
SIMILARITY SEARCH

The Metric Space Approach

Pavel Zezula, Vlastislav Dohnal,
Giuseppe Amato



Based on a Book

Published by Springer



Similarity Search

The Metric Space Approach

Series: Advances in Database Systems, Vol. 32

Zezula, P., Amato, G., Dohnal, V., Batko, M.

2006, XVIII, 220 p., Hardcover

ISBN: 0-387-29146-6

January 2006

Web page:

<http://www.nmis.isti.cnr.it/amato/similarity-search-book/>



Based on a Book by Springer



Table of contents:

Dedication.- Foreword.- Preface.- Acknowledgements.

Part I Metric Searching in a Nutshell:

- *Foundations of Metric Space Searching.*
- *Survey of Existing Approaches.*

Part II Metric Searching in Large Collections of Data:

- *Centralized Index Structures.*
- *Approximate Similarity Search.*
- *Parallel and Distributed Indexes.*

References.- Author Index.- Index.- Abbreviations.



Outline of the talk

1. **similarity, metric space and distance measures (15')**
2. similarity queries, metric partitioning principles (15')
3. query execution strategies (15')
4. avoiding distance computations (15')
5. filtering, pivot choosing and metric transformations (15')
6. short survey of metric space indexes (15')
7. M-tree family, D-index - performance evaluation (30')
8. approximate similarity search (30')
9. scalable and distributed indexes (30')

The Importance of Similarity

- Quotation:

“An ability to assess similarity lies close to the core of cognition. The sense of sameness is the very keel and backbone of our thinking. An understanding of problem solving, categorization, memory retrieval, inductive reasoning, and other cognitive processes require that we understand how humans assess similarity.”

- *MIT Encyclopedia of the Cognitive Sciences, Cambridge, MA, MIT Press 2006, pp. 763-765*

Digital Data Explosion

- Everything we *see, read, hear, write, and measure* can now be in a **digital** form!!
- In the next three years, we will create more data than has been produced in all of human history.
- Estimations:
 - 93% of produced data is digital
 - digital text is important – current technology is functional
 - multimedia, scientific, sensor, etc. data is becoming **prevalent**

The Search Problem

- Research reports indicate that employees spend roughly 25 to 35 percent of their time searching for the information they need to do their jobs.

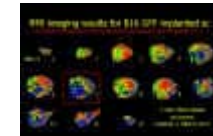
- **Claim**

As the variety of data types is fast going towards creating a database utilised by people, the computer systems must be able to model their fundamental reasoning paradigms, naturally based on similarity.

Requirements of New Applications

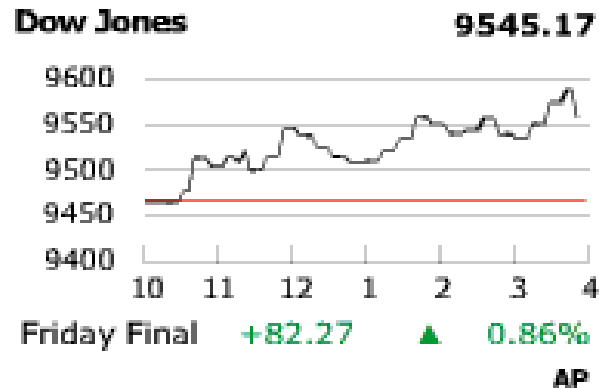
Medicine:

- *Magnetic Resonance Images (MRI)*



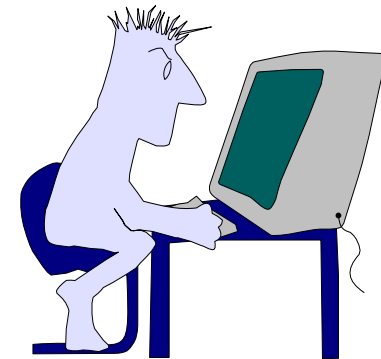
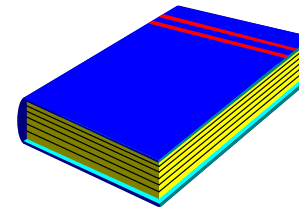
Finance:

- *stocks with similar time behavior*

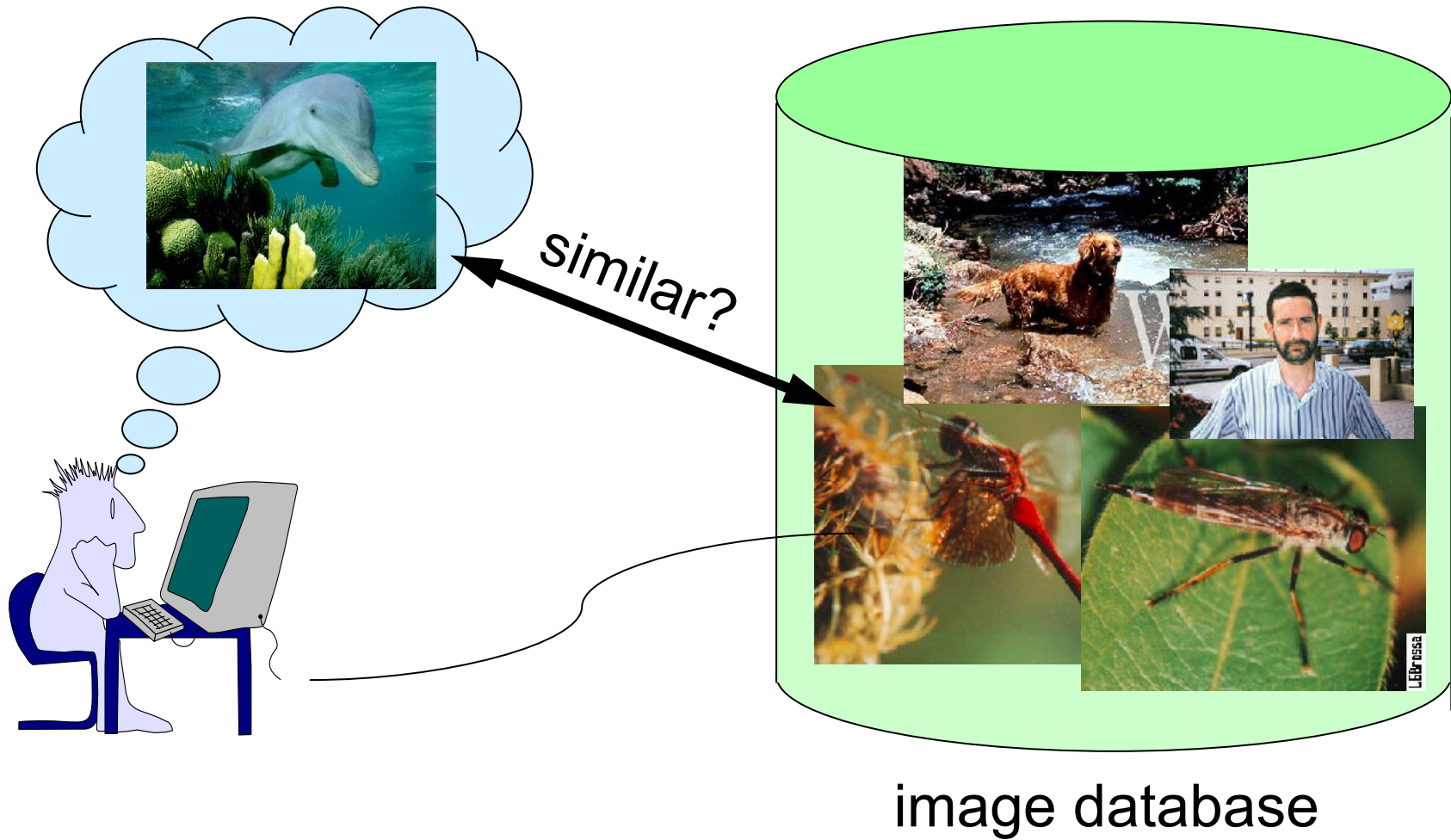


Digital library:

- *text retrieval*
- *multimedia information retrieval*

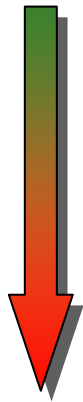


Search Problem

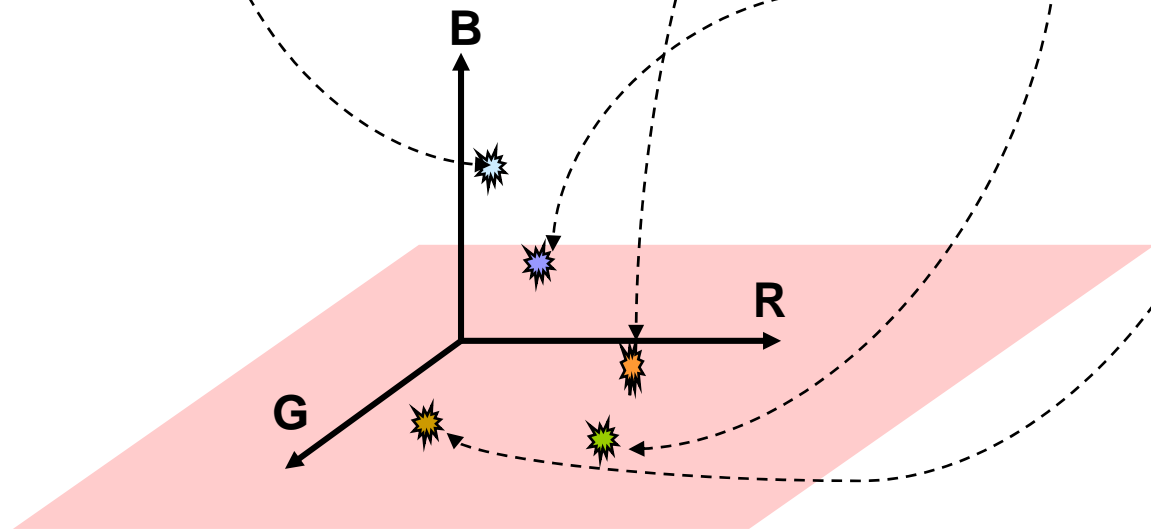
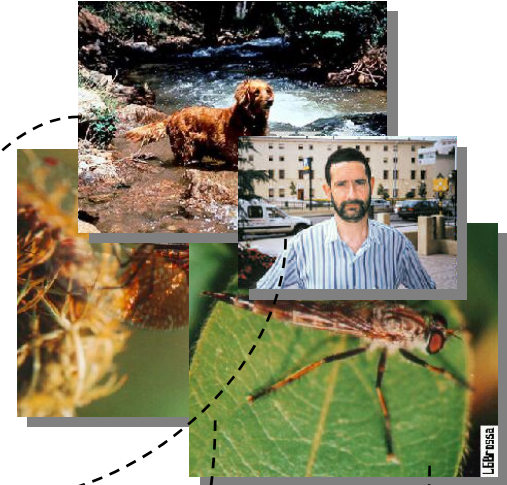


Feature-based Approach

image layer



feature layer



Distance Searching Problem

Definition (divide and conquer):

- Let \mathcal{D} be a feature domain and d a distance measure on objects from \mathcal{D}
- Given a set $X \subseteq \mathcal{D}$ of n elements:

preprocess or structure the data so that similarity queries are answered efficiently.

Metric Space: Abstraction of Similarity

- $\mathcal{M} = (\mathcal{D}, d)$ [Kel55]

- Data domain \mathcal{D}

- Total (distance) function $d: \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$ (metric function or metric)

- The metric space postulates:

- *non negativity* $\forall x, y \in \mathcal{D}, d(x, y) \geq 0$

- *symmetry* $\forall x, y \in \mathcal{D}, d(x, y) = d(y, x)$

- *identity* $\forall x, y \in \mathcal{D}, x = y \Leftrightarrow d(x, y) = 0$

- *triangle inequality* $\forall x, y, z \in \mathcal{D}, d(x, z) \leq d(x, y) + d(y, z)$

Metric Space

- Another specification:

- (p1) *non negativity* $\forall x, y \in \mathcal{D}, d(x, y) \geq 0$
- (p2) *symmetry* $\forall x, y \in \mathcal{D}, d(x, y) = d(y, x)$
- (p3) *reflexivity* $\forall x \in \mathcal{D}, d(x, x) = 0$
- (p4) *positiveness* $\forall x, y \in \mathcal{D}, x \neq y \Rightarrow d(x, y) > 0$
- (p5) *triangle inequality* $\forall x, y, z \in \mathcal{D}, d(x, z) \leq d(x, y) + d(y, z)$

Pseudo Metric

- Property (*p4 - positiveness*) does not hold
- If all objects at distance 0 are considered as a single object, we get the metric space:
 - To be proved $d(x, y) = 0 \Rightarrow \forall z \in \mathcal{D}, d(x, z) = d(y, z)$
 - Since $d(x, z) \leq d(x, y) + d(y, z)$
 $d(y, z) \leq d(x, y) + d(x, z)$
 - We get $d(x, z) = d(y, z), d(x, y) = 0$

Quasi Metric

- Property (*p2 - symmetry*) does not hold, e.g.
 - Locations in cities – one way streets
- Transformation to the metric space:

$$d_{sym}(x, y) = d_{asym}(x, y) + d_{asym}(y, x)$$

Super Metric

- Also called the ultra metric
- Stronger constraint on (*p5 – triangle inequality*)

$$\forall x, y, z \in \mathcal{D} : d(x, z) \leq \max\{d(x, y), d(y, z)\}$$

- At least two sides of equal length - isosceles triangle
- Used in evolutionary biology (phylogenetic trees)

Distance Measures

- Discrete
 - functions returning only a small (pre-defined) set of values

- Continuous
 - functions in which the cardinality of the set of values returned is very large or infinite.

Minkowski Distances

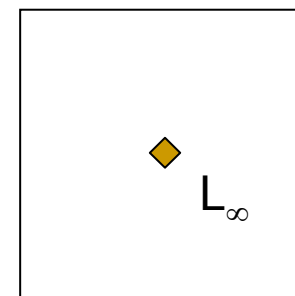
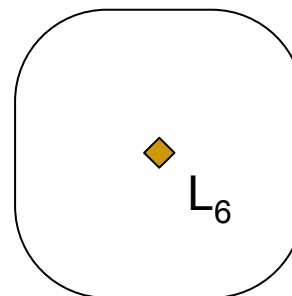
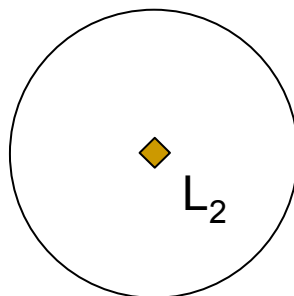
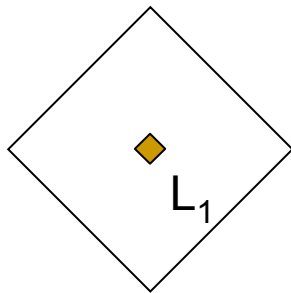
- Also called the L_p metrics
- Defined on n dimensional vectors

$$L_p[(x_1, \dots, x_n), (y_1, \dots, y_n)] = \sqrt[p]{\sum_{i=1}^n |x_i - y_i|^p}$$

Special Cases

- L_1 – Manhattan (City-Block) distance
- L_2 – Euclidean distance
- L_∞ – maximum (infinity) distance

$$L_\infty = \max_{i=1}^n |x_i - y_i|$$



Quadratic Form Distance

- Correlated dimensions – cross talk – e.g. color histograms [FBF⁺94,HSE⁺95,SK97]

$$d_M(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \cdot M \cdot (\vec{x} - \vec{y})}$$

- M – positive semidefinite matrix $n \times n$
 - if $M = \text{diag}(w_1, \dots, w_n)$ → weighted Euclidean distance

$$d_M(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^n w_i (x_i - y_i)^2}$$

Example

- 3-dim vectors of blue, red, and orange colors:

- Pure red: $\vec{v}_{red} = (0,1,0)$
- Pure orange: $\vec{v}_{orange} = (0,0,1)$
- Pure blue: $\vec{v}_{blue} = (1,0,0)$

- Blue and orange images are equidistant from the red one

$$L_2(\vec{v}_{red}, \vec{v}_{orange}) = L_2(\vec{v}_{red}, \vec{v}_{blue}) = \sqrt{2}$$

Example (cont.)

- Human color perception:
 - Red and orange are more alike than red and blue.
- Matrix specification:

$$M = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.9 \\ 0.0 & 0.9 & 1.0 \end{bmatrix}$$

blue	red	orange
------	-----	--------

blue red orange

- Distance of red and orange is $\sqrt{0.2}$
- Distance of red and blue is $\sqrt{2}$

Edit Distance

- Also called the Levenstein distance [Lev95]:
 - minimum number of atomic operations to transform string x into string y

- **insert** character c into string x at position i

$$ins(x, i, c) = x_1 x_2 \cdots x_{i-1} c x_i \cdots x_n$$

- **delete** character at position i in string x

$$del(x, i) = x_1 x_2 \cdots x_{i-1} x_{i+1} \cdots x_n$$

- **replace** character at position i in string x with c

$$replace(x, i, c) = x_1 x_2 \cdots x_{i-1} c x_{i+1} \cdots x_n$$

Edit Distance with Operation Weights

- If the weights (costs) of insert and delete operations differ, the edit distance is not symmetric.

- Example: $w_{insert} = 2$, $w_{delete} = 1$, $w_{replace} = 1$

$$d_{edit}(\text{"combine"}, \text{"combination"}) = 9$$

replacement $e \rightarrow a$, insertion t, i, o, n

$$d_{edit}(\text{"combination"}, \text{"combine"}) = 5$$

replacement $a \rightarrow e$, deletion t, i, o, n

Edit Distance: Generalizations

- Replacement of different characters can be different: $a \rightarrow b$ different from $a \rightarrow c$
- If it is symmetric, it is still the metric: $a \rightarrow b$ must be the same as $b \rightarrow a$
- Edit distance can be generalized to tree structures [AG97]

Jaccard's Coefficient

- Distance measure for sets A and B

$$d(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- Tanimoto similarity for vectors [Koh84]

$$d_{TS}(\vec{x}, \vec{y}) = 1 - \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|^2 + \|\vec{y}\|^2 - \vec{x} \cdot \vec{y}}$$

$\vec{x} \cdot \vec{y}$ is the *scalar product*

$\|\vec{x}\|$ is the *Euclidean norm*

Hausdorff Distance

- Distance measure for sets [HKR93]
- Compares elements by a distance d_e – not only $\{0,1\}$

Measures the extent to which each point of the “model” set A lies near some point of the “image” set B and vice versa.

IN OTHER WORDS

Two sets are within Hausdorff distance r from each other if and only if any point of one set is within the distance r from some point of the other set.

Hausdorff Distance (cont.)

$$d_p(x, B) = \inf_{y \in B} d_e(x, y),$$

$$d_p(A, y) = \inf_{x \in A} d_e(x, y),$$

$$d_s(A, B) = \sup_{x \in A} d_p(x, B),$$

$$d_s(B, A) = \sup_{y \in B} d_p(A, y).$$

$$d(A, B) = \max\{d_s(A, B), d_s(B, A)\}.$$

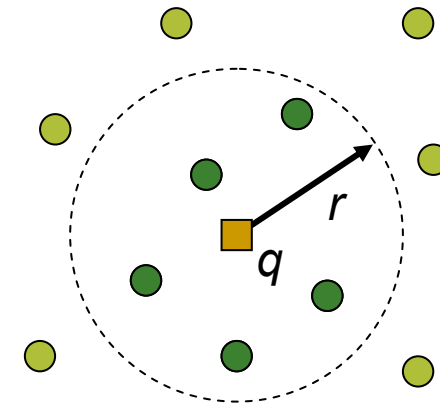
Outline of the talk

1. similarity, metric space and distance measures (15')
2. **similarity queries, metric partitioning principles (15')**
3. query execution strategies (15')
4. avoiding distance computations (15')
5. filtering, pivot choosing and metric transformations (15')
6. short survey of metric space indexes (15')
7. M-tree family, D-index - performance evaluation (30')
8. approximate similarity search (30')
9. scalable and distributed indexes (30')

Similarity Queries

- Range query
- Nearest neighbor query
- Reverse nearest neighbor query
- Similarity join
- Combined queries
- Complex queries

Similarity Range Query

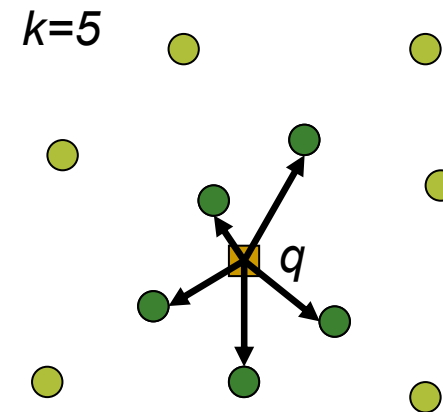


- range query
 - $R(q,r) = \{ x \in X \mid d(q,x) \leq r \}$

... all museums up to 2km from my hotel ...

Nearest Neighbor Query

- the nearest neighbor query
 - $NN(q) = x$
 - $x \in X, \forall y \in X, d(q,x) \leq d(q,y)$
- k-nearest neighbor query
 - $k\text{-}NN(q,k) = A$
 - $A \subseteq X, |A| = k$
 - $\forall x \in A, y \in X - A, d(q,x) \leq d(q,y)$



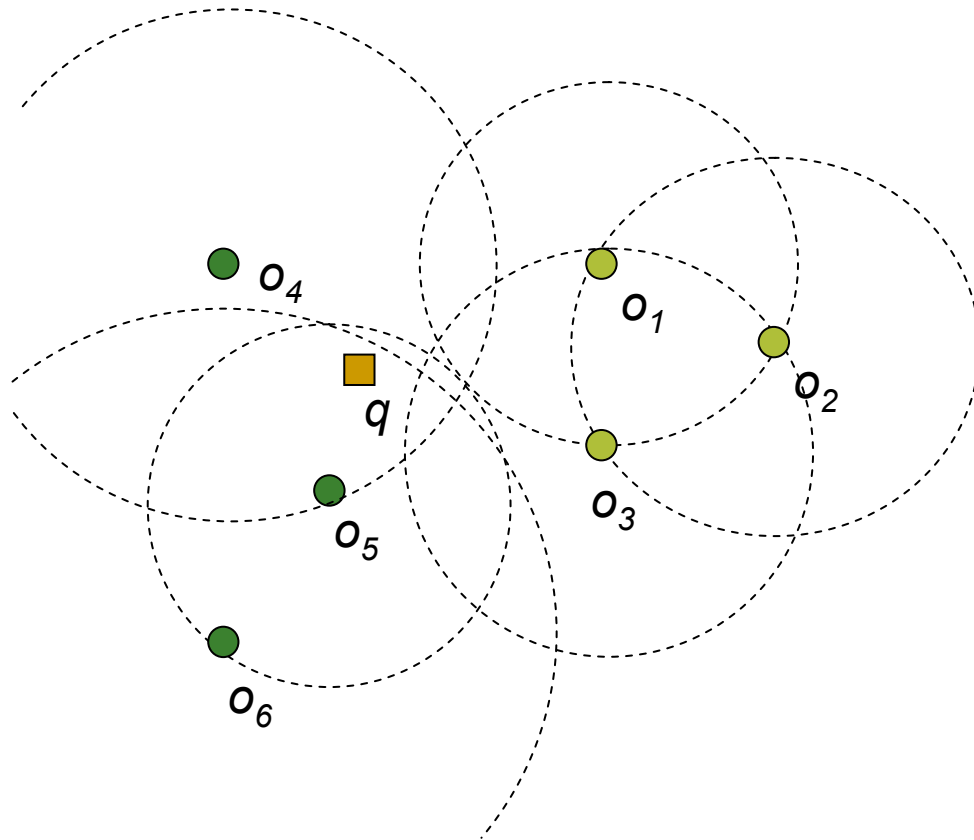
... five closest museums to my hotel ...

Reverse Nearest Neighbor [KM00]

$$kRNN(q) = \{R \subseteq X, \forall x \in R : q \in kNN(x) \wedge \\ \forall x \in X - R : q \notin kNN(x)\}$$

... all hotels with a specific museum as their nearest cultural heritage site ...

Example of 2-RNN



Objects o_4 , o_5 , and o_6 have q between their two nearest neighbor.

Similarity Join Queries [DGZ03]

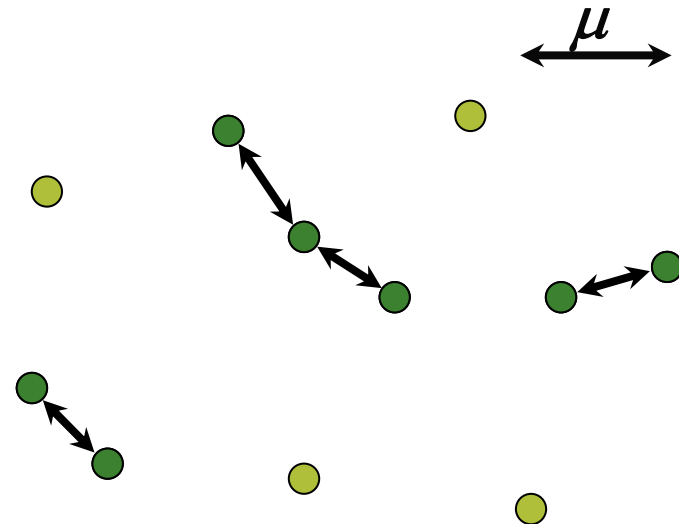
- similarity join of two data sets

$$X \subseteq \mathcal{D}, Y \subseteq \mathcal{D}, \mu \geq 0$$

$$J(X, Y, \mu) = \{(x, y) \in X \times Y : d(x, y) \leq \mu\}$$

- similarity self join $\Leftrightarrow X = Y$

*...pairs of hotels and museums
which are five minutes walk
apart ...*



Combined Queries

- Range + Nearest neighbors

$$kNN(q, r) = \{R \subseteq X, |R| \leq k \wedge \forall x \in R, y \in X - R : \\ d(q, x) \leq d(q, y) \wedge d(q, x) \leq r\}$$

- Nearest neighbor + similarity joins
 - by analogy

Complex Queries

- Find the best matches of *circular* **shape** objects with *red* **color**
- The best match for circular shape or red color needs not be the best match combined!!!

\mathcal{A}_0 Algorithm (FA) [Fag96]

- For each predicate (metric feature type) i
 - objects delivered in decreasing similarity (sorted access)
 - incrementally build sets X_i with best matches till

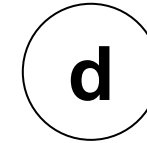
$$\forall i \mid \bigcap_i X_i \mid = k$$

- For all $o \in \bigcup_i X_i$
 - do random access to find unknown predicate values
 - establish the final rank $t(o)$ (fuzzy algebra, weighted sets, etc.) – monotonic function

Threshold Algorithm (TA) [Fag98]

- For each predicate i
 - deliver objects in decreasing similarity (sorted access)
 - as o is seen, do a random access to compute $t(o)$ - keep the k best objects
 - at each stage of sorted access, maintain the *threshold* \mathcal{T} considering current values in each predicate
 - as soon as at least k objects have been seen with grades not worse than \mathcal{T} , we have the result

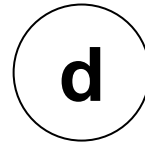
Threshold Algorithm: Example



$t(o) = \text{avg}(d_1, d_2)$: $\text{avg}(3, 2) = 2.5$ $\text{avg}(1, 3) = 2$ $\text{avg}(2, 4) = 3$ $\text{avg}(4, 1) = 2.5$

X_1 list (color)

X_2 list (shape)



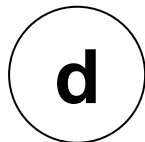
$\tau = 1$



$\tau = 2$



$\tau = 3$



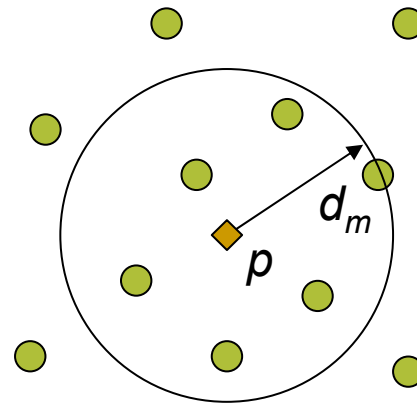
$\tau = 4$

Partitioning Principles

- Given a set $X \subseteq \mathcal{D}$ in $\mathcal{M}=(\mathcal{D},d)$, three basic partitioning principles have been defined:
 - Ball partitioning
 - Generalized hyper-plane partitioning
 - Excluded middle partitioning

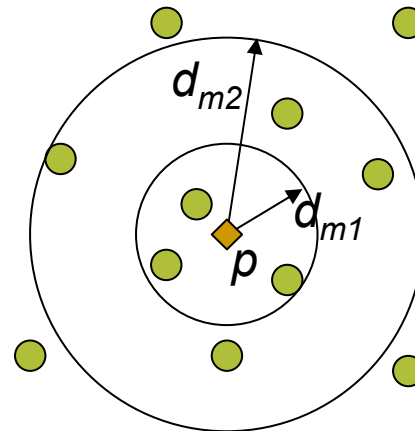
Ball Partitioning [Uhl91]

- Inner set: $\{ x \in X \mid d(p, x) \leq d_m \}$
- Outer set: $\{ x \in X \mid d(p, x) > d_m \}$



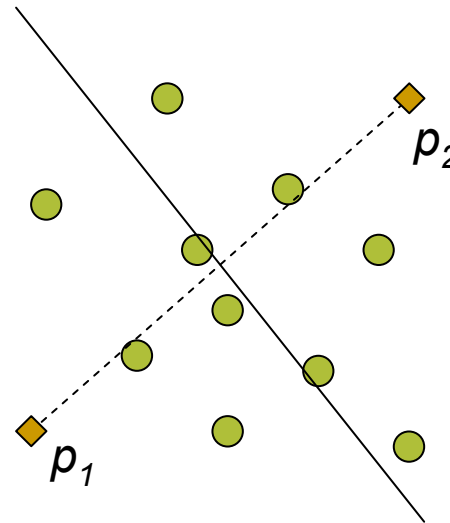
Multi-way Ball Partitioning

- Inner set: $\{ x \in X \mid d(p,x) \leq d_{m1} \}$
- Middle set: $\{ x \in X \mid d(p,x) > d_{m1} \wedge d(p,x) \leq d_{m2} \}$
- Outer set: $\{ x \in X \mid d(p,x) > d_{m2} \}$



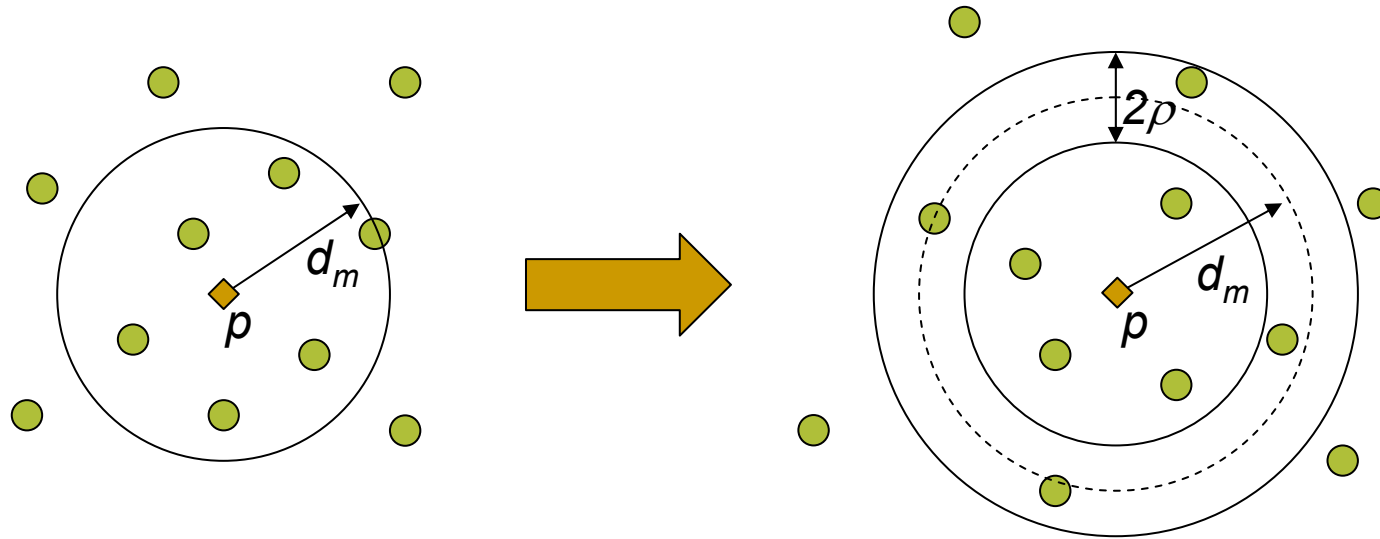
Generalized Hyper-plane [Uhl91]

- $\{ x \in X \mid d(p_1, x) \leq d(p_2, x) \}$
- $\{ x \in X \mid d(p_1, x) > d(p_2, x) \}$



Excluded Middle Partitioning [Yia99]

- Inner set: $\{ x \in X \mid d(p,x) \leq d_m - \rho \}$
- Outer set: $\{ x \in X \mid d(p,x) > d_m + \rho \}$



- Excluded set: otherwise

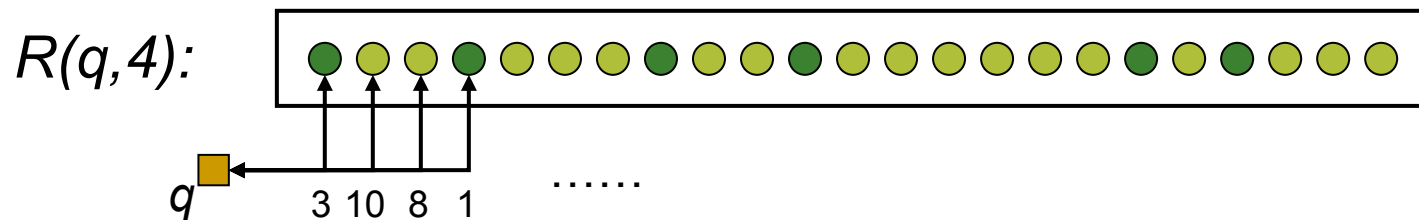
Outline of the talk

1. similarity, metric space and distance measures (15')
2. similarity queries, metric partitioning principles (15')
3. **query execution strategies (15')**
4. avoiding distance computations (15')
5. filtering, pivot choosing and metric transformations (15')
6. short survey of metric space indexes (15')
7. M-tree family, D-index - performance evaluation (30')
8. approximate similarity search (30')
9. scalable and distributed indexes (30')



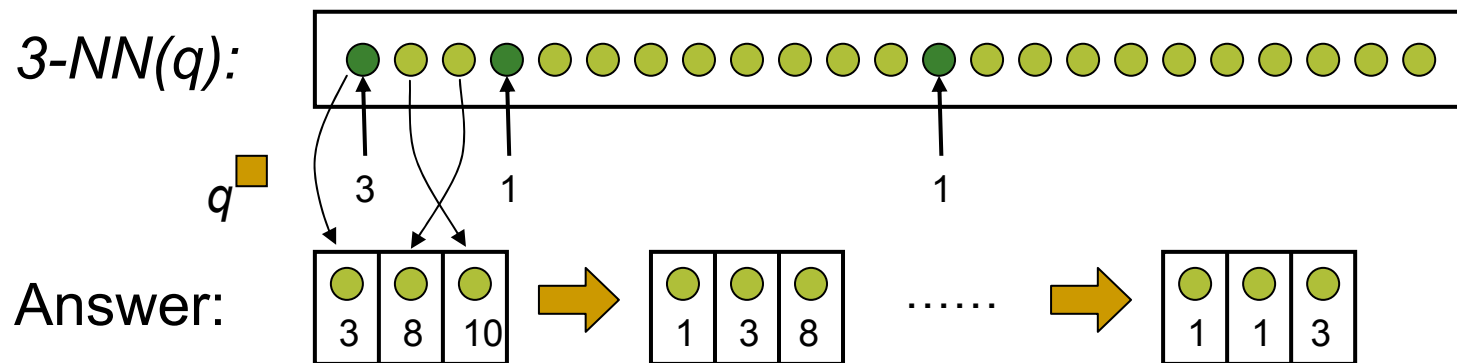
Basic Strategies

- Costs to answer a query are influenced by
 - Partitioning principle
 - Query execution algorithm
- Sequential organization & range query $R(q,r)$
 - All database objects are consecutively scanned and $d(q,o)$ are evaluated.
 - Whenever $d(q,o) \leq r$, o is reported on result



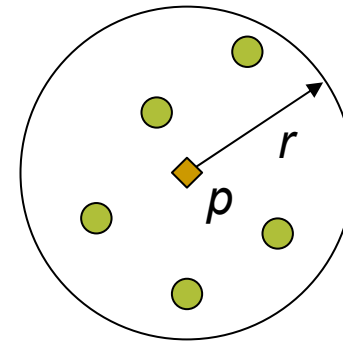
Basic Strategies (cont.)

- Sequential organization & k -NN query 3 -NN(q)
 - Initially: take the first k objects and order them with respect to the distance from q .
 - All other objects are consecutively scanned and $d(q, o)$ are evaluated.
 - If $d(q, o_i) \leq d(q, o_k)$, o_i is inserted to a correct position in answer and the last neighbor o_k is eliminated.



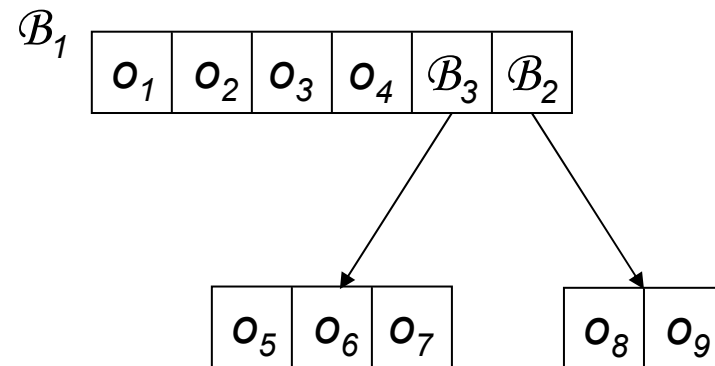
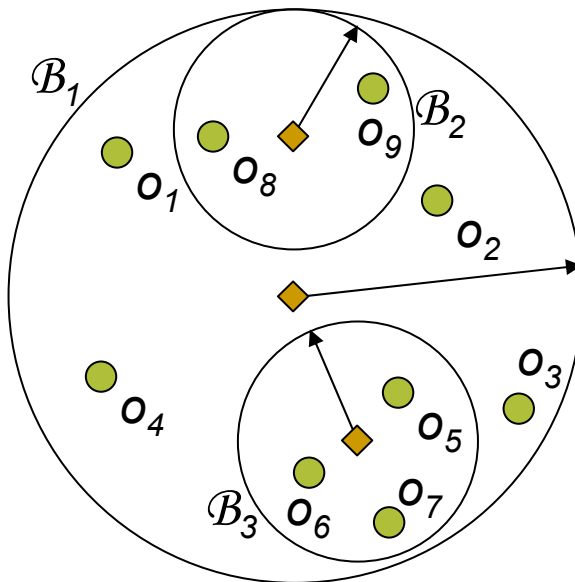
Hypothetical Index Organization

- A hierarchy of entries (nodes) $N=(G, \mathcal{R}(G))$
 - $G = \{e \mid e \text{ is object or } e \text{ is another entry}\}$
 - *Bounding region* $\mathcal{R}(G)$ covers all elements of G .
 - E.g. *ball region*: $\forall o, d(o,p) \leq r$
 - Each element belongs exactly to one G .
 - There is one *root* entry N .
- Any similarity query Q returns a set of objects
 - We can define $\mathcal{R}(Q)$ which covers all objects in response.



Example of Index Organization

- Using ball regions
 - Root node organizes four objects and two ball regions.
 - Child ball regions have two and three objects respectively.



Range Search Algorithm

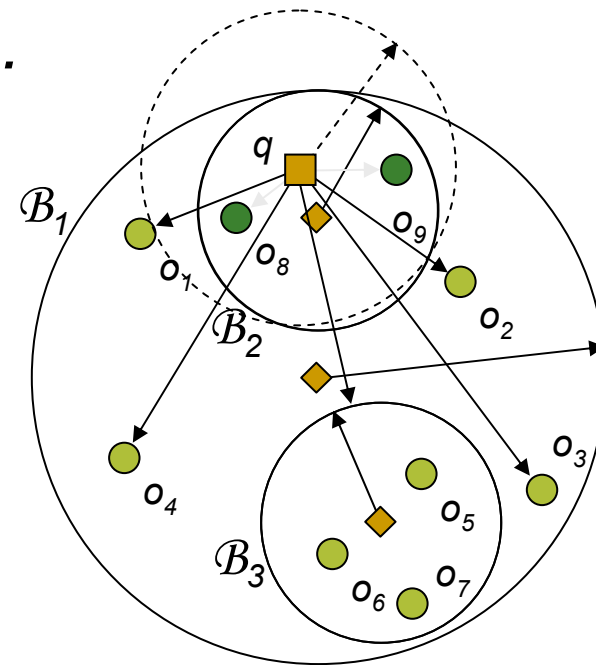
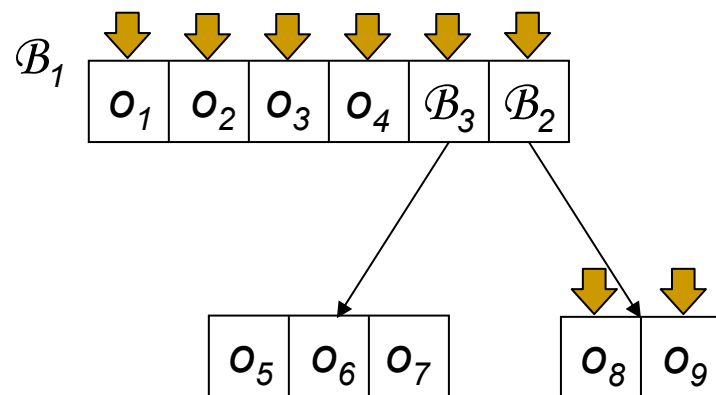
Given $Q = R(q,r)$:

- Start at the root.
- In the current node $N=(G, \mathcal{R}(G))$, process all elements of G :
 - object element $o_j \in G$:
 - if $d(q,o_j) \leq r$, report o_j on output.
 - non-object element $N'=(G', \mathcal{R}(G')) \in G$
 - if $\mathcal{R}(G')$ and $\mathcal{R}(Q)$ intersect, recursively search in N' .

Range Search Algorithm (cont.)

$R(q,r)$:

- Start inspecting elements in \mathcal{B}_1 .
- \mathcal{B}_3 is not intersected.
- Inspect elements in \mathcal{B}_2 .
- Search is complete.



Response = o_8, o_9

Nearest Neighbor Search Algorithm

- No query radius is given.
 - We do not know the distance to the k -th nearest neighbor.
- To allow filtering of unnecessary branches
 - The query radius is defined as the distance to the current k -th neighbor.
- *Priority queue PR* is maintained.
 - It contains regions that may include objects relevant to the query.
 - The regions are sorted with decreasing relevance.

NN Search Algorithm (cont.)

Given $Q=k\text{-}NN(q)$:

- Assumptions:

- The query region $\mathcal{R}(Q)$ is limited by the distance (r) to the current k -th neighbor in the response.
- Whenever PR is updated, its entries are sorted with decreasing proximity to q .
- Objects in the response are sorted with increasing distance to q . The response can contain k objects at maximum.

- Initialization:

- Put the root node to PR .
- Pick k database objects at random and insert them into response.



NN Search Algorithm (cont.)

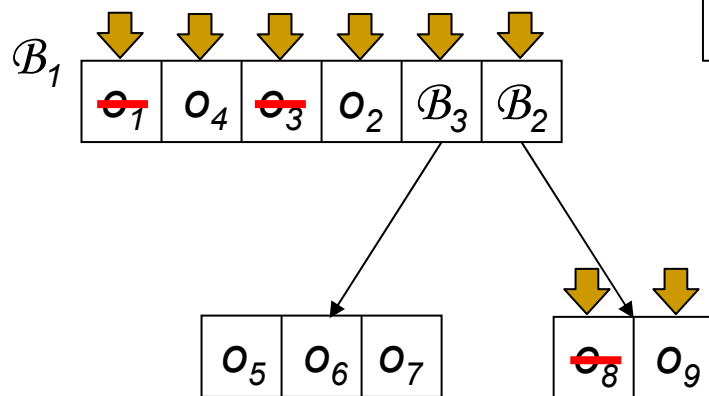
- While PR is not empty, repeat:
 - Pick an entry $N=(G, \mathcal{R}(G))$ from PR .
 - For each object element $o_j \in G$:
 - if $d(q, o_j) \leq r$, add o_j to the response. Update r and $\mathcal{R}(Q)$.
 - Remove entries from PR that cannot intersect the query.
 - For each non-object element $N'=(G', \mathcal{R}(G')) \in G$
 - if $\mathcal{R}(G')$ and $\mathcal{R}(Q)$ intersect, insert N' into PR .

- The response contains k nearest neighbors to q .

NN Search Algorithm (cont.)

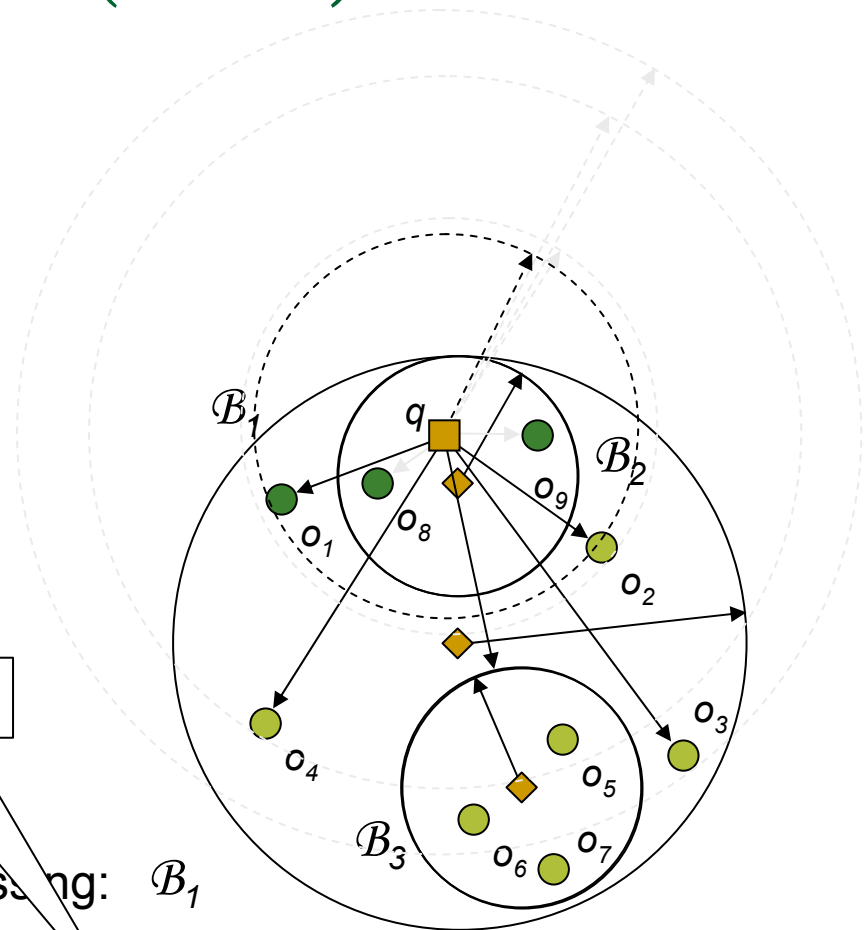
3-NN(q):

- Pick three random objects.
- Process \mathcal{B}_1
- Skip \mathcal{B}_3
- Process \mathcal{B}_2
- PR is empty, quit.



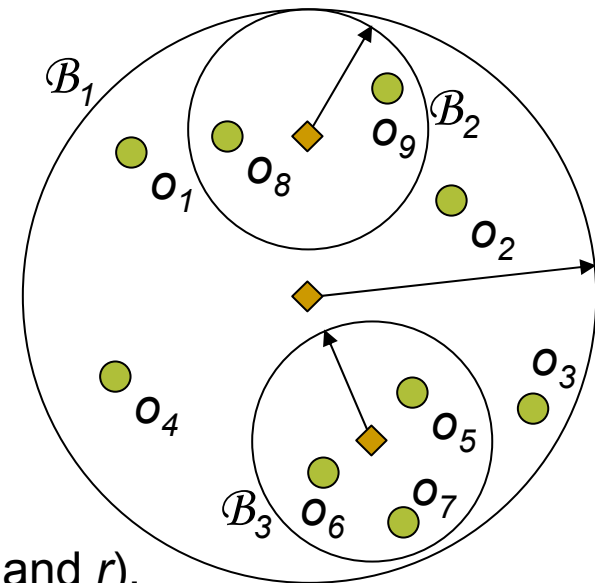
Final result

Processing: \mathcal{B}_1
 PR = \mathcal{B}_2
 Response = o_8, o_9, o_1



Incremental Similarity Search

- Hypothetical index structure is slightly modified:
 - Elements of type 0 are objects e_0 .
 - Elements e_1 are ball regions ($\mathcal{B}_2, \mathcal{B}_3$) containing only objects, i.e. elements e_0 .
 - Elements e_2 contain elements e_0 and e_1 , e.g., \mathcal{B}_1 .
 - Elements have associated distance functions from the query object q :
 - $d_0(q, e_0)$ – for elements of type e_0 .
 - $d_t(q, e_t)$ – for elements of type e_t .
 - E.g., $d_t(q, e_t) = d(q, p) - r$ (e_t is a ball with p and r).
 - For correctness: $d_t(q, e_t) \leq d_0(q, e_0)$



Incremental *NN* Search

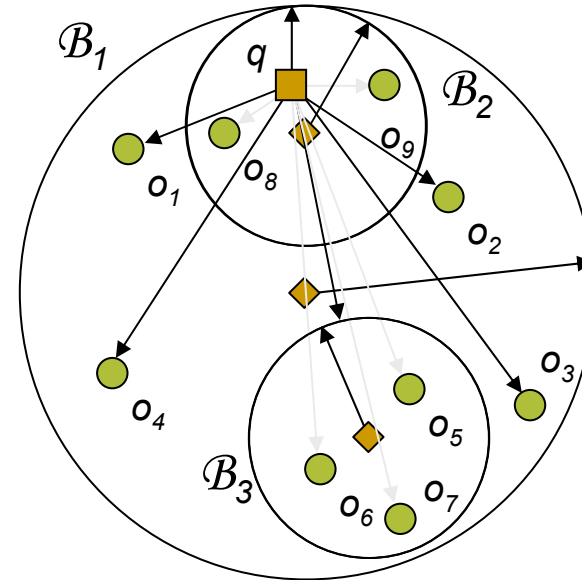
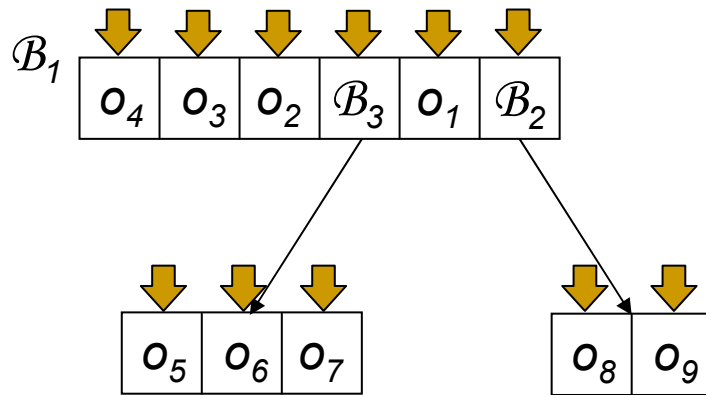
- Based on *priority queue PR* again
 - Each element e_t in *PR* knows also the distance $d_t(q, e_t)$.
 - Entries in the queue are sorted with respect to these distances.
- Initialization:
 - Insert the root element with the distance 0 into *PR*.

Incremental *NN* Search (cont.)

- While PR is not empty do
 - $e_t \leftarrow$ the first element from PR
 - if $t = 0$ (e_t is an object) then report e_t as the next nearest neighbor.
 - else insert each child element e_l of e_t with the distance $d_l(q, e_l)$ into PR .

Incremental *NN* Search (cont.)

$NN(q)$:



Processing: O_7

PR = (O_5, O_6) (O_5, O_7) (O_6, O_7) (O_8, O_9) (O_1, O_2) (O_2, O_3) (O_3, O_4)

Response = $O_8, O_9, O_1, O_2, O_5, O_4, O_6, O_3, O_7$

Outline of the talk

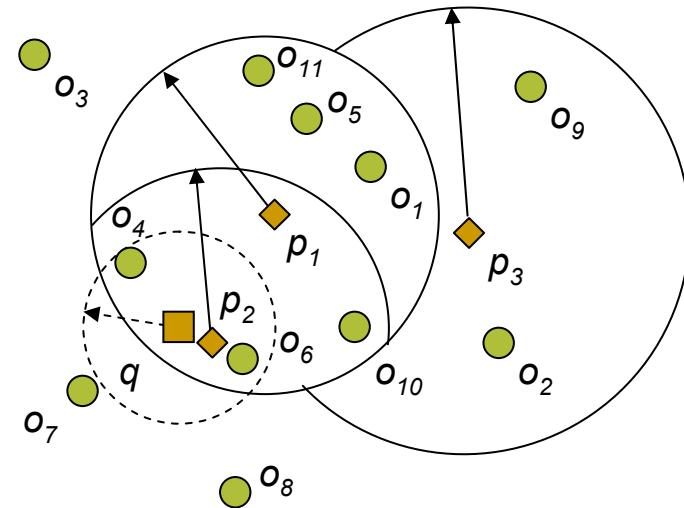
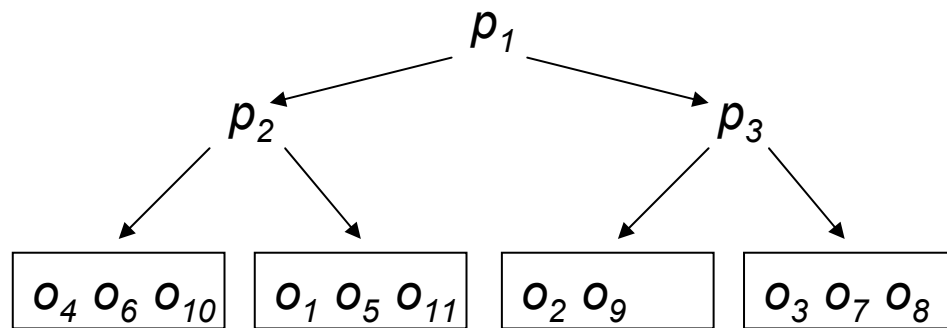
1. similarity, metric space and distance measures (15')
2. similarity queries, metric partitioning principles (15')
3. query execution strategies (15')
4. **avoiding distance computations (15')**
5. filtering, pivot choosing and metric transformations (15')
6. short survey of metric space indexes (15')
7. M-tree family, D-index - performance evaluation (30')
8. approximate similarity search (30')
9. scalable and distributed indexes (30')

Avoiding Distance Computations

- In metric spaces, the distance measure is expensive
 - E.g. edit distance, quadratic form distance, ...
- Limit the number of distance evaluations
 - It speeds up processing of similarity queries
- Pruning strategies [HS00,HS03a,Doh04]
 - Based on pre-computed distances
 - object-pivot
 - range-pivot
 - pivot-pivot
 - double-pivot
 - pivot filtering

Explanatory Example

- An index structure is built over 11 objects $\{o_1, \dots, o_{11}\}$
 - applies ball-partitioning



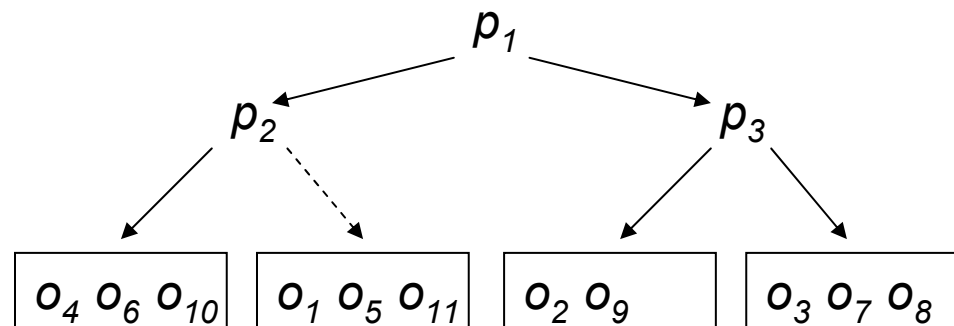
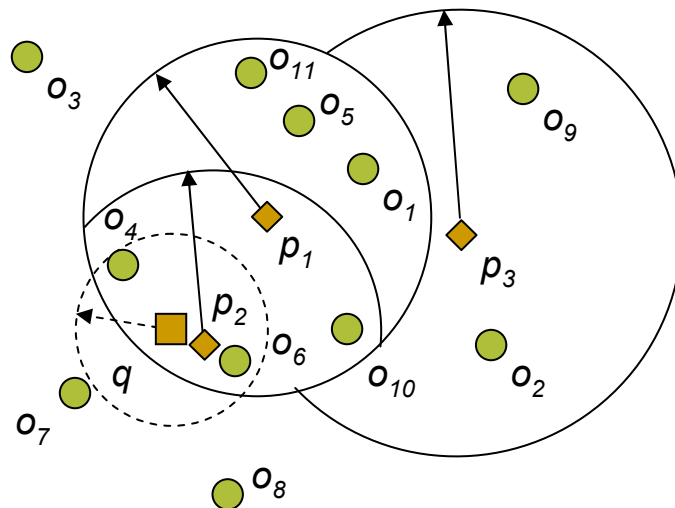
- Range query $R(q,r)$
 - Sequential scan needs 11 distance computations.
 - Reported objects: $\{o_4, o_6\}$

Explanatory Example & No Constraints

- Range query $R(q,r) = \{o_4, o_6\}$

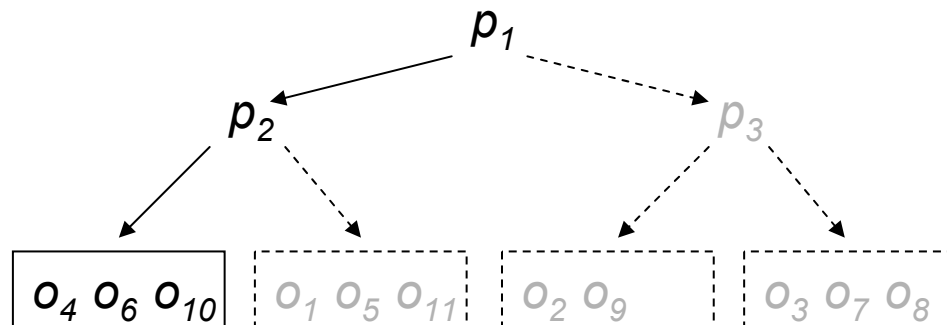
- In “ p_1 ”, both branches 1 distance
- In “ p_2 ”, the left branch only! 1 + 3 distances
- In “ p_3 ”, both branches 1 + 2 + 3 distances
- No constraints: 3+8 distance computations

3+3 node accesses



Object-Pivot Distance Constraint

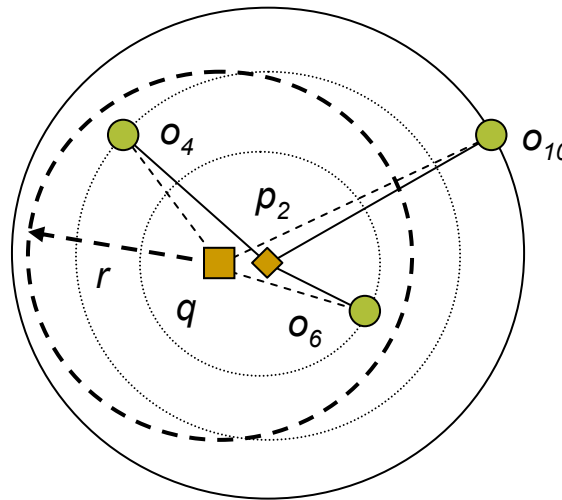
- Usually applied in leaf nodes
- Assume the left-most leaf is visited
 - Distances from q to o_4, o_6, o_{10} must be computed



- During insertion
 - Distances p_2 to o_4, o_6, o_{10} were computed

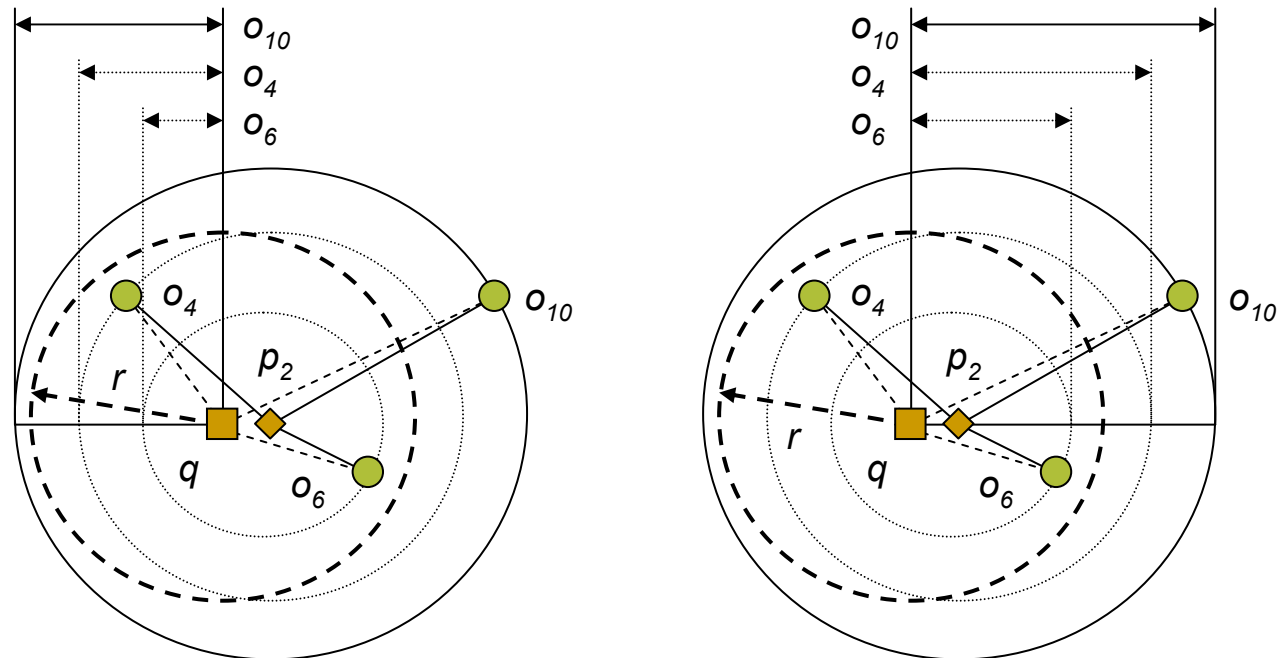
Object-Pivot Constraint (cont.)

- Having $d(p_2, o_4)$, $d(p_2, o_6)$, $d(p_2, o_{10})$ and $d(p_2, q)$
 - some distance calculations can be omitted
- Estimation of $d(q, o_{10})$
 - using only distances we cannot determine position of o_{10}
 - o_{10} can lie anywhere on the solid-line circle



Object-Pivot Constraint (cont.)

- o_{10} has two extreme positions



- Lower bound on $d(q, o_{10})$ is $|d(p_2, o_{10}) - d(q, p_2)|$
 - If greater than the query radius, an object cannot qualify. (o_{10})
- Upper bound on $d(q, o_{10})$ is $d(q, p_2) + d(p_2, o_{10})$
 - If less than the query radius, an object directly qualifies! (o_6)

Object-Pivot Constraint (summary)

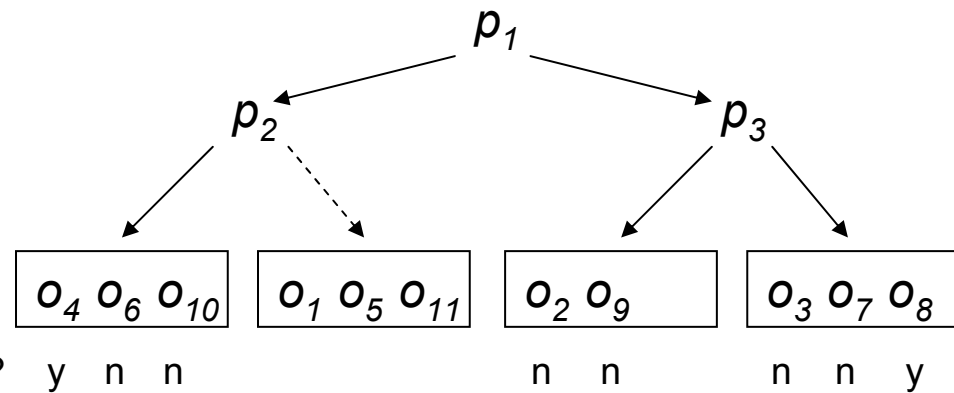
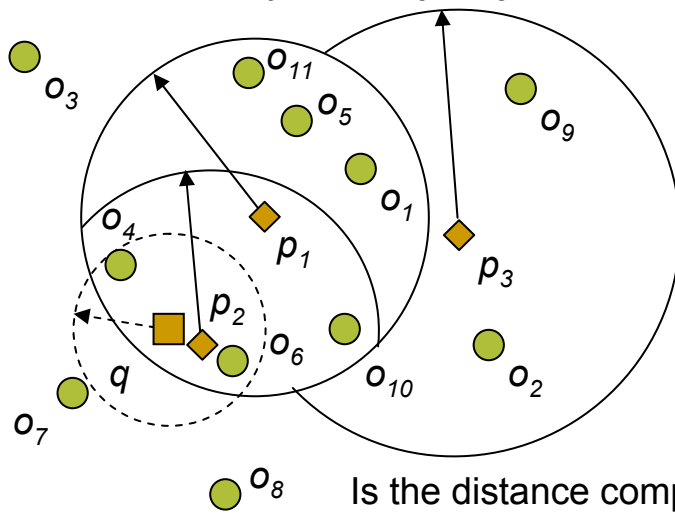


- Given a metric space $\mathcal{M}=(\mathcal{D},d)$ and three objects $q,p,o \in \mathcal{D}$, the distance $d(q,o)$ can be constrained:

$$|d(q,p) - d(p,o)| \leq d(q,o) \leq d(q,p) + d(p,o)$$

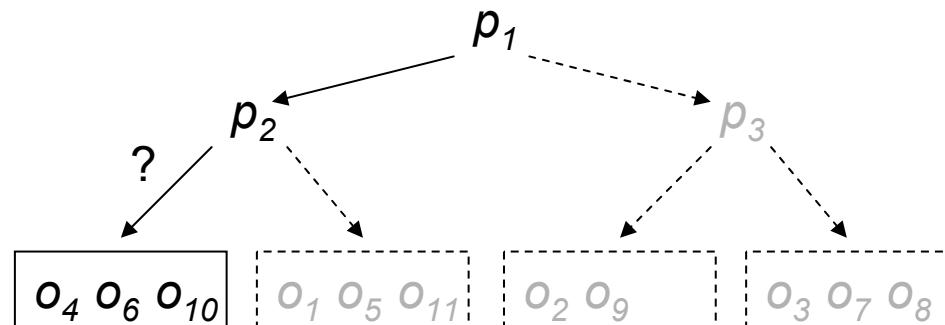
Explanatory Example & Object-Pivot Constraint

- Range query $R(q,r) = \{o_4, o_6\}$
 - Only *object-pivot* in leaves: 3+2 distance computations
3+3 node accesses
 - o_6 is included without computing $d(q, o_6)$
 - $o_{10}, o_2, o_9, o_3, o_7$ are eliminated directly.



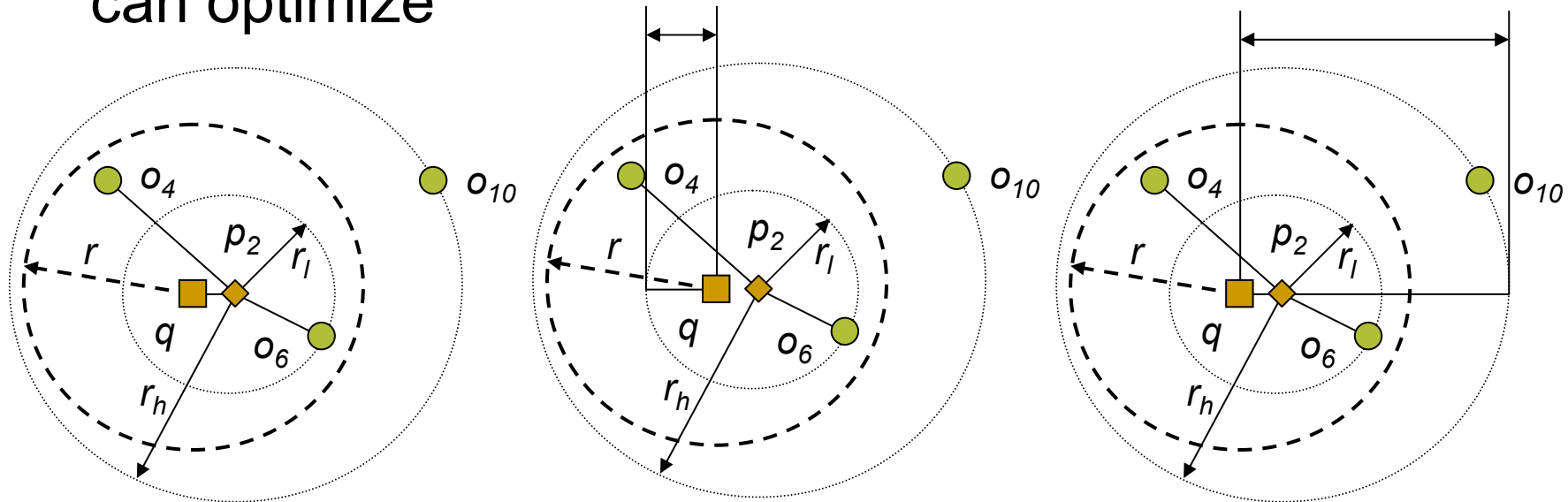
Range-Pivot Distance Constraint

- Some structures do not store all distances between database objects o_i and a pivot p
 - A range $[r_l, r_h]$ of distances between p and all o_i is stored
- Assume the left-most leaf is to be entered
 - Using the range of distances to leaf objects, we can decide whether to enter or not



Range-Pivot Constraint (cont.)

- Knowing interval $[r_l, r_h]$ of distance in the leaf, we can optimize

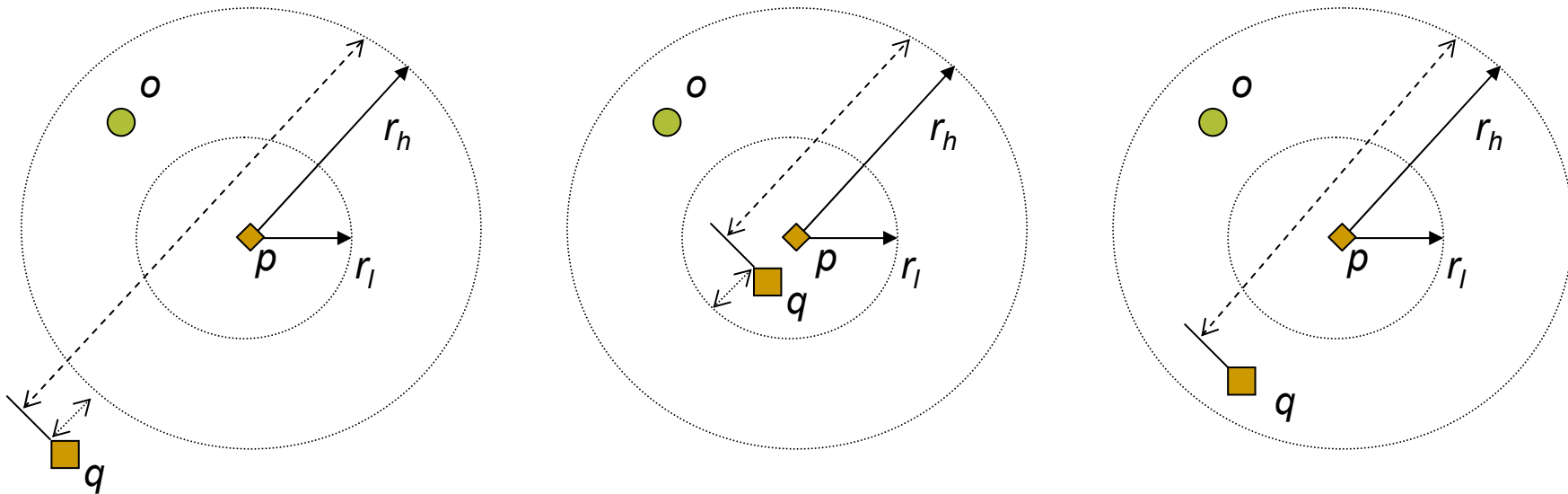


- Lower bound is $r_l - d(q, p_2)$
 - If greater than the query radius r , no object can qualify.
- Upper bound is $r_h + d(q, p_2)$
 - If less than the query radius r , all objects qualify!

Range-Pivot Constraint (cont.)



- We have considered one position of q
- Three are possible:



$\langle \dots \rangle$ upper bound

$\langle \dots \rangle$ lower bound

Range-Pivot Constraint (summary)

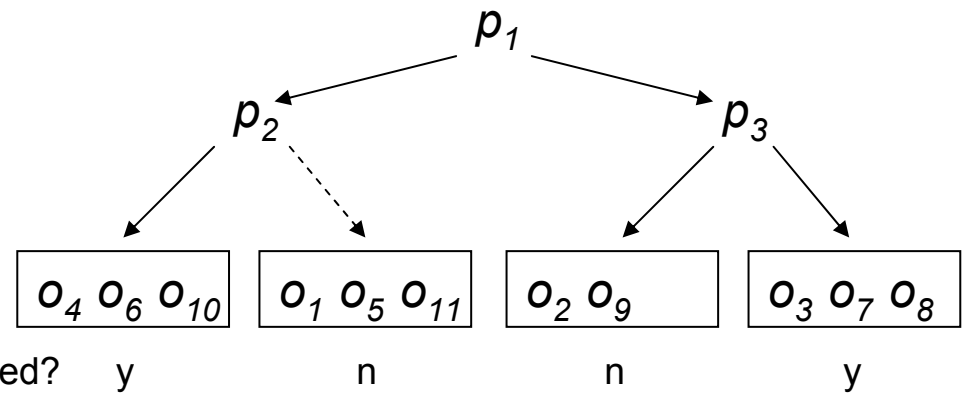
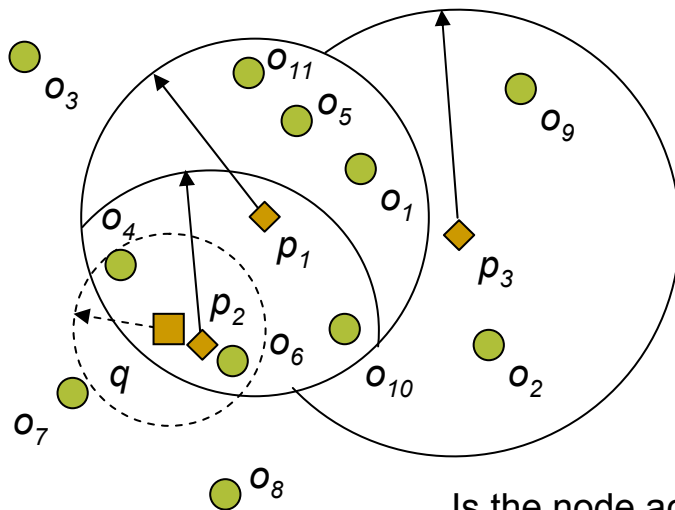


- Given a metric space $\mathcal{M}=(\mathcal{D},d)$ and objects $p,o\in\mathcal{D}$ such that $r_l \leq d(o,p) \leq r_h$. Given $q\in\mathcal{D}$ with known $d(q,p)$. The distance $d(q,o)$ is restricted by:

$$\max\{d(q,p) - r_h, r_l - d(q,p), 0\} \leq d(q,o) \leq d(q,p) + r_h$$

Explanatory Example & Range-Pivot Constraint

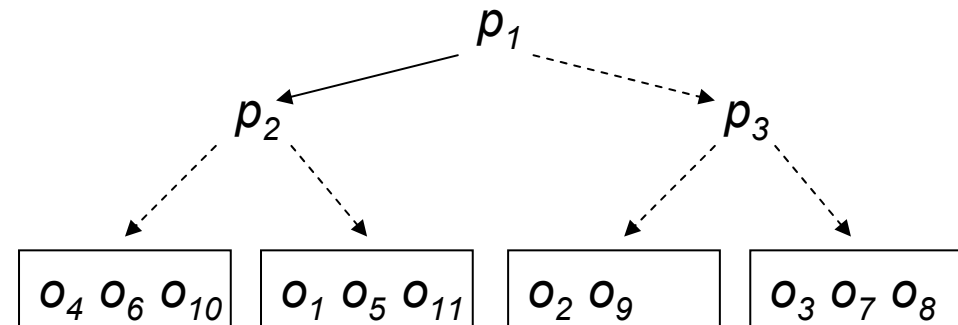
- Range query $R(q,r) = \{o_4, o_6\}$
 - Only *range-pivot*: 3+6 distance computations
3+2 node accesses
 - The leaf with o_2, o_9 is pruned!
 - *object-pivot + range-pivot*: 3+2 distance computations
3+2 node accesses



Other Constraints

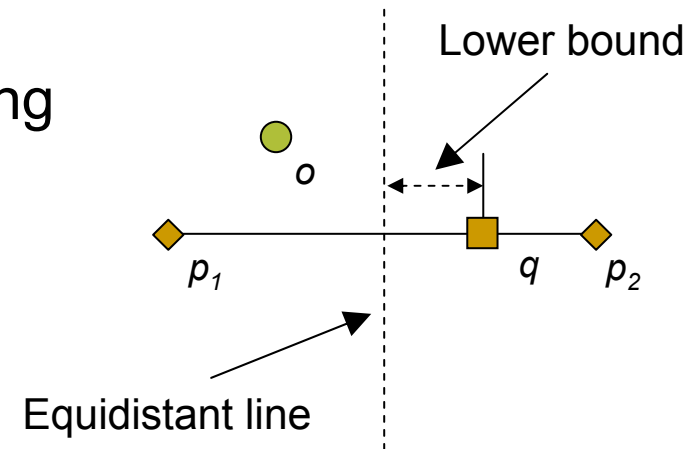
■ Pivot-pivot

- Applied in internal nodes
- Estimate $d(q, p_2)$ using
 - $d(q, p_1), d(p_1, p_2)$
- $d(o, p_2) \in [r_l, r_h]$



■ Double-pivot

- Applied when hyper-plane partitioning is used
- No upper bound can be defined.
- Lower bound is $(d(q, p_1) - d(q, p_2))/2$



Pivot-Pivot Constraint (summary)



- Given a metric space $\mathcal{M}=(\mathcal{D},d)$ and objects $q,p,o\in\mathcal{D}$ such that $r_l \leq d(o,p) \leq r_h$ and $r_l' \leq d(q,p) \leq r_h'$. The distance $d(q,o)$ can be restricted by:

$$\max\{r_l' - r_h, r_l - r_h', 0\} \leq d(q,o) \leq r_h' + r_h$$

Double-Pivot Constraint (summary)



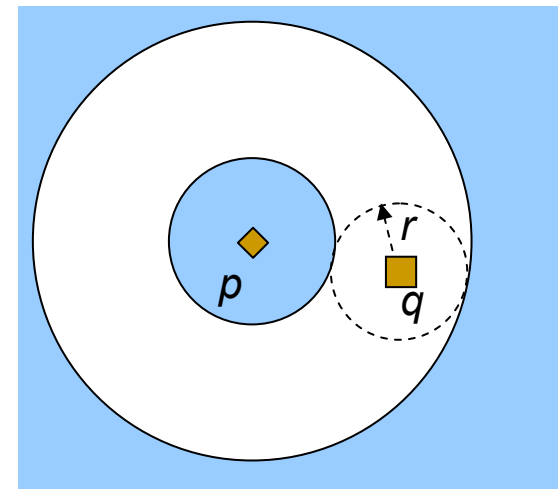
Extra

- Given a metric space $\mathcal{M}=(\mathcal{D},d)$ and objects $o,p_1,p_2\in\mathcal{D}$ such that $d(o,p_1)\leq d(o,p_2)$. Given a query object $q\in\mathcal{D}$ with $d(q,p_1)$ and $d(q,p_2)$. The distance $d(q,o)$ can be lower-bounded by:

$$\max\left\{\frac{d(q,p_1)-d(q,p_2)}{2}, 0\right\}\leq d(q,o)$$

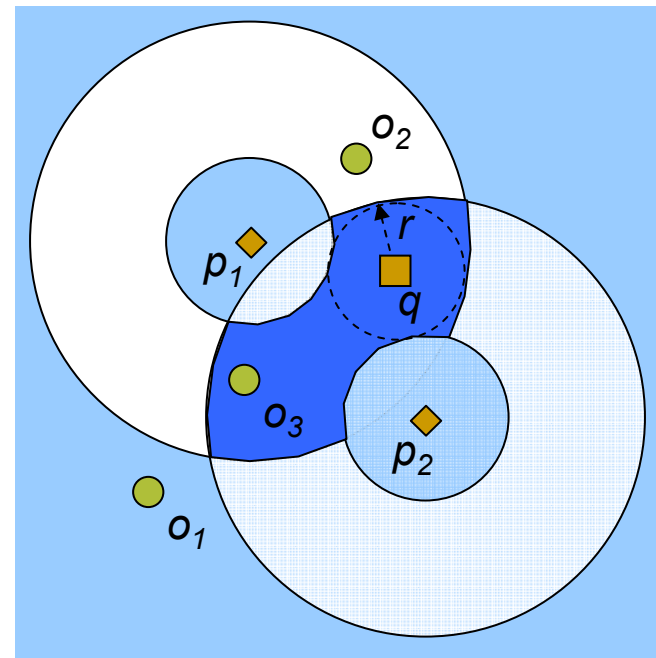
Pivot Filtering [Nav02]

- Extended object-pivot constraint
 - Uses more pivots
- Uses triangle inequality for pruning
- All distances between objects and a pivot p are known.
- Prune the object $o \in X$ if any holds
 - $d(p,o) < d(p,q) - r$
 - $d(p,o) > d(p,q) + r$



Pivot Filtering (cont.)

- Filtering with two pivots
 - Only objects in the dark blue region have to be checked.
 - Effectiveness is improved using more pivots.



Pivot Filtering (summary)



- Given a metric space $\mathcal{M}=(\mathcal{D},d)$ and a set of pivots $P = \{ p_1, p_2, p_3, \dots, p_n \}$. We define a mapping function $\Psi: (\mathcal{D},d) \rightarrow (\mathbb{R}^n, L_\infty)$ as follows:

$$\Psi(o) = (d(o,p_1), \dots, d(o,p_n))$$

Then, we can bound the distance $d(q,o)$ from below:

$$L_\infty(\Psi(o), \Psi(q)) \leq d(q,o)$$

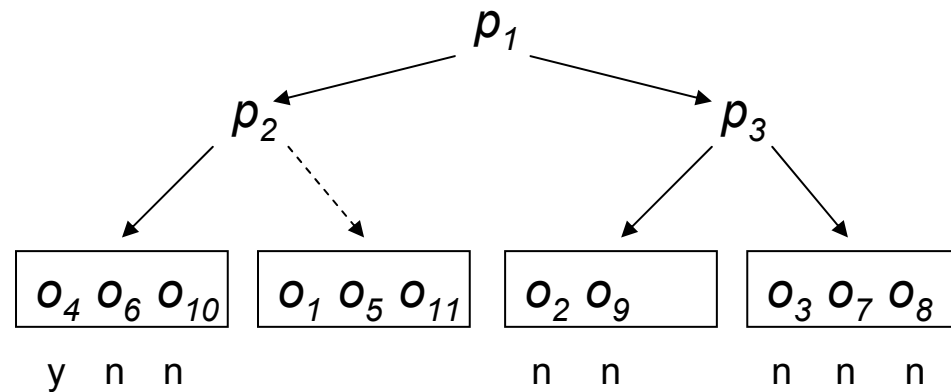
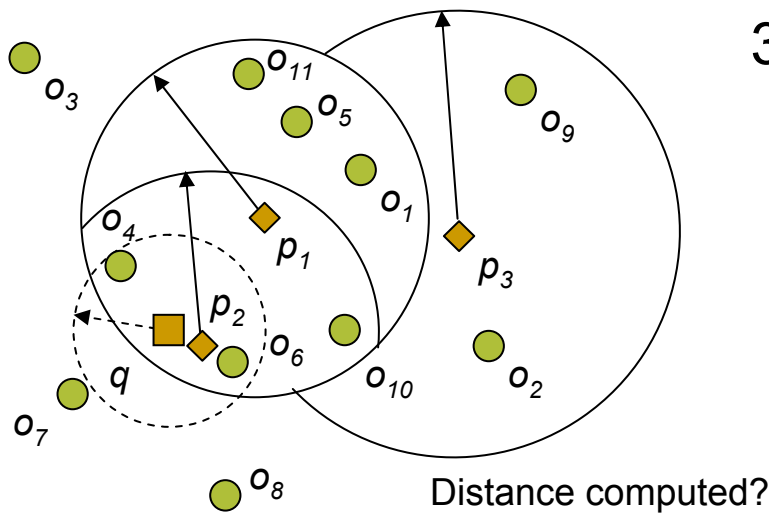
Pivot Filtering (consideration)



- Given a range query $R(q,r)$
 - We want to report all objects o such that $d(q,o) \leq r$
- Apply the pivot filtering
- We can discard objects for which
 - $L_\infty(\Psi(o), \Psi(q)) > r$ holds, i.e. the lower bound on $d(q,o)$ is greater than r .
- The mapping Ψ is contractive:
 - No eliminated object can qualify.
 - Some qualifying objects need not be relevant.
 - These objects have to be checked against the original function $d()$.

Explanatory Example & Pivot Filtering


- Range query $R(q,r) = \{o_4, o_6\}$
 - Objects know distances to pivots along paths to the root.
 - Only *pivot filtering*: 3+1 distance computations (to o_4)
3+3 node accesses
 - All constraints together: 3+1 distance computations
3+2 node accesses



Outline of the talk

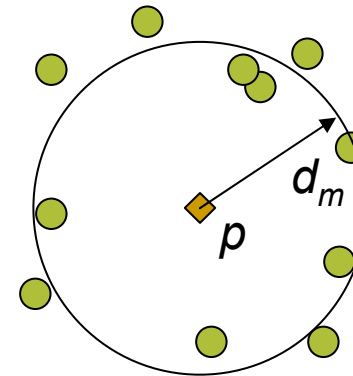
1. similarity, metric space and distance measures (15')
2. similarity queries, metric partitioning principles (15')
3. query execution strategies (15')
4. avoiding distance computations (15')
5. **pivot choosing and metric transformations (15')**
6. short survey of metric space indexes (15')
7. M-tree family, D-index - performance evaluation (30')
8. approximate similarity search (30')
9. scalable and distributed indexes (30')

Choosing Pivots [BNC01]

- All but naïve index structures need pivots (reference objects).
 - Pivots are essential for partitioning and search pruning.
 - Pivots influence performance:
 - The higher & more narrowly-focused distance density with respect to a pivot
- 
- The greater change for a query object to be located at the most frequent distance from the pivot.

Choosing Pivots: Example

- Pivots influence performance:
 - Consider ball partitioning:
 - The distance d_m is the most frequent.



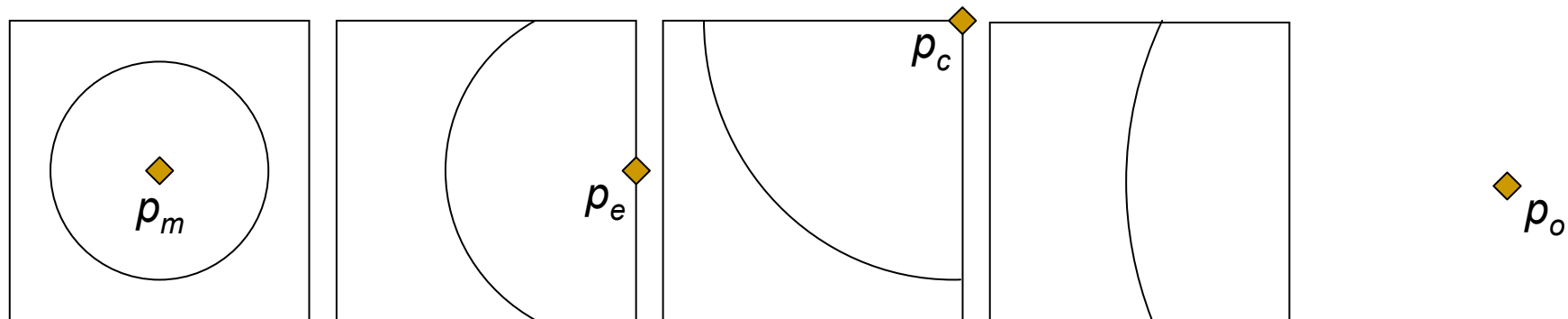
- If all other distance are not very different



- Both subsets are very likely to be accessed by any query.

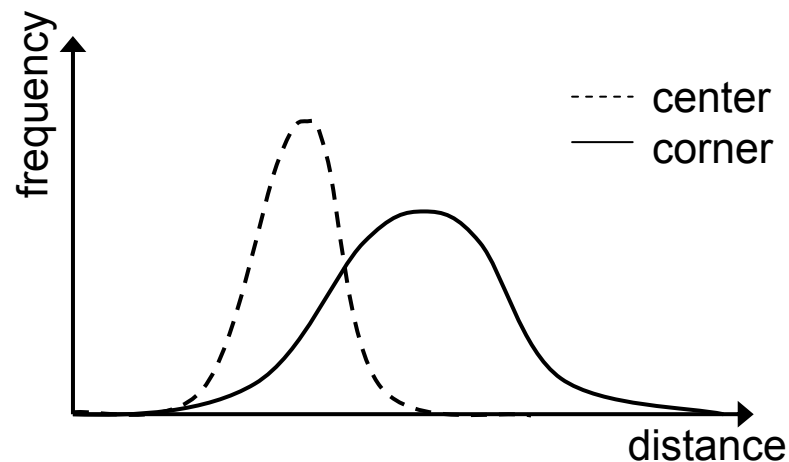
Choosing Pivots: Example 2

- Position of a “good” pivot:
 - A unit square with uniform distribution
- The shortest boundary has the pivot p_o outside the data space. [Yia93]



Choosing Pivots: Example 3

- Different view on a “good” pivot: [BÖ99]
 - 20-D Euclidean space
 - Density with respect to a corner pivot is flatter.
 - Density with respect to a central pivot is sharper & thinner.



Choosing Good Pivots

- Good pivots should be *outliers* of the space
 - i.e. an object located far from the others
 - or an object near the boundary of the space.
- Selecting good pivots is difficult
 - Square or cubic complexities are common.
 - Often chosen at random.
 - Even being the most trivial and not optimizing, many implementations use it!

Choosing Pivots: Heuristics

- There is no definition of a corner in metric spaces.
- A corner object is 'far away' from others.

- Algorithm for an outlier: [BÖ99]
 1. Choose a random object
 2. Compute distances from this object to all others
 3. Pick the furthest object as the pivot
- This does not guarantee the best possible pivot.
 - Helps to choose a better pivot than the random choice.
 - Brings 5-10% performance gain

Choosing More Pivots



- The problem of selecting more pivots is more complicated - pivots should be fairly far apart.
- Algorithm for choosing m pivots: [Bri95]
 - Choose $3m$ objects at random from the given set of n objects.
 - Pick an object. The furthest object from this is the first pivot.
 - Second pivot is the furthest object from the first pivot.
 - The third pivot is the furthest object from the previous pivots. The minimum $\min(d(p_1, p_3), d(p_2, p_3))$ is maximized.
 - ...
 - Until we have m pivots.

Choosing Reference Points: Summary



- Current rules are:
 - Good pivots are *far away* from other objects in the metric space.
 - Good pivots are *far away* from each other.
 - The best pivot is the query object itself.

- Heuristics sometimes fail:
 - A dataset with Jaccard's coefficient
 - The outlier principle might select pivot p such that $d(p,o)=1$ for any other database object o .
 - Such pivot is useless for partitioning & filtering!

Metric Space Transformations

- Change one metric space into another space
 - Transformation of the original objects
 - Changing the metric function
 - Transforming both the function and the objects

Purpose:

- User-defined search functions
- Metric space embedding
 - A cheaper distance function
 - Spatial indexes can be used (R-trees, ...).

Metric Space Transformation

- $\mathcal{M}_1 = (\mathcal{D}_1, d_1) \longrightarrow \mathcal{M}_2 = (\mathcal{D}_2, d_2)$

- Function $f : \mathcal{D}_1 \rightarrow \mathcal{D}_2$

$$\forall o, o' \in \mathcal{D}_1 : d_1(o, o') \approx d_2(f(o), f(o'))$$

- Transformed distances do not need to be equal.

Lower Bounding Metric Functions [CP02]

- Bounds on the transformation
- Exploitable by index structures

- Having functions $d_1, d_2: \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$
- d_1 is a *lower-bounding* distance function of d_2

$$\forall o, o' \in \mathcal{D} : d_1(o, o') \leq d_2(o, o')$$

- Any L_p metric is lower-bounding for L_p if $p \leq p'$
 - L_1 is never smaller than L_2

User-defined Metric Functions

- Different users have different preferences.
 - Some people prefer car's performance to its color.
 - Others prefer lower prices.
 - etc...
- Preferences might be complex [Cho02].
 - Color histograms
 - Users may not be able to form their preferences.
 - data-mining systems
 - Can be learnt automatically [CFG00,OCM02]
 - from the previous behavior of a user

User-defined Metric Functions

- Preferences expressed as another *distance function* d_u [CP02]
 - Can be different for different users
 - Example: matrices for quadratic form distance functions
- Database indexed with a fixed metric d_b
- Lower-bounding metric function d_p
 - d_p lower-bounds d_b and d_u
 - It is applied during the search.
 - An index structure can be exploited.

User-defined Metric Functions

- Searching using d_p
 - Search the index, but use d_p instead of d_b

- Possible, because

$$\forall o_1, o_2 \in \mathcal{D} : d_p(o_1, o_2) \leq d_b(o_1, o_2)$$

- Every object that would match a similarity query using d_b will certainly match with d_p .

- False-positives in the result

- Filtered afterwards - using d_u
- Possible, because

$$\forall o_1, o_2 \in \mathcal{D} : d_p(o_1, o_2) \leq d_u(o_1, o_2)$$

Embedding the Metric Space

- Transform the metric space
 - Cheaper metric function d_2
 - Approximate the original distance d_1

$$d_1(o_1, o_2) \geq d_2(f(o_1), f(o_2))$$

- Drawbacks
 - Must transform objects using the function f
 - False-positives
 - Resolved by using the original metric function d_1

Embedding Examples

- Lipschitz Embedding [Bou85]
 - Mapping to an n -dimensional vector space
 - Coordinates correspond to chosen subsets S_i of objects
 - An object is then a vector of distances to the closest object from a particular coordinate set S_i

$$f(o) = (d(o, S_1), d(o, S_2), \dots, d(o, S_n))$$

- Transformation is very expensive
 - SparseMap extension reduces its costs [HS03b].

Embedding Examples (cont.)

- Karhunen-Loeve Transformation [Fuk90]
 - Linear transformation of vector spaces
 - Dimensionality reduction technique
 - Similar to Principal Component Analysis [Dun89]
 - Projects an object o onto the first $k < n$ basis vectors

$$V = \{ \vec{v}_1, \vec{v}_2, \dots, \vec{v}_n \}$$

- Transformation is contractive
 - Used in the FastMap technique [FL95]
- MetricMap [WWL⁺00]
 - Designed for generic metric spaces

Outline of the talk

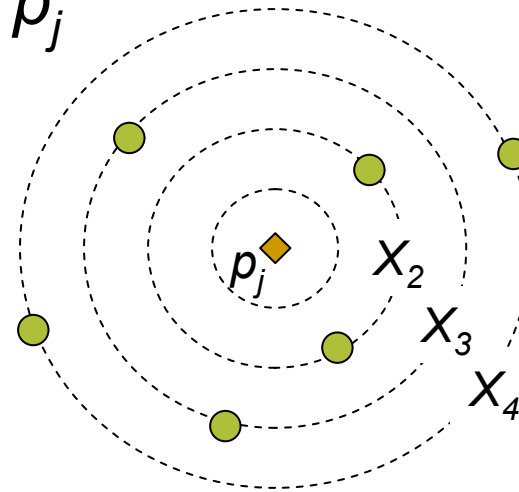
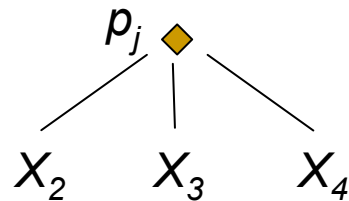
1. similarity, metric space and distance measures (15')
2. similarity queries, metric partitioning principles (15')
3. query execution strategies (15')
4. avoiding distance computations (15')
5. pivot choosing and metric transformations (15')
6. **short survey of metric space indexes (15')**
7. M-tree family, D-index - performance evaluation (30')
8. approximate similarity search (30')
9. scalable and distributed indexes (30')

Metric Index Structures [CNBM01,HS03a]

- Ball-partitioning methods
 - Fixed Queries Tree, Vantage Point Tree
- Hyper-plane partitioning methods
 - Generalized Hyper-plane tree
- Precomputed distances
 - AESA, Spaghettis
- Hybrid methods
 - Multi Vantage Point Tree, GNAT, Spatial Approximation Tree
- Others
 - M-tree, D-Index

Burkhard-Keller Tree (BKT) [BK73]

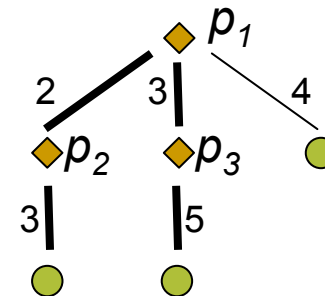
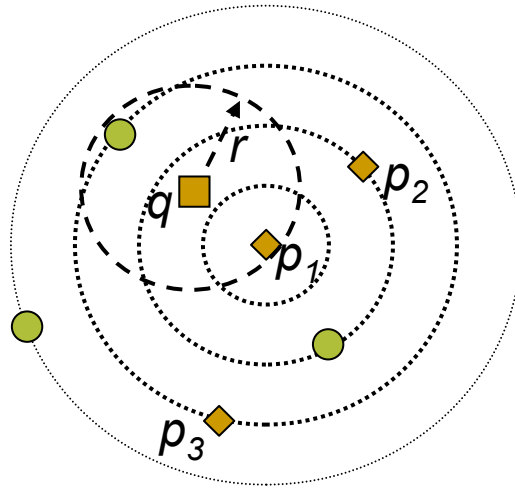
- Applicable to discrete distance functions only
- Recursively divides a given dataset X
- Choose an arbitrary point $p_j \in X$, form subsets:
$$X_i = \{o \in X, d(o, p_j) = i\} \quad \text{for each distance } i \geq 0.$$
- For each X_i create a sub-tree of p_j
 - Empty subsets are ignored.



BKT: Range Query

Given a query $R(q,r)$:

- Traverse the tree starting from root
- In each internal node p_j , do:
 - Report p_j on output if $d(q,p_j) \leq r$
 - Enter a child i if $\max\{d(q,p_j) - r, 0\} \leq i \leq d(q,p_j) + r$

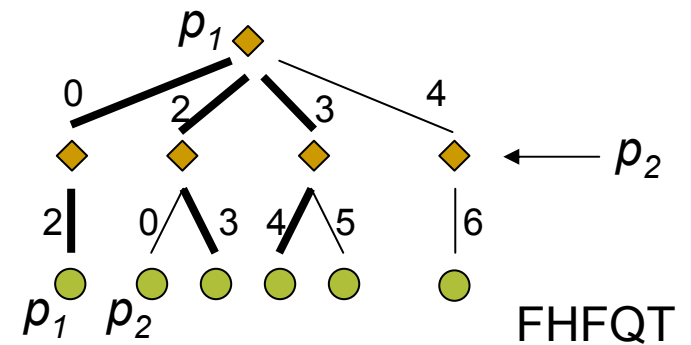
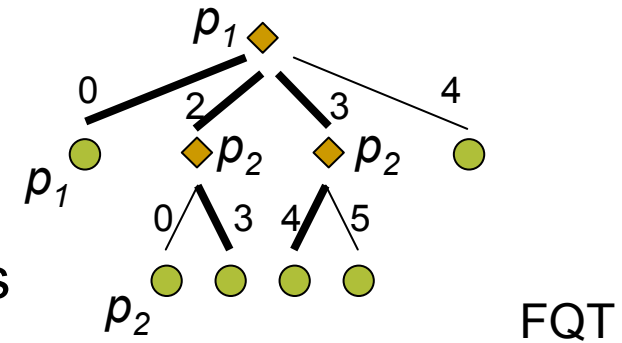


Fixed Queries Tree/Array

- Fixed Queries Tree [BCMW94]
 - Each level has a single pivot.
 - All objects stored in leaves.
 - During search, distance computations are saved.

- Fixed-Height FQT [Bae97]
 - All leaf nodes at the same level
 - Increased filtering using more routing objects (pivots)

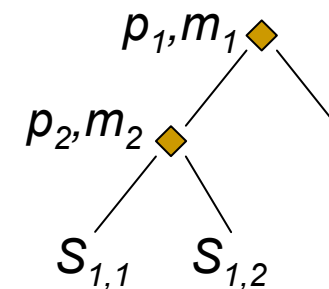
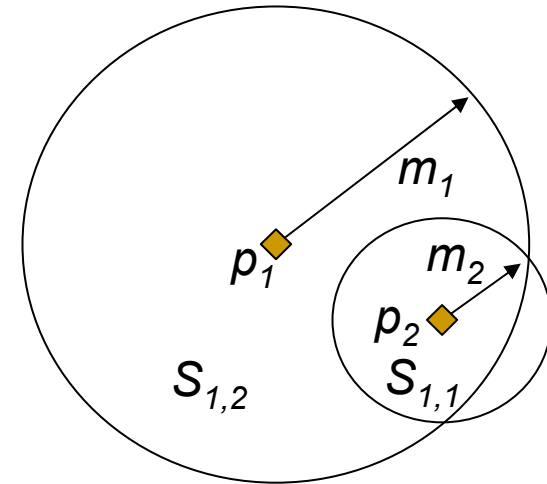
- Fixed Queries Array [CMN01,CMN99]
 - Tree transformed to an array of paths



0	2	2	3	3	4	← p_1
2	0	3	4	5	6	← p_2

Vantage Point Tree (VPT) [Yia93]

- Uses ball partitioning
 - Recursively divides given data set X
- Choose a vantage point $p \in X$, compute the median m
 - $S_1 = \{x \in X - \{p\} \mid d(x, p) \leq m\}$
 - $S_2 = \{x \in X - \{p\} \mid d(x, p) \geq m\}$
 - The equality sign ensures balancing.
- The pivots p_1, p_2 belong to the dataset.
- More than one object can be accommodated in leaves.

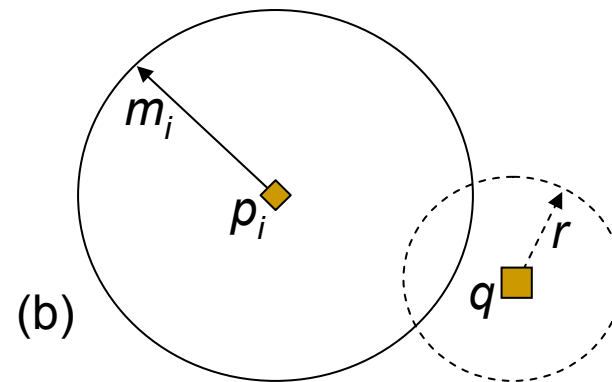
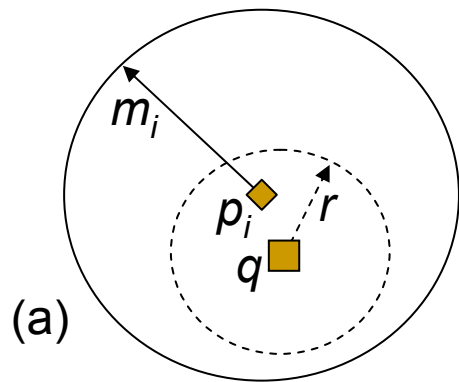


VPT: Range Search



Given a query $R(q,r)$:

- Traverse the tree starting from its root
- In each internal node (p_i, m_i) , do:
 - if $d(q, p_i) \leq r$ report p_i on output
 - if $d(q, p_i) - r \leq m_i$ search the left sub-tree (a,b)
 - if $d(q, p_i) + r \geq m_i$ search the right sub-tree (b)



VPT: k -NN Search

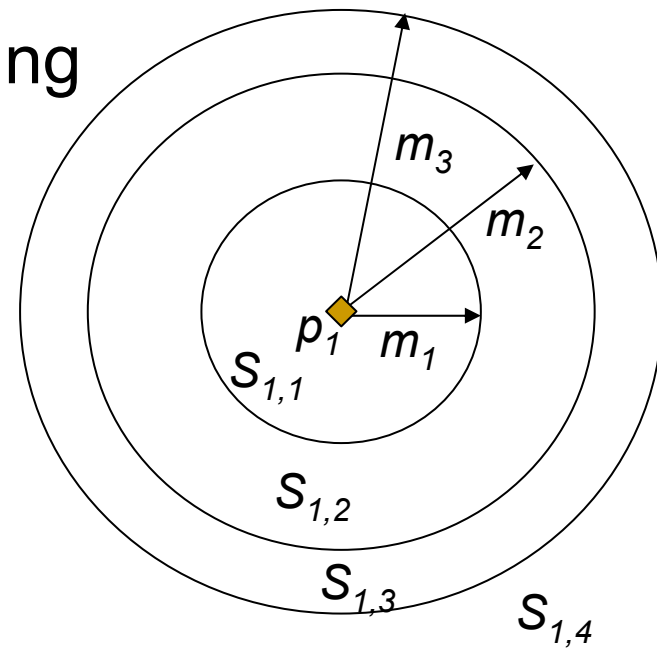
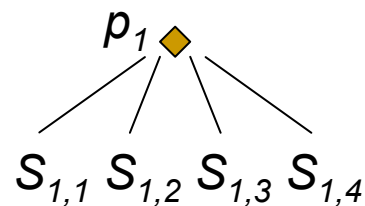


Given a query $NN(q)$:

- Initialization: $d_{NN} = d_{max}$ $NN = nil$
- Traverse the tree starting from its root
- In each internal node (p_i, m_i) , do:
 - if $d(q, p_i) \leq d_{NN}$ set $d_{NN} = d(q, p_i)$, $NN = p_i$
 - if $d(q, p_i) - d_{NN} \leq m_i$ search the left sub-tree
 - if $d(q, p_i) + d_{NN} \geq m_i$ search the right sub-tree
- k -NN search only requires the arrays $d_{NN}[k]$ and $NN[k]$.
 - The arrays are kept ordered with respect to the distance to q .

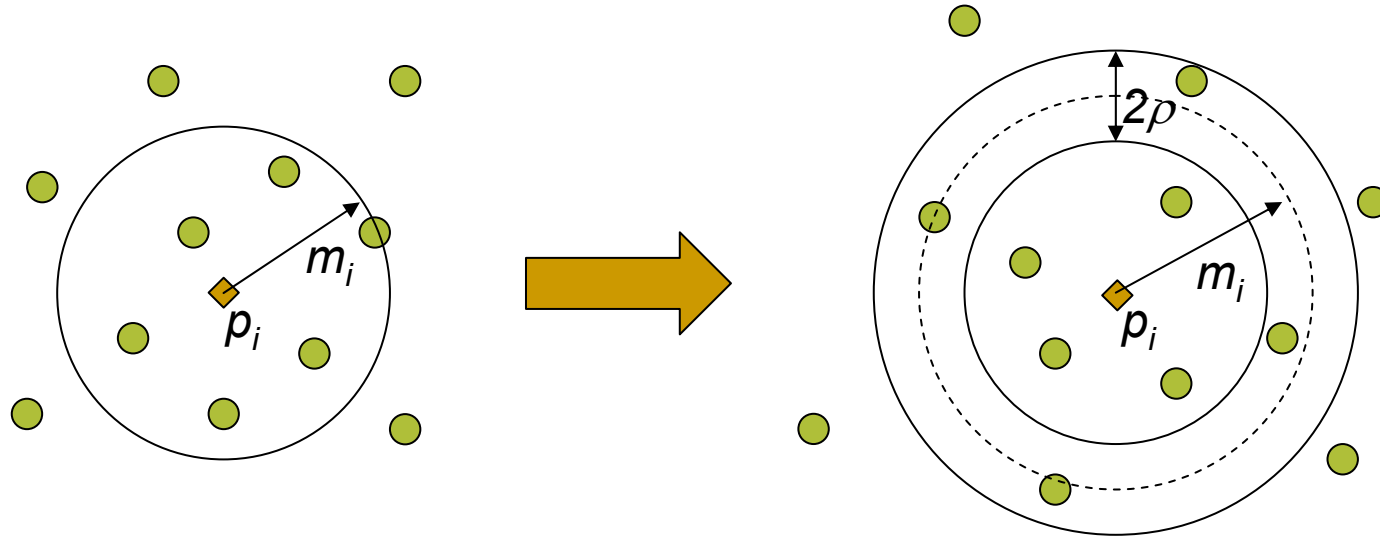
Multi-Way Vantage Point Tree [BÖ97]

- Inherits all principles from VPT
 - But partitioning is modified to m -ary
- m -ary balanced tree
- Applies multi-way ball partitioning



Vantage Point Forest (VPF) [Yia99]

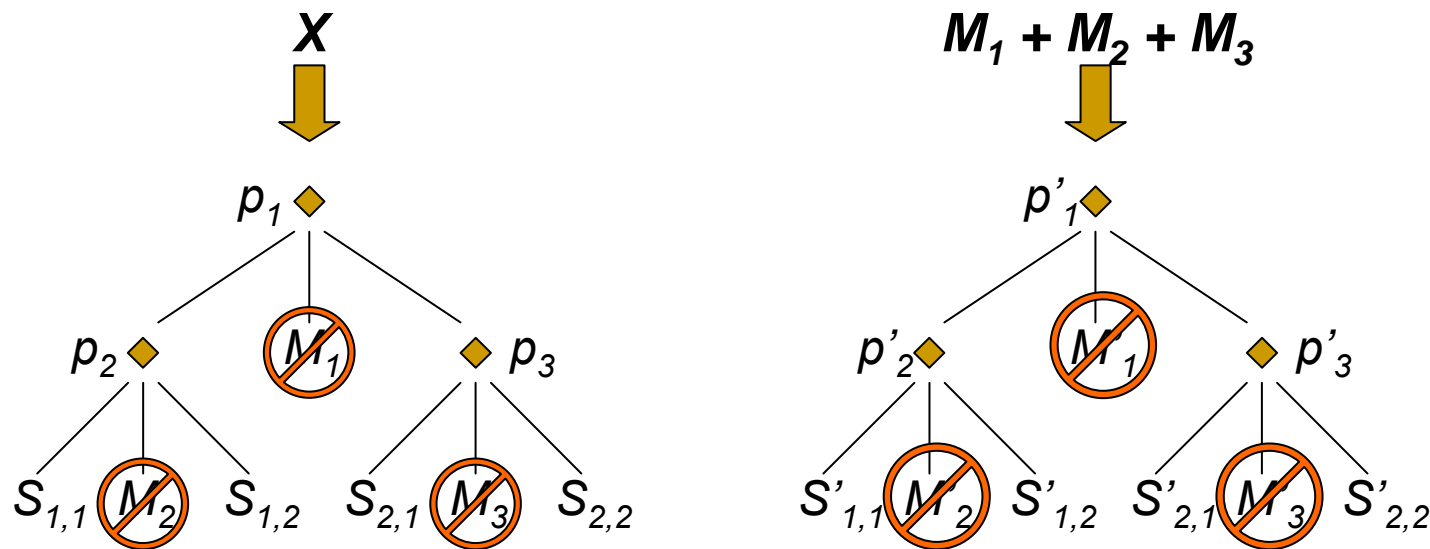
- A forest of binary trees
- Uses excluded middle partitioning



- Middle area is excluded from the process of tree building.

VPF (cont.)

- The given data set X is recursively divided and a binary tree is built.
- The excluded middle areas are used for building another binary tree.



Bisector Tree (BT) [KM83]

- Apply generalized hyper-plane partitioning:

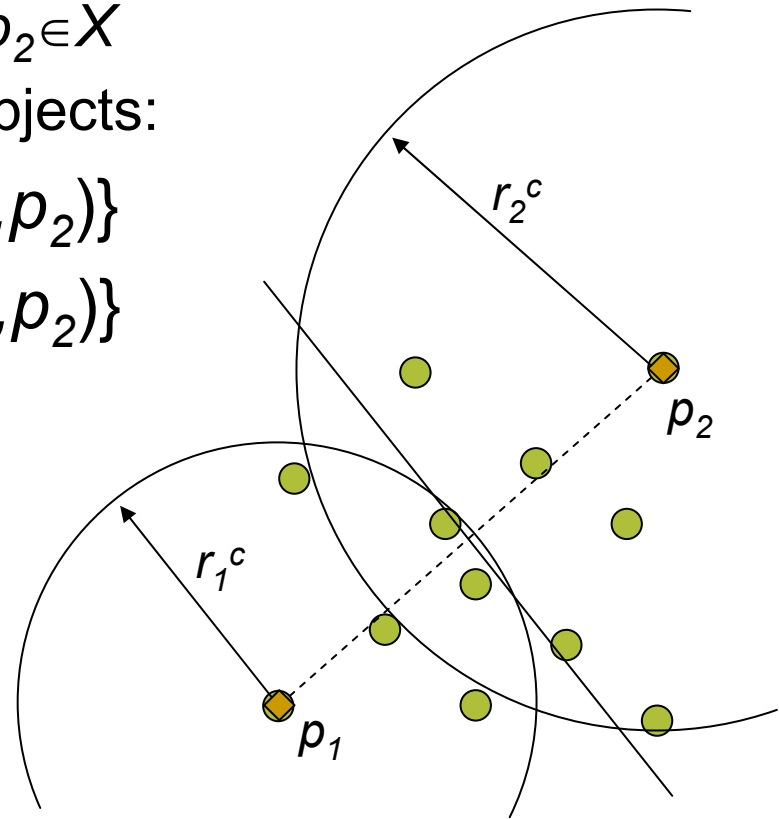
- Choose two arbitrary points $p_1, p_2 \in X$
- Form subsets from remaining objects:

$$S_1 = \{o \in X, d(o, p_1) \leq d(o, p_2)\}$$

$$S_2 = \{o \in X, d(o, p_1) > d(o, p_2)\}$$

- Covering radii r_1^c and r_2^c are established:

- The balls can intersect!

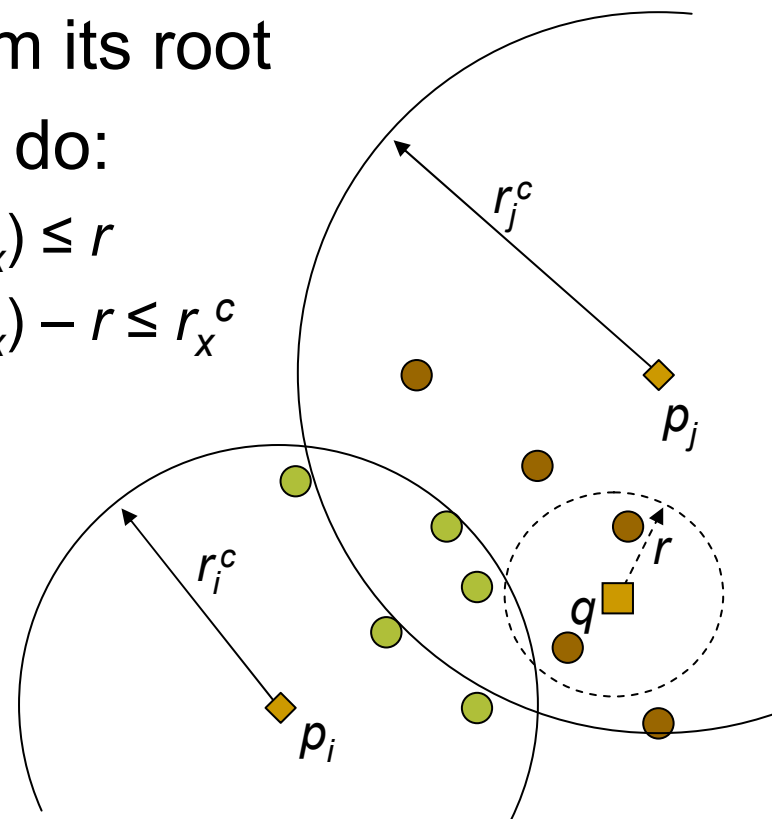
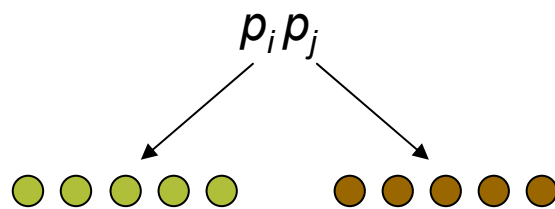


BT: Range Query



Given a query $R(q,r)$:

- Traverse the tree starting from its root
- In each internal node $\langle p_i, p_j \rangle$, do:
 - Report p_x on output if $d(q, p_x) \leq r$
 - Enter a child of p_x if $d(q, p_x) - r \leq r_x^c$



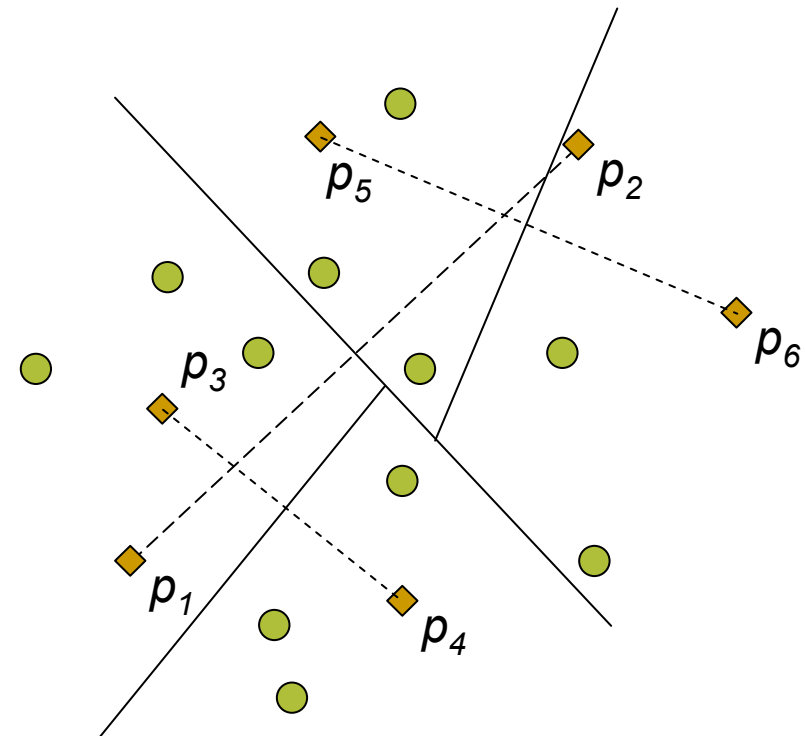
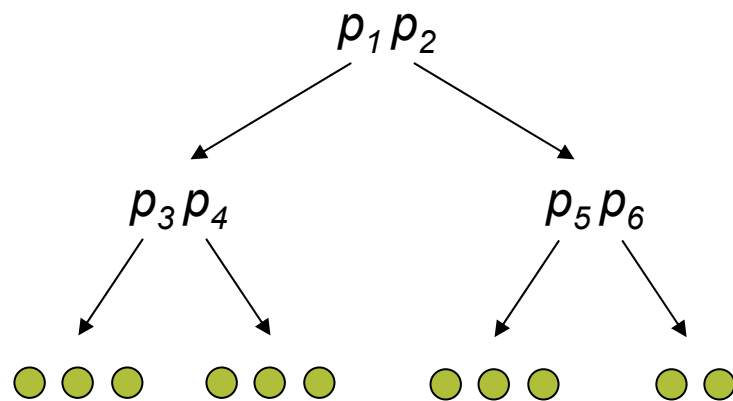
Extensions of BT

- Monotonous BT [NVZ92b,NVZ92a]
 - One pivot from a parent node is inherited to child nodes.
- Voronoi Tree [DN87,Nol89]
 - More pivots in each internal node (usually three)

Generalized Hyper-plane Tree (GHT)

[Uh191]

- Similar to Bisector Trees
- Covering radii are not used



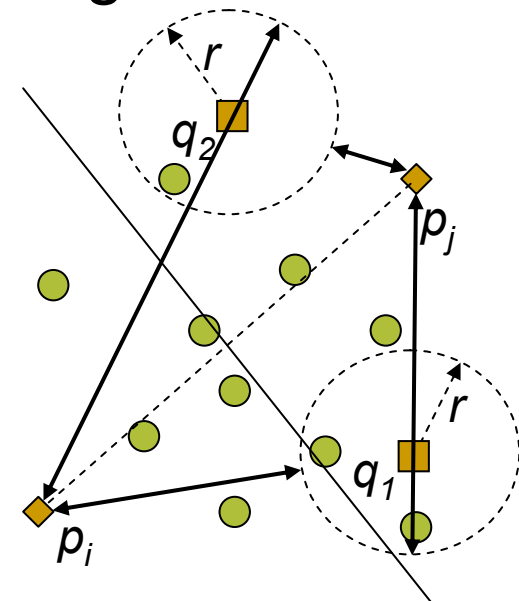
GHT: Range Query



- Pruning based on hyper-plane partitioning

Given a query $R(q, r)$:

- Traverse the tree starting from its root
- In each internal node $\langle p_i, p_j \rangle$, do:
 - Report p_x on output if $d(q, p_x) \leq r$
 - Enter the left child if $d(q, p_i) - r \leq d(q, p_j) + r$
 - Enter the right child if $d(q, p_i) + r \geq d(q, p_j) - r$

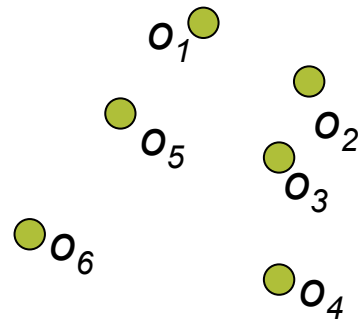


Exploiting Pre-computed Distances

- During insertion of an object into a structure, some distances are evaluated.
- If they are remembered, we can exploit them in filtering when processing a query.

AESA [Vid86,Vid94]

- Approximating and Eliminating Search Algorithm
- Matrix $n \times n$ of distances
 - Due to the symmetry, only a half $(n(n-1)/2)$ is stored.



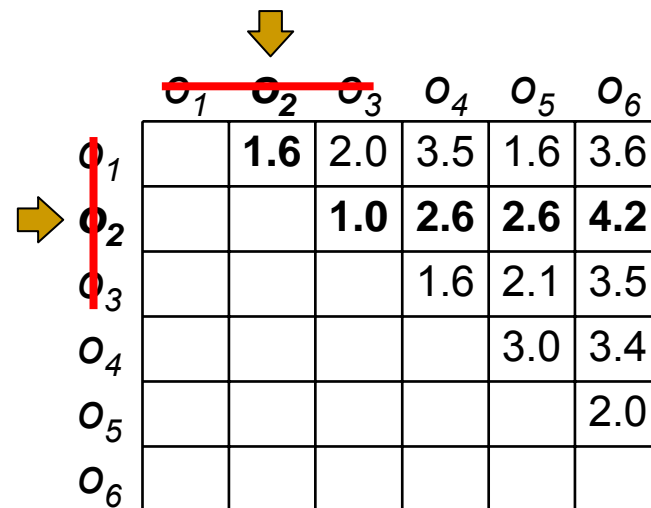
	o_1	o_2	o_3	o_4	o_5	o_6
o_1	0	1.6	2.0	3.5	1.6	3.6
o_2	1.6	0	1.0	2.6	2.6	4.2
o_3	2.0	1.0	0	1.6	2.1	3.5
o_4	3.5	2.6	1.6	0	3.0	3.4
o_5	1.6	2.6	2.1	3.0	0	2.0
o_6	3.6	4.2	3.5	3.4	2.0	0

- Every object can play a role of *pivot*.

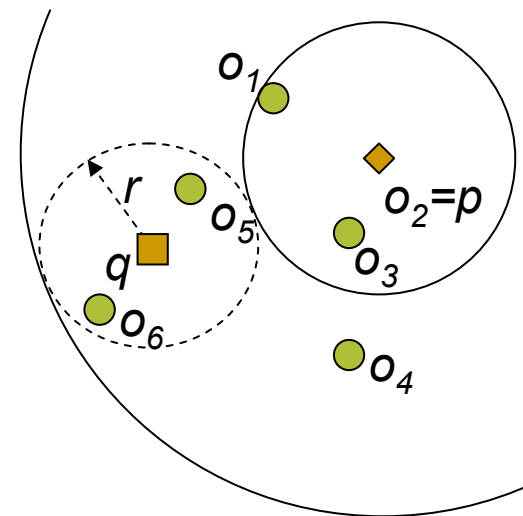
AESA: Range Query

Given a query $R(q,r)$:

- Randomly pick an object and use it as the pivot p
- Compute $d(q,p)$
- Filter out an object o if $|d(q,p) - d(p,o)| > r$

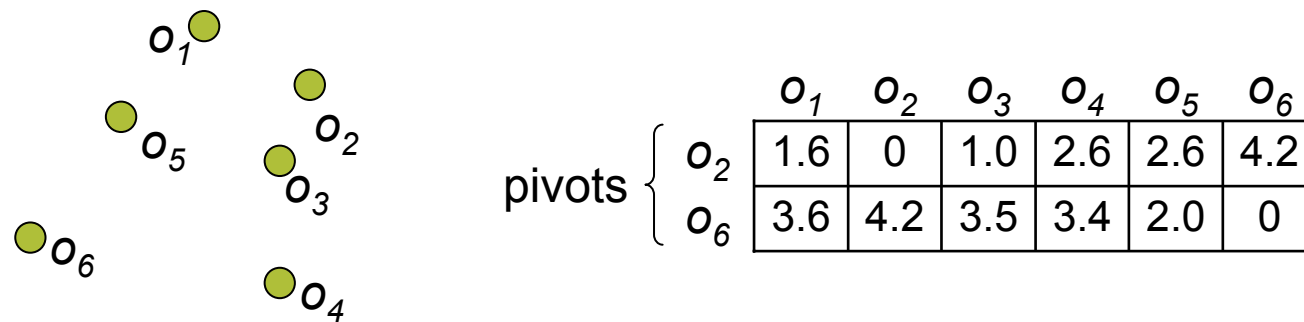


	o_1	o_2	o_3	o_4	o_5	o_6
o_1		1.6	2.0	3.5	1.6	3.6
o_2			1.0	2.6	2.6	4.2
o_3				1.6	2.1	3.5
o_4					3.0	3.4
o_5						2.0
o_6						



Linear AESA (LAESA) [MOV92,MOV94]

- AESA is quadratic in space.
- LAESA stores distances to m pivots only.
- Pivots should be selected conveniently.
 - Pivots as far away from each other as possible are chosen.



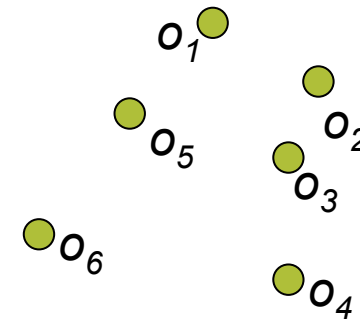
- Search:
 - First, all pivots are used for filtering.
 - Next, remaining objects are directly compared to q .

Extensions of LAESA

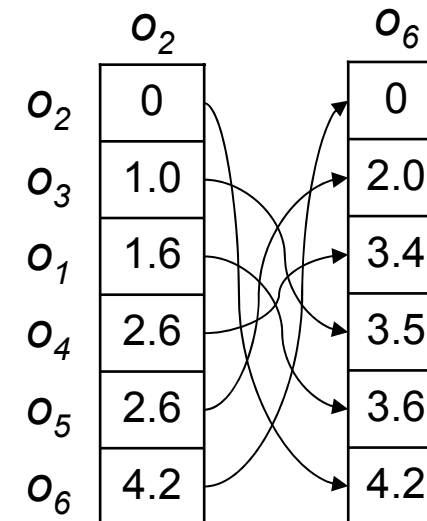
- Shapiro [Sha77]
 - Database objects are sorted with respect to the first pivot.
 - Search: start with the closest o_i and continue to the left and to the right.

- Spaghettis [CMB99]
 - Matrix $m \times n$ is stored in m arrays of length n .
 - Each array is sorted according to the distances in it.
 - Search: intersection of intervals defined on individual arrays.

- Reduced Overhead LAESA [Vil95]



	o_2	o_3	o_1	o_4	o_5	o_6
o_2	0	1.0	1.6	2.6	2.6	4.2
o_6	4.2	3.5	3.6	3.4	2.0	0

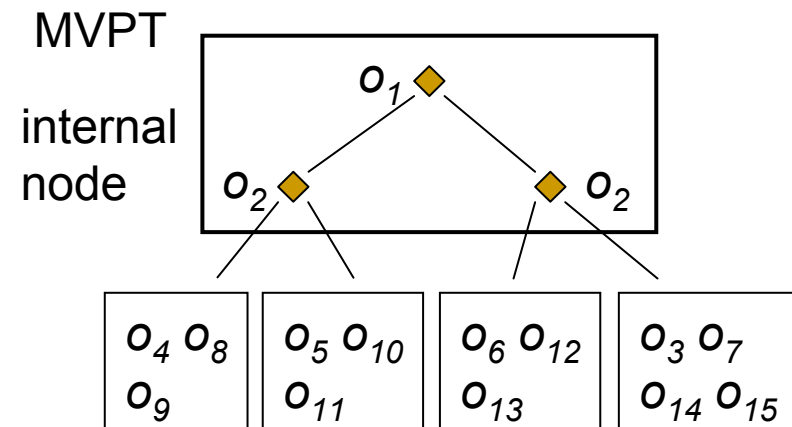
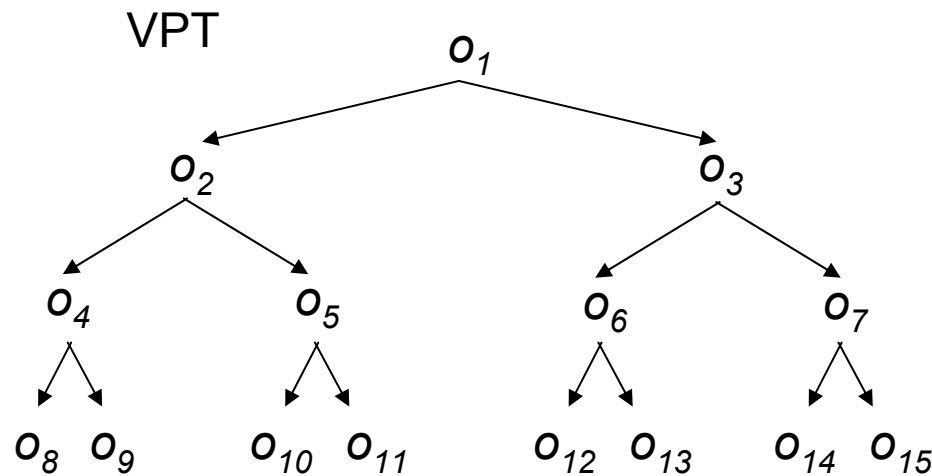


Hybrid approaches

- Structures that store pre-computed distances have high space requirements.
 - But good performance boost during query processing
- Hybrid approaches combine partitioning and pre-computed distances into a single system.
 - Lower space requirements
 - Good query performance

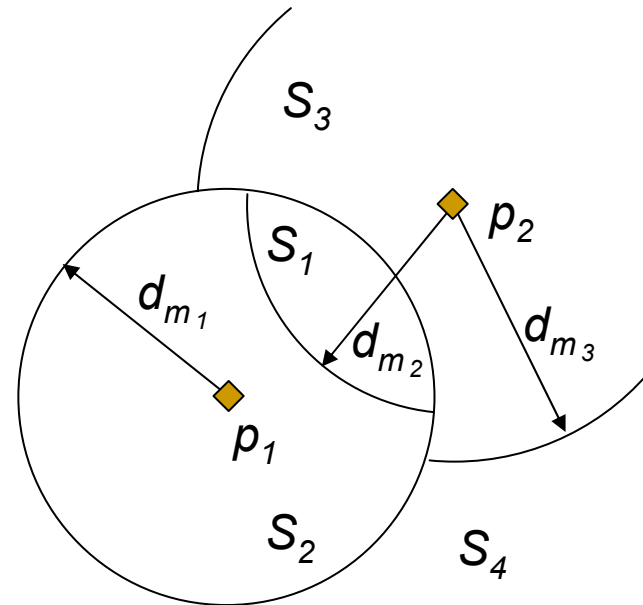
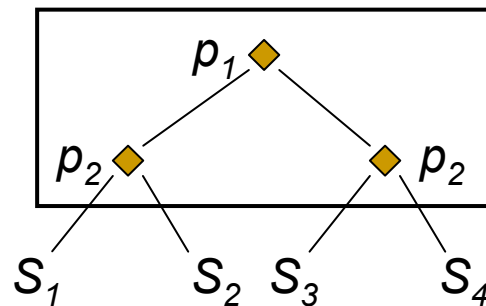
Multi Vantage Point Tree [BÖ97,BÖ99]

- Tries to decrease the number of pivots
 - Two pivots are used in each internal node.
 - Idea: two levels of VPT collapsed into a single node
 - Distances to the first h pivots are stored along every object in leaves.



MPVT: Internal Node

- Ball partitioning is applied
 - Pivot p_2 is shared

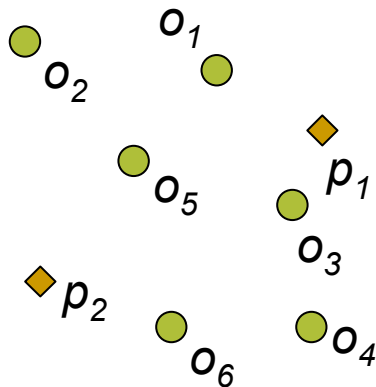


- In general, MVPT can use k pivots in every node.
 - Number of children is 2^k .
 - Multi-way partitioning can be used as well $\rightarrow m^k$ children.

MVPT: Leaf Node



- Leaf node stores two “pivots” as well.
 - The first pivot is selected randomly.
 - The second pivot is picked as the furthest from the first one.
 - The same selection is used in internal nodes.
- Capacity is c objects + 2 pivots.



Distances from objects to the first h pivots on the path from the root

	O_1	O_2	O_3	O_4	O_5	O_6
p_1	1.6	4.1	1.0	2.6	2.6	3.3
p_2	3.6	3.4	3.5	3.4	2.0	2.5
}						

Voronoi-based Approaches

- Geometric Near-neighbor Access Tree (GNAT) [Bri95]
 - Voronoi partitioning using m pivots
 - Every pair <pivot,partition> has a distance range associated.
 - $m \times m$ table of distance ranges
- Spatial Approximation Tree (SAT) [Nav99,Nav02]
 - A graph of relations between Voronoi partitions, i.e., an edge is between directly neighboring partitions.
 - For correctness in metric spaces, this would require to have edges between all pairs of objects in X .
 - SAT approximates such a graph.
- Dynamic SAT [NR02]

Other Hybrids

- M-tree family
 - M-tree [CPZ97]
 - Slim Tree [TTSF00]
 - Pivoting M-tree [Sko04]
 - DBM-tree [VTCT04]
 - M⁺-tree (BM⁺-tree) [ZWYY03, ZWZY05]
 - M²-tree [CP00a]
- D-index [DGSZ03]

Outline of the talk

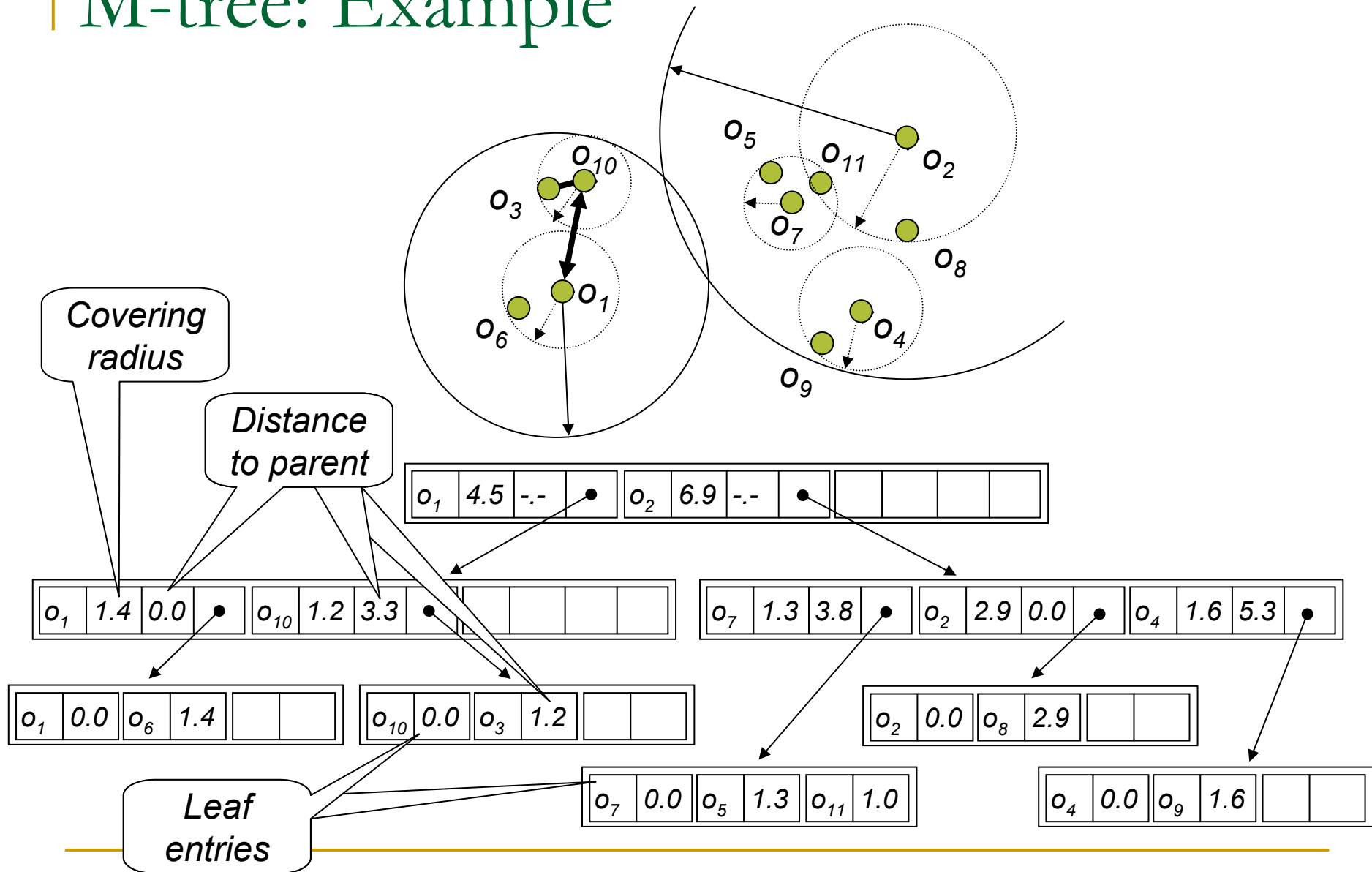
1. similarity, metric space and distance measures (15')
2. similarity queries, metric partitioning principles (15')
3. query execution strategies (15')
4. avoiding distance computations (15')
5. pivot choosing and metric transformations (15')
6. short survey of metric space indexes (15')
7. **M-tree family, D-index - performance evaluation (30')**
8. approximate similarity search (30')
9. scalable and distributed indexes (30')

The M-tree [CPZ97]

- Inherently dynamic structure
- Disk-oriented (fixed-size nodes)
- Built in a bottom-up fashion
 - Inspired by R-trees [Gut84] and B-trees

- All data in *leaf nodes*
- *Internal nodes*: pointers to subtrees and additional information
- Similar to GNAT, but objects are stored in leaves.

M-tree: Example



M-tree: Insert

Insert a new object o_N :

- Recursively descend the tree to locate the *most suitable leaf* for o_N
- In each step, enter the subtree with pivot p for which:
 - No enlargement of radius r^c needed, i.e., $d(o_N, p) \leq r^c$
 - In case of ties, choose one with the closest p to o_N
 - Minimize the enlargement of r^c

M-tree: Insert (cont.)

- When reaching the leaf node N then:
 - if N is not full, then store o_N in N
 - else **Split**(N, o_N).

M-tree: Split

Split(N, o_N):

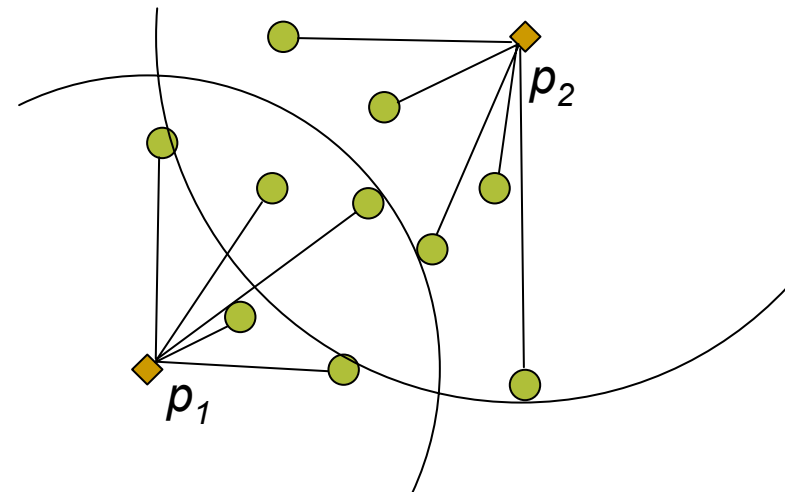
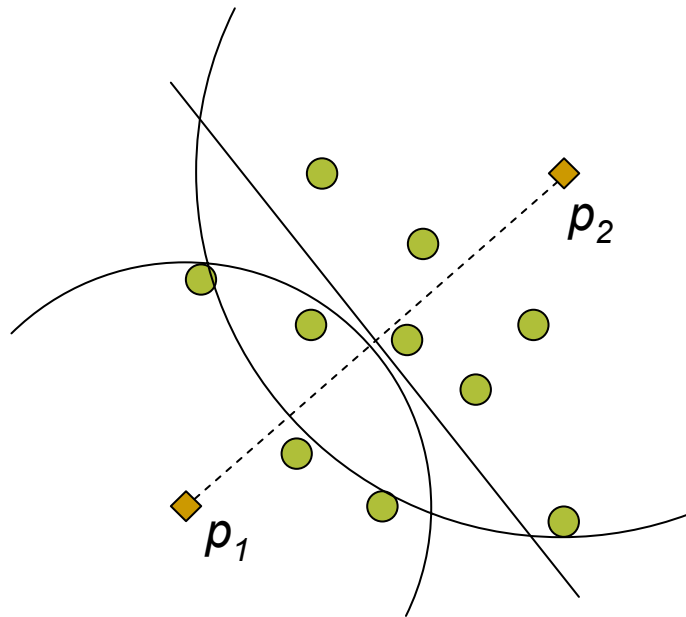
- Let S be the set containing all entries of N and o_N
- Select pivots p_1 and p_2 from S
- Partition S to S_1 and S_2 according to p_1 and p_2
- Store S_1 in N and S_2 in a new allocated node N'
- If N is root
 - Allocate a new root and store entries for p_1, p_2 there
- *else (let N^p and p^p be the parent node and parent pivot of N)*
 - Replace entry p^p with p_1
 - If N^p is full, then **Split**(N^p, p_2)
 - *else store p_2 in node N^p*

M-tree: Pivot Selection

- Several pivots selection policies
 - **RANDOM** – select pivots p_1, p_2 randomly
 - **m_RAD** – select p_1, p_2 with minimum $(r_1^c + r_2^c)$
 - **mM_RAD** – select p_1, p_2 with minimum $\max(r_1^c, r_2^c)$
 - **M_LB_DIST** – let $p_1 = p^p$ and $p_2 = o_i \mid \max \{ d(o_i, p^p) \}$
 - Uses the pre-computed distances only
- Two versions (for most of the policies):
 - **Confirmed** – reuse the original pivot p^p and select only one pivot
 - **Unconfirmed** – select two pivots (notation: **RANDOM_2**)
- In the following, the **mM_RAD_2** policy is used.

M-tree: Split Policy

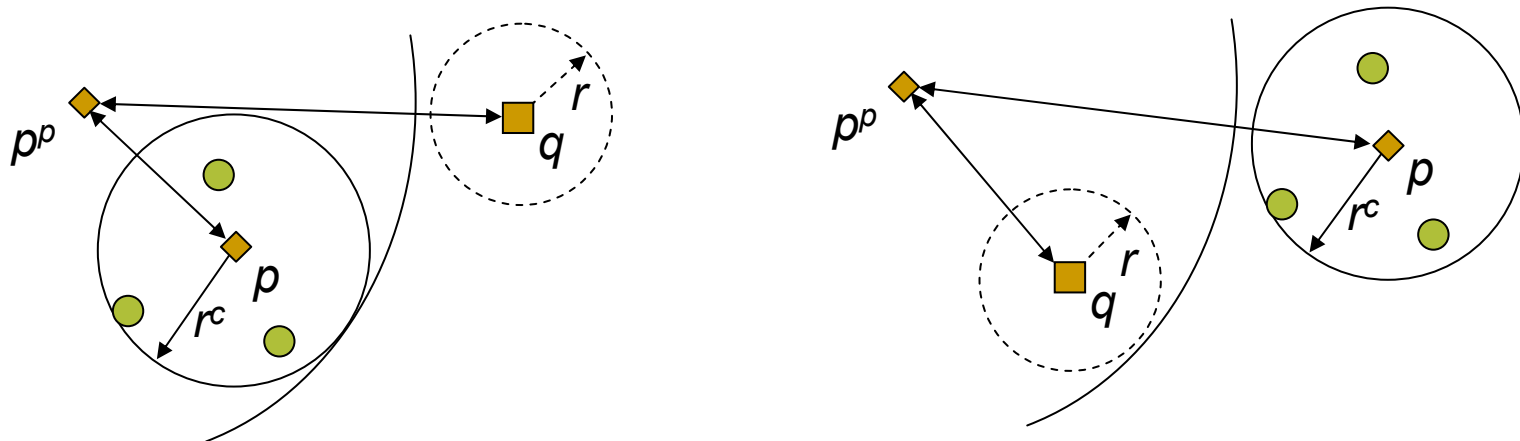
- Partition S to S_1 and S_2 according to p_1 and p_2
- Unbalanced
 - Generalized hyper-plane
- Balanced
 - Larger covering radii
 - Worse than unbalanced



M-tree: Range Search

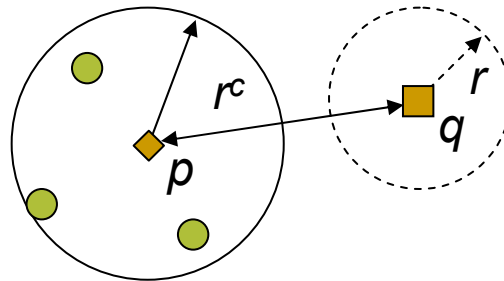
Given $R(q,r)$:

- Traverse the tree in a depth-first manner
- In an internal node, for each entry $\langle p, r^c, d(p, p^p), ptr \rangle$
 - Prune the subtree if $|d(q, p^p) - d(p, p^p)| - r^c > r$
 - Application of the pivot-pivot constraint



M-tree: Range Search (cont.)

- If not discarded, compute $d(q,p)$ and
 - Prune the subtree if $d(q,p) - r^c > r$
 - Application of the range-pivot constraint



- All non-pruned entries are searched recursively.

M-tree: Range Search in Leaf Nodes

- In a leaf node, for each entry $\langle o, d(o, o^p) \rangle$
 - Ignore entry if $|d(q, o^p) - d(o, o^p)| > r$
 - else compute $d(q, o)$ and check $d(q, o) \leq r$
 - Application of the object-pivot constraint

M-tree Family

- Bulk-Loading Algorithm
 - Top-down on static collection [CP98]
 - Plus bottom-up (Radius-based Tree) [MXSM03]
- Multi-Way Insertion Algorithm [SPSK03]

- Slim Tree [TTSF00]
 - Spanning tree based split
 - Slim-down Algorithm [TTSF00]
 - Generalized Slim-down algorithm [SPSK03]
- M-tree & pivot filtering
 - DF-Tree (Distance Field Tree) [TTFF02]
 - Pivoting M-tree [Sko04]
 - Adds ranges in internal nodes

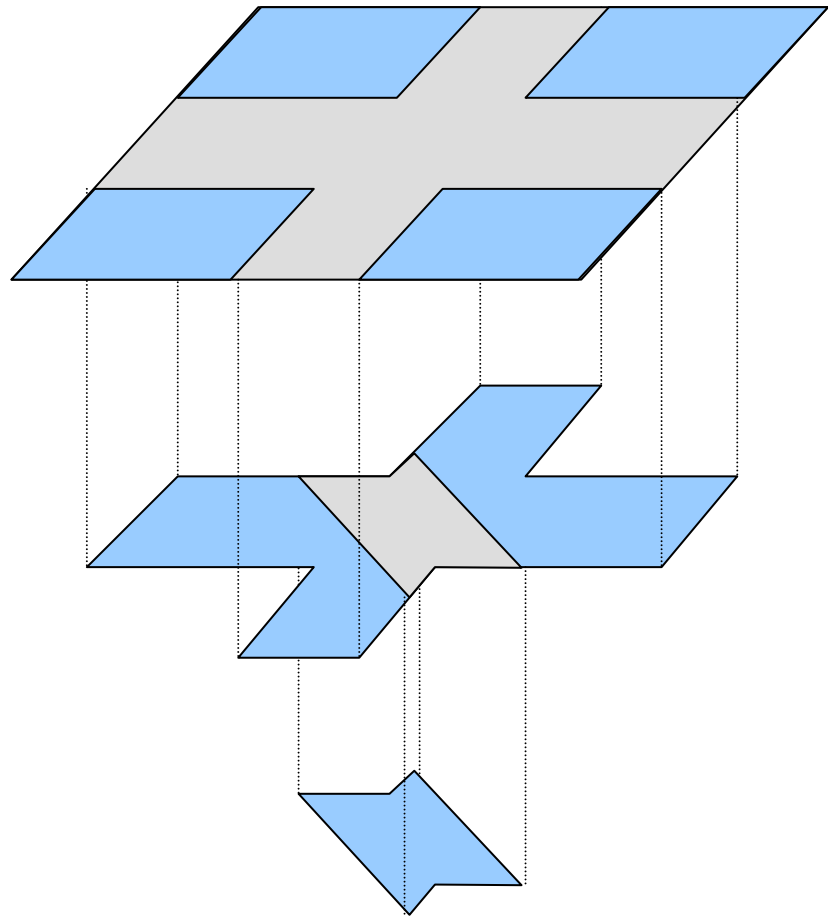
M-tree Family (cont.)

- DBM-tree [VTCT04]
 - Unbalanced in tree height (for dense regions)
- M⁺-tree [ZWYY03]
 - Key-dimension partitioning of balls
- BM⁺-tree [ZWZY05]
 - Two key-dimensions
- M²-tree [CP00a]
 - Complex similarity queries

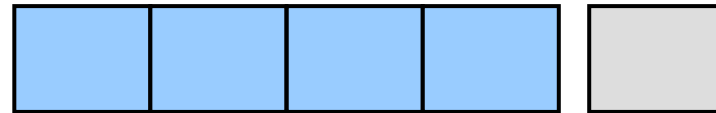
Distance Index (D-index) [DGSZ03]

- Hybrid structure
 - combines pivot-filtering and partitioning
- Multilevel structure based on hashing
 - one ρ -split function per level
- The first level splits the whole data set.
- Next level partitions the exclusion zone of the previous level.
- The exclusion zone of the last level forms the exclusion bucket of the whole structure.

D-index: Structure



4 separable buckets at the first level



2 separable buckets at the second level



exclusion bucket of the whole structure



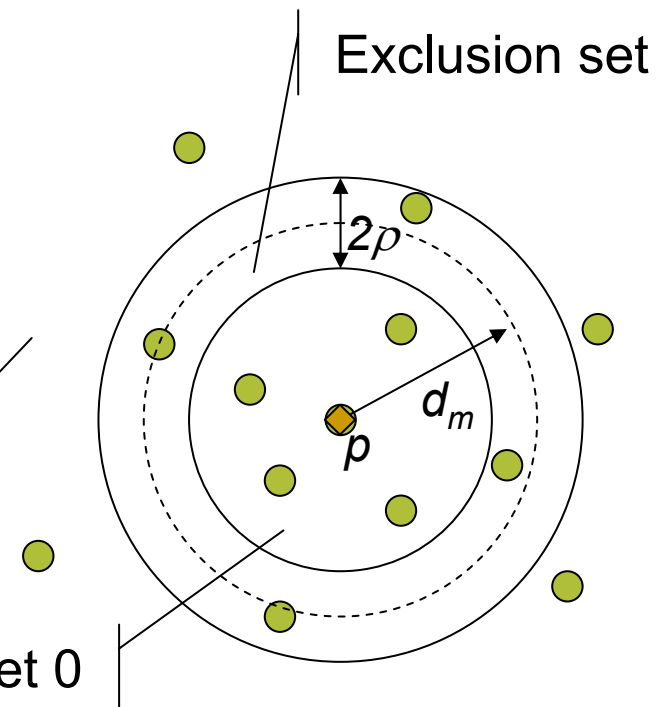
D-index: Partitioning

- Based on excluded middle partitioning

$$\square \text{ bps}^{1,\rho}(x) = \begin{cases} 0 & \text{if } d(x,p) \leq d_m - \rho \\ 1 & \text{if } d(x,p) > d_m + \rho \\ - & \text{otherwise} \end{cases}$$

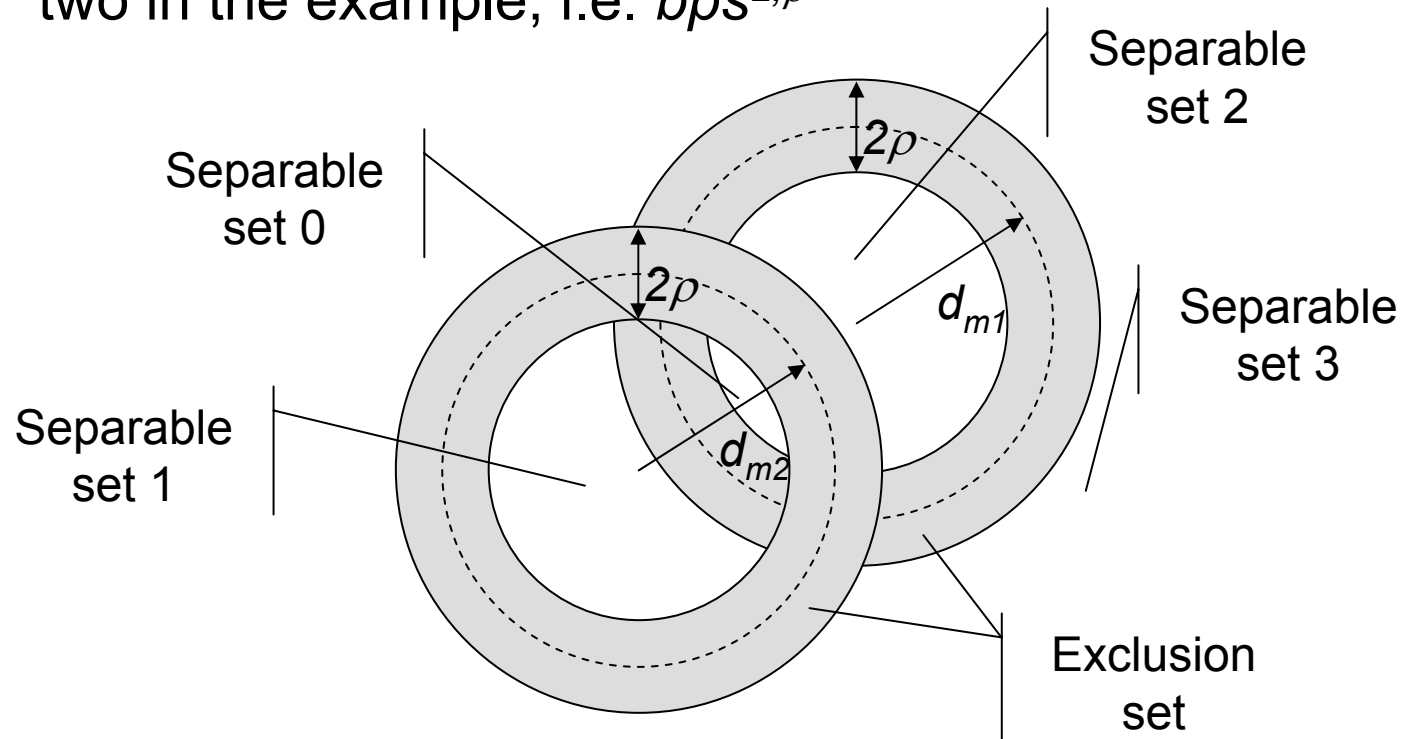
Separable set 1

Separable set 0



D-index: General ρ -Split Function

- Combination of several binary ρ -split functions
 - two in the example, i.e. $bps^{2,\rho}$

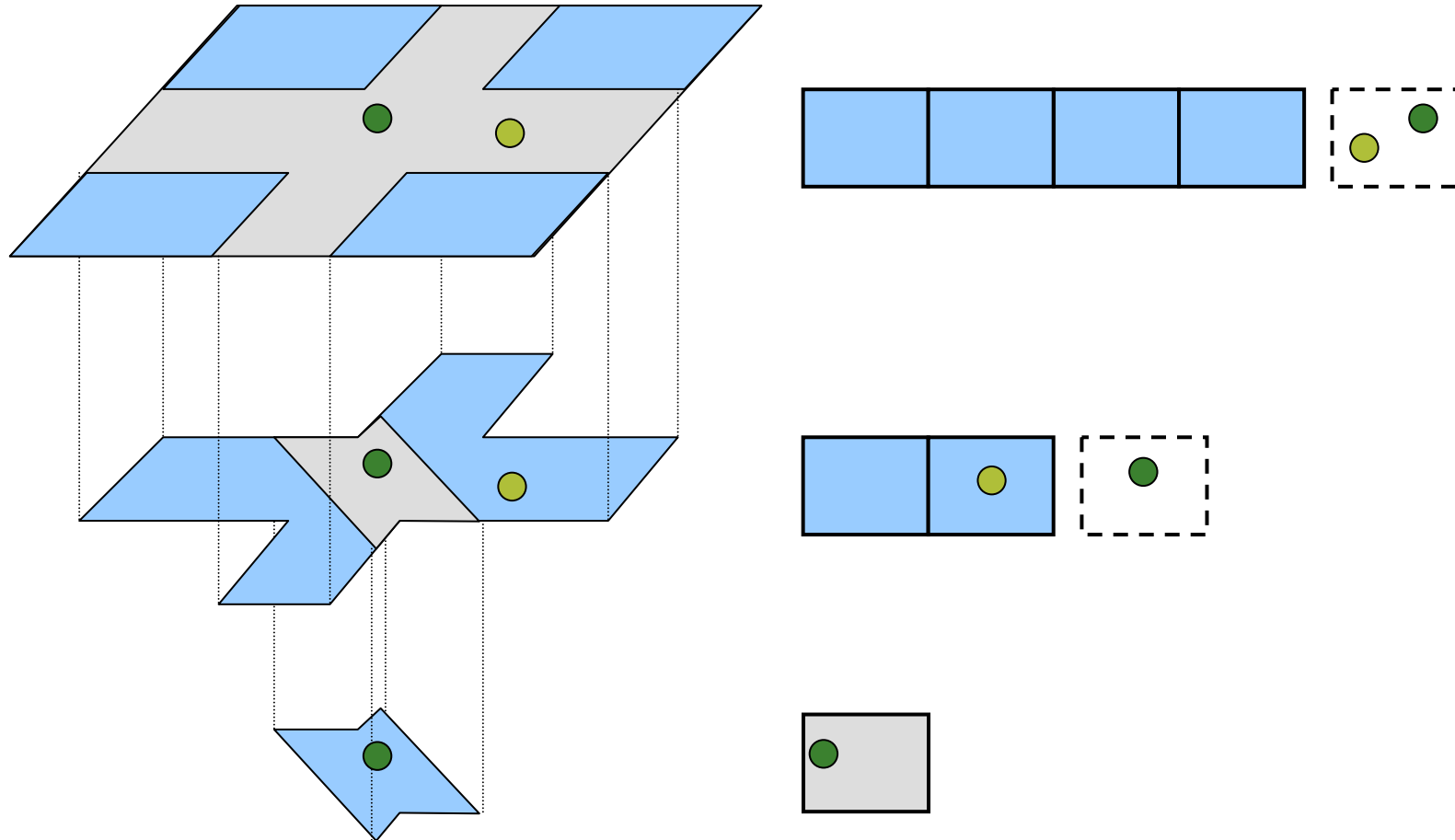


D-index: General ρ -Split Function



- A combination of n first order ρ -split functions:
 - $bps^{n,\rho}: \mathcal{D} \rightarrow \{0..2^n-1, -\}$
 - $bps^{n,\rho}(x) = \begin{cases} - & \text{if } \exists i, bps_i^{1,\rho}(x) = - \\ b & \text{all } bps_i^{1,\rho}(x) \text{ form a binary number } b \end{cases}$
- Resulting sets are also separable up to 2ρ .

D-index: Insertion



D-index: Insertion Algorithm

- $Dindex^\rho(X, m_1, m_2, \dots, m_h)$
 - h – number of levels
 - m_i – number of binary functions combined on level i
- Algorithm – insert the object o_N :
 - for** $i=1$ **to** h **do**
 - if** $bps^{m_i, \rho}(o_N) \neq '-'$ **then**
 - $o_N \rightarrow$ *bucket with the index* $bps^{m_i, \rho}(o_N)$
 - exit**
 - end if**
 - end do**
 - $o_N \rightarrow$ *global exclusion bucket*

D-index: Insertion Algorithm (cont.)

- The new object is inserted with one bucket access.
- Requires $\sum_{i=1}^j m_i$ distance computations
 - Assuming o_N was inserted in a bucket on the level j .

D-index: Range Query

- $Dindex^{\rho}(X, m_1, m_2, \dots, m_h)$
 - h – number of levels
 - m_i – number of binary functions combined on level i

Given a query $R(q,r)$ with $r \leq \rho$:

for $i=1$ to h do

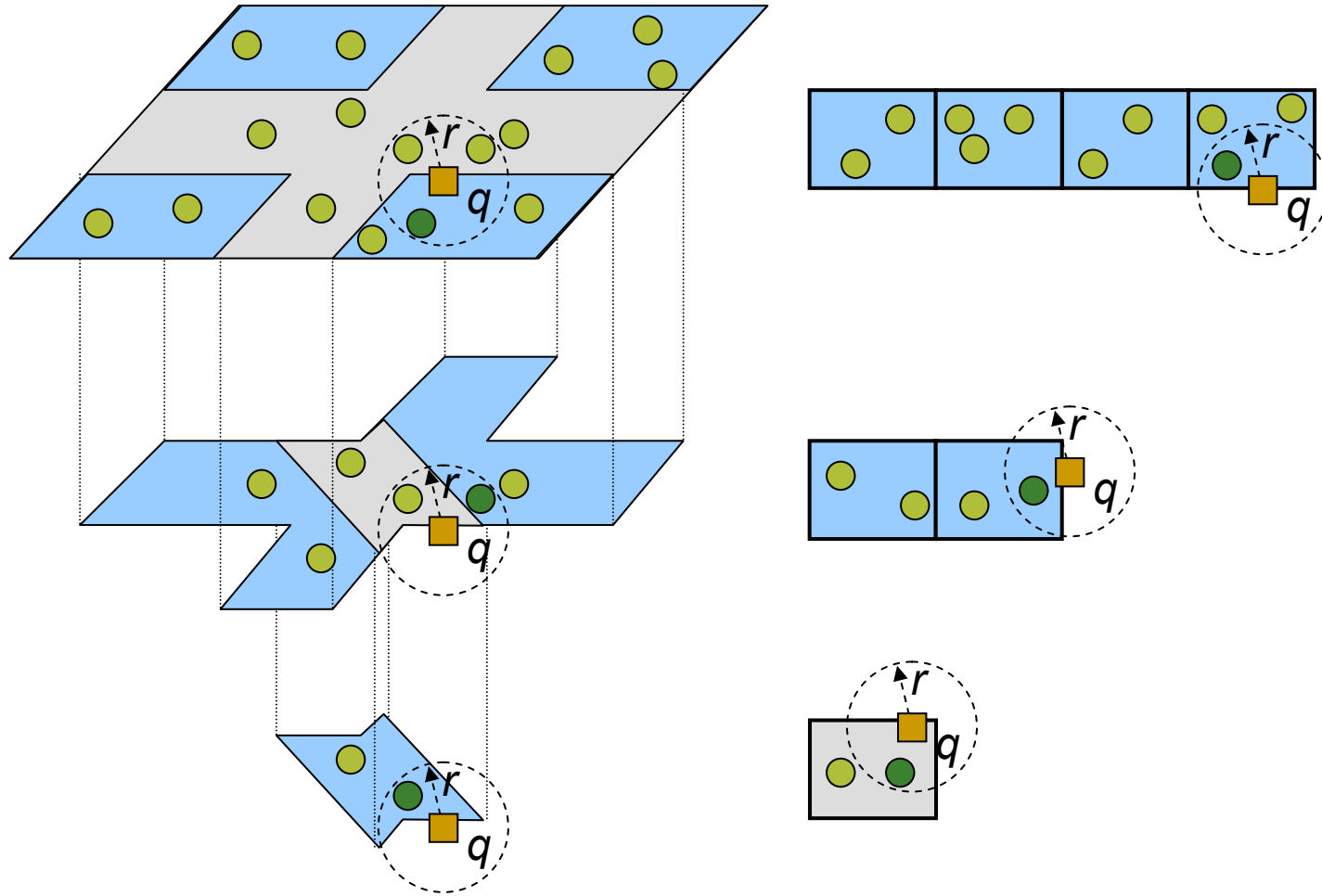
search in the bucket with the index $bps^{m_i,0}(q)$

end do

search in the global exclusion bucket

- Objects o , $d(q,o) \leq r$, are reported on the output.

D-index: Range Search (cont.)



D-index: Range Query (cont.)

- The call of $bps^{m_i,0}(q)$ always returns a value between 0 and $2^{m_i}-1$.
- Exactly one bucket per level is accessed if $r \leq \rho$
 - $h+1$ bucket accesses
- Reducing the number of bucket accesses:
 - The query region is in the exclusion set \Rightarrow proceed the next level directly.
 - The query region is in a separable set \Rightarrow terminate the search.

D-index: Features

- Supports disk storage
- Insertion needs one bucket access
 - Distance computations vary from m_1 up to $\sum_{i=1..h} m_i$
- $h+1$ bucket accesses at maximum
 - For all queries such that qualifying objects are within ρ
- Exact match ($R(q, 0)$)
 - Successful – one bucket access
 - Unsuccessful – typically no bucket is accessed
- Extension to similarity joins: eD-index [DGZ03]

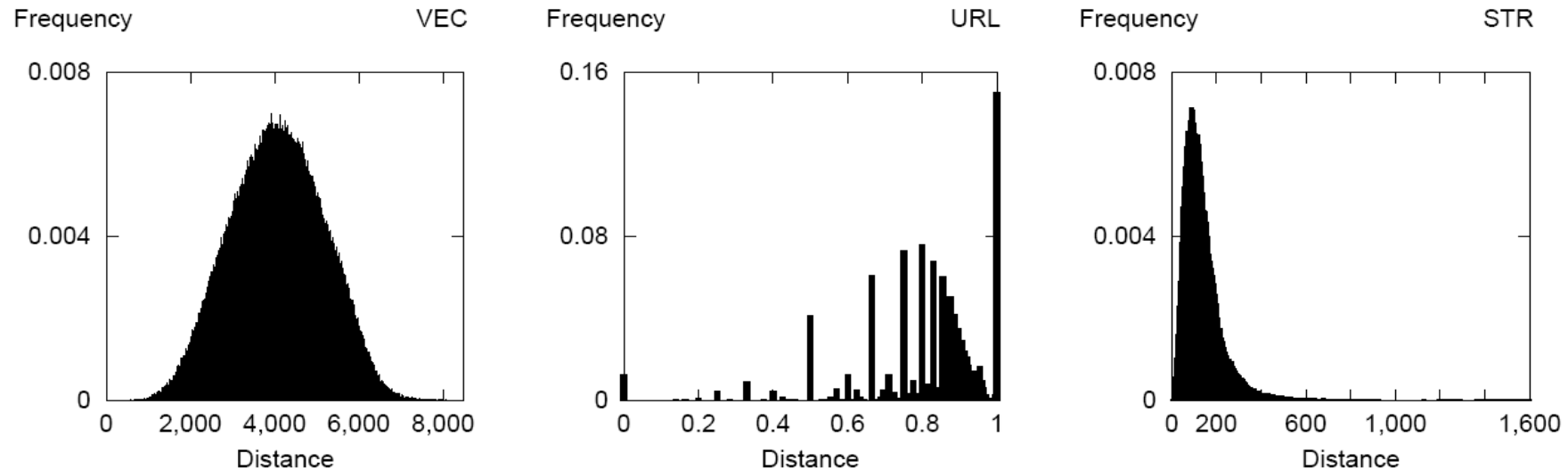
Performance Trials

- Experiments on M-tree and D-index [DGSZ03]
 1. **Comparison** of M-tree (tree-based approach) vs. D-index (hash-based approach)
 2. **Scalability** of the centralized indexes – growing the size of indexed dataset

Datasets and Distance Measures

- Trials performed on three datasets:
 - **VEC**: 45-dimensional vectors of image color features compared by the *quadratic distance* measure
 - **URL**: sets of URL addresses; the distance measure is based on the similarity of sets (*Jaccard's coefficient*).
 - **STR**: sentences of a Czech language corpus compared using the *edit distance*

Datasets: Distance Density



- Density of distances within the datasets:
 - VEC: practically normal distance density
 - URL: discrete density
 - STR: skewed density

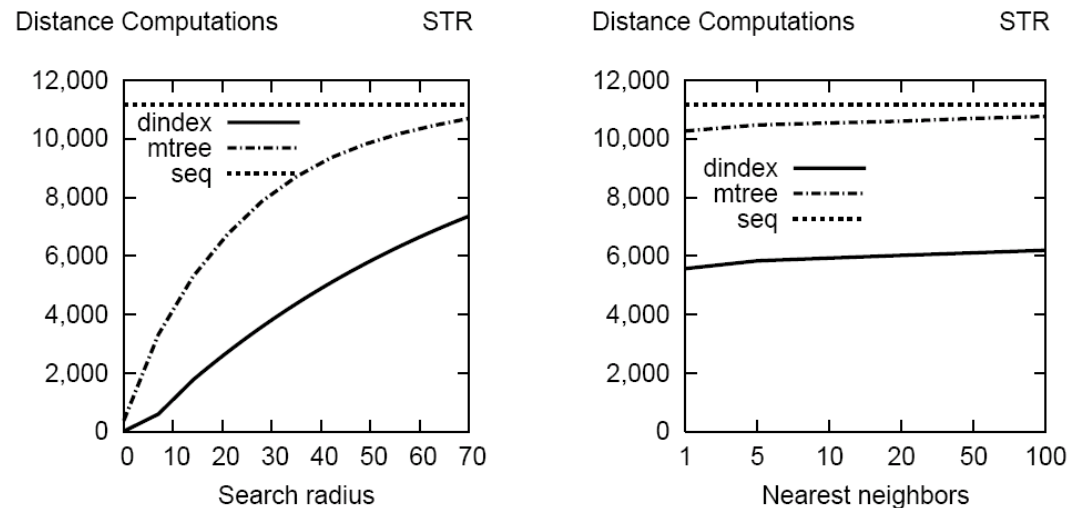
Trials: Measurements & Settings

- CPU costs: number of distance computations
- I/O costs: number of block reads
 - The same size of disk blocks
- Query objects follow the dataset distribution.
- Average values over 50 queries:
 - Different query objects
 - The same radius or number of nearest neighbors

Comparison of Indexes

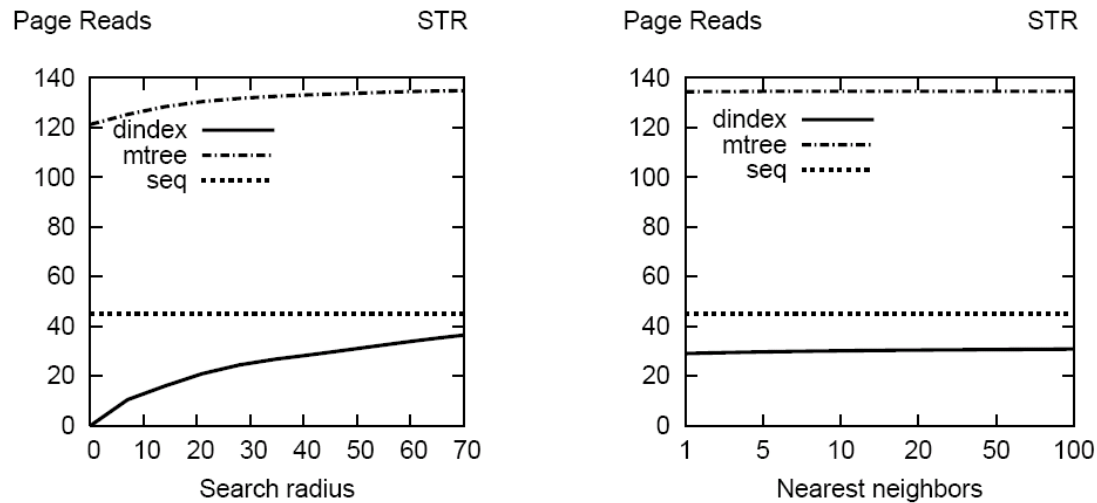
- Comparing performance of
 - M-tree – a tree-based approach
 - D-index – hash-based approach
 - *sequential scan* (baseline)
- Dataset of 11,100 objects
- Range queries – increasing radius
 - maximal selectivity about 20% of the dataset

Range vs. k -NN: CPU Costs



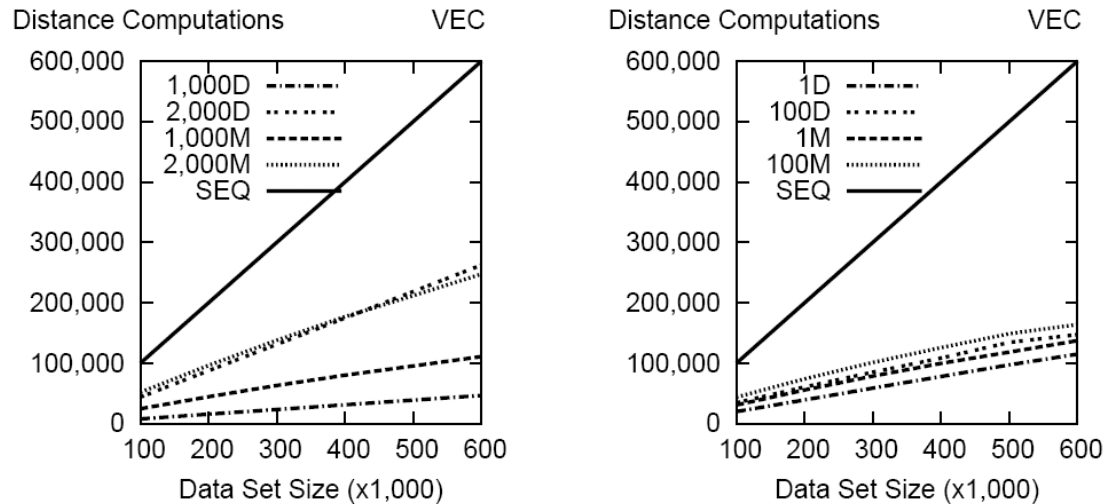
- Nearest neighbor query:
 - Similar trends for M-tree and D-index
 - D-index's advantage of small radii processing decreases.
 - Expensive even for small k – similar costs for both 1 and 100
 - D-index still twice as fast as M-tree

Range vs. k -NN: I/O Costs



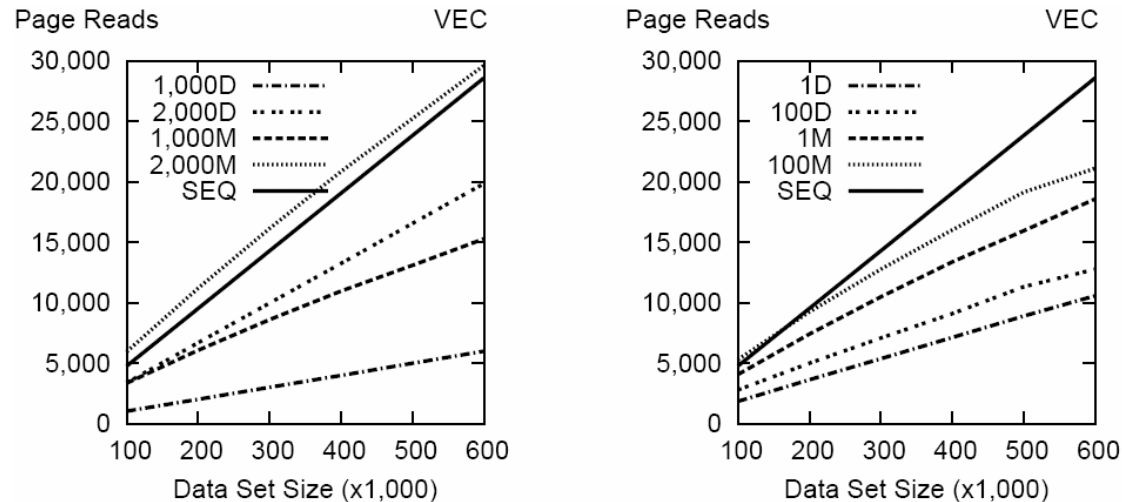
- Nearest neighbor query:
 - Similar trends for I/O costs as for CPU costs
 - D-index four times faster than M-tree

Scalability: CPU Costs



- Range query: $r = 1,000; 2,000$ □ k -NN query: $k = 1; 100$
- Labels: $r, k + D$ (D-index), M (M-tree), SEQ
- Data: from 100,000 to 600,000 objects
- M-tree and D-index are faster (D-index slightly better)
- Linear trends

Scalability: I/O Costs



- The same trends as for CPU costs
- D-index more efficient than M-tree
- *Exact match* contrast:
 - M-tree: 6,000 blocks read + 20,000 d. c. for 600,000 objects
 - D-index: 1 block read + 18 d. c. regardless of the data size

Scalability Experiments: Conclusions

- Similarity search is expensive.
- The scalability of centralized indexes is linear.
- They cannot be applied to huge data archives.
 - Become inefficient after a certain point

Possible solutions:

- Sacrifice some precision: *approximate techniques*
- Use more storage & computational power:
distributed data structures

Outline of the talk

1. similarity, metric space and distance measures (15')
2. similarity queries, metric partitioning principles (15')
3. query execution strategies (15')
4. avoiding distance computations (15')
5. filtering, pivot choosing and metric transformations (15')
6. short survey of metric space indexes (15')
7. M-tree family, D-index - performance evaluation (30')
8. **approximate similarity search (30')**
9. scalable and distributed indexes (30')

Principles of Approx. Similarity Search

- Approximate similarity search overcomes typical problems of exact similarity search.
 - Moderate improvement of performance with respect to the sequential scan
 - Dimensionality curse
- Similarity search returns mathematically precise result sets.
 - Similarity is often subjective, so in some cases also approximate results satisfy the user's needs.
 - Useful references:
 - [Ama02, AMN+98, ARSZ03, CP00b, FTAA01, LCGW02, PAL99, TFR02, Vol2, WB00, ZSAR98]

Principles of Approx. Similarity Search (cont.)

- Approximate similarity search processes a query faster at the price of imprecision in results returned.
 - Useful, for instance, in interactive systems:
 - Similarity search is typically an iterative process.
 - Users submit several search queries before being satisfied.
 - Fast approximate similarity search in intermediate queries can be useful.
 - Improvements up to *two* orders of magnitude

Approx. Similarity Search: Basic Strategies

- Space transformation
 - Distance preserving transformations [HS03b]
 - Distances in the transformed space are smaller than in the original space.
 - Possible false hits
 - Example:
 - Dimensionality reduction [CP97, EF00, OF03, WRR03] techniques such as
 - KLT, DFT, DCT, DWT
 - VA-files [WB00, FTAA01]
 - We will not discuss this approximation strategy in this tutorial.

Approx. Similarity Search: Basic Strategies

- Reducing volume of examined data
 - Not promising data is not accessed.
 - False dismissals can occur.

- This class of strategies will be discussed deeply in the following slides.

Reducing Volume of Examined Data

- Possible strategies:
 - *Early termination strategies*
 - A search algorithm might stop before all the needed data has been accessed.

 - *Relaxed branching strategies*
 - Data regions overlapping the query region can be discarded depending on a specific relaxed pruning strategy.

Early Termination Strategies

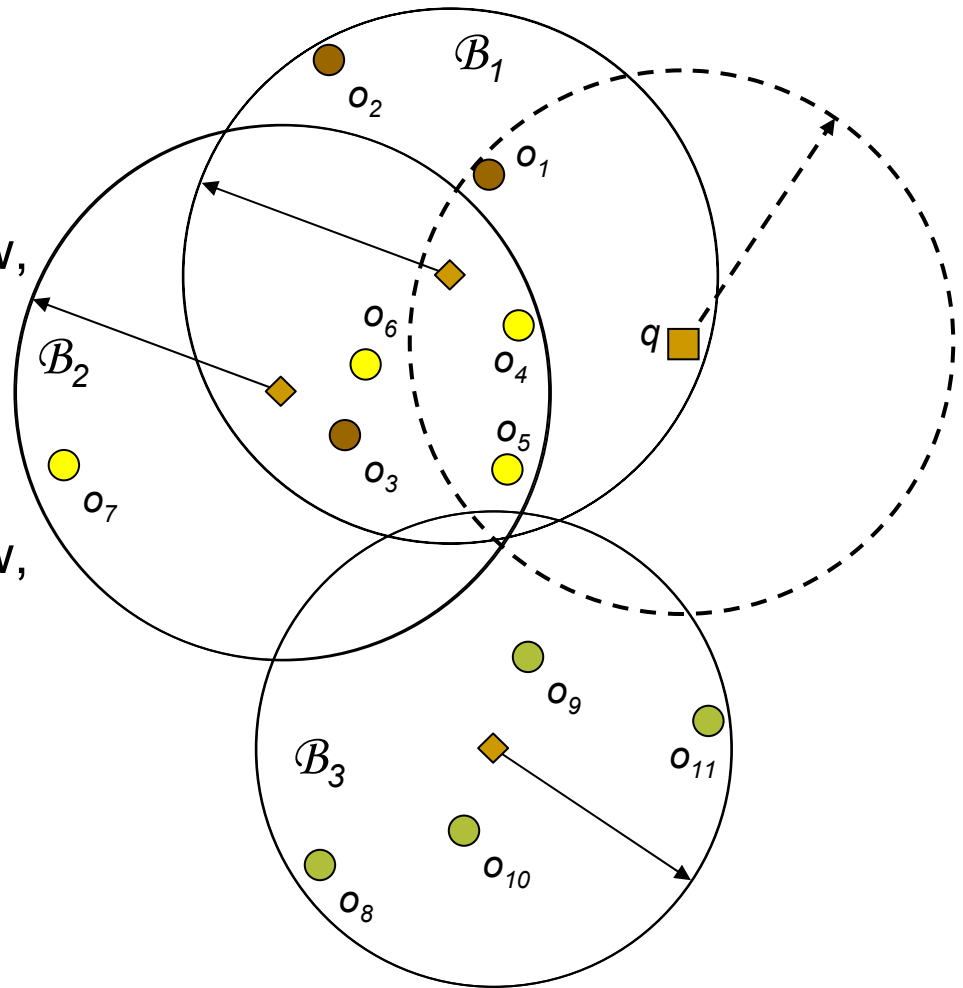
- Approximate similarity search algorithms
 - Use a “relaxed” *stop condition*
 - stops the algorithm when little chances of improving the current results are detected.
- The hypothesis is that
 - A good approximation is obtained after a few iterations.
 - Further steps would consume most of the total search costs and would only marginally improve the result-set.

Relaxed Branching Strategies

- Exact similarity search algorithms
 - Access all data regions overlapping the query region and discard all the others.
- Approximate similarity search algorithms
 - Use a “relaxed” *pruning condition* that
 - Rejects regions overlapping the query region when it detects a low probability that data objects are contained in the intersection.
- In particular, useful and effective with access methods based on hierarchical decomposition of the space.

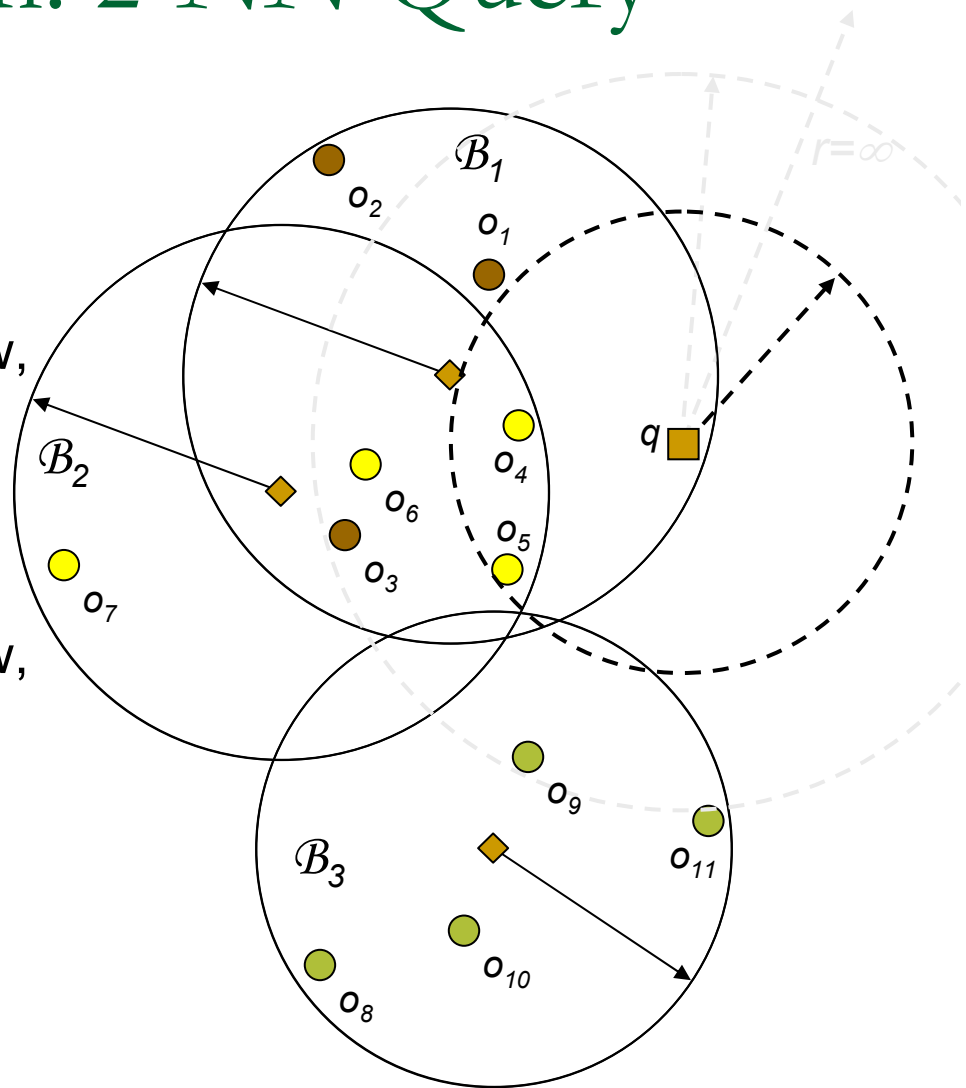
Approximate Search: Range Query

- Given a range query:
- Access \mathcal{B}_1
 - Report o_1
 - If early termination stopped now, we would lose objects.
- Access \mathcal{B}_2
 - Report o_4, o_5
 - If early termination stopped now, we would not lose anything.
- Access \mathcal{B}_3
 - Nothing to report
 - A relaxed branching strategy may discard this region – we don't lose anything.



Approximate Search: 2-NN Query

- Given a 2-NN query:
- Access \mathcal{B}_1
 - Neighbors: o_1, o_3
 - If early termination stopped now, we would lose objects.
- Access \mathcal{B}_2
 - Neighbors: o_4, o_5
 - If early termination stopped now, we would not lose anything.
- Access \mathcal{B}_3
 - Neighbors: o_4, o_5 – no change
 - A relaxed branching strategy may discard this region – we don't lose anything.



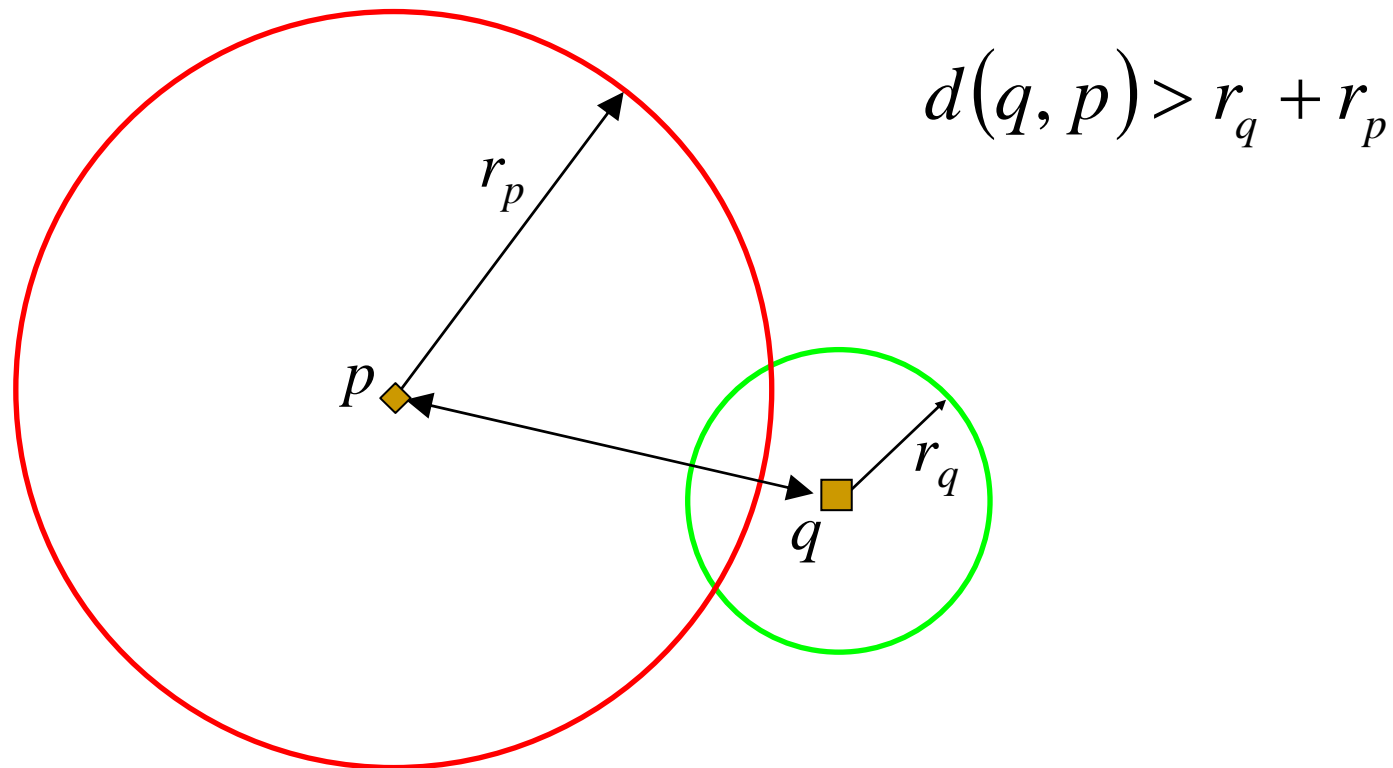
Discussed Approximation Techniques

- **relative error approximation (pruning condition)**
 - Range and k-NN search queries
- good fraction approximation (stop condition)
 - k-NN search queries
- small chance improvement approx. (stop cond.)
 - k-NN search queries
- proximity-based approximation (pruning condition)
 - Range and k-NN search queries
- PAC NN searching (pruning & stop cond.)
 - 1-NN search queries

Relative Error Approximation

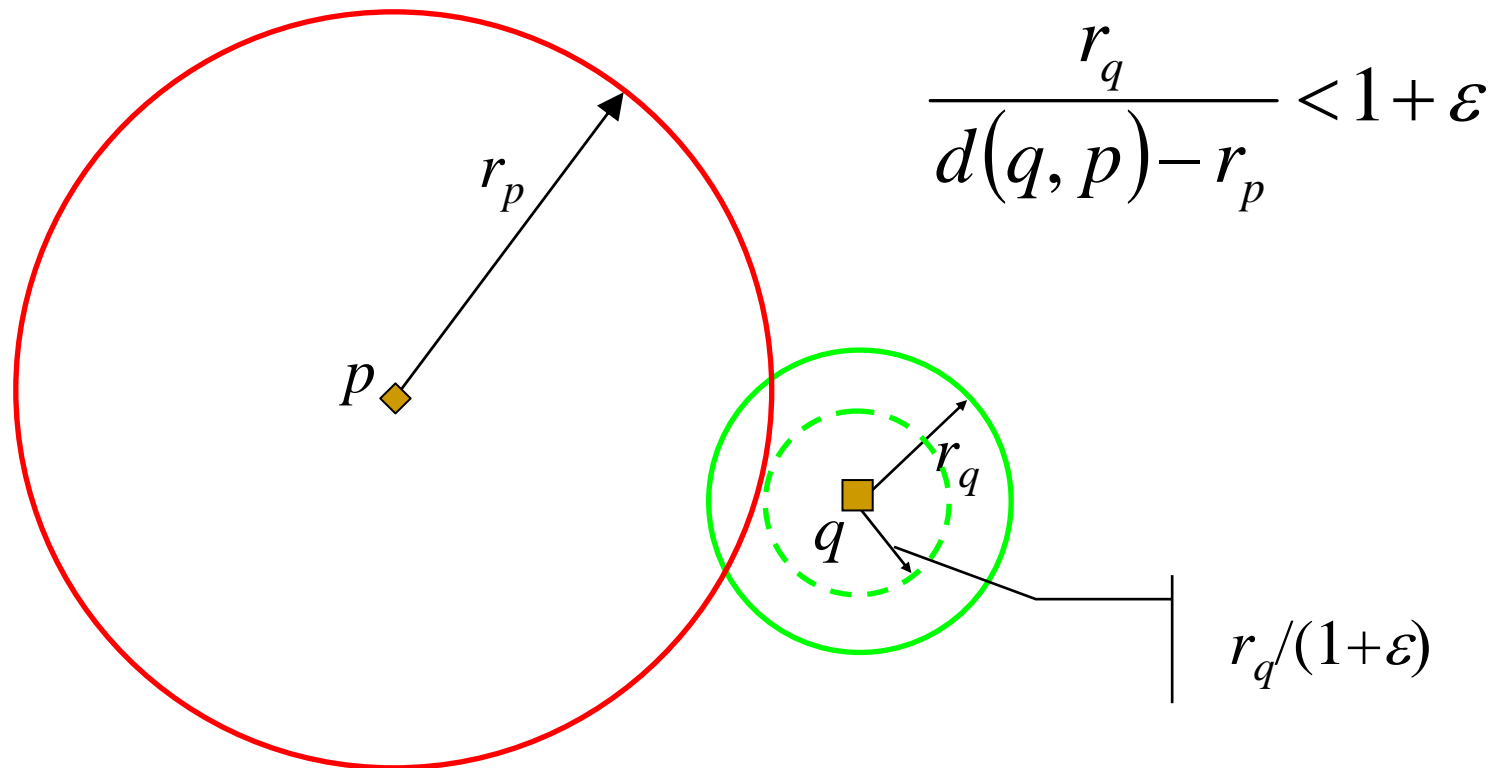
[Ama02, ZSAR98]

- Exact pruning strategy:



Relative Error Approximation (cont.)

- Approximate pruning strategy:

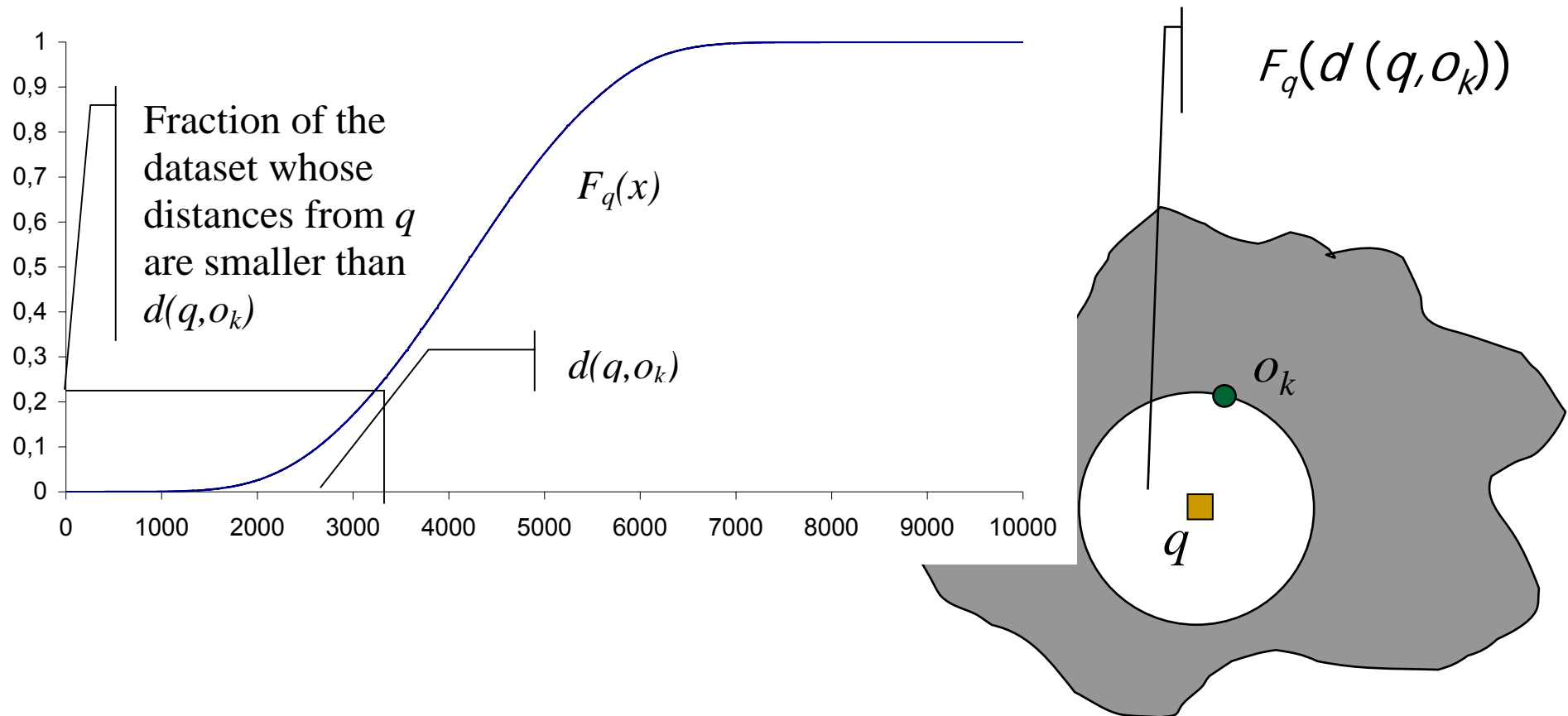


Discussed Approximation Techniques

- relative error approximation (pruning condition)
 - Range and k-NN search queries
- **good fraction approximation (stop condition)**
 - **k-NN search queries**
- small chance improvement approx. (stop cond.)
 - k-NN search queries
- proximity-based approximation (pruning condition)
 - Range and k-NN search queries
- PAC NN searching (pruning & stop cond.)
 - 1-NN search queries

Good Fraction Approximation

[Ama02, ZSAR98]



Discussed Approximation Techniques

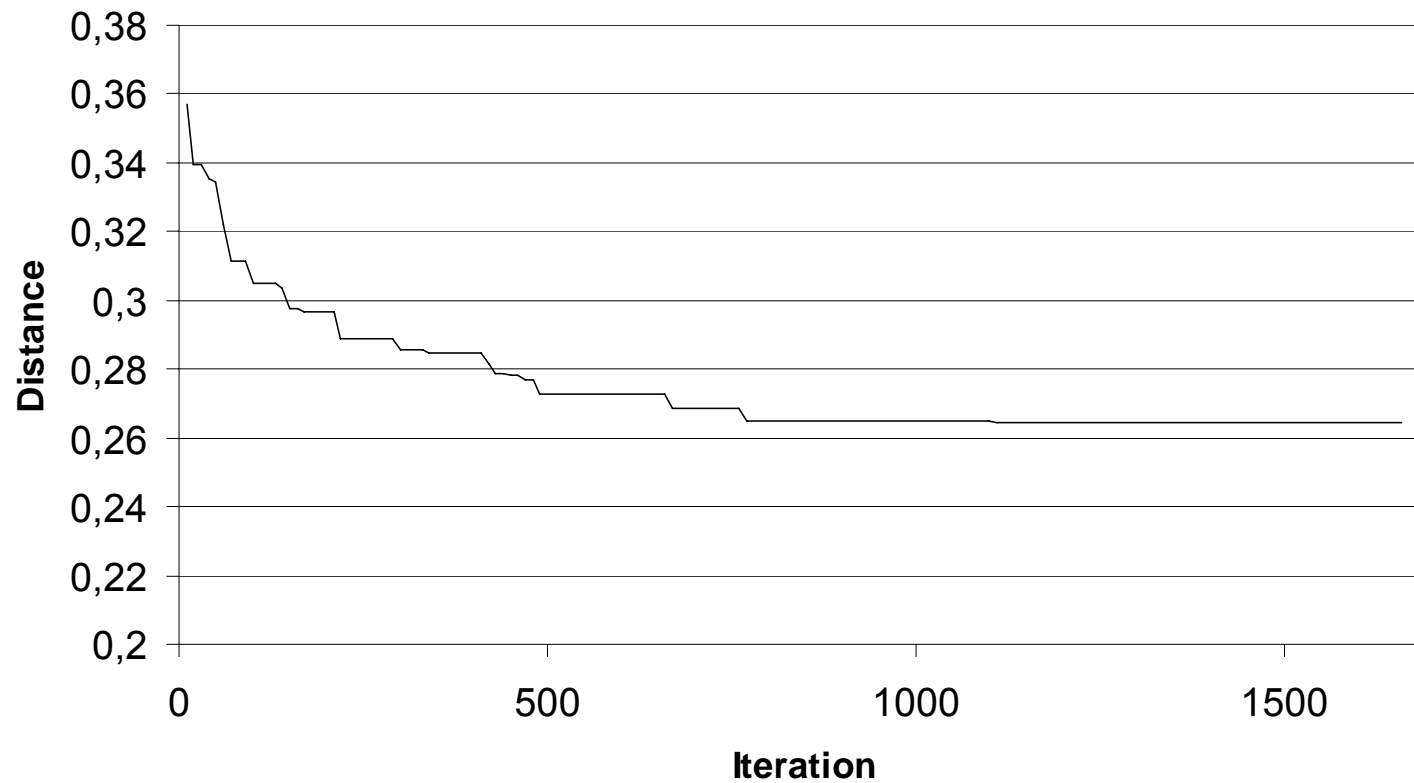
- relative error approximation (pruning condition)
 - Range and k-NN search queries
- good fraction approximation (stop condition)
 - k-NN search queries
- **small chance improvement approx. (stop cond.)**
 - **k-NN search queries**
- proximity-based approximation (pruning condition)
 - Range and k-NN search queries
- PAC NN searching (pruning & stop cond.)
 - 1-NN search queries

Small Chance Improvement Approximation

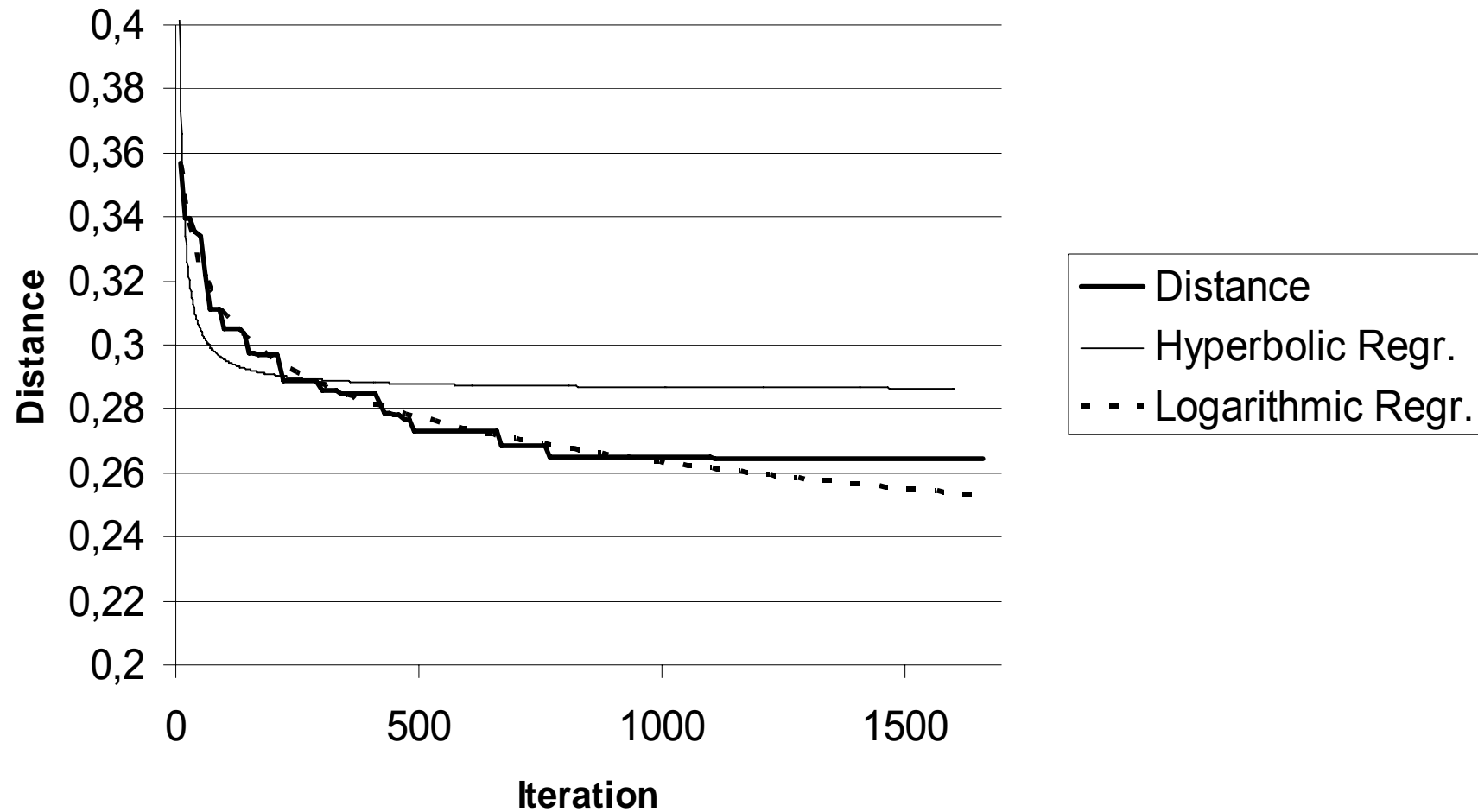
- The M-Tree's k -NN algorithm determines the final result by improving the current result set
- In each step of the algorithm, the temporary result is improved and the distance of the k -th element decreases
- When the improvement of the temporary result set slows down, the algorithms can stop.
- [Ama02, ZSAR98]

Small Chance Improvement Approximation (cont.)

$$f(x) : \longrightarrow d(q, o_k^A)$$



Regression Curves



Discussed Approximation Techniques

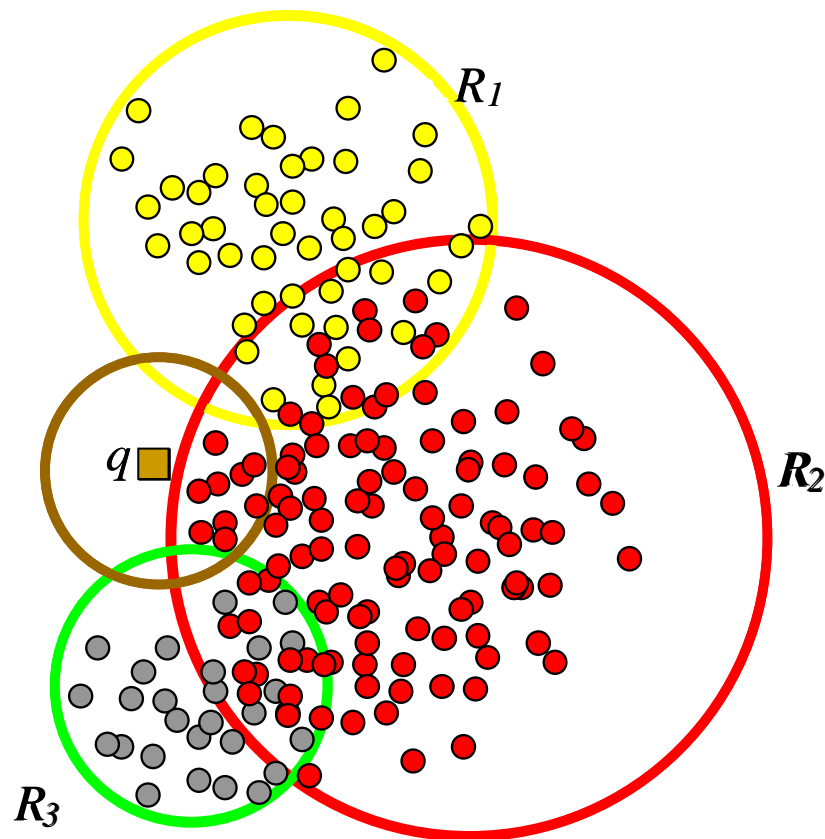
- relative error approximation (pruning condition)
 - Range and k-NN search queries
- good fraction approximation (stop condition)
 - k-NN search queries
- small chance improvement approx. (stop cond.)
 - k-NN search queries
- **proximity-based approximation (pruning condition)**
 - **Range and k-NN search queries**
- PAC NN searching (pruning & stop cond.)
 - 1-NN search queries

Proximity-based Approximation

[Ama02, ARSZ03]

- Regions whose probability of containing qualifying objects is below a certain threshold are pruned even if they overlap the query region.
 - *Proximity* between regions is used to obtain such a probability.
- This results in an increase of performance of two orders of magnitude both for range queries and nearest neighbour queries.

Proximity-based Approximation (cont.)



Discussed Approximation Techniques

- relative error approximation (pruning condition)
 - Range and k-NN search queries
- good fraction approximation (stop condition)
 - k-NN search queries
- small chance improvement approx. (stop cond.)
 - k-NN search queries
- proximity-based approximation (pruning condition)
 - Range and k-NN search queries
- **PAC NN searching (pruning & stop cond.)**
 - **1-NN search queries**

PAC Nearest Neighbor Searching

[CP00b]

- It uses both a relaxed branching condition and a stop condition.
 - The relaxed branching condition is the same as one used for the relative error approximation to find a result bounded by the relative distance error ε .
 - In addition, it halts prematurely when the probability that we have found the bounded result, is above a threshold δ .
- It can only be used for 1-*NN* search queries.



Measures of Performance

- Performance assessments of approximate similarity search should consider
 - Improvement in efficiency
 - Accuracy of approximate results
- Typically there is a trade-off between the two
 - High improvement in efficiency is obtained at the expense of accuracy in the results.
- Good approximate search algorithms should
 - Offer high improvement in efficiency with high accuracy in the results.

Improvement in Efficiency

- Improvement in Efficiency (IE) is expressed as
 - the ratio between the cost of the exact and approximate execution of a query Q :

$$IE = \frac{Cost(Q)}{Cost^A(Q)}$$

- $Cost$ and $Cost^A$ denote the *number of disk accesses* or alternatively the *number of distance computations* for the precise and approximate execution of Q , respectively.
- Q is a range or k-nearest neighbors query.

Recall

- *Recall (R)*: percentage of qualifying objects that are retrieved by the approximate algorithm.

$$R = \frac{|S \cap S^A|}{|S|}$$

- S – qualifying objects, i.e., objects retrieved by the precise algorithm
- S^A – actually retrieved objects, i.e., objects retrieved by the approximate algorithm

Error on Position

- The *error on position (EP)* is defined as

$$EP = \frac{\sum_{i=1}^{S^A} (OX(o_i) - S^A(o_i))}{|S^A| \cdot |X|}$$

- Where:

OX – the list containing the entire dataset ordered with respect to q

S^A – the approx. result ordered with respect to q

$S^A(o)$ and $OX(o)$ – the position of object o in the list

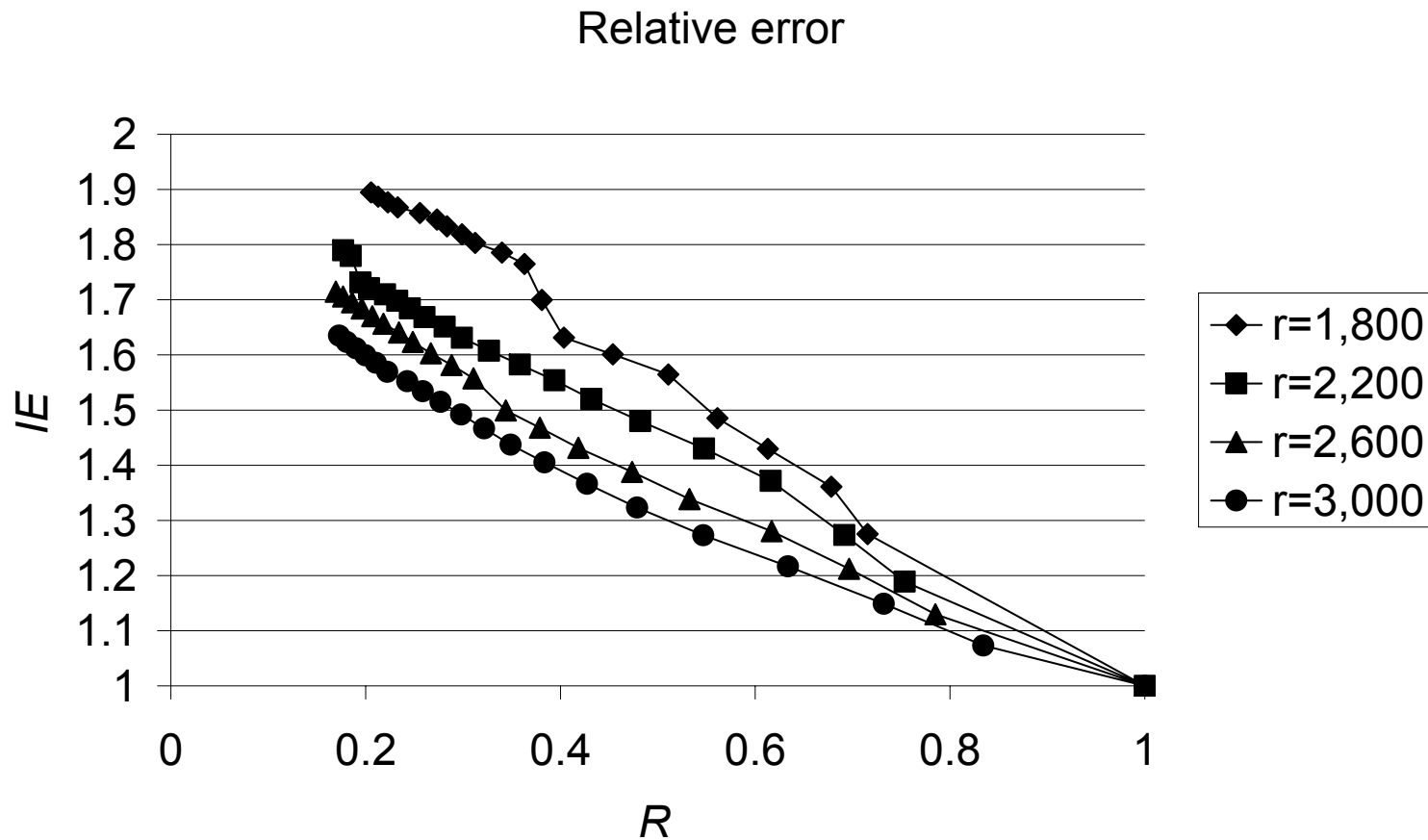
$|S^A| \cdot |X|$ – a normalization factor

- See Also [Dia88, DKNS01]

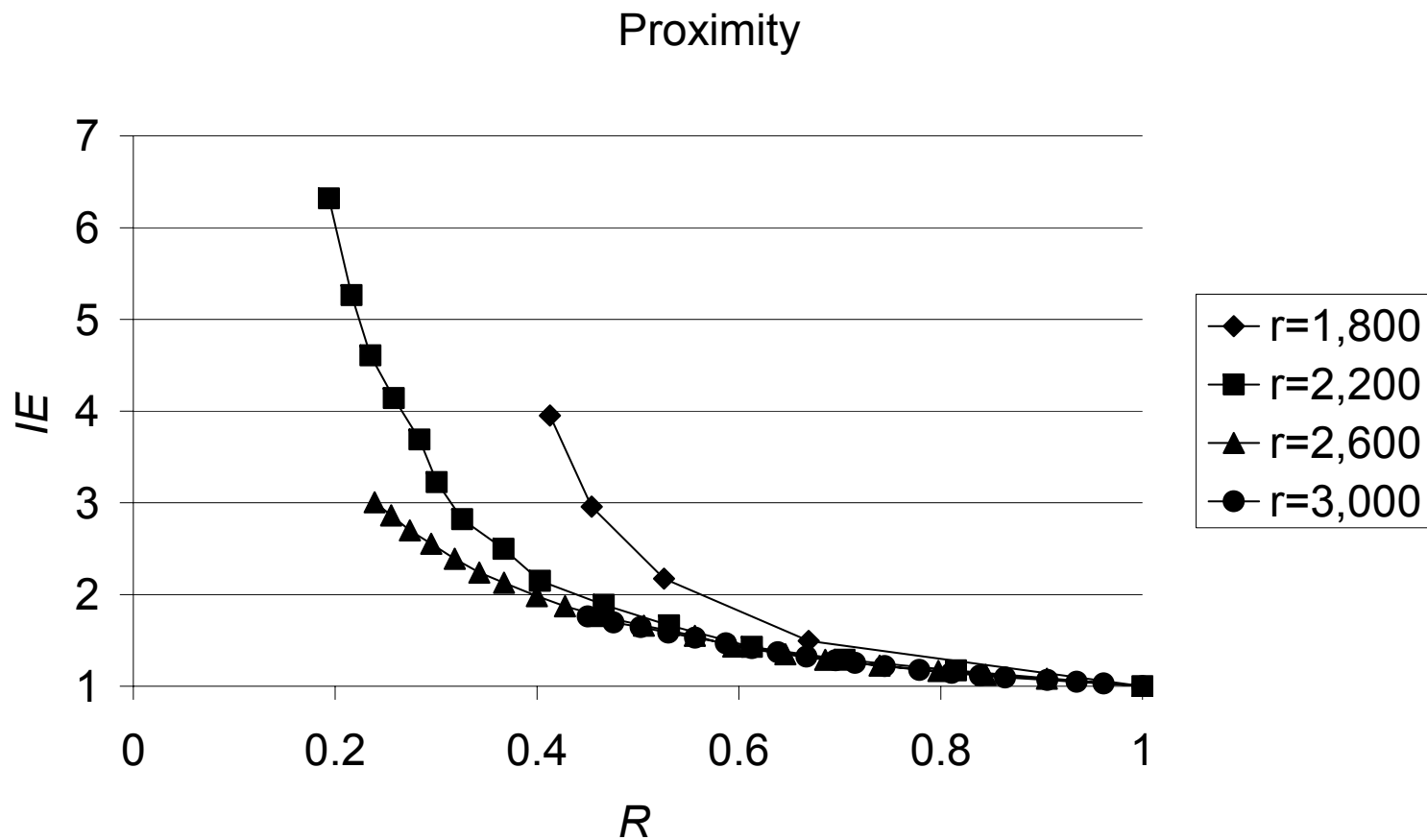
Comparison

- Tests on the **VEC** dataset of 11,100 objects
 - Objects are vectors of 45 dimensions.
- We compared the five approximation approaches.
- Range queries tested with:
 - Relative error
 - Proximity
- Nearest-neighbor queries tested with:
 - All methods

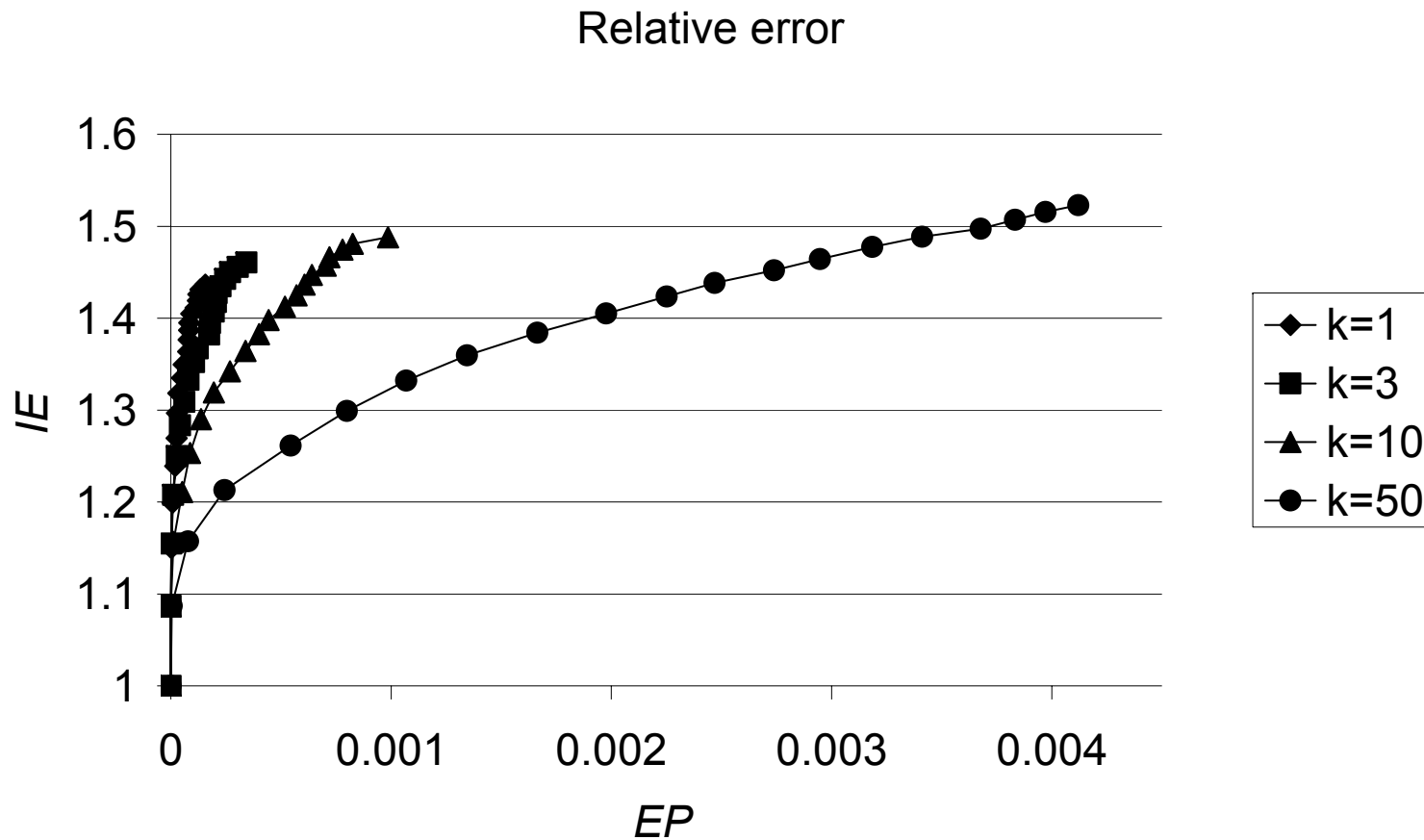
Comparison: Range Queries



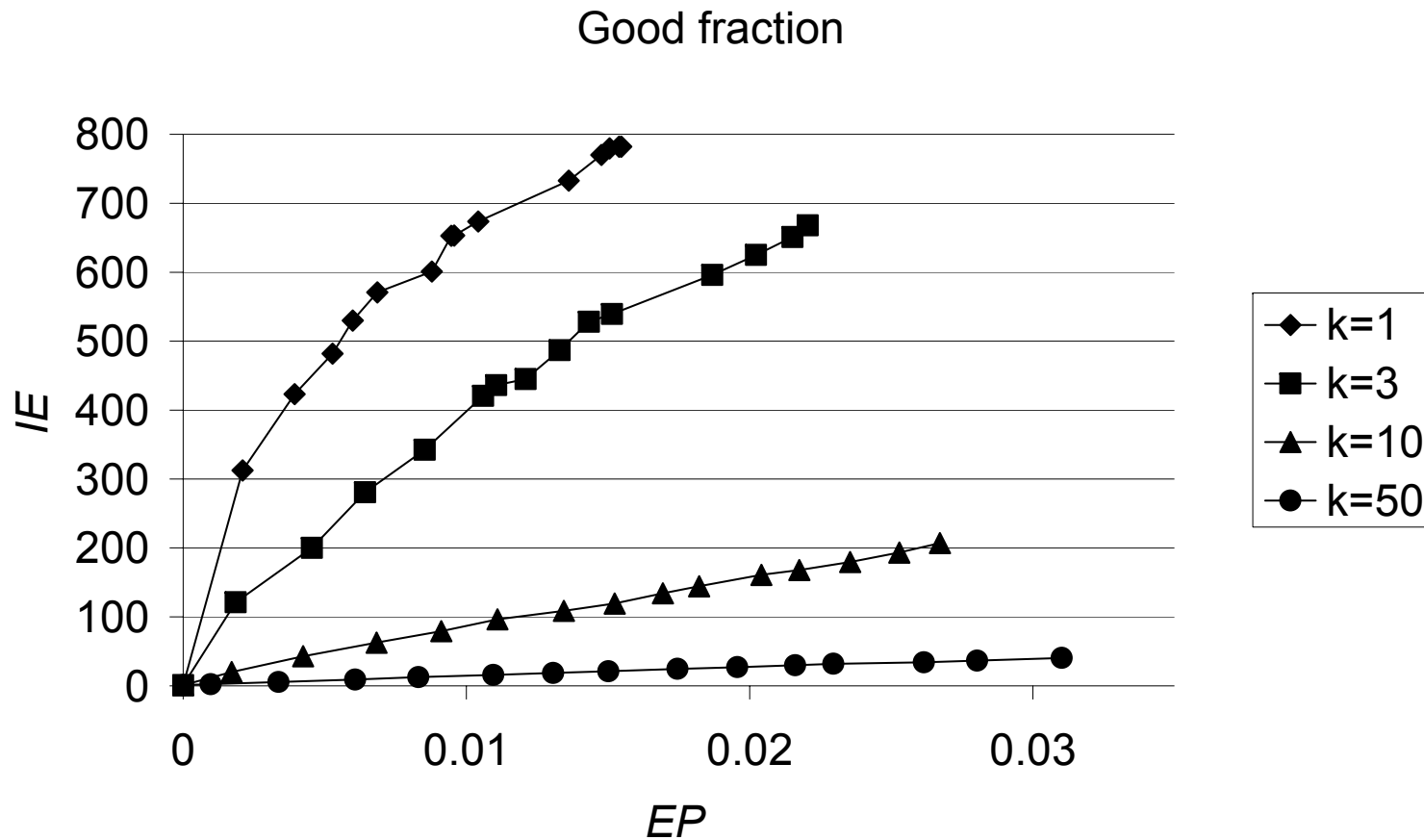
Comparison: Range Queries (cont.)



Comparison: *NN* Queries

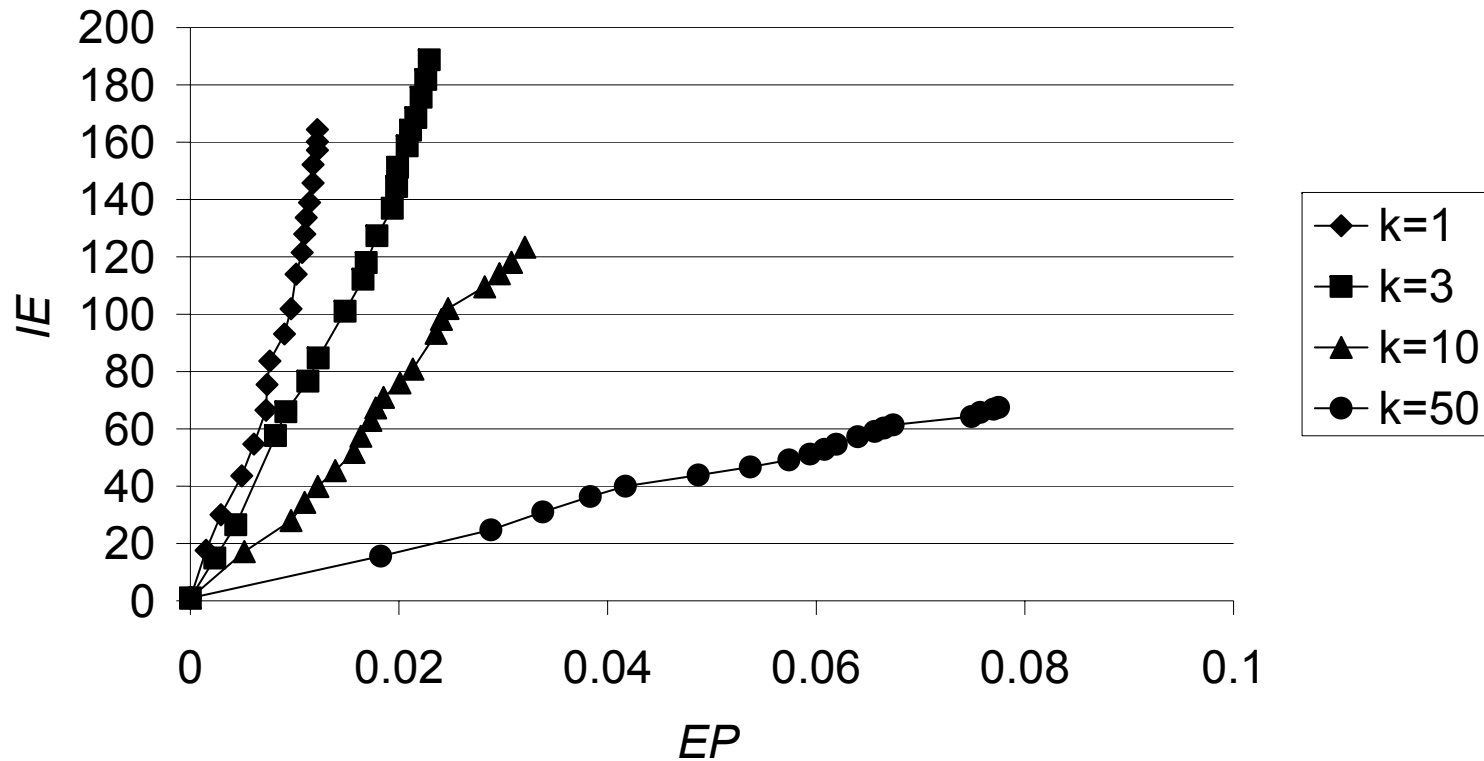


Comparison: *NN* Queries (cont.)

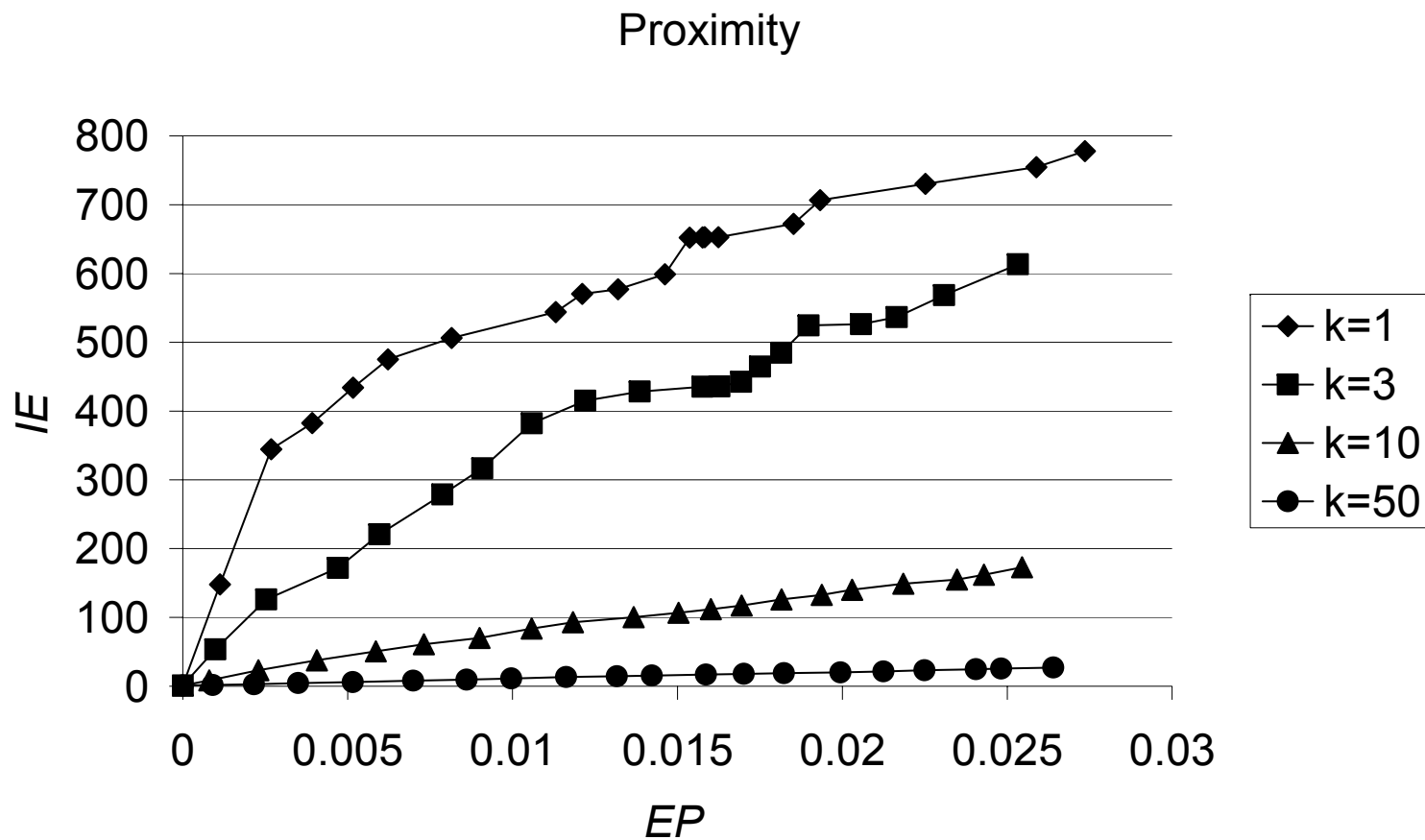


Comparison: *NN* Queries (cont.)

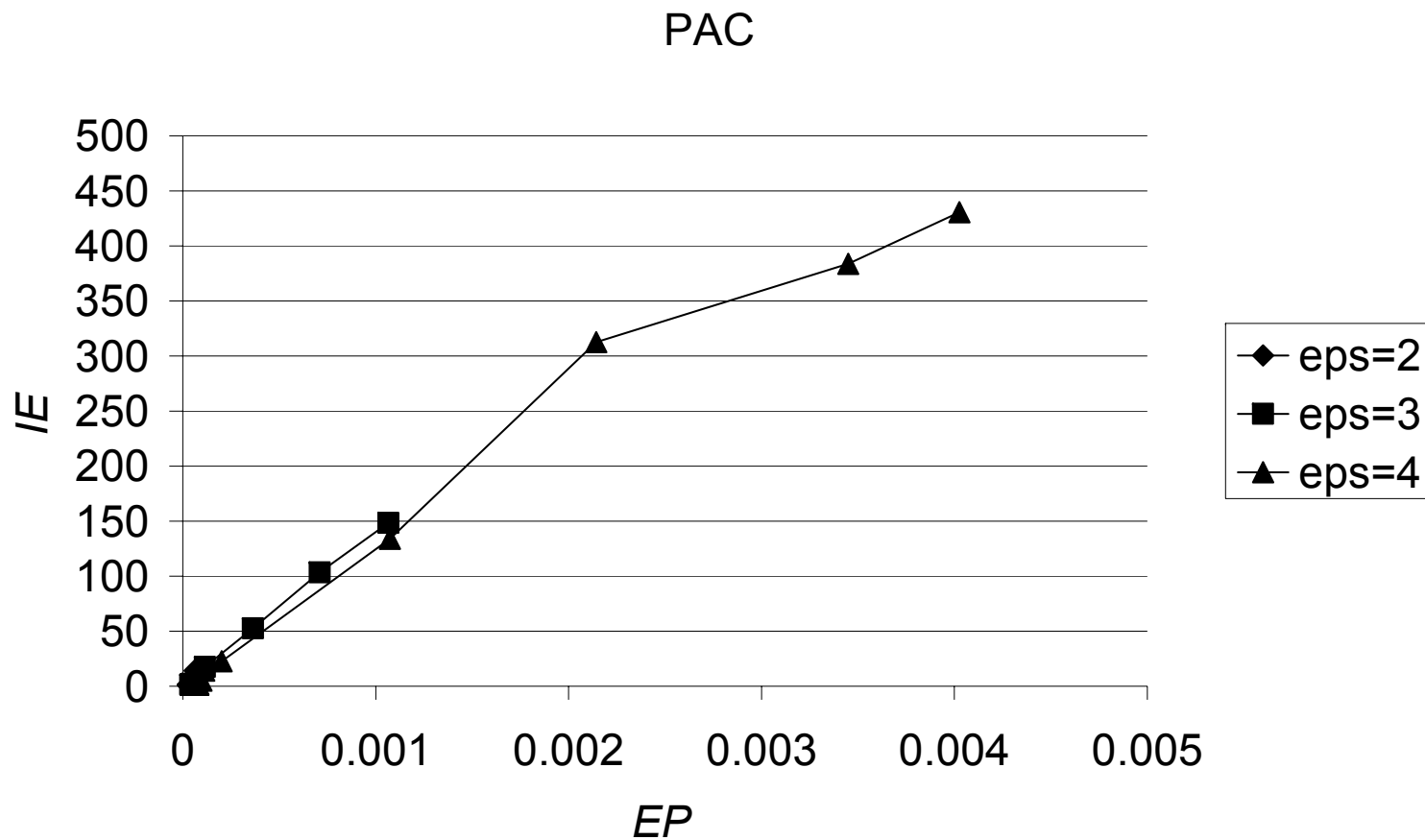
Small chance improvement



Comparison: *NN* Queries (cont).



Comparison: *NN* Queries (cont.)



Conclusion

- These techniques for approximate similarity searching can be applied to generic metric spaces.
 - Vector spaces are a special case of metric spaces.
- High accuracy of approximate results are generally obtained with high improvement of efficiency.
 - Best performance obtained with the good fraction approximation methods
 - The proximity-based method is a bit worse than the good fraction approximation but it can be used for range queries and k -NN queries.

Outline of the talk

1. similarity, metric space and distance measures (15')
2. similarity queries, metric partitioning principles (15')
3. query execution strategies (15')
4. avoiding distance computations (15')
5. filtering, pivot choosing and metric transformations (15')
6. short survey of metric space indexes (15')
7. M-tree family, D-index - performance evaluation (30')
8. approximate similarity search (30')
9. **scalable and distributed indexes (30')**

Implementation Postulates of Distributed Indexes

- **scalability** – nodes (computers) can be added (removed)
- **no hot-spots** – no centralized nodes, no flooding by messages
- **update independence** – network update at one site does not require an immediate change propagation to all the other sites

Distributed Similarity Search Structures

- Native metric structures:
 - GHT* (Generalized Hyperplane Tree) [BGZ05]
 - VPT* (Vantage Point Tree) [BNFZ06]
- Transformation approaches:
 - M-CAN (Metric Content Addressable Network) [FGZ05]
 - M-Chord (Metric Chord) [NZ06]

GHT* Architecture

■ Peers

- pose queries, insert and update objects
- store data in buckets and process queries
- identified by NNID - unique within the network

■ Buckets

- bounded storage capacity
- multiple buckets per a peer
- identified by BID - unique within a peer

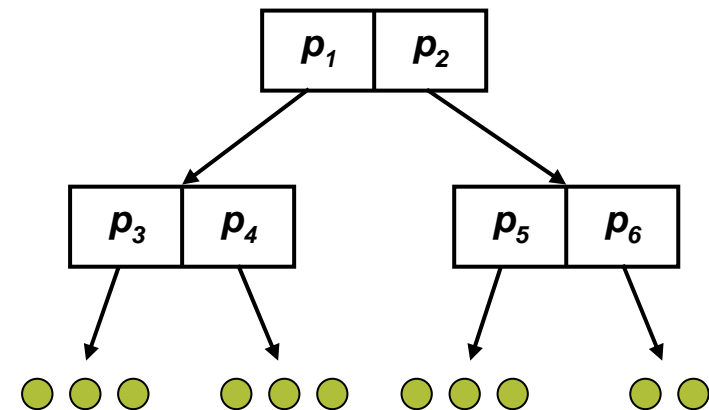
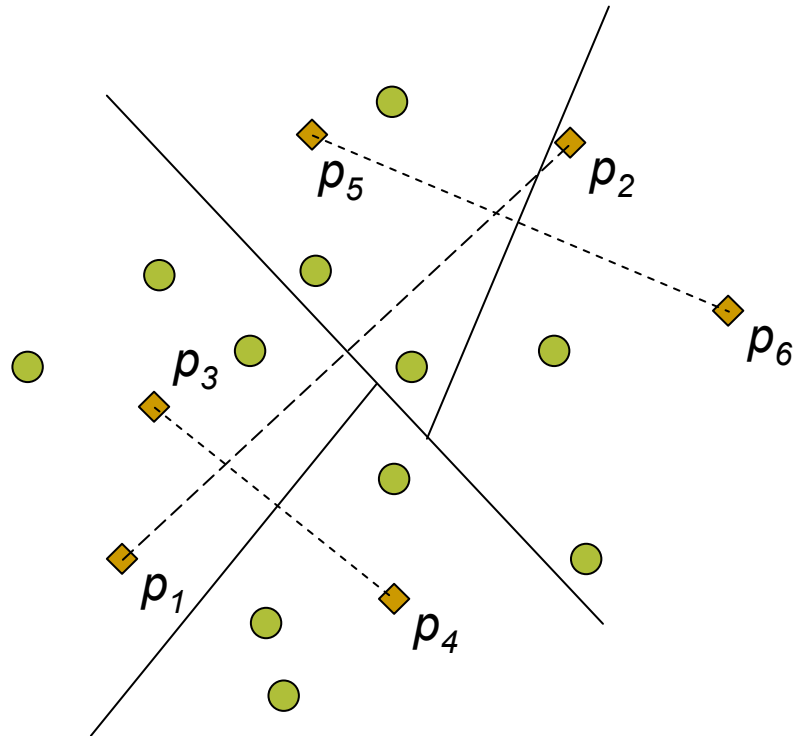
■ Address Search Tree (AST)

- tree-based navigation (routing) structure present in every peer
- object localization in several subsequent steps



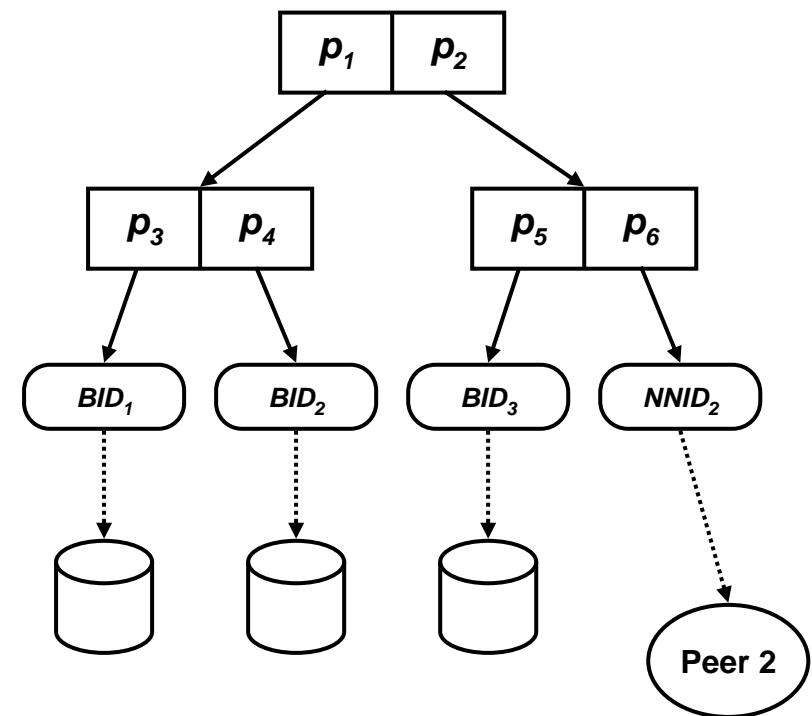
GHT* Address Search Tree

- Based on the Generalized Hyperplane Tree [Uhl91]
 - two pivots for binary partitioning

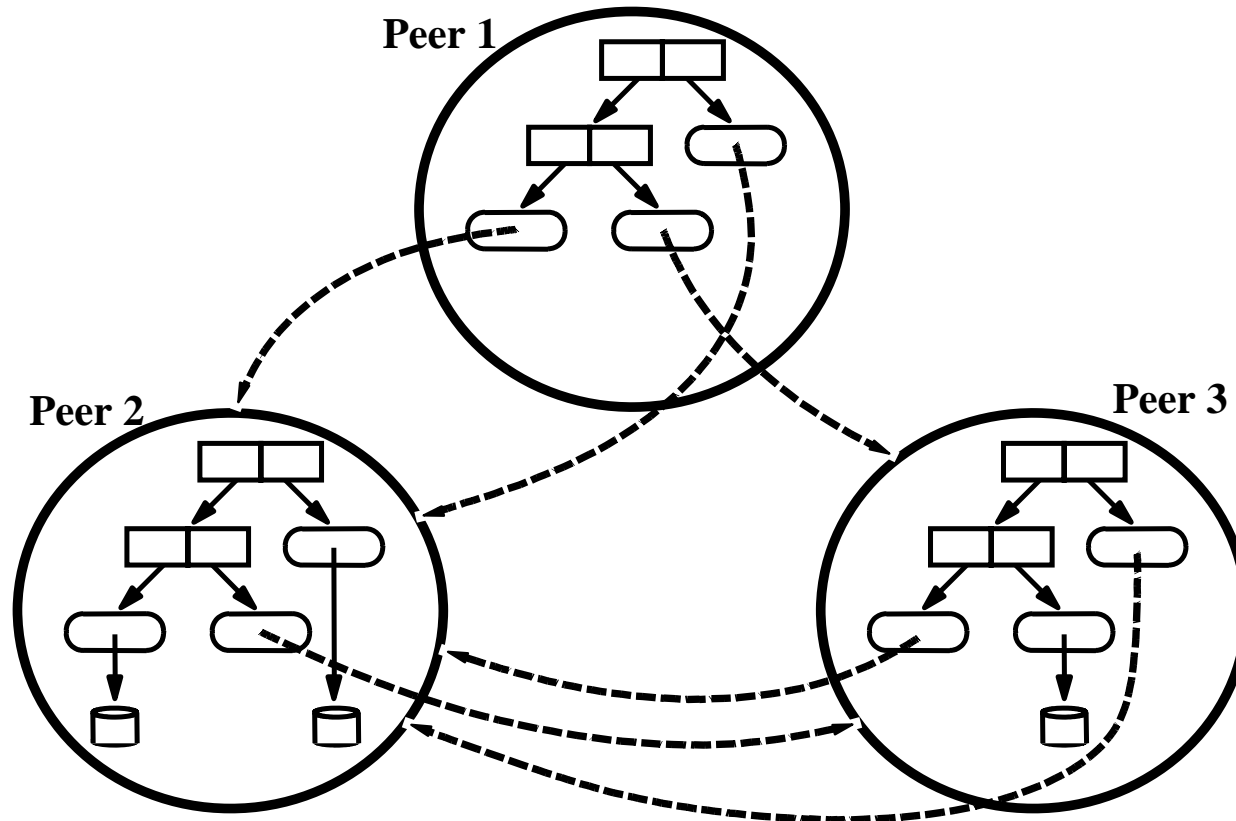


GHT* Address Search Tree

- Inner node
 - two pivots (reference objects)
- Leaf node
 - *BID* pointer to a bucket if data stored on the current peer
 - *NNID* pointer to a peer if data stored on a different peer

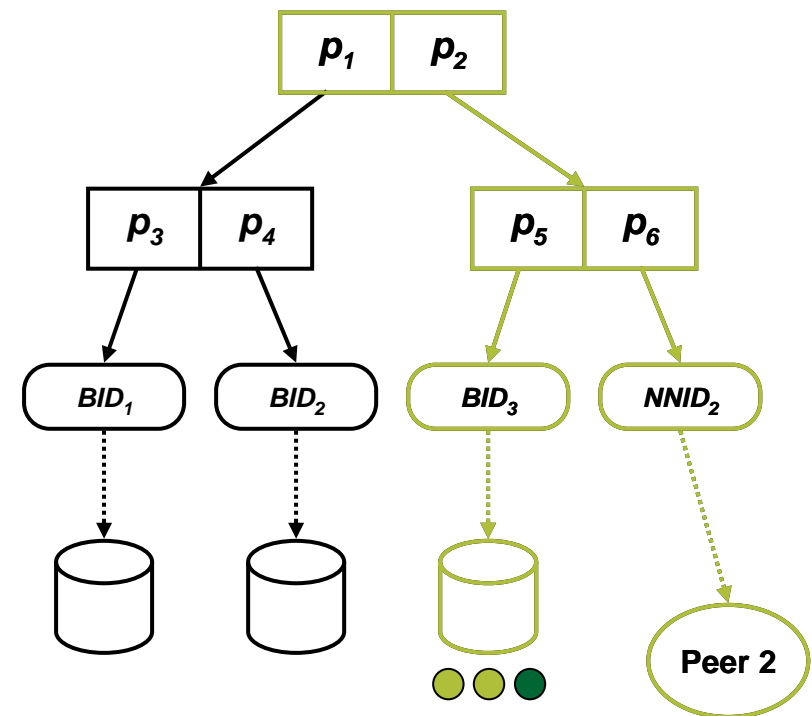
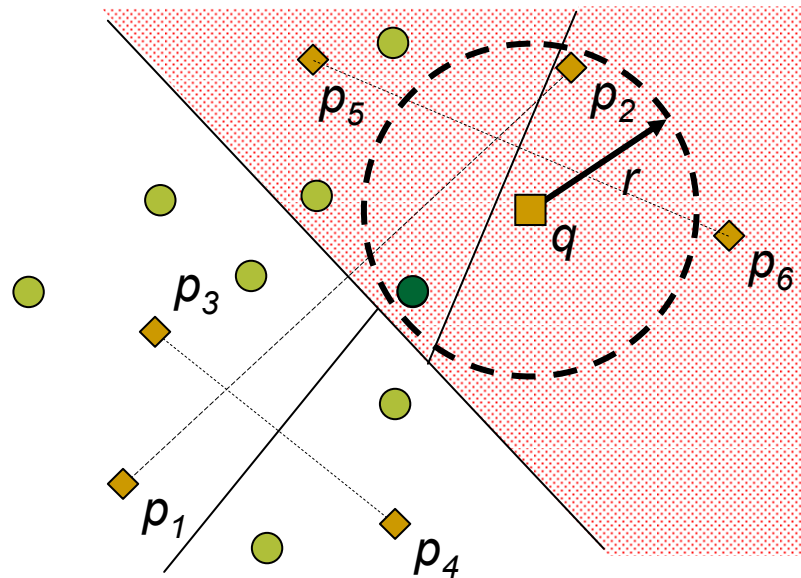


GHT* Address Search Tree



GHT* Range Query

- Range query $R(q,r)$
 - traverse peer's own AST
 - search buckets for all $BIDs$ found
 - forward query to all $NNIDs$ found



AST: Logarithmic replication

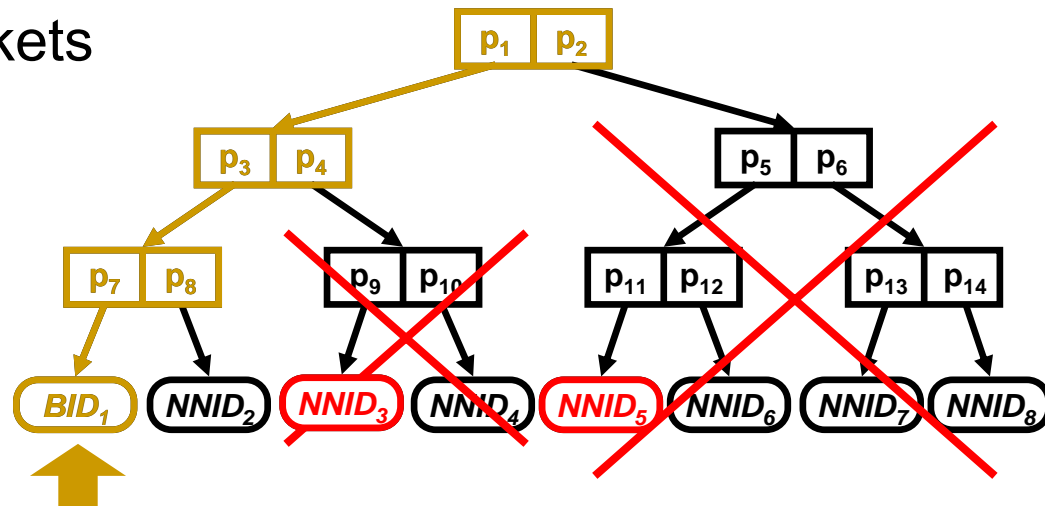
- Full AST on every peer is space consuming
 - replication of pivots grows in a linear way

- Store only a part of the AST:

- all paths to local buckets

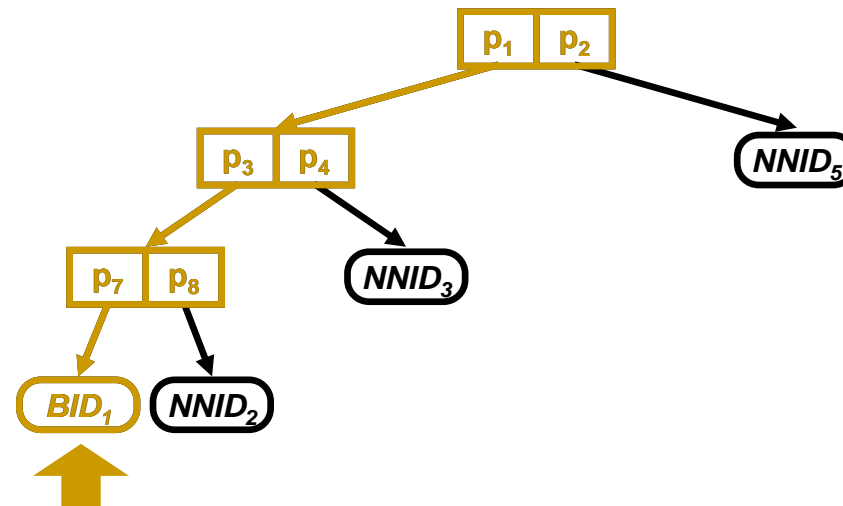
- Deleted sub-trees:

- replaced by *NNID* of the leftmost peer



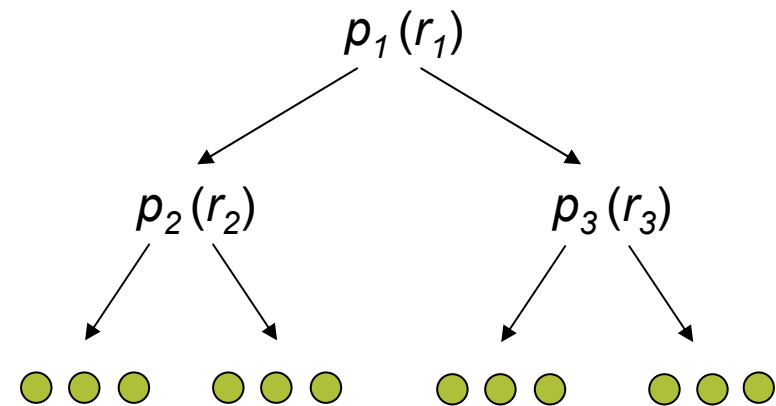
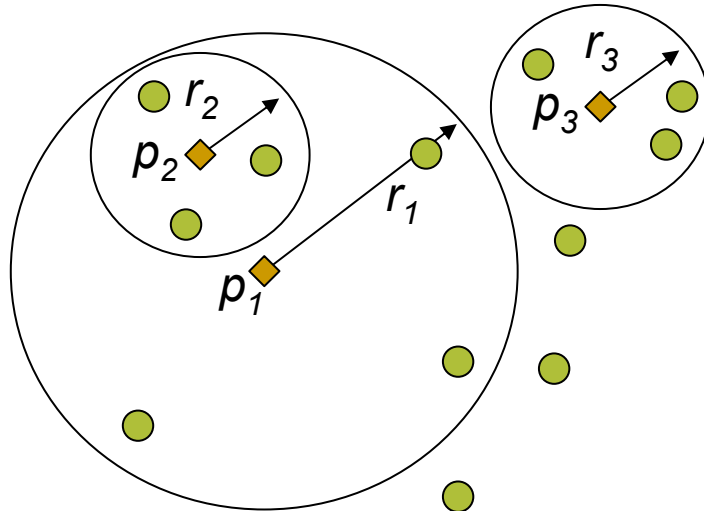
AST: Logarithmic Replication (cont.)

- Resulting tree
 - replication of pivots grows in a logarithmic way



VPT* Structure

- Similar to the GHT* - ball partitioning is used for AST
 - Based on the Vantage Point Tree [Yia93]
 - inner nodes have one pivot and a radius
 - different traversing conditions



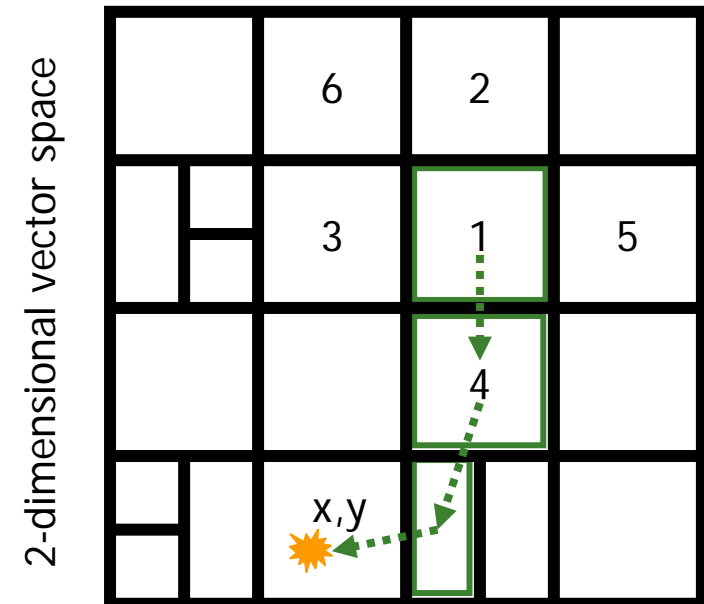
M-CAN: The Metric CAN

- Based on the Content-Addressable Network (CAN) [RFH⁺01]
 - a DHT navigating in an N -dimensional vector space

- The Idea:
 1. Map the *metric space* to a *vector space*
 - given N pivots: p_1, p_2, \dots, p_N , transform every o into vector $F(o)$
 2. Use CAN to
 - distribute the vector space zones among the nodes
 - navigate in the network

CAN: Principles & Navigation

- CAN – the principles
 - the space is divided in zones
 - each node “owns” a zone
 - nodes know their neighbors
- CAN – the navigation
 - greedy routing
 - in every step, move to the neighbor *closer* to the target location



M-CAN: Contractiveness & Filtering

- Use the L_∞ as a distance measure

- the mapping F is *contractive*

$$L_\infty(F(x), F(y)) \leq d(x, y)$$

- More pivots \Rightarrow better filtering

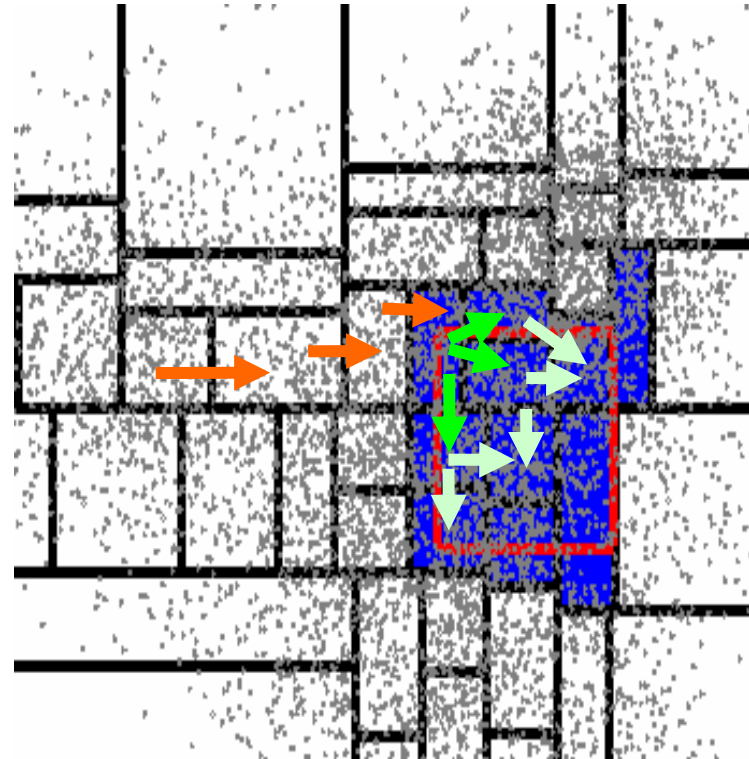
- but, CAN routing is better for less dimensions

- Additional filtering

- some pivots are only used for *filtering* data (inside the explored nodes)
- they are not used for mapping into CAN vector space

M-CAN: Range Query Execution

- Range query $R(q,r)$
 - map q on $F(q)$
 - route the query towards $F(q)$
- Candidate regions contain objects x
 - $L_\infty(F(x),F(q)) \leq r$
- Propagate the query over candidate regions using the multicast algorithm of CAN
- Check objects using the original d

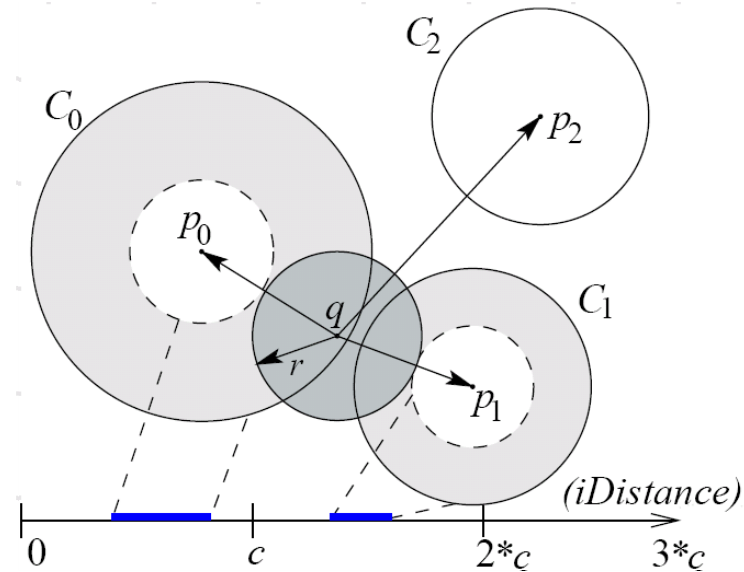


M-Chord: The Metric Chord

- Transform metric space to a one-dimensional domain
 - use a generalized version of the *iDistance* [JOT⁺05]
- Divide the domain into intervals
 - assign each interval to a peer
- Use the *Chord* P2P protocol for navigation [SMK⁺01]
- Apply object filtering
 - inside the explored peers
 - use more pivots

M-Chord: Indexing the Distance

- *iDistance* – indexing technique for vector domains
 - cluster the vector space & identify cluster centers p_i
 - assign *iDistance* keys to objects $x \in C_i$
$$iDist(x) = d(p_i, x) + i \cdot c$$
 - range query $R(q, r)$: identify *intervals of interest*
- Generalization to metric spaces
 - select pivots $\{p_1, \dots, p_n\}$
 - then partition: *Voronoi-style* [DN87]



M-Chord: Chord Protocol

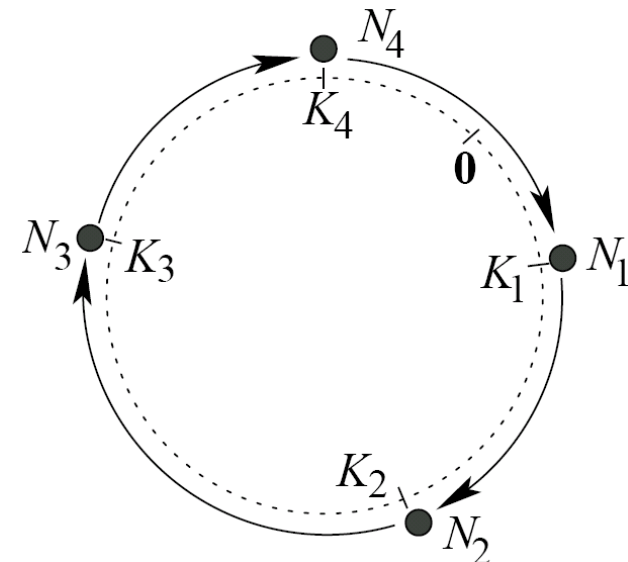
- Peer-to-Peer navigation protocol
- Peers are responsible for *intervals* of keys
- $O(\log n)$ hops to localize a node storing a given *key*

- M-Chord

- set the *iDistance* domain
- make it *uniform*: function h

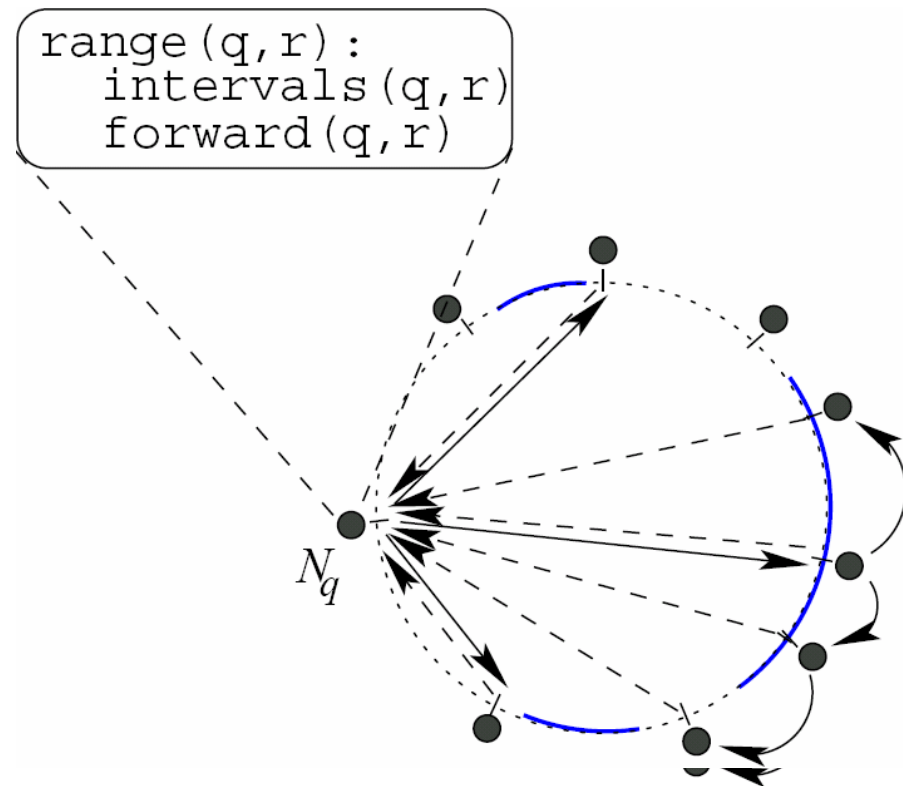
$$mchord(x) = h(d(p_i, x) + i \cdot c)$$

- Use *Chord* on this domain



M-Chord: Range Query

- Node N_q initiates the search
- Determine intervals
 - generalized *iDistance*
- Forward requests to peers with identified intervals
- Search in the nodes
 - using additional filtering
- Merge the received partial answers

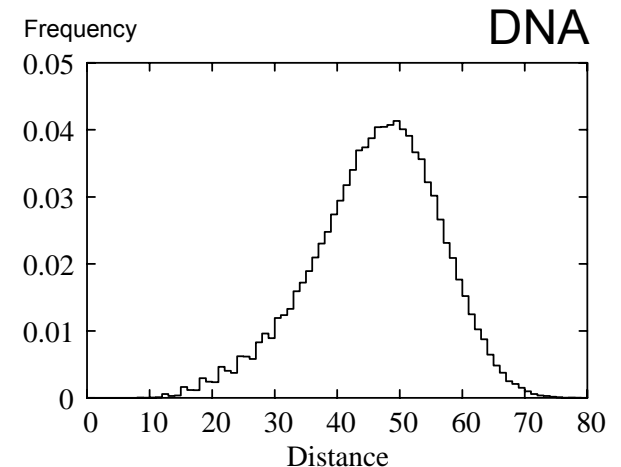
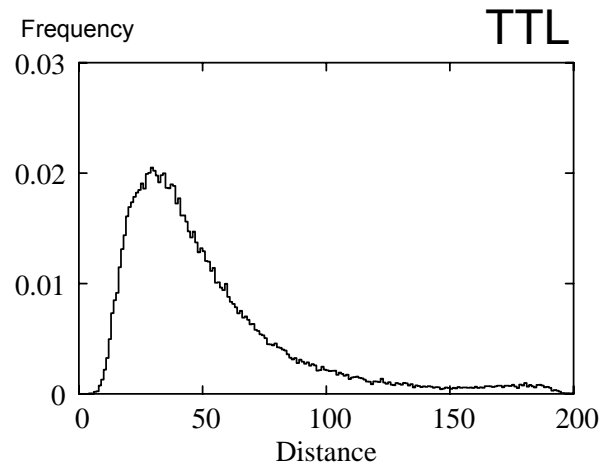
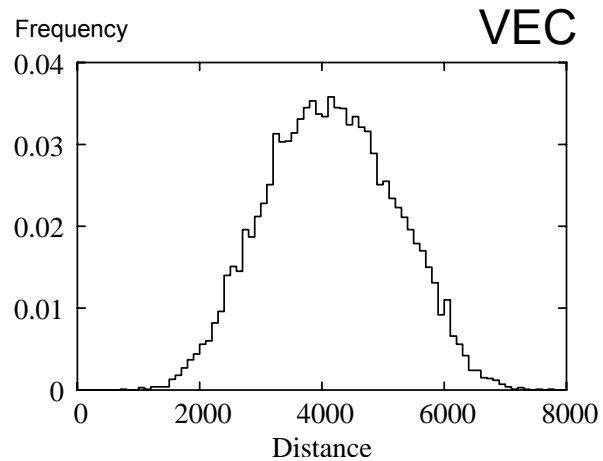


Scalability Performance Evaluation

[BNFZ06]

- Computing infrastructure
 - 300 workstations connected by 100Mbps network
 - Memory based storage
 - Maximum of 5000 objects per peer
- Datasets
 - **VEC**: 45-dimensional vectors of image color features compared by the *quadratic distance* measure
 - **TTL**: titles and subtitles of Czech books and periodicals compared using the *edit distance*
 - **DNA**: protein symbol sequences of length sixteen compared using the *weighted edit distance* (Needleman-Wunsch algorithm)

Datasets: Distance Density



- Density of the distances within the datasets
 - VEC: practically **normal** distance density
 - TTL: **skewed** density
 - DNA: **discrete** metric function with small variety of values

Measurements: CPU Costs Estimates

- Total distance computations
 - on all peers engaged
 - represents the CPU costs of a centralized structure
- Parallel distance computations
 - the maximal number of distance evaluations performed in a serial manner
 - represents the real CPU costs

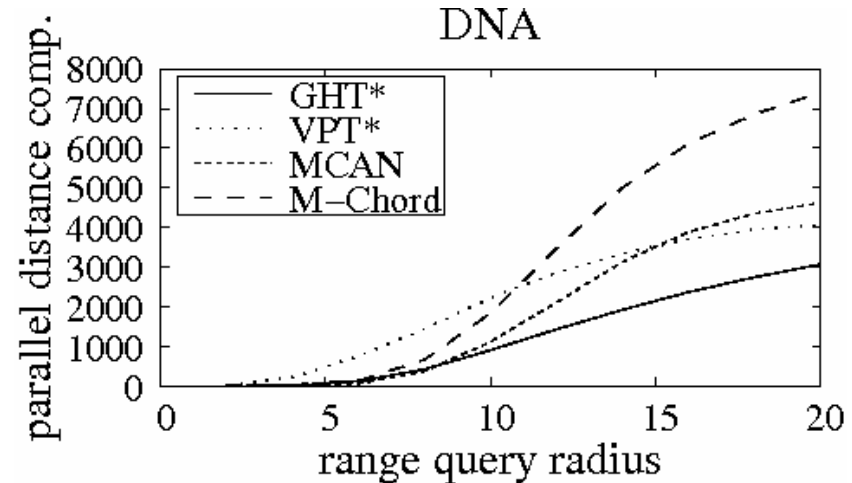
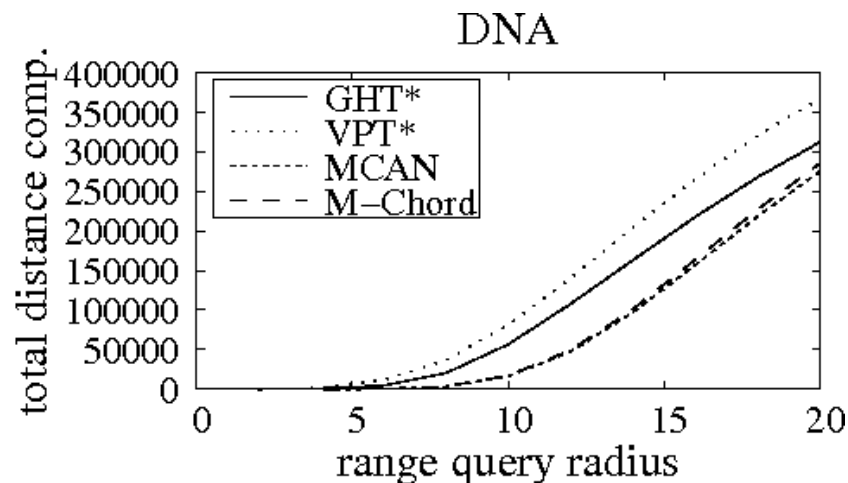
Measurements: Communication Costs Estimates

- Total number of messages
 - the number of all messages (requests and responses) sent during a query execution
 - represents the overall network communication load
- Maximal hop count
 - the maximal number of messages sent in a serial way to complete the query
 - represents the parallel communication costs

Changing the Query Size

- Range queries with changing radii r
 - Average costs over 100 different query objects for fixed r
 - **VEC** dataset: query radii between 200 and 2,000
 - **TTL** and **DNA** datasets: query radii between 2 and 20
- All four structures filled with 500,000 objects
- Peer storage load factors 60-70%
- Approximately 150 active nodes

Total & Parallel Distance Computations

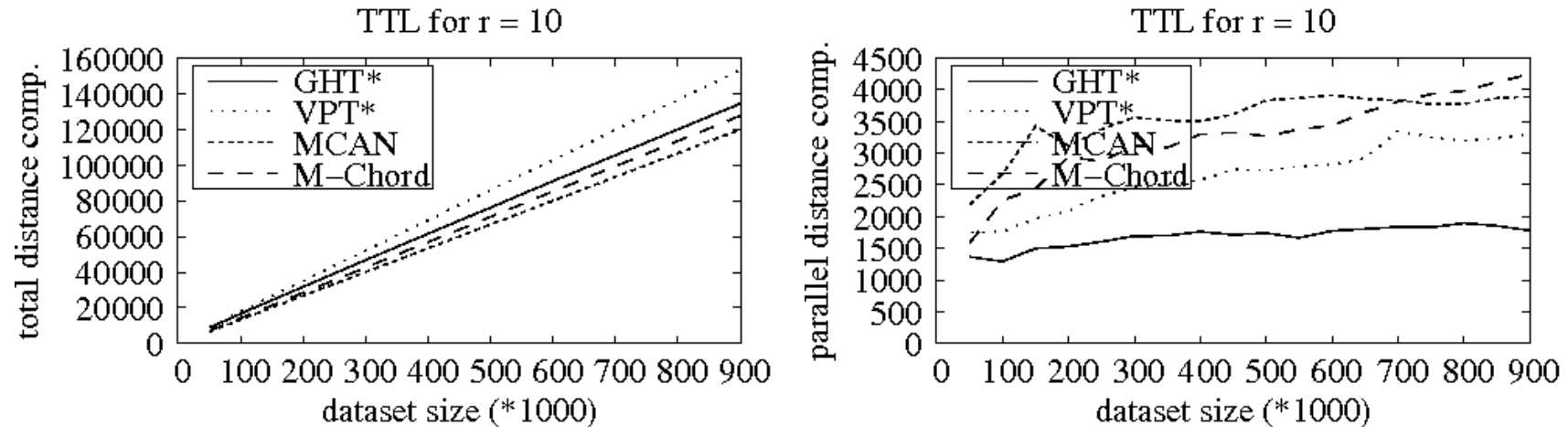


- Total costs comparable to centralized structures
 - pivot filtering differentiates the structures
- The parallel costs:
 - limited storage per peer – max. 5000 dist. computations per peer
 - M-Chord in DNA: very small and discrete domain causes multiple mapping collisions (mapping to one key)

Changing the Dataset Size

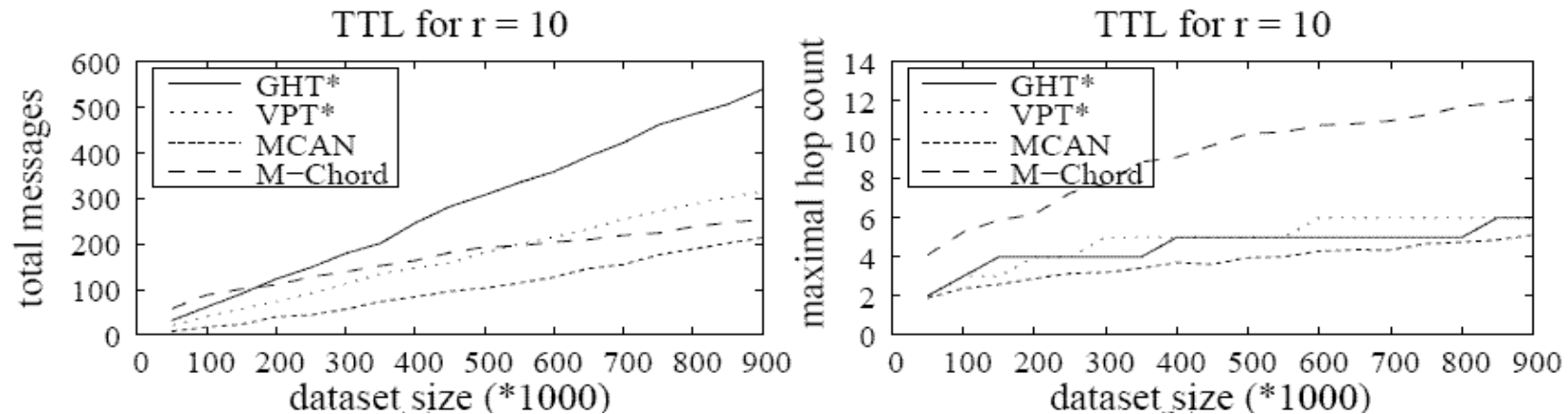
- Data volume size from 50,000 to 1,000,000 objects
 - increased in blocks of 50,000 objects
- Range queries executed after every block insertion
 - values averaged over 100 different query objects
 - **VEC** dataset: query radii fixed to 500, 1000 and 1500
 - **TTL** and **DNA** datasets: query radii fixed to 5, 10 and 15
- Measures:
 - distance computations and messaging costs
 - only selected results with typical behavior shown

Total & Parallel Distance Computations



- Total distance comp. costs grow linearly
 - M-Chord, MCAN filtering with 50 pivots is most efficient
- A very slow increase in parallel distance computations
 - response times do not increase significantly
 - M-Chord & MCAN: higher costs - relevant objects clustered on peers

Total Messages and Hop Count

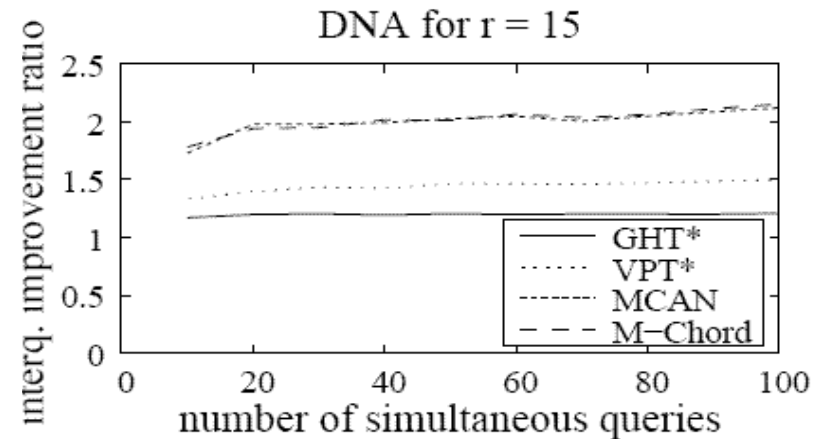
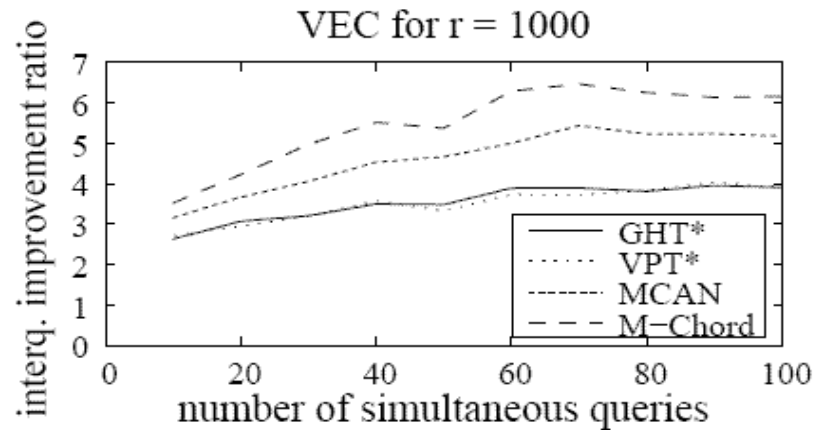


- Total messages correlate with visited peers
 - highest for GHT* – many peers visited – declustering effect
- Hop counts practically negligible
 - M-Chord significantly worse
 - sequential hops during adjacent clusters search

Changing the Number of Queries

- Simultaneously executed queries
 - queries from different peers started at the same time
 - parallel evaluation, whenever possible
 - sequential processing of queries at individual peers
- Groups from 10 to 100 range queries
 - average costs reported
 - VEC dataset: query radii fixed to 500, 1000 and 1500
 - TTL and DNA datasets: query radii fixed to 5, 10 and 15
- Data volume of size 500,000 objects

Interquery Improvement Ratio



- Number of simultaneous queries processed without performance degradation
- Influenced by the spread of objects among peers
 - GHT* visits the most peers for each query thus it has the lowest interquery improvement.

Concluding Summary

- Compared 4 distributed similarity search structures
 - Query size scalability
 - Dataset size scalability
 - Capability of simultaneous query processing

	single query	multiple queries
GHT*	excellent	poor
VPT*	good	satisfactory
MCAN	satisfactory	good
M-Chord	satisfactory	very good

SAPIR: Search In Audio Visual Content Using Peer-to-peer IR (IST FP6 Project)

- <https://sysrun.haifa.il.ibm.com/sapir/news.html>
- Partners:



A New Initiative in Similarity Searching

- Conference series on similarity searching
 - Web page with available software tools
 - Benchmark for testing – data-sets and queries
- Yair Bartal
Christian Bohm
Edgar Chavez
Paolo Ciaccia
Christos Faloutsos
Piotr Indyk
Daniel Keim
Gonzalo Navarro
Marco Patella
Hanan Samet
Enrique Vidal
Peter Yianilos
Pavel Zezula

Acknowledgements

- Partially supported by
 - The national research project 1ET100300419
 - The Czech Grant Agency project 201/07/P240