course:

Searching the Web (NDBI038)

Searching the Web and Multimedia Databases (BI-VWM)

lecture 9:

Indexing metric similarity for efficient multimedia retrieval

prof. RNDr. Tomáš Skopal, Ph.D.
Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague
Department of Software Engineering, Faculty of Information Technology, Czech Technical University in Prague

Today's lecture outline

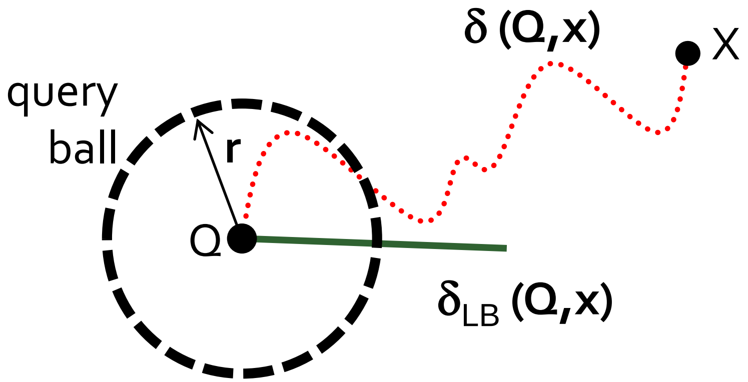
- metric access methods
  - motivation
  - pivot tables
    - AESA, LAESA
  - hierarchical structures
    - GNAT, M-tree
  - hashed indexes
    - D-index
  - hybrid structures
    - PM-tree, M-index
- performance measures
  - distance computations, I/O cost, internal cost, realtime cost

Motivation

- similarity search
  - effective (quality of similarity measure)
  - **efficient = fast**
- lower-bounding
  - general mechanism for efficient similarity search
  - using cheap lower bound instead of expensive distance
- metric space model
  - lower-bounding based on pivots and metric postulates
  - allows partitioning of the space
  - database indexing based on metric space model
    - metric access methods

General lower-bounding

- a cheap determination of **lower-bound distance** $\delta_{LB}(*,*) \leq \delta(*,*)$
- provides a mechanism how to quickly filter irrelevant objects from search
  - consider query ball **(Q, r)** and data object **x**
  - if $\delta_{LB}(Q,x) > r$, **x** is irrelevent
- **tight** lower bound needed
  - increasing probability that $\delta_{LB}(*,*) > r$
  - e.g., near-zero LB is useless



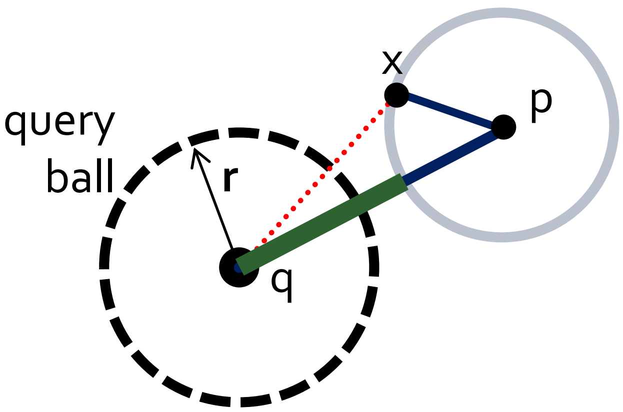
Metric postulates

- metric postulates support common assumptions on similarity
  - reflexivity – object is maximally similar to itself
  - non-negativity – two distinct objects are somehow dissimilar
  - symmetry – the direction of similarity judgement is not important
  - triangle inequality – similarity is transitive

$$
\begin{array}{rcllc}
\delta(x,y) & = & 0 & \Leftrightarrow x = y & \text{reflexivity} \\
\delta(x,y) & > & 0 & \Leftrightarrow x \neq y & \text{non-negativity} \\
\delta(x,y) & = & \delta(y,x) & & \text{symmetry} \\
\delta(x,y) + \delta(y,z) & \geq & \delta(x,z) & & \text{triangle inequality}
\end{array}
$$

Metric lower-bounding

- metric space model provides a specific means of lower-bounding
  - **pivot objects** – static objects (selected from the database)
  - triangle inequality for lower bound construction (using a pivot)
  - precomputed distances from objects to pivots
    - stored in a metric index



<u>The task:</u> check if **x** is inside query ball

•we know **δ(q,p)**

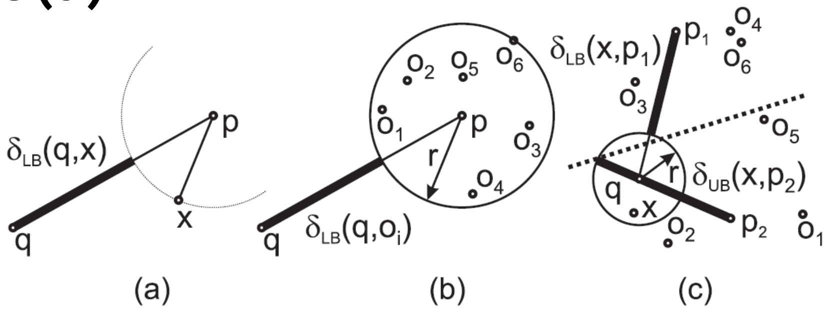
•we know **δ(x,p)**

•we do not know **δ(q,x)**

•we do not have to compute **δ(q,x)**, because its lower bound |δ(q,p)-δ (p,x)| is larger than **r**, so **x** surely cannot be in the query ball, so **x** is ignored

Metric lower-bounding

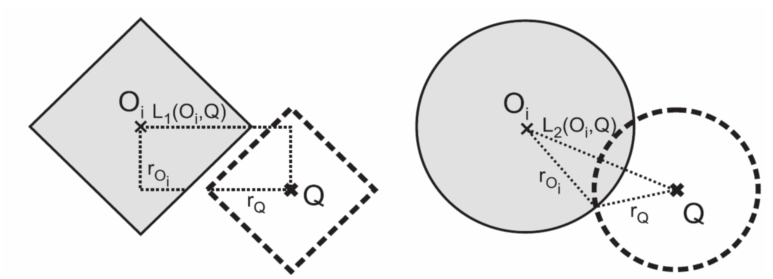
- lower bound to a database object **(a)**
  - basic concept
- lower bound to a region **(b)**
  - ball-shaped regions
  - logical combinations of predicates
- combination of lower and upper bounds **(c)**
  - hyper-plane partitioning



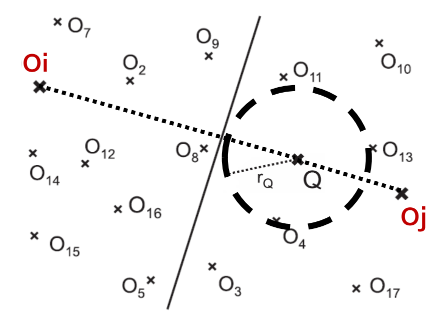
- usage: filtering whole regions (or directly the objects)
  - the search is then performed just in several partitions (subset of objects), while computing as few distance computations of **δ(q,x)** as possible → **efficient search**

Filtering metric regions

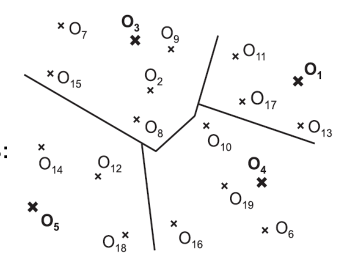
- ball-shaped regions
  - for two balls (O_i, r_{Oi}) and (q, r_q) it holds:
    if $\delta(O_i, q) > r_{Oi} + r_q$, then
    the balls do not overlap (and vice versa)



- hyperplane-separated regions
  - for two regions determined by a hyperplane between O_i ,O_j, it holds:
    if $\delta(O_i, q) - r_q > \delta(O_j, q) + r_q$, then
    first (Oi) region does not overlap query
    if $\delta(O_j, q) - r_q > \delta(O_i, q) + r_q$, then
    second (Oj) region does not overlap query
    - simply: if lower-bound distance to query from one pivot is greater than upper-bound distance to query from the second pivot, the first region does not overlap

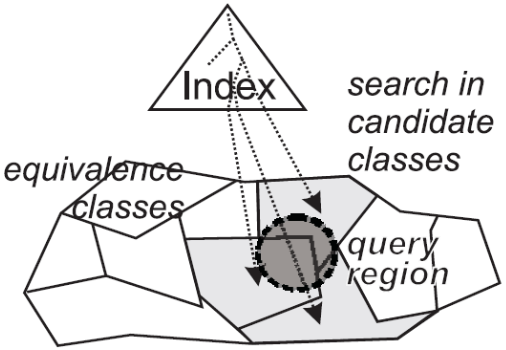


generalization
to multiple pivots:



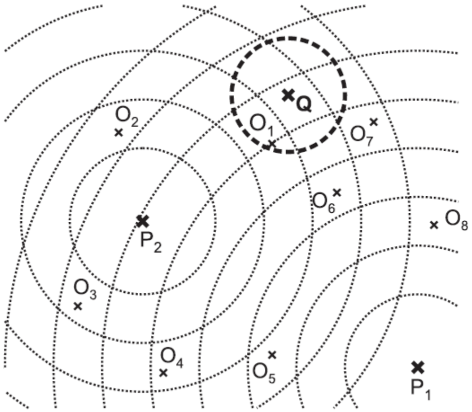
Metric access methods – motivation

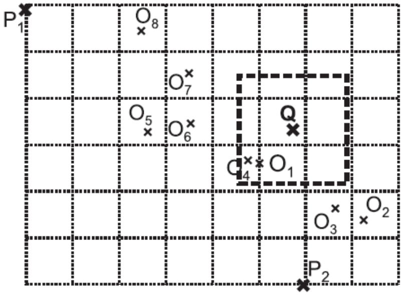
- metric access methods (MAM), or metric indexes
  - database methods for **fast similarity search in database S**
  - various types of cost measuring the "fast"
    (realtime, number of distance comp., I/O, internal, etc.)
- the means: **metric indexing**
  - based on lower-bounding using pivots
  - general metric space assummed
    (i.e., only distances could be used for indexing, not the actual object content)
  - metric postulates needed (the similarity function must be metric distance)
- various structural MAM designs
  - flat indexes (pivot tables)
  - hierarchical indexes (trees)
  - hashed indexes
  - hybrid indexes
  - index-free methods



Pivot tables

- class of indexes using mapping of data to the pivot space
  - each data object **x** maps to a vector **v**
  - i.e., we get distance matrix
- originally AESA/LAESA
  (Linear) Approximating and Eliminating Search Algorithm
- AESA
  - every data object is regarded as a pivot, i.e., we have square matrix of size **O(|S|²)**
  - empirical average **O(1)** time for nearest neighbor search, expensive construction
- LAESA
  - only some **k** data objects selected as pivots, i.e., matrix size **O(|S|)**
- contractive mapping
  - L_∞ distance in pivot space is lower bound to the original distance δ

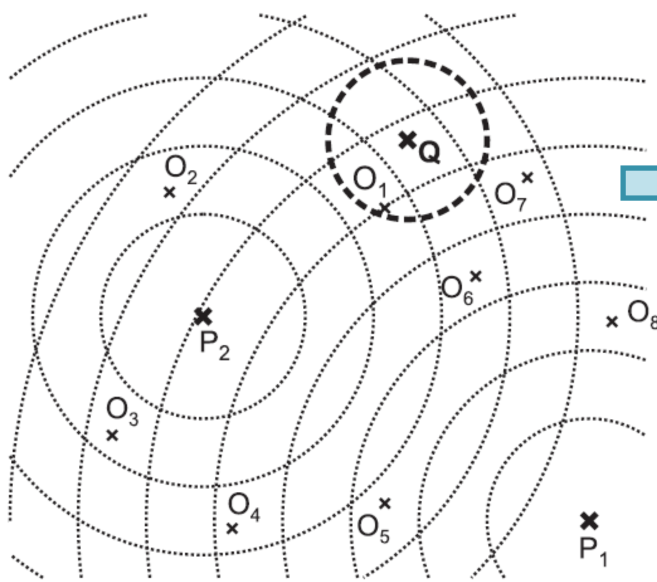


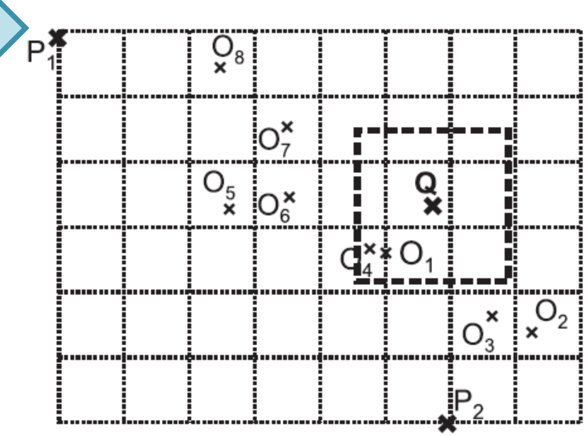


LAESA range query

	P_1	P_2
O_1	5	2.6
O_2	7.2	1.4
O_3	6.7	1.6
O_4	4.8	2.7
O_5	2.6	3.2
O_6	3.6	3.5
O_7	3.6	4.5
O_8	2.5	5.5

- simple 2 phases
  - sequential processing of the distance matrix, filtering
  - the non-filtered candidates must be refined in the original space





LAESA nearest neighbor query

- one-phase (as in AESA, see next slide)
  - suitable for large number of pivots
- two-phase
  1. the entire query vector $\mathbf{v}_q$ is determined from the query object
  2. the database objects $\mathbf{o}_i \in \mathbf{S}$ are sorted asc. w.r.t $\mathbf{L}_\infty(\mathbf{v}_q, \mathbf{v}_{oi})$
  3. in this order the $\delta(\mathbf{q}, \mathbf{o}_i)$ is being computed, updating the NN candidate $\mathbf{o}_{nn}$
     - if $\delta(\mathbf{q}, \mathbf{o}_{nn}) < \mathbf{L}_\infty(\mathbf{v}_q, \mathbf{v}_{oi})$, the filtering terminates (there is no better candidate than $\mathbf{o}_{nn}$)

AESA nearest neighbor query

- **problem:**
  cannot compute the entire query vector because all database objects are pivots (i.e., would result in naive sequential search)

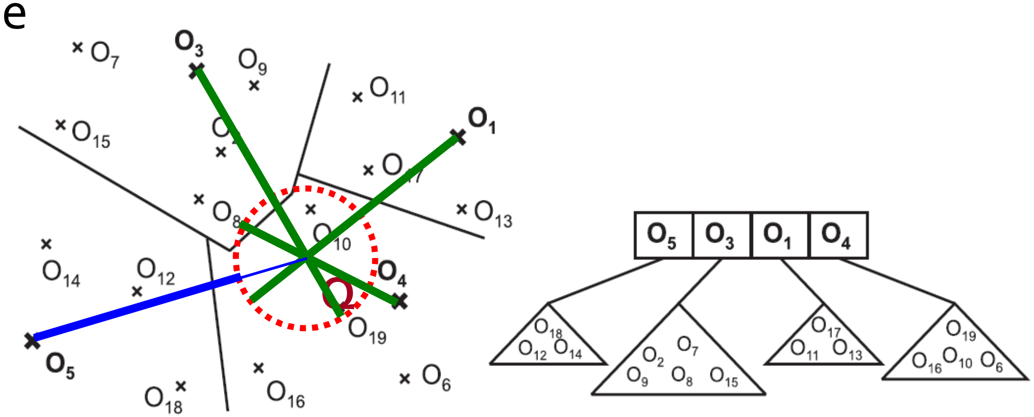
- **the idea:**
  - multiple passes of the matrix, each considering one more column (pivot), starting with one column (random)
  - after each pass
    - the lower bounds are increasing, because we get more pivots
    - because of that also more objects is filtered from search

Hierarchical metric access methods

- partitioning on the metric space, creating a hierarchy of metric regions
  - direct partitioning – close objects are in the same partition
    - either balls (e.g., **M-tree**) or hyperplane-separated subspaces (e.g., **GNAT**)
  - recursive – partitions are decomposed the same way as the whole database
  - using **local** pivots, instead of **global** pivots
    - local pivot is context-dependent, e.g., selected from objects in a subtree
    - local pivots are not fixed for the index lifetime,
they could be replaced as the database is being updated
- natural approach (inspired by R-tree, kd-tree, etc.), but
  - problem of huge volume of regions
  - problem of overlaps between sibling regions
  - problem of the shape of regions

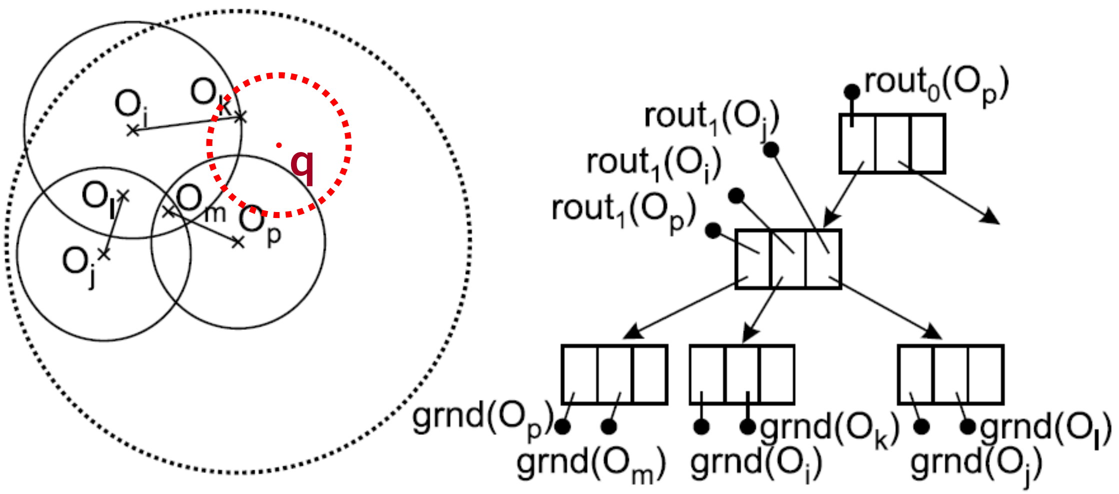
GNAT

- base on hyperplane partitioning of the space
- generalization of gh-tree into n-ary tree
  - multiple pivots in each node
  - some other extensions
    - distance ranges
- range query processing
  - test overlap of all partitions
  - recursively proceed down the hierarchy in the overlapping partitions
  - filtering of partition/subtree belonging to $\mathbf{O_5}$ (in the example):
    - if the **closest possible object** inside the query (w.r.t $\mathbf{O_5}$) is more distant than the **furthest possible object** inside the query (w.r.t. any of $\mathbf{O_1}$, $\mathbf{O_3}$, $\mathbf{O_4}$), then the subtree can be filtered out



M-tree

- inspired by R-tree – modification into metric spaces
  - hierarchy of nested balls; balanced, paged, dynamic index
  - suitable for secondary storage
  - compact hierarchy is necessary, ball overlaps lead to inefficient query processing
  - many descendants – e.g., M*-tree, PM-tree, Slim-tree, etc.
- query processing
  - traversing the hierarchy, accessing just the overlapping nodes/subtrees



Metric access methods based on hashing

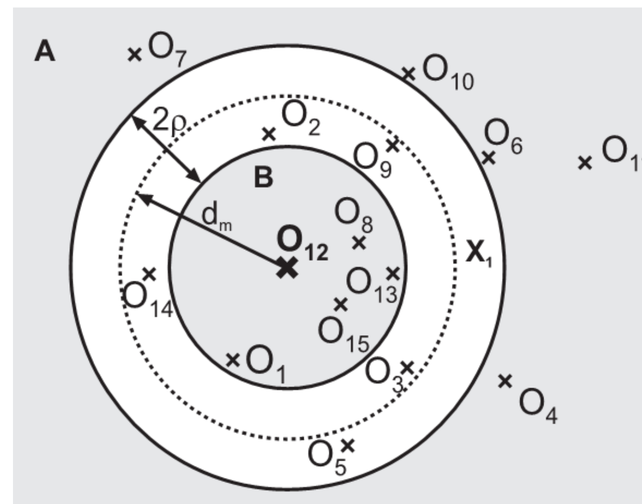
- similarity hashing function is crucial
  - each object is hashed into a bucket
  - locality-sensitive hashing function – close object should be hashed to the same bucket
  - could be only achieved based on distance to a (set of) pivot(s)
    - remember, the content of the object is not accessible to MAMs
- having the hashing function, the index design can use different structures
  - plain hash tables
  - hierarchical hash tables (e.g., **D-index**)
  - etc.

D-index

- D-index = hash-based index

- based on hashing functions $bps^{1,\rho,j}$ (ball-partitioning split functions), where $\mathbf{p_j}$ is pivot, $\mathbf{d_m}$ is median distance from $\mathbf{p_j}$ to the objects $\mathbf{o_i}$, and $\boldsymbol{\rho}$ is a splitting parameter

- the function assigns to an object o_i:
  - 2 , if it is inside the ring (p_j, d_m-ρ, d_m+ρ)
  - 1, if it is outside the greater ball (p_j, d_m+ρ)
  - 0, if it is inside the smaller ball (p_j, d_m-ρ)

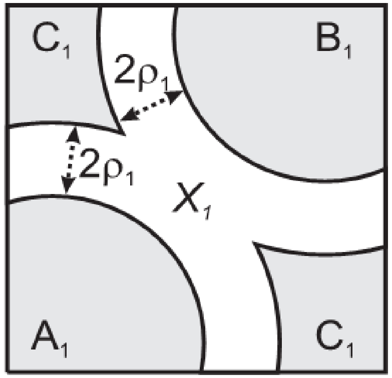


$$bps^{1,\rho,j}(O_i) = \begin{cases} 0 & \text{if } d(O_i, P_j) \le d_m - \rho \\ 1 & \text{if } d(O_i, P_j) > d_m + \rho \\ 2 & \text{otherwise} \end{cases}$$

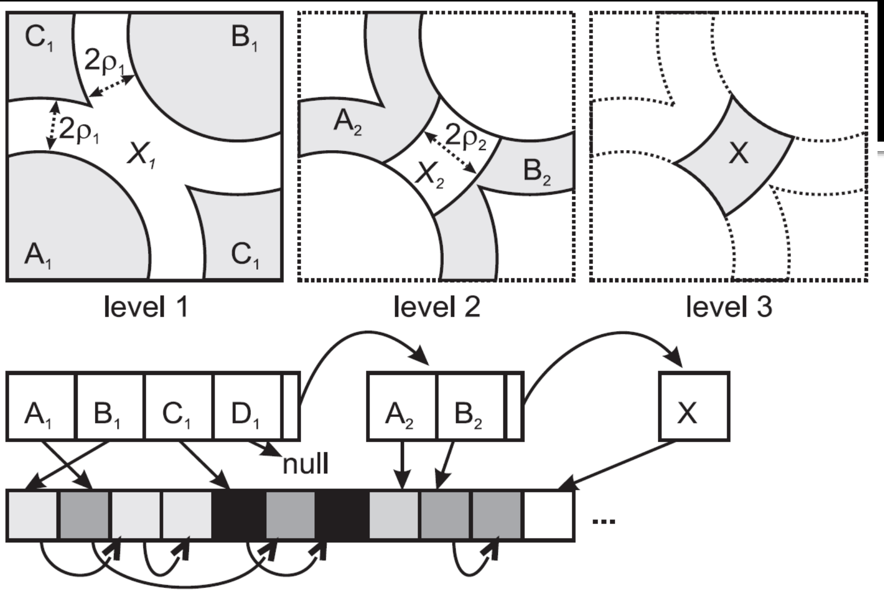
D-index

- the $bps^{1,\rho,j}$ functions can be combined
- if there **is no** 2 in the hash of the combined function, the string 0, 1 is a binary code of a partition (up to 2^n)
- if there **is a 2** in the hash code, the hashed object belongs to so-called exclusion set

- the motivation is to keep the partitions separated by the region of exclusion set
- the exclusion set is hashed again (based on different $bps^{1,\rho,j}$ functions), recursively, until it gets sufficiently small



D-index



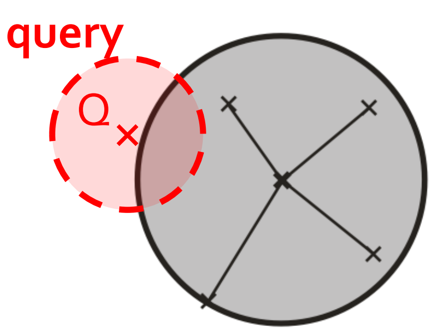
- D-index structure
  - hashed regions are organized in buckets on the disk
  - the rehashing of exclusion set defines another level of D-index
- advantages
  - for small query radius (**r < ρ**), at most one bucket is accessed at each level
  - if the whole query lies inside a region, only that bucket is accessed
  - if the whole query lies inside the exclusion set, no bucket at that level is accessed and the query processing proceeds at the next level
- disadvantages
  - selection of the parameters ρ, d_m is difficult and/or D-index is not balanced
  - the ρ parameter should be very small, in order to not become everything into the exclusion set (i.e., suitable only for small and point queries)

Hybrid metric access methods

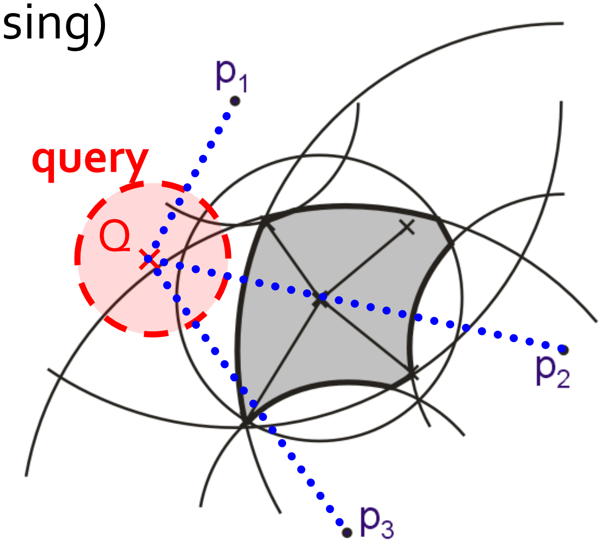
- combinations of different principles
  - pivot tables with hierarchical indexes
    - e.g., PM-tree
  - even more complex, combining pivot tables,
    hierarchical ball-shaped + hyperplane-based indexes, and hashing
    - e.g., M-index
- synergistic effect
  - different principles are most efficient under different conditions,
    so let's employ them all

PM-tree

- PM-tree = M-tree additionally using a set of ***p*** global pivots
- each ball region is further reduced by ***p*** rings, centered in the pivots
- smaller volume of regions
  - better filtering (faster query processing)

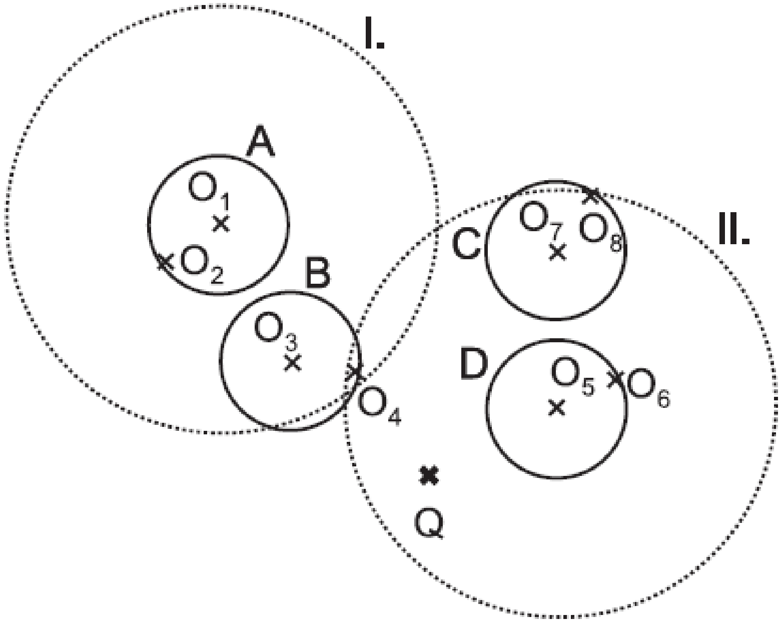


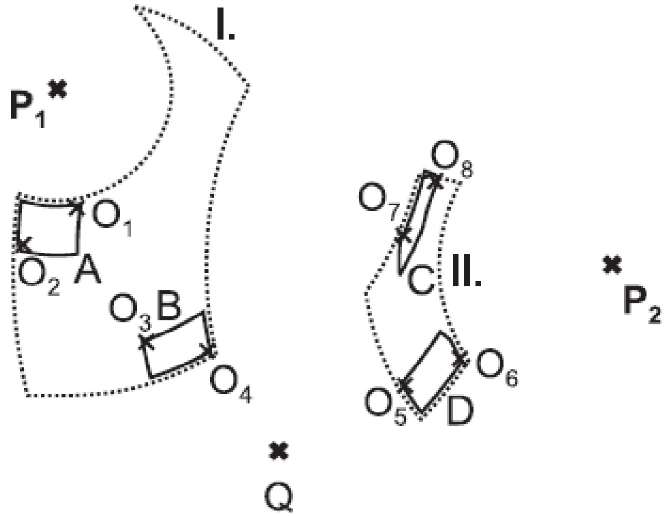
M-tree region



PM-tree region

M-tree vs. PM-tree hierarchy



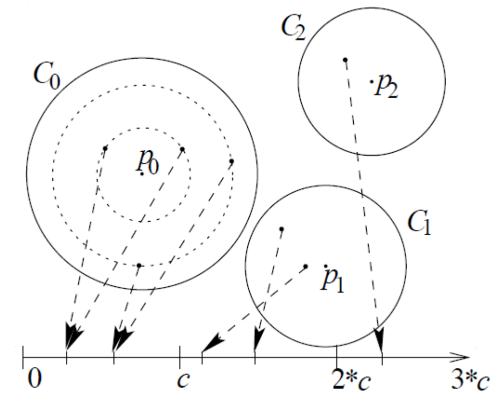


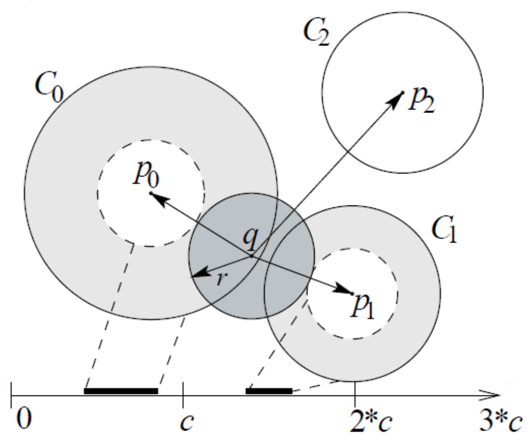
M-index

- combines all of the principles used in MAMs so far
  - pivot tables (for each object the distances to global pivots are stored)
  - hyperplane partitioning
  - ball partitioning
  - hashing (even preserving order)
- the index itself could be any database structure maintaining linear ordering of keys
  - e.g., B+-tree

M-index

- inspired by iDistance (mapping scheme into 1D)
  - select **n** pivots $\mathbf{p_i}$ as centers of clusters $\mathbf{C_i}$
  - let's divide the real axis among **n** intervals <i*c, (i+1)*c>, where **i** is the Id of cluster $\mathbf{C_i}$
  - each object **o** is mapped (hashed) to the Id **i** of the nearest pivot
  - the distance is added to the integer key as the fractional part $iDist(o) = d(p_i, o) + i \cdot c$
    - suppose c=1 and distances normalized to 0..1
  - a range query is then transformed into a set of intervals that have to be searched





(images from Novak and Batko, Metric Index: An Efficient and Scalable Solution for, Similarity Search, SISAP 2009)

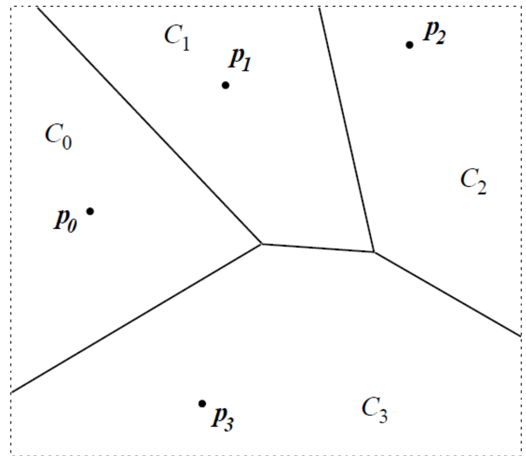
M-index

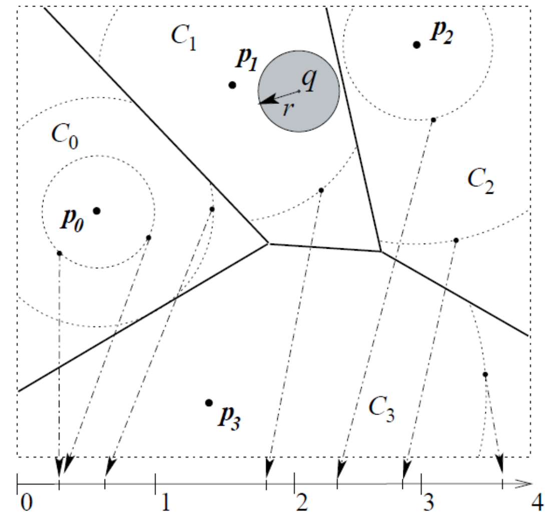
(images from Novak and Batko, Metric Index: An Efficient and Scalable Solution for, Similarity Search, SISAP 2009)

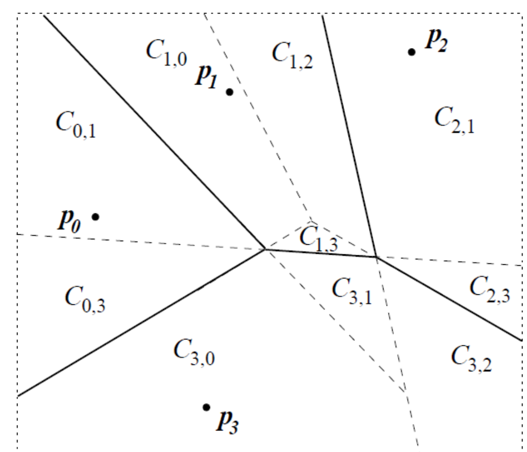
- because the data objects are partitioned among the nearest pivots, the clusters can be filtered as in GNAT

- as in GNAT, recursive partitioning is considered in M-index (so-called **multi-level M-index**), leading to recursive splitting of the intervals on the real axis. The final key of an object **o** is then:

$$key_l(o) = d(p_{(0)_o}, o) + \sum_{i=0}^{l-1} (i)_o \cdot n^{(l-1-i)}$$

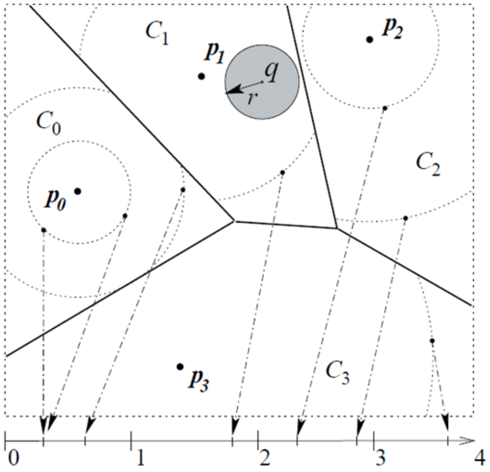






M-index

- in addition to the mapping to real axis, the M-index stores information for additional filtering
  - for each object a vector of distances to the pivots is stored, allowing to consider the non-filtered candidates as a pivot table
  - for each cluster the minimal and maximal radius is stored, to allow ball filtering (as used in M-tree)



Performance measures

- performance of MAM in terms of
  - indexing
  - querying
- cost measures
  - logic cost
    - number of distance computations
    - I/O cost (disk access)
    - internal cost (task-specific algorithms, internal data structures)
    - other (networking, synchronization of parallel processes, etc.)
  - physical overall cost
    - realtime (wall clock time, the response time)

Performance measures

- optimization of the total (index+query) cost
  - sequential scan as reference method
    - no indexing and linear cost queries
  - in database scenario =
    index cost is amortized by reduced query cost
    - frequent queries and not frequent updates
    - outperforms sequential scan in long term
- cost measure selection
  - logic cost measures used in specific evaluation should be dominant components in the realtime
    - other logic cost measures ignored
    - wall clock realtime ignored (affected by external factors)

Distance computations (DC)

- DC = number of distance **δ(*,*)** computations
- DC dominating component in the realtime when
  - expensive distance function is used
    - ≥ $O(n^2)$ and/or large object size (n)
  - rather small database is used
    - fits in main memory

I/O cost (disk accesses)

- classic database-oriented cost (also transactions/per second)
- example comparison HDD vs. SSD for 100 MB index, 4kB disk page, i.e., 25,600 pages
  - HDD (magnetic drive technology)
    - 1990
      - seek time + latency = **65ms**, transfer **600 kB/s** (Seagate ST 225 20MB)
      - sequential scan, **100% pages**, contiguous access = **167 sec**
      - a hierarchical MAM, **1% pages**, random access = **16.6 sec seek + 1.6 sec read = 18.2 sec** (**~9.2x speedup**)
    - 2020
      - seek time + latency = **6ms**, transfer **100 MB/s**
      - sequential scan, **100% pages**, contiguous access = **1 sec**
      - a hierarchical MAM, **1% pages**, random access = **1.5 sec seek + 0.01 sec read = 1.51 sec** (**~0.66x slowdown**)
  - SSD (solid-state drive)
    - sequential read **2150 MB/s**, random read **300k IO/s** (4kB IO, M.2 NAND NVMe), 2020
    - sequential scan, **100% pages**, contiguous access = **45 ms**
    - a hierarchical MAM, **1% pages**, random access = **0.85 ms** (**~52x speedup** vs. 100x theoretical best)

  - hierarchical indexing (random access) makes sense again in SSD (or RAM)!
    - optimized sequential scan could be a surprise in HDD

Internal cost

- more sophisticated MAM $\rightarrow$ more internal overhead
  - various auxiliary main-memory structures + processing
  - overhead data in the index + processing
- examples
  - incremental kNN processing (Hjaltason and Samet)
    - optimal in DC (w.r.t. equivalent range query), **but**
    - huge time/space overhead when managing the heap of requests
  - pivot tables (basic LAESA)
    - scanning the distance matrix
    - consider, e.g., 128 dimensional vector dataset + any L_p distance, 128 pivots $\rightarrow$ distance matrix processing means the **same or worse than simple sequential query (!)**

Realtime cost

- realtime cost (wall-clock time)
  - **cons:**
    - **optimization- and platform-dependent**
    - **harder to set up fair comparison**
  - **pros:**
    - **the only objective measure when it comes to real-world application!**
- real-world example
  - database of up to 5.6 million peptide spectra (pieces of proteins), dim ≈ 32 (intrinsic dim. ≈ 3)
  - **O(n)** variant of Hausdorff distance