course:

Searching the Web (NDBI038)

Searching the Web and Multimedia Databases (BI-VWM)

lecture 10:

Approximate similarity search

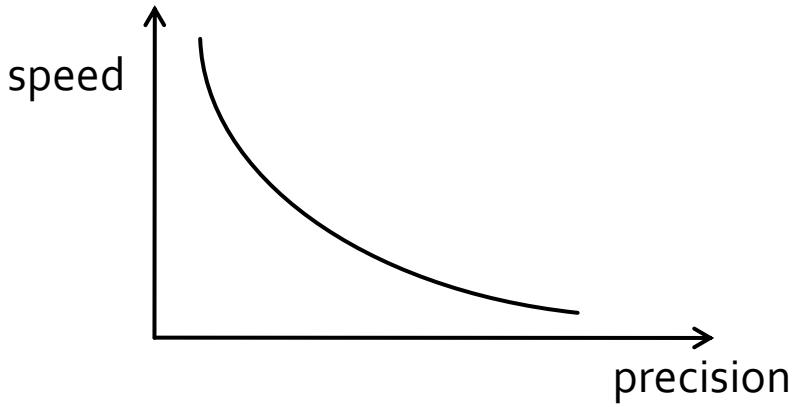
prof. RNDr. Tomáš Skopal, Ph.D.
Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague
Department of Software Engineering, Faculty of Information Technology, Czech Technical University in Prague

Today's lecture outline

- motivation
  - indexability
- mapping techniques
  - pivot space (simple usage of pivot tables)
  - FastMap
- approximate kNN search using M-tree
  - exact kNN algorithm
  - approximate kNN variants
- probabilistic queries using pivot tables
- permutation indexes

Motivation – exact search

- "exact" similarity search could be too slow
  - even when indexing technique is employed
- we might want to trade the retrieval precision for the speed
  - huge gain in speed usually expected (orders of magnitude) while only small loss in retrieval precision (several percent)

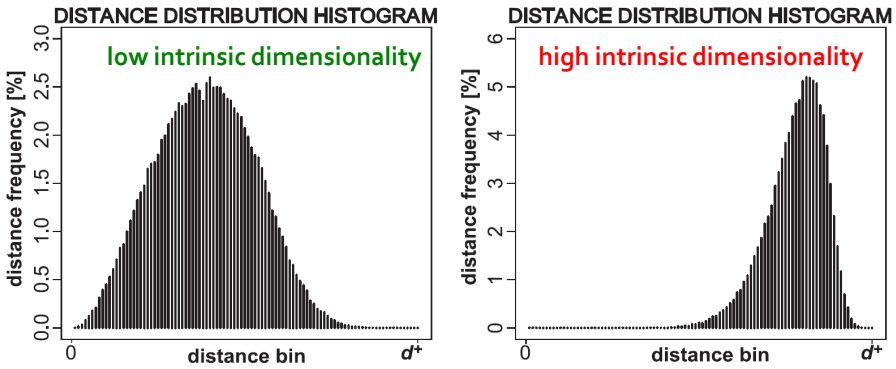


Motivation – exact search

- what is "too slow" exact search?
  - metric postulates alone do not ensure efficient indexing/search
  - also data distribution in space is very important
  - indexability = ability of structuring the database into distinct classes, such as clusters
    - **intrinsic dimensionality** – an indexability indicator

Motivation – exact search

- **intrinsic dimensionality** $\rho(S,d) = \mu^2 / 2\sigma^2$
  (μ is **mean** and σ^2 is **variance** of distance distribution in S)
- low ρ (e.g., below 10) means the dataset is **well-structured** – i.e. there exist tight clusters of objects
- high ρ means the dataset is **poorly** structured – i.e. objects are almost equally distant
  - intrinsically high-dimensional datasets are hard to organize
  - **exact search using any MAM becomes inefficient!** (the same as sequential scan)



Motivation – imperfect similarity

- moreover, similarity function often does not mimic perfectly the human perceived similarity (especially in multimedia databases)
  - even sequential search is thus subjectively approximate!
- hence, approximate search techniques only increase the subjective error of retrieval
  - i.e., just quantitative change, not a "change of category"

Motivation – approximate search

- what is "approximate"?
  - guarantee of a bound (strongest)
    - every object in query result is relevant to some extent
  - probabilistic estimates
    - every object in query result is relevant with some probability
    - i.e., on average, some percentage of objects is relevant, the rest could be completely wrong
  - no guarantee (weakest)
    - just experimental observation based on sampling, etc.

Motivation – approximate search

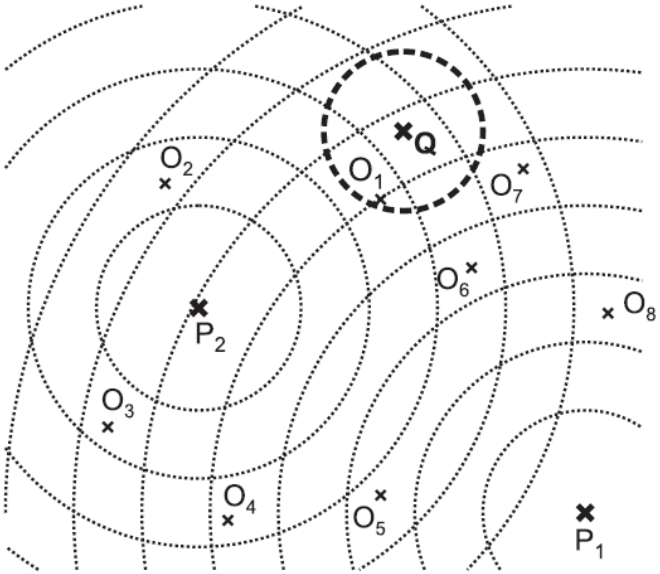
- how to speedup search?
  - realtime cost is crucial
    - indexing to reduce the distance computation (more aggressive than exact search)
    - transformation of the metric space into a simpler but only approximate target (vector) space
      - sequential search in the target space or a kind of indexing again

Mapping techniques

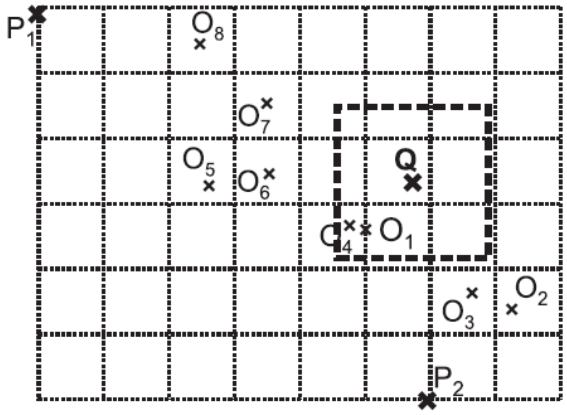
- expensive distance function δ in an arbitrary metric space could be replaced by cheap L_p distance
  - the metric space is embedded into a suitable vector space
- pre-processing of the database
  - mapping the database
    - mostly static (the embedding scheme is data-dependent)
    - some dynamic (the scheme is data-independent, e.g., simple **pivot tables**)
  - indexing the target space by metric access methods (or spatial access methods)
- many mapping techniques for metric spaces developed
  - mostly mapping to Euclidean vector space
  - metric multidimensional scaling (best but slow)
  - faster methods: **FastMap**, MetricMap, SparseMap

Mapping techniques – pivot tables

- the pivot tables (LAESA) could be used as a simple approximate method based on mapping (omitting the refinement step)
  - coordinate axis = distances to a pivot
  - using $L_\infty()$ as an approximation of $\delta()$



2NN query:
result = {O_1, O_4}



	P_1	P_2	$L_\infty(q,O_i)$
O_1	5	2.6	0.7
O_2	7.2	1.4	2.1
O_3	6.7	1.6	1.8
O_4	4.8	2.7	1
O_5	2.6	3.2	3.1
O_6	3.6	3.5	2.2
O_7	3.6	4.5	2.2
O_8	2.5	5.5	3.2

Mapping techniques – pivot tables

- will this work?
  - L_∞ in the pivot space lower-bounds δ in the metric space
  - the more (good) pivots, the tighter lower bound
    - and thus better approximation
  - limiting the need of distance $\delta(*,*)$ computations to
    - mapping the query object into the pivot space
    - optionally, re-ranking
      - e.g., 100NN query in the pivot space + re-ranking to 10NN result in the metric space
  - reasonable number of good pivots needed
  - no guarantee of retrieval precision (just empirical)

Mapping techniques – FastMap

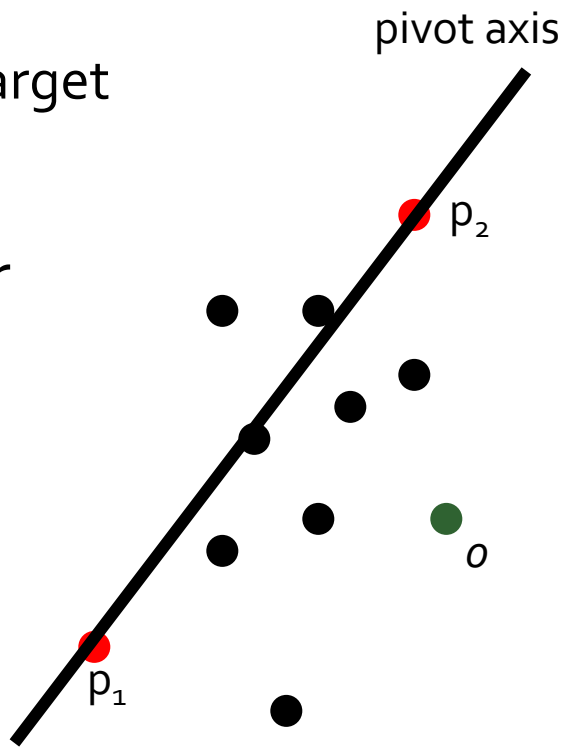
- considering an arbitrary metric space
  as an "unknown" Euclidean space
  (of user-defined dimensionality)

- the (assumed) coordinates are "recovered" by use of
  - establishing artificial coordinate system using pairs of pivots (each pair means one **pivot axis**)
  - **cosine law** and **Pythagorean theorem**,
    allowing to **project** the database objects onto the "pivot axis" (taking into account the orthogonal hyperplane formed by the already established pivot axes)

Mapping techniques – FastMap

the FastMap algorithm:

- step 0:
  - let *k* be the number of dimensions of the target Euclidean space we construct
- step 1: select first (next) best pivot pair
  - select object *o* at random
  - find p_1 being the furthest from *o*
  - find p_2 being the furthest from p_1
  - we have virtual "axis" (p_1, p_2), i.e., first (next) dimension was established



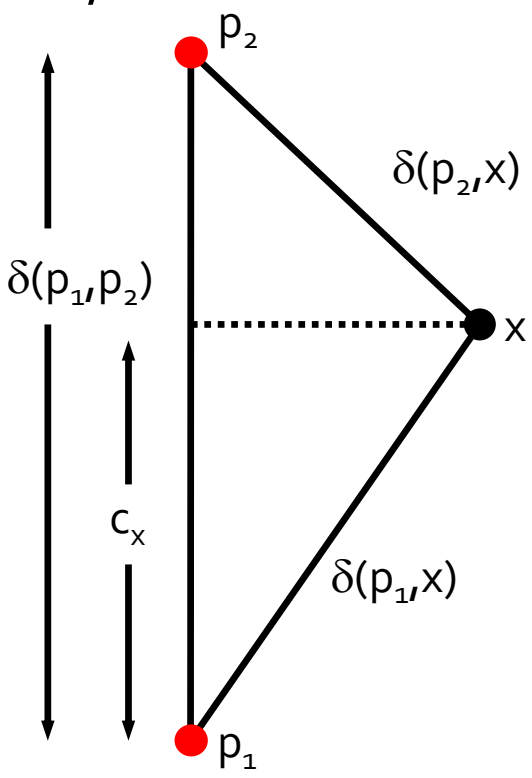
Mapping techniques – FastMap

- step 2:
  - given pivots (p_1, p_2), for any database point x, we use the **cosine law** to determine the relation of x along the axis p_1p_2

$$\delta(p_2, x)^2 = \delta(p_1, x)^2 + \delta(p_1, p_2) - 2c_x\delta(p_1, p_2)$$

$$c_x = \frac{\delta(p_1, x)^2 + \delta(p_1, p_2) - \delta(p_2, x)^2}{2\delta(p_1, p_2)}$$

c_x is the (next) coordinate of x in the target space, created by pseudo-projection

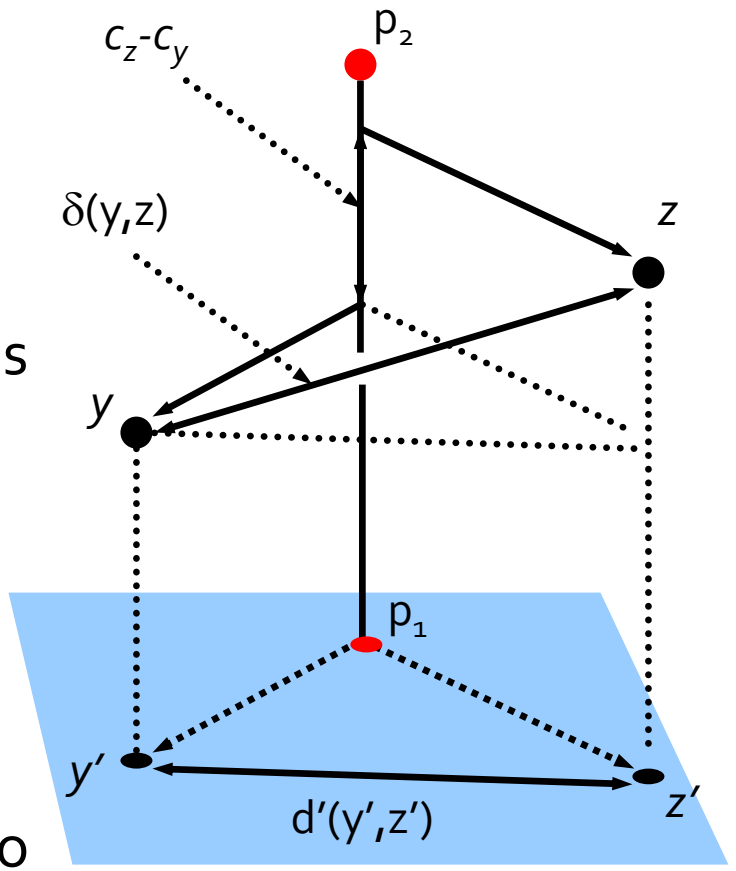


Mapping techniques – FastMap

- step 3:
  - we need to update the function $\delta(*,*)$ in order to include the already established coordinates of database objects
  - we can compute distances within the "orthogonal hyperplane" using the Pythagorean theorem

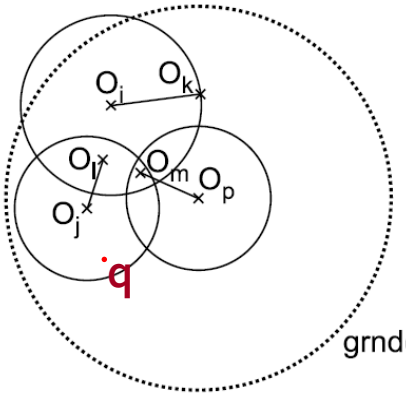
$$d'(y',z') = \sqrt{\delta(y,z)^2 - (c_z - c_y)^2}$$

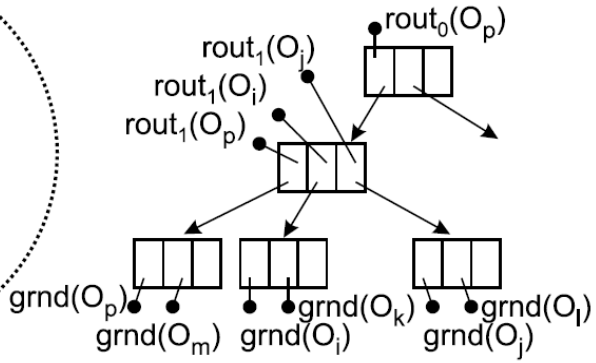
  - hence, let **$\delta(*,*)$:= d'(*,*)** and goto step 1 (or terminate if all k dimensions have been established)



M-tree

- M-tree is a metric index consisting of recursively nested ball regions (see previous lecture)
  - idea of R-tree generalized to metric spaces
- algorithm for exact kNN search developed (branch-and-bound)
  - based on a **priority queue PR** that maintains the nodes that cannot be excluded from the search
    - ordering of nodes by their (balls') lower-bound distances to q
  - kNN candidates + their distances to q are stored in a **NN array**
    - includes also candidate nodes (and upper-bound distances to q)
    - the k^{th} entry of NN means the dynamic query radius

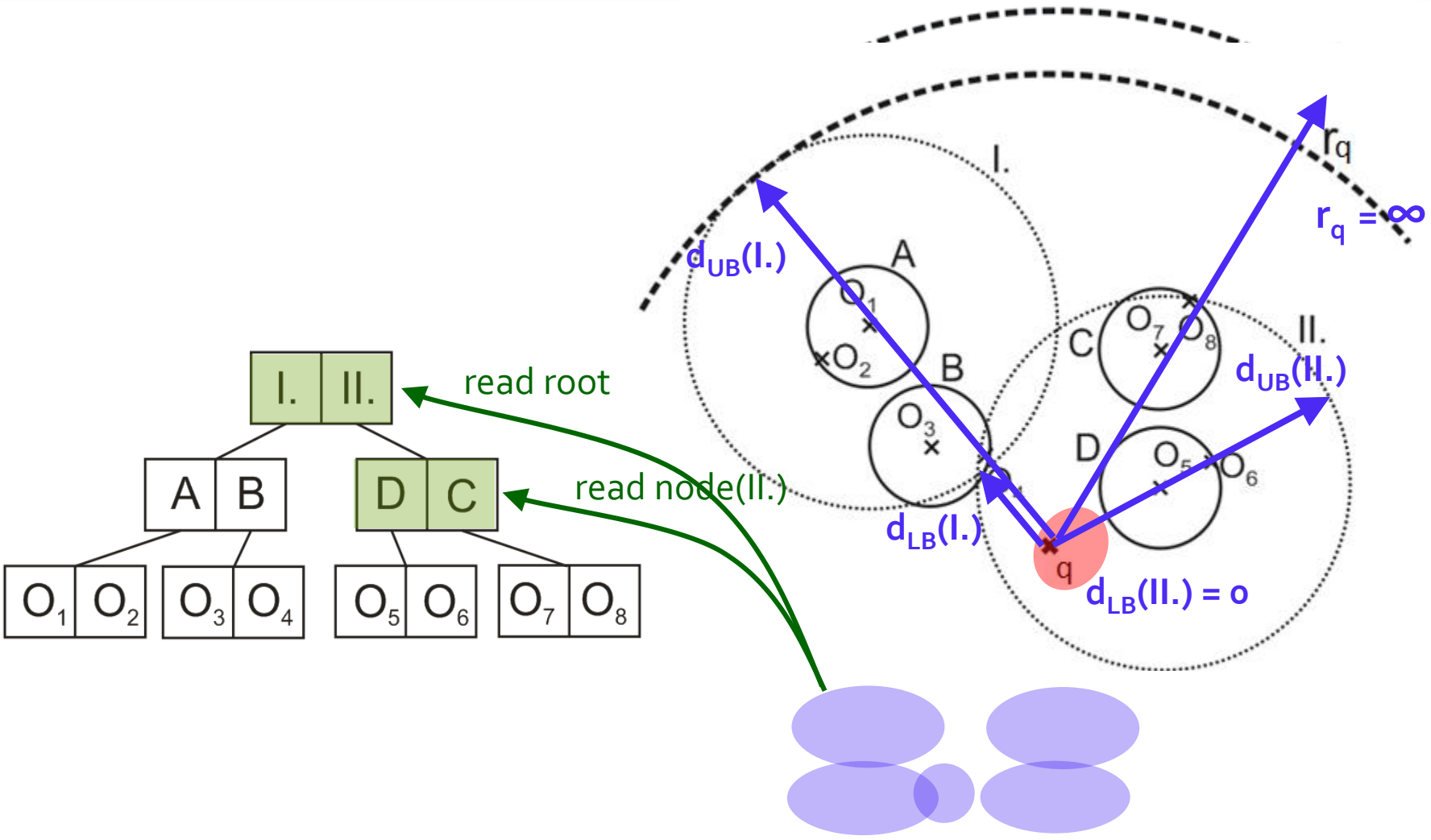




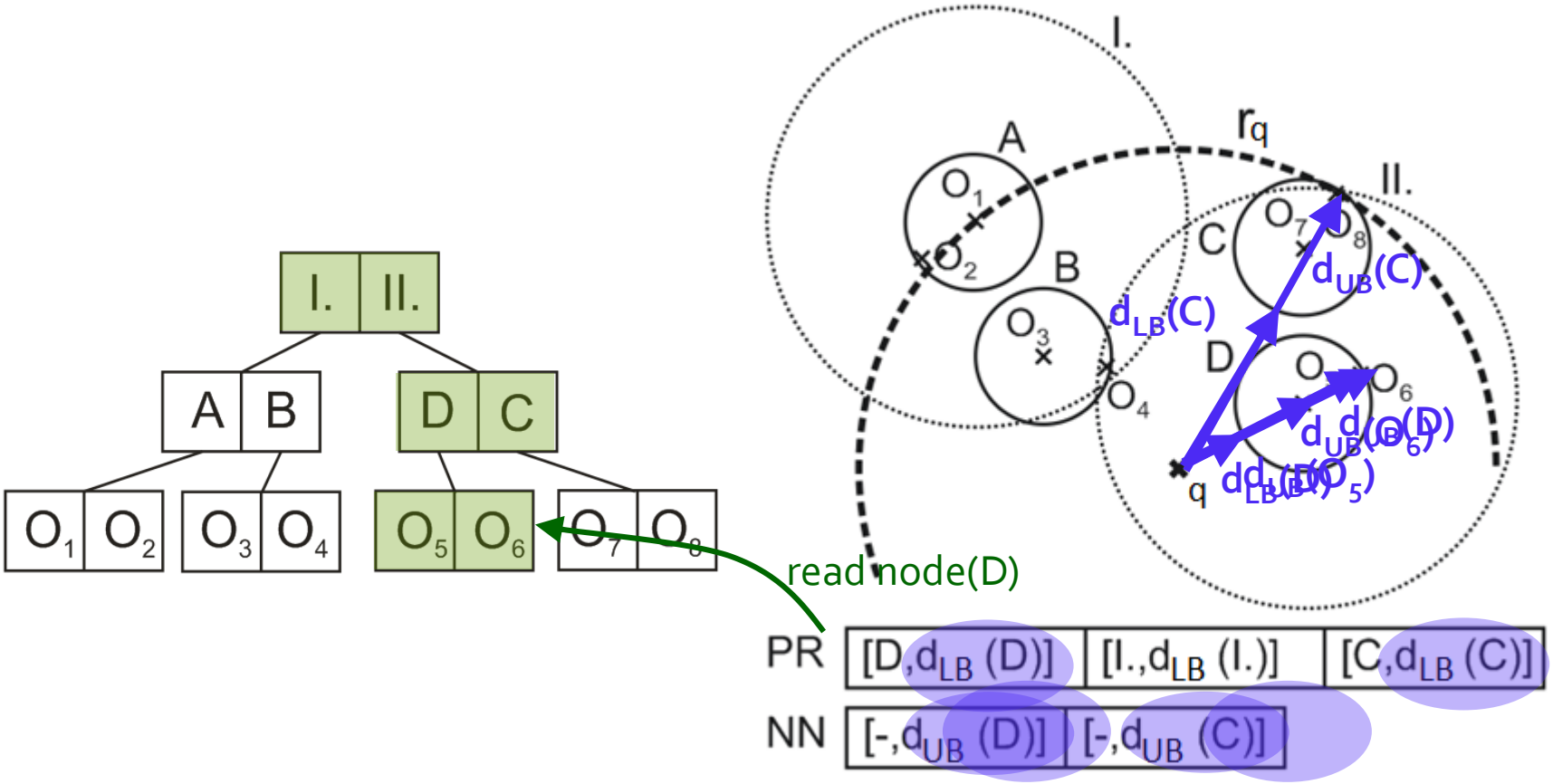
M-tree – exact kNN search

- the idea
  - consider a range query **(q, r_q)** processing,
    however the query radius **r_q** is dynamic
  - at the beginning the dynamic radius **r_q** is infinite,
    PR queue contains just the M-tree root, and **NN array** is empty
  - the M-tree is traversed, inserting into the priority queue PR nodes that have to be processed (i.e., the query ball overlaps the nodes' balls)
    - for each node in PR, its lower-bound distance (to the ball) is also stored
  - in the NN array, candidates to k nearest neighbors are stored
    - either directly objects (and distances to q)
      or candidate balls (and upper-bound distances to q)
    - NN[k] determines the dynamic query radius
  - during kNN processing, nodes are fetched from PR, updating PR and NN
    - note that NN[k] is decreasing, so that some nodes in PR are pruned prior to their fetch

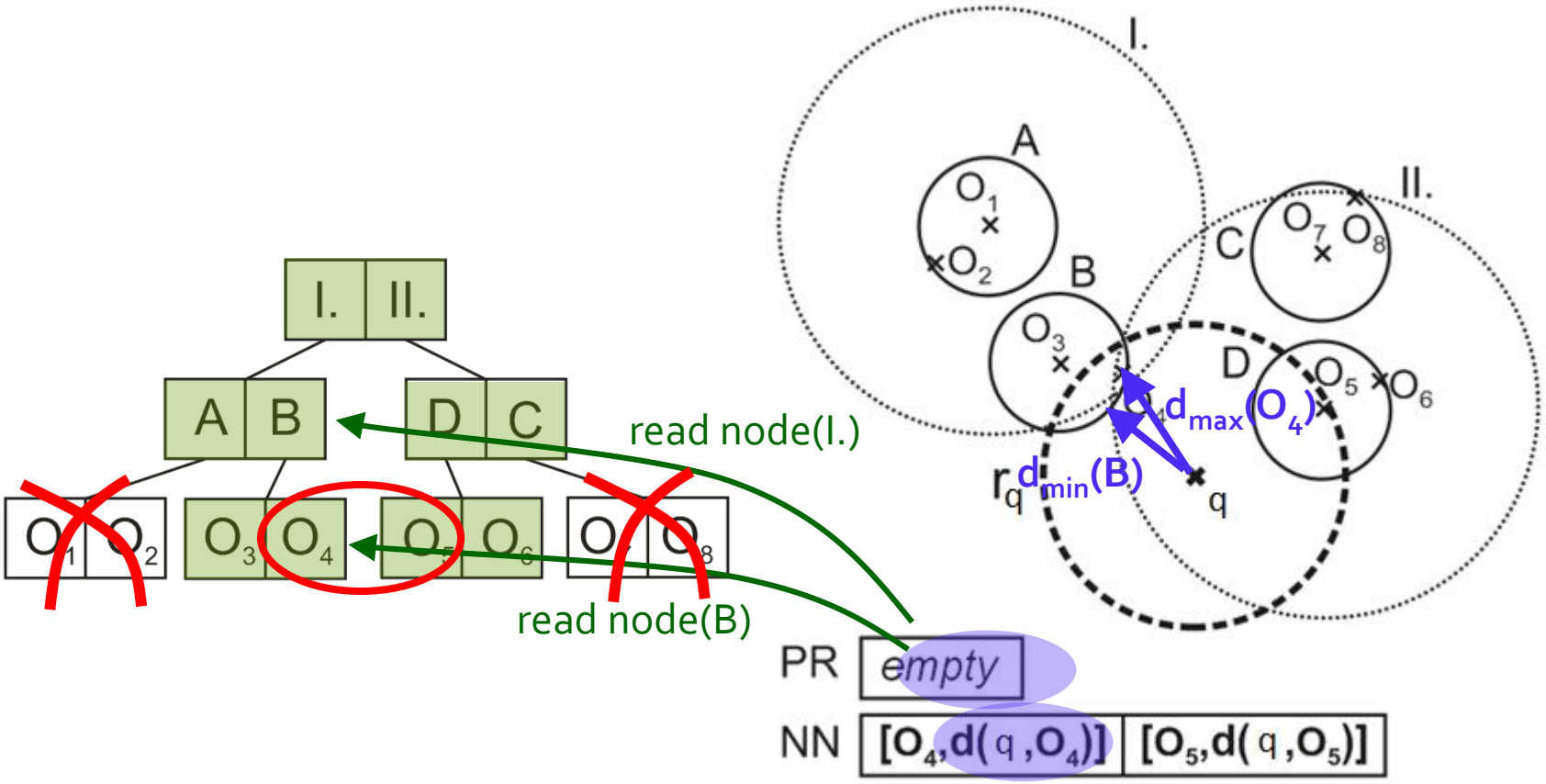
Example: exact 2NN search in M-tree



Example: exact 2NN search in M-tree



Example: exact 2NN search in M-tree

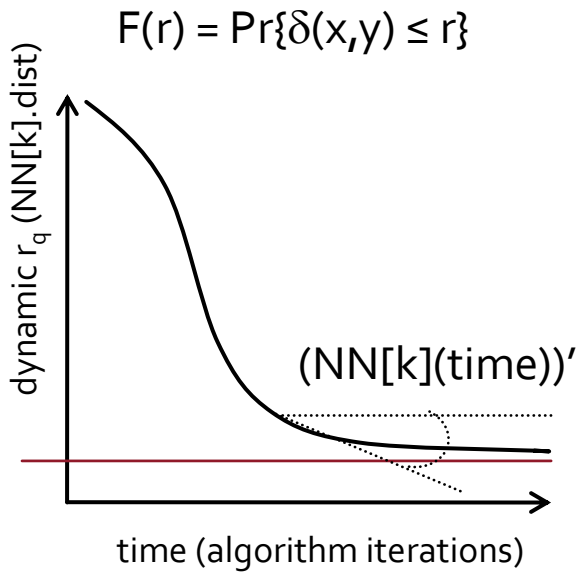


M-tree – approximate kNN search

- the exact kNN algorithm could be adjusted to provide faster approximate search
- we consider four heuristics
  - slowdown of improvements
    - if the NN array changes only slowly, let's terminate
  - approximately correct search
    - returned k nearest neighbors are guaranteed to be not much further than the true k nearest neighbors
  - good fraction approximation
    - if the distance to the candidate k^{th} neighbor is likely to represent small enough portion of the database, let's terminate
  - PAC queries (probably approximately correct queries)

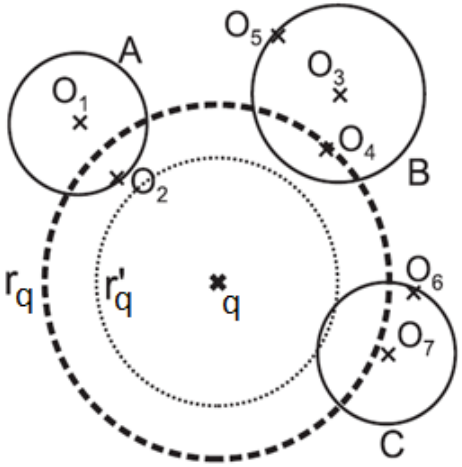
M-tree – approximate kNN search

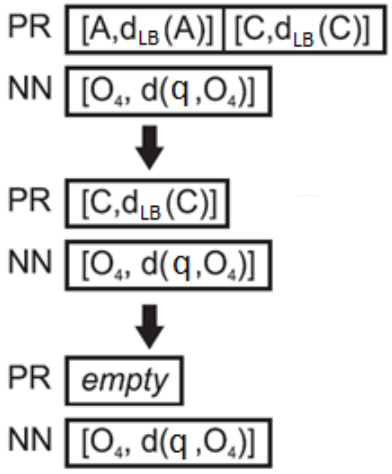
- slowdown of improvements
  - consider the function NN[k] in time, i.e., how the dynamic query radius improves during the kNN processing
  - given a user-defined parameter κ, let's terminate the kNN search as soon as the first derivative (absolute value) of NN[k](time) falls below κ
    - i.e., the small improvements of the result are not worth the effort



M-tree – approximate kNN search

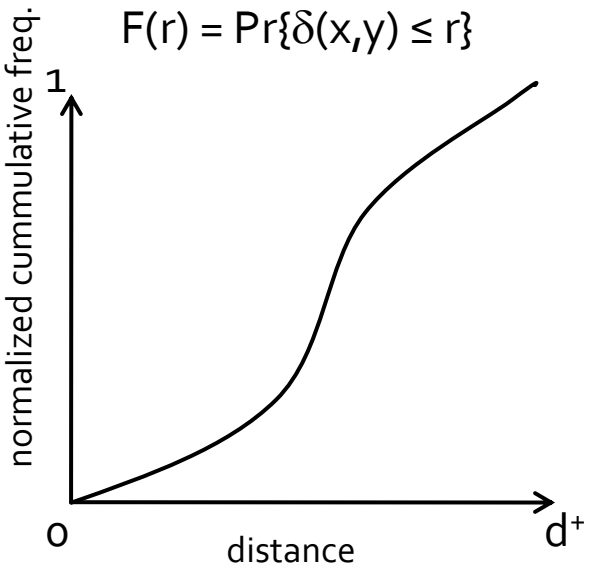
- approximately correct search
  - consider a user-defined error threshold ε, while relaxing the task of kNN search to $(1+\varepsilon)$kNN search
    - i.e., the neighbor returned is allowed to be up to $(1+\varepsilon)$ times further from **q** than the real one
  - algorithm adjustment
    - nodes from PR are filtered more aggressively using $r_q' = r_q/(1+\varepsilon)$ instead of r_q





M-tree – approximate kNN search

- good fraction approximation
  - given query object **q** and a user-defined fraction **κ** (of the database) that has to be inspected during a (kNN) search
  - for a range query **(q, r)**, let's estimate the fraction of database falling into **(q, r)**
  - then, in the kNN algorithm we test if the database fraction estimated for the actual **(q, NN[k])** is greater than **κ**
    – if so, terminate
    - the database fraction for (q, r) is computed as $F(r) = Pr\{\delta(x,y) \leq r\}$ using the cummulative distance distribution histogram

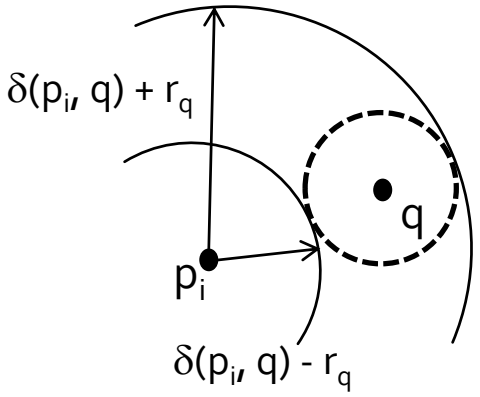


M-tree – approximate kNN search

- PAC queries
  - probably approximately correct queries
  - combination of the probabilistic and approximately correct methods
    - e.g., the two previous methods
  - in M-tree's kNN algorithm
    - nearest neighbors are searched, but guaranteeing only probability κ that the result objects will be at least $(1+\varepsilon)$ nearest neighbors
    - the terminating condition in the kNN algorithm
      - estimate the distance to k^{th} nearest neighbor (query radius $\mathbf{r_q}$)
        - by testing $F(r_q)$, until $F(r_q)$ * size of the database = k
      - nodes from PR are filtered using $r_q' = r_q/(1+\varepsilon)$

Approximate search in pivot tables

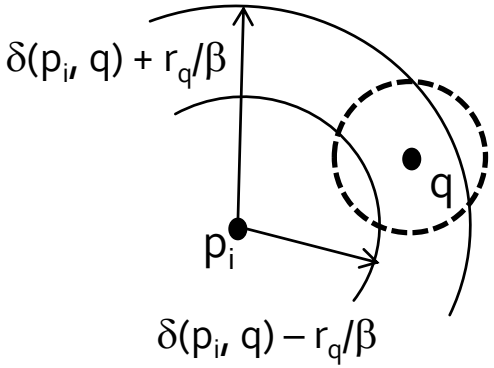
- consider pivot tables and a range query **(q, r_q)**
- using pivots, the exact search algorithm must check all objects that are within all the rings obtained by the pivots and the ranqe query (see on the right)



- in the approximate version, consider a user-defined threshold δ, that guarantees that the search is correct with the probability δ
  - the rings are "thinned" using β, computed as

$$\beta \leq \frac{r_Q \sqrt{1-(1-\delta)^{\frac{1}{p}}}}{\sqrt{2}\sigma}$$

where p is number of pivots, and σ is the variance over the distance distribution



Permutation indexes

- consider a set of *k* pivots p_i and a pivot table (distance matrix)
- instead of using the distance matrix, let's create for each database object u a permutation of pivots Π_u, represented as an ordered set of pivots' indices
- hence, no distances are stored in the index, just the ordering of pivots
- allows compact storage
  - $kn \log_2 k$ bits
    instead of $32kn$ bits used by pivot tables

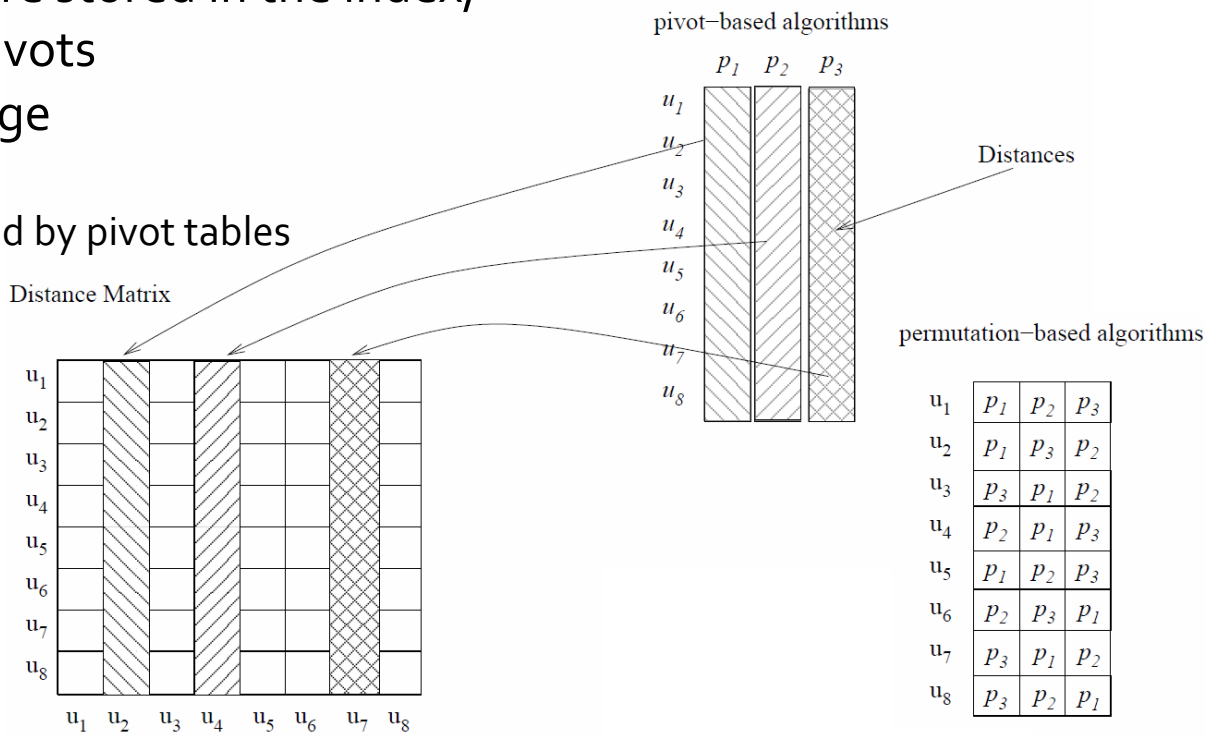


image from Chávez et al., Effective Proximity Retrieval by Ordering Permutations, IEEE Tran. on Patt. Anal. and Mach. Int., 30(9), 2008

Permutation indexes

- similarity for permutations
  - Spearman Rho

$$S_\rho(\Pi_u, \Pi_q) = \sum_{1 \le i \le k} \left(\Pi_u^{-1}(i) - \Pi_q^{-1}(i)\right)^2$$

  - Spearman Footrule

$$F(\Pi_u, \Pi_q) = \sum_{1 \le i \le k} |\Pi_u^{-1}(i) - \Pi_q^{-1}(i)|$$

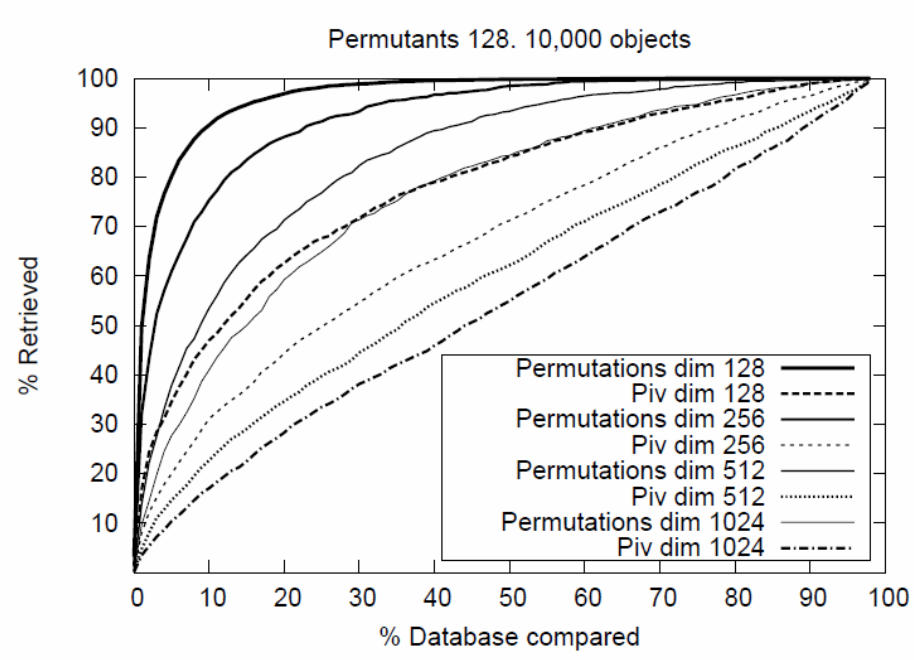
  where $\Pi^{-1}{}_u(i)$ is the position of pivot p_i in the permutation Π_u

- thus, instead of the **geometric** framework the **combinatorial** framework is used

Permutation indexes

- the approximate queries using permutation indexes are, in fact, similar to the pivot-table ones
- instead of L_∞ on vectors in pivot space, the Spearman Rho/Footrule on pivot permutations is used for sorting the objects in the first step
- the second (refinement) step is the same, using some early termination condition

Permutation indexes



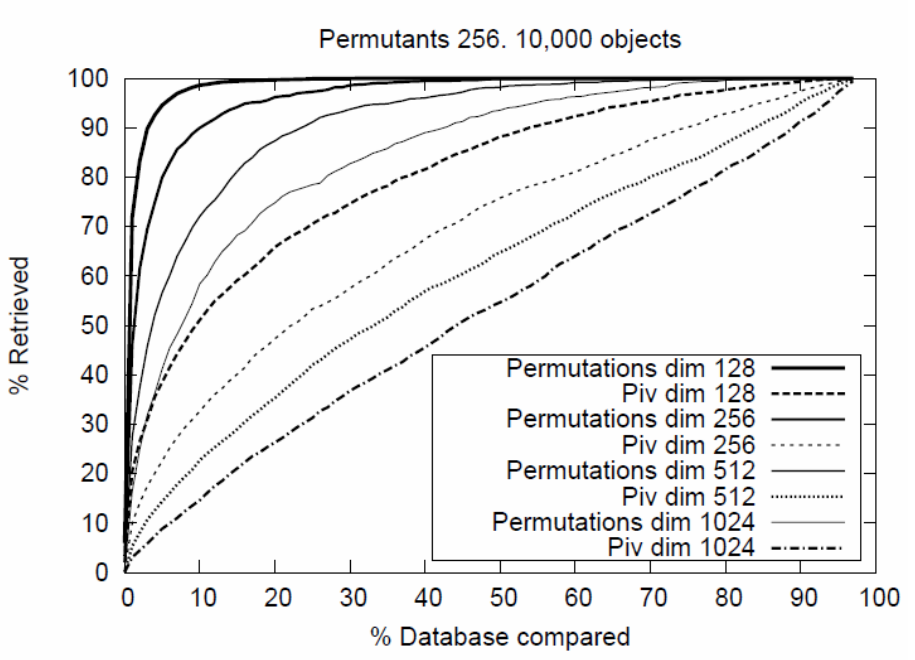


image from Chávez et al., Effective Proximity Retrieval by Ordering Permutations, IEEE Tran. on Patt. Anal. and Mach. Int., 30(9), 2008