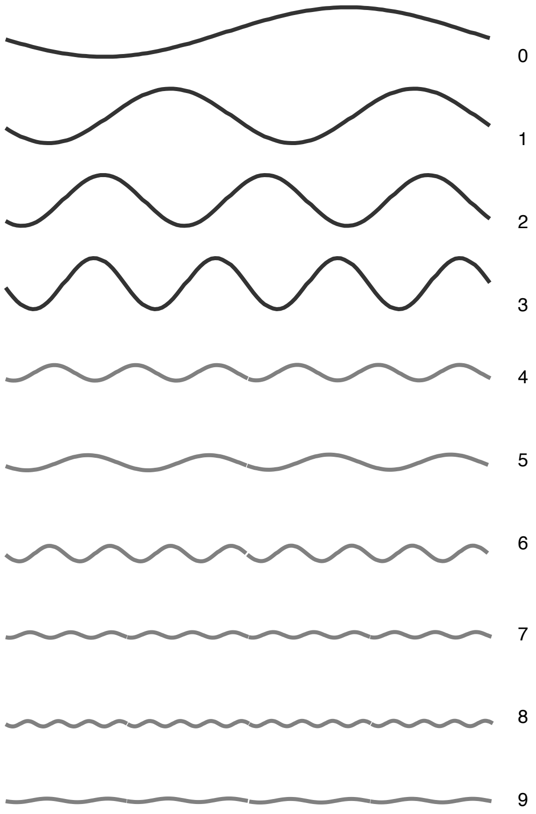
Dimensionality Reduction Example: Discrete Fourier Transform (DFT)



- 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*).



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

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.
  - Piecewise approximations are also used for time-series data.

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 $D(p,q) \leq \varepsilon$

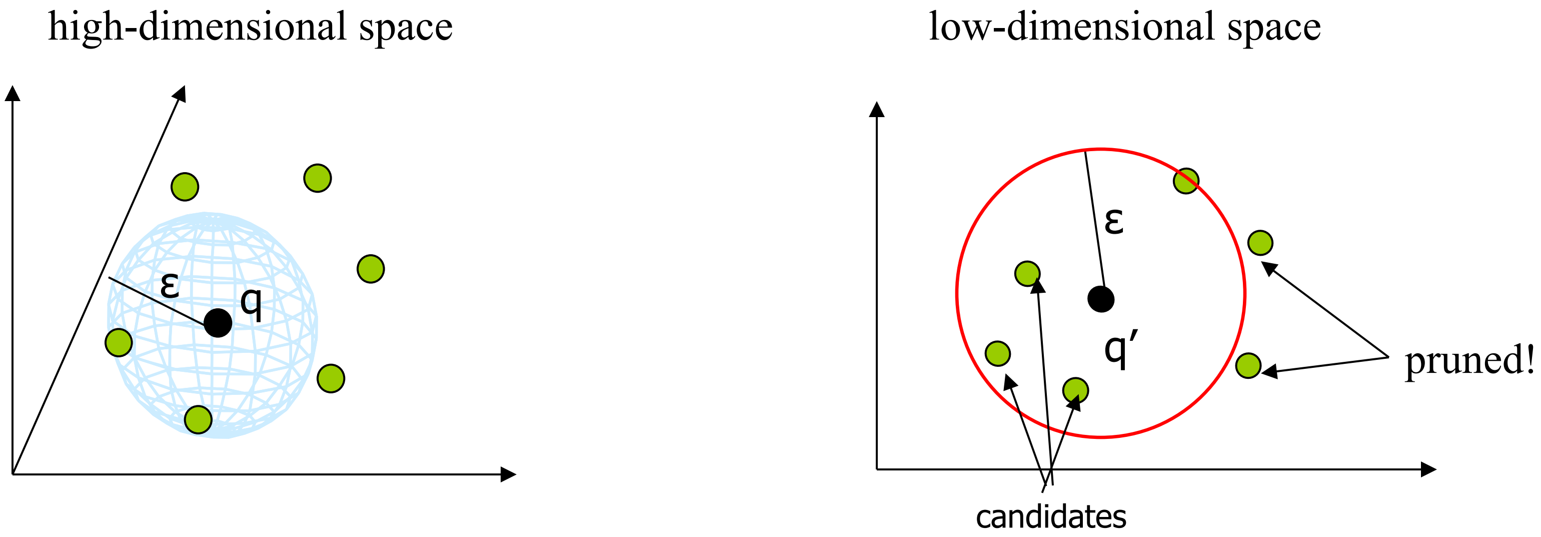
Two-step processing of range similarity queries

- 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

- Step 1:
  - convert q to q′ using the same dimensionality reduction technique
  - apply an R-tree range search to find fast a S′⊆S, such that for all p′ in S′, D′(p′,q′) ≤ ε
- Step 2:
  - for each p′ in S′, get high-dimensional vector p, compute D(p,q) and if D(p,q) ≤ ε 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 $\forall s \in S$, $D(p,q) \leq D(s,q)$

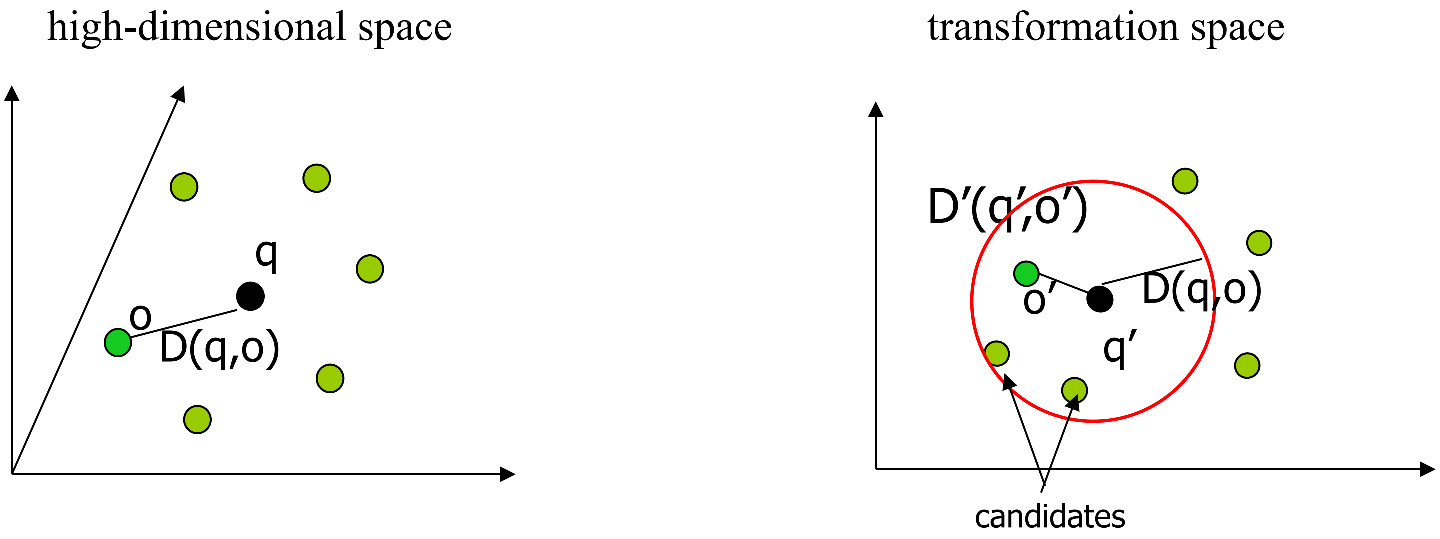
Two-step processing of nearest neighbor similarity queries

- 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

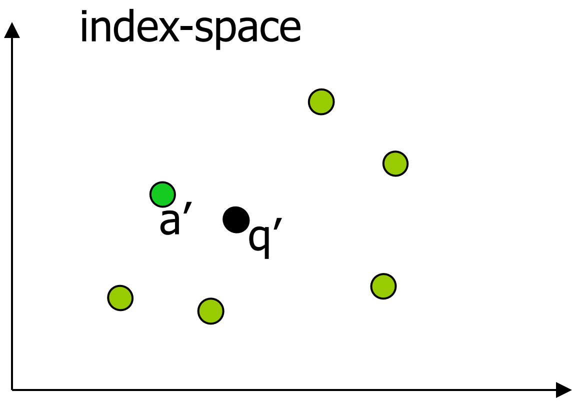
- Q: Will this method compute the correct result? why?

Multi-step processing of nearest neighbor similarity queries

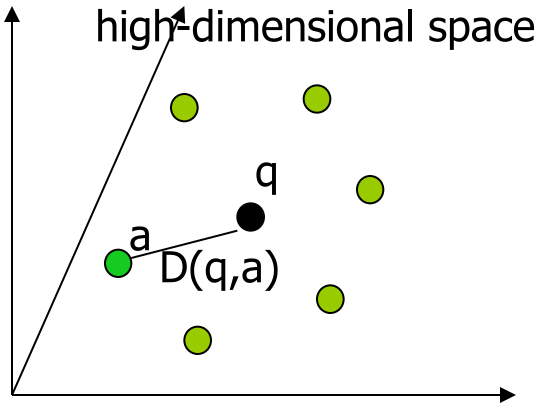
- Convert q to q′ using the same dimensionality reduction technique.
- NN = NULL; D(q,NN) = ∞;
- Repeat:
  - apply an incremental R-tree nearest neighbor search to find fast the next nearest p′ to q′.
  - If D′(q′,p′) < D(q,NN) compute actual D(q,p).
    - If D(q,p)<D(q,NN) then NN = p
- Until D′(q′,p′)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

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

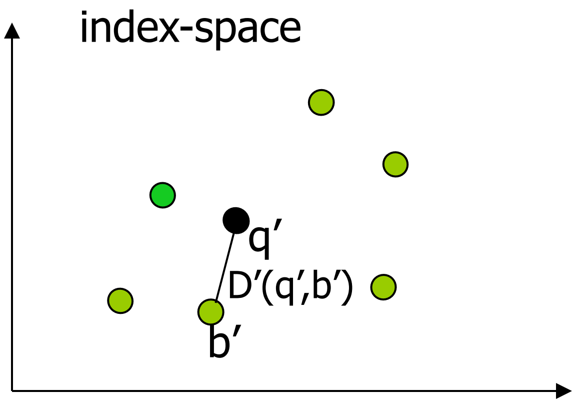
Example of multi-step processing



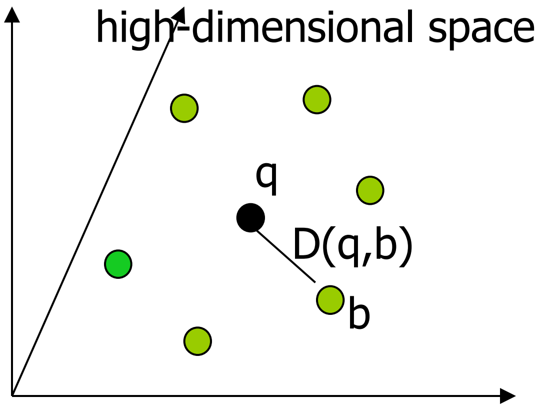
1. Get 1st NN (a′)



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

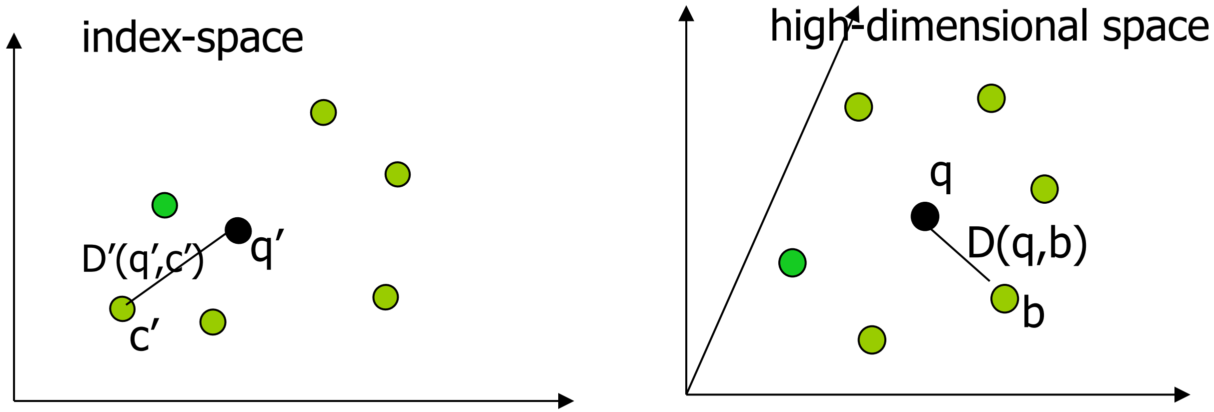


3. Get 2nd NN (a′). D′(q′,b′)< D(q,a), so goto step 4



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 3^{rd} NN (c′). D′(q′,c′)> D(q,b), so terminate and report b as NN.

Compression-based similarity search

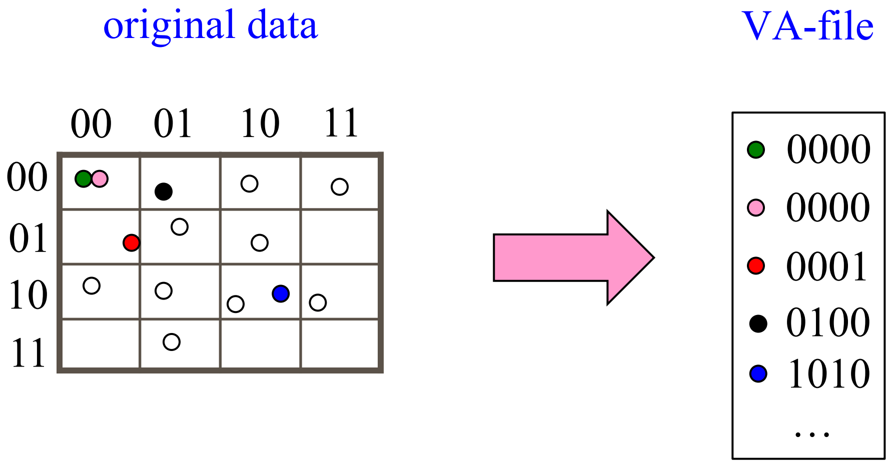
- Motivation:
  - In very high dimensional spaces dimensionality reduction is hard to apply
    - Should examine multiple possible dimension-sets to potentially reduce
  - Sometimes we have to resort to linear scan
    - Expensive because the entire (large) set of feature vectors have to be scanned and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

Compression-based similarity search

- 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

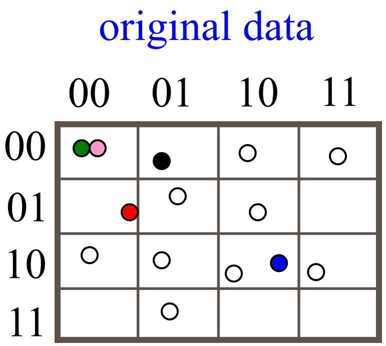
The Vector-Approximation File

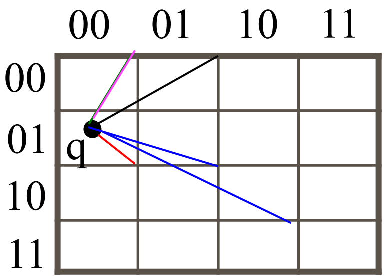
- data preprocessing phase

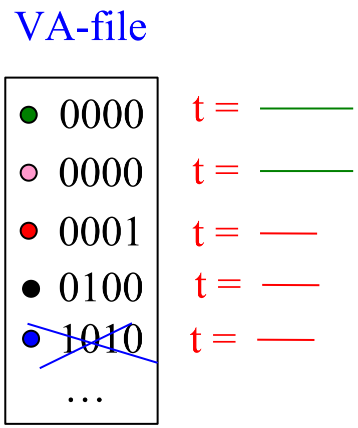


The Vector-Approximation File

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






VA-file

● 0000 t = ——

● 0000 t = ——

● 0001 t = ——

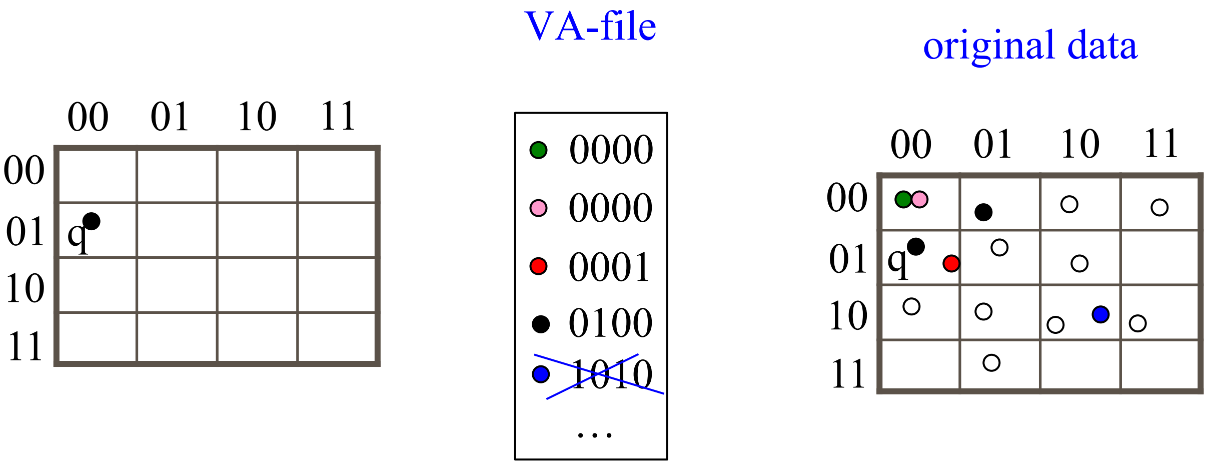
● 0100 t = ——

● ~~1010~~ t = ——

…

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



VA-file

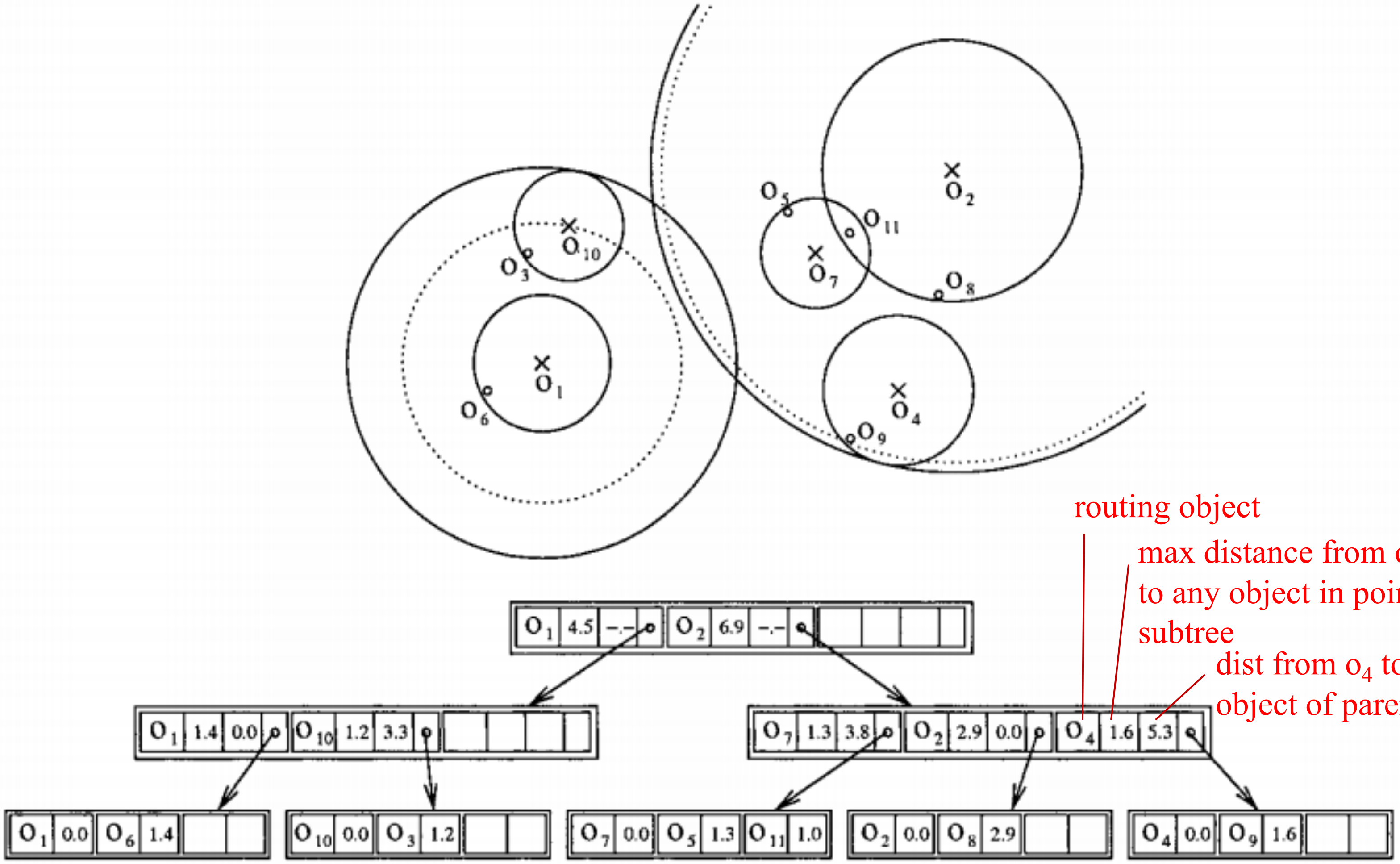
● 0000
○ 0000
● 0001
● 0100
● ~~1010~~
…

original data

Indexing metric spaces: the M-tree

- Idea:
  - Instead of indexing the feature space of the objects, index them by their metric distances
  - Group objects into index nodes hierarchically using only the distances between them
    - 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)

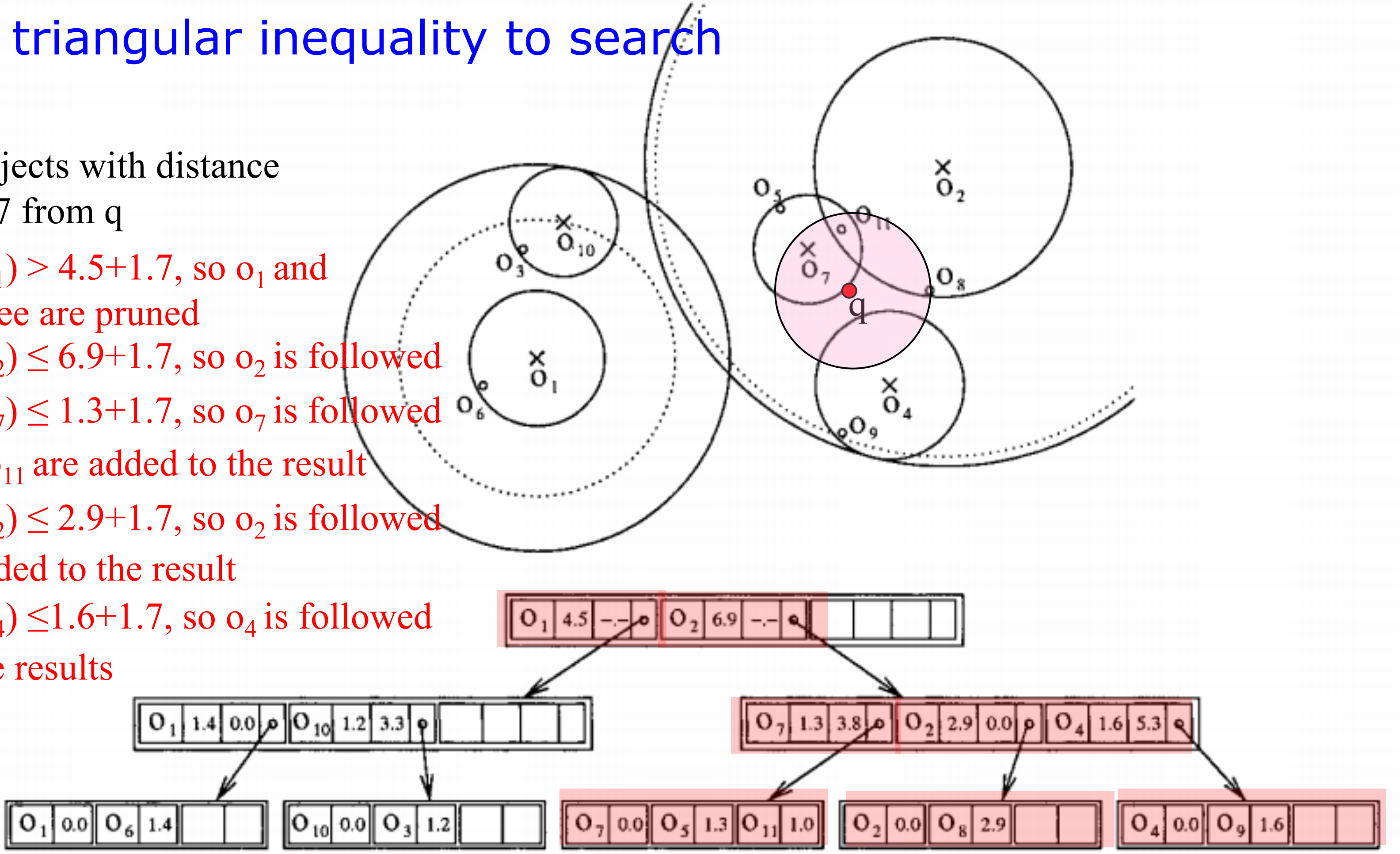
[Figure taken from http://itu-algorithms.github.io/events/2015/05/20/Zezulacourse]

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) ≤ 6.9+1.7, so o_2 is followed
3) dist(q,o_7) ≤ 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) ≤ 2.9+1.7, so o_2 is followed
6) o_8 is added to the result
7) dist(q,o_4) ≤1.6+1.7, so o_4 is followed
8) no more results



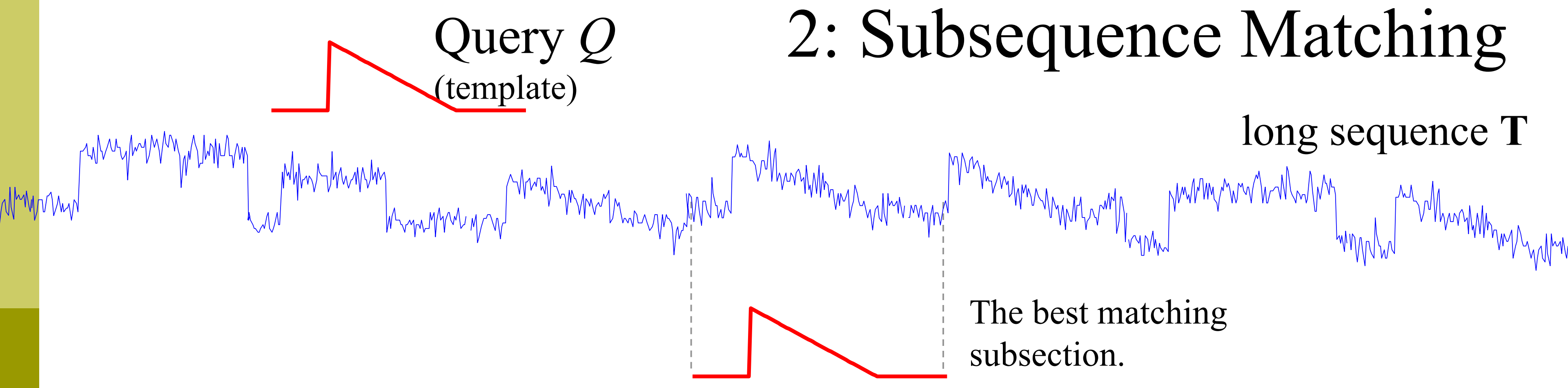
[Figure taken from http://itu-algorithms.github.io/events/2015/05/20/Zezulacourse]

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

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

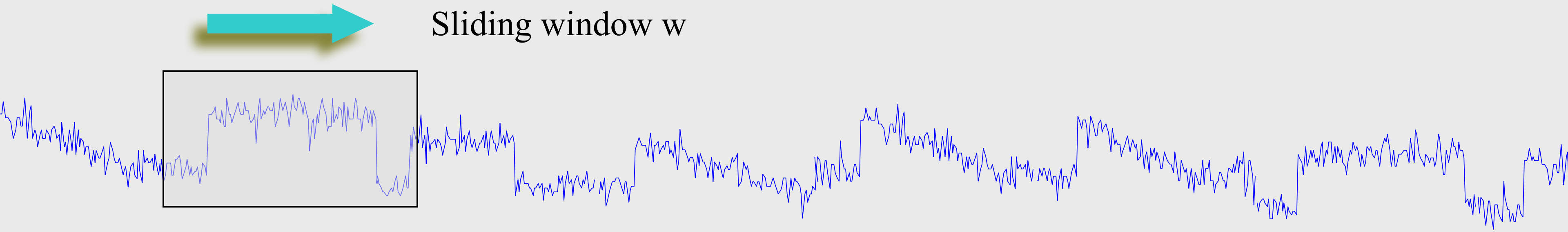
- Problem:
  - Given one or possibly more long real-valued sequences (i.e., time-series), develop an index for subsequence matching queries:
  - Subsequence Matching: given a database of long sequences S an a query sequence q, find all subsequences s in S, such that $D(s,q) \leq \varepsilon$, where ε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

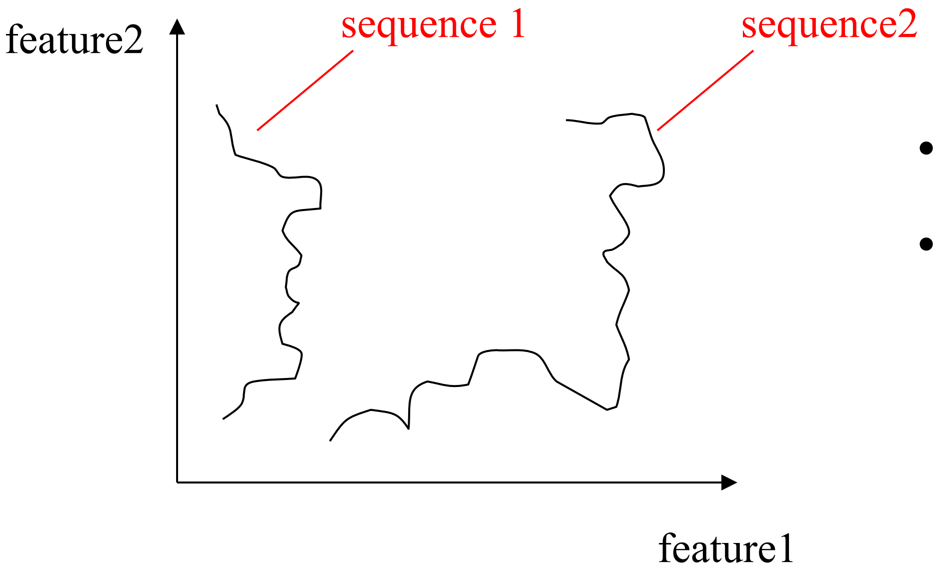
- Each position of the window defines a subsequence of length w
- Each subsequence can be transformed to a point in a low dimensional space
    - E.g., using DFT, WDT, etc.
- The Euclidean low-dimensional distance lower-bounds D.



Sliding window w

Subsequence Matching

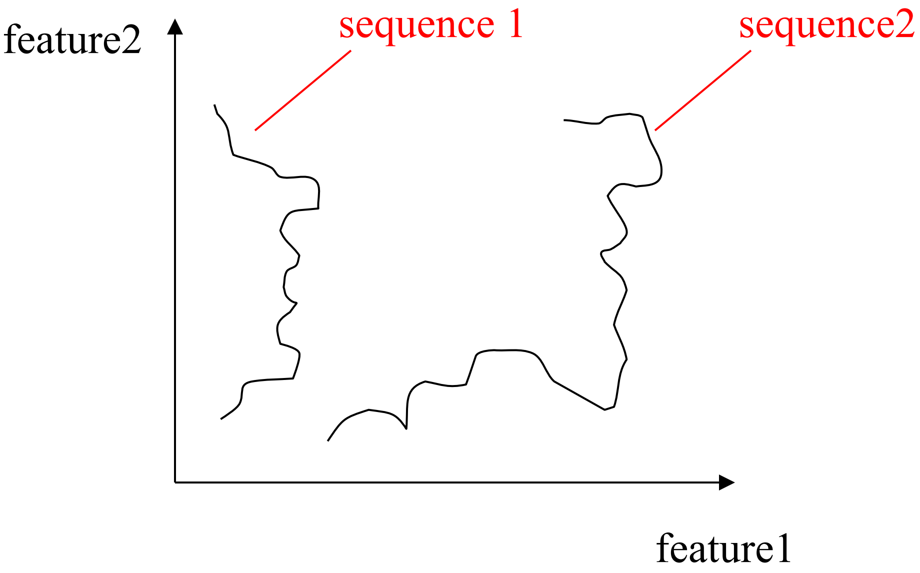
- The low-dimensional vectors of consecutive subsequences define a trail in the low-dimensional space



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

Subsequence Matching

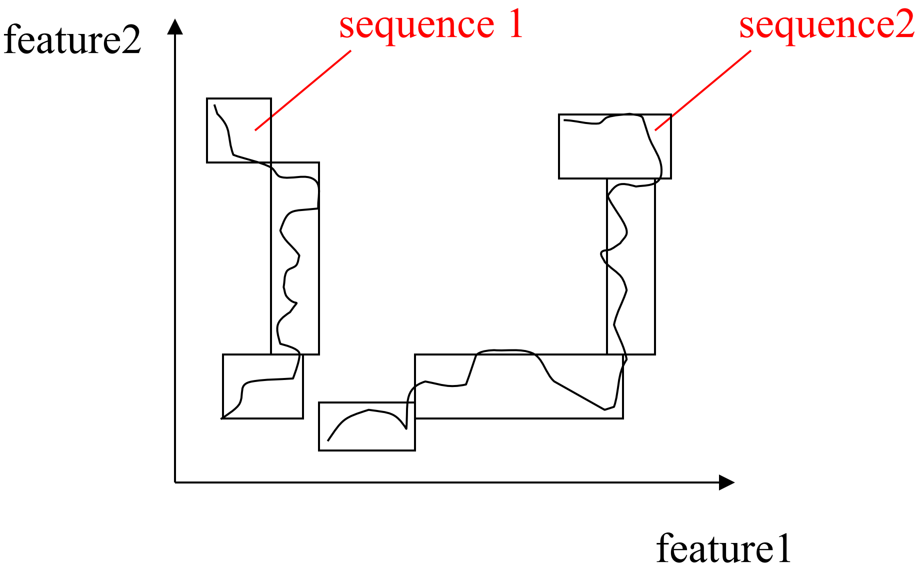
- However, the number of points can be too large, resulting in a large (and slow index)



- 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

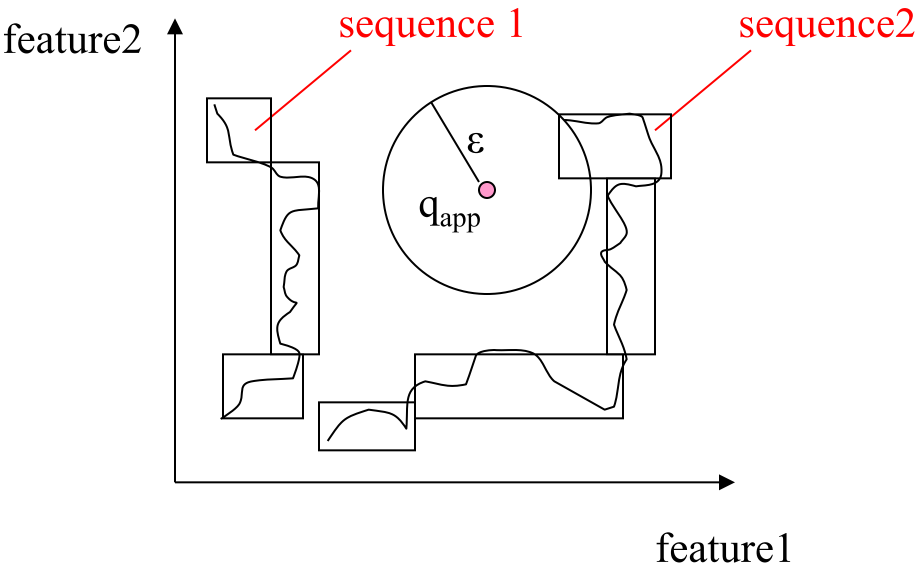
  a) the id of the sequence

  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)

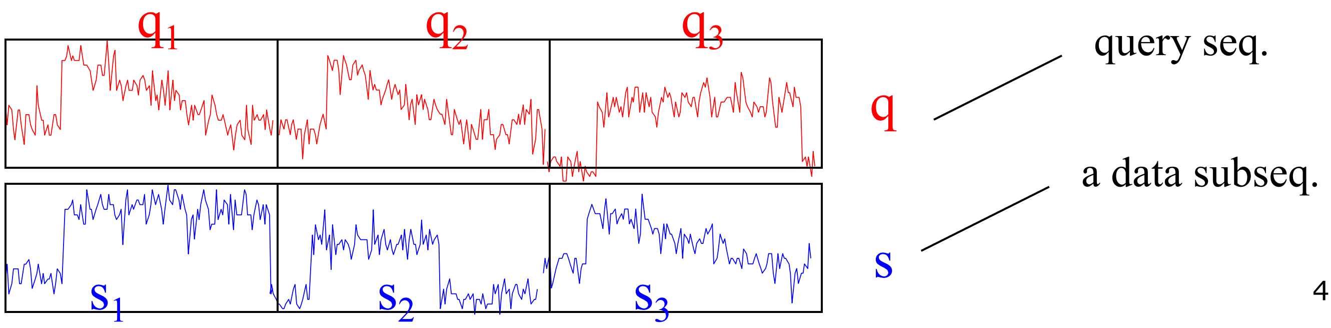
Subsequence Matching

- 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 $\varepsilon=4$ ($\varepsilon^2=16$)
- 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

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

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

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

$\varepsilon^2/2=8$

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

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

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

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

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

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

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

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

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

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

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

$\varepsilon^2/2=8$

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 $\varepsilon=4$ distance away

Subsequence Matching: Example

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

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

$\varepsilon^2=16$

$P = \{1,2,3\}$

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

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

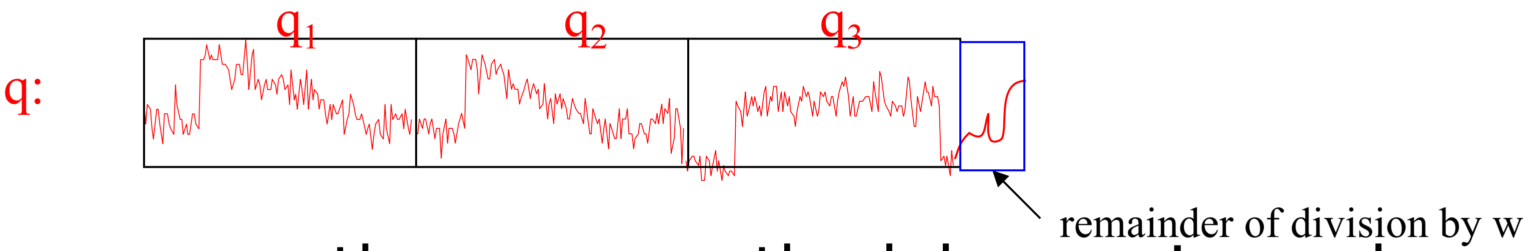
$d^2(q,1)=0+0+0+0+0+2^2=4 \leq \varepsilon^2$ ⇨ result!

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

Acknowledgement

- Some slides are taken/modified from

A Tutorial on Indexing and Mining Time Series Data

http://www.cs.ucr.edu/%7Eamonn/tutorial_on_time_series.ppt

  - **Dr Eamonn Keogh**
  - Computer Science & Engineering Department
    University of California - Riverside
    Riverside,CA 92521